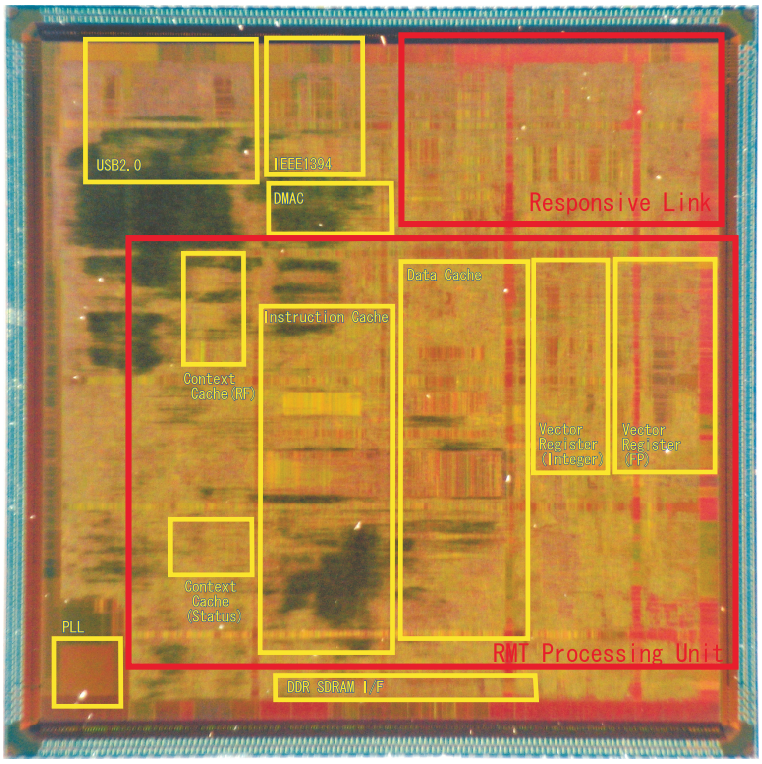
*Responsive Multithreaded Processor*仕様書

第 0版

平成18 年10 月30 日
慶應義塾大学 理工学部 山崎研究室

— NY0002 —



http://www.ny.ics.keio.ac.jp/research/rmt/

目 次

1

概要

1.1 Overview

RMT Processor は，分散リアルタイムシステムを実現するために，リアルタイム通信・処理・制御を同時にハードウェアレベルで行うことを目的にして設計を行ったシステム LSI である．分散リアルタイムシステムを容易かつ効率的に実現するには，リアルタイム通信及びリアルタイム処理を行なうための基本機能を有した共通プラットホームを用意し，それらをブロックを組み立てるように組み合わせてシステムを構築できるようにすれば良いと考えられる．プラットホームに必要な機能としては，リアルタイム処理機能，リアルタイム通信機能，コンピュータ用周辺機能，各種周辺制御機能が考えられる．プラットホームとして様々なシステムの中に容易に組み込んで使用できるようにするために，*RMT Processor* は以下の機能を全て 1 チップに集積 (System-on-a-chip) している．

- リアルタイム処理機能 (*RMT Processing Unit*)
- リアルタイム通信機能 (*Responsive Link*)
- コンピュータ用周辺機能 (PCI-X, IEEE1394, DDR SDRAM I/Fs, DMAC, etc.)
- 各種周辺制御機能 (PWM Generators，Pulse Counters, etc.)

システム設計者は本チップに必要な I/O（センサ，アクチュエータ，ディジタルカメラ等）を接続するだけで必要な機能を実現できる．それら I/O を接続し固有の機能を有した *RMT Processor* をそのシステムにふさわしいトポロジで *Responsive Link* を用いて複数個接続することによって，分散リアルタイムシステムを構築する．

1.2 設計ポリシ

RMT Processor はリアルタイム処理・通信の理論をそのまま実現できることを目標にして設計されている．リアルタイムスケジューラには，動的スケジューリングとして EDF(Earliest Deadline First) 等があり，静的スケジューリングとして RM(Rate Monotonic) 等があるが，ほぼ全てのリアルタイムスケジューリングは，優先度に従ってプリエンプションを行いながら実行や通信を行うことを要求する．プリエンプションは，演算（処理）の場合はコンテキストスイッチに相当し，通信の場合はパケットの追い越しに相当する．

従って，処理の場合は優先度付きスレッドのハードウェアによる優先実行やコンテキストスイッチのオーバヘッドの削減等を実現する．通信の場合は，従来までの通信では実現されていなかった，優先度付きパケットの追い越し機構等を実現する．

RMT Processor はこれらの機能を実装することにより，リアルタイムスケジューリング理論を背景に設計されたリアルタイムスケジューラ（ソフトウェア）により優先度付けされた処理や通信を，そのまま理論通りにリアルタイム処理および通信を実現することのできるハードウェアを実現している．

1.3 全体構成

図 1.1 に *RMT Processor* のブロック図を示す．*RMT PU* は，256bit のバスを介して DDR SDRAM I/F と接続している．バンド幅の広いバスを用いてプロセッシングコアとメインメモリを接続することにより，命令フェッチや後に述べるベクトル演算において，メモリアクセスのスループットを改善している．

Responsive Link，各種 I/O は 32bit バスに接続されている．32bit バスと 256bit バスはゲートウェイ（GW）を介して接続されている．それぞれのバスを流れるデータはゲートウェイにおいてバスサイジングが行われ，もう片方のバスに送られる．また，*Responsive Link* のイベントリンクは *RMT PU* のメモリアクセスユニットに直接接続され，プロセッシングコアからはバスを介さず，制御レジスタの一部としてアクセスすることができる．これにより，高速にイベントリンクにアクセスすることが可能である．

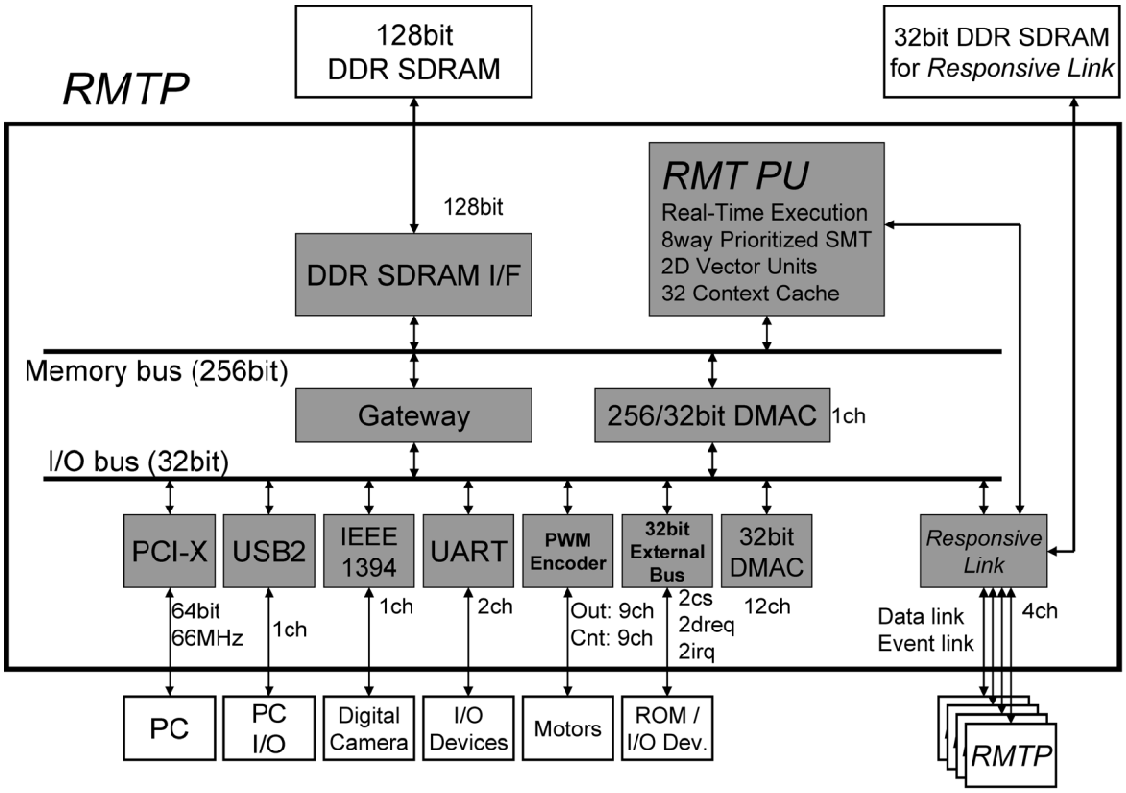


図 1.1: *RMT Processor* のブロック図

以下に *RMT Processor* が持つ I/O を示す．

- *Responsive Link* (4ch)

- PCI-X : 64bit, 66MHz (1ch)
- USB2 (1ch)
- IEEE1394 (1ch)
- UART (2ch)
- PWM Encoder
  - Output (9ch)
  - Pulse Counter (9ch)
- External Bus I/F (2cs, 2dreq, 2irq)
- 32bit DMA Controller (4ch × 3)
- 256/32bit DMA Controller (1ch)
- 128bit DDR SDRAM I/F (1ch)

1.4 Responsive Multithreaded Processing Unit

RMT PU は 8way の細粒度マルチスレッディングに優先度を用いた制御を行うことにより，ハードウェアで様々なレベルのリアルタイム処理を支援する．マルチスレッドアーキテクチャでは複数のスレッドが並列に実行されるため，スレッド間で演算器やキャッシュシステム等の計算資源の競合が起こる．競合が起こった場合，*RMT PU* はスレッド毎に設定された優先度を基に，優先度のより高い命令に対して先に計算資源を割り当てる．これにより並列に実行しているスレッドの中で，優先度の高いスレッドから優先的に実行する．

図 1.2 に *RMT PU* のブロック図を示す．*RMT PU* は命令発行ユニット（Issue Unit），命令演算ユニット（Execution Unit），キャッシュユニット（Cache Unit）の大きく 3 つに分かれる．命令発行ユニットは各スレッドの実行を制御し，優先度に従って命令演算ユニットに対して各スレッドの命令を送る．命令演算ユニットは命令発行ユニットから送られてきた命令を演算する．キャッシュユニットは命令発行ユニットからの命令フェッチ要求，命令演算ユニットからのデータアクセス要求を処理する．

1.4.1 命令発行ユニット

命令発行ユニットの役割は各スレッドの実行を制御し，命令演算ユニットに対して命令を発行することである．表 1.1 に命令発行ユニットの概要を示す．

アクティブスレッドとはプロセッサ内に保持されているスレッドで，すぐに実行を開始することができる．キャッシュスレッドとは後で述べるコンテキストキャッシュ内に保持されているスレッドを示す．優先度は 8bit を用いて 256level で表し，値が大きいほど優先度は高くなる．

各スレッドの制御はスレッド制御ユニットで行う．アクティブスレッドはスレッド制御ユニット内にあるスレッドテーブルによって管理される．スレッドテーブルのフォーマットを 図 1.3 に示す．ENABLE フィールドはアクティブスレッドが有効であるかどうかを示す．STATE フィールドはアクティブスレッドの状態を示し，実行中，停止中，後述するコンテキストキャッシュへの退避中等といった状態を示す．KEEP フィールドはアクティブスレッドをプロセッサ内に保持しつづけるかどうかを示す．PRIORITY フィールドはスレッドの優先度を示し，この値が *RMT PU* 全体で使用される．

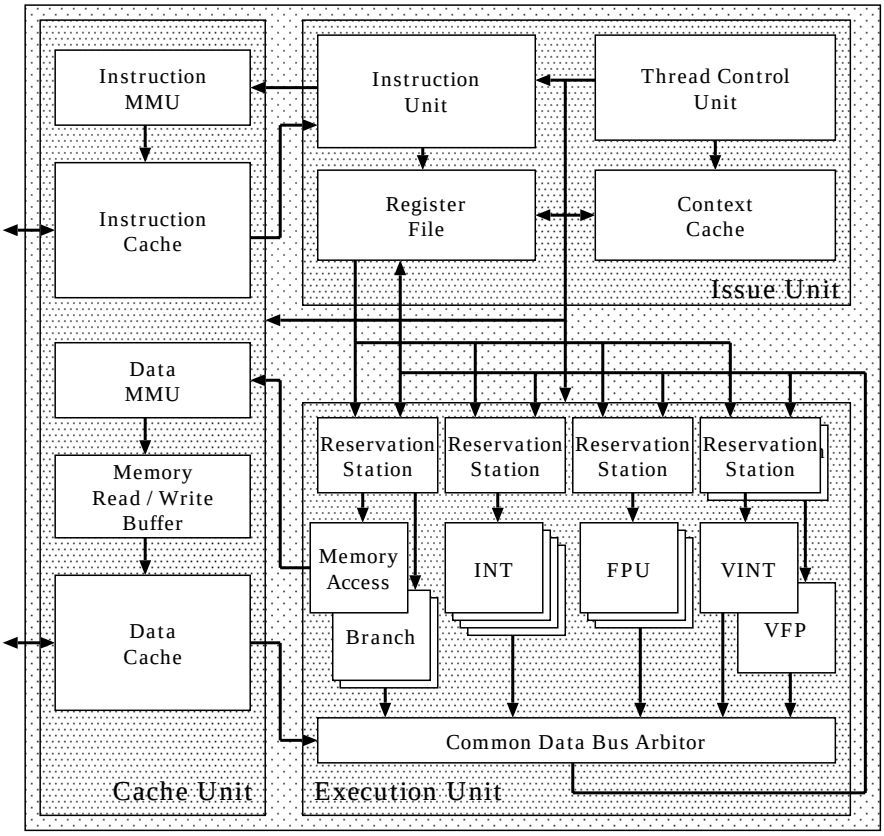


図 1.2: *RMT PU* のブロック図

13	12:9	8	7:0
ENABLE	STATE	KEEP	PRIORITY

図 1.3: スレッドテーブルのフォーマット

RMT PU ではスレッドの生成，削除，実行，停止，優先度の設定等のために新たに命令を追加した．スレッド制御ユニットはこれらの命令が発行されると，命令に応じてスレッドテーブルを書き換え，アクティブスレッドの制御を行う．

先に述べた通り，*RMT PU* では 8 つのコンテキストをプロセッサ内に保持して実行することができる．しかしそれ以上のスレッドを実行する場合，コンテキストスイッチが発生する．コンテキストスイッチは現在実行しているスレッドのコンテキストをメモリに退避し，新しく実行するスレッドのコンテキストをメモリから復帰しなければならないため，オーバヘッドが大きくなる．

RMT PU ではコンテキストを格納するための専用キャッシュをオンチップに用意し，レジスタファイルとの間を広いバス（GPR:256bit，FPR:128bit）で接続している．コンテキストキャッシュは 32 個のコンテキストを格納することができ，コンテキストスイッチをハードウェアにより 4 クロックサイクルで行う．これによりコンテキストスイッチにかかるオーバヘッドを大幅に削減する．

アクティブスレッドのコンテキストキャッシュへの退避，キャッシュスレッドのプロセッサ内への復帰，アクティブスレッドとキャッシュスレッドの入れ替えは新たに追加した命令により，スレッド制御ユニットが行う．スレッド制御ユニットはスレッドの退避命令や復帰命令，入れ替え命令を受け取ると，内部に保

表 1.1: 命令発行ユニットの概要

アクティブスレッド数	8
キャッシュスレッド数	32
優先度の指定	256level
命令フェッチ数	8
同時命令発行数	4
同時命令完了数	4
整数レジスタ数	32bit × 32entry × 8set
整数リネームレジスタ数	32bit × 64entry
浮動小数点レジスタ数	64bit × 8entry × 8set
浮動小数点リネームレジスタ数	64bit × 64entry

表 1.2: 命令演算ユニットの概要

整数演算器	4 + 1（Divider）
浮動小数点演算器	2 + 1（Divider）
64bit 整数演算器	1
整数ベクトル演算器	1（8IU × 2 line）
浮動小数点ベクトル演算器	1（4FPU × 2 line）
分岐ユニット	2
メモリアクセスユニット	1
同期ユニット	1

持しているキャッシュスレッドのテーブルを検索し，コンテキストキャッシュをアクセスするためのアドレスを生成して，コンテキストキャッシュをアクセスする．

命令発行ユニットは命令キャッシュアクセスと命令演算ユニットへの命令発行スロットで，優先度を用いた調停を行う．

1.4.2 命令演算ユニット

命令演算ユニットの役割は命令発行ユニットから送られてくる命令を演算することである．表 1.2 に命令演算ユニットの概要を示す．

RMT PU はリザベーションステーションとリオーダバッファを用いて，アウトオブオーダ実行を行う．*RMT PU* では複数のスレッドが並列に実行されているため，各演算器においてスレッド間で競合が起こる．命令演算ユニットではリザベーションステーションにおいて，優先度による制御を行う．リザベーションステーションでは演算に必要なオペランドがそろうまで命令は保持される．演算に必要なオペランドがそろい命令の実行が可能になると，各演算器に対して命令が発行される．*RMT PU* では複数の命令が実行可能になった場合，リザベーションステーションは，各命令の優先度を調べ，優先度の高い命令から先に演算器に発行する．これにより優先度の高いスレッドの命令に対して，先に演算器を割り当てる．

一方，マルチメディア処理のようなソフトリアルタイム処理では多くのデータを繰り返し演算しなければならないため，高い演算性能が要求される．このような処理ではデータの並列性を利用して演算性能を高めることができる．

表 1.3: キャッシュユニットの概要

TLB エントリ （命令，データ）	64 entry
キャッシュサイズ （命令，データ）	32K byte
victim cache （命令，データ）	512 byte

RMT PU ではベクトル演算機構を用いている．ベクトル演算により，少ない命令スロットを有効に活用し，ソフトリアルタイム処理に要求される高い演算性能を実現する．また，ベクトル演算を行うスレッドの数やプログラムによって必要とされるベクトルレジスタの構成は異なってくる．そこで *RMT PU* では整数，浮動小数点共に 512 セットあるベクトルレジスタを，ベクトル長やレジスタの個数等の構成を動的に変更してスレッド間で共有することにより，複数のスレッドで柔軟なベクトル演算を可能としている．

ベクトル演算は整数演算，浮動小数点演算共に 2 つの演算パイプラインが並列に動作することにより，複数のスレッドで並列してベクトル演算を行うことができる．各演算パイプラインは，整数演算パイプラインで 8 個，浮動小数点演算パイプラインで 4 個の演算器を持つことにより，複数のベクトル要素を並列に演算する．また，プログラマが複合演算を定義し，定義した命令を 1 命令で実行することにより，ベクトル演算器の使用率を向上させ，ベクトル演算の性能を向上させている．

1.4.3 キャッシュユニット

キャッシュユニットの役割は命令発行ユニットから送られてくる命令フェッチ要求と，命令実行ユニットから送られてくるデータアクセス要求を処理することである．表 1.3 にキャッシュユニットの概要を示す．

キャッシュユニットは MMU（Memory Management Unit）を持ち，ハードウェアでアドレス変換を行うため，各スレッドは仮想アドレスを用いてプログラミングを行うことができる．

MMU が置かれる場所により，仮想アドレスでキャッシュをアクセスするか物理アドレスでキャッシュをアクセスするかが決まる．図 1.2 に示した通り，*RMT PU* では MMU はキャッシュよりも前に置かれ，キャッシュアクセスを行う前にアドレス変換を行う．よってキャッシュは物理アドレスを用いてアクセスされる．キャッシュアクセスの前にアドレス変換を行うため，キャッシュアクセスにかかるレイテンシが増加するが，実行するスレッドがコンテキストスイッチにより切り替わった場合でもキャッシュをフラッシュする必要がなくなる．また，複数のスレッドでメモリ領域を共有する場合，仮想アドレスでキャッシュをアクセスすると同一の物理メモリのデータが複数キャッシュされる問題（synonym）が起こるが，物理アドレスを用いてキャッシュをアクセスすることによりその問題を回避することができる．

MMU における TLB エントリには仮想ページ番号，物理ページ番号の他に，複数スレッドで共有するための共有情報，コンテキストグループ番号を指定する．*RMT PU* は最大 8 つのスレッドが動作するため，TLB エントリのミス率が高くなることが考えられる．共有情報を用いることにより，複数のスレッドで TLB エントリを共有し，使用する TLB のエントリ数を削減することができる．

共有情報を設定した後に，新しいスレッドを共有情報に追加する場合はコンテキストグループ番号を用いる．TLB を設定する場合，コンテキストグループ番号を指定することにより，コンテキストグループ番号の一致するエントリの共有情報に自身のスレッドを追加する．これにより，使用する TLB のエントリ数を増やすことなく TLB を有効化することができる．

キャッシュは命令キャッシュ，データキャッシュ共に 8way の set-associative 方式，ブロックサイズは 32byte で，キャッシュアクセスはパイプライン化されている．キャッシュミスが起こった場合，入れ換えるブロックの選択方法は LRU と優先度がある．優先度を基に入れ換えるブロックを選択する場合，より優先度の低いスレッドが使用しているブロックから先にキャッシュを追い出される．これにより，優先度の高いスレッドのキャッシュブロックが追い出されることを防ぐ．

victim cache は，キャッシュブロックの入れ換えに伴ないキャッシュを追い出されたブロックを，full associative 方式で保持する．キャッシュミスを起した場合，victim cache にデータが残っていれば，そのブロックをキャッシュに戻すことにより．キャッシュミスによる内部バスへの要求を減らし，メモリアクセスの遅延を減少させる．

キャッスミス等によりバスを介して下位メモリをアクセスする場合にも優先度を用いた制御を行う．メモリアクセスはキャッシュよりも低速なため，待ち行列が発生する．この場合，より優先度の高いスレッドからバスを使用して下位メモリにアクセスする．

1.5 *Responsive Link*

Responsive Link は，柔軟なリアルタイム通信を実現するために，ソフトリアルタイム通信（データリンク）とハードリアルタイム通信（イベントリンク）の分離，パケットに優先度を付加しノード毎に高優先度パケットが低優先度パケットの追い越し，パケットの優先度が異なると優先度毎に別経路を設定して専用回線や迂回路を実現可能，ノード毎に優先度を付け替えることができ分散管理型でパケットの加減速を制御可能，ハードウェアによるエラー訂正，通信速度を動的に変更可能，トポロジーフリー，Hot-Plug&Play 等の様々な機能を実現する．

Responsive Link は国内では情報処理学会試行標準 (IPSJ-TS 2003:0006) として標準化されており，国際的にはでは ISO/IEC JTC1 SC25 WG4 において標準化作業が行われている．

2

PIN配置

チップのピン配置の一覧を以下に示す.

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
1		LEFT	pci_iopad0_uiov_lo_30	PDB24DGZ	57.4
	INOUT	AD[30] (PCI バス)			
2		LEFT	pci_iopad0_uiov_lo_29	PDB24DGZ	57.4
	INOUT	AD[29] (PCI バス)			
3		LEFT	vss_io_pci11	PVSS2DGZ	259
		IO digital VSS(0v)			
4		LEFT	pci_iopad0_uiov_lo_28	PDB24DGZ	57.4
	INOUT	AD[28] (PCI バス)			
5		LEFT	vdd_io_pci11	PVDD2DGZ	80
		IO digital VDD(2.5v)			
6		LEFT	pci_iopad0_uiov_lo_27	PDB24DGZ	57.4
	INOUT	AD[27] (PCI バス)			
7		LEFT	pci_iopad0_uiov_lo_26	PDB24DGZ	57.4
	INOUT	AD[26] (PCI バス)			
8		LEFT	pci_iopad0_uiov_lo_25	PDB24DGZ	57.4
	INOUT	AD[25] (PCI バス)			
9		LEFT	pci_iopad0_uiov_lo_24	PDB24DGZ	57.4
	INOUT	AD[24] (PCI バス)			
10		LEFT	pci_iopad0_uiov2_lo_3	PDB24DGZ	57.4
	INOUT	C/BE[3] (PCI バス)			
11		LEFT	vdd_core_l18	PVDD1DGZ	79
		Core digital VDD(1.0v)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
12		LEFT	pci_iopad0_uin1	PDB24DGZ	57.4
	INOUT	IDSEL (PCI バス)			
13		LEFT	vss_core_l31	PVSS1DGZ	79
		Core digital VSS(0v)			
14		LEFT	pci_iopad0_uiov_lo_23	PDB24DGZ	57.4
	INOUT	AD[23] (PCI バス)			
15		LEFT	vss_io_pci10	PVSS2DGZ	259
		IO digital VSS(0v)			
16		LEFT	pci_iopad0_uiov_lo_22	PDB24DGZ	57.4
	INOUT	AD[22] (PCI バス)			
17		LEFT	vdd_io_pci10	PVDD2DGZ	80
		IO digital VDD(2.5v)			
18		LEFT	pci_iopad0_uiov_lo_21	PDB24DGZ	57.4
	INOUT	AD[21] (PCI バス)			
19		LEFT	pci_iopad0_uiov_lo_20	PDB24DGZ	57.4
	INOUT	AD[20] (PCI バス)			
20		LEFT	pci_iopad0_uiov_lo_19	PDB24DGZ	57.4
	INOUT	AD[19] (PCI バス)			
21		LEFT	vss_core_l30	PVSS1DGZ	79
		Core digital VSS(0v)			
22		LEFT	pci_iopad0_uiov_lo_18	PDB24DGZ	57.4
	INOUT	AD[18] (PCI バス)			
23		LEFT	vdd_core_l17	PVDD1DGZ	79
		Core digital VDD(1.0v)			
24		LEFT	pci_iopad0_uiov_lo_17	PDB24DGZ	57.4
	INOUT	AD[17] (PCI バス)			
25		LEFT	vss_io_pci9	PVSS2DGZ	259
		IO digital VSS(0v)			
26		LEFT	pci_iopad0_uiov_lo_16	PDB24DGZ	57.4
	INOUT	AD[16] (PCI バス)			
27		LEFT	vdd_io_pci9	PVDD2DGZ	80
		IO digital VDD(2.5v)			
28		LEFT	pci_iopad0_uiov2_lo_2	PDB24DGZ	57.4
	INOUT	C/BE[2] (PCI バス)			
29		LEFT	vss_core_l29	PVSS1DGZ	79
		Core digital VSS(0v)			
30		LEFT	pci_iopad0_uio3	PDB24DGZ	57.4
	INOUT	FRAME# (PCI バス)			
31		LEFT	pci_iopad0_uio4	PDB24DGZ	57.4
	INOUT	IRDY# (PCI バス)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
32		LEFT	pci_iopad0_uio5	PDB24DGZ	57.4
	INOUT	TRDY# (PCI バス)			
33		LEFT	vdd_core_l16	PVDD1DGZ	79
		Core digital VDD(1.0v)			
34		LEFT	pci_iopad0_uio7	PDB24DGZ	57.4
	INOUT	DEVSEL# (PCI バス)			
35		LEFT	vss_core_l28	PVSS1DGZ	79
		Core digital VSS(0v)			
36		LEFT	pci_iopad0_uio6	PDB24DGZ	57.4
	INOUT	STOP# (PCI バス)			
37		LEFT	vss_io_pci8	PVSS2DGZ	259
		IO digital VSS(0v)			
38		LEFT	pci_iopad0_uin3	PDB24DGZ	57.4
	INOUT	LOCK# (PCI バス)			
39		LEFT	vdd_io_pci8	PVDD2DGZ	80
		IO digital VDD(2.5v)			
40		LEFT	pci_iopad0_uio10	PDB24DGZ	57.4
	INOUT	PERR# (PCI バス)			
41		LEFT	pci_iopad0_uout2	PDB24DGZ	57.4
	INOUT	SERR# (PCI バス)			
42		LEFT	pci_iopad0_uio1	PDB24DGZ	57.4
	INOUT	PAR (PCI バス)			
43		LEFT	vss_core_l27	PVSS1DGZ	79
		Core digital VSS(0v)			
44		LEFT	pci_iopad0_uiov2_lo_1	PDB24DGZ	57.4
	INOUT	C/BE[1] (PCI バス)			
45		LEFT	vdd_core_l15	PVDD1DGZ	79
		Core digital VDD(1.0v)			
46		LEFT	pci_iopad0_uiov_lo_15	PDB24DGZ	57.4
	INOUT	AD[15] (PCI バス)			
47		LEFT	pci_iopad0_uiov_lo_14	PDB24DGZ	57.4
	INOUT	AD[14] (PCI バス)			
48		LEFT	pci_iopad0_uiov_lo_13	PDB24DGZ	57.4
	INOUT	AD[13] (PCI バス)			
49		LEFT	vss_core_l26	PVSS1DGZ	79
		Core digital VSS(0v)			
50		LEFT	pci_iopad0_uiov_lo_12	PDB24DGZ	57.4
	INOUT	AD[12] (PCI バス)			
51		LEFT	vss_io_pci7	PVSS2DGZ	259
		IO digital VSS(0v)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
52		LEFT	pci_iopad0_uiov_lo_11	PDB24DGZ	57.4
	INOUT	AD[11] (PCI バス)			
53		LEFT	vdd_io_pci7	PVDD2DGZ	80
		IO digital VDD(2.5v)			
54		LEFT	pci_iopad0_uiov_lo_10	PDB24DGZ	57.4
	INOUT	AD[10] (PCI バス)			
55		LEFT	pci_iopad0_uin4	PDB24DGZ	57.4
	INOUT	M66EN (PCI バス)			
56		LEFT	pci_iopad0_uiov_lo_9	PDB24DGZ	57.4
	INOUT	AD[9] (PCI バス)			
57		LEFT	vdd_core_l14	PVDD1DGZ	79
		Core digital VDD(1.0v)			
58		LEFT	pci_iopad0_uiov_lo_8	PDB24DGZ	57.4
	INOUT	AD[8] (PCI バス)			
59		LEFT	vss_core_l25	PVSS1DGZ	79
		Core digital VSS(0v)			
60		LEFT	pci_iopad0_uiov2_lo_0	PDB24DGZ	57.4
	INOUT	C/BE[0] (PCI バス)			
61		LEFT	vss_io_pci6	PVSS2DGZ	259
		IO digital VSS(0v)			
62		LEFT	pci_iopad0_uiov_lo_7	PDB24DGZ	57.4
	INOUT	AD[7] (PCI バス)			
63		LEFT	vdd_io_pci6	PVDD2DGZ	80
		IO digital VDD(2.5v)			
64		LEFT	pci_iopad0_uiov_lo_6	PDB24DGZ	57.4
	INOUT	AD[6] (PCI バス)			
65		LEFT	vss_core_l24	PVSS1DGZ	79
		Core digital VSS(0v)			
66		LEFT	pci_iopad0_uiov_lo_5	PDB24DGZ	57.4
	INOUT	AD[5] (PCI バス)			
67		LEFT	pci_iopad0_uiov_lo_4	PDB24DGZ	57.4
	INOUT	AD[4] (PCI バス)			
68		LEFT	pci_iopad0_uiov_lo_3	PDB24DGZ	57.4
	INOUT	AD[3] (PCI バス)			
69		LEFT	vdd_core_l13	PVDD1DGZ	79
		Core digital VDD(1.0v)			
70		LEFT	pci_iopad0_uiov_lo_2	PDB24DGZ	57.4
	INOUT	AD[2] (PCI バス)			
71		LEFT	vss_core_l23	PVSS1DGZ	79
		Core digital VSS(0v)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
72		LEFT	pci_iopad0_uiov_lo_1	PDB24DGZ	57.4
	INOUT	AD[1] (PCI バス)			
73		LEFT	vss_io_pci5	PVSS2DGZ	259
		IO digital VSS(0v)			
74		LEFT	pci_iopad0_uiov_lo_0	PDB24DGZ	57.4
	INOUT	AD[0] (PCI バス)			
75		LEFT	vdd_io_pci5	PVDD2DGZ	80
		IO digital VDD(2.5v)			
76		LEFT	pci_iopad0_uio8	PDB24DGZ	57.4
	INOUT	ACK64# (PCI バス)			
77		LEFT	vss_core_l22	PVSS1DGZ	79
		Core digital VSS(0v)			
78		LEFT	pci_iopad0_uio9	PDB24DGZ	57.4
	INOUT	REQ64# (PCI バス)			
79		LEFT	pci_iopad0_uiov2_hi_3	PDB24DGZ	57.4
	INOUT	C/BE[7] (PCI バス)			
80		LEFT	pci_iopad0_uiov2_hi_2	PDB24DGZ	57.4
	INOUT	C/BE[6] (PCI バス)			
81		LEFT	vdd_core_l12	PVDD1DGZ	79
		Core digital VDD(1.0v)			
82		LEFT	pci_iopad0_uiov2_hi_1	PDB24DGZ	57.4
	INOUT	C/BE[5] (PCI バス)			
83		LEFT	vss_core_l21	PVSS1DGZ	79
		Core digital VSS(0v)			
84		LEFT	pci_iopad0_uiov2_hi_0	PDB24DGZ	57.4
	INOUT	C/BE[4] (PCI バス)			
85		LEFT	vss_io_pci4	PVSS2DGZ	259
		IO digital VSS(0v)			
86		LEFT	pci_iopad0_uio2	PDB24DGZ	57.4
	INOUT	PAR64 (PCI バス)			
87		LEFT	vdd_io_pci4	PVDD2DGZ	80
		IO digital VDD(2.5v)			
88		LEFT	pci_iopad0_uiov_hi_31	PDB24DGZ	57.4
	INOUT	AD[63] (PCI バス)			
89		LEFT	pci_iopad0_uiov_hi_30	PDB24DGZ	57.4
	INOUT	AD[62] (PCI バス)			
90		LEFT	pci_iopad0_uiov_hi_29	PDB24DGZ	57.4
	INOUT	AD[61] (PCI バス)			
91		LEFT	vss_core_l20	PVSS1DGZ	79
		Core digital VSS(0v)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
92		LEFT	pci_iopad0_uiov_hi_28	PDB24DGZ	57.4
	INOUT	AD[60] (PCI バス)			
93		LEFT	vdd_core_l11	PVDD1DGZ	79
		Core digital VDD(1.0v)			
94		LEFT	pci_iopad0_uiov_hi_27	PDB24DGZ	57.4
	INOUT	AD[59] (PCI バス)			
95		LEFT	vss_io_pci3	PVSS2DGZ	259
		IO digital VSS(0v)			
96		LEFT	pci_iopad0_uiov_hi_26	PDB24DGZ	57.4
	INOUT	AD[58] (PCI バス)			
97		LEFT	vdd_io_pci3	PVDD2DGZ	80
		IO digital VDD(2.5v)			
98		LEFT	pci_iopad0_uiov_hi_25	PDB24DGZ	57.4
	INOUT	AD[57] (PCI バス)			
99		LEFT	vss_core_l19	PVSS1DGZ	79
		Core digital VSS(0v)			
100		LEFT	pci_iopad0_uiov_hi_24	PDB24DGZ	57.4
	INOUT	AD[56] (PCI バス)			
101		LEFT	pci_iopad0_uiov_hi_23	PDB24DGZ	57.4
	INOUT	AD[55] (PCI バス)			
102		LEFT	pci_iopad0_uiov_hi_22	PDB24DGZ	57.4
	INOUT	AD[54] (PCI バス)			
103		LEFT	vdd_core_l10	PVDD1DGZ	79
		Core digital VDD(1.0v)			
104		LEFT	pci_iopad0_uiov_hi_21	PDB24DGZ	57.4
	INOUT	AD[53] (PCI バス)			
105		LEFT	pci_iopad0_uiov_hi_20	PDB24DGZ	57.4
	INOUT	AD[52] (PCI バス)			
106		LEFT	pci_iopad0_uiov_hi_19	PDB24DGZ	57.4
	INOUT	AD[51] (PCI バス)			
107		LEFT	vss_core_l18	PVSS1DGZ	79
		Core digital VSS(0v)			
108		LEFT	pci_iopad0_uiov_hi_18	PDB24DGZ	57.4
	INOUT	AD[50] (PCI バス)			
109		LEFT	pci_iopad0_uiov_hi_17	PDB24DGZ	57.4
	INOUT	AD[49] (PCI バス)			
110		LEFT	pci_iopad0_uiov_hi_16	PDB24DGZ	57.4
	INOUT	AD[48] (PCI バス)			
111		LEFT	vdd_core_l9	PVDD1DGZ	79
		Core digital VDD(1.0v)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
112		LEFT	pci_iopad0_uiov_hi_15	PDB24DGZ	57.4
	INOUT	AD[47] (PCI バス)			
113		LEFT	vss_io_pci2	PVSS2DGZ	259
		IO digital VSS(0v)			
114		LEFT	pci_iopad0_uiov_hi_14	PDB24DGZ	57.4
	INOUT	AD[46] (PCI バス)			
115		LEFT	vdd_io_pci2	PVDD2DGZ	80
		IO digital VDD(2.5v)			
116		LEFT	pci_iopad0_uiov_hi_13	PDB24DGZ	57.4
	INOUT	AD[45] (PCI バス)			
117		LEFT	vss_core_l17	PVSS1DGZ	79
		Core digital VSS(0v)			
118		LEFT	pci_iopad0_uiov_hi_12	PDB24DGZ	57.4
	INOUT	AD[44] (PCI バス)			
119		LEFT	vdd_core_l8	PVDD1DGZ	79
		Core digital VDD(1.0v)			
120		LEFT	pci_iopad0_uiov_hi_11	PDB24DGZ	57.4
	INOUT	AD[43] (PCI バス)			
121		LEFT	vss_io_pci1	PVSS2DGZ	259
		IO digital VSS(0v)			
122		LEFT	pci_iopad0_uiov_hi_10	PDB24DGZ	57.4
	INOUT	AD[42] (PCI バス)			
123		LEFT	vdd_io_pci1	PVDD2DGZ	80
		IO digital VDD(2.5v)			
124		LEFT	pci_iopad0_uiov_hi_9	PDB24DGZ	57.4
	INOUT	AD[41] (PCI バス)			
125		LEFT	vss_core_l16	PVSS1DGZ	79
		Core digital VSS(0v)			
126		LEFT	pci_iopad0_uiov_hi_8	PDB24DGZ	57.4
	INOUT	AD[40] (PCI バス)			
127		LEFT	pci_iopad0_uiov_hi_7	PDB24DGZ	57.4
	INOUT	AD[39] (PCI バス)			
128		LEFT	pci_iopad0_uiov_hi_6	PDB24DGZ	57.4
	INOUT	AD[38] (PCI バス)			
129		LEFT	vdd_core_l7	PVDD1DGZ	79
		Core digital VDD(1.0v)			
130		LEFT	pci_iopad0_uiov_hi_5	PDB24DGZ	57.4
	INOUT	AD[37] (PCI バス)			
131		LEFT	vss_core_l15	PVSS1DGZ	79
		Core digital VSS(0v)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
132		LEFT	pci_iopad0_uiov_hi_4	PDB24DGZ	57.4
	INOUT	AD[36] (PCI バス)			
133		LEFT	vss_io_pci0	PVSS2DGZ	259
		IO digital VSS(0v)			
134		LEFT	pci_iopad0_uiov_hi_3	PDB24DGZ	57.4
	INOUT	AD[35] (PCI バス)			
135		LEFT	vdd_io_pci0	PVDD2DGZ	80
		IO digital VDD(2.5v)			
136		LEFT	pci_iopad0_uiov_hi_2	PDB24DGZ	57.4
	INOUT	AD[34] (PCI バス)			
137		LEFT	vss_core_l14	PVSS1DGZ	79
		Core digital VSS(0v)			
138		LEFT	pci_iopad0_uiov_hi_1	PDB24DGZ	57.4
	INOUT	AD[33] (PCI バス)			
139		LEFT	vdd_core_l6	PVDD1DGZ	79
		Core digital VDD(1.0v)			
140		LEFT	pci_iopad0_uiov_hi_0	PDB24DGZ	57.4
	INOUT	AD[32] (PCI バス)			
141		LEFT	vss_io_other1	PVSS2DGZ	259
		IO digital VSS(0v)			
142		LEFT	pp_iopad0_pp_pwm_out_pad8	PDO04CDG	10.3
	OUTPUT	PWM 出力 channel8 (PWM ジェネレータ)			
143		LEFT	vss_core_l13	PVSS1DGZ	79
		Core digital VSS(0v)			
144		LEFT	pp_iopad0_pp_pwm_out_pad7	PDO04CDG	10.3
	OUTPUT	PWM 出力 channel7 (PWM ジェネレータ)			
145		LEFT	pp_iopad0_pp_pwm_out_pad6	PDO04CDG	10.3
	OUTPUT	PWM 出力 channel6 (PWM ジェネレータ)			
146		LEFT	pp_iopad0_pp_pwm_out_pad5	PDO04CDG	10.3
	OUTPUT	PWM 出力 channel5 (PWM ジェネレータ)			
147		LEFT	vss_core_l12	PVSS1DGZ	79
		Core digital VSS(0v)			
148		LEFT	pp_iopad0_pp_pwm_out_pad4	PDO04CDG	10.3
	OUTPUT	PWM 出力 channel4 (PWM ジェネレータ)			
149		LEFT	vdd_core_l5	PVDD1DGZ	79
		Core digital VDD(1.0v)			
150		LEFT	pp_iopad0_pp_pwm_out_pad3	PDO04CDG	10.3
	OUTPUT	PWM 出力 channel3 (PWM ジェネレータ)			
151		LEFT	vss_core_l11	PVSS1DGZ	79
		Core digital VSS(0v)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
152		LEFT	pp_iopad0_pp_pwm_out_pad2	PDO04CDG	10.3
	OUTPUT	PWM 出力 channel2 (PWM ジェネレータ)			
153		LEFT	vss_io_pp0	PVSS2DGZ	259
		IO digital VSS(0v)			
154		LEFT	pp_iopad0_pp_pwm_out_pad1	PDO04CDG	10.3
	OUTPUT	PWM 出力 channel1 (PWM ジェネレータ)			
155		LEFT	vdd_io_pp0	PVDD2DGZ	80
		IO digital VDD(2.5v)			
156		LEFT	pp_iopad0_pp_pwm_out_pad0	PDO04CDG	10.3
	OUTPUT	PWM 出力 channel0 (PWM ジェネレータ)			
157		LEFT	vss_core_l10	PVSS1DGZ	79
		Core digital VSS(0v)			
158		LEFT	pp_iopad0_pp_pz_pad8	PDIDGZ	
	INPUT	エンコーダ Z フェーズ入力 channel8 (パルスカウンタ)			
159		LEFT	pp_iopad0_pp_pz_pad7	PDIDGZ	
	INPUT	エンコーダ Z フェーズ入力 channel7 (パルスカウンタ)			
160		LEFT	pp_iopad0_pp_pz_pad6	PDIDGZ	
	INPUT	エンコーダ Z フェーズ入力 channel6 (パルスカウンタ)			
161		LEFT	vss_core_l9	PVSS1DGZ	79
		Core digital VSS(0v)			
162		LEFT	pp_iopad0_pp_pz_pad5	PDIDGZ	
	INPUT	エンコーダ Z フェーズ入力 channel5 (パルスカウンタ)			
163		LEFT	vdd_core_l4	PVDD1DGZ	79
		Core digital VDD(1.0v)			
164		LEFT	pp_iopad0_pp_pz_pad4	PDIDGZ	
	INPUT	エンコーダ Z フェーズ入力 channel4 (パルスカウンタ)			
165		LEFT	pp_iopad0_pp_pz_pad3	PDIDGZ	
	INPUT	エンコーダ Z フェーズ入力 channel3 (パルスカウンタ)			
166		LEFT	pp_iopad0_pp_pz_pad2	PDIDGZ	
	INPUT	エンコーダ Z フェーズ入力 channel2 (パルスカウンタ)			
167		LEFT	vss_core_l8	PVSS1DGZ	79
		Core digital VSS(0v)			
168		LEFT	pp_iopad0_pp_pz_pad1	PDIDGZ	
	INPUT	エンコーダ Z フェーズ入力 channel1 (パルスカウンタ)			
169		LEFT	pp_iopad0_pp_pz_pad0	PDIDGZ	
	INPUT	エンコーダ Z フェーズ入力 channel0 (パルスカウンタ)			
170		LEFT	pp_iopad0_pp_pb_pad8	PDIDGZ	
	INPUT	エンコーダ B フェーズ入力 channel8 (パルスカウンタ)			
171		LEFT	vss_core_l7	PVSS1DGZ	79
		Core digital VSS(0v)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
172		LEFT	pp_iopad0_pp_pb_pad7	PDIDGZ	
	INPUT	エンコーダ B フェーズ入力 channel7 (パルスカウンタ)			
173		LEFT	vdd_core_l3	PVDD1DGZ	79
		Core digital VDD(1.0v)			
174		LEFT	pp_iopad0_pp_pb_pad6	PDIDGZ	
	INPUT	エンコーダ B フェーズ入力 channel6 (パルスカウンタ)			
175		LEFT	pp_iopad0_pp_pb_pad5	PDIDGZ	
	INPUT	エンコーダ B フェーズ入力 channel5 (パルスカウンタ)			
176		LEFT	pp_iopad0_pp_pb_pad4	PDIDGZ	
	INPUT	エンコーダ B フェーズ入力 channel4 (パルスカウンタ)			
177		LEFT	vss_core_l6	PVSS1DGZ	79
		Core digital VSS(0v)			
178		LEFT	pp_iopad0_pp_pb_pad3	PDIDGZ	
	INPUT	エンコーダ B フェーズ入力 channel3 (パルスカウンタ)			
179		LEFT	pp_iopad0_pp_pb_pad2	PDIDGZ	
	INPUT	エンコーダ B フェーズ入力 channel2 (パルスカウンタ)			
180		LEFT	pp_iopad0_pp_pb_pad1	PDIDGZ	
	INPUT	エンコーダ B フェーズ入力 channel1 (パルスカウンタ)			
181		LEFT	vss_core_l5	PVSS1DGZ	79
		Core digital VSS(0v)			
182		LEFT	pp_iopad0_pp_pb_pad0	PDIDGZ	
	INPUT	エンコーダ B フェーズ入力 channel0 (パルスカウンタ)			
183		LEFT	vdd_core_l2	PVDD1DGZ	79
		Core digital VDD(1.0v)			
184		LEFT	pp_iopad0_pp_pa_pad8	PDIDGZ	
	INPUT	エンコーダ A フェーズ入力 channel8 (パルスカウンタ)			
185		LEFT	pp_iopad0_pp_pa_pad7	PDIDGZ	
	INPUT	エンコーダ A フェーズ入力 channel7 (パルスカウンタ)			
186		LEFT	pp_iopad0_pp_pa_pad6	PDIDGZ	
	INPUT	エンコーダ A フェーズ入力 channel6 (パルスカウンタ)			
187		LEFT	vss_core_l4	PVSS1DGZ	79
		Core digital VSS(0v)			
188		LEFT	pp_iopad0_pp_pa_pad5	PDIDGZ	
	INPUT	エンコーダ A フェーズ入力 channel5 (パルスカウンタ)			
189		LEFT	pp_iopad0_pp_pa_pad4	PDIDGZ	
	INPUT	エンコーダ A フェーズ入力 channel4 (パルスカウンタ)			
190		LEFT	pp_iopad0_pp_pa_pad3	PDIDGZ	
	INPUT	エンコーダ A フェーズ入力 channel3 (パルスカウンタ)			
191		LEFT	vss_core_l3	PVSS1DGZ	79
		Core digital VSS(0v)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
192		LEFT	pp_iopad0_pp_pa_pad2	PDIDGZ	
	INPUT	エンコーダ A フェーズ入力 channel2 (パルスカウンタ)			
193		LEFT	vdd_core_l1	PVDD1DGZ	79
		Core digital VDD(1.0v)			
194		LEFT	pp_iopad0_pp_pa_pad1	PDIDGZ	
	INPUT	エンコーダ A フェーズ入力 channel1 (パルスカウンタ)			
195		LEFT	vss_io_other0	PVSS2DGZ	259
		IO digital VSS(0v)			
196		LEFT	pp_iopad0_pp_pa_pad0	PDIDGZ	
	INPUT	エンコーダ A フェーズ入力 channel0 (パルスカウンタ)			
197		LEFT	vss_core_l2	PVSS1DGZ	79
		Core digital VSS(0v)			
198		LEFT	clk_iopad0_reset_outer	PDO24CDG	57.4
	OUTPUT	外部ユニット用リセット			
199		LEFT	vdd_core_la8	PVDD1DGZ	79
		Core digital VDD(1.0v)			
200		LEFT	vdd_core_la7	PVDD1DGZ	79
		Core digital VDD(1.0v)			
201		LEFT	vss_core_l1	PVSS1DGZ	79
		Core digital VSS(0v)			
202		LEFT	clk_iopad0_fout_a	PDO24CDG	57.4
	OUTPUT	PLL クロック出力			
203		LEFT	vdd_core_l0	PVDD1DGZ	79
		Core digital VDD(1.0v)			
204		LEFT	vss_core_l7	PVSS1DGZ	79
		Core digital VSS(0v)			
205		LEFT	vss_core_l0	PVSS1DGZ	79
		Core digital VSS(0v)			
206		LEFT	vdd_core_la6	PVDD1DGZ	79
		Core digital VDD(1.0v)			
207		LEFT	vss_core_la6	PVSS1DGZ	79
		Core digital VSS(0v)			
208		LEFT	vdd_core_la5	PVDD1DGZ	79
		Core digital VDD(1.0v)			
209		LEFT	vss_core_la5	PVSS1DGZ	79
		Core digital VSS(0v)			
210		LEFT	vss_core_la4	PVSS1DGZ	79
		Core digital VSS(0v)			
211		LEFT	clk_iopad0_clk_outer	PDO24CDG	57.4
	OUTPUT	外部ユニット用クロック			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
212		LEFT	vdd_io_clk0	PVDD2DGZ	80
		IO digital VDD(2.5v)			
213		LEFT	vdd_core_la4	PVDD1DGZ	79
		Core digital VDD(1.0v)			
214		LEFT	vdd_core_la3	PVDD1DGZ	79
		Core digital VDD(1.0v)			
215		LEFT	vss_core_la3	PVSS1DGZ	79
		Core digital VSS(0v)			
216		LEFT	vdd_core_la2	PVDD1DGZ	79
		Core digital VDD(1.0v)			
217		LEFT	vss_io_clk0	PVSS2DGZ	259
		IO digital VSS(0v)			
218		LEFT	vss_core_la2	PVSS1DGZ	79
		Core digital VSS(0v)			
219		LEFT	vdd_core_la1	PVDD2DGZ	80
		Core digital VDD(1.0v)			
220		LEFT	vss_core_la1	PVSS1DGZ	79
		Core digital VSS(0v)			
221		LEFT	vss_core_la0	PVSS1DGZ	79
		Core digital VSS(0v)			
222		LEFT	vdd_core_la0	PVDD1DGZ	79
		Core digital VDD(1.0v)			
223		LEFT	clk_iopad0_reset_in	PDUDGZ	
	INPUT	リセット入力			
224		LEFT	clk_iopad0_f_a0	PDDDGZ	
	INPUT	PLL 設定 Feedback Divider[0]			
225		LEFT	clk_iopad0_f_a1	PDDDGZ	
	INPUT	PLL 設定 Feedback Divider[1]			
226		LEFT	clk_iopad0_f_a2	PDDDGZ	
	INPUT	PLL 設定 Feedback Divider[2]			
227		LEFT	clk_iopad0_f_a3	PDDDGZ	
	INPUT	PLL 設定 Feedback Divider[3]			
228		LEFT	clk_iopad0_f_a4	PDDDGZ	
	INPUT	PLL 設定 Feedback Divider[4]			
229		LEFT	clk_iopad0_f_a5	PDDDGZ	
	INPUT	PLL 設定 Feedback Divider[5]			
230		LEFT	clk_iopad0_f_a6	PDDDGZ	
	INPUT	PLL 設定 Feedback Divider[6]			
231		LEFT	clk_iopad0_f_a7	PDDDGZ	
	INPUT	PLL 設定 Feedback Divider[7]			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
232		LEFT	clk_iopad0_fin_a	PDISDGZ	
	INPUT	クロック入力			
233		LEFT	clk_iopad0_bp_a	PDDDGZ	
	INPUT	PLL バイパス信号			
234		LEFT	clk_iopad0_r_a0	PDUDGZ	
	INPUT	PLL 設定 Input Divider[0]			
235		LEFT	clk_iopad0_r_a1	PDUDGZ	
	INPUT	PLL 設定 Input Divider[1]			
236		LEFT	clk_iopad0_r_a2	PDUDGZ	
	INPUT	PLL 設定 Input Divider[2]			
237		LEFT	clk_iopad0_r_a3	PDUDGZ	
	INPUT	PLL 設定 Input Divider[3]			
238		LEFT	clk_iopad0_r_a4	PDUDGZ	
	INPUT	PLL 設定 Input Divider[4]			
239		LEFT	clk_iopad0_oeb_a	PDDDGZ	
		PLL 設定 FOUT enable pin			
240		LEFT	clk_iopad0_od_a	PDDDGZ	
		PLL 設定 Output Divider			
241		LEFT	clk_iopad0_pd_a	PDDDGZ	
		PLL 設定 Power Down Mode			
242		LEFT	clk_iopad0_prdiode1	PRDIODE	
		Power Cut Cell			
243		LEFT	clk_iopad0_pvss2p_a	PVSS2P	
		IO analog VSS(0v)			
244		LEFT	clk_iopad0_pvdd2p_a	PVDD2P	
		IO analog VDD(2.5v)			
245		LEFT	clk_iopad0_pvdd1p_a1	PVDD1P	
		PLL analog VDD(2.5v)			
246		LEFT	clk_iopad0_pvdd1p_a0	PVDD1P	
		PLL analog VDD(2.5v)			
247		LEFT	clk_iopad0_pvss1p_a2	PVSS1P	
		PLL digital VSS(0v)			
248		LEFT	clk_iopad0_pvss1p_a1	PVSS1P	
		PLL analog VSS(0v)			
249		LEFT	clk_iopad0_pvss1p_a0	PVSS1P	
		PLL analog VSS(0v)			
250		LEFT	clk_iopad0_pvdd1pc_a0	PVDD1PC	
		PLL digital VDD(1.0v)			
251		LEFT	clk_iopad0_prdiode0	PRDIODE	
		Power Cut Cell			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
252		BOTTOM	uart_iopad0_uart1_stx_pad	PDO04CDG	10.3
	OUTPUT	TxD channel1 (UART)			
253		BOTTOM	uart_iopad0_uart1_srx_pad	PDIDGZ	
	INPUT	RxD channel1 (UART)			
254		BOTTOM	uart_iopad0_uart0_dtr_pad	PDO04CDG	10.3
	OUTPUT	DTR channel1 (UART)			
255		BOTTOM	uart_iopad0_uart0_rts_pad	PDO04CDG	10.3
	OUTPUT	RTS channel1 (UART)			
256		BOTTOM	uart_iopad0_uart0_stx_pad	PDO04CDG	10.3
	OUTPUT	TxD channel0 (UART)			
257		BOTTOM	uart_iopad0_uart0_dcd_pad	PDIDGZ	
	INPUT	DCD channel0 (UART)			
258		BOTTOM	uart_iopad0_uart0_ri_pad	PDIDGZ	
	INPUT	RI channel0 (UART)			
259		BOTTOM	uart_iopad0_uart0_dsr_pad	PDIDGZ	
	INPUT	DSR channel0 (UART)			
260		BOTTOM	uart_iopad0_uart0_cts_pad	PDIDGZ	
	INPUT	CTS channel0 (UART)			
261		BOTTOM	uart_iopad0_uart0_srx_pad	PDIDGZ	
	INPUT	RxD channel0 (UART)			
262		BOTTOM	sdram_iopad0_oe	PDO24CDG	57.4
	OUTPUT	Output Enable $< action : 0\ stop : 1 >$ (SDRAM)			
263		BOTTOM	sdram_iopad0_dir	PDO24CDG	57.4
	OUTPUT	Direction $< read : 1\ write : 0 >$ (SDRAM)			
264		BOTTOM	vss_io_other4	PVSS2DGZ	259
		IO digital VSS(0v)			
265		BOTTOM	sdram_iopad0_dq127	PDB24SDGZ	57.4
	INOUT	Dq[127] (SDRAM)			
266		BOTTOM	sdram_iopad0_dq063	PDB24SDGZ	57.4
	INOUT	Dq[63] (SDRAM)			
267		BOTTOM	sdram_iopad0_dq123	PDB24SDGZ	57.4
	INOUT	Dq[123] (SDRAM)			
268		BOTTOM	sdram_iopad0_dq059	PDB24SDGZ	57.4
	INOUT	Dq[59] (SDRAM)			
269		BOTTOM	sdram_iopad0_dq126	PDB24SDGZ	57.4
	INOUT	Dq[126] (SDRAM)			
270		BOTTOM	sdram_iopad0_dq062	PDB24SDGZ	57.4
	INOUT	Dq[62] (SDRAM)			
271		BOTTOM	sdram_iopad0_dq122	PDB24SDGZ	57.4
	INOUT	Dq[122] (SDRAM)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
272		BOTTOM	sdram_iopad0_dq058	PDB24SDGZ	57.4
	INOUT	Dq[58] (SDRAM)			
273		BOTTOM	sdram_iopad0_dqm15	PDO24CDG	57.4
	OUTPUT	Dm[15] (SDRAM)			
274		BOTTOM	sdram_iopad0_dqm07	PDO24CDG	57.4
	OUTPUT	Dm[7] (SDRAM)			
275		BOTTOM	sdram_iopad0_dqs015	PDB24SDGZ	57.4
	INOUT	Dqs[15] (SDRAM)			
276		BOTTOM	sdram_iopad0_dqs007	PDB24SDGZ	57.4
	INOUT	Dqs[7] (SDRAM)			
277		BOTTOM	sdram_iopad0_dq125	PDB24SDGZ	57.4
	INOUT	Dq[125] (SDRAM)			
278		BOTTOM	vdd_io_sdram17	PVDD2DGZ	80
		IO digital VDD(2.5v)			
279		BOTTOM	sdram_iopad0_dq061	PDB24SDGZ	57.4
	INOUT	Dq[61] (SDRAM)			
280		BOTTOM	vss_io_sdram17	PVSS2DGZ	259
		IO digital VSS(0v)			
281		BOTTOM	sdram_iopad0_dq121	PDB24SDGZ	57.4
	INOUT	Dq[121] (SDRAM)			
282		BOTTOM	vss_core_b35	PVSS1DGZ	79
		Core digital VSS(0v)			
283		BOTTOM	sdram_iopad0_dq057	PDB24SDGZ	57.4
	INOUT	Dq[57] (SDRAM)			
284		BOTTOM	vdd_core_b20	PVDD1DGZ	79
		Core digital VDD(1.0v)			
285		BOTTOM	sdram_iopad0_dq124	PDB24SDGZ	57.4
	INOUT	Dq[124] (SDRAM)			
286		BOTTOM	vss_core_b34	PVSS1DGZ	79
		Core digital VSS(0v)			
287		BOTTOM	sdram_iopad0_dq060	PDB24SDGZ	57.4
	INOUT	Dq[60] (SDRAM)			
288		BOTTOM	sdram_iopad0_dq120	PDB24SDGZ	57.4
	INOUT	Dq[120] (SDRAM)			
289		BOTTOM	sdram_iopad0_dq056	PDB24SDGZ	57.4
	INOUT	Dq[56] (SDRAM)			
290		BOTTOM	vss_core_b33	PVSS1DGZ	79
		Core digital VSS(0v)			
291		BOTTOM	sdram_iopad0_dq119	PDB24SDGZ	57.4
	INOUT	Dq[119] (SDRAM)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
292		BOTTOM	vss_io_sdram16	PVSS2DGZ	259
		IO digital VSS(0v)			
293		BOTTOM	sdram_iopad0_dq055	PDB24SDGZ	57.4
	INOUT	Dq[55] (SDRAM)			
294		BOTTOM	vdd_io_sdram16	PVDD2DGZ	80
		IO digital VDD(2.5v)			
295		BOTTOM	sdram_iopad0_dq115	PDB24SDGZ	57.4
	INOUT	Dq[115] (SDRAM)			
296		BOTTOM	vdd_core_b19	PVDD1DGZ	79
		Core digital VDD(1.0v)			
297		BOTTOM	sdram_iopad0_dq051	PDB24SDGZ	57.4
	INOUT	Dq[51] (SDRAM)			
298		BOTTOM	vss_core_b32	PVSS1DGZ	79
		Core digital VSS(0v)			
299		BOTTOM	sdram_iopad0_dq118	PDB24SDGZ	57.4
	INOUT	Dq[118] (SDRAM)			
300		BOTTOM	sdram_iopad0_dq054	PDB24SDGZ	57.4
	INOUT	Dq[54] (SDRAM)			
301		BOTTOM	sdram_iopad0_dq114	PDB24SDGZ	57.4
	INOUT	Dq[114] (SDRAM)			
302		BOTTOM	vss_core_b31	PVSS1DGZ	79
		Core digital VSS(0v)			
303		BOTTOM	sdram_iopad0_dq050	PDB24SDGZ	57.4
	INOUT	Dq[50] (SDRAM)			
304		BOTTOM	vss_io_sdram15	PVSS2DGZ	259
		IO digital VSS(0v)			
305		BOTTOM	sdram_iopad0_dqm14	PDO24CDG	57.4
	OUTPUT	Dm[14] (SDRAM)			
306		BOTTOM	vdd_io_sdram15	PVDD2DGZ	80
		IO digital VDD(2.5v)			
307		BOTTOM	sdram_iopad0_dqm06	PDO24CDG	57.4
	OUTPUT	Dm[6] (SDRAM)			
308		BOTTOM	vdd_core_b18	PVDD1DGZ	79
		Core digital VDD(1.0v)			
309		BOTTOM	sdram_iopad0_dqs014	PDB24SDGZ	57.4
	INOUT	Dqs[14] (SDRAM)			
310		BOTTOM	vss_core_b30	PVSS1DGZ	79
		Core digital VSS(0v)			
311		BOTTOM	sdram_iopad0_dqs006	PDB24SDGZ	57.4
	INOUT	Dqs[6] (SDRAM)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
312		BOTTOM	sdram_iopad0_dq117	PDB24SDGZ	57.4
	INOUT	Dq[117] (SDRAM)			
313		BOTTOM	sdram_iopad0_dq053	PDB24SDGZ	57.4
	INOUT	Dq[53] (SDRAM)			
314		BOTTOM	vss_core_b29	PVSS1DGZ	79
		Core digital VSS(0v)			
315		BOTTOM	sdram_iopad0_dq113	PDB24SDGZ	57.4
	INOUT	Dq[113] (SDRAM)			
316		BOTTOM	vdd_core_b17	PVDD1DGZ	79
		Core digital VDD(1.0v)			
317		BOTTOM	sdram_iopad0_dq049	PDB24SDGZ	57.4
	INOUT	Dq[49] (SDRAM)			
318		BOTTOM	vss_core_b28	PVSS1DGZ	79
		Core digital VSS(0v)			
319		BOTTOM	sdram_iopad0_dq116	PDB24SDGZ	57.4
	INOUT	Dq[116] (SDRAM)			
320		BOTTOM	sdram_iopad0_dq052	PDB24SDGZ	57.4
	INOUT	Dq[52] (SDRAM)			
321		BOTTOM	sdram_iopad0_dq112	PDB24SDGZ	57.4
	INOUT	Dq[112] (SDRAM)			
322		BOTTOM	vss_core_b27	PVSS1DGZ	79
		Core digital VSS(0v)			
323		BOTTOM	sdram_iopad0_dq048	PDB24SDGZ	57.4
	INOUT	Dq[48] (SDRAM)			
324		BOTTOM	vdd_core_b16	PVDD1DGZ	79
		Core digital VDD(1.0v)			
325		BOTTOM	sdram_iopad0_dq111	PDB24SDGZ	57.4
	INOUT	Dq[111] (SDRAM)			
326		BOTTOM	vss_core_b26	PVSS1DGZ	79
		Core digital VSS(0v)			
327		BOTTOM	sdram_iopad0_dq047	PDB24SDGZ	57.4
	INOUT	Dq[47] (SDRAM)			
328		BOTTOM	vss_io_sdram14	PVSS2DGZ	259
		IO digital VSS(0v)			
329		BOTTOM	sdram_iopad0_dq107	PDB24SDGZ	57.4
	INOUT	Dq[107] (SDRAM)			
330		BOTTOM	vdd_io_sdram14	PVDD2DGZ	80
		IO digital VDD(2.5v)			
331		BOTTOM	sdram_iopad0_dq043	PDB24SDGZ	57.4
	INOUT	Dq[43] (SDRAM)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
332		BOTTOM	vss_core_b25	PVSS1DGZ	79
		Core digital VSS(0v)			
333		BOTTOM	sdram_iopad0_dq110	PDB24SDGZ	57.4
	INOUT	Dq[110] (SDRAM)			
334		BOTTOM	vdd_core_b15	PVDD1DGZ	79
		Core digital VDD(1.0v)			
335		BOTTOM	sdram_iopad0_dq046	PDB24SDGZ	57.4
	INOUT	Dq[46] (SDRAM)			
336		BOTTOM	vss_core_b24	PVSS1DGZ	79
		Core digital VSS(0v)			
337		BOTTOM	sdram_iopad0_dq106	PDB24SDGZ	57.4
	INOUT	Dq[106] (SDRAM)			
338		BOTTOM	sdram_iopad0_dq042	PDB24SDGZ	57.4
	INOUT	Dq[42] (SDRAM)			
339		BOTTOM	sdram_iopad0_dqm13	PDO24CDG	57.4
	OUTPUT	Dm[13] (SDRAM)			
340		BOTTOM	vss_core_b23	PVSS1DGZ	79
		Core digital VSS(0v)			
341		BOTTOM	sdram_iopad0_dqm05	PDO24CDG	57.4
	OUTPUT	Dm[5] (SDRAM)			
342		BOTTOM	vdd_core_b14	PVDD1DGZ	79
		Core digital VDD(1.0v)			
343		BOTTOM	sdram_iopad0_dqs013	PDB24SDGZ	57.4
	INOUT	Dqs[13] (SDRAM)			
344		BOTTOM	vss_io_sdram13	PVSS2DGZ	259
		IO digital VSS(0v)			
345		BOTTOM	sdram_iopad0_dqs005	PDB24SDGZ	57.4
	INOUT	Dqs[5] (SDRAM)			
346		BOTTOM	vdd_io_sdram13	PVDD2DGZ	80
		IO digital VDD(2.5v)			
347		BOTTOM	sdram_iopad0_dq109	PDB24SDGZ	57.4
	INOUT	Dq[109] (SDRAM)			
348		BOTTOM	vss_core_b22	PVSS1DGZ	79
		Core digital VSS(0v)			
349		BOTTOM	sdram_iopad0_dq045	PDB24SDGZ	57.4
	INOUT	Dq[45] (SDRAM)			
350		BOTTOM	vdd_core_b13	PVDD1DGZ	79
		Core digital VDD(1.0v)			
351		BOTTOM	sdram_iopad0_dq105	PDB24SDGZ	57.4
	INOUT	Dq[105] (SDRAM)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
352		BOTTOM	sdram_iopad0_dq041	PDB24SDGZ	57.4
	INOUT	Dq[41] (SDRAM)			
353		BOTTOM	sdram_iopad0_dq108	PDB24SDGZ	57.4
	INOUT	Dq[108] (SDRAM)			
354		BOTTOM	vss_core_b21	PVSS1DGZ	79
		Core digital VSS(0v)			
355		BOTTOM	sdram_iopad0_dq044	PDB24SDGZ	57.4
	INOUT	Dq[44] (SDRAM)			
356		BOTTOM	vss_io_sdram12	PVSS2DGZ	259
		IO digital VSS(0v)			
357		BOTTOM	sdram_iopad0_dq104	PDB24SDGZ	57.4
	INOUT	Dq[104] (SDRAM)			
358		BOTTOM	vdd_io_sdram12	PVDD2DGZ	80
		IO digital VDD(2.5v)			
359		BOTTOM	sdram_iopad0_dq040	PDB24SDGZ	57.4
	INOUT	Dq[40] (SDRAM)			
360		BOTTOM	vdd_core_b12	PVDD1DGZ	79
		Core digital VDD(1.0v)			
361		BOTTOM	sdram_iopad0_dq103	PDB24SDGZ	57.4
	INOUT	Dq[103] (SDRAM)			
362		BOTTOM	vss_core_b20	PVSS1DGZ	79
		Core digital VSS(0v)			
363		BOTTOM	sdram_iopad0_dq039	PDB24SDGZ	57.4
	INOUT	Dq[39] (SDRAM)			
364		BOTTOM	sdram_iopad0_dq099	PDB24SDGZ	57.4
	INOUT	Dq[99] (SDRAM)			
365		BOTTOM	sdram_iopad0_dq035	PDB24SDGZ	57.4
	INOUT	Dq[35] (SDRAM)			
366		BOTTOM	sdram_iopad0_dq102	PDB24SDGZ	57.4
	INOUT	Dq[102] (SDRAM)			
367		BOTTOM	sdram_iopad0_dq038	PDB24SDGZ	57.4
	INOUT	Dq[38] (SDRAM)			
368		BOTTOM	vss_core_b19	PVSS1DGZ	79
		Core digital VSS(0v)			
369		BOTTOM	sdram_iopad0_dq098	PDB24SDGZ	57.4
	INOUT	Dq[98] (SDRAM)			
370		BOTTOM	sdram_iopad0_dq034	PDB24SDGZ	57.4
	INOUT	Dq[34] (SDRAM)			
371		BOTTOM	sdram_iopad0_dqm12	PDO24CDG	57.4
	OUTPUT	Dm[12] (SDRAM)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
372		BOTTOM	vdd_core_b11	PVDD1DGZ	79
		Core digital VDD(1.0v)			
373		BOTTOM	sdram_iopad0_dqm04	PDO24CDG	57.4
	OUTPUT	Dm[4] (SDRAM)			
374		BOTTOM	vss_io_sdram11	PVSS2DGZ	259
		IO digital VSS(0v)			
375		BOTTOM	sdram_iopad0_dqs012	PDB24SDGZ	57.4
	INOUT	Dqs[12] (SDRAM)			
376		BOTTOM	vdd_io_sdram11	PVDD2DGZ	80
		IO digital VDD(2.5v)			
377		BOTTOM	sdram_iopad0_dqs004	PDB24SDGZ	57.4
	INOUT	Dqs[4] (SDRAM)			
378		BOTTOM	vss_core_b18	PVSS1DGZ	79
		Core digital VSS(0v)			
379		BOTTOM	sdram_iopad0_dq101	PDB24SDGZ	57.4
	INOUT	Dq[101] (SDRAM)			
380		BOTTOM	sdram_iopad0_dq037	PDB24SDGZ	57.4
	INOUT	Dq[37] (SDRAM)			
381		BOTTOM	sdram_iopad0_dq097	PDB24SDGZ	57.4
	INOUT	Dq[97] (SDRAM)			
382		BOTTOM	vdd_core_b10	PVDD1DGZ	79
		Core digital VDD(1.0v)			
383		BOTTOM	sdram_iopad0_dq033	PDB24SDGZ	57.4
	INOUT	Dq[33] (SDRAM)			
384		BOTTOM	vss_core_b17	PVSS1DGZ	79
		Core digital VSS(0v)			
385		BOTTOM	sdram_iopad0_dq100	PDB24SDGZ	57.4
	INOUT	Dq[100] (SDRAM)			
386		BOTTOM	vss_io_sdram10	PVSS2DGZ	259
		IO digital VSS(0v)			
387		BOTTOM	sdram_iopad0_dq036	PDB24SDGZ	57.4
	INOUT	Dq[36] (SDRAM)			
388		BOTTOM	vdd_io_sdram10	PVDD2DGZ	80
		IO digital VDD(2.5v)			
389		BOTTOM	sdram_iopad0_dq096	PDB24SDGZ	57.4
	INOUT	Dq[96] (SDRAM)			
390		BOTTOM	vss_core_b16	PVSS1DGZ	79
		Core digital VSS(0v)			
391		BOTTOM	sdram_iopad0_dq032	PDB24SDGZ	57.4
	INOUT	Dq[32] (SDRAM)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
392		BOTTOM	vdd_core_b9	PVDD1DGZ	79
		Core digital VDD(1.0v)			
393		BOTTOM	sdram_iopad0_cs1	PDO24CDG	57.4
	OUTPUT	CS[1] (SDRAM)			
394		BOTTOM	sdram_iopad0_cs0	PDO24CDG	57.4
	OUTPUT	CS[0] (SDRAM)			
395		BOTTOM	sdram_iopad0_cas	PDO24CDG	57.4
	OUTPUT	CAS (SDRAM)			
396		BOTTOM	sdram_iopad0_we	PDO24CDG	57.4
	OUTPUT	WE (SDRAM)			
397		BOTTOM	sdram_iopad0_ras	PDO24CDG	57.4
	OUTPUT	RAS (SDRAM)			
398		BOTTOM	vss_core_b15	PVSS1DGZ	79
		Core digital VSS(0v)			
399		BOTTOM	sdram_iopad0_bank0	PDO24CDG	57.4
	OUTPUT	Ba[0] (SDRAM)			
400		BOTTOM	sdram_iopad0_bank1	PDO24CDG	57.4
	OUTPUT	Ba[1] (SDRAM)			
401		BOTTOM	sdram_iopad0_addr10	PDO24CDG	57.4
	OUTPUT	A[10] (SDRAM)			
402		BOTTOM	vdd_core_b8	PVDD1DGZ	79
		Core digital VDD(1.0v)			
403		BOTTOM	sdram_iopad0_addr00	PDO24CDG	57.4
	OUTPUT	A[0] (SDRAM)			
404		BOTTOM	vss_core_b14	PVSS1DGZ	79
		Core digital VSS(0v)			
405		BOTTOM	sdram_iopad0_addr01	PDO24CDG	57.4
	OUTPUT	A[1] (SDRAM)			
406		BOTTOM	vss_io_sdram9	PVSS2DGZ	259
		IO digital VSS(0v)			
407		BOTTOM	sdram_iopad0_addr02	PDO24CDG	57.4
	OUTPUT	A[2] (SDRAM)			
408		BOTTOM	vdd_io_sdram9	PVDD2DGZ	80
		IO digital VDD(2.5v)			
409		BOTTOM	sdram_iopad0_addr03	PDO24CDG	57.4
	OUTPUT	A[3] (SDRAM)			
410		BOTTOM	vss_core_b13	PVSS1DGZ	79
		Core digital VSS(0v)			
411		BOTTOM	sdram_iopad0_addr04	PDO24CDG	57.4
	OUTPUT	A[4] (SDRAM)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
412		BOTTOM	sdram_iopad0_addr05	PDO24CDG	57.4
	OUTPUT	A[5] (SDRAM)			
413		BOTTOM	sdram_iopad0_addr06	PDO24CDG	57.4
	OUTPUT	A[6] (SDRAM)			
414		BOTTOM	vdd_core_b7	PVDD1DGZ	79
		Core digital VDD(1.0v)			
415		BOTTOM	sdram_iopad0_addr07	PDO24CDG	57.4
	OUTPUT	A[7] (SDRAM)			
416		BOTTOM	vss_core_b12	PVSS1DGZ	79
		Core digital VSS(0v)			
417		BOTTOM	sdram_iopad0_addr08	PDO24CDG	57.4
	OUTPUT	A[8] (SDRAM)			
418		BOTTOM	sdram_iopad0_addr09	PDO24CDG	57.4
	OUTPUT	A[9] (SDRAM)			
419		BOTTOM	sdram_iopad0_addr11	PDO24CDG	57.4
	OUTPUT	A[11] (SDRAM)			
420		BOTTOM	sdram_iopad0_addr12	PDO24CDG	57.4
	OUTPUT	A[12] (SDRAM)			
421		BOTTOM	sdram_iopad0_cke	PDO24CDG	57.4
	OUTPUT	CKE (SDRAM)			
422		BOTTOM	vss_core_b11	PVSS1DGZ	79
		Core digital VSS(0v)			
423		BOTTOM	sdram_iopad0_dq095	PDB24SDGZ	57.4
	INOUT	Dq[95] (SDRAM)			
424		BOTTOM	vdd_core_b6	PVDD1DGZ	79
		Core digital VDD(1.0v)			
425		BOTTOM	sdram_iopad0_dq031	PDB24SDGZ	57.4
	INOUT	Dq[31] (SDRAM)			
426		BOTTOM	vss_io_sdram8	PVSS2DGZ	259
		IO digital VSS(0v)			
427		BOTTOM	sdram_iopad0_dq091	PDB24SDGZ	57.4
	INOUT	Dq[91] (SDRAM)			
428		BOTTOM	vdd_io_sdram8	PVDD2DGZ	80
		IO digital VDD(2.5v)			
429		BOTTOM	sdram_iopad0_dq027	PDB24SDGZ	57.4
	INOUT	Dq[27] (SDRAM)			
430		BOTTOM	vss_core_b10	PVSS1DGZ	79
		Core digital VSS(0v)			
431		BOTTOM	sdram_iopad0_dq094	PDB24SDGZ	57.4
	INOUT	Dq[94] (SDRAM)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
432		BOTTOM	sdram_iopad0_dq030	PDB24SDGZ	57.4
	INOUT	Dq[30] (SDRAM)			
433		BOTTOM	sdram_iopad0_dq090	PDB24SDGZ	57.4
	INOUT	Dq[90] (SDRAM)			
434		BOTTOM	vss_core_b9	PVSS1DGZ	79
		Core digital VSS(0v)			
435		BOTTOM	sdram_iopad0_dq026	PDB24SDGZ	57.4
	INOUT	Dq[26] (SDRAM)			
436		BOTTOM	vdd_core_b5	PVDD1DGZ	79
		Core digital VDD(1.0v)			
437		BOTTOM	sdram_iopad0_dqm11	PDO24CDG	57.4
	OUTPUT	Dm[11] (SDRAM)			
438		BOTTOM	vss_io_sdram7	PVSS2DGZ	259
		IO digital VSS(0v)			
439		BOTTOM	sdram_iopad0_dqm03	PDO24CDG	57.4
	OUTPUT	Dm[3] (SDRAM)			
440		BOTTOM	vdd_io_sdram7	PVDD2DGZ	80
		IO digital VDD(2.5v)			
441		BOTTOM	sdram_iopad0_dqs011	PDB24SDGZ	57.4
	INOUT	Dqs[11] (SDRAM)			
442		BOTTOM	vss_core_b8	PVSS1DGZ	79
		Core digital VSS(0v)			
443		BOTTOM	sdram_iopad0_dqs003	PDB24SDGZ	57.4
	INOUT	Dqs[3] (SDRAM)			
444		BOTTOM	sdram_iopad0_dq093	PDB24SDGZ	57.4
	INOUT	Dq[93] (SDRAM)			
445		BOTTOM	sdram_iopad0_dq029	PDB24SDGZ	57.4
	INOUT	Dq[29] (SDRAM)			
446		BOTTOM	vdd_core_b4	PVDD1DGZ	79
		Core digital VDD(1.0v)			
447		BOTTOM	sdram_iopad0_dq089	PDB24SDGZ	57.4
	INOUT	Dq[89] (SDRAM)			
448		BOTTOM	vss_core_b7	PVSS1DGZ	79
		Core digital VSS(0v)			
449		BOTTOM	sdram_iopad0_dq025	PDB24SDGZ	57.4
	INOUT	Dq[25] (SDRAM)			
450		BOTTOM	sdram_iopad0_dq092	PDB24SDGZ	57.4
	INOUT	Dq[92] (SDRAM)			
451		BOTTOM	sdram_iopad0_dq028	PDB24SDGZ	57.4
	INOUT	Dq[28] (SDRAM)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
452		BOTTOM	vss_io_sdram6	PVSS2DGZ	259
		IO digital VSS(0v)			
453		BOTTOM	sdram_iopad0_dq088	PDB24SDGZ	57.4
	INOUT	Dq[88] (SDRAM)			
454		BOTTOM	vdd_io_sdram6	PVDD2DGZ	80
		IO digital VDD(2.5v)			
455		BOTTOM	sdram_iopad0_dq024	PDB24SDGZ	57.4
	INOUT	Dq[24] (SDRAM)			
456		BOTTOM	vss_core_b6	PVSS1DGZ	79
		Core digital VSS(0v)			
457		BOTTOM	sdram_iopad0_dq087	PDB24SDGZ	57.4
	INOUT	Dq[87] (SDRAM)			
458		BOTTOM	vdd_core_b3	PVDD1DGZ	79
		Core digital VDD(1.0v)			
459		BOTTOM	sdram_iopad0_dq023	PDB24SDGZ	57.4
	INOUT	Dq[23] (SDRAM)			
460		BOTTOM	sdram_iopad0_dq083	PDB24SDGZ	57.4
	INOUT	Dq[83] (SDRAM)			
461		BOTTOM	sdram_iopad0_dq019	PDB24SDGZ	57.4
	INOUT	Dq[19] (SDRAM)			
462		BOTTOM	sdram_iopad0_dq086	PDB24SDGZ	57.4
	INOUT	Dq[86] (SDRAM)			
463		BOTTOM	sdram_iopad0_dq022	PDB24SDGZ	57.4
	INOUT	Dq[22] (SDRAM)			
464		BOTTOM	vss_core_b5	PVSS1DGZ	79
		Core digital VSS(0v)			
465		BOTTOM	sdram_iopad0_dq082	PDB24SDGZ	57.4
	INOUT	Dq[82] (SDRAM)			
466		BOTTOM	sdram_iopad0_dq018	PDB24SDGZ	57.4
	INOUT	Dq[18] (SDRAM)			
467		BOTTOM	sdram_iopad0_dqm10	PDO24CDG	57.4
	OUTPUT	Dm[10] (SDRAM)			
468		BOTTOM	vdd_core_b2	PVDD1DGZ	79
		Core digital VDD(1.0v)			
469		BOTTOM	sdram_iopad0_dqm02	PDO24CDG	57.4
	OUTPUT	Dm[2] (SDRAM)			
470		BOTTOM	vss_core_b4	PVSS1DGZ	79
		Core digital VSS(0v)			
471		BOTTOM	sdram_iopad0_dqs010	PDB24SDGZ	57.4
	INOUT	Dqs[10] (SDRAM)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
472		BOTTOM	vss_io_sdram5	PVSS2DGZ	259
		IO digital VSS(0v)			
473		BOTTOM	sdram_iopad0_dqs002	PDB24SDGZ	57.4
	INOUT	Dqs[2] (SDRAM)			
474		BOTTOM	vdd_io_sdram5	PVDD2DGZ	80
		IO digital VDD(2.5v)			
475		BOTTOM	sdram_iopad0_dq085	PDB24SDGZ	57.4
	INOUT	Dq[85] (SDRAM)			
476		BOTTOM	vss_core_b3	PVSS1DGZ	79
		Core digital VSS(0v)			
477		BOTTOM	sdram_iopad0_dq021	PDB24SDGZ	57.4
	INOUT	Dq[21] (SDRAM)			
478		BOTTOM	sdram_iopad0_dq081	PDB24SDGZ	57.4
	INOUT	Dq[81] (SDRAM)			
479		BOTTOM	sdram_iopad0_dq017	PDB24SDGZ	57.4
	INOUT	Dq[17] (SDRAM)			
480		BOTTOM	vdd_core_b1	PVDD1DGZ	79
		Core digital VDD(1.0v)			
481		BOTTOM	sdram_iopad0_dq084	PDB24SDGZ	57.4
	INOUT	Dq[84] (SDRAM)			
482		BOTTOM	vss_core_b2	PVSS1DGZ	79
		Core digital VSS(0v)			
483		BOTTOM	sdram_iopad0_dq020	PDB24SDGZ	57.4
	INOUT	Dq[20] (SDRAM)			
484		BOTTOM	vss_io_sdram4	PVSS2DGZ	259
		IO digital VSS(0v)			
485		BOTTOM	sdram_iopad0_dq080	PDB24SDGZ	57.4
	INOUT	Dq[80] (SDRAM)			
486		BOTTOM	vdd_io_sdram4	PVDD2DGZ	80
		IO digital VDD(2.5v)			
487		BOTTOM	sdram_iopad0_dq016	PDB24SDGZ	57.4
	INOUT	Dq[16] (SDRAM)			
488		BOTTOM	vss_core_b1	PVSS1DGZ	79
		Core digital VSS(0v)			
489		BOTTOM	sdram_iopad0_clk_	PDO24CDG	57.4
	OUTPUT	Clk (SDRAM)			
490		BOTTOM	sdram_iopad0_clk	PDO24CDG	57.4
	OUTPUT	Clk_n (SDRAM)			
491		BOTTOM	sdram_iopad0_dq079	PDB24SDGZ	57.4
	INOUT	Dq[79] (SDRAM)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
492		BOTTOM	vdd_core_b0	PVDD1DGZ	79
		Core digital VDD(1.0v)			
493		BOTTOM	sdram_iopad0_dq015	PDB24SDGZ	57.4
	INOUT	Dq[15] (SDRAM)			
494		BOTTOM	vss_core_b0	PVSS1DGZ	79
		Core digital VSS(0v)			
495		BOTTOM	sdram_iopad0_dq075	PDB24SDGZ	57.4
	INOUT	Dq[75] (SDRAM)			
496		BOTTOM	sdram_iopad0_dq011	PDB24SDGZ	57.4
	INOUT	Dq[11] (SDRAM)			
497		BOTTOM	sdram_iopad0_dq078	PDB24SDGZ	57.4
	INOUT	Dq[78] (SDRAM)			
498		BOTTOM	vss_io_sdram3	PVSS2DGZ	259
		IO digital VSS(0v)			
499		BOTTOM	sdram_iopad0_dq014	PDB24SDGZ	57.4
	INOUT	Dq[14] (SDRAM)			
500		BOTTOM	vdd_io_sdram3	PVDD2DGZ	80
		IO digital VDD(2.5v)			
501		BOTTOM	sdram_iopad0_dq074	PDB24SDGZ	57.4
	INOUT	Dq[74] (SDRAM)			
502		BOTTOM	sdram_iopad0_dq010	PDB24SDGZ	57.4
	INOUT	Dq[10] (SDRAM)			
503		RIGHT	sdram_iopad0_dqm09	PDO24CDG	57.4
	OUTPUT	Dm[9] (SDRAM)			
504		RIGHT	sdram_iopad0_dqm01	PDO24CDG	57.4
	OUTPUT	Dm[1] (SDRAM)			
505		RIGHT	sdram_iopad0_dqs009	PDB24DGZ	57.4
	INOUT	Dqs[9] (SDRAM)			
506		RIGHT	sdram_iopad0_dqs001	PDB24DGZ	57.4
	INOUT	Dqs[1] (SDRAM)			
507		RIGHT	sdram_iopad0_dq077	PDB24SDGZ	57.4
	INOUT	Dq[77] (SDRAM)			
508		RIGHT	sdram_iopad0_dq013	PDB24SDGZ	57.4
	INOUT	Dq[13] (SDRAM)			
509		RIGHT	sdram_iopad0_dq073	PDB24SDGZ	57.4
	INOUT	Dq[73] (SDRAM)			
510		RIGHT	sdram_iopad0_dq009	PDB24SDGZ	57.4
	INOUT	Dq[9] (SDRAM)			
511		RIGHT	sdram_iopad0_dq076	PDB24SDGZ	57.4
	INOUT	Dq[76] (SDRAM)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
512		RIGHT	sdram_iopad0_dq012	PDB24SDGZ	57.4
	INOUT	Dq[12] (SDRAM)			
513		RIGHT	vss_io_sdram2	PVSS2DGZ	259
		IO digital VSS(0v)			
514		RIGHT	sdram_iopad0_dq072	PDB24SDGZ	57.4
	INOUT	Dq[72] (SDRAM)			
515		RIGHT	vdd_io_sdram2	PVDD2DGZ	80
		IO digital VDD(2.5v)			
516		RIGHT	sdram_iopad0_dq008	PDB24SDGZ	57.4
	INOUT	Dq[8] (SDRAM)			
517		RIGHT	vdd_core_r20	PVDD1DGZ	79
		Core digital VDD(1.0v)			
518		RIGHT	sdram_iopad0_dq071	PDB24SDGZ	57.4
	INOUT	Dq[71] (SDRAM)			
519		RIGHT	vss_core_r34	PVSS1DGZ	79
		Core digital VSS(0v)			
520		RIGHT	sdram_iopad0_dq007	PDB24SDGZ	57.4
	INOUT	Dq[7] (SDRAM)			
521		RIGHT	sdram_iopad0_dq067	PDB24SDGZ	57.4
	INOUT	Dq[67] (SDRAM)			
522		RIGHT	sdram_iopad0_dq003	PDB24SDGZ	57.4
	INOUT	Dq[3] (SDRAM)			
523		RIGHT	sdram_iopad0_dq070	PDB24SDGZ	57.4
	INOUT	Dq[70] (SDRAM)			
524		RIGHT	sdram_iopad0_dq006	PDB24SDGZ	57.4
	INOUT	Dq[6] (SDRAM)			
525		RIGHT	vss_core_r33	PVSS1DGZ	79
		Core digital VSS(0v)			
526		RIGHT	sdram_iopad0_dq066	PDB24SDGZ	57.4
	INOUT	Dq[66] (SDRAM)			
527		RIGHT	vdd_core_r19	PVDD1DGZ	79
		Core digital VDD(1.0v)			
528		RIGHT	sdram_iopad0_dq002	PDB24SDGZ	57.4
	INOUT	Dq[2] (SDRAM)			
529		RIGHT	vss_io_sdram1	PVSS2DGZ	259
		IO digital VSS(0v)			
530		RIGHT	sdram_iopad0_dqm08	PDO24CDG	57.4
	OUTPUT	Dm[8] (SDRAM)			
531		RIGHT	vdd_io_sdram1	PVDD2DGZ	80
		IO digital VDD(2.5v)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
532		RIGHT	sdram_iopad0_dqm00	PDO24CDG	57.4
	OUTPUT	Dm[0] (SDRAM)			
533		RIGHT	vss_core_r32	PVSS1DGZ	79
		Core digital VSS(0v)			
534		RIGHT	sdram_iopad0_dqs008	PDB24DGZ	57.4
	INOUT	Dqs[8] (SDRAM)			
535		RIGHT	sdram_iopad0_dqs000	PDB24DGZ	57.4
	INOUT	Dqs[0] (SDRAM)			
536		RIGHT	sdram_iopad0_dq069	PDB24SDGZ	57.4
	INOUT	Dq[61] (SDRAM)			
537		RIGHT	sdram_iopad0_dq005	PDB24SDGZ	57.4
	INOUT	Dq[5] (SDRAM)			
538		RIGHT	sdram_iopad0_dq065	PDB24SDGZ	57.4
	INOUT	Dq[65] (SDRAM)			
539		RIGHT	vss_core_r31	PVSS1DGZ	79
		Core digital VSS(0v)			
540		RIGHT	sdram_iopad0_dq001	PDB24SDGZ	57.4
	INOUT	Dq[1] (SDRAM)			
541		RIGHT	vdd_core_r18	PVDD1DGZ	79
		Core digital VDD(1.0v)			
542		RIGHT	sdram_iopad0_dq068	PDB24SDGZ	57.4
	INOUT	Dq[68] (SDRAM)			
543		RIGHT	vss_io_sdram0	PVSS2DGZ	259
		IO digital VSS(0v)			
544		RIGHT	sdram_iopad0_dq004	PDB24SDGZ	57.4
	INOUT	Dq[4] (SDRAM)			
545		RIGHT	vdd_io_sdram0	PVDD2DGZ	80
		IO digital VDD(2.5v)			
546		RIGHT	sdram_iopad0_dq064	PDB24SDGZ	57.4
	INOUT	Dq[64] (SDRAM)			
547		RIGHT	vss_core_r30	PVSS1DGZ	79
		Core digital VSS(0v)			
548		RIGHT	sdram_iopad0_dq000	PDB24SDGZ	57.4
	INOUT	Dq[0] (SDRAM)			
549		RIGHT	ext_iopad0_bit8	PDISDGZ	
	INPUT	外部バスサイズの指定: 8bit (外部バス)			
550		RIGHT	ext_iopad0_bit16	PDISDGZ	
	INPUT	外部バスサイズの指定: 16bit (外部バス)			
551		RIGHT	vdd_core_r17	PVDD1DGZ	79
		Core digital VDD(1.0v)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
552		RIGHT	ext_iopad0_dreq0	PDISDGZ	
	INPUT	外部バス DMA リクエスト 1 (外部バス)			
553		RIGHT	ext_iopad0_auto_rdy_en	PDDDGZ	
	INPUT	外部バス設定用信号線 0 で外部の ready を取り込まない 1 で取り入れる			
554		RIGHT	ext_iopad0_dreq1	PDISDGZ	
	INPUT	外部バス DMA リクエスト 2 (外部バス)			
555		RIGHT	vdd_io_ext0	PVDD2DGZ	80
		IO digital VDD(2.5v)			
556		RIGHT	ext_iopad0_req	PDISDGZ	
	INPUT	外部バスリクエスト (外部バス)			
557		RIGHT	vss_io_ext0	PVSS2DGZ	259
		IO digital VSS(0v)			
558		RIGHT	ext_iopad0_irq0	PDISDGZ	
	INPUT	外部割込み 0 (外部バス)			
559		RIGHT	vss_core_r28	PVSS1DGZ	79
		Core digital VSS(0v)			
560		RIGHT	ext_iopad0_irq1	PDISDGZ	
	INPUT	外部割込み 1 (外部バス)			
561		RIGHT	vdd_core_r16	PVDD1DGZ	79
		Core digital VDD(1.0v)			
562		RIGHT	ext_iopad0_oe	PDO24CDG	57.4
	OUTPUT	外部バス Output Enable (外部バス)			
563		RIGHT	ext_iopad0_ie	PDO24CDG	57.4
	OUTPUT	外部バス Input Enable (外部バス)			
564		RIGHT	ext_iopad0_data_dir	PDO24CDG	57.4
	OUTPUT	外部バス Data Direction (外部バス)			
565		RIGHT	vss_core_r27	PVSS1DGZ	79
		Core digital VSS(0v)			
566		RIGHT	ext_iopad0_dack0	PDO24CDG	57.4
	OUTPUT	外部バス DMA Acknowledge0 (外部バス)			
567		RIGHT	vss_io_ext1	PVSS2DGZ	259
		IO digital VSS(0v)			
568		RIGHT	ext_iopad0_dack1	PDO24CDG	57.4
	OUTPUT	外部バス DMA Acknowledge1 (外部バス)			
569		RIGHT	vdd_io_ext1	PVDD2DGZ	80
		IO digital VDD(2.5v)			
570		RIGHT	ext_iopad0_cs0	PDO24CDG	57.4
	OUTPUT	外部バスチップセレクト 0 (外部バス)			
571		RIGHT	vss_core_r26	PVSS1DGZ	79
		Core digital VSS(0v)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
572		RIGHT	ext_iopad0_cs1	PDO24CDG	57.4
	OUTPUT	外部バスチップセレクト 1 (外部バス)			
573		RIGHT	vdd_core_r15	PVDD1DGZ	79
		Core digital VDD(1.0v)			
574		RIGHT	ext_iopad0_grant	PDO24CDG	57.4
	OUTPUT	外部バスグラント (外部バス)			
575		RIGHT	ext_iopad0_addr2	PDB24SDGZ	57.4
	INOUT	外部バスアドレス [2] (外部バス)			
576		RIGHT	ext_iopad0_addr3	PDB24SDGZ	57.4
	INOUT	外部バスアドレス [3] (外部バス)			
577		RIGHT	vss_core_r25	PVSS1DGZ	79
		Core digital VSS(0v)			
578		RIGHT	ext_iopad0_addr4	PDB24SDGZ	57.4
	INOUT	外部バスアドレス [4] (外部バス)			
579		RIGHT	ext_iopad0_addr5	PDB24SDGZ	57.4
	INOUT	外部バスアドレス [5] (外部バス)			
580		RIGHT	ext_iopad0_addr6	PDB24SDGZ	57.4
	INOUT	外部バスアドレス [6] (外部バス)			
581		RIGHT	ext_iopad0_addr7	PDB24SDGZ	57.4
	INOUT	外部バスアドレス [7] (外部バス)			
582		RIGHT	ext_iopad0_addr8	PDB24SDGZ	57.4
	INOUT	外部バスアドレス [8] (外部バス)			
583		RIGHT	vdd_core_r14	PVDD1DGZ	79
		Core digital VDD(1.0v)			
584		RIGHT	ext_iopad0_addr9	PDB24SDGZ	57.4
	INOUT	外部バスアドレス [9] (外部バス)			
585		RIGHT	vss_core_r24	PVSS1DGZ	79
		Core digital VSS(0v)			
586		RIGHT	ext_iopad0_addr10	PDB24SDGZ	57.4
	INOUT	外部バスアドレス [10] (外部バス)			
587		RIGHT	ext_iopad0_addr11	PDB24SDGZ	57.4
	INOUT	外部バスアドレス [11] (外部バス)			
588		RIGHT	ext_iopad0_addr12	PDB24SDGZ	57.4
	INOUT	外部バスアドレス [12] (外部バス)			
589		RIGHT	ext_iopad0_addr13	PDB24SDGZ	57.4
	INOUT	外部バスアドレス [13] (外部バス)			
590		RIGHT	ext_iopad0_addr14	PDB24SDGZ	57.4
	INOUT	外部バスアドレス [14] (外部バス)			
591		RIGHT	vss_core_r23	PVSS1DGZ	79
		Core digital VSS(0v)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
592		RIGHT	ext_iopad0_addr15	PDB24SDGZ	57.4
	INOUT	外部バスアドレス [15] (外部バス)			
593		RIGHT	vdd_core_r13	PVDD1DGZ	79
		Core digital VDD(1.0v)			
594		RIGHT	ext_iopad0_addr16	PDB24SDGZ	57.4
	INOUT	外部バスアドレス [16] (外部バス)			
595		RIGHT	ext_iopad0_addr17	PDB24SDGZ	57.4
	INOUT	外部バスアドレス [17] (外部バス)			
596		RIGHT	ext_iopad0_addr18	PDB24SDGZ	57.4
	INOUT	外部バスアドレス [18] (外部バス)			
597		RIGHT	vss_core_r22	PVSS1DGZ	79
		Core digital VSS(0v)			
598		RIGHT	ext_iopad0_addr19	PDB24SDGZ	57.4
	INOUT	外部バスアドレス [19] (外部バス)			
599		RIGHT	vdd_io_ext2	PVDD2DGZ	80
		IO digital VDD(2.5v)			
600		RIGHT	ext_iopad0_addr20	PDB24SDGZ	57.4
	INOUT	外部バスアドレス [20] (外部バス)			
601		RIGHT	vss_io_ext2	PVSS2DGZ	259
		IO digital VSS(0v)			
602		RIGHT	ext_iopad0_addr21	PDB24SDGZ	57.4
	INOUT	外部バスアドレス [21] (外部バス)			
603		RIGHT	vdd_core_r12	PVDD1DGZ	79
		Core digital VDD(1.0v)			
604		RIGHT	ext_iopad0_addr22	PDB24SDGZ	57.4
	INOUT	外部バスアドレス [22] (外部バス)			
605		RIGHT	vss_core_r21	PVSS1DGZ	79
		Core digital VSS(0v)			
606		RIGHT	ext_iopad0_addr23	PDB24SDGZ	57.4
	INOUT	外部バスアドレス [23] (外部バス)			
607		RIGHT	ext_iopad0_addr24	PDB24SDGZ	57.4
	INOUT	外部バスアドレス [24] (外部バス)			
608		RIGHT	ext_iopad0_addr25	PDB24SDGZ	57.4
	INOUT	外部バスアドレス [25] (外部バス)			
609		RIGHT	ext_iopad0_addr26	PDB24SDGZ	57.4
	INOUT	外部バスアドレス [26] (外部バス)			
610		RIGHT	ext_iopad0_addr27	PDB24SDGZ	57.4
	INOUT	外部バスアドレス [27] (外部バス)			
611		RIGHT	vss_core_r20	PVSS1DGZ	79
		Core digital VSS(0v)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
612		RIGHT	ext_iopad0_addr28	PDB24SDGZ	57.4
	INOUT	外部バスアドレス [28] (外部バス)			
613		RIGHT	vdd_core_r11	PVDD1DGZ	79
		Core digital VDD(1.0v)			
614		RIGHT	ext_iopad0_addr29	PDB24SDGZ	57.4
	INOUT	外部バスアドレス [29] (外部バス)			
615		RIGHT	vdd_io_ext3	PVDD2DGZ	80
		IO digital VDD(2.5v)			
616		RIGHT	ext_iopad0_addr30	PDB24SDGZ	57.4
	INOUT	外部バスアドレス [30] (外部バス)			
617		RIGHT	vss_io_ext3	PVSS2DGZ	259
		IO digital VSS(0v)			
618		RIGHT	ext_iopad0_addr31	PDB24SDGZ	57.4
	INOUT	外部バスアドレス [31] (外部バス)			
619		RIGHT	vss_core_r19	PVSS1DGZ	79
		Core digital VSS(0v)			
620		RIGHT	ext_iopad0_as	PDB24SDGZ	57.4
	INOUT	外部バスアドレスストローブ (外部バス)			
621		RIGHT	ext_iopad0_rw	PDB24SDGZ	57.4
	INOUT	外部バス Read / Write (外部バス)			
622		RIGHT	ext_iopad0_burst0	PDB24SDGZ	57.4
	INOUT	外部バスバーストモード [0] (外部バス)			
623		RIGHT	vss_core_r18	PVSS1DGZ	79
		Core digital VSS(0v)			
624		RIGHT	ext_iopad0_burst1	PDB24SDGZ	57.4
	INOUT	外部バスバーストモード [1] (外部バス)			
625		RIGHT	ext_iopad0_be0	PDB24SDGZ	57.4
	INOUT	外部バスバイトイネーブル [0] (外部バス)			
626		RIGHT	ext_iopad0_be1	PDB24SDGZ	57.4
	INOUT	外部バスバイトイネーブル [1] (外部バス)			
627		RIGHT	vdd_core_r10	PVDD1DGZ	79
		Core digital VDD(1.0v)			
628		RIGHT	ext_iopad0_be2	PDB24SDGZ	57.4
	INOUT	外部バスバイトイネーブル [2] (外部バス)			
629		RIGHT	vss_core_r17	PVSS1DGZ	79
		Core digital VSS(0v)			
630		RIGHT	ext_iopad0_be3	PDB24SDGZ	57.4
	INOUT	外部バスバイトイネーブル [3] (外部バス)			
631		RIGHT	vdd_io_ext4	PVDD2DGZ	80
		IO digital VDD(2.5v)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
632		RIGHT	ext_iopad0_br_ack0	PDB24SDGZ	57.4
	INOUT	外部バスバーストリクエスト Achnowledge[0] (外部バス)			
633		RIGHT	vss_io_ext4	PVSS2DGZ	259
		IO digital VSS(0v)			
634		RIGHT	ext_iopad0_br_ack1	PDB24SDGZ	57.4
	INOUT	外部バスバーストリクエスト Achnowledge[1] (外部バス)			
635		RIGHT	ext_iopad0_ready	PDB24SDGZ	57.4
	INOUT	外部バスレディ (外部バス)			
636		RIGHT	ext_iopad0_err	PDB24SDGZ	57.4
	INOUT	外部バスエラー (外部バス)			
637		RIGHT	vss_core_r16	PVSS1DGZ	79
		Core digital VSS(0v)			
638		RIGHT	ext_iopad0_data0	PDB24SDGZ	57.4
	INOUT	外部バスデータ [0]			
639		RIGHT	vdd_core_r9	PVDD1DGZ	79
		Core digital VDD(1.0v)			
640		RIGHT	ext_iopad0_data1	PDB24SDGZ	57.4
	INOUT	外部バスデータ [1]			
641		RIGHT	ext_iopad0_data2	PDB24SDGZ	57.4
	INOUT	外部バスデータ [2]			
642		RIGHT	ext_iopad0_data3	PDB24SDGZ	57.4
	INOUT	外部バスデータ [3]			
643		RIGHT	vss_core_r15	PVSS1DGZ	79
		Core digital VSS(0v)			
644		RIGHT	ext_iopad0_data4	PDB24SDGZ	57.4
	INOUT	外部バスデータ [4]			
645		RIGHT	vdd_io_ext5	PVDD2DGZ	80
		IO digital VDD(2.5v)			
646		RIGHT	ext_iopad0_data5	PDB24SDGZ	57.4
	INOUT	外部バスデータ [5]			
647		RIGHT	vss_io_ext5	PVSS2DGZ	259
		IO digital VSS(0v)			
648		RIGHT	ext_iopad0_data6	PDB24SDGZ	57.4
	INOUT	外部バスデータ [6]			
649		RIGHT	vdd_core_r8	PVDD1DGZ	79
		Core digital VDD(1.0v)			
650		RIGHT	ext_iopad0_data7	PDB24SDGZ	57.4
	INOUT	外部バスデータ [7]			
651		RIGHT	vss_core_r14	PVSS1DGZ	79
		Core digital VSS(0v)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
652		RIGHT	ext_iopad0_data8	PDB24SDGZ	57.4
	INOUT	外部バスデータ [8]			
653		RIGHT	ext_iopad0_data9	PDB24SDGZ	57.4
	INOUT	外部バスデータ [9]			
654		RIGHT	ext_iopad0_data10	PDB24SDGZ	57.4
	INOUT	外部バスデータ [10]			
655		RIGHT	ext_iopad0_data11	PDB24SDGZ	57.4
	INOUT	外部バスデータ [11]			
656		RIGHT	ext_iopad0_data12	PDB24SDGZ	57.4
	INOUT	外部バスデータ [12]			
657		RIGHT	vss_core_r13	PVSS1DGZ	79
		Core digital VSS(0v)			
658		RIGHT	ext_iopad0_data13	PDB24SDGZ	57.4
	INOUT	外部バスデータ [13]			
659		RIGHT	vdd_core_r7	PVDD1DGZ	79
		Core digital VDD(1.0v)			
660		RIGHT	ext_iopad0_data14	PDB24SDGZ	57.4
	INOUT	外部バスデータ [14]			
661		RIGHT	vdd_io_ext6	PVDD2DGZ	80
		IO digital VDD(2.5v)			
662		RIGHT	ext_iopad0_data15	PDB24SDGZ	57.4
	INOUT	外部バスデータ [15]			
663		RIGHT	vss_io_ext6	PVSS2DGZ	259
		IO digital VSS(0v)			
664		RIGHT	ext_iopad0_data16	PDB24SDGZ	57.4
	INOUT	外部バスデータ [16]			
665		RIGHT	vss_core_r12	PVSS1DGZ	79
		Core digital VSS(0v)			
666		RIGHT	ext_iopad0_data17	PDB24SDGZ	57.4
	INOUT	外部バスデータ [17]			
667		RIGHT	ext_iopad0_data18	PDB24SDGZ	57.4
	INOUT	外部バスデータ [18]			
668		RIGHT	ext_iopad0_data19	PDB24SDGZ	57.4
	INOUT	外部バスデータ [19]			
669		RIGHT	vss_core_r11	PVSS1DGZ	79
		Core digital VSS(0v)			
670		RIGHT	ext_iopad0_data20	PDB24SDGZ	57.4
	INOUT	外部バスデータ [20]			
671		RIGHT	vdd_core_r6	PVDD1DGZ	79
		Core digital VDD(1.0v)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
672		RIGHT	ext_iopad0_data21	PDB24SDGZ	57.4
	INOUT	外部バスデータ [21]			
673		RIGHT	ext_iopad0_data22	PDB24SDGZ	57.4
	INOUT	外部バスデータ [22]			
674		RIGHT	ext_iopad0_data23	PDB24SDGZ	57.4
	INOUT	外部バスデータ [23]			
675		RIGHT	vss_core_r10	PVSS1DGZ	79
		Core digital VSS(0v)			
676		RIGHT	ext_iopad0_data24	PDB24SDGZ	57.4
	INOUT	外部バスデータ [24]			
677		RIGHT	vdd_io_ext7	PVDD2DGZ	80
		IO digital VDD(2.5v)			
678		RIGHT	ext_iopad0_data25	PDB24SDGZ	57.4
	INOUT	外部バスデータ [25]			
679		RIGHT	vss_io_ext7	PVSS2DGZ	259
		IO digital VSS(0v)			
680		RIGHT	ext_iopad0_data26	PDB24SDGZ	57.4
	INOUT	外部バスデータ [26]			
681		RIGHT	vdd_core_r5	PVDD1DGZ	79
		Core digital VDD(1.0v)			
682		RIGHT	ext_iopad0_data27	PDB24SDGZ	57.4
	INOUT	外部バスデータ [27]			
683		RIGHT	vss_core_r9	PVSS1DGZ	79
		Core digital VSS(0v)			
684		RIGHT	ext_iopad0_data28	PDB24SDGZ	57.4
	INOUT	外部バスデータ [28]			
685		RIGHT	ext_iopad0_data29	PDB24SDGZ	57.4
	INOUT	外部バスデータ [29]			
686		RIGHT	ext_iopad0_data30	PDB24SDGZ	57.4
	INOUT	外部バスデータ [30]			
687		RIGHT	vss_io_other3	PVSS2DGZ	259
		IO digital VSS(0v)			
688		RIGHT	ext_iopad0_data31	PDB24SDGZ	57.4
	INOUT	外部バスデータ [31]			
689		RIGHT	vss_core_r8	PVSS1DGZ	79
		Core digital VSS(0v)			
690		RIGHT	link_sdram_iopad0_oe	PDO24CDG	57.4
	OUTPUT	Output Enable ¡action:0 stop:1¿ (LINK SDRAM)			
691		RIGHT	vdd_core_r4	PVDD1DGZ	79
		Core digital VDD(1.0v)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
692		RIGHT	link_sdram_iopad0_dir	PDO24CDG	57.4
	OUTPUT	Direction ¡read:1 write:0¿ (LINK SDRAM)			
693		RIGHT	vss_io_link_sdram7	PVSS2DGZ	259
		IO digital VSS(0v)			
694		RIGHT	link_sdram_iopad0_addr04	PDO24CDG	57.4
	OUTPUT	A[4] (LINK SDRAM)			
695		RIGHT	vdd_io_link_sdram7	PVDD2DGZ	80
		IO digital VDD(2.5v)			
696		RIGHT	link_sdram_iopad0_addr03	PDO24CDG	57.4
	OUTPUT	A[3] (LINK SDRAM)			
697		RIGHT	vss_core_r7	PVSS1DGZ	79
		Core digital VSS(0v)			
698		RIGHT	link_sdram_iopad0_addr05	PDO24CDG	57.4
	OUTPUT	A[5] (LINK SDRAM)			
699		RIGHT	link_sdram_iopad0_addr02	PDO24CDG	57.4
	OUTPUT	A[2] (LINK SDRAM)			
700		RIGHT	link_sdram_iopad0_addr06	PDO24CDG	57.4
	OUTPUT	A[6] (LINK SDRAM)			
701		RIGHT	link_sdram_iopad0_addr01	PDO24CDG	57.4
	OUTPUT	A[1] (LINK SDRAM)			
702		RIGHT	link_sdram_iopad0_addr07	PDO24CDG	57.4
	OUTPUT	A[7] (LINK SDRAM)			
703		RIGHT	vss_core_r6	PVSS1DGZ	79
		Core digital VSS(0v)			
704		RIGHT	link_sdram_iopad0_addr00	PDO24CDG	57.4
	OUTPUT	A[0] (LINK SDRAM)			
705		RIGHT	vdd_core_r3	PVDD1DGZ	79
		Core digital VDD(1.0v)			
706		RIGHT	link_sdram_iopad0_addr08	PDO24CDG	57.4
	OUTPUT	A[8] (LINK SDRAM)			
707		RIGHT	vss_io_link_sdram6	PVSS2DGZ	259
		IO digital VSS(0v)			
708		RIGHT	link_sdram_iopad0_addr10	PDO24CDG	57.4
	OUTPUT	A[10] (LINK SDRAM)			
709		RIGHT	vdd_io_link_sdram6	PVDD2DGZ	80
		IO digital VDD(2.5v)			
710		RIGHT	link_sdram_iopad0_addr09	PDO24CDG	57.4
	OUTPUT	A[9] (LINK SDRAM)			
711		RIGHT	vss_core_r5	PVSS1DGZ	79
		Core digital VSS(0v)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
712		RIGHT	link_sdram_iopad0_bank1	PDO24CDG	57.4
	OUTPUT	Ba[1] (LINK SDRAM)			
713		RIGHT	link_sdram_iopad0_addr11	PDO24CDG	57.4
	OUTPUT	A[11] (LINK SDRAM)			
714		RIGHT	link_sdram_iopad0_bank0	PDO24CDG	57.4
	OUTPUT	Ba[0] (LINK SDRAM)			
715		RIGHT	vdd_core_r2	PVDD1DGZ	79
		Core digital VDD(1.0v)			
716		RIGHT	link_sdram_iopad0_addr12	PDO24CDG	57.4
	OUTPUT	A[12] (LINK SDRAM)			
717		RIGHT	vss_core_r4	PVSS1DGZ	79
		Core digital VSS(0v)			
718		RIGHT	link_sdram_iopad0_cs1	PDO24CDG	57.4
	OUTPUT	CS[1] (LINK SDRAM)			
719		RIGHT	vss_io_link_sdram5	PVSS2DGZ	259
		IO digital VSS(0v)			
720		RIGHT	link_sdram_iopad0_cs0	PDO24CDG	57.4
	OUTPUT	CS[0] (LINK SDRAM)			
721		RIGHT	vdd_io_link_sdram5	PVDD2DGZ	80
		IO digital VDD(2.5v)			
722		RIGHT	link_sdram_iopad0_cke	PDO24CDG	57.4
	OUTPUT	CKE (LINK SDRAM)			
723		RIGHT	vss_core_r3	PVSS1DGZ	79
		Core digital VSS(0v)			
724		RIGHT	link_sdram_iopad0_ras	PDO24CDG	57.4
	OUTPUT	RAS (LINK SDRAM)			
725		RIGHT	link_sdram_iopad0_cas	PDO24CDG	57.4
	OUTPUT	CAS (LINK SDRAM)			
726		RIGHT	link_sdram_iopad0_we	PDO24CDG	57.4
	OUTPUT	WE (LINK SDRAM)			
727		RIGHT	link_sdram_iopad0_dqm3	PDO24CDG	57.4
	OUTPUT	Dm[3] (LINK SDRAM)			
728		RIGHT	link_sdram_iopad0_dqm1	PDO24CDG	57.4
	OUTPUT	Dm[1] (LINK SDRAM)			
729		RIGHT	vdd_core_r1	PVDD1DGZ	79
		Core digital VDD(1.0v)			
730		RIGHT	link_sdram_iopad0_dqm2	PDO24CDG	57.4
	OUTPUT	Dm[2] (LINK SDRAM)			
731		RIGHT	vss_core_r2	PVSS1DGZ	79
		Core digital VSS(0v)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
732		RIGHT	link_sdram_iopad0_dqm0	PDO24CDG	57.4
	OUTPUT	Dm[0] (LINK SDRAM)			
733		RIGHT	vss_io_link_sdram4	PVSS2DGZ	259
		IO digital VSS(0v)			
734		RIGHT	link_sdram_iopad0_dqs3	PDB24SDGZ	57.4
	INOUT	Dqs[3] (LINK SDRAM)			
735		RIGHT	vdd_io_link_sdram4	PVDD2DGZ	80
		IO digital VDD(2.5v)			
736		RIGHT	link_sdram_iopad0_dqs1	PDB24SDGZ	57.4
	INOUT	Dqs[1] (LINK SDRAM)			
737		RIGHT	vss_core_r1	PVSS1DGZ	79
		Core digital VSS(0v)			
738		RIGHT	link_sdram_iopad0_dqs2	PDB24SDGZ	57.4
	INOUT	Dqs[2] (LINK SDRAM)			
739		RIGHT	link_sdram_iopad0_dqs0	PDB24SDGZ	57.4
	INOUT	Dqs[0] (LINK SDRAM)			
740		RIGHT	link_sdram_iopad0_dq24	PDB24SDGZ	57.4
	INOUT	Dq[24] (LINK SDRAM)			
741		RIGHT	vdd_core_r0	PVDD1DGZ	79
		Core digital VDD(1.0v)			
742		RIGHT	link_sdram_iopad0_dq08	PDB24SDGZ	57.4
	INOUT	Dq[8] (LINK SDRAM)			
743		RIGHT	vss_core_r0	PVSS1DGZ	79
		Core digital VSS(0v)			
744		RIGHT	link_sdram_iopad0_dq23	PDB24SDGZ	57.4
	INOUT	Dq[23] (LINK SDRAM)			
745		RIGHT	vss_io_link_sdram3	PVSS2DGZ	259
		IO digital VSS(0v)			
746		RIGHT	link_sdram_iopad0_dq07	PDB24SDGZ	57.4
	INOUT	Dq[7] (LINK SDRAM)			
747		RIGHT	vdd_io_link_sdram3	PVDD2DGZ	80
		IO digital VDD(2.5v)			
748		RIGHT	link_sdram_iopad0_dq25	PDB24SDGZ	57.4
	INOUT	Dq[25] (LINK SDRAM)			
749		RIGHT	link_sdram_iopad0_dq09	PDB24SDGZ	57.4
	INOUT	Dq[9] (LINK SDRAM)			
750		RIGHT	link_sdram_iopad0_dq22	PDB24SDGZ	57.4
	INOUT	Dq[22] (LINK SDRAM)			
751		RIGHT	link_sdram_iopad0_dq06	PDB24SDGZ	57.4
	INOUT	Dq[6] (LINK SDRAM)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
752		RIGHT	link_sdram_iopad0_dq26	PDB24SDGZ	57.4
	INOUT	Dq[26] (LINK SDRAM)			
753		RIGHT	link_sdram_iopad0_dq10	PDB24SDGZ	57.4
	INOUT	Dq[10] (LINK SDRAM)			
754		TOP	link_sdram_iopad0_dq21	PDB24SDGZ	57.4
	INOUT	Dq[21] (LINK SDRAM)			
755		TOP	link_sdram_iopad0_dq05	PDB24SDGZ	57.4
	INOUT	Dq[5] (LINK SDRAM)			
756		TOP	vss_io_link_sdram2	PVSS2DGZ	259
		IO digital VSS(0v)			
757		TOP	link_sdram_iopad0_dq27	PDB24SDGZ	57.4
	INOUT	Dq[27] (LINK SDRAM)			
758		TOP	vdd_io_link_sdram2	PVDD2DGZ	80
		IO digital VDD(2.5v)			
759		TOP	link_sdram_iopad0_dq11	PDB24SDGZ	57.4
	INOUT	Dq[11] (LINK SDRAM)			
760		TOP	link_sdram_iopad0_dq20	PDB24SDGZ	57.4
	INOUT	Dq[20] (LINK SDRAM)			
761		TOP	link_sdram_iopad0_dq04	PDB24SDGZ	57.4
	INOUT	Dq[4] (LINK SDRAM)			
762		TOP	vss_core_t45	PVSS1DGZ	79
		Core digital VSS(0v)			
763		TOP	link_sdram_iopad0_dq28	PDB24SDGZ	57.4
	INOUT	Dq[28] (LINK SDRAM)			
764		TOP	link_sdram_iopad0_dq12	PDB24SDGZ	57.4
	INOUT	Dq[12] (LINK SDRAM)			
765		TOP	link_sdram_iopad0_dq19	PDB24SDGZ	57.4
	INOUT	Dq[19] (LINK SDRAM)			
766		TOP	vdd_core_t26	PVDD1DGZ	79
		Core digital VDD(1.0v)			
767		TOP	link_sdram_iopad0_dq03	PDB24SDGZ	57.4
	INOUT	Dq[3] (LINK SDRAM)			
768		TOP	vss_core_t44	PVSS1DGZ	79
		Core digital VSS(0v)			
769		TOP	link_sdram_iopad0_dq29	PDB24SDGZ	57.4
	INOUT	Dq[29] (LINK SDRAM)			
770		TOP	vss_io_link_sdram1	PVSS2DGZ	259
		IO digital VSS(0v)			
771		TOP	link_sdram_iopad0_dq13	PDB24SDGZ	57.4
	INOUT	Dq[13] (LINK SDRAM)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
772		TOP	vdd_io_link_sdram1	PVDD2DGZ	80
		IO digital VDD(2.5v)			
773		TOP	link_sdram_iopad0_dq18	PDB24SDGZ	57.4
	INOUT	Dq[18] (LINK SDRAM)			
774		TOP	link_sdram_iopad0_dq02	PDB24SDGZ	57.4
	INOUT	Dq[2] (LINK SDRAM)			
775		TOP	link_sdram_iopad0_dq30	PDB24SDGZ	57.4
	INOUT	Dq[30] (LINK SDRAM)			
776		TOP	vss_core_t43	PVSS1DGZ	79
		Core digital VSS(0v)			
777		TOP	link_sdram_iopad0_dq14	PDB24SDGZ	57.4
	INOUT	Dq[14] (LINK SDRAM)			
778		TOP	vdd_core_t25	PVDD1DGZ	79
		Core digital VDD(1.0v)			
779		TOP	link_sdram_iopad0_dq17	PDB24SDGZ	57.4
	INOUT	Dq[17] (LINK SDRAM)			
780		TOP	vss_io_link_sdram0	PVSS2DGZ	259
		IO digital VSS(0v)			
781		TOP	link_sdram_iopad0_dq01	PDB24SDGZ	57.4
	INOUT	Dq[1] (LINK SDRAM)			
782		TOP	vdd_io_link_sdram0	PVDD2DGZ	80
		IO digital VDD(2.5v)			
783		TOP	link_sdram_iopad0_dq31	PDB24SDGZ	57.4
	INOUT	Dq[31] (LINK SDRAM)			
784		TOP	vss_core_t42	PVSS1DGZ	79
		Core digital VSS(0v)			
785		TOP	link_sdram_iopad0_dq15	PDB24SDGZ	57.4
	INOUT	Dq[15] (LINK SDRAM)			
786		TOP	link_sdram_iopad0_dq16	PDB24SDGZ	57.4
	INOUT	Dq[16] (LINK SDRAM)			
787		TOP	link_sdram_iopad0_dq00	PDB24SDGZ	57.4
	INOUT	Dq[0] (LINK SDRAM)			
788		TOP	vdd_core_t24	PVDD1DGZ	79
		Core digital VDD(1.0v)			
789		TOP	link_iopad0_data_p_in4_iopad3	PDISDGZ	
	INPUT	(LINK)			
790		TOP	vss_core_t41	PVSS1DGZ	79
		Core digital VSS(0v)			
791		TOP	link_iopad0_data_p_in4_iopad2	PDISDGZ	
	INPUT	(LINK)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
792		TOP	vss_io_link7	PVSS2DGZ	259
		IO digital VSS(0v)			
793		TOP	link_iopad0_data_p_in4_iopad1	PDISDGZ	
	INPUT	(LINK)			
794		TOP	vdd_io_link7	PVDD2DGZ	80
		IO digital VDD(2.5v)			
795		TOP	link_iopad0_data_s_in_iopad4	PDISDGZ	
	INPUT	(LINK)			
796		TOP	link_iopad0_data_p_out4_iopad3	PDO24CDG	57.4
	OUTPUT	(LINK)			
797		TOP	link_iopad0_data_p_out4_iopad2	PDO24CDG	57.4
	OUTPUT	(LINK)			
798		TOP	vss_core_t40	PVSS1DGZ	79
		Core digital VSS(0v)			
799		TOP	link_iopad0_data_p_out4_iopad1	PDO24CDG	57.4
	OUTPUT	(LINK)			
800		TOP	vdd_core_t23	PVDD1DGZ	79
		Core digital VDD(1.0v)			
801		TOP	link_iopad0_data_s_out_iopad4	PDO24CDG	57.4
	OUTPUT	(LINK)			
802		TOP	link_iopad0_data_p_in3_iopad3	PDISDGZ	
	INPUT	(LINK)			
803		TOP	link_iopad0_data_p_in3_iopad2	PDISDGZ	
	INPUT	(LINK)			
804		TOP	vss_core_t39	PVSS1DGZ	79
		Core digital VSS(0v)			
805		TOP	link_iopad0_data_p_in3_iopad1	PDISDGZ	
	INPUT	(LINK)			
806		TOP	vss_io_link6	PVSS2DGZ	259
		IO digital VSS(0v)			
807		TOP	link_iopad0_data_s_in_iopad3	PDISDGZ	
	INPUT	(LINK)			
808		TOP	vdd_io_link6	PVDD2DGZ	80
		IO digital VDD(2.5v)			
809		TOP	link_iopad0_data_p_out3_iopad3	PDO24CDG	57.4
	OUTPUT	(LINK)			
810		TOP	vss_core_t38	PVSS1DGZ	79
		Core digital VSS(0v)			
811		TOP	link_iopad0_data_p_out3_iopad2	PDO24CDG	57.4
	OUTPUT	(LINK)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
812		TOP	vdd_core_t22	PVDD1DGZ	79
		Core digital VDD(1.0v)			
813		TOP	link_iopad0_data_p_out3_iopad1	PDO24CDG	57.4
	OUTPUT	(LINK)			
814		TOP	link_iopad0_data_s_out_iopad3	PDO24CDG	57.4
	OUTPUT	(LINK)			
815		TOP	link_iopad0_data_p_in2_iopad3	PDISDGZ	
	INPUT	(LINK)			
816		TOP	vss_core_t37	PVSS1DGZ	79
		Core digital VSS(0v)			
817		TOP	link_iopad0_data_p_in2_iopad2	PDISDGZ	
	INPUT	(LINK)			
818		TOP	vss_io_link5	PVSS2DGZ	259
		IO digital VSS(0v)			
819		TOP	link_iopad0_data_p_in2_iopad1	PDISDGZ	
	INPUT	(LINK)			
820		TOP	vdd_io_link5	PVDD2DGZ	80
		IO digital VDD(2.5v)			
821		TOP	link_iopad0_data_s_in_iopad2	PDISDGZ	
	INPUT	(LINK)			
822		TOP	vss_core_t36	PVSS1DGZ	79
		Core digital VSS(0v)			
823		TOP	link_iopad0_data_p_out2_iopad3	PDO24CDG	57.4
	OUTPUT	(LINK)			
824		TOP	vdd_core_t21	PVDD1DGZ	79
		Core digital VDD(1.0v)			
825		TOP	link_iopad0_data_p_out2_iopad2	PDO24CDG	57.4
	OUTPUT	(LINK)			
826		TOP	link_iopad0_data_p_out2_iopad1	PDO24CDG	57.4
	OUTPUT	(LINK)			
827		TOP	link_iopad0_data_s_out_iopad2	PDO24CDG	57.4
	OUTPUT	(LINK)			
828		TOP	vss_core_t35	PVSS1DGZ	79
		Core digital VSS(0v)			
829		TOP	link_iopad0_data_p_in1_iopad3	PDISDGZ	
	INPUT	(LINK)			
830		TOP	link_iopad0_data_p_in1_iopad2	PDISDGZ	
	INPUT	(LINK)			
831		TOP	link_iopad0_data_p_in1_iopad1	PDISDGZ	
	INPUT	(LINK)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
832		TOP	vdd_core_t20	PVDD1DGZ	79
		Core digital VDD(1.0v)			
833		TOP	link_iopad0_data_s_in_iopad1	PDISDGZ	
	INPUT	(LINK)			
834		TOP	vss_core_t34	PVSS1DGZ	79
		Core digital VSS(0v)			
835		TOP	link_iopad0_data_p_out1_iopad3	PDO24CDG	57.4
	OUTPUT	(LINK)			
836		TOP	vss_io_link4	PVSS2DGZ	259
		IO digital VSS(0v)			
837		TOP	link_iopad0_data_p_out1_iopad2	PDO24CDG	57.4
	OUTPUT	(LINK)			
838		TOP	vdd_io_link4	PVDD2DGZ	80
		IO digital VDD(2.5v)			
839		TOP	link_iopad0_data_p_out1_iopad1	PDO24CDG	57.4
	OUTPUT	(LINK)			
840		TOP	link_iopad0_data_s_out_iopad1	PDO24CDG	57.4
	OUTPUT	(LINK)			
841		TOP	link_iopad0_event_p_in4_iopad3	PDISDGZ	
	INPUT	(LINK)			
842		TOP	vss_core_t33	PVSS1DGZ	79
		Core digital VSS(0v)			
843		TOP	link_iopad0_event_p_in4_iopad2	PDISDGZ	
	INPUT	(LINK)			
844		TOP	vdd_core_t19	PVDD1DGZ	79
		Core digital VDD(1.0v)			
845		TOP	link_iopad0_event_p_in4_iopad1	PDISDGZ	
	INPUT	(LINK)			
846		TOP	vss_io_link3	PVSS2DGZ	259
		IO digital VSS(0v)			
847		TOP	link_iopad0_event_s_in_iopad4	PDISDGZ	
	INPUT	(LINK)			
848		TOP	vdd_io_link3	PVDD2DGZ	80
		IO digital VDD(2.5v)			
849		TOP	link_iopad0_event_p_out4_iopad3	PDO24CDG	57.4
	OUTPUT	(LINK)			
850		TOP	vss_core_t32	PVSS1DGZ	79
		Core digital VSS(0v)			
851		TOP	link_iopad0_event_p_out4_iopad2	PDO24CDG	57.4
	OUTPUT	(LINK)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
852		TOP	link_iopad0_event_p_out4_iopad1	PDO24CDG	57.4
	OUTPUT	(LINK)			
853		TOP	link_iopad0_event_s_out_pad4	PDO24CDG	57.4
	OUTPUT	(LINK)			
854		TOP	link_iopad0_event_p_in3_iopad3	PDISDGZ	
	INPUT	(LINK)			
855		TOP	link_iopad0_event_p_in3_iopad2	PDISDGZ	
	INPUT	(LINK)			
856		TOP	vss_core_t31	PVSS1DGZ	79
		Core digital VSS(0v)			
857		TOP	link_iopad0_event_p_in3_iopad1	PDISDGZ	
	INPUT	(LINK)			
858		TOP	vss_io_link2	PVSS2DGZ	259
		IO digital VSS(0v)			
859		TOP	link_iopad0_event_s_in_iopad3	PDISDGZ	
	INPUT	(LINK)			
860		TOP	vdd_io_link2	PVDD2DGZ	80
		IO digital VDD(2.5v)			
861		TOP	link_iopad0_event_p_out3_iopad3	PDO24CDG	57.4
	OUTPUT	(LINK)			
862		TOP	vdd_core_t18	PVDD1DGZ	79
		Core digital VDD(1.0v)			
863		TOP	link_iopad0_event_p_out3_iopad2	PDO24CDG	57.4
	OUTPUT	(LINK)			
864		TOP	vss_core_t30	PVSS1DGZ	79
		Core digital VSS(0v)			
865		TOP	link_iopad0_event_p_out3_iopad1	PDO24CDG	57.4
	OUTPUT	(LINK)			
866		TOP	link_iopad0_event_s_out_pad3	PDO24CDG	57.4
	OUTPUT	(LINK)			
867		TOP	link_iopad0_event_p_in2_iopad3	PDISDGZ	
	INPUT	(LINK)			
868		TOP	vdd_core_t17	PVDD1DGZ	79
		Core digital VDD(1.0v)			
869		TOP	link_iopad0_event_p_in2_iopad2	PDISDGZ	
	INPUT	(LINK)			
870		TOP	vss_core_t29	PVSS1DGZ	79
		Core digital VSS(0v)			
871		TOP	link_iopad0_event_p_in2_iopad1	PDISDGZ	
	INPUT	(LINK)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
872		TOP	vss_io_link1	PVSS2DGZ	259
		IO digital VSS(0v)			
873		TOP	link_iopad0_event_s_in_iopad2	PDISDGZ	
	INPUT	(LINK)			
874		TOP	vdd_io_link1	PVDD2DGZ	80
		IO digital VDD(2.5v)			
875		TOP	link_iopad0_event_p_out2_iopad3	PDO24CDG	57.4
	OUTPUT	(LINK)			
876		TOP	vss_core_t28	PVSS1DGZ	79
		Core digital VSS(0v)			
877		TOP	link_iopad0_event_p_out2_iopad2	PDO24CDG	57.4
	OUTPUT	(LINK)			
878		TOP	link_iopad0_event_p_out2_iopad1	PDO24CDG	57.4
	OUTPUT	(LINK)			
879		TOP	link_iopad0_event_s_out_pad2	PDO24CDG	57.4
	OUTPUT	(LINK)			
880		TOP	vdd_core_t16	PVDD1DGZ	79
		Core digital VDD(1.0v)			
881		TOP	link_iopad0_event_p_in1_iopad3	PDISDGZ	
	INPUT	(LINK)			
882		TOP	vss_core_t27	PVSS1DGZ	79
		Core digital VSS(0v)			
883		TOP	link_iopad0_event_p_in1_iopad2	PDISDGZ	
	INPUT	(LINK)			
884		TOP	vss_io_link0	PVSS2DGZ	259
		IO digital VSS(0v)			
885		TOP	link_iopad0_event_p_in1_iopad1	PDISDGZ	
	INPUT	(LINK)			
886		TOP	vdd_io_link0	PVDD2DGZ	80
		IO digital VDD(2.5v)			
887		TOP	link_iopad0_event_s_in_iopad1	PDISDGZ	
	INPUT	(LINK)			
888		TOP	link_iopad0_event_p_out1_iopad3	PDO24CDG	57.4
	OUTPUT	(LINK)			
889		TOP	link_iopad0_event_p_out1_iopad2	PDO24CDG	57.4
	OUTPUT	(LINK)			
890		TOP	vss_core_t26	PVSS1DGZ	79
		Core digital VSS(0v)			
891		TOP	link_iopad0_event_p_out1_iopad1	PDO24CDG	57.4
	OUTPUT	(LINK)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
892		TOP	vdd_core_t15	PVDD1DGZ	79
		Core digital VDD(1.0v)			
893		TOP	link_iopad0_event_s_out_pad1	PDO24CDG	57.4
	OUTPUT	(LINK)			
894		TOP	ieee1394_iopad0_ieee_d7	PDB04DGZ	10.3
	INOUT	Data[7] (IEEE1394)			
895		TOP	ieee1394_iopad0_ieee_d6	PDB04DGZ	10.3
	INOUT	Data[6] (IEEE1394)			
896		TOP	vss_core_t25	PVSS1DGZ	79
		Core digital VSS(0v)			
897		TOP	ieee1394_iopad0_ieee_d5	PDB04DGZ	10.3
	INOUT	Data[5] (IEEE1394)			
898		TOP	ieee1394_iopad0_ieee_d4	PDB04DGZ	10.3
	INOUT	Data[4] (IEEE1394)			
899		TOP	ieee1394_iopad0_ieee_d3	PDB04DGZ	10.3
	INOUT	Data[3] (IEEE1394)			
900		TOP	vdd_core_t14	PVDD1DGZ	79
		Core digital VDD(1.0v)			
901		TOP	ieee1394_iopad0_ieee_d2	PDB04DGZ	10.3
	INOUT	Data[2] (IEEE1394)			
902		TOP	vss_core_t24	PVSS1DGZ	79
		Core digital VSS(0v)			
903		TOP	ieee1394_iopad0_ieee_d1	PDB04DGZ	10.3
	INOUT	Data[1] (IEEE1394)			
904		TOP	vss_io_ieee0	PVSS2DGZ	259
		IO digital VSS(0v)			
905		TOP	ieee1394_iopad0_ieee_d0	PDB04DGZ	10.3
	INOUT	Data[0] (IEEE1394)			
906		TOP	vdd_io_ieee0	PVDD2DGZ	80
		IO digital VDD(2.5v)			
907		TOP	ieee1394_iopad0_ieee_isox	PDO04CDG	10.3
	OUTPUT	Isolation Control (IEEE1394)			
908		TOP	vss_core_t23	PVSS1DGZ	79
		Core digital VSS(0v)			
909		TOP	ieee1394_iopad0_ieee_pc2	PDO04CDG	10.3
	OUTPUT	Power Class[2] (IEEE1394)			
910		TOP	vdd_core_t13	PVDD1DGZ	79
		Core digital VDD(1.0v)			
911		TOP	ieee1394_iopad0_ieee_pc1	PDO04CDG	10.3
	OUTPUT	Power Class[1] (IEEE1394)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
912		TOP	vss_core_t22	PVSS1DGZ	79
		Core digital VSS(0v)			
913		TOP	ieee1394_iopad0_ieee_pc0	PDO04CDG	10.3
	OUTPUT	Power Class[0] (IEEE1394)			
914		TOP	ieee1394_iopad0_ieee_pd	PDO04CDG	10.3
	OUTPUT	PD (IEEE1394)			
915		TOP	ieee1394_iopad0_ieee_lps	PDO04CDG	10.3
	OUTPUT	LPS (IEEE1394)			
916		TOP	vss_core_t21	PVSS1DGZ	79
		Core digital VSS(0v)			
917		TOP	ieee1394_iopad0_ieee_lreq	PDO04CDG	10.3
	OUTPUT	LREQ (IEEE1394)			
918		TOP	vdd_core_t12	PVDD1DGZ	79
		Core digital VDD(1.0v)			
919		TOP	ieee1394_iopad0_ieee_cna	PDIDGZ	
	INPUT	Cable Not Active (IEEE1394)			
920		TOP	vss_core_t20	PVSS1DGZ	79
		Core digital VSS(0v)			
921		TOP	ieee1394_iopad0_ieee_lkon	PDIDGZ	
	INPUT	Link On (IEEE1394)			
922		TOP	vss_core_t19	PVSS1DGZ	79
		Core digital VSS(0v)			
923		TOP	ieee1394_iopad0_ieee_ctl1	PDB04DGZ	10.3
	INOUT	CTL[1] (IEEE1394)			
924		TOP	vdd_core_t11	PVDD1DGZ	79
		Core digital VDD(1.0v)			
925		TOP	ieee1394_iopad0_ieee_ctl0	PDB04DGZ	10.3
	INOUT	CTL[0] (IEEE1394)			
926		TOP	vss_core_t18	PVSS1DGZ	79
		Core digital VSS(0v)			
927		TOP	ieee1394_iopad0_ieee_sclk	PDIDGZ	
	INPUT	Syclk (IEEE1394)			
928		TOP	vss_core_t17	PVSS1DGZ	79
		Core digital VSS(0v)			
929		TOP	ieee1394_iopad0_ieee_rstx	PDO04CDG	10.3
	OUTPUT	RST (IEEE1394)			
930		TOP	vss_io_other2	PVSS2DGZ	259
		IO digital VSS(0v)			
931		TOP	usb_iopad0_usb_db168	PDO04CDG	10.3
	OUTPUT	Data Bus 16 / 8 (USB)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
932		TOP	vdd_core_t10	PVDD1DGZ	79
		Core digital VDD(1.0v)			
933		TOP	vss_io_other5	PVSS2DGZ	80
		IO digital VSS(0v)			
934		TOP	vss_core_t16	PVSS1DGZ	79
		Core digital VSS(0v)			
935		TOP	usb_iopad0_usb_txval	PDO04CDG	10.3
	OUTPUT	TX Valid (USB)			
936		TOP	usb_iopad0_usb_term	PDO04CDG	10.3
	OUTPUT	Termination Select (USB)			
937		TOP	usb_iopad0_usb_xcvr	PDO04CDG	10.3
	OUTPUT	Transceiver Select (USB)			
938		TOP	vdd_core_t9	PVDD1DGZ	79
		Core digital VDD(1.0v)			
939		TOP	usb_iopad0_usb_sus	PDO04CDG	10.3
	OUTPUT	Suspend (USB)			
940		TOP	vss_core_t15	PVSS1DGZ	79
		Core digital VSS(0v)			
941		TOP	usb_iopad0_usb_op1	PDO04CDG	10.3
	OUTPUT	Operation Mode[1] (USB)			
942		TOP	vss_core_t14	PVSS1DGZ	79
		Core digital VSS(0v)			
943		TOP	usb_iopad0_usb_op0	PDO04CDG	10.3
	OUTPUT	Operation Mode[0] (USB)			
944		TOP	vdd_core_t8	PVDD1DGZ	79
		Core digital VDD(1.0v)			
945		TOP	usb_iopad0_usb_d15	PDD04DGZ	10.3
	INOUT	Data[15] (USB)			
946		TOP	vss_core_t13	PVSS1DGZ	79
		Core digital VSS(0v)			
947		TOP	usb_iopad0_usb_d14	PDD04DGZ	10.3
	INOUT	Data[14] (USB)			
948		TOP	vss_core_t12	PVSS1DGZ	79
		Core digital VSS(0v)			
949		TOP	usb_iopad0_usb_d13	PDD04DGZ	10.3
	INOUT	Data[13] (USB)			
950		TOP	vdd_core_t7	PVDD1DGZ	79
		Core digital VDD(1.0v)			
951		TOP	usb_iopad0_usb_d12	PDD04DGZ	10.3
	INOUT	Data[12] (USB)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
952		TOP	vss_core_t11	PVSS1DGZ	79
		Core digital VSS(0v)			
953		TOP	usb_iopad0_usb_d11	PDD04DGZ	10.3
	INOUT	Data[11] (USB)			
954		TOP	usb_iopad0_usb_d10	PDD04DGZ	10.3
	INOUT	Data[10] (USB)			
955		TOP	usb_iopad0_usb_d9	PDD04DGZ	10.3
	INOUT	Data[9] (USB)			
956		TOP	vdd_core_t6	PVDD1DGZ	79
		Core digital VDD(1.0v)			
957		TOP	usb_iopad0_usb_d8	PDD04DGZ	10.3
	INOUT	Data[8] (USB)			
958		TOP	vss_core_t10	PVSS1DGZ	79
		Core digital VSS(0v)			
959		TOP	usb_iopad0_usb_d7	PDD04DGZ	10.3
	INOUT	Data[7] (USB)			
960		TOP	vss_core_t9	PVSS1DGZ	79
		Core digital VSS(0v)			
961		TOP	usb_iopad0_usb_d6	PDD04DGZ	10.3
	INOUT	Data[6] (USB)			
962		TOP	vdd_core_t5	PVDD1DGZ	79
		Core digital VDD(1.0v)			
963		TOP	usb_iopad0_usb_d5	PDD04DGZ	10.3
	INOUT	Data[5] (USB)			
964		TOP	vss_core_t8	PVSS1DGZ	79
		Core digital VSS(0v)			
965		TOP	usb_iopad0_usb_d4	PDD04DGZ	10.3
	INOUT	Data[4] (USB)			
966		TOP	vss_core_t7	PVSS1DGZ	79
		Core digital VSS(0v)			
967		TOP	usb_iopad0_usb_d3	PDD04DGZ	10.3
	INOUT	Data[3] (USB)			
968		TOP	vdd_core_t4	PVDD1DGZ	79
		Core digital VDD(1.0v)			
969		TOP	usb_iopad0_usb_d2	PDD04DGZ	10.3
	INOUT	Data[2] (USB)			
970		TOP	vss_io_usb0	PVSS2DGZ	259
		IO digital VSS(0v)			
971		TOP	usb_iopad0_usb_d1	PDD04DGZ	10.3
	INOUT	Data[1] (USB)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
972		TOP	vss_core_t6	PVSS1DGZ	79
		Core digital VSS(0v)			
973		TOP	usb_iopad0_usb_psw	PDD04DGZ	10.3
	INOUT	PSW (USB)			
974		TOP	vdd_io_usb0	PVDD2DGZ	80
		IO digital VDD(2.5v)			
975		TOP	usb_iopad0_usb_d0	PDD04DGZ	10.3
	INOUT	Data[0] (USB)			
976		TOP	vss_core_t5	PVSS1DGZ	79
		Core digital VSS(0v)			
977		TOP	usb_iopad0_usb_txready	PDIDGZ	
	INPUT	Transmit Data Ready (USB)			
978		TOP	vdd_core_t3	PVDD1DGZ	79
		Core digital VDD(1.0v)			
979		TOP	usb_iopad0_usb_valh	PDD04DGZ	10.3
	INOUT	ValidH (USB)			
980		TOP	vss_core_t4	PVSS1DGZ	79
		Core digital VSS(0v)			
981		TOP	usb_iopad0_usb_rxval	PDIDGZ	
	INPUT	Receive Data Valid (USB)			
982		TOP	usb_iopad0_usb_rxac	PDIDGZ	
	INPUT	Receive Active (USB)			
983		TOP	usb_iopad0_usb_rxer	PDIDGZ	
	INPUT	Receive Error			
984		TOP	vdd_core_t2	PVDD1DGZ	79
		Core digital VDD(1.0v)			
985		TOP	usb_iopad0_usb_line1	PDIDGZ	
	INPUT	Line State[1] (USB)			
986		TOP	vss_core_t3	PVSS1DGZ	79
		Core digital VSS(0v)			
987		TOP	usb_iopad0_usb_line0	PDIDGZ	
	INPUT	Line State[0] (USB)			
988		TOP	vss_core_t2	PVSS1DGZ	79
		Core digital VSS(0v)			
989		TOP	usb_iopad0_usb_spbp	PDIDGZ	
	INPUT	SPBP (USB)			
990		TOP	vdd_core_t1	PVDD1DGZ	79
		Core digital VDD(1.0v)			
991		TOP	usb_iopad0_usb_vbus	PDIDGZ	
	INPUT	VBUS (USB)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
992		TOP	vss_core_t1	PVSS1DGZ	79
		Core digital VSS(0v)			
993		TOP	usb_iopad0_usb_clk	PDIDGZ	
	INPUT	CLK (USB)			
994		TOP	vdd_core_t0	PVDD1DGZ	79
		Core digital VDD(1.0v)			
995		TOP	usb_iopad0_usb_reset	PDO04CDG	10.3
	OUTPUT	Reset (USB)			
996		TOP	vss_core_t0	PVSS1DGZ	79
		Core digital VSS(0v)			
997		TOP	pci_iopad0_pinta	PDB24DGZ	57.4
	INOUT	INTA# (PCI)			
998		TOP	pci_iopad0_uprst	PDB24DGZ	57.4
	INOUT	RST# (PCI)			
999		TOP	pci_iopad0_upclk	PDB24DGZ	57.4
	INOUT	CLK (PCI)			
1000		TOP	vss_io_pci12	PVSS2DGZ	259
		IO digital VSS(0v)			
1001		TOP	pci_iopad0_uin2	PDB24DGZ	57.4
	INOUT	GNT# (PCI)			
1002		TOP	vdd_io_pci12	PVDD2DGZ	80
		IO digital VDD(2.5v)			
1003		TOP	pci_iopad0_uout1	PDB24DGZ	57.4
	INOUT	REQ# (PCI)			
1004		TOP	pci_iopad0_uiov_lo_31	PDB24DGZ	57.4
	INOUT	AD[31] (PCI)			

3

命令セット

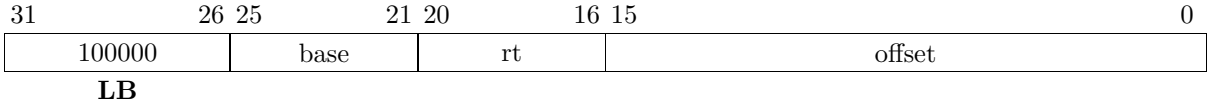
3.1 MIPS 互換の命令

Responsive Multithreaded Processor は MIPS 互換の命令をサポートしている．以下に MIPS 互換の命令を示す．

3.1.1 Load / Store 命令

LB Load Byte

8bit ロード MIPS I



ニーモニック:

LB rt, offset(base)

機能 :

GPR[rt] ← sign_extend(MEM[GPR[base] + offset])

例外 :

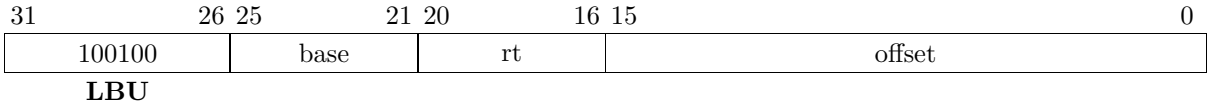
D-TLB No Entry Matched :
D-TLB Protection Error :
Data Address Miss Align (Load) :

概要 :

1byte をロードする．ロードされた値は 32bit に符号拡張される．

LBU Load Byte Unsigned

8bit 符号なしロード MIPS I



ニーモニック:

LBU rt, offset(base)

機能 :

GPR[rt] ← zero_extend(MEM[GPR[base] + offset])

例外 :

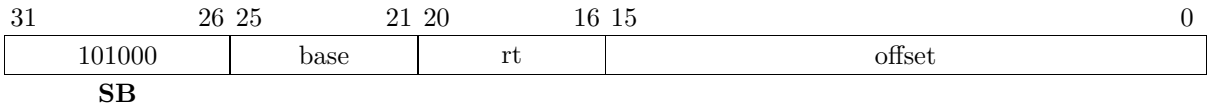
D-TLB No Entry Matched :
D-TLB Protection Error :
Data Address Miss Align (Load) :

概要 :

1byte をロードする．ロードされた値は 32bit に 0 拡張される．

SB Store Byte

8bit ストア MIPS I



ニーモニック:

SB rt, offset(base)

機能 :

MEM[GPR[base] + offset] ← GPR[rt]

例外 :

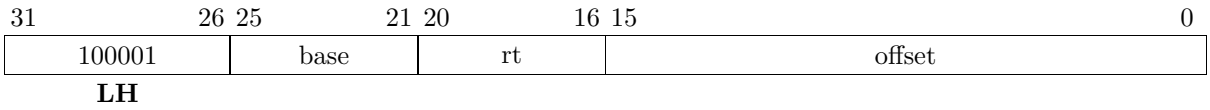
D-TLB No Entry Matched :
D-TLB Protection Error :
Data Address Miss Align (Store) :

概要 :

1byte をストアする.

LH Load Halfword

16bit ロード MIPS I



ニーモニック:

LH rt, offset(base)

機能 :

GPR[rt] ← sign_extend(MEM[GPR[base] + offset])

例外 :

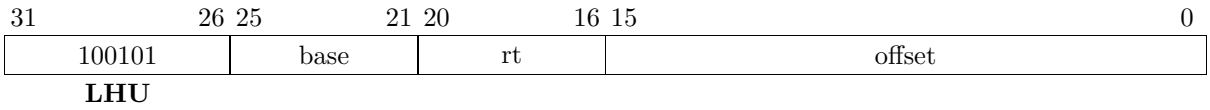
D-TLB No Entry Matched :
D-TLB Protection Error :
Data Address Miss Align (Load) :

概要 :

Half word をロードする．ロードされた値は 32bit に符号拡張される．

LHU Load Halfword Unsigned

16bit 符号なしロード MIPS I



ニーモニック:

LHU rt, offset(base)

機能 :

GPR[rt] ← zero_extend(MEM[GPR[base] + offset])

例外 :

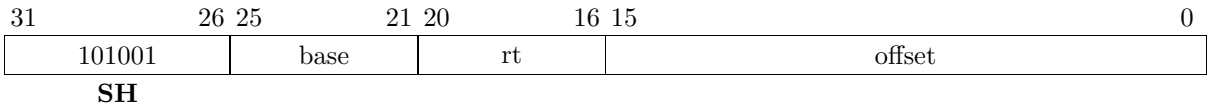
D-TLB No Entry Matched :
D-TLB Protection Error :
Data Address Miss Align (Load) :

概要 :

Half word をロードする．ロードされた値は 32bit に 0 拡張される．

SH Store Halfword

16bit ストア MIPS I



ニーモニック:

SH rt, offset(base)

機能 :

MEM[GPR[base] + offset] ← GPR[rt]

例外 :

D-TLB No Entry Matched :
D-TLB Protection Error :
Data Address Miss Align (Store) :

概要 :

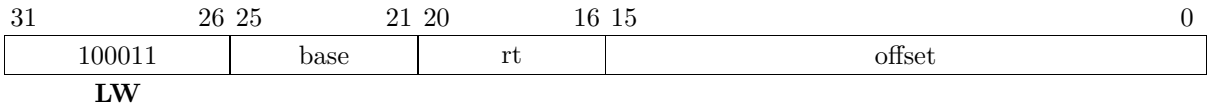
Half word をストアする.

LW

Load Word

32bit ロード

MIPS I



ニーモニック:

LW rt, offset(base)

機能 :

GPR[rt] ← sign_extend(MEM[GPR[base] + offset])

例外 :

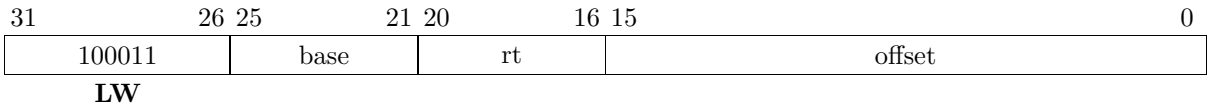
D-TLB No Entry Matched :
D-TLB Protection Error :
Data Address Miss Align (Load) :

概要 :

1Word をロードする.

SW — Store Word

32bit ロード — MIPS I



ニーモニック:

SW rt, offset(base)

機能 :

MEM[GPR[base] + offset] ← GPR[rt]

例外 :

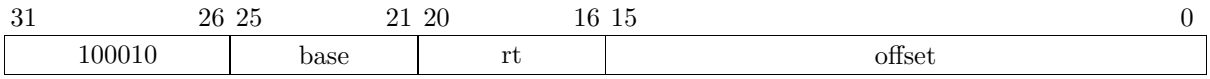
D-TLB No Entry Matched :
D-TLB Protection Error :
Data Address Miss Align (Store) :

概要 :

1Word をストアする.

LWL — Load Word Left

32bit unaligned ロード　　MIPS I



LWL

ニーモニック:

LWL rt, offset(base)

機能：

GPR[rt] ← merge(GPR[rt], MEM[GPR[base] + offset])

例外：

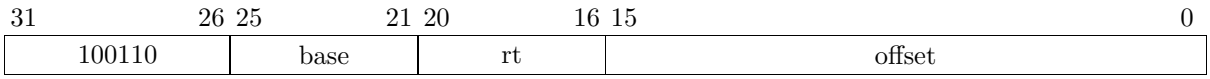
D-TLB No Entry Matched :
D-TLB Protection Error :

概要：

アラインされていないアドレスに対してロードを行う．LWR 命令との組み合わせにより，アラインされていないアドレスから 1Word ロードすることができる．

LWR Load Word Right

32bit unaligned ロード MIPS I



LWR

ニーモニック:

LWR rt, offset(base)

機能 :

GPR[rt] ← merge(MEM[GPR[base] + offset], GPR[rt])

例外 :

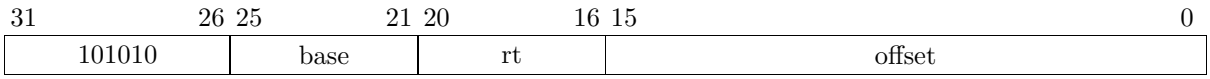
D-TLB No Entry Matched :
D-TLB Protection Error :

概要 :

アラインされていないアドレスに対してロードを行う．LWL 命令との組み合わせにより，アラインされていないアドレスから 1Word ロードすることができる．

SWL Store Word Left

32bit unaligned ストア MIPS I



SWL

ニーモニック:

SWL rt, offset(base)

機能：

MEM[GPR[base] + offset] ← GPR[rt]

例外：

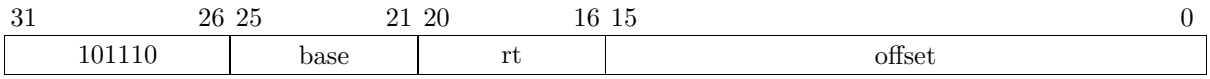
D-TLB No Entry Matched :
D-TLB Protection Error :

概要：

アラインされていないアドレスに対してストアを行う．SWR 命令との組み合わせにより，アラインされていないアドレスに 1Word ストアすることができる．

SWR Store Word Right

32bit unaligned ストア MIPS I



SWR

ニーモニック:

SWR rt, offset(base)

機能 :

MEM[GPR[base] + offset] ← GPR[rt]

例外 :

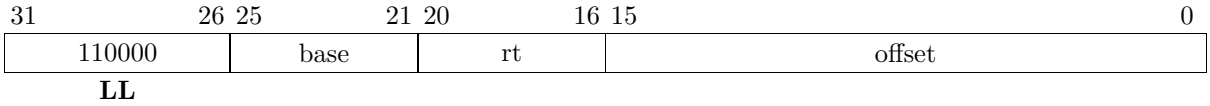
D-TLB No Entry Matched :
D-TLB Protection Error :

概要 :

アラインされていないアドレスに対してストアを行う．SWL 命令との組み合わせにより，アラインされていないアドレスに 1Word ストアすることができる．

LL Load Linked Word

atomic read-modify-write 用 32bit ロード MIPS II



ニーモニック:

LL rt, offset(base)

機能 :

GPR[rt] ← MEM[GPR[base] + offset]

例外 :

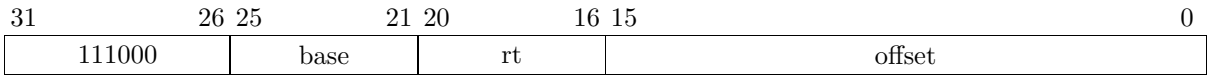
D-TLB No Entry Matched :
D-TLB Protection Error :
Data Address Miss Align (Load) :

概要 :

Atomic Read-Modify-Write のロードを行う.

SC — Store Conditional Word

atomic read-modify-write 用 32bit ストア — MIPS II



ニーモニック:

SC rt, offset(base)

機能：

if atomic_update then MEM[GPR[base] + offset] ← GPR[rt], GPR[rt] ← else GPR[rt] ← 0

例外：

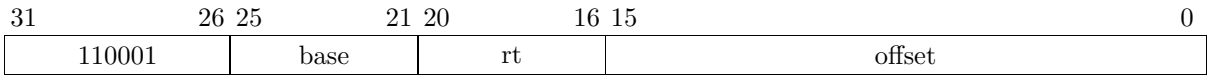
D-TLB No Entry Matched :
D-TLB Protection Error :
Data Address Miss Align (Load) :

概要：

Atomic Read-Modify-Write のストアを行う．Atomic Read-Modify-Write が成功すると 1 が返り，失敗すると 0 が返る．

LWC1 Load Word to Floating Point

浮動小数点レジスタ用 32bit ロード MIPS I



LWC1

ニーモニック:

LWC1 ft, offset(base)

機能 :

FPR[ft] ← MEM[GPR[base] + offset]

例外 :

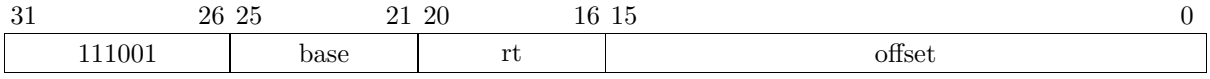
D-TLB No Entry Matched :
D-TLB Protection Error :
Data Address Miss Align (Load) :

概要 :

浮動小数点レジスタへ 1Word ロードする.

SWC1 Store Word from Floating Point

浮動小数点レジスタ用 32bit ストア MIPS I



SWC1

ニーモニック:

SWC1 ft, offset(base)

機能 :

MEM[GPR[base] + offset] ← FPR[ft]

例外 :

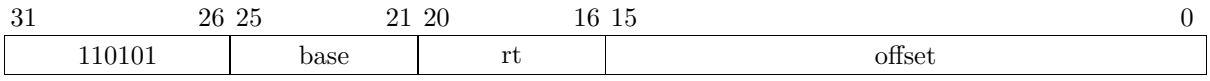
D-TLB No Entry Matched :
D-TLB Protection Error :
Data Address Miss Align (Store) :

概要 :

浮動小数点レジスタから 1Word ストアする.

LDC1 Load Doubleword to Floating Point

浮動小数点レジスタ用 64bit ロード MIPS I



LDC1

ニーモニック:

LDC1 ft, offset(base)

機能 :

FPR[ft] ← MEM[GPR[base] + offset]

例外 :

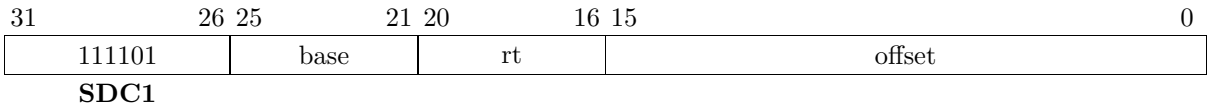
D-TLB No Entry Matched :
D-TLB Protection Error :
Data Address Miss Align (Load) :

概要 :

浮動小数点レジスタへ Double word ロードする.

SDC1 Store Doubleword from Floating Point

浮動小数点レジスタ用 64bit ストア MIPS I



SDC1

ニーモニック:

SDC1 ft, offset(base)

機能 :

MEM[GPR[base] + offset] ← FPR[ft]

例外 :

D-TLB No Entry Matched :
D-TLB Protection Error :
Data Address Miss Align (Store) :

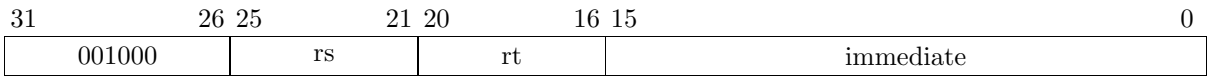
概要 :

浮動小数点レジスタから Double word ストアする.

3.1.2　演算命令

ADDI — Add Immediate Word

即値加算　　MIPS I



ADDI

ニーモニック:

ADDI rt, rs, immediate

機能 :

GPR[rt] ← GPR[rs] + sign_extend(immediate)

例外 :

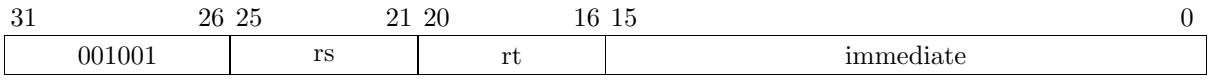
Overflow :

概要 :

レジスタと即値を加算する.

ADDIU — Add Immediate Unsigned Word

符号無し即値加算　　MIPS I



ADDIU

ニーモニック:

ADDIU rt, rs, immediate

機能 :

GPR[rt] ← GPR[rs] + sign_extend(immediate)

例外 :

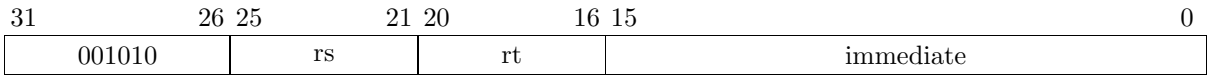
None

概要 :

レジスタと即値を加算する．オーバフローが発生しても例外を起さない.

SLTI — Set on Less Than Immediate

符号付き即値比較 MIPS I



SLTI

ニーモニック:

SLTI rt, rs, immediate

機能 :

GPR[rt] ← (GPR[rs] < sign_extend(immediate))

例外 :

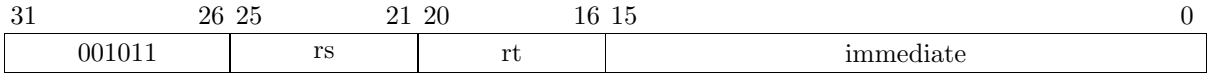
None

概要 :

レジスタと即値を比較する.

SLTIU — Set on Less Than Immediate Unsigned

符号無し即値比較 MIPS I



SLTIU

ニーモニック:

SLTIU rt, rs, immediate

機能 :

GPR[rt] ← (GPR[rs] < sign_extend(immediate))

例外 :

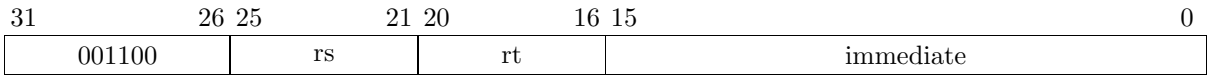
None

概要 :

レジスタと即値を符号無しで比較する.

ANDI — And Immediate

即値論理積 MIPS I



ANDI

ニーモニック:

ANDI rt, rs, immediate

機能 :

GPR[rt] ← GPR[rs] and zero_extend(immediate))

例外 :

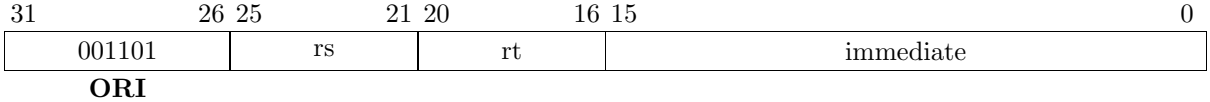
None

概要 :

レジスタと即値の論理積をとる.

ORI — Or Immediate

即値論理和 MIPS I



ORI

ニーモニック:

ORI rt, rs, immediate

機能 :

GPR[rt] ← GPR[rs] or zero_extend(immediate))

例外 :

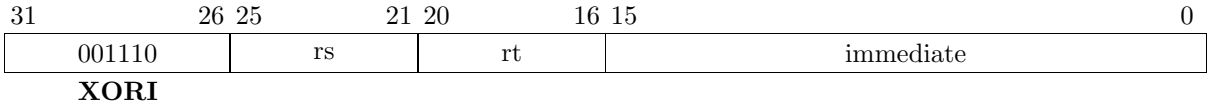
None

概要 :

レジスタと即値の論理和をとる.

XORI — Exclusive Or Immediate

即値排他的論理和 MIPS I



ニーモニック:

XORI rt, rs, immediate

機能 :

GPR[rt] ← GPR[rs] xor zero_extend(immediate))

例外 :

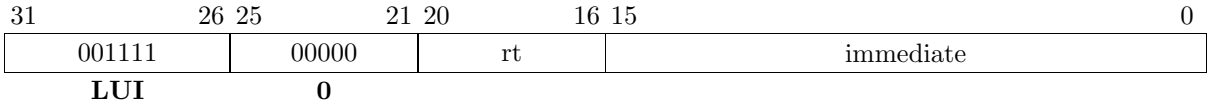
None

概要 :

レジスタと即値の排他的論理和をとる.

LUI — Load Upper Immediate

上位即値ロード MIPS I



ニーモニック:

LUI rt, immediate

機能 :

GPR[rt] ← immediate ¡¡ 16

例外 :

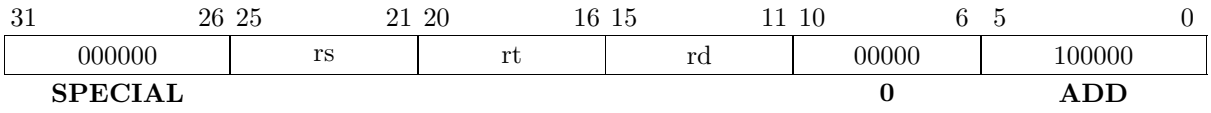
None

概要 :

即値をレジスタの上位 16bit にロードする.

ADD — Add Word

加算 MIPS I



ニーモニック:

ADD rd, rs, rt

機能 :

GPR[rd] ← GPR[rs] + GPR[rt]

例外 :

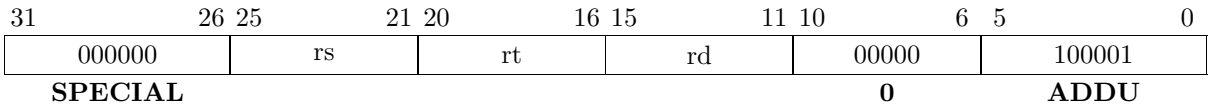
Overflow :

概要 :

レジスタの値を加算する.

ADDU — Add Unsigned Word

符号無し加算 MIPS I



ニーモニック:

ADDU rd, rs, rt

機能 :

GPR[rd] ← GPR[rs] + GPR[rt]

例外 :

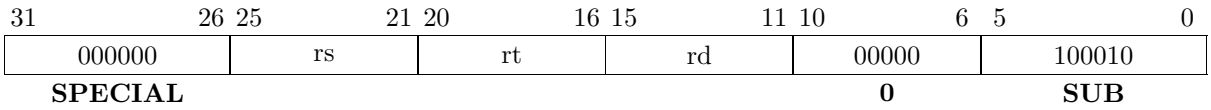
None

概要 :

レジスタの値を加算する．オーバフローを起しても例外を発生させない.

SUB — Subtract Word

減算 MIPS I



ニーモニック:

SUB rd, rs, rt

機能 :

GPR[rd] ← GPR[rs] − GPR[rt]

例外 :

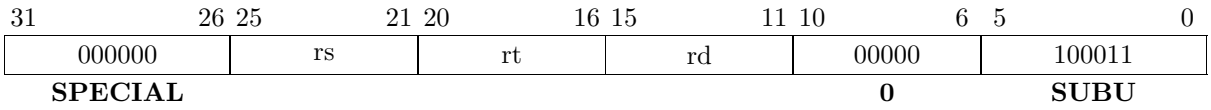
Overflow :

概要 :

レジスタの値を減算する.

SUBU — Subtract Unsigned Word

符号無し減算 MIPS I



ニーモニック:

SUBU rd, rs, rt

機能 :

GPR[rd] ← GPR[rs] − GPR[rt]

例外 :

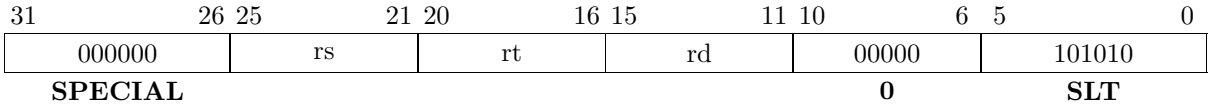
None

概要 :

レジスタの値を減算する．オーバフローを起しても例外を発生させない

SLT — Set on Less Than

符号付き比較 MIPS I



ニーモニック:

SLT rd, rs, rt

機能 :

GPR[rd] ← (GPR[rs] < GPR[rt])

例外 :

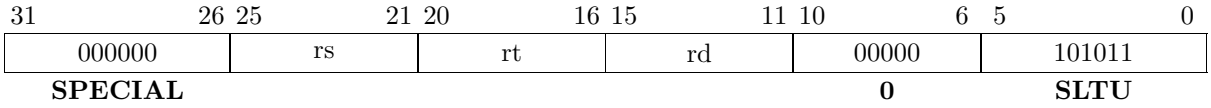
None

概要 :

レジスタの値を比較する.

SLTU — Set on Less Than Unsigned

符号無し比較 MIPS I



ニーモニック:

SLTU rd, rs, rt

機能 :

GPR[rd] ← (GPR[rs] < GPR[rt])

例外 :

None

概要 :

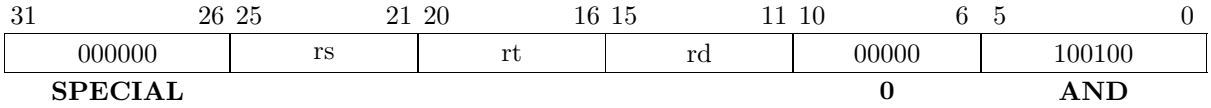
レジスタの値を符号無しで比較する.

AND

And

論理積

MIPS I



ニーモニック:

AND rd, rs, rt

機能 :

GPR[rd] ← GPR[rs] and GPR[rt]

例外 :

None

概要 :

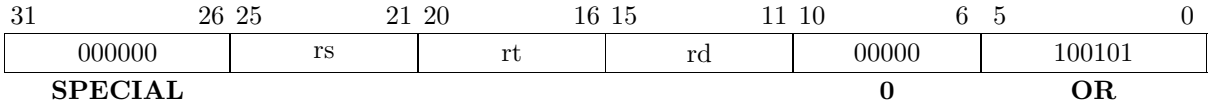
レジスタの値の論理積をとる.

OR

Or

論理和

MIPS I



ニーモニック:

OR rd, rs, rt

機能 :

GPR[rd] ← GPR[rs] or GPR[rt]

例外 :

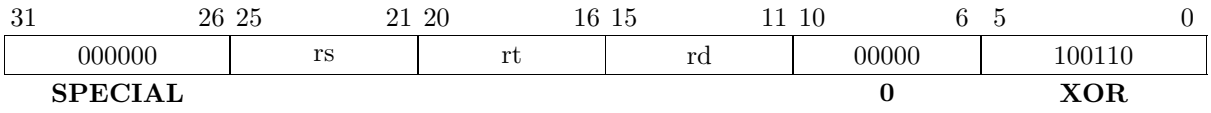
None

概要 :

レジスタの値の論理和をとる.

XOR　Exclusive Or

排他的論理和　MIPS I



ニーモニック:

XOR rd, rs, rt

機能：

GPR[rd] ← GPR[rs] xor GPR[rt]

例外：

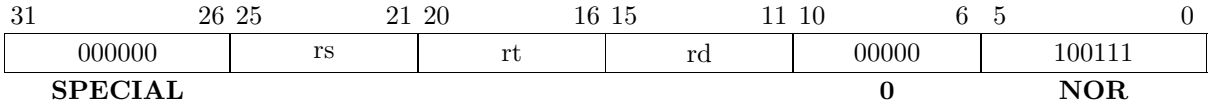
None

概要：

レジスタの値の排他的論理和をとる.

NOR　Not Or

否定論理和　MIPS I



ニーモニック:

NOR rd, rs, rt

機能：

GPR[rd] ← GPR[rs] nor GPR[rt]

例外：

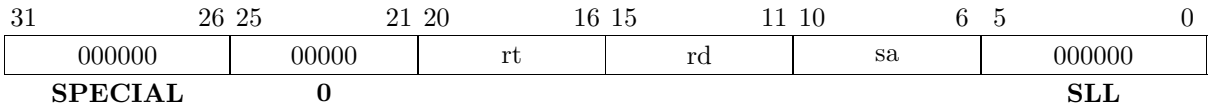
None

概要：

レジスタの値の否定論理和をとる.

SLL — Shift Word Left Logical

左論理シフト MIPS I



ニーモニック:

SLL rd, rt, sa

機能 :

GPR[rd] ← GPR[rt] << sa

例外 :

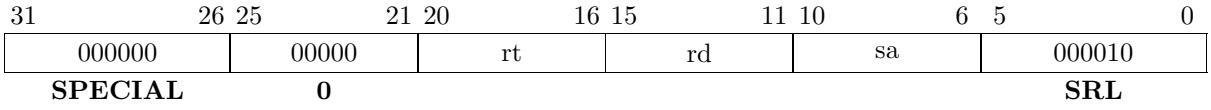
None

概要 :

レジスタの値を左論理シフトする.

SRL — Shift Word Right Logical

右論理シフト MIPS I



ニーモニック:

SRL rd, rt, sa

機能 :

GPR[rd] ← GPR[rt] >> sa

例外 :

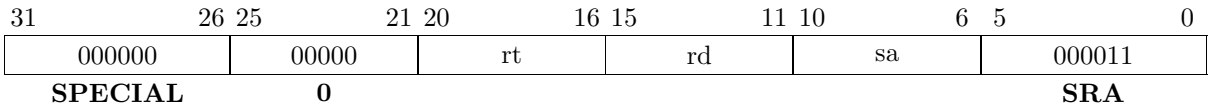
None

概要 :

レジスタの値を右論理シフトする.

SRA — Shift Word Right Arithmetic

右算術シフト　　MIPS I



ニーモニック:

SRA rd, rt, sa

機能：

GPR[rd] ← GPR[rt] >> sa

例外：

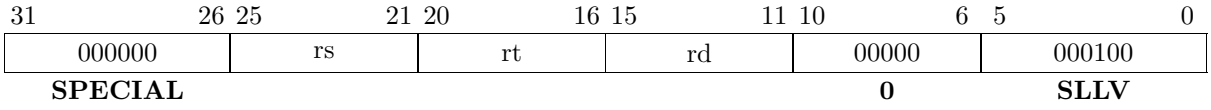
None

概要：

レジスタの値を右算術シフトする.

SLLV — Shift Word Left Logical Variable

左論理シフト　　MIPS I



ニーモニック:

SLLV rd, rt, rs

機能：

GPR[rd] ← GPR[rt] << GPR[rs]

例外：

None

概要：

レジスタの値を左論理シフトする.

SRLV — Shift Word Right Logical Variable

右論理シフト MIPS I

31–26	25–21	20–16	15–11	10–6	5–0
000000	rs	rt	rd	00000	000110
SPECIAL				**0**	**SRLV**

ニーモニック:

SRLV rd, rt, rs

機能 :

GPR[rd] ← GPR[rt] >> GPR[rs]

例外 :

None

概要 :

レジスタの値を右論理シフトする.

SRAV — Shift Word Right Arithmetic Variable

右算術シフト MIPS I

31–26	25–21	20–16	15–11	10–6	5–0
000000	rs	rt	rd	00000	000111
SPECIAL				**0**	**SRAV**

ニーモニック:

SRAV rd, rt, rs

機能 :

GPR[rd] ← GPR[rt] >> GPR[rs]

例外 :

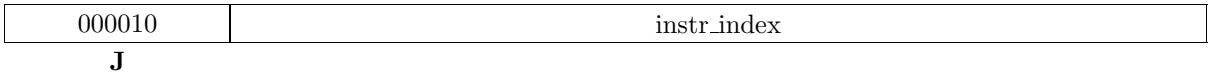
None

概要 :

レジスタの値を右算術シフトする.

3.1.3 Jump / 分岐命令

J	Jump
ジャンプ	MIPS I



J

ニーモニック:

J target

機能 :

pc ← { pc[31:28], instr_index, 00 }

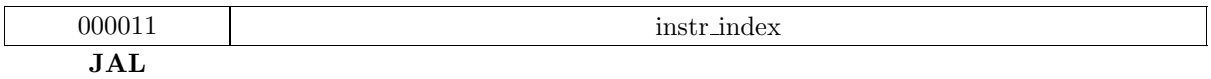
例外 :

None

概要 :

指定したアドレスに無条件ジャンプする.

JAL	Jump and Link
プロシージャコール	MIPS I



JAL

ニーモニック:

JAL target

機能 :

pc ← { pc[31:28], instr_index, 00 }
GPR[31] ← pc + 8

例外 :

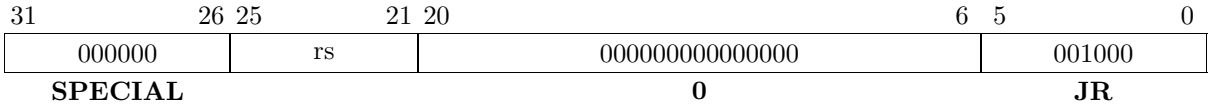
None

概要 :

指定したアドレスに無条件ジャンプする. リターンアドレスを R31 に保存する.

JR — Jump Register

レジスタ間接ジャンプ MIPS I



ニーモニック:

JR rs

機能 :

pc ← GPR[rs]

例外 :

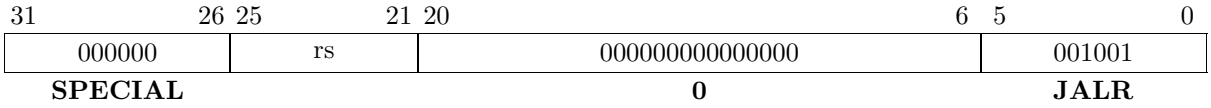
None

概要 :

レジスタで指定したアドレスに無条件ジャンプする.

JALR — Jump and Link Register

レジスタ間接プロシージャコール MIPS I



ニーモニック:

JALR rs

機能 :

pc ← GPR[rs]
GPR[31] ← pc + 8

例外 :

None

概要 :

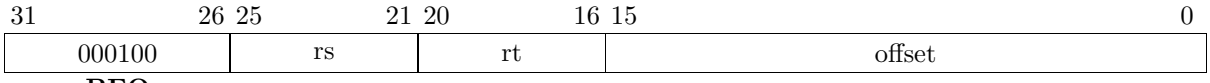
指定したアドレスに無条件ジャンプする. リターンアドレスを R31 に保存する.

BEQ

Branch on Equal

条件分岐

MIPS I



BEQ

ニーモニック:

BEQ rs, rt, offset

機能：

if GPR[rs] = GPR[rt] then pc-relative conditional branch

例外：

None

概要：

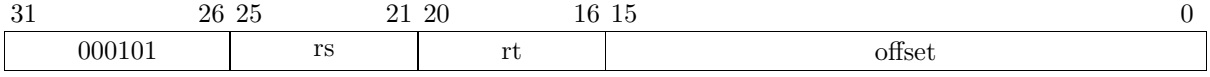
指定したアドレスに条件分岐する.

BNE

Branch on Not Equal

条件分岐

MIPS I



BNE

ニーモニック:

BNE rs, rt, offset

機能：

if GPR[rs] $\neq$ GPR[rt] then pc-relative conditional branch

例外：

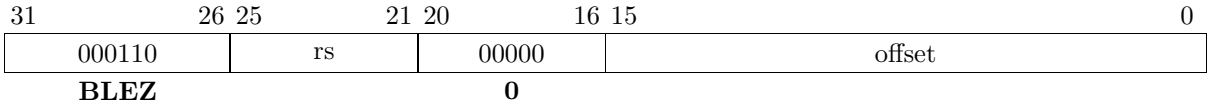
None

概要：

指定したアドレスに条件分岐する.

BLEZ — Branch on Less Than or Equal to Zero

条件分岐 MIPS I



ニーモニック:

BLEZ rs, offset

機能：

if GPR[rs] $\leq$ 0 then pc-relative conditional branch

例外：

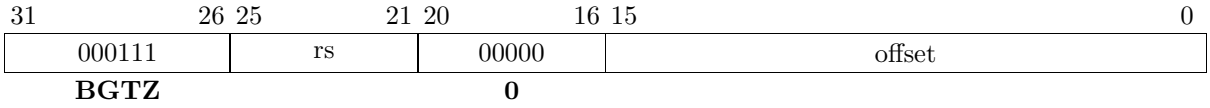
None

概要：

指定したアドレスに条件分岐する.

BGTZ — Branch on Greater Than Zero

条件分岐 MIPS I



ニーモニック:

BGTZ rs, offset

機能：

if GPR[rs] $>$ 0 then pc-relative conditional branch

例外：

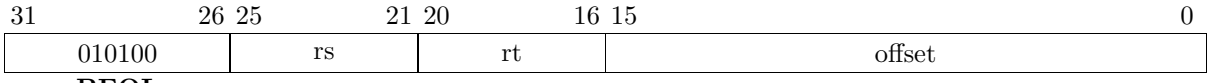
None

概要：

指定したアドレスに条件分岐する.

BEQL — Branch on Equal Likely

条件分岐 MIPS II



BEQL

ニーモニック:

BEQL rs, rt, offset

機能 :

if GPR[rs] = GPR[rt] then branch_likely

例外 :

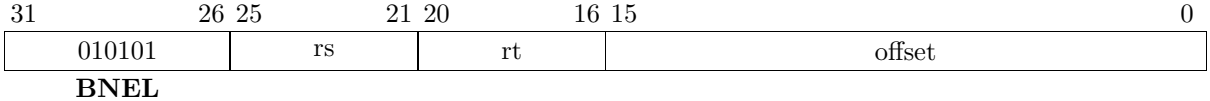
None

概要 :

指定したアドレスに条件分岐する.

BNEL — Branch on Not Equal Likely

条件分岐 MIPS II



BNEL

ニーモニック:

BNEL rs, rt, offset

機能 :

if GPR[rs] $\neq$ GPR[rt] then branch_likely

例外 :

None

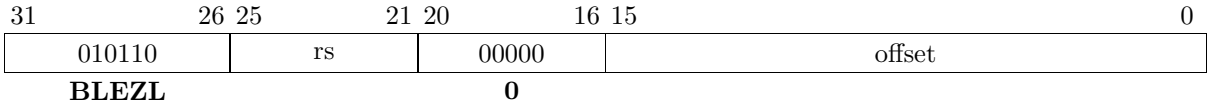
概要 :

指定したアドレスに条件分岐する.

BLEZL

Branch on Less Than or Equal to Zero Likely

条件分岐 MIPS II



ニーモニック:

BLEZL rs, offset

機能 :

if GPR[rs] $\leq$ 0 then branch_likely

例外 :

None

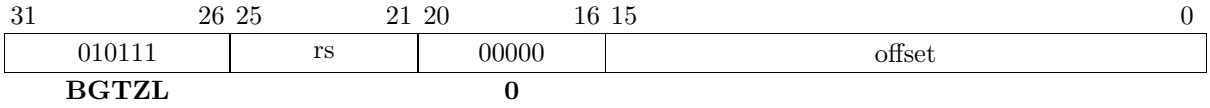
概要 :

指定したアドレスに条件分岐する.

BGTZL

Branch on Greater Than to Zero Likely

条件分岐 MIPS II



ニーモニック:

BGTZL rs, offset

機能 :

if GPR[rs] $>$ 0 then branch_likely

例外 :

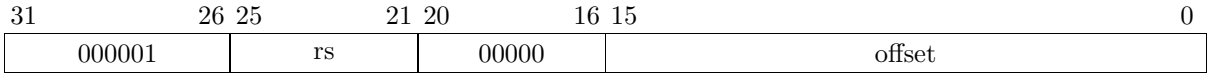
None

概要 :

指定したアドレスに条件分岐する.

BLTZ — Branch on Less Than Zero

条件分岐 MIPS I



ニーモニック:

BLTZ rs, offset

機能：

if GPR[rs] $<$ 0 then branch

例外：

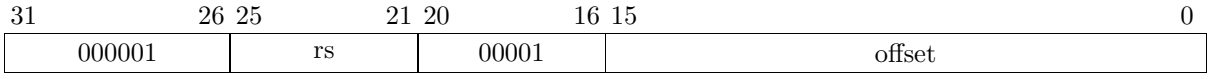
None

概要：

指定したアドレスに条件分岐する.

BGEZ — Branch on Greater Than or Equal to Zero

条件分岐 MIPS I



ニーモニック:

BGEZ rs, offset

機能：

if GPR[rs] $\geq$ 0 then branch

例外：

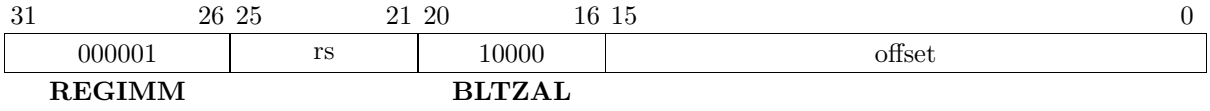
None

概要：

指定したアドレスに条件分岐する.

BLTZAL — Branch on Less Than Zero and Link

条件プロシージャコール MIPS I



ニーモニック:

BGEZ rs, offset

機能 :

if GPR[rs] $<$ 0 then branch
GPR[31] $\leftarrow$ pc + 8

例外 :

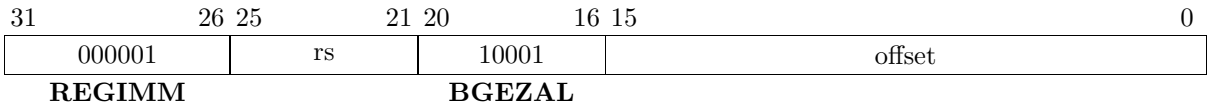
None

概要 :

指定したアドレスに条件分岐する．リターンアドレスを R31 にセーブする．

BGEZAL — Branch on Greater Than or Equal to Zero and Link

条件プロシージャコール MIPS I



ニーモニック:

BGEZAL rs, offset

機能 :

if GPR[rs] $\geq$ 0 then branch
GPR[31] $\leftarrow$ pc + 8

例外 :

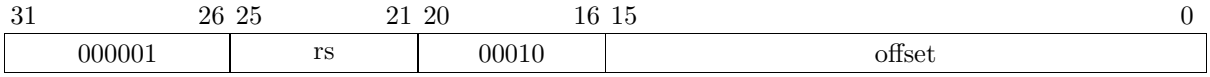
None

概要 :

指定したアドレスに条件分岐する．リターンアドレスを R31 にセーブする．

BLTZL Branch on Less Than Zero Likely

条件分岐 MIPS II



ニーモニック:

BLTZL rs, offset

機能 :

if GPR[rs] $<$ 0 then branch_likely

例外 :

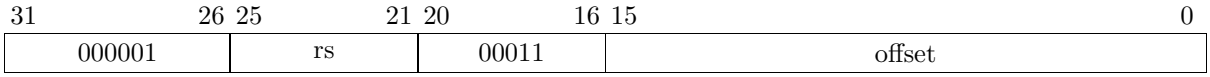
None

概要 :

指定したアドレスに条件分岐する.

BGEZL Branch on Greater Than or Equal to Zero Likely

条件分岐 MIPS II



ニーモニック:

BGEZL rs, offset

機能 :

if GPR[rs] $\geq$ 0 then branch_likely

例外 :

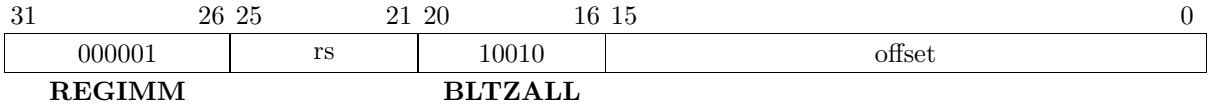
None

概要 :

指定したアドレスに条件分岐する.

BLTZALL Branch on Less Than Zero and Link Likely

条件プロシージャコール MIPS II



ニーモニック:

BLTZALL rs, offset

機能 :

if GPR[rs] $<$ 0 then branch_likely
GPR[31] $\leftarrow$ pc + 8

例外 :

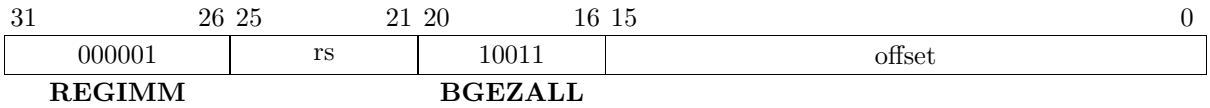
None

概要 :

指定したアドレスに条件分岐する．R31 にリターンアドレスをセーブする．

BGEZALL Branch on Greater Than or Equal to Zero and Link Likely

条件プロシージャコール MIPS II



ニーモニック:

BGEZALL rs, offset

機能 :

if GPR[rs] $\geq$ 0 then branch_likely
GPR[31] $\leftarrow$ pc + 8

例外 :

None

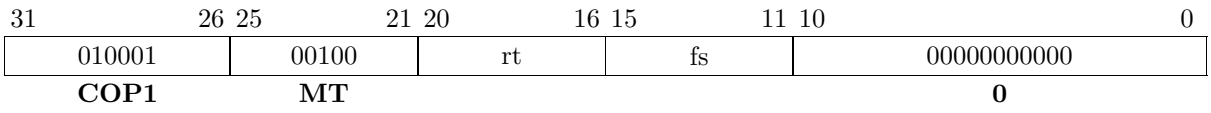
概要 :

指定したアドレスに条件分岐する．リターンアドレスを R31 にセーブする．

3.1.4　浮動小数点命令

MTC1 — Move Word to Floating Point

レジスタ間転送　　MIPS I



ニーモニック:

MTC1 rt, fs

機能：

FPR[fs] ← GPR[rt]

例外：

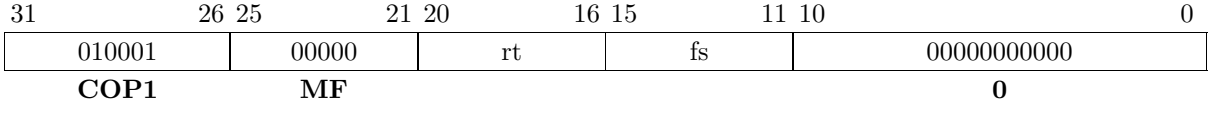
None

概要：

汎用レジスタの値を浮動小数点レジスタに入れる.

MFC1 — Move Word from Floating Point

レジスタ間転送　　MIPS I



ニーモニック:

MFC1 rt, fs

機能：

GPR[rt] ← FPR[fs]

例外：

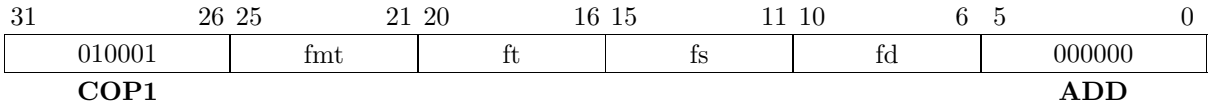
None

概要：

浮動小数点レジスタの値を汎用レジスタに入れる.

ADD.fmt Floating Point Add

浮動小数点加算 MIPS I



ニーモニック:

ADD.S fd, fs, ft (fmt = 10000)
ADD.D fd, fs, ft (fmt = 10001)

機能 :

FPR[fd] ← FPR[fs] + FPR[ft]

例外 :

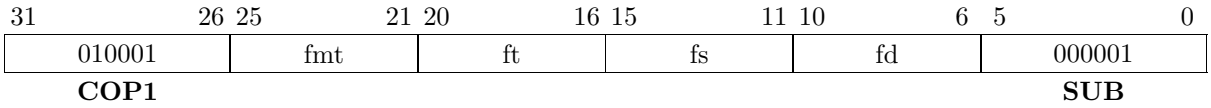
Floating Point Invalid Operation :
Floating Point Inexact :
Floating Point Overflow :
Floating Point Underflow :

概要 :

レジスタの値を加算する.

SUB.fmt Floating Point Subtract

浮動小数点減算 MIPS I



ニーモニック:

SUB.S fd, fs, ft (fmt = 10000)
SUB.D fd, fs, ft (fmt = 10001)

機能 :

FPR[fd] ← FPR[fs] − FPR[ft]

例外 :

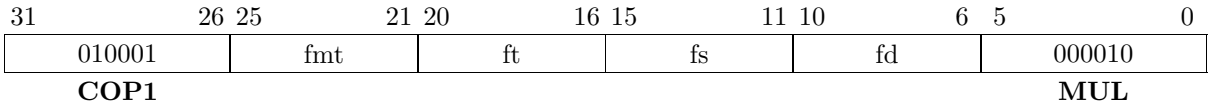
Floating Point Invalid Operation :
Floating Point Inexact :
Floating Point Overflow :
Floating Point Underflow :

概要 :

レジスタの値を減算する.

MUL.fmt Floating Point Multiply

浮動小数点乗算 MIPS I



ニーモニック:

MUL.S fd, fs, ft (fmt = 10000)
MUL.D fd, fs, ft (fmt = 10001)

機能 :

FPR[fd] ← FPR[fs] × FPR[ft]

例外 :

Floating Point Invalid Operation :
Floating Point Inexact :
Floating Point Overflow :
Floating Point Underflow :

概要 :

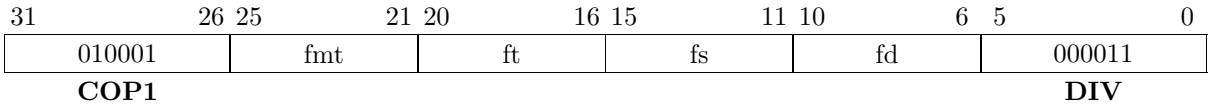
レジスタの値を乗算する.

DIV.fmt

Floating Point Divide

浮動小数点除算

MIPS I



ニーモニック:

DIV.S fd, fs, ft (fmt = 10000)

DIV.D fd, fs, ft (fmt = 10001)

機能 :

FPR[fd] ← FPR[fs] / FPR[ft]

例外 :

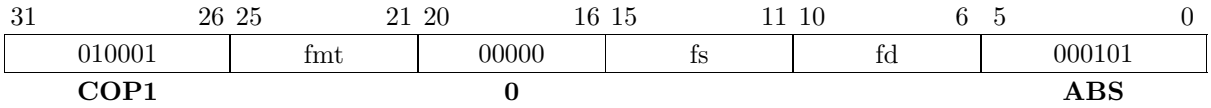
Floating Point Invalid Operation :
Floating Point Inexact :
Floating Point Overflow :
Floating Point Underflow :
Floating Point Divide By 0 :

概要 :

レジスタの値を除算する.

ABS.fmt Floating Point Absolute Value

浮動小数点絶対値 MIPS I



ニーモニック:

ABS.S fd, fs (fmt = 10000)
ABS.D fd, fs (fmt = 10001)

機能 :

FPR[fd] ← abs(FPR[fs])

例外 :

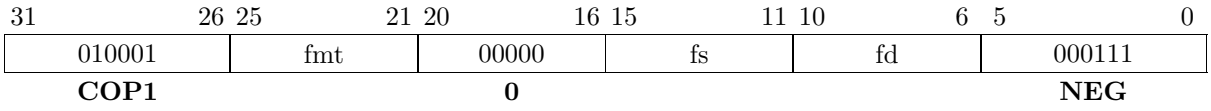
Floating Point Invalid Operation :

概要 :

レジスタの値の絶対値をとる.

NEG.fmt Floating Point Negate

浮動小数点符号反転 MIPS I



ニーモニック:

NEG.S fd, fs (fmt = 10000)
NEG.D fd, fs (fmt = 10001)

機能 :

FPR[fd] ← −(FPR[fs])

例外 :

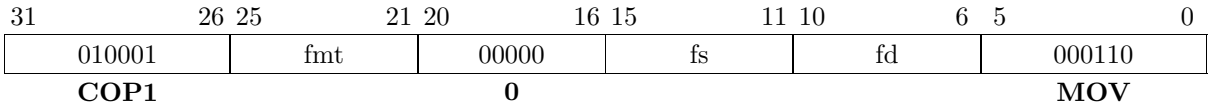
Floating Point Invalid Operation :

概要 :

レジスタの値を符号反転する.

MOV.fmt Floating Point Move

浮動小数点移動 MIPS I



ニーモニック:

MOV.S fd, fs (fmt = 10000)
MOV.D fd, fs (fmt = 10001)

機能 :

FPR[fd] ← FPR[fs]

例外 :

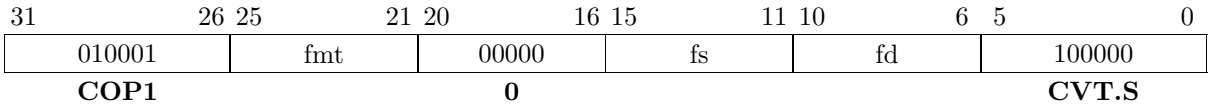
None

概要 :

レジスタの値を移動する.

CVT.S.fmt Floating Point Convert to Single Floating Point

浮動小数点フォーマット変換 MIPS I



31 26	25 21	20 16	15 11	10 6	5 0
010001	fmt	00000	fs	fd	100000
COP1		**0**			**CVT.S**

ニーモニック:

CVT.S.D fd, fs (fmt = 10001)
CVT.S.W fd, fs (fmt = 10100)

機能 :

FPR[fd] ← convert_and_round(FPR[fs])

例外 :

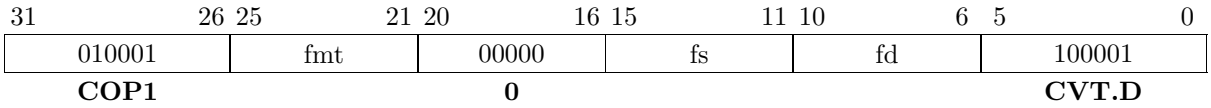
Floating Point Invalid Operation :
Floating Point Inexact :
Floating Point Overflow :
Floating Point Underflow :

概要 :

単精度にフォーマットを変換する.

CVT.D.fmt Floating Point Convert to Double Floating Point

浮動小数点フォーマット変換 MIPS I



ニーモニック:

CVT.D.S fd, fs (fmt = 10000)
CVT.D.W fd, fs (fmt = 10100)

機能 :

FPR[fd] ← convert_and_round(FPR[fs])

例外 :

Floating Point Invalid Operation :
Floating Point Inexact :

概要 :

倍精度にフォーマットを変換する.

ROUND.W.fmt Floating Point Round to Word Fixed Point

浮動小数点フォーマット変換 MIPS II

31 26	25 21	20 16	15 11	10 6	5 0
010001	fmt	00000	fs	fd	001100
COP1		**0**			**ROUND.W**

ニーモニック:

ROUND.W.S fd, fs (fmt = 10000)
ROUND.W.D fd, fs (fmt = 10001)

機能 :

FPR[fd] ← convert_and_round(FPR[fs])

例外 :

Floating Point Invalid Operation :
Floating Point Inexact :
Floating Point Overflow :

概要 :

整数にフォーマットを変換する.

TRUNC.W.fmt Floating Point Truncate to Word Fixed Point

浮動小数点フォーマット変換 MIPS II

31 26	25 21	20 16	15 11	10 6	5 0
010001	fmt	00000	fs	fd	001101
COP1		**0**			**TRUNC.W**

ニーモニック:

TRUNC.W.S fd, fs (fmt = 10000)
TRUNC.W.D fd, fs (fmt = 10001)

機能 :

FPR[fd] ← convert_and_round(FPR[fs])

例外 :

Floating Point Invalid Operation :
Floating Point Inexact :
Floating Point Overflow :

概要 :

整数にフォーマットを変換する.

CEIL.W.fmt Floating Point Ceiling to Word Fixed Point

浮動小数点フォーマット変換 MIPS II

31 26	25 21	20 16	15 11	10 6	5 0
010001	fmt	00000	fs	fd	001110
COP1		**0**			**CEIL.W**

ニーモニック:

CEIL.W.S fd, fs (fmt = 10000)
CEIL.W.D fd, fs (fmt = 10001)

機能 :

FPR[fd] ← convert_and_round(FPR[fs])

例外 :

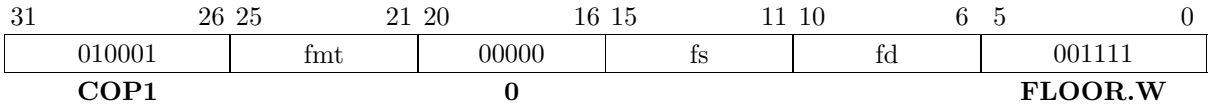
Floating Point Invalid Operation :
Floating Point Inexact :
Floating Point Overflow :

概要 :

整数にフォーマットを変換する.

FLOOR.W.fmt Floating Point Floor Convert to Word Fixed Point

浮動小数点フォーマット変換 MIPS II



ニーモニック:

FLOOR.W.S fd, fs (fmt = 10000)
FLOOR.W.D fd, fs (fmt = 10001)

機能 :

FPR[fd] ← convert_and_round(FPR[fs])

例外 :

Floating Point Invalid Operation :
Floating Point Inexact :
Floating Point Overflow :

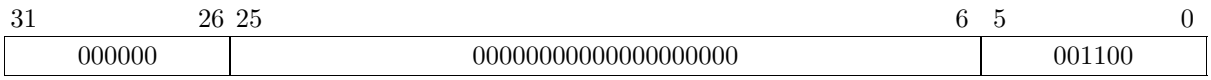
概要 :

整数にフォーマットを変換する.

3.1.5 その他の命令

SYSCALL — System Call

システムコール MIPS I



ニーモニック:

SYSCALL

機能 :

exception(system_call)

例外 :

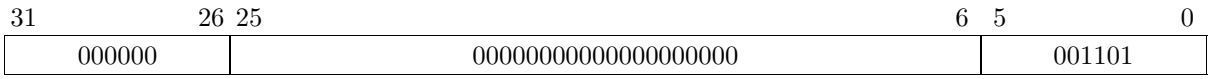
System Call :

概要 :

システムコール例外を発生する.

BREAK — Breakpoint

ブレークポイント MIPS I



ニーモニック:

BREAK

機能 :

exception(breakpoint)

例外 :

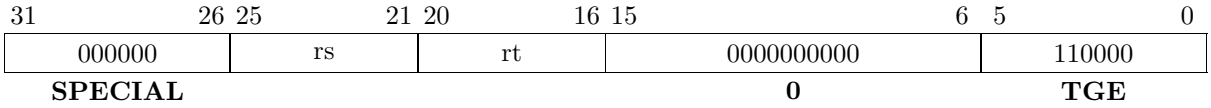
Break Point :

概要 :

ブレークポイント例外を発生する.

TGE — Trap if Greater or Equal

条件トラップ MIPS II



ニーモニック:

TGE rs, rt

機能 :

if GPR[rs] ≥ GPR[rt] then exception(trap)

例外 :

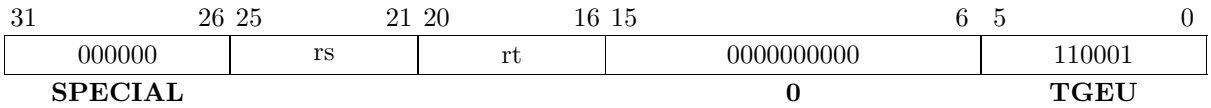
Trap :

概要 :

条件によりトラップを発生する.

TGEU — Trap if Greater or Equal Unsigned

条件トラップ MIPS II



ニーモニック:

TGEU rs, rt

機能 :

if GPR[rs] ≥ GPR[rt] then exception(trap)

例外 :

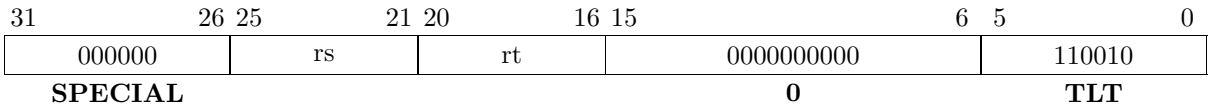
Trap :

概要 :

条件によりトラップを発生する.

TLT — Trap if Less Than

条件トラップ MIPS II



ニーモニック:

TLT rs, rt

機能 :

if GPR[rs] < GPR[rt] then exception(trap)

例外 :

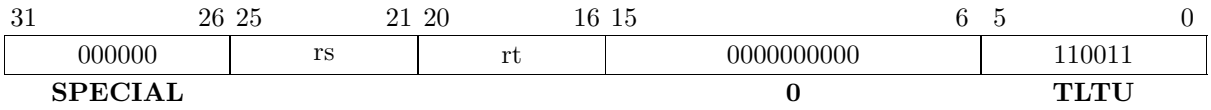
Trap :

概要 :

条件によりトラップを発生する.

TLTU — Trap if Less Than Unsigned

条件トラップ MIPS II



ニーモニック:

TLTU rs, rt

機能 :

if GPR[rs] < GPR[rt] then exception(trap)

例外 :

Trap :

概要 :

条件によりトラップを発生する.

TEQ — Trap if Equal

条件トラップ　MIPS II

31–26	25–21	20–16	15–6	5–0
000000	rs	rt	0000000000	110100
SPECIAL			**0**	**TEQ**

ニーモニック:

TEQ rs, rt

機能 :

if GPR[rs] = GPR[rt] then exception(trap)

例外 :

Trap :

概要 :

条件によりトラップを発生する.

TNE — Trap if Not Equal

条件トラップ　MIPS II

31–26	25–21	20–16	15–6	5–0
000000	rs	rt	0000000000	110110
SPECIAL			**0**	**TNE**

ニーモニック:

TEQ rs, rt

機能 :

if GPR[rs] $\neq$ GPR[rt] then exception(trap)

例外 :

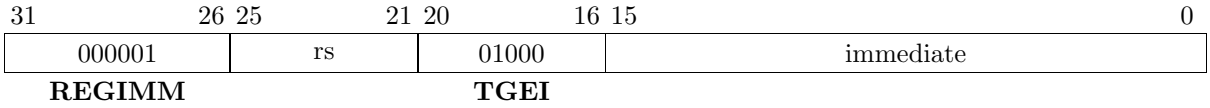
Trap :

概要 :

条件によりトラップを発生する.

TGEI — Trap if Greater or Equal Immediate

条件トラップ MIPS II



ニーモニック:

TGEI rs, immediate

機能 :

if GPR[rs] ≥ sign_extend(immediate) then exception(trap)

例外 :

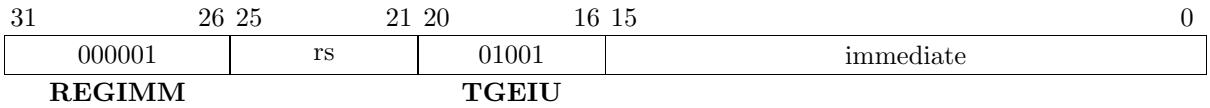
Trap :

概要 :

条件によりトラップを発生する.

TGEIU — Trap if Greater or Equal Immediate Unsigned

条件トラップ MIPS II



ニーモニック:

TGEIU rs, immediate

機能 :

if GPR[rs] ≥ sign_extend(immediate) then exception(trap)

例外 :

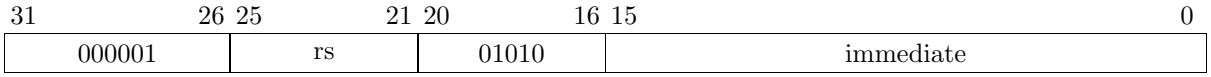
Trap :

概要 :

条件によりトラップを発生する.

TLTI — Trap if Less Than Immediate

条件トラップ　MIPS II



ニーモニック:

TLTI rs, immediate

機能 :

if GPR[rs] < sign_extend(immediate) then exception(trap)

例外 :

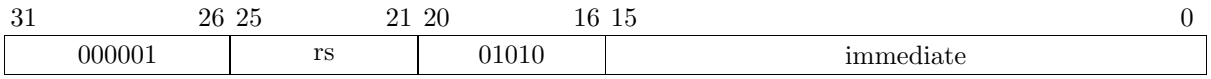
Trap :

概要 :

条件によりトラップを発生する.

TLTIU — Trap if Less Than Immediate Unsigned

条件トラップ　MIPS II



ニーモニック:

TLTIU rs, immediate

機能 :

if GPR[rs] < sign_extend(immediate) then exception(trap)

例外 :

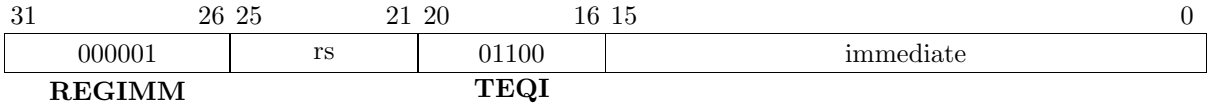
Trap :

概要 :

条件によりトラップを発生する.

TEQI — Trap if Equal Immediate

条件トラップ MIPS II



ニーモニック:

TEQI rs, immediate

機能 :

if GPR[rs] = sign_extend(immediate) then exception(trap)

例外 :

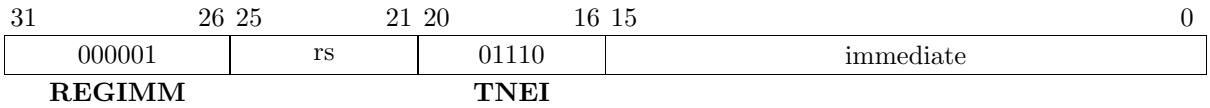
Trap :

概要 :

条件によりトラップを発生する.

TNEI — Trap if Not Equal Immediate

条件トラップ MIPS II



ニーモニック:

TNEI rs, immediate

機能 :

if GPR[rs] ≠ sign_extend(immediate) then exception(trap)

例外 :

Trap :

概要 :

条件によりトラップを発生する.

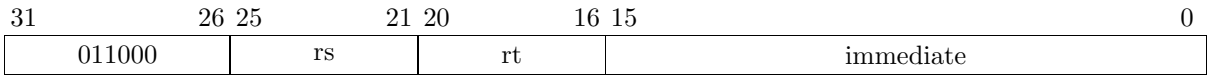
3.2　MIPS 命令と動作の異なる命令

以下に *Responsive Multithreaded Processor* の中で MIPS 命令と動作の異なる命令を示す.

3.2.1　演算命令

DADDI Doubleword Add Immediate

64bit 即値加算 MIPS III 動作改



DADDI

ニーモニック:

DADDI rt, rs, immediate

機能 :

FPR[rt] ← FPR[rs] + sign_extension(immediate)

例外 :

Overflow :

概要 :

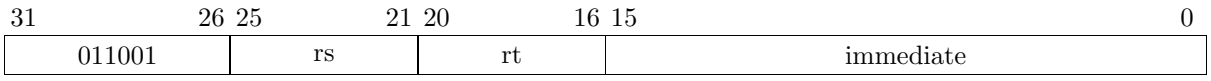
Responsive Multithreaded Processor では浮動小数点レジスタを用いて演算を行う.

DADDIU

Doubleword Add Immediate Unsigned

64bit 即値加算

MIPS III 動作改



DADDIU

ニーモニック:

DADDIU rt, rs, immediate

機能 :

FPR[rt] ← FPR[rs] + sign_extension(immediate)

例外 :

None

概要 :

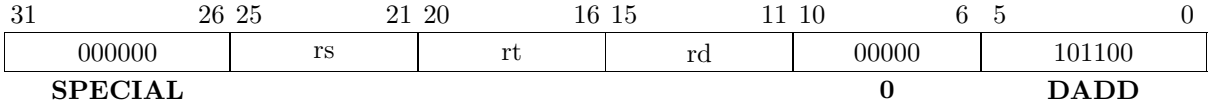
Responsive Multithreaded Processor では浮動小数点レジスタを用いて演算を行う.

DADD

Doubleword Add

64bit 加算

MIPS III 動作改



SPECIAL **0** **DADD**

ニーモニック:

DADD rd, rs, rt

機能 :

FPR[rd] ← FPR[rs] + FPR[rt]

例外 :

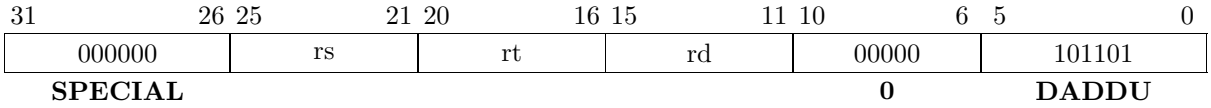
Overflow :

概要 :

Responsive Multithreaded Processor では浮動小数点レジスタを用いて演算を行う.

DADDU — Doubleword Add Unsigned

64bit 加算 MIPS III 動作改



31–26	25–21	20–16	15–11	10–6	5–0
000000	rs	rt	rd	00000	101101
SPECIAL				**0**	**DADDU**

ニーモニック:

DADDU rd, rs, rt

機能 :

FPR[rd] ← FPR[rs] + FPR[rt]

例外 :

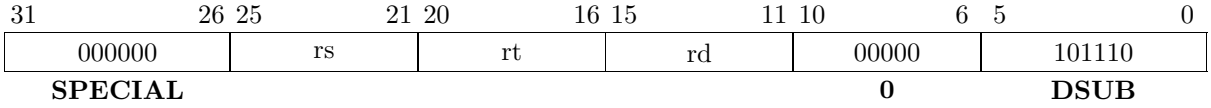
None

概要 :

Responsive Multithreaded Processor では浮動小数点レジスタを用いて演算を行う.

DSUB — Doubleword Subtract

64bit 減算 MIPS III 動作改



31–26	25–21	20–16	15–11	10–6	5–0
000000	rs	rt	rd	00000	101110
SPECIAL				**0**	**DSUB**

ニーモニック:

DSUB rd, rs, rt

機能 :

FPR[rd] ← FPR[rs] − FPR[rt]

例外 :

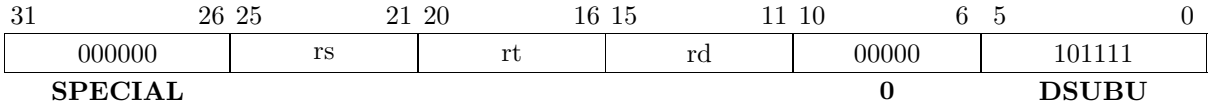
Overflow :

概要 :

Responsive Multithreaded Processor では浮動小数点レジスタを用いて演算を行う.

DSUBU — Doubleword Subtract Unsigned

64bit 減算 MIPS III 動作改



ニーモニック:

DSUBU rd, rs, rt

機能 :

FPR[rd] ← FPR[rs] − FPR[rt]

例外 :

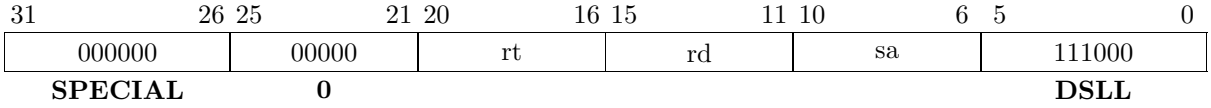
None

概要 :

Responsive Multithreaded Processor では浮動小数点レジスタを用いて演算を行う.

DSLL — Doubleword Shift Left Logical

64bit 左論理シフト MIPS III 動作改



ニーモニック:

DSLL rd, rt, sa

機能 :

FPR[rd] ← FPR[rt] ≪ sa

例外 :

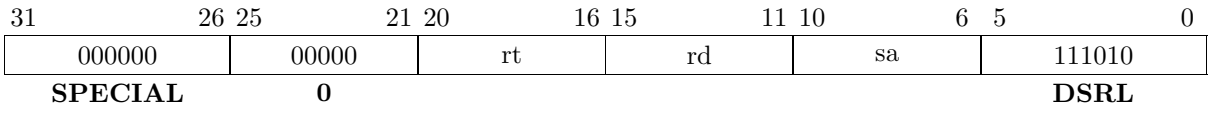
None

概要 :

Responsive Multithreaded Processor では浮動小数点レジスタを用いて演算を行う.

DSRL — Doubleword Shift Right Logical

64bit 右論理シフト　　MIPS III 動作改



ニーモニック:

DSRL rd, rt, sa

機能 :

FPR[rd] ← FPR[rt] ≫ sa

例外 :

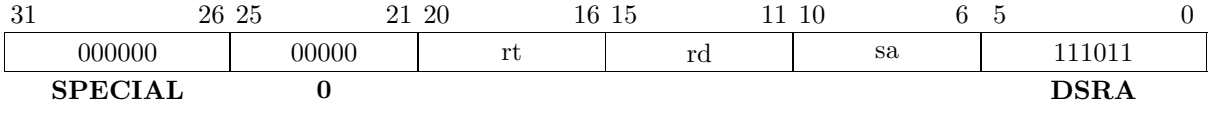
None

概要 :

Responsive Multithreaded Processor では浮動小数点レジスタを用いて演算を行う.

DSRA — Doubleword Shift Right Arithmetic

64bit 右算術シフト　　MIPS III 動作改



ニーモニック:

DSRA rd, rt, sa

機能 :

FPR[rd] ← FPR[rt] ≫ sa

例外 :

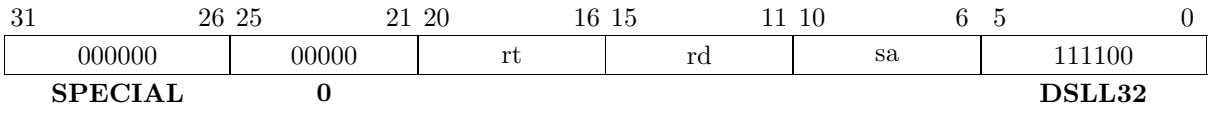
None

概要 :

Responsive Multithreaded Processor では浮動小数点レジスタを用いて演算を行う.

DSLL32 Doubleword Shift Left Logical plus 32

64bit 左論理シフト MIPS III 動作改



ニーモニック:

DSLL32 rd, rt, sa

機能 :

FPR[rd] ← FPR[rt] ≪ (sa + 32)

例外 :

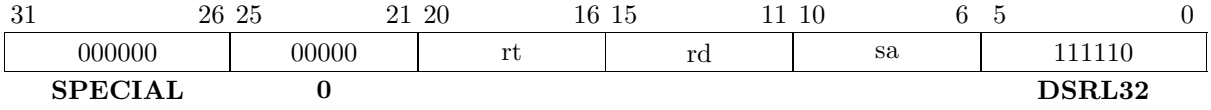
None

概要 :

Responsive Multithreaded Processor では浮動小数点レジスタを用いて演算を行う.

DSRL32 Doubleword Shift Right Logical plus 32

64bit 右論理シフト MIPS III 動作改



ニーモニック:

DSRL32 rd, rt, sa

機能 :

FPR[rd] ← FPR[rt] ≫ (sa + 32)

例外 :

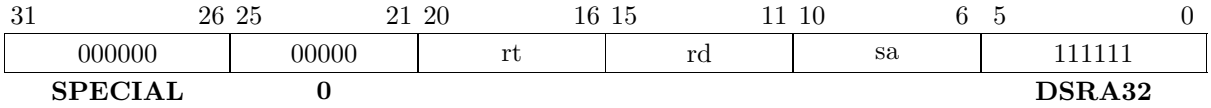
None

概要 :

Responsive Multithreaded Processor では浮動小数点レジスタを用いて演算を行う.

DSRA32 Doubleword Shift Right Arithmetic plus 32

64bit 右算術シフト　　MIPS III 動作改



ニーモニック:

DSRA32 rd, rt, sa

機能：

FPR[rd] ← FPR[rt] ≫ (sa + 32)

例外：

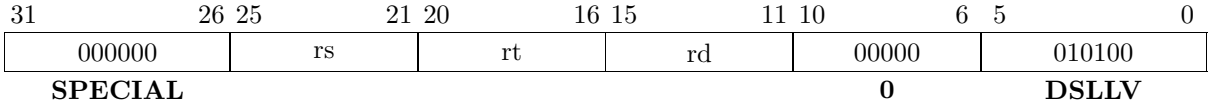
None

概要：

Responsive Multithreaded Processor では浮動小数点レジスタを用いて演算を行う.

DSLLV Doubleword Shift Left Logical Variable

64bit 左論理シフト　　MIPS III 動作改



ニーモニック:

DSLLV rd, rt, rs

機能：

FPR[rd] ← FPR[rt] ≪ FPR[rs]

例外：

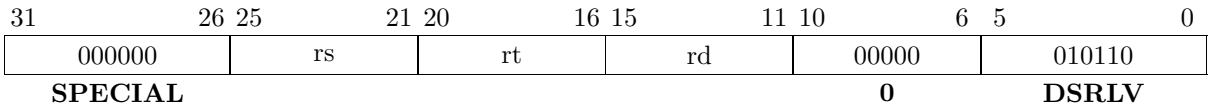
None

概要：

Responsive Multithreaded Processor では浮動小数点レジスタを用いて演算を行う.

DSRLV — Doubleword Shift Right Logical Variable

64bit 右論理シフト — MIPS III 動作改



ニーモニック:

DSRLV rd, rt, rs

機能 :

FPR[rd] ← FPR[rt] ≫ FPR[rs]

例外 :

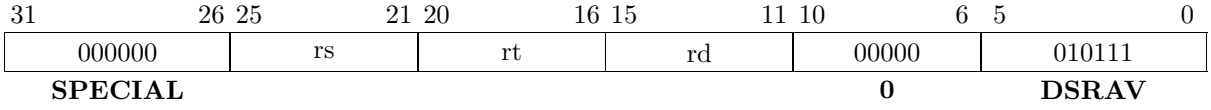
None

概要 :

Responsive Multithreaded Processor では浮動小数点レジスタを用いて演算を行う.

DSRAV — Doubleword Shift Right Arithmetic Variable

64bit 右算術シフト — MIPS III 動作改



ニーモニック:

DSRAV rd, rt, rs

機能 :

FPR[rd] ← FPR[rt] ≫ FPR[rs]

例外 :

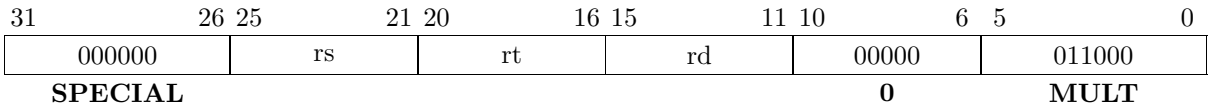
None

概要 :

Responsive Multithreaded Processor では浮動小数点レジスタを用いて演算を行う.

MULT — Multiply Word

符号付き乗算 MIPS I 動作改



ニーモニック:

MULT rd, rs, rt

機能 :

GPR[rd] ← GPR[rs] × GPR[rt]

例外 :

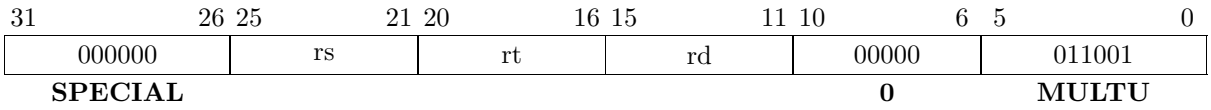
None

概要 :

Responsive Multithreaded Processor では 3 オペランド命令で演算結果の下位 32bit をデスティネーションレジスタに格納する.

MULTU — Multiply Word Unsigned

符号無し乗算 MIPS I 動作改



ニーモニック:

MULTU rd, rs, rt

機能 :

GPR[rd] ← GPR[rs] × GPR[rt]

例外 :

None

概要 :

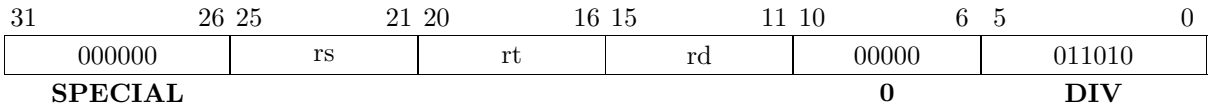
Responsive Multithreaded Processor では 3 オペランド命令で演算結果の下位 32bit をデスティネーションレジスタに格納する.

DIV

Divide Word

符号付き除算

MIPS I 動作改



ニーモニック:

DIV rd, rs, rt

機能 :

GPR[rd] ← GPR[rs] ÷ GPR[rt]

例外 :

Divide by Zero :

概要 :

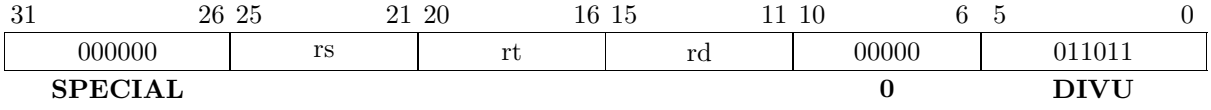
Responsive Multithreaded Processor では 3 オペランド命令で，商をデスティネーションレジスタに格納する．

DIVU

Divide Word Unsigned

符号無し除算

MIPS I 動作改



ニーモニック:

DIVU rd, rs, rt

機能 :

GPR[rd] ← GPR[rs] ÷ GPR[rt]

例外 :

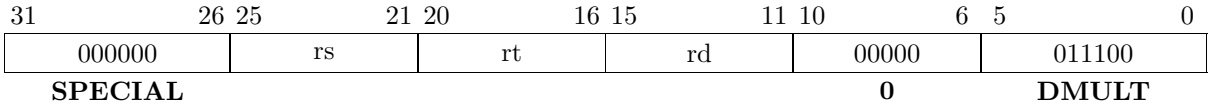
Divide by Zero :

概要 :

商をデスティネーションレジスタに格納する．*Responsive Multithreaded Processor* では 3 オペランド命令になる．

DMULT — Doubleword Multiply

符号付き 64bit 乗算　　MIPS III 動作改



ニーモニック:

DMULT rd, rs, rt

機能 :

FPR[rd] ← FPR[rs] × FPR[rt]

例外 :

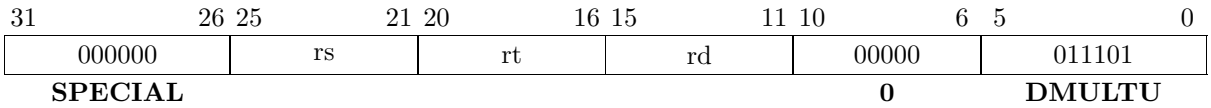
None

概要 :

Responsive Multithreaded Processor では 3 オペランド命令で，浮動小数点レジスタを用いて演算を行う．演算結果の下位 64bit をデスティネーションレジスタに格納する．

DMULTU — Doubleword Multiply Unsigned

符号無し 64bit 乗算　　MIPS III 動作改



ニーモニック:

DMULTU rd, rs, rt

機能 :

FPR[rd] ← FPR[rs] × FPR[rt]

例外 :

None

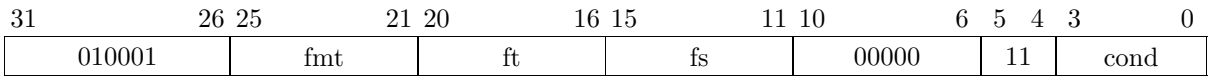
概要 :

Responsive Multithreaded Processor では 3 オペランド命令で，浮動小数点レジスタを用いて演算を行う．演算結果の下位 64bit をデスティネーションレジスタに格納する．

3.2.2 浮動小数点命令

C.cond.fmt Floating-Point Compare

浮動小数点比較 MIPS I 動作改



31 26	25 21	20 16	15 11	10 6	5 4	3 0
010001	fmt	ft	fs	00000	11	cond
COP1				**0**	**FC**	

ニーモニック:

C.cond.S fs, ft (fmt = 10000)
C.cond.D fs, ft (fmt = 10001)

機能 :

FPR[7] ← FPR[fs] conpare_cond FPR[ft]

例外 :

Floating Point Invalid :

概要 :

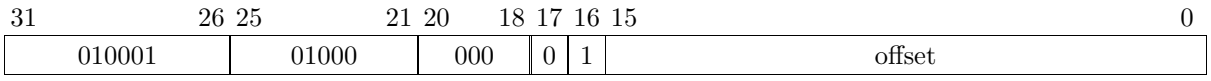
Responsive Multithreaded Processor ではアウトオブオーダーで命令が実行されるため，ステータスレジスタではなく，浮動小数点レジスタに結果が格納される．

BC1T

Branch on FP True

浮動小数点分岐

MIPS I 動作改



ニーモニック:

BC1T offset

機能 :

if (FPR[7] == 1) then branch

例外 :

None

概要 :

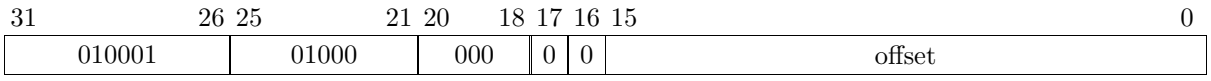
Responsive Multithreaded Processor ではステータスレジスタではなく，浮動小数点レジスタの内容により分岐を判断する．

BC1F

Branch on FP False

浮動小数点分岐

MIPS I 動作改



ニーモニック:

BC1F offset

機能 :

if (FPR[7] == 0) then branch

例外 :

None

概要 :

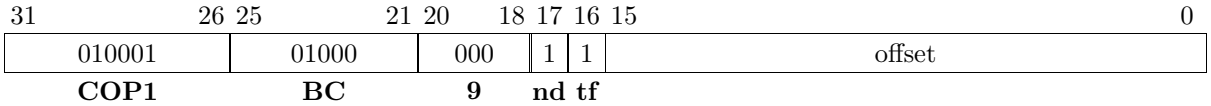
Responsive Multithreaded Processor ではステータスレジスタではなく，浮動小数点レジスタの内容により分岐を判断する．

BC1TL

Branch on FP True Likely

浮動小数点分岐

MIPS II 動作改



ニーモニック:

BC1TL offset

機能 :

if (FPR[7] == 1) then branch_likely

例外 :

None

概要 :

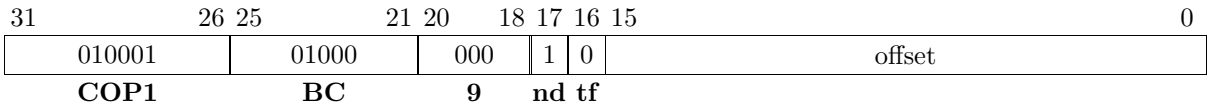
Responsive Multithreaded Processor ではステータスレジスタではなく，浮動小数点レジスタの内容により分岐を判断する.

BC1FL

Branch on FP False Likely

浮動小数点分岐

MIPS II 動作



ニーモニック:

BC1FL offset

機能 :

if (FPR[7] == 0) then branch_likely

例外 :

None

概要 :

Responsive Multithreaded Processor ではステータスレジスタではなく，浮動小数点レジスタの内容により分岐を判断する.

3.2.3　その他の命令

SYNC Synchronize Operation

命令実行順序制御 MIPS II 動作改

31 26	25 6	5 0
000000	00000000000000	001111
SPECIAL	**0**	**SYNC**

ニーモニック:

SYNC

機能：

synchronize_operation_order()

例外：

None

概要：

Responsive Multithreaded Processor ではアウトオブオーダーで命令が実行されるが，この命令の前後での実行順序が保証される．つまり sync 命令より後の命令は sync 命令より前の命令より先に実行されることはない．また，sync 命令は投機実行されないため，投機実行の制御を行うことができる．

3.2.4　サポートしていない MIPS II 命令

Responsive Multithreaded Processor は基本的に MIPS II 命令セット互換であるが，いくつかの命令をサポートしていない．以下に MIPS II 命令で *Responsive Multithreaded Processor* がサポートしていない命令を示す．

ニーモニック	概要	
LWC2	Load Word to Coprocessor-2	MIPS I
LWC3	Load Word to Coprocessor-3	MIPS I
SWC2	Store Word to Coprocessor-2	MIPS I
SWC3	Store Word to Coprocessor-3	MIPS I
LDC2	Load Doubleword to Coprocessor-2	MIPS II
LDC3	Load Doubleword to Coprocessor-3	MIPS II
SDC2	Store Doubleword to Coprocessor-2	MIPS II
SDC3	Store Doubleword to Coprocessor-3	MIPS II
MFHI	Move From HI	MIPS I
MTHI	Move To HI	MIPS I
MFLO	Move From LO	MIPI I
MTLO	Move To LO	MIPS I
CTC1	Move Control Word To Floating-Point	MIPS I
CFC1	Move Control Word From Floating-Point	MIPS I
SQRT.fmt	Floating-Point Square Root	MIPS II

3.3 Responsive Multithreaded Processor 固有の命令

以下に *Responsive Multithreaded Processor* 固有の命令を示す.

3.3.1 Load / Store 命令

IOLB — Load Byte of I/O

I/O 用ロード命令 RESPII

31 26	25 21	20 16	15 6	5 0
010000	rs	rt	0000000000	110000
COP0			**0**	**IOLB**

ニーモニック:

IOLB rt, rs

機能：

GPR[rt] ← MEM[GPR[rs]]

例外：

D-TLB No Entry Matched :
D-TLB Protection Error :
Data Address Miss Align (Load) :

概要：

通常の Load 命令は投機実行されるが，この命令は投機実行されないため，I/O のように 1 度ロードすると状態が変わるものに用いる．

IOLH

Load Half Word of I/O

I/O 用ロード命令 RESPII

31 26	25 21	20 16	15 6	5 0
010000	rs	rt	0000000000	110001
COP0			**0**	**IOLH**

ニーモニック:

IOLH rt, rs

機能 :

GPR[rt] ← MEM[GPR[rs]]

例外 :

D-TLB No Entry Matched :
D-TLB Protection Error :
Data Address Miss Align (Load) :

概要 :

通常の Load 命令は投機実行されるが，この命令は投機実行されないため，I/O のように 1 度ロードすると状態が変わるものに用いる．

IOLW

Load Word of I/O

I/O 用ロード命令 RESPII

31 26	25 21	20 16	15 6	5 0
010000	rs	rt	0000000000	110010
COP0			**0**	**IOLW**

ニーモニック:

IOLW rt, rs

機能 :

GPR[rt] ← MEM[GPR[rs]]

例外 :

D-TLB No Entry Matched :
D-TLB Protection Error :
Data Address Miss Align (Load) :

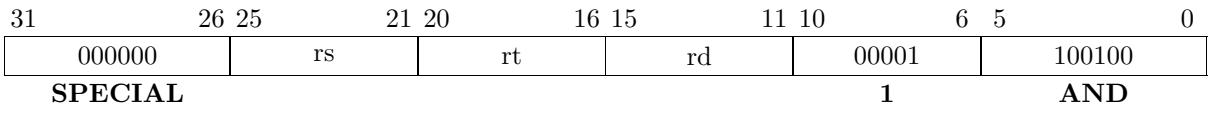
概要 :

通常の Load 命令は投機実行されるが，この命令は投機実行されないため，I/O のように 1 度ロードすると状態が変わるものに用いる．

3.3.2 演算命令

DAND — Doubleword And

64bit 論理積 RESPII



ニーモニック:

DAND rd, rs, rt

機能 :

FPR[rd] ← FPR[rs] and FPR[rt]

例外 :

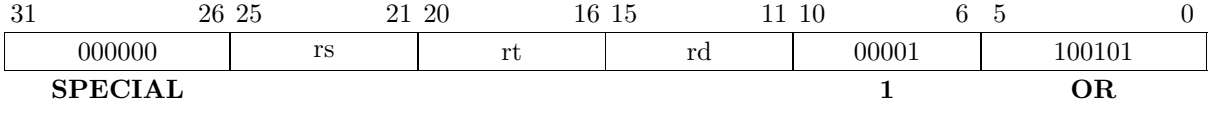
None

概要 :

浮動小数点レジスタを用いて演算を行う.

DOR — Doubleword Or

64bit 論理和 RESPII



ニーモニック:

DOR rd, rs, rt

機能 :

FPR[rd] ← FPR[rs] or FPR[rt]

例外 :

None

概要 :

浮動小数点レジスタを用いて演算を行う.

DXOR — Doubleword Exclusive Or

64bit 排他的論理和 RESPII

31 26	25 21	20 16	15 11	10 6	5 0
000000	rs	rt	rd	00001	100110
SPECIAL				**1**	**XOR**

ニーモニック:

DXOR rd, rs, rt

機能 :

FPR[rd] ← FPR[rs] xor FPR[rt]

例外 :

None

概要 :

浮動小数点レジスタを用いて演算を行う.

DNOR — Doubleword Not Or

64bit 否定論理和 RESPII

31 26	25 21	20 16	15 11	10 6	5 0
000000	rs	rt	rd	00001	100111
SPECIAL				**1**	**NOR**

ニーモニック:

DNOR rd, rs, rt

機能 :

FPR[rd] ← FPR[rs] nor FPR[rt]

例外 :

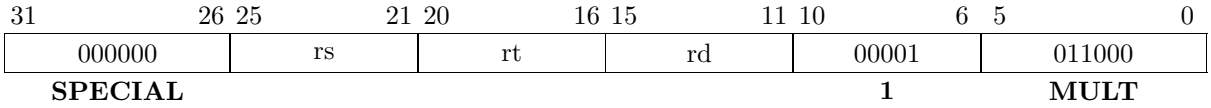
None

概要 :

浮動小数点レジスタを用いて演算を行う.

MULTH — Multiply Word on High Bit

符号付き乗算 RESPII



ニーモニック:

MULTH rd, rs, rt

機能 :

GPR[rd] ← GPR[rs] × GPR[rt]

例外 :

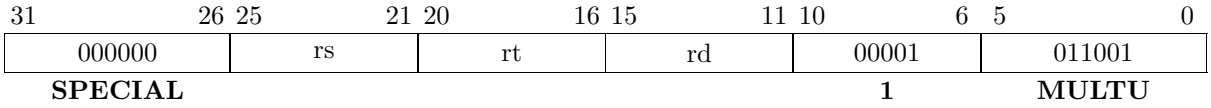
None

概要 :

デスティネーションレジスタに乗算結果の上位 32bit を格納する.

MULTUH — Multiply Word Unsigned on High Bit

符号無し乗算 RESPII



ニーモニック:

MULTUH rd, rs, rt

機能 :

GPR[rd] ← GPR[rs] × GPR[rt]

例外 :

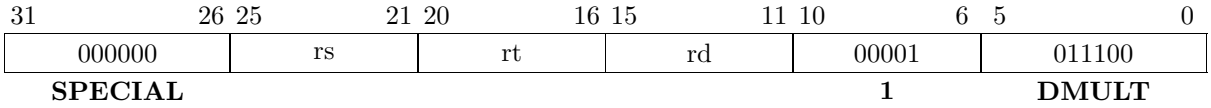
None

概要 :

デスティネーションレジスタに乗算結果の上位 32bit を格納する.

DMULTH — Doubleword Multiply on High Bit

符号付き 64bit 乗算 RESPII



ニーモニック:

DMULTH rd, rs, rt

機能 :

FPR[rd] ← FPR[rs] × FPR[rt]

例外 :

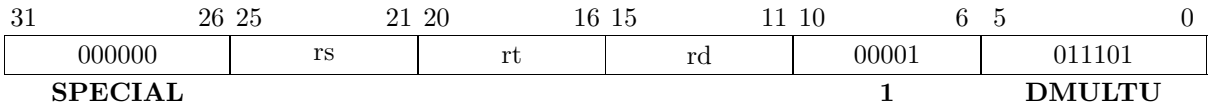
None

概要 :

浮動小数点レジスタを用いて演算を行う．デスティネーションレジスタに乗算結果の上位 64bit を格納する．

DMULTUH — Doubleword Multiply Unsigned on High Bit

符号無し 64bit 乗算 RESPII



ニーモニック:

DMULTUH rd, rs, rt

機能 :

FPR[rd] ← FPR[rs] × FPR[rt]

例外 :

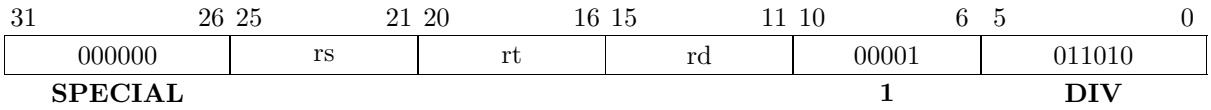
None

概要 :

浮動小数点レジスタを用いて演算を行う．デスティネーションレジスタに乗算結果の上位 64bit を格納する．

REM — Reminder Word

符号付き剰余 RESPII



31–26	25–21	20–16	15–11	10–6	5–0
000000	rs	rt	rd	00001	011010
SPECIAL				**1**	**DIV**

ニーモニック:

REM rd, rs, rt

機能 :

GPR[rd] ← GPR[rs] ÷ GPR[rt]

例外 :

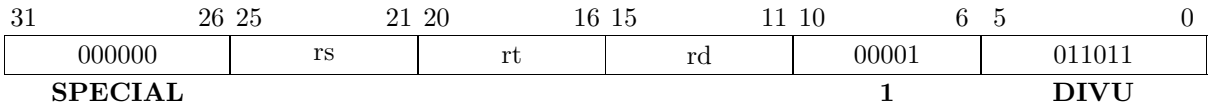
Divide by Zero :

概要 :

デスティネーションレジスタに剰余の結果を格納する.

REMU — Reminder Word Unsigned

符号無し剰余 RESPII



31–26	25–21	20–16	15–11	10–6	5–0
000000	rs	rt	rd	00001	011011
SPECIAL				**1**	**DIVU**

ニーモニック:

REMU rd, rs, rt

機能 :

GPR[rd] ← GPR[rs] ÷ GPR[rt]

例外 :

Divide by Zero :

概要 :

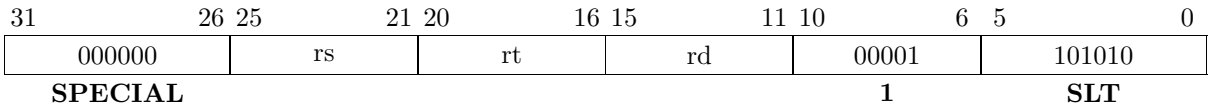
デスティネーションレジスタに剰余の結果を格納する.

DSLT

Doubleword Set on Less Than

符号付き 64bit 比較

RESPII



ニーモニック:

DSLT rd, rs, rt

機能 :

FPR[rd] ← (FPR[rs] < FPR[rt])

例外 :

None

概要 :

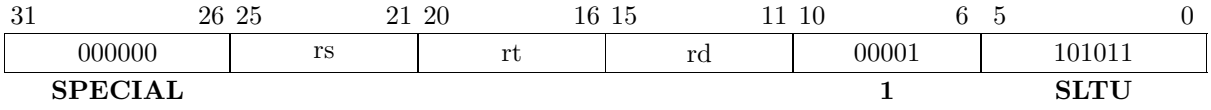
浮動小数点レジスタを用いて演算を行う.

DSLTU

Doubleword Set on Less Than Unsigned

符号無し 64bit 比較

RESPII



ニーモニック:

DSLTU rd, rs, rt

機能 :

FPR[rd] ← (FPR[rs] < FPR[rt])

例外 :

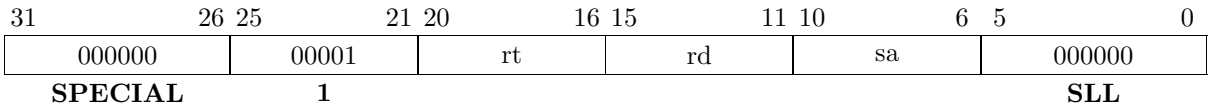
None

概要 :

浮動小数点レジスタを用いて演算を行う.

RTL — Rotate Left

左ローテーション RESPII



ニーモニック:

RTL rd, rt, sa

機能 :

GPR[rd] ← GPR[rt] <<< sa

例外 :

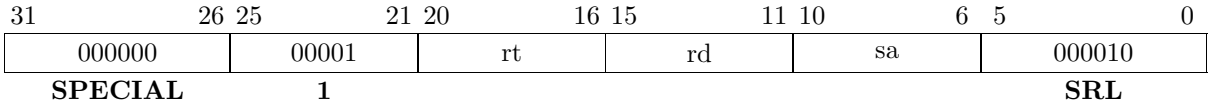
None

概要 :

左ローテーション演算.

RTR — Rotate Right

右ローテーション RESPII



ニーモニック:

RTR rd, rt, sa

機能 :

GPR[rd] ← GPR[rt] >>> sa

例外 :

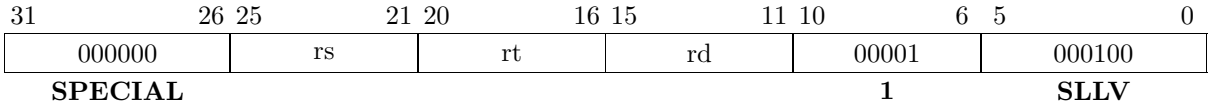
None

概要 :

右ローテーション演算.

RTLV Rotate Left Variable

左ローテーション RESPII



31 26	25 21	20 16	15 11	10 6	5 0
000000	rs	rt	rd	00001	000100
SPECIAL				**1**	**SLLV**

ニーモニック:

RTLV rd, rt, rs

機能 :

GPR[rd] ← GPR[rt] <<< GPR[rs]

例外 :

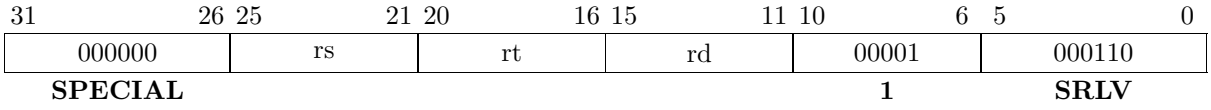
None

概要 :

左ローテーション演算.

RTRV Rotate Right Variable

右ローテーション RESPII



31 26	25 21	20 16	15 11	10 6	5 0
000000	rs	rt	rd	00001	000110
SPECIAL				**1**	**SRLV**

ニーモニック:

RTRV rd, rt, rs

機能 :

GPR[rd] ← GPR[rt] >>> GPR[rs]

例外 :

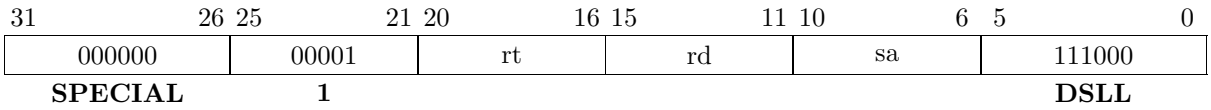
None

概要 :

右ローテーション演算.

DRTL — Doubleword Rotate Left

64bit 左ローテーション RESPII



ニーモニック:

DRTL rd, rt, sa

機能 :

FPR[rd] ← FPR[rt] <<< sa

例外 :

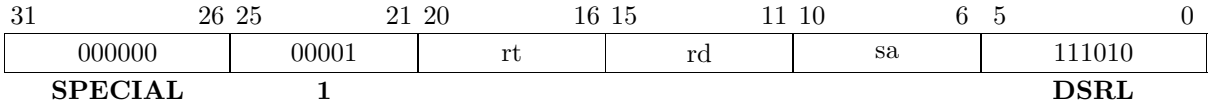
None

概要 :

浮動小数点レジスタを用いて演算を行う.

DRTR — Doubleword Rotate Right

64bit 右ローテーション RESPII



ニーモニック:

DRTR rd, rt, sa

機能 :

FPR[rd] ← FPR[rt] >>> sa

例外 :

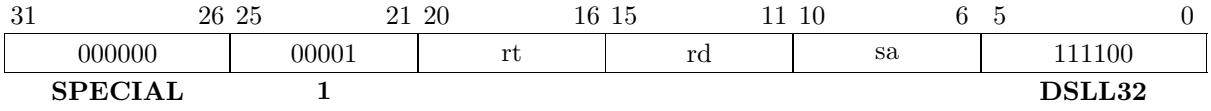
None

概要 :

浮動小数点レジスタを用いて演算を行う.

DRTL32 Doubleword Rotate Left plus 32

64bit 左ローテーション RESPII



ニーモニック:

DRTL rd, rt, sa

機能 :

FPR[rd] ← FPR[rt] <<< (sa + 32)

例外 :

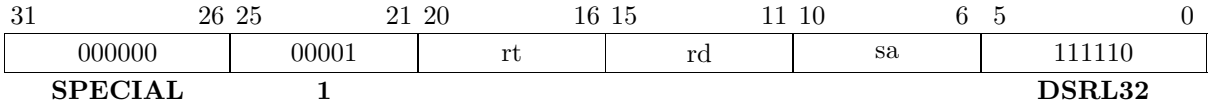
None

概要 :

浮動小数点レジスタを用いて演算を行う.

DRTR32 Doubleword Rotate Right plus 32

64bit 右ローテーション RESPII



ニーモニック:

DRTR32 rd, rt, sa

機能 :

FPR[rd] ← FPR[rt] >>> (sa + 32)

例外 :

None

概要 :

浮動小数点レジスタを用いて演算を行う.

DRTLV Doubleword Rotate Left Variable

64bit 左ローテーション RESPII

31 26	25 21	20 16	15 11	10 6	5 0
000000	rs	rt	rd	00001	010100
SPECIAL				**1**	**DSLLV**

ニーモニック:

DRTLV rd, rt, rs

機能 :

FPR[rd] ← FPR[rt] <<< FPR[rs]

例外 :

None

概要 :

浮動小数点レジスタを用いて演算を行う.

DRTRV Doubleword Rotate Right Variable

64bit 右ローテーション RESPII

31 26	25 21	20 16	15 11	10 6	5 0
000000	rs	rt	rd	00001	010110
SPECIAL				**1**	**DSRLV**

ニーモニック:

DRTRV rd, rt, rs

機能 :

FPR[rd] ← FPR[rt] >>> FPR[rs]

例外 :

None

概要 :

浮動小数点レジスタを用いて演算を行う.

3.3.3　転送命令

MTC1H — Move Word to Floating Point on High bit

レジスタ間転送 RESPII

31 26	25 21	20 16	15 11	10 6	5 0
010001	00100	rt	fs	00001	000000
COP1	**MT**			**1**	**0**

ニーモニック:

MTC1H rt, fs

機能：

FPR[fs] ← GPR[rt] || 0^{32}

例外：

None

概要：

浮動小数点レジスタの上位ビットに汎用レジスタの値を転送する.

MFC1H — Move Word from Floating Point on High bit

レジスタ間転送 RESPII

31 26	25 21	20 16	15 11	10 6	5 0
010001	00000	rt	fs	00001	000000
COP1	**MF**			**1**	**0**

ニーモニック:

MFC1H rt, fs

機能：

GPR[rt] ← FPR[fs] $_{63-32}$

例外：

None

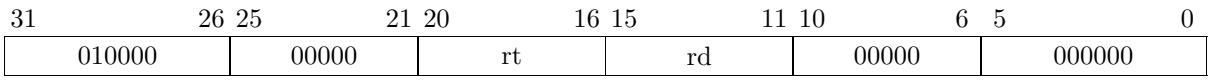
概要：

汎用レジスタに浮動小数点レジスタの上位ビットを転送する.

3.3.4 システム制御命令

MFC0 Move from System Control Register

システムレジスタリード命令 SYSTEM



31–26	25–21	20–16	15–11	10–6	5–0
010000	00000	rt	rd	00000	000000
COP0	**MF**			**0**	**CTRL**

ニーモニック:

MFC0 rt, rd

機能 :

GPR[rt] ← SYSTEM[GPR[rd]]

例外 :

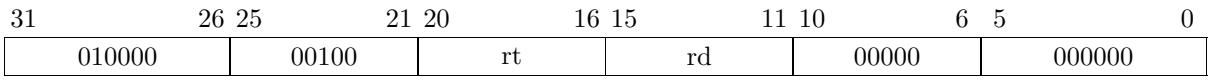
Coprocessor Unusable :

概要 :

システムレジスタから値を読み込む．システムレジスタのアドレスは GPR[rd] で指定する．

MTC0 Move to System Control Register

システムレジスタライト命令 SYSTEM



31–26	25–21	20–16	15–11	10–6	5–0
010000	00100	rt	rd	00000	000000
COP0	**MT**			**0**	**CTRL**

ニーモニック:

MTC0 rt, rd

機能 :

SYSTEM[GPR[rd]] ← GPR[rt]

例外 :

Coprocessor Unusable :

概要 :

システムレジスタに値を書き込む．システムレジスタのアドレスは GPR[rd] で指定する．

MFIMM　Move from Instruction MMU Control Register

命令 MMU 制御レジスタリード命令　SYSTEM

31 26	25 21	20 16	15 11	10 6	5 0
010000	00000	rt	rd	00000	000010
COP0	**MF**			**0**	**IMMU**

ニーモニック:

MFIMM rt, rd

機能 :

GPR[rt] ← IMMU[GPR[rd]]

例外 :

Coprocessor Unusable :

概要 :

命令 MMU 制御レジスタから値を読み込む．制御レジスタのアドレスは GPR[rd] で指定する．

MTIMM　Move to Instruction MMU Control Register

命令 MMU 制御レジスタライト命令　SYSTEM

31 26	25 21	20 16	15 11	10 6	5 0
010000	00100	rt	rd	00000	000010
COP0	**MT**			**0**	**IMMU**

ニーモニック:

MTIMM rt, rd

機能 :

IMMU[GPR[rd]] ← GPR[rt]

例外 :

Coprocessor Unusable :

概要 :

命令 MMU 制御レジスタに値を書き込む．制御レジスタのアドレスは GPR[rd] で指定する．

MFDMM — Move from Data MMU Control Register

データ MMU 制御レジスタリード命令 SYSTEM

31–26	25–21	20–16	15–11	10–6	5–0
010000	00000	rt	rd	00000	000011
COP0	**MF**			**0**	**DMMU**

ニーモニック:

MFDMM rt, rd

機能 :

GPR[rt] ← DMMU[GPR[rd]]

例外 :

Coprocessor Unusable :

概要 :

データ MMU 制御レジスタから値を読み込む. 制御レジスタのアドレスは GPR[rd] で指定する.

MTDMM — Move to Data MMU Control Register

データ MMU 制御レジスタライト命令 SYSTEM

31–26	25–21	20–16	15–11	10–6	5–0
010000	00100	rt	rd	00000	000011
COP0	**MT**			**0**	**DMMU**

ニーモニック:

MTDMM rt, rd

機能 :

DMMU[GPR[rd]] ← GPR[rt]

例外 :

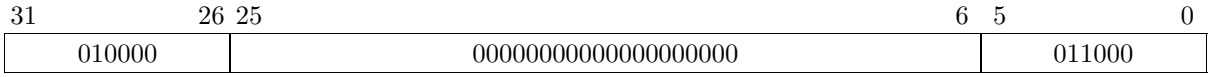
Coprocessor Unusable :

概要 :

データ MMU 制御レジスタに値を書き込む. 制御レジスタのアドレスは GPR[rd] で指定する.

ERET — Exception Return

例外復帰命令　　SYSTEM



31–26	25–6	5–0
010000	00000000000000000000	011000
COP0	**0**	**ERET**

ニーモニック:

ERET

機能 :

Exception Return

例外 :

None

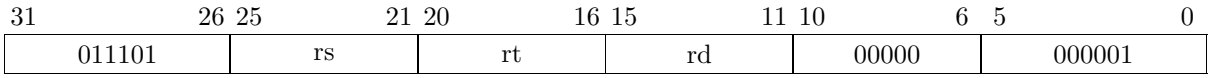
概要 :

例外処理から復帰する.

3.3.5　スレッド制御命令

MKTH — Make Thread

スレッド生成　　THREAD



31–26	25–21	20–16	15–11	10–6	5–0
011101	rs	rt	rd	00000	000001
THREAD				**0**	**MKTH**

ニーモニック:

MKTH rd, rs, rt

機能 :

GPR[rd] ← if(成功) then 1 else 0

例外 :

Coprocessor Unusable :

概要 :

スレッドの生成を行う．GPR[rs] に作成するスレッドの ID，GPR[rt] にスタートアドレスを指定する．スレッドの生成に成功すると GPR[rd] に 1 が，失敗すると 0 が返る.

DELTH

Delete Thread

スレッド削除

THREAD

31 26	25 21	20 16	15 11	10 6	5 0
011101	rs	00000	rd	00000	000010
THREAD		**0**		**0**	**DELTH**

ニーモニック:

DELTH rd, rs

機能 :

GPR[rd] ← if(成功) then 1 else 0

例外 :

Coprocessor Unusable :

概要 :

スレッドの削除を行う．GPR[rs] に削除するスレッドの ID を指定する．スレッドの削除に成功すると GPR[rd] に 1 が，失敗すると 0 が返る．

CHGPR

Change Priority

優先度の変更

THREAD

31 26	25 21	20 16	15 11	10 6	5 0
011101	rs	rt	rd	00000	000011
THREAD				**0**	**CHGPR**

ニーモニック:

CHGPR rd, rs, rt

機能 :

GPR[rd] ← if(成功) then 1 else 0

例外 :

Coprocessor Unusable :

概要 :

優先度の変更を行う．GPR[rs] に変更するスレッドの ID，GPR[rt] に新しい優先度を指定する．優先度の変更に成功すると GPR[rd] に 1 が，失敗すると 0 が返る．

CHGST — Change Status

状態の変更 THREAD

31–26	25–21	20–16	15–11	10–6	5–0
011101	rs	rt	rd	00000	000100
THREAD				**0**	**CHGST**

ニーモニック:

CHGST rd, rs, rt

機能 :

GPR[rd] ← if(成功) then 1 else 0

例外 :

Coprocessor Unusable :

概要 :

スレッドの状態を変更する．GPR[rs] に変更するスレッド ID の，GPR[rt] に新しい状態を指定する．状態の変更に成功すると GPR[rd] に 1 が，失敗すると 0 が返る．

RUNTH — Run Thread

スレッドの実行 THREAD

31–26	25–21	20–16	15–11	10–6	5–0
011101	rs	00000	rd	00000	000101
THREAD		**0**		**0**	**RUNTH**

ニーモニック:

RUNTH rd, rs

機能 :

GPR[rd] ← if(成功) then 1 else 0

例外 :

Coprocessor Unusable :

概要 :

スレッドを実行状態する．GPR[rs] に実行するスレッドの ID を指定する．スレッドの実行に成功すると GPR[rd] に 1 が，失敗すると 0 が返る．

STOPTH — Stop Thread

スレッドの停止 THREAD

31–26	25–21	20–16	15–11	10–6	5–0
011101	rs	00000	rd	00000	000110
THREAD		**0**		**0**	**STOPTH**

ニーモニック:

STOPTH rd, rs

機能 :

GPR[rd] ← if(成功) then 1 else 0

例外 :

Coprocessor Unusable :

概要 :

スレッドを停止状態にする．GPR[rs] に停止するスレッド ID を指定する．スレッドの停止に成功すると GPR[rd] に 1 が，失敗すると 0 が返る．

STOPSLF — Stop Myself

スレッドの停止 THREAD

31–26	25–16	15–11	10–6	5–0
011101	0000000000	rd	00000	000111
THREAD	**0**		**0**	**STOPSLF**

ニーモニック:

STOPSLF rd

機能 :

GPR[rd] ← if(成功) then 1 else 0

例外 :

Coprocessor Unusable :

概要 :

自分自身のスレッドを停止する．スレッドの停止に成功すると GPR[rd] に 1 が，失敗すると 0 が返る．

BKUPTH

Backup Thread

スレッドの退避

THREAD

31 26	25 21	20 16	15 11	10 6	5 0
011101	rs	00000	rd	00000	001000
THREAD		**0**		**0**	**BKUPTH**

ニーモニック:

BKUPTH rd, rs

機能 :

GPR[rd] ← if(成功) then 1 else 0

例外 :

Coprocessor Unusable :

概要 :

アクティブスレッドをコンテキストキャッシュに退避する．GPR[rs] に退避するスレッドの ID を指定する．スレッドの退避に成功すると GPR[rd] に 1 が，失敗すると 0 が返る．

BKUPSLF

Backup Myself

スレッドの退避

THREAD

31 26	25 16	15 11	10 6	5 0
011101	0000000000	rd	00000	001001
THREAD	**0**		**0**	**BKUPSLF**

ニーモニック:

BKUPSLF rd

機能 :

GPR[rd] ← if(成功) then 1 else 0

例外 :

Coprocessor Unusable :

概要 :

自分自身をコンテキストキャッシュに退避する．優先度の変更に成功すると GPR[rd] に 1 が，失敗すると 0 が返る．

RSTRTH

Restore Thread

スレッドの復帰

THREAD

31 26	25 21	20 16	15 11	10 6	5 0
011101	rs	00000	rd	00000	001010
THREAD		**0**		**0**	**RSTRTH**

ニーモニック:

RSTRTH rd, rs

機能 :

GPR[rd] ← if(成功) then 1 else 0

例外 :

Coprocessor Unusable :

概要 :

キャッシュスレッドをコンテキストキャッシュから復帰する．GPR[rs] に復帰するスレッドの ID を指定する．スレッドの退避に成功すると GPR[rd] に 1 が，失敗すると 0 が返る．

SWAPTH

Swap Thread

スレッドの入れ換え

THREAD

31 26	25 21	20 16	15 11	10 6	5 0
011101	rs	rt	rd	00000	001011
THREAD				**0**	**SWAPTH**

ニーモニック:

SWAPTH rd, rs, rt

機能 :

GPR[rd] ← if(成功) then 1 else 0

例外 :

Coprocessor Unusable :

概要 :

アクティブスレッドとキャッシュスレッドを入れ換える．GPR[rs] に退避するアクティブスレッドの ID，GPR[rt] に復帰するキャッシュスレッドを指定する．スレッドの入れ換えに成功すると GPR[rd] に 1 が，失敗すると 0 が返る．

SWAPSLF — Swap Myself

スレッドの入れ換え　　THREAD

31–26	25–21	20–16	15–11	10–6	5–0
011101	00000	rt	rd	00000	001100
THREAD	**0**			**0**	**SWAPSLF**

ニーモニック:

SWAPSLF rd, rt

機能 :

GPR[rd] ← if(成功) then 1 else 0

例外 :

Coprocessor Unusable :

概要 :

自分自身とキャッシュスレッドを入れ換える．GPR[rt] に復帰するキャッシュスレッドを指定する．スレッドの入れ換えに成功すると GPR[rd] に 1 が，失敗すると 0 が返る．

CPTHTOA — Copy Thread to Active Thread

スレッドのコピー　　THREAD

31–26	25–21	20–16	15–11	10–6	5–0
011101	rs	rt	rd	00000	001101
THREAD				**0**	**CPTHTOA**

ニーモニック:

CPTHTOA rd, rs, rt

機能 :

GPR[rd] ← if(成功) then 1 else 0

例外 :

Coprocessor Unusable :

概要 :

アクティブスレッドを別のアクティブスレッドとしてコピーする．GPR[rs] にコピー元のアクティブスレッドの ID，GPR[rt] にコピー先のアクティブスレッドの ID を指定する．スレッドのコピーに成功すると GPR[rd] に 1 が，失敗すると 0 が返る．

CPTHTOM Copy Thread to Cache Thread (Memory)

スレッドのコピー THREAD

31–26	25–21	20–16	15–11	10–6	5–0
011101	rs	rt	rd	00000	001110
THREAD				**0**	**CPTHTOM**

ニーモニック:

CPTHTOM rd, rs, rt

機能 :

GPR[rd] ← if(成功) then 1 else 0

例外 :

Coprocessor Unusable :

概要 :

アクティブスレッドを別のキャッシュスレッドとしてコピーする．GPR[rs] にコピー元のアクティブスレッドの ID，GPR[rt] にコピー先のキャッシュスレッドの ID を指定する．スレッドのコピーに成功すると GPR[rd] に 1 が，失敗すると 0 が返る．

GETTT Get Thread Table

スレッドテーブルの参照 THREAD

31–26	25–21	20–16	15–11	10–6	5–0
011101	rs	00000	rd	00000	001111
THREAD		**0**		**0**	**GETTT**

ニーモニック:

GETTT rd, rs

機能 :

GPR[rd] ← ThreadTable of GPR[rs]

例外 :

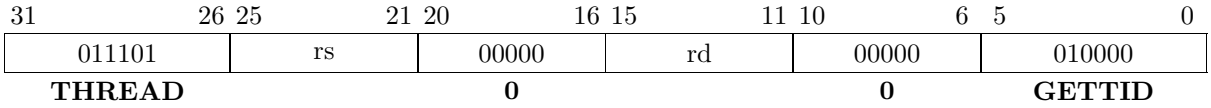
Coprocessor Unusable :

概要 :

スレッドテーブルから値を読み込む．GPR[rs] に読み込むスレッドの ID を指定する．GPR[rd] にスレッドテーブルの内容が返る．

GETTID — Get Thread ID

スレッド ID の参照　　THREAD



31–26	25–21	20–16	15–11	10–6	5–0
011101	rs	00000	rd	00000	010000
THREAD		**0**		**0**	**GETTID**

ニーモニック:

GETTID rd, rs

機能 :

GPR[rd] ← ThreadID of GPR[rs]

例外 :

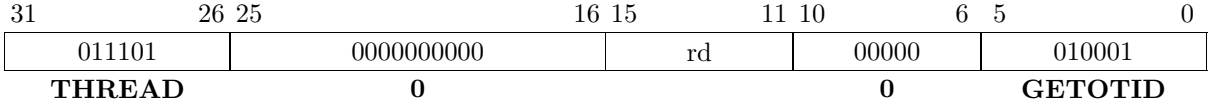
Coprocessor Unusable :

概要 :

コンテキスト ID からスレッド ID を調べる．GPR[rs] に調べるスレッドのコンテキスト ID を指定する．GPR[rd] にスレッド ID が返る．

GETOTID — Get Own Thread ID

スレッド ID の参照　　THREAD



31–26	25–16	15–11	10–6	5–0
011101	0000000000	rd	00000	010001
THREAD	**0**		**0**	**GETOTID**

ニーモニック:

GETOTID rd

機能 :

GPR[rd] ← ThreadID of Myself

例外 :

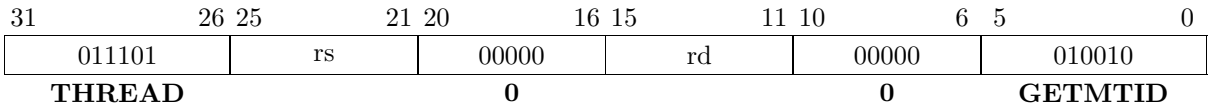
Coprocessor Unusable :

概要 :

自分自身のスレッド ID を調べる．GPR[rd] にスレッド ID が返る．

GETMTID Get Cache Thread ID (Get Memory Thread ID)

スレッド ID の参照 THREAD



31–26	25–21	20–16	15–11	10–6	5–0
011101	rs	00000	rd	00000	010010
THREAD		**0**		**0**	**GETMTID**

ニーモニック:

GETMTID rd, rs

機能 :

GPR[rd] ← ThreadID of GPR[rs]

例外 :

Coprocessor Unusable :

概要 :

コンテキスト ID からスレッド ID を調べる．GPR[rs] に調べるキャッシュスレッドのコンテキスト ID を指定する．GPR[rd] にスレッド ID が返る．

GETCNUM Get Context ID Number

コンテキスト ID の参照 THREAD

31 26	25 21	20 16	15 11	10 6	5 0
011101	rs	00000	rd	00000	010011
THREAD		**0**		**0**	**GETCNUM**

ニーモニック:

GETCNUM rd, rs

機能 :

GPR[rd] ← ContextID of GPR[rs]

例外 :

Coprocessor Unusable :

概要 :

スレッド ID からコンテキスト ID を調べる．GPR[rs] に調べるスレッドの ID を指定する．GPR[rd] にコンテキスト ID が返る．8bit 目が 1 の場合，スレッドはアクティブスレッドにあり，2bit 目から 0bit 目にコンテキスト ID が返る．6bit 目が 1 の場合，スレッドはキャッシュスレッドにあり，4bit 目から 0bit 目にコンテキストキャッシュにおけるコンテキスト ID が返る．

3.3.6 SIMD 演算命令

SADD.size SIMD Add

SIMD 加算 SIMD

31 26	25 21	20 16	15 11	10 8	7 6	5 0
011100	rs	rt	rd	000	size	100000
SIMD				**0**		**ADD**

ニーモニック:

SADD.8 rd, rs, rt (size = 01)
SADD.16 rd, rs, rt (size = 10)
SADD.32 rd, rs, rt (size = 11)

機能 :

FPR[rd] ← FPR[rs] + FPR[rt]

例外 :

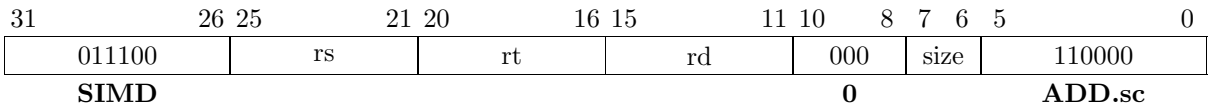
None

概要 :

浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算.

SADD.size.sc SIMD Add Scalar

SIMD 加算 SIMD



ニーモニック:

SADD.8.sc rd, rs, rt	(size = 01)
SADD.16.sc rd, rs, rt	(size = 10)
SADD.32.sc rd, rs, rt	(size = 11)

機能 :

FPR[rd] ← FPR[rs] + FPR[rt]

例外 :

None

概要 :

浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算．sc(scalar) は FPR[rt] の下位 bit が各フィールドに複製されて演算される．

SSUB.size SIMD Subtract

SIMD 減算 SIMD

31 26	25 21	20 16	15 11	10 8	7 6	5 0
011100	rs	rt	rd	000	size	100010
SIMD				**0**		**SUB**

ニーモニック:

SSUB.8 rd, rs, rt (size = 01)
SSUB.16 rd, rs, rt (size = 10)
SSUB.32 rd, rs, rt (size = 11)

機能 :

FPR[rd] ← FPR[rs] − FPR[rt]

例外 :

None

概要 :

浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算.

SSUB.size.sc SIMD Subtract Scalar

SIMD 減算 SIMD

31 26	25 21	20 16	15 11	10 8	7 6	5 0
011100	rs	rt	rd	000	size	110010
SIMD				**0**		**SUB.sc**

ニーモニック:

SSUB.8.sc rd, rs, rt (size = 01)
SSUB.16.sc rd, rs, rt (size = 10)
SSUB.32.sc rd, rs, rt (size = 11)

機能 :

FPR[rd] ← FPR[rs] − FPR[rt]

例外 :

None

概要 :

浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算．sc(scalar) は FPR[rt] の下位 bit が各フィールドに複製されて演算される．

SMULT.size — SIMD Multiply

符号付き SIMD 乗算 SIMD

31 26	25 21	20 16	15 11	10 8	7 6	5 0
011100	rs	rt	rd	000	size	011000
SIMD				**0**		**MULT**

ニーモニック:

SMULT.8 rd, rs, rt (size = 01)
SMULT.16 rd, rs, rt (size = 10)
SMULT.32 rd, rs, rt (size = 11)

機能 :

FPR[rd] ← FPR[rs] × FPR[rt]

例外 :

None

概要 :

浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算.

SMULT.size.sc　SIMD Multiply Scalar

符号付き SIMD 乗算　SIMD

31–26	25–21	20–16	15–11	10–8	7–6	5–0
011100	rs	rt	rd	000	size	101000
SIMD				**0**		**MULT.sc**

ニーモニック:

SMULT.8.sc rd, rs, rt	(size = 01)
SMULT.16.sc rd, rs, rt	(size = 10)
SMULT.32.sc rd, rs, rt	(size = 11)

機能：

FPR[rd] ← FPR[rs] × FPR[rt]

例外：

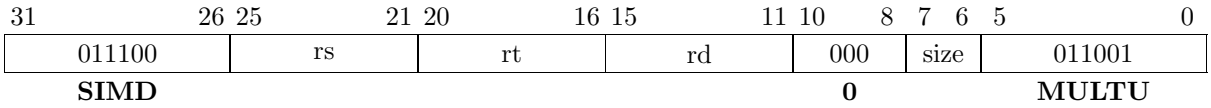
None

概要：

浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算．sc(scalar) は FPR[rt] の下位 bit が各フィールドに複製されて演算される．

SMULTU.size SIMD Multiply Unsigned

符号無し SIMD 乗算 SIMD



31–26	25–21	20–16	15–11	10–8	7–6	5–0
011100	rs	rt	rd	000	size	011001
SIMD				**0**		**MULTU**

ニーモニック:

SMULTU.8 rd, rs, rt (size = 01)
SMULTU.16 rd, rs, rt (size = 10)
SMULTU.32 rd, rs, rt (size = 11)

機能 :

FPR[rd] ← FPR[rs] × FPR[rt]

例外 :

None

概要 :

浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算.

SMULTU.size.sc SIMD Multiply Unsigned Scalar

符号無し SIMD 乗算 SIMD

31 26	25 21	20 16	15 11	10 8	7 6	5 0
011100	rs	rt	rd	000	size	101001
SIMD				**0**		**MULTU.sc**

ニーモニック:

SMULTU.8.sc rd, rs, rt (size = 01)
SMULTU.16.sc rd, rs, rt (size = 10)
SMULTU.32.sc rd, rs, rt (size = 11)

機能 :

FPR[rd] ← FPR[rs] × FPR[rt]

例外 :

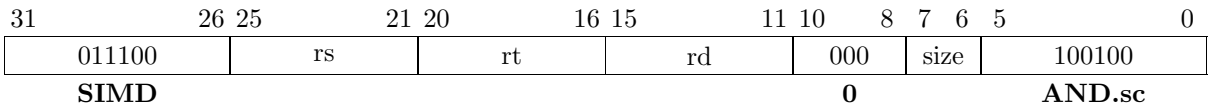
None

概要 :

浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算．sc(scalar) は FPR[rt] の下位 bit が各フィールドに複製されて演算される．

SAND.size.sc SIMD And Scalar

SIMD 論理積 SIMD



ニーモニック:

SAND.8.sc rd, rs, rt (size = 01)
SAND.16.sc rd, rs, rt (size = 10)
SAND.32.sc rd, rs, rt (size = 11)

機能 :

FPR[rd] ← FPR[rs] and FPR[rt]

例外 :

None

概要 :

浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算．sc(scalar) は FPR[rt] の下位 bit が各フィールドに複製されて演算される．

SOR.size.sc SIMD Or Scalar

SIMD 論理和 SIMD

31 26	25 21	20 16	15 11	10 8	7 6	5 0
011100	rs	rt	rd	000	size	100101
SIMD				**0**		**OR.sc**

ニーモニック:

SOR.8.sc rd, rs, rt (size = 01)
SOR.16.sc rd, rs, rt (size = 01)
SOR.32.sc rd, rs, rt (size = 01)

機能 :

FPR[rd] ← FPR[rs] or FPR[rt]

例外 :

None

概要 :

浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算．sc(scalar) は FPR[rt] の下位 bit が各フィールドに複製されて演算される．

SXOR.size.sc SIMD Exclusive Or Scalar

SIMD 排他的論理和 SIMD

31 26	25 21	20 16	15 11	10 8	7 6	5 0
011100	rs	rt	rd	000	size	100110
SIMD				**0**		**XOR.sc**

ニーモニック:

SXOR.8.sc rd, rs, rt (size = 01)
SXOR.16.sc rd, rs, rt (size = 10)
SXOR.32.sc rd, rs, rt (size = 11)

機能 :

FPR[rd] ← FPR[rs] xor FPR[rt]

例外 :

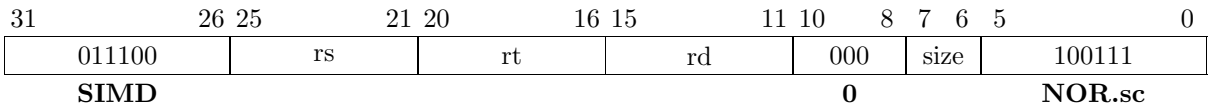
None

概要 :

浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算．sc(scalar) は FPR[rt] の下位 bit が各フィールドに複製されて演算される．

SNOR.size.sc SIMD Not Or Scalar

SIMD 否定論理和 SIMD



31 26	25 21	20 16	15 11	10 8	7 6	5 0
011100	rs	rt	rd	000	size	100111
SIMD				**0**		**NOR.sc**

ニーモニック:

SNOR.8.sc rd, rs, rt	(size = 01)
SNOR.16.sc rd, rs, rt	(size = 10)
SNOR.32.sc rd, rs, rt	(size = 11)

機能 :

FPR[rd] ← FPR[rs] nor FPR[rt]

例外 :

None

概要 :

浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算．sc(scalar) は FPR[rt] の下位 bit が各フィールドに複製されて演算される．

SSLT.size

SIMD Set Less Than

SIMD 大小判定

SIMD

31 26	25 21	20 16	15 11	10 8	7 6	5 0
011100	rs	rt	rd	000	size	101010
SIMD				**0**		**SLT**

ニーモニック:

SSLT.8 rd, rs, rt (size = 01)
SSLT.16 rd, rs, rt (size = 10)
SSLT.32 rd, rs, rt (size = 11)

機能 :

FPR[rd] ← (FPR[rs] < FPR[rt])

例外 :

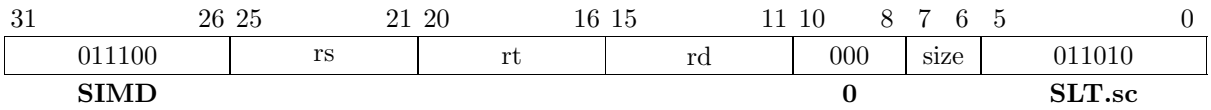
None

概要 :

浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算.

SSLT.size.sc SIMD Set Less Than Scalar

SIMD 大小判定 SIMD



ニーモニック:

SSLT.8.sc rd, rs, rt (size = 01)
SSLT.16.sc rd, rs, rt (size = 10)
SSLT.32.sc rd, rs, rt (size = 11)

機能 :

FPR[rd] ← (FPR[rs] < FPR[rt])

例外 :

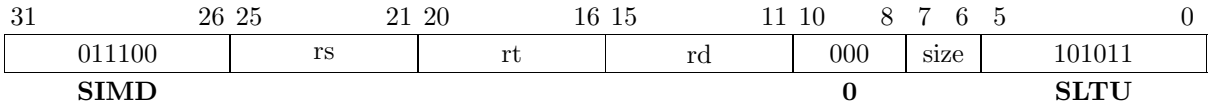
None

概要 :

浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算．sc(scalar) は FPR[rt] の下位 bit が各フィールドに複製されて演算される．

SSLTU.size — SIMD Set Less Than Unsigned

符号無し SIMD 大小判定 SIMD



31–26	25–21	20–16	15–11	10–8	7–6	5–0
011100	rs	rt	rd	000	size	101011
SIMD				**0**		**SLTU**

ニーモニック:

SSLTU.8 rd, rs, rt (size = 01)
SSLTU.16 rd, rs, rt (size = 10)
SSLTU.32 rd, rs, rt (size = 11)

機能 :

FPR[rd] ← (FPR[rs] < FPR[rt])

例外 :

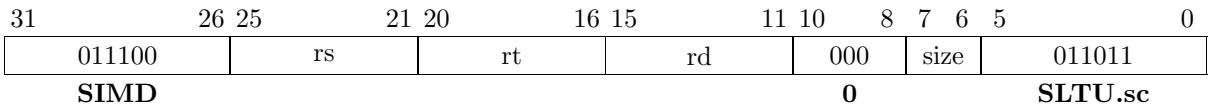
None

概要 :

浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算.

SSLTU.size.sc SIMD Set Less Than Unsigned Scalar

符号無し SIMD 大小判定 SIMD



ニーモニック:

SSLTU.8.sc rd, rs, rt (size = 01)
SSLTU.16.sc rd, rs, rt (size = 10)
SSLTU.32.sc rd, rs, rt (size = 11)

機能 :

FPR[rd] ← (FPR[rs] < FPR[rt])

例外 :

None

概要 :

浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算．sc(scalar) は FPR[rt] の下位 bit が各フィールドに複製されて演算される．

SSLLV.size — SIMD Shift Left Logical Variable

SIMD 左論理シフト SIMD

31 26	25 21	20 16	15 11	10 8	7 6	5 0
011100	rs	rt	rd	000	size	000100
SIMD				**0**		**SLLV**

ニーモニック:

SSLLV.8 rd, rt, rs (size = 01)
SSLLV.16 rd, rt, rs (size = 10)
SSLLV.32 rd, rt, rs (size = 11)

機能 :

FPR[rd] ← FPR[rt] ≪ FPR[rs]

例外 :

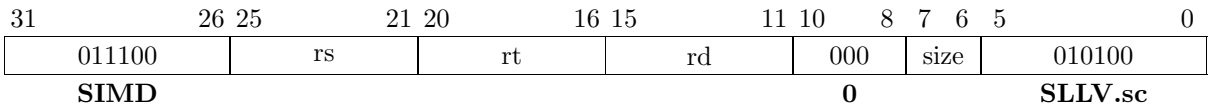
None

概要 :

浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算.

SSLLV.size.sc SIMD Shift Left Logical Variable Scalar

SIMD 左論理シフト SIMD



31 26	25 21	20 16	15 11	10 8	7 6	5 0
011100	rs	rt	rd	000	size	010100
SIMD				**0**		**SLLV.sc**

ニーモニック:

SSLLV.8.sc rd, rt, rs	(size = 01)
SSLLV.16.sc rd, rt, rs	(size = 10)
SSLLV.32.sc rd, rt, rs	(size = 11)

機能 :

FPR[rd] ← FPR[rt] ≪ FPR[rs]

例外 :

None

概要 :

浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算．sc(scalar) は FPR[rt] の下位 bit が各フィールドに複製されて演算される．

SSRLV.size — SIMD Shift Right Logical Variable

SIMD 右論理シフト SIMD

31 26	25 21	20 16	15 11	10 8	7 6	5 0
011100	rs	rt	rd	000	size	000110
SIMD				**0**		**SRLV**

ニーモニック:

SSRLV.8 rd, rt, rs (size = 01)
SSRLV.16 rd, rt, rs (size = 10)
SSRLV.32 rd, rt, rs (size = 11)

機能 :

FPR[rd] ← FPR[rt] ≫ FPR[rs]

例外 :

None

概要 :

浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算.

SSRLV.size.sc SIMD Shift Right Logical Variable Scalar

SIMD 右論理シフト SIMD

31 26	25 21	20 16	15 11	10 8	7 6	5 0
011100	rs	rt	rd	000	size	010110
SIMD				**0**		**SRLV.sc**

ニーモニック:

SSRLV.8.sc rd, rt, rs (size = 01)
SSRLV.16.sc rd, rt, rs (size = 10)
SSRLV.32.sc rd, rt, rs (size = 11)

機能 :

FPR[rd] ← FPR[rt] ≫ FPR[rs]

例外 :

None

概要 :

浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算．sc(scalar) は FPR[rt] の下位 bit が各フィールドに複製されて演算される．

SSRAV.size SIMD Shift Right Arithmetic Variable

SIMD 右算術シフト SIMD

31 26	25 21	20 16	15 11	10 8	7 6	5 0
011100	rs	rt	rd	000	size	000111
SIMD				**0**		**SRAV**

ニーモニック:

SSRAV.8 rd, rt, rs (size = 01)
SSRAV.16 rd, rt, rs (size = 10)
SSRAV.32 rd, rt, rs (size = 11)

機能 :

FPR[rd] ← FPR[rt] ≫ FPR[rs]

例外 :

None

概要 :

浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算.

SSRAV.size.sc SIMD Shift Right Arithmetic Variable Scalar

SIMD 右算術シフト SIMD

31 26	25 21	20 16	15 11	10 8	7 6	5 0
011100	rs	rt	rd	000	size	010111
SIMD				**0**		**SRAV.sc**

ニーモニック:

SSRAV.8.sc rd, rt, rs (size = 01)
SSRAV.16.sc rd, rt, rs (size = 10)
SSRAV.32.sc rd, rt, rs (size = 11)

機能 :

FPR[rd] ← FPR[rt] ≫ FPR[rs]

例外 :

None

概要 :

浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算．sc(scalar) は FPR[rt] の下位 bit が各フィールドに複製されて演算される．

SRTLV.size SIMD Rotate Left Variable

SIMD 左ローテーション SIMD

31 26	25 21	20 16	15 11	10 8	7 6	5 0
011100	rs	rt	rd	000	size	000000
SIMD				**0**		**RTL**

ニーモニック:

SRTLV.8 rd, rt, rs (size = 01)
SRTLV.16 rd, rt, rs (size = 10)
SRTLV.32 rd, rt, rs (size = 11)

機能 :

FPR[rd] ← FPR[rt] <<< FPR[rs]

例外 :

None

概要 :

浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算.

SRTLV.size.sc SIMD Rotate Left Variable Scalar

SIMD 左ローテーション SIMD

31 26	25 21	20 16	15 11	10 8	7 6	5 0
011100	rs	rt	rd	000	size	010000
SIMD				**0**		**RTL.sc**

ニーモニック:

SRTLV.8.sc rd, rt, rs (size = 01)
SRTLV.16.sc rd, rt, rs (size = 10)
SRTLV.32.sc rd, rt, rs (size = 11)

機能 :

FPR[rd] ← FPR[rt] <<< FPR[rs]

例外 :

None

概要 :

浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算．sc(scalar) は FPR[rt] の下位 bit が各フィールドに複製されて演算される．

SRTRV.size

SIMD Rotate Right Variable

SIMD 右ローテーション

SIMD

31 26	25 21	20 16	15 11	10 8	7 6	5 0
011100	rs	rt	rd	000	size	000010
SIMD				**0**		**RTR**

ニーモニック:

SRTRV.8 rd, rt, rs (size = 01)
SRTRV.16 rd, rt, rs (size = 10)
SRTRV.32 rd, rt, rs (size = 11)

機能 :

FPR[rd] ← FPR[rt] >>> FPR[rs]

例外 :

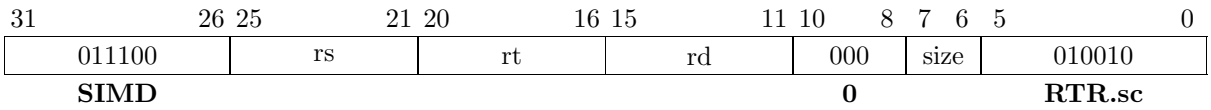
None

概要 :

浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算.

SRTRV.size.sc　SIMD Rotate Right Variable Scalar

SIMD 右ローテーション　SIMD



31–26	25–21	20–16	15–11	10–8	7–6	5–0
011100	rs	rt	rd	000	size	010010
SIMD				**0**		**RTR.sc**

ニーモニック:

SRTRV.8.sc rd, rt, rs　(size = 01)
SRTRV.16.sc rd, rt, rs　(size = 10)
SRTRV.32.sc rd, rt, rs　(size = 11)

機能：

FPR[rd] ← FPR[rt] >>> FPR[rs]

例外：

None

概要：

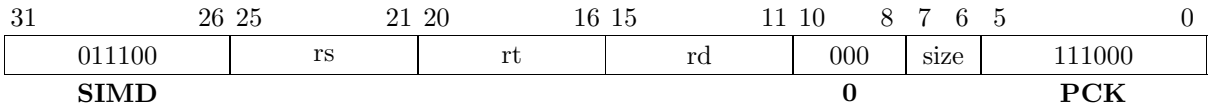
浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算．sc(scalar) は FPR[rt] の下位 bit が各フィールドに複製されて演算される．

PCK

Pack Data

データパッキング

SIMD



ニーモニック:

PCK.8 rd, rs, rt (size = 01)
PCK.16 rd, rs, rt (size = 10)
PCK.32 rd, rs, rt (size = 11)

機能 :

FPR[rd] ← pack(FPR[rs], FPR[rt])

例外 :

None

概要 :

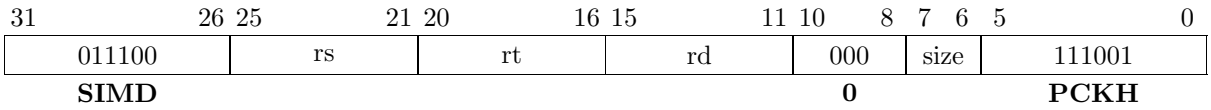
浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算.

PCKH

Pack Data on High Bit

データパッキング

SIMD



ニーモニック:

PCKH.8 rd, rs, rt (size = 01)
PCKH.16 rd, rs, rt (size = 10)
PCKH.32 rd, rs, rt (size = 11)

機能 :

FPR[rd] ← packh(FPR[rs], FPR[rt])

例外 :

None

概要 :

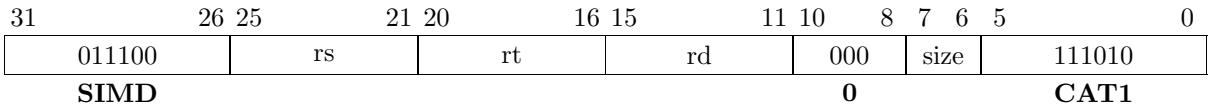
浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算.

CAT1

Concatenate Data Type1

データ結合

SIMD



ニーモニック:

CAT1.8 rd, rs, rt (size = 01)

CAT1.16 rd, rs, rt (size = 10)

CAT1.32 rd, rs, rt (size = 11)

機能 :

FPR[rd] ← cat1(FPR[rs], FPR[rt])

例外 :

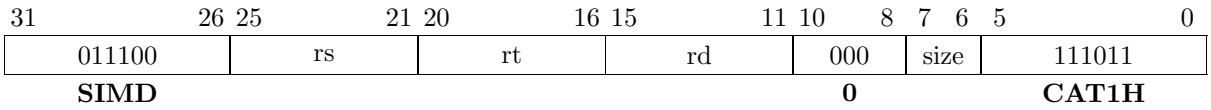
None

概要 :

浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算.

CAT1H Concatenate Data Type1 on High Bit

データ結合 SIMD



ニーモニック:

CAT1H.8 rd, rs, rt (size = 01)
CAT1H.16 rd, rs, rt (size = 10)
CAT1H.32 rd, rs, rt (size = 11)

機能 :

FPR[rd] ← cat1h(FPR[rs], FPR[rt])

例外 :

None

概要 :

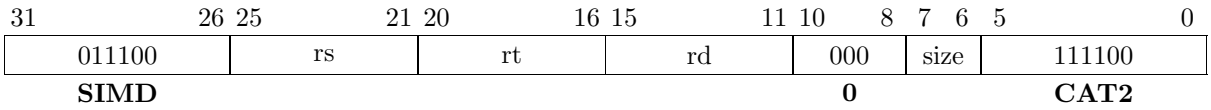
浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算.

CAT2

Concatenate Data Type2

データ結合

SIMD



ニーモニック:

CAT2.8 rd, rs, rt (size = 01)

CAT2.16 rd, rs, rt (size = 10)

CAT2.32 rd, rs, rt (size = 11)

機能 :

FPR[rd] ← cat2(FPR[rs], FPR[rt])

例外 :

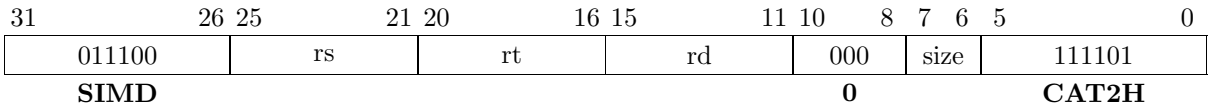
None

概要 :

浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算.

CAT2H Concatenate Data Type2 on High Bit

データ結合 SIMD



ニーモニック:

CAT2H.8 rd, rs, rt	(size = 01)
CAT2H.16 rd, rs, rt	(size = 10)
CAT2H.32 rd, rs, rt	(size = 11)

機能 :

FPR[rd] ← cat2h(FPR[rs], FPR[rt])

例外 :

None

概要 :

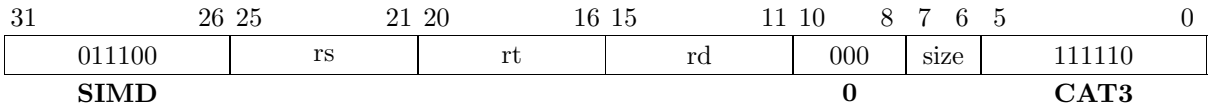
浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算.

CAT3

Concatenate Data Type3

データ結合

SIMD



ニーモニック:

CAT3.8 rd, rs, rt (size = 01)
CAT3.16 rd, rs, rt (size = 10)
CAT3.32 rd, rs, rt (size = 11)

機能 :

FPR[rd] ← cat3(FPR[rs], FPR[rt])

例外 :

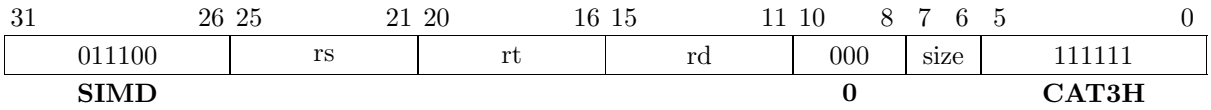
None

概要 :

浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算.

CAT3H Concatenate Data Type3 on High Bit

データ結合 SIMD



ニーモニック:

CAT3H.8 rd, rs, rt (size = 01)
CAT3H.16 rd, rs, rt (size = 10)
CAT3H.32 rd, rs, rt (size = 11)

機能 :

FPR[rd] ← cat3h(FPR[rs], FPR[rt])

例外 :

None

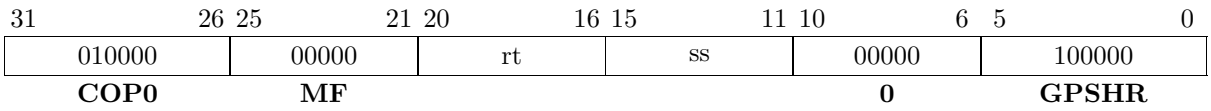
概要 :

浮動小数点レジスタを用いて演算を行う．8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算.

3.3.7 同期命令

RGPSH Read Shared(General Purpose Register)

同期命令 SYNC



31–26	25–21	20–16	15–11	10–6	5–0
010000	00000	rt	ss	00000	100000
COP0	**MF**			**0**	**GPSHR**

ニーモニック:

RGPSH rt, ss

機能 :

GPR[rt] ← SHARE[ss]

例外 :

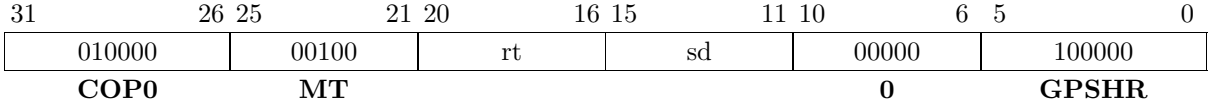
None

概要 :

共有レジスタから汎用レジスタに値を読み込む.

WGPSH Write Shared(General Purpose Register)

同期命令 SYNC



31–26	25–21	20–16	15–11	10–6	5–0
010000	00100	rt	sd	00000	100000
COP0	**MT**			**0**	**GPSHR**

ニーモニック:

WGPSH rt, sd

機能 :

SHARE[sd] ← GPR[rt]

例外 :

None

概要 :

汎用レジスタから共有レジスタに値を書き込む.

RFPSH — Read Shared(Floating-Point Register)

同期命令 SYNC

31 26	25 21	20 16	15 11	10 6	5 0
010000	00000	rt	ss	00000	100100
COP0	**MF**			**0**	**FPSHR**

ニーモニック:

RFPSH rt, ss

機能 :

FPR[rt] ← SHARE[ss]

例外 :

None

概要 :

共有レジスタから浮動小数点レジスタに値を読み込む.

WFPSH — Write Shared(Floating-Point Register)

同期命令 SYNC

31 26	25 21	20 16	15 11	10 6	5 0
010000	00100	rt	sd	00000	100100
COP0	**MT**			**0**	**FPSHR**

ニーモニック:

WFPSH rt, sd

機能 :

SHARE[sd] ← FPR[rt]

例外 :

None

概要 :

浮動小数点レジスタから共有レジスタに値を書き込む.

RGPEX Read Exclusive(General Purpose Register)

同期命令 SYNC

31 26	25 21	20 16	15 11	10 6	5 0
010000	00000	rt	ss	00000	100001
COP0	**MF**			**0**	**GPLOCK**

ニーモニック:

RGPEX rt, ss

機能 :

GPR[rt] ← SHARE[ss]

例外 :

None

概要 :

共有レジスタから汎用レジスタに値を読み込む.

WGPEX Write Exclusive(General Purpose Register)

同期命令 SYNC

31 26	25 21	20 16	15 11	10 6	5 0
010000	00100	rt	sd	00000	100001
COP0	**MT**			**0**	**GPLOCK**

ニーモニック:

WGPEX rt, sd

機能 :

SHARE[sd] ← GPR[rt]

例外 :

None

概要 :

汎用レジスタから共有レジスタに値を書き込む.

RFPEX　Read Exclusive(Floating-Point Register)

同期命令　SYNC

31 26	25 21	20 16	15 11	10 6	5 0
010000	00000	rt	ss	00000	100101
COP0	**MF**			**0**	**FPLOCK**

ニーモニック:

RFPEX rt, ss

機能 :

FPR[rt] ← SHARE[ss]

例外 :

None

概要 :

共有レジスタから浮動小数点レジスタに値を読み込む.

WFPEX　Write Exclusive(Floating-Point Register)

同期命令　SYNC

31 26	25 21	20 16	15 11	10 6	5 0
010000	00100	rt	sd	00000	100101
COP0	**MT**			**0**	**FPLOCK**

ニーモニック:

WFPEX rt, sd

機能 :

SHARE[sd] ← FPR[rt]

例外 :

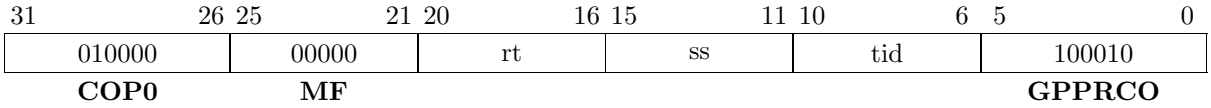
None

概要 :

浮動小数点レジスタから共有レジスタに値を書き込む.

GPCO Read Consumer(General Purpose Register)

同期命令 SYNC



ニーモニック:

GPCO rt, ss, tid

機能 :

GPR[rt] ← SHARE[ss]

例外 :

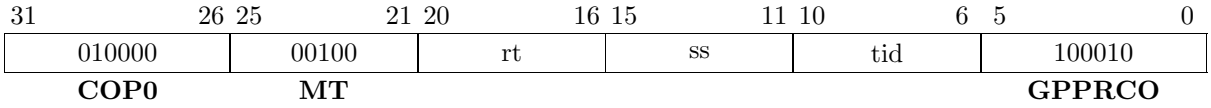
None

概要 :

共有レジスタから汎用レジスタに値を読み込む．tid にはスレッドの ID を指定する．

GPPR Write Producer(General Purpose Register)

同期命令 SYNC



ニーモニック:

GPPR rt, sd, tid

機能 :

SHARE[sd] ← GPR[rt]

例外 :

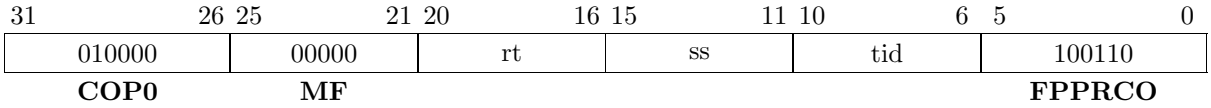
None

概要 :

汎用レジスタから共有レジスタに値を書き込む．tid にはスレッドの ID を指定する．

FPCO　Read Consumer(Floating-Point Register)

同期命令　SYNC



ニーモニック:

FPCO rt, ss, tid

機能 :

FPR[rt] ← SHARE[ss]

例外 :

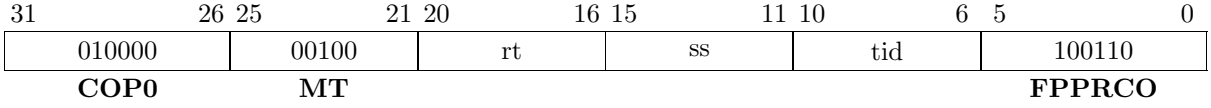
None

概要 :

共有レジスタから浮動小数点レジスタに値を読み込む．tid にはスレッドの ID を指定する．

FPPR　Write Producer(Floating-Point Register)

同期命令　SYNC



ニーモニック:

FPPR rt, sd, tid

機能 :

SHARE[sd] ← FPR[rt]

例外 :

None

概要 :

浮動小数点レジスタから共有レジスタに値を書き込む．tid にはスレッドの ID を指定する．

BAR

Barrier

同期命令

SYNC

31 26	25 21	20 16	15 11	10 6	5 0
010000	00100	rt	sd	00000	100011
COP0	**MT**			**0**	**BARRIER**

ニーモニック:

BAR rt, sd

機能 :

SHARE[sd] ← GPR[rt] + 1

例外 :

None

概要 :

—

PBAR

Pre Barrier

同期命令

SYNC

31 26	25 21	20 16	15 11	10 6	5 0
010000	000000	00000	sd	00000	100011
COP0	**MF**	**0**		**0**	**BARRIER**

ニーモニック:

PBAR sd

機能 :

—

例外 :

None

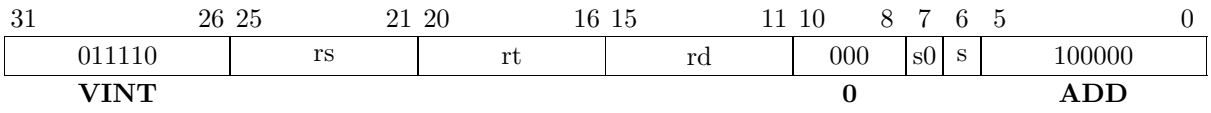
概要 :

—

3.3.8　整数ベクトル命令

VADD　Vector Add

ベクトル加算　VECTOR



ニーモニック:

VADD.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VADD.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VADD.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VADD.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能：

VGPR[rd] ← VGPR[rs] + VGPR[rt]

例外：

Vector Integer Exception :

概要：

ベクトル加算．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VSUB　　　　Vector Subtract

ベクトル減算　　　　VECTOR

31–26	25–21	20–16	15–11	10–9	8	7	6	5–0
011110	rs	rt	rd	00	s1	s0	s	100010
VINT				**0**				**SUB**

ニーモニック:

VSUB.vv rd, rs, rt　(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 0)
VSUB.vs rd, rs, rt　(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 0)
VSUB.sv rd, rs, rt　(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 0)
VSUB.vv.sy rd, rs, rt　(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 1)
VSUB.vs.sy rd, rs, rt　(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 1)
VSUB.sv.sy rd, rs, rt　(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 1)

機能：

VGPR[rd] ← VGPR[rs] − VGPR[rt]

例外：

Vector Integer Exception :

概要：

ベクトル減算．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sc1 の場合は VGPR[rs] の代わりに，スカラレジスタ (SGPR[rs]) を用いて演算を行う．sync により，投機実行を抑制する．

VMULT Vector Multiply

ベクトル乗算 VECTOR

31 26	25 21	20 16	15 11	10 8	7	6	5 0
011110	rs	rt	rd	000	s0	s	011000
VINT				**0**			**MULT**

ニーモニック:

VMULT.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VMULT.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VMULT.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VMULT.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] × VGPR[rt]

例外 :

Vector Integer Exception :

概要 :

符号付きベクトル乗算．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VMULTU Vector Multiply Unsigned

ベクトル乗算 VECTOR

31–26	25–21	20–16	15–11	10–8	7	6	5–0
011110	rs	rt	rd	000	s0	s	011001
VINT				**0**			**MULTU**

ニーモニック:

VMULTU.vv rd, rs, rt (scalar0(s0) = 0, sync(s) = 0)
VMULTU.vs rd, rs, rt (scalar0(s0) = 1, sync(s) = 0)
VMULTU.vv.sy rd, rs, rt (scalar0(s0) = 0, sync(s) = 1)
VMULTU.vs.sy rd, rs, rt (scalar0(s0) = 1, sync(s) = 1)

機能：

VGPR[rd] ← VGPR[rs] × VGPR[rt]

例外：

Vector Integer Exception :

概要：

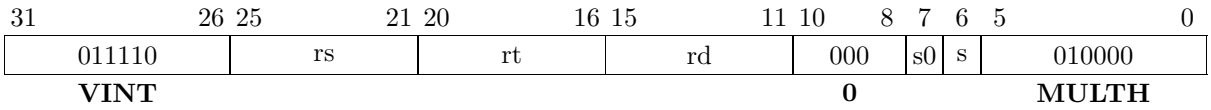
符号無しベクトル乗算．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VMULTH

Vector Multiply on High Bit

ベクトル乗算

VECTOR



ニーモニック:

VMULTH.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VMULTH.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VMULTH.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VMULTH.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] × VGPR[rt]

例外 :

Vector Integer Exception :

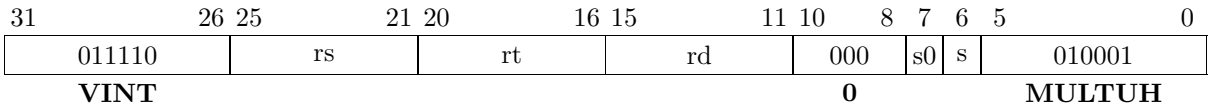
概要 :

符号付きベクトル乗算．演算結果の上位 bit(63-32bit) が VGPR[rd] に格納される．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VMULTUH

Vector Multiply Unsigned on High Bit

ベクトル乗算 VECTOR



ニーモニック:

VMULTUH.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VMULTUH.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VMULTUH.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VMULTUH.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] × VGPR[rt]

例外 :

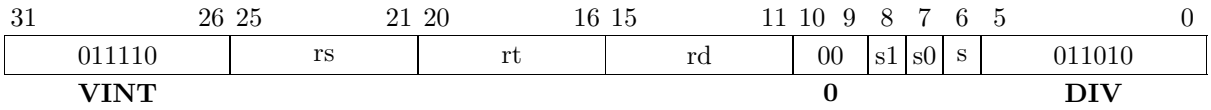
Vector Integer Exception :

概要 :

符号無しベクトル乗算．演算結果の上位 bit(63-32bit) が VGPR[rd] に格納される．sc0 が 1 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VDIV Vector Divide

ベクトル除算 VECTOR



ニーモニック:

VDIV.vv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 0)
VDIV.vs rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 0)
VDIV.sv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 0)
VDIV.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 1)
VDIV.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 1)
VDIV.sv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] ÷ VGPR[rt]

例外 :

Vector Integer Exception :

概要 :

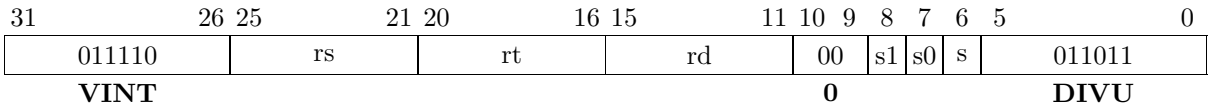
符号付きベクトル除算．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sc1 の場合は VGPR[rs] の代わりに，スカラレジスタ (SGPR[rs]) を用いて演算を行う．sync により，投機実行を抑制する．

VDIVU

Vector Divide Unsigned

ベクトル除算

VECTOR



ニーモニック:

VDIVU.vv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 0)
VDIVU.vs rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 0)
VDIVU.sv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 0)
VDIVU.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 1)
VDIVU.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 1)
VDIVU.sv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] ÷ VGPR[rt]

例外 :

Vector Integer Exception :

概要 :

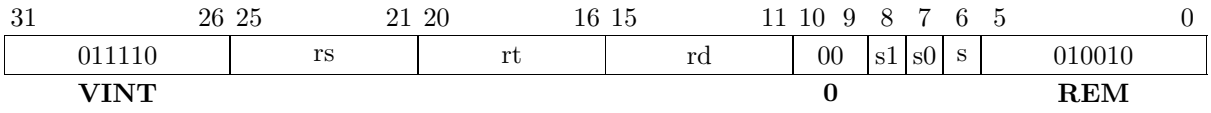
符号無しベクトル除算．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sc1 の場合は VGPR[rs] の代わりに，スカラレジスタ (SGPR[rs]) を用いて演算を行う．sync により，投機実行を抑制する．

VREM

Vector Reminder

ベクトル剰余

VECTOR



ニーモニック:

VREM.vv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 0)
VREM.vs rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 0)
VREM.sv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 0)
VREM.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 1)
VREM.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 1)
VREM.sv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] ÷ VGPR[rt]

例外 :

Vector Integer Exception :

概要 :

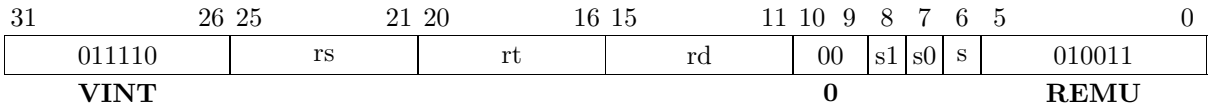
符号付きベクトル剰余．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sc1 の場合は VGPR[rs] の代わりに，スカラレジスタ (SGPR[rs]) を用いて演算を行う．sync により，投機実行を抑制する．

VREMU

Vector Reminder Unsigned

ベクトル剰余

VECTOR



ニーモニック:

VREMU.vv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 0)
VREMU.vs rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 0)
VREMU.sv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 0)
VREMU.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 1)
VREMU.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 1)
VREMU.sv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] ÷ VGPR[rt]

例外 :

Vector Integer Exception :

概要 :

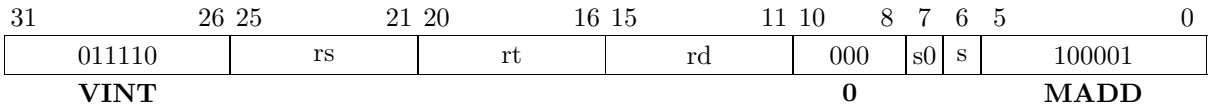
符号無しベクトル剰余．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sc1 の場合は VGPR[rs] の代わりに，スカラレジスタ (SGPR[rs]) を用いて演算を行う．sync により，投機実行を抑制する．

VMADD

Vector Multiply and Add

ベクトル積和演算

VECTOR



ニーモニック:

VMADD.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VMADD.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VMADD.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VMADD.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] × VGPR[rt] + VGPR[rd]

例外 :

Vector Integer Exception :

概要 :

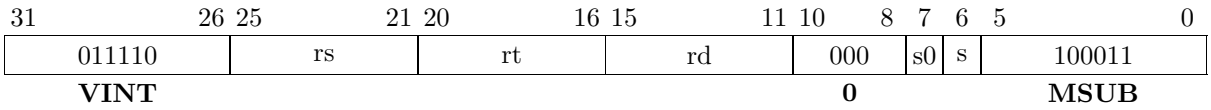
ベクトル積和演算．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VMSUB

Vector Multiply and Subtract

ベクトル積差演算

VECTOR



ニーモニック:

VMSUB.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VMSUB.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VMSUB.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VMSUB.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] × VGPR[rt] − VGPR[rd]

例外 :

Vector Integer Exception :

概要 :

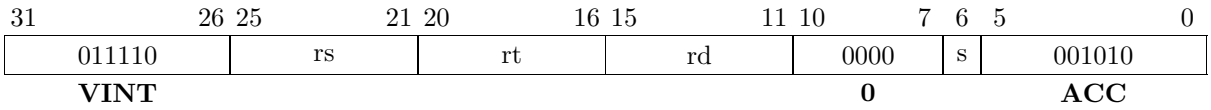
ベクトル積差演算．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VACC

Vector Accumulate

ベクトル累算

VECTOR



ニーモニック:

VACC rd, rs　(sync(s) = 0)

VACC.sy rd, rs　(sync(s) = 1)

機能 :

SGPR[rd] ← $\sum$ VGPR[rs]

例外 :

Vector Integer Exception :

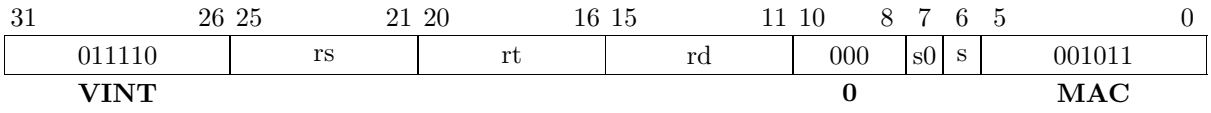
概要 :

ベクトル累算．ベクトルの要素を全て加算する．sync により，投機実行を抑制する．

VMAC

Vector Multiply and Accumulate

ベクトル積和演算 VECTOR



ニーモニック:

VMAC.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VMAC.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VMAC.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VMAC.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

$\mathrm{SGPR[rd]} \leftarrow \sum \mathrm{VGPR[rs]} \times \mathrm{VGPR[rt]}$

例外 :

Vector Integer Exception :

概要 :

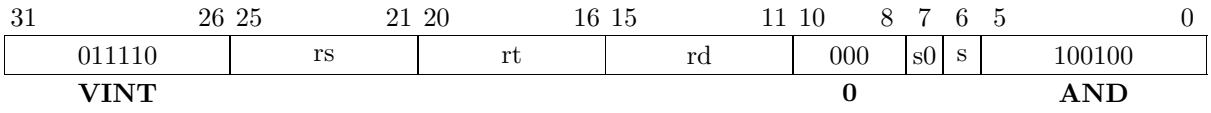
ベクトル積和演算．2 つのベクトル要素を乗算し，それを全て加算する．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VAND

Vector And

ベクトル論理積

VECTOR



ニーモニック:

VAND.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VAND.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VAND.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VAND.vs.sync rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] and VGPR[rt]

例外 :

Vector Integer Exception :

概要 :

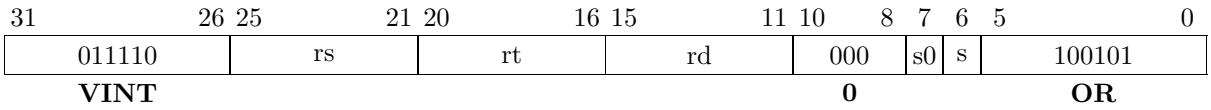
ベクトル論理積．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VOR

Vector Or

ベクトル論理和

VECTOR



ニーモニック:

VOR.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VOR.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VOR.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VOR.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] or VGPR[rt]

例外 :

Vector Integer Exception :

概要 :

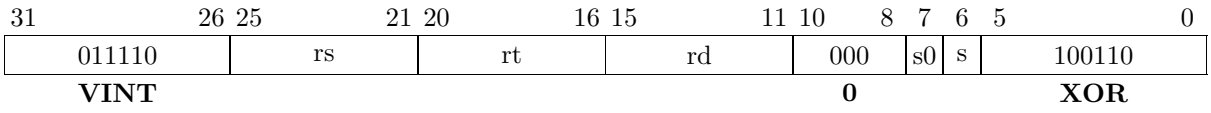
ベクトル論理和．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VXOR

Vector Exclusive Or

ベクトル排他的論理和

VECTOR



ニーモニック:

VXOR.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VXOR.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VXOR.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VXOR.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] xor VGPR[rt]

例外 :

Vector Integer Exception :

概要 :

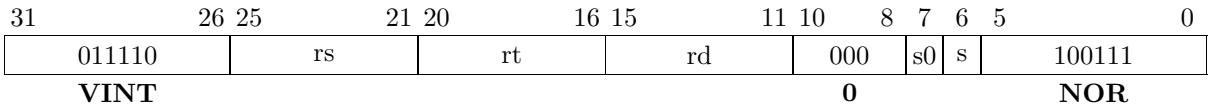
ベクトル排他的論理和．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VNOR

Vector Not Or

ベクトル否定論理和

VECTOR



ニーモニック:

VNOR.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VNOR.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VNOR.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VNOR.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] nor VGPR[rt]

例外 :

Vector Integer Exception :

概要 :

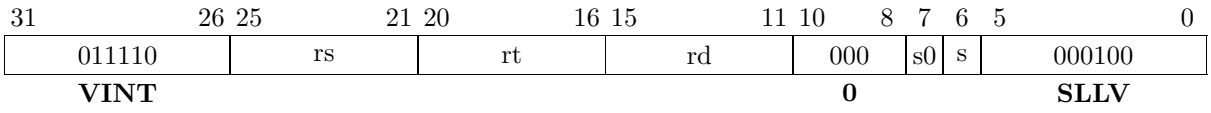
ベクトル否定論理和．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VSLLV

Vector Shift Left Logical Variable

ベクトル左論理シフト

VECTOR



ニーモニック:

VSLLV.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VSLLV.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VSLLV.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VSLLV.vs.sync rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rt] ≪ VGPR[rs]

例外 :

Vector Integer Exception :

概要 :

ベクトル左論理シフト．sc0 の場合は VGPR[rs] の代わりに，スカラレジスタ (SGPR[rs]) を用いて演算を行う．sync により，投機実行を抑制する．

VSRLV

Vector Shift Right Logical Variable

ベクトル右論理シフト

VECTOR

31 26	25 21	20 16	15 11	10 8	7	6	5 0
011110	rs	rt	rd	000	s0	s	000110
VINT				**0**			**SRLV**

ニーモニック:

VSRLV.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VSRLV.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VSRLV.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VSRLV.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rt] ≫ VGPR[rs]

例外 :

Vector Integer Exception :

概要 :

ベクトル右論理シフト．sc01 の場合は VGPR[rs] の代わりに，スカラレジスタ (SGPR[rs]) を用いて演算を行う．sync により，投機実行を抑制する．

VSRAV

Vector Shift Right Arithmetic Variable

ベクトル右算術シフト

VECTOR

31 26	25 21	20 16	15 11	10 8	7	6	5 0
011110	rs	rt	rd	000	s0	s	000111
VINT				**0**			**SRAV**

ニーモニック:

VSRAV.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VSRAV.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VSRAV.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VSRAV.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rt] ≫ VGPR[rs]

例外 :

Vector Integer Exception :

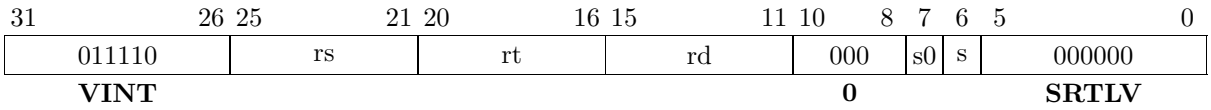
概要 :

ベクトル右算術シフト．sc0 の場合は VGPR[rs] の代わりに，スカラレジスタ (SGPR[rs]) を用いて演算を行う．sync により，投機実行を抑制する．

VRTLV

Vector Rotate Left Variable

ベクトル左ローテーション VECTOR



ニーモニック:

VRTLV.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VRTLV.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VRTLV.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VRTLV.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rt] <<< VGPR[rs]

例外 :

Vector Integer Exception :

概要 :

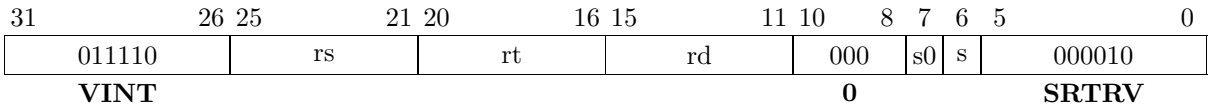
ベクトル左ローテーション．sc0 の場合は VGPR[rs] の代わりに，スカラレジスタ (SGPR[rs]) を用いて演算を行う．sync により，投機実行を抑制する．

VRTRV

Vector Rotate Right Variable

ベクトル右ローテーション

VECTOR



ニーモニック:

VRTRV.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VRTRV.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VRTRV.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VRTRV.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rt] >>> VGPR[rs]

例外 :

Vector Integer Exception :

概要 :

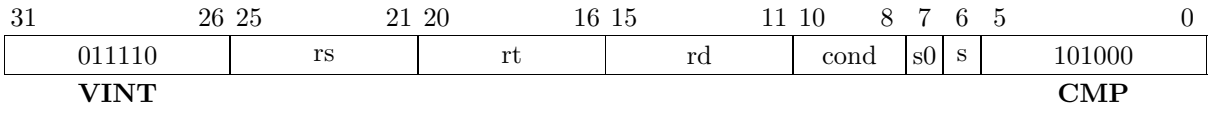
ベクトル右ローテーション．sc0 の場合は VGPR[rs] の代わりに，スカラレジスタ (SGPR[rs]) を用いて演算を行う．sync により，投機実行を抑制する．

VCMP

Vector Compare

ベクトル比較

VECTOR



ニーモニック:

VCMP.cond.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMP.cond.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMP.cond.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMP.cond.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← (VGPR[rs] cond VGPR[rt])

例外 :

Vector Integer Exception :

概要 :

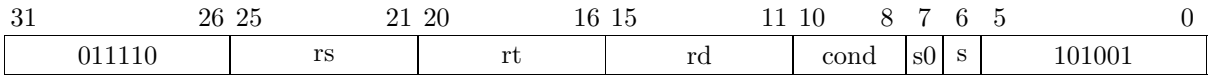
ベクトル比較命令．条件 (cond) により VGPR[rd] に 1 または 0 が入る．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VCMPU

Vector Compare Unsigned

ベクトル比較

VECTOR



ニーモニック:

VCMPU.cond.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMPU.cond.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMPU.cond.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMPU.cond.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← (VGPR[rs] cond VGPR[rt])

例外 :

Vector Integer Exception :

概要 :

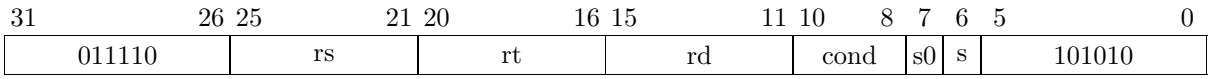
符号無しベクトル比較命令．条件 (cond) により VGPR[rd] に 1 または 0 が入る．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VCMPTS

Vector Compare to Scalar Register

ベクトル比較

VECTOR



VINT **CMPTS**

ニーモニック:

VCMPTS.cond.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMPTS.cond.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMPTS.cond.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMPTS.cond.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

SGPR[rd] ← (VGPR[rs] cond VGPR[rt])

例外 :

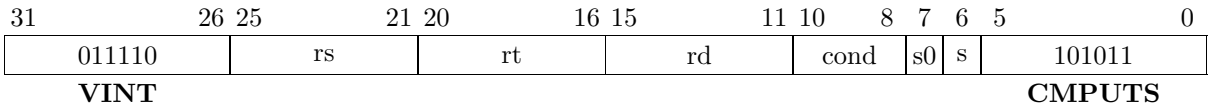
Vector Integer Exception :

概要 :

ベクトル比較命令．結果は各要素ごとに 1bit を割り当ててスカラレジスタに格納される．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VCMPUTS Vector Compare Unsigned to Scalar Register

ベクトル比較 VECTOR



ニーモニック:

VCMPUTS.cond.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMPUTS.cond.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMPUTS.cond.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMPUTS.cond.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

SGPR[rd] ← (VGPR[rs] cond VGPR[rt])

例外 :

Vector Integer Exception :

概要 :

符号無しベクトル比較命令．結果は各要素ごとに 1bit を割り当ててスカラレジスタに格納される．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VIMFC

Move from Vector Integer Control Register

制御レジスタリード VECTOR

31 26	25 21	20 16	15 11	10 7	6	5 0
011110	rs	00000	rd	0000	s	110000
VINT		**0**		**0**		**MFC**

ニーモニック:

VIMFC rd, rs (sync(s) = 0)

VIMFC.sy rd, rs (sync(s) = 1)

機能 :

GPR[rd] ← VICTRL[rs]

例外 :

Vector Integer Exception :

概要 :

整数ベクトル制御レジスタリード命令．rs で指定された制御レジスタの値を汎用レジスタに格納する．sync により，投機実行を抑制する．

VIMTC

Move to Vector Integer Control Register

制御レジスタライト　　VECTOR

31 26	25 21	20 16	15 11	10 7	6	5 0
011110	rs	00000	rd	0000	s	110001
VINT		**0**		**0**		**MTC**

ニーモニック:

VIMTC rd, rs　　(sync(s) = 0)

VIMTC.sy rd, rs　　(sync(s) = 1)

機能 :

VICTRL[rd] ← GPR[rs]

例外 :

Vector Integer Exception :

概要 :

整数ベクトル制御レジスタライト命令．rd で指定された制御レジスタに汎用レジスタの値を格納する．sync により，投機実行を抑制する．

VIMFS

Move from Vector Integer Scalar Register

整数スカラレジスタリード VECTOR

31 26	25 21	20 16	15 11	10 7	6	5 0
011110	rs	00000	rd	0000	s	110010
VINT		**0**		**0**		**MFS**

ニーモニック:

VIMFS rd, rs (sync(s) = 0)

VIMFS.sy rd, rs (sync(s) = 1)

機能 :

GPR[rd] ← SGPR[rs]

例外 :

Vector Integer Exception :

概要 :

整数スカラレジスタリード命令．rs で指定された整数スカラレジスタの値を汎用レジスタに格納する．sync により，投機実行を抑制する．

VIMTS

Move to Vector Integer Scalar Register

整数スカラレジスタライト　　VECTOR

31 26	25 21	20 16	15 11	10 7	6	5 0
011110	rs	00000	rd	0000	s	110011
VINT		**0**		**0**		**MTS**

ニーモニック:

VIMTS rd, rs　　(sync(s) = 0)

VIMTS.sy rd, rs　　(sync(s) = 1)

機能 :

SGPR[rd] ← GPR[rs]

例外 :

Vector Integer Exception :

概要 :

整数スカラレジスタライト命令．rd で指定された整数スカラレジスタに汎用レジスタの値を格納する．sync により，投機実行を抑制する．

VIMFV

Move from Vector Integer Vector Register

整数ベクトルレジスタリード VECTOR

31 26	25 21	20 16	15 11	10 7	6	5 0
011110	rs	rt	rd	0000	s	110100
VINT				**0**		**MFV**

ニーモニック:

VIMFV rd, rs, rt (sync(s) = 0)

VIMFV.sy rd, rs, rt (sync(s) = 1)

機能 :

SGPR[rd] ← VGPR[rs][rt]

例外 :

Vector Integer Exception :

概要 :

整数ベクトルレジスタリード命令．rs で指定された整数ベクトルレジスタの rt 番目の要素の値を整数スカラレジスタに格納する．sync により，投機実行を抑制する．

VIMTV

Move to Vector Integer Vector Register

整数ベクトルレジスタライト　　VECTOR

31 26	25 21	20 16	15 11	10 7	6	5 0
011110	rs	rt	rd	0000	s	110101
VINT				**0**		**MTV**

ニーモニック:

VIMTV rd, rs, rt　　(sync(s) = 0)

VIMTV.sy rd, rs, rt　　(sync(s) = 1)

機能 :

VGPR[rd][rt] ← SGPR[rs]

例外 :

Vector Integer Exception :

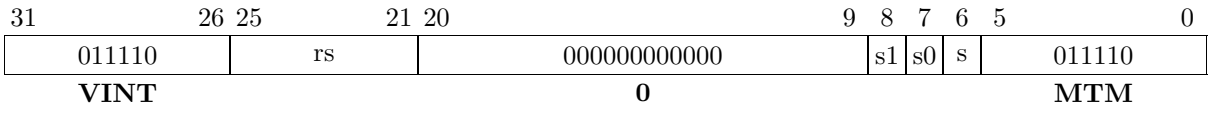
概要 :

整数ベクトルレジスタリード命令．rd で指定された整数ベクトルレジスタの rt 番目の要素に整数スカラレジスタの値を書き込む．sync により，投機実行を抑制する．

VIMTM

Move to Vector Integer Mask Register

整数ベクトルマスクレジスタライト VECTOR



ニーモニック:

VIMTM.lo rs	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 0)
VIMTM.hi rs	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 0)
VIMTM.lo.sy rs	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 1)
VIMTM.hi.sy rs	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 1)

機能 :

VICTRL[*Mask Regieter*] ← SGPR[rs]

例外 :

Vector Integer Exception :

概要 :

整数ベクトルマスクレジスタライト命令．rs で指定した整数スカラレジスタの値をマスクレジスタに格納する．sc0 の場合，マスクレジスタの下位 32bit に値を格納し，sc1 の場合，マスクレジスタの上位 32bit に値を格納する．sync により，投機実行を抑制する．

VIRSV

Vector Integer Register Reserve

整数ベクトルレジスタ予約

VECTOR

31 26	25 21	20 16	15 11	10 7	6	5 0
011110	rs	00000	rd	0000	s	110110
VINT		**0**		**0**		**RSV**

ニーモニック:

VIRSV rd, rs　(sync(s) = 0)

VIRSV.sy rd, rs　(sync(s) = 1)

機能：

GPR[rd] ← if(成功) then 1 else 0

例外：

None

概要：

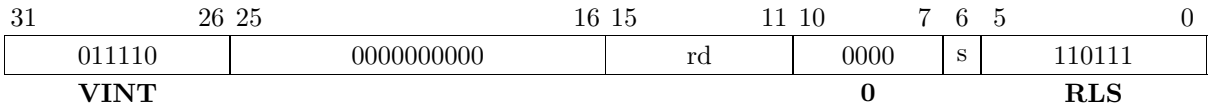
整数ベクトルレジスタ予約命令．GPR[rs] に予約するレジスタの構成を指定する．予約に成功した場合は GPR[rd] に 1 が，失敗した場合は 0 が格納される．sync により，投機実行を抑制する．

VIRLS

Vector Integer Register Release

整数ベクトルレジスタ開放

VECTOR



ニーモニック:

VIRLS rd	(sync(s) = 0)
VIRLS.sy rd	(sync(s) = 1)

機能 :

GPR[rd] ← if(成功) then 1 else 0

例外 :

None

概要 :

整数ベクトルレジスタ開放命令．開放に成功した場合は GPR[rd] に 1 が，失敗した場合は 0 が格納される．sync により，投機実行を抑制する．

VIDCI　Vector Integer Define Compound Instruction

整数ベクトル複合命令定義　VECTOR

31 26	25 21	20 16	15 11	10 7	6	5 0
011110	rs	00000	rd	0000	s	101110
VINT		**0**		**0**		**DCI**

ニーモニック:

VIDCI rd, rs　(sync(s) = 0)

VIDCI.sy rd, rs　(sync(s) = 1)

機能 :

VICPD[rd] ← GPR[rs]

例外 :

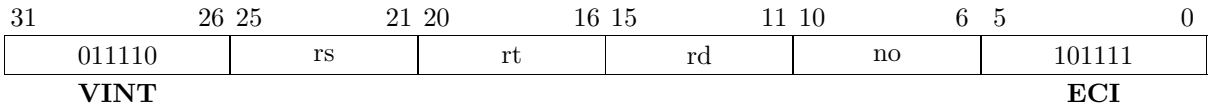
Vector Integer Exception :

概要 :

整数ベクトル複合命令の定義を行う．GPR[rs] で定義した命令を rd で指定した複合命令バッファのアドレスに格納する．sync により，投機実行を抑制する．

VIECI Vector Integer Execute Compound Instruction

整数ベクトル複合命令実行 VECTOR



ニーモニック:

VIECI rd, rs, rt, no

機能 :

VGPR[rd] ← VGPR[rs] op VGPR[rt]

例外 :

Vector Integer Exception :

概要 :

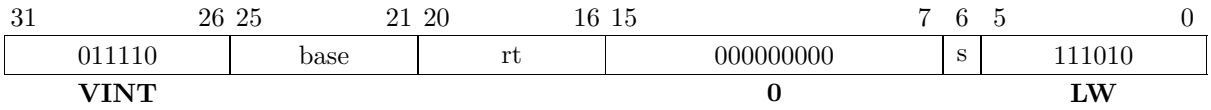
整数ベクトル複合命令の実行を行う．no で指定した複合命令バッファのアドレスから命令を実行する．

VILW

Vector Integer Load Word

整数ベクトルロード

VECTOR



ニーモニック:

VILW rt, base (sync(s) = 0)

VILW.sy rt, base (sync(s) = 1)

機能 :

VGPR[rt] ← MEM[GPR[base]]

例外 :

Vector Integer Exception :
D-TLB No Entry Matched :
D-TLB Protection Error :
Data Address Miss Align (Load) :

概要 :

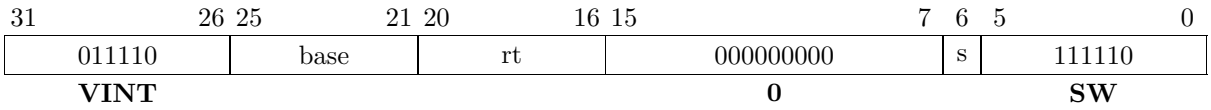
メモリから整数ベクトルレジスタにロードする．sync により，投機実行を抑制する．

VISW

Vector Integer Store Word

整数ベクトルストア

VECTOR



ニーモニック:

VISW rt, base (sync(s) = 0)

VISW.sy rt, base (sync(s) = 1)

機能 :

MEM[GPR[base]] ← VGPR[rt]

例外 :

Vector Integer Exception :

D-TLB No Entry Matched :

D-TLB Protection Error :

Data Address Miss Align (Store) :

概要 :

整数ベクトルレジスタをメモリにストアする．sync により，投機実行を抑制する．

VADD.QB Vector Add Quad Byte

ベクトル加算 VECTOR

31 26	25 21	20 16	15 11	10	9	8	7	6	5 0
111110	rs	rt	rd	0	s2	0	s0	s	100000
VINT.QB				**0**		**0**			**ADD**

ニーモニック:

VADD.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VADD.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VADD.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VADD.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VADD.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VADD.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VADD.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VADD.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] + VGPR[rt]

例外 :

Vector Integer Exception :

概要 :

8 bit × 4 ベクトル加算．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sc2 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う．sync により，投機実行を抑制する．

VSUB.QB

Vector Subtract Quad Byte

ベクトル減算

VECTOR

31 26	25 21	20 16	15 11	10	9	8	7	6	5 0
111110	rs	rt	rd	0	s2	s1	s0	s	100010
VINT.QB				**0**					**SUB**

ニーモニック:

VSUB.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) =0, scalar2(s2) = 0, sync(s) = 0)
VSUB.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) =0, scalar2(s2) = 0, sync(s) = 0)
VSUB.QB.sv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) =1, scalar2(s2) = 0, sync(s) = 0)
VSUB.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) =0, scalar2(s2) = 1, sync(s) = 0)
VSUB.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) =0, scalar2(s2) = 0, sync(s) = 1)
VSUB.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) =0, scalar2(s2) = 1, sync(s) = 0)
VSUB.QB.sv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) =1, scalar2(s2) = 1, sync(s) = 0)
VSUB.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) =0, scalar2(s2) = 0, sync(s) = 1)
VSUB.QB.sv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) =1, scalar2(s2) = 0, sync(s) = 1)
VSUB.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) =0, scalar2(s2) = 1, sync(s) = 1)
VSUB.QB.sv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) =1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] - VGPR[rt]

例外 :

Vector Integer Exception :

概要 :

8 bit × 4 ベクトル減算．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sc1 の場合は VGPR[rs] の代わりに，スカラレジスタ (SGPR[rs]) を用いて演算を行う．sc2 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う．sync により，投機実行を抑制する．

VMULT.QB Vector Multiply Quad Byte

ベクトル乗算 VECTOR

31–26	25–21	20–16	15–11	10	9–8	7	6	5–0
111110	rs	rt	rd	0	s2	0	s0	s	011000
VINT.QB				**0**		**0**		**MULT**

ニーモニック:

VMULT.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMULT.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMULT.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMULT.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMULT.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMULT.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMULT.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMULT.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] × VGPR[rt]

例外 :

Vector Integer Exception :

概要 :

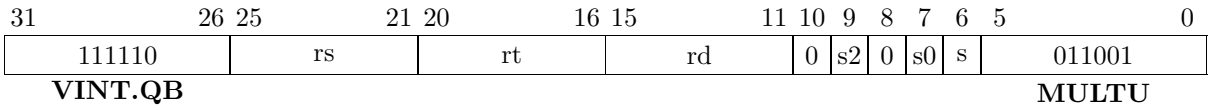
8 bit × 4 符号付きベクトル乗算. sc0 の場合は VGPR[rt] の代わりに, スカラレジスタ (SGPR[rt]) を用いて演算を行う. sc2 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う. sync を 1 にすることにより, 投機実行を抑制する.

VMULTU.QB

Vector Multiply Unsigned

ベクトル乗算

VECTOR



ニーモニック:

VMULTU.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMULTU.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMULTU.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMULTU.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMULTU.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMULTU.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMULTU.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMULTU.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] × VGPR[rt]

例外 :

Vector Integer Exception :

概要 :

8 bit × 4 符号無しベクトル乗算. sc0 の場合は VGPR[rt] の代わりに, スカラレジスタ (SGPR[rt]) を用いて演算を行う. sc2 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う. sync により, 投機実行を抑制する.

VMULTH.QB Vector Multiply Quad Byte on High Bit

ベクトル乗算 VECTOR

31 26	25 21	20 16	15 11	10	9	8	7	6	5 0
111110	rs	rt	rd	0	s2	0	s0	s	010000
VINT.QB				**0**		**0**			**MULTH**

ニーモニック:

VMULTH.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMULTH.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMULTH.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMULTH.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMULTH.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMULTH.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMULTH.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMULTH.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] × VGPR[rt]

例外 :

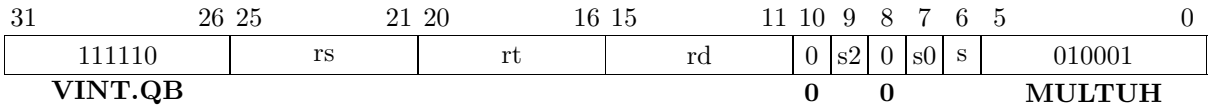
Vector Integer Exception :

概要 :

8 bit × 2 符号付きベクトル乗算．8 bit 毎に演算結果の上位 bit(16-8bit) が VGPR[rd] に格納される．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sc2 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う．sync により，投機実行を抑制する．

VMULTUH.QB Vector Multiply Unsigned Quad Byte on High Bit

ベクトル乗算 VECTOR



ニーモニック:

VMULTUH.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMULTUH.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMULTUH.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMULTUH.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMULTUH.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMULTUH.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMULTUH.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMULTUH.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] × VGPR[rt]

例外 :

Vector Integer Exception :

概要 :

8 bit × 4 符号無しベクトル乗算．8 bit 毎に演算結果の上位 bit(16-8bit) が VGPR[rd] に格納される．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sc2 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う．sync により，投機実行を抑制する．

VMADD.QB Vector Multiply and Add Quad Byte

ベクトル積和演算 VECTOR

31 26	25 21	20 16	15 11	10	9	8	7	6	5 0
111110	rs	rt	rd	0	s2	0	s0	s	100001
VINT.QB				**0**		**0**			**MADD**

ニーモニック:

VMADD.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMADD.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMADD.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMADD.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMADD.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMADD.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMADD.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMADD.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] × VGPR[rt] + VGPR[rd]

例外 :

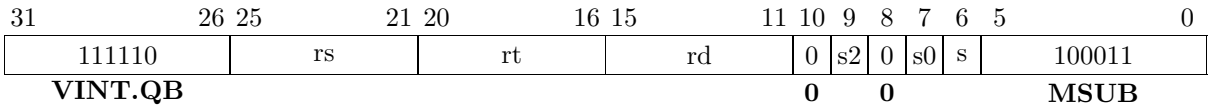
Vector Integer Exception :

概要 :

8 bit × 4 ベクトル積和演算．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sc2 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う．sync により，投機実行を抑制する．

VMSUB.QB Vector Multiply and Subtract Quad Byte

ベクトル積差演算 VECTOR



ニーモニック:

VMSUB.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMSUB.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMSUB.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMSUB.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMSUB.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMSUB.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMSUB.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMSUB.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] × VGPR[rt] - VGPR[rd]

例外 :

Vector Integer Exception :

概要 :

8 bit × 4 ベクトル積差演算．sc の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sc2 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う．sync により，投機実行を抑制する．

VACC.QB

Vector Accumulate Quad Byte

ベクトル累算

VECTOR

31 26	25 21	20 16	15 11	10 6	5 0
111110	rs	rt	rd	00000	001010
VINT.QB				**0**	**ACC**

ニーモニック:

VACC.QB rd, rs (sync(s) = 0)

VACC.QB.sy rd, rs (sync(s) = 1)

機能 :

SGPR[rd] ← ∑VGPR[rs]

例外 :

Vector Integer Exception :

概要 :

8 bit × 4 ベクトル累算．ベクトルの要素を全て加算する．sync により，投機実行を抑制する．

VMAC.QB Vector Multiply and Accumulate Quad Byte

ベクトル積和演算 VECTOR

31 26	25 21	20 16	15 11	10	9	8	7	6	5 0
111110	rs	rt	rd	0	s2	0	s0	s	001011
VINT.QB				**0**		**0**			**MAC**

ニーモニック:

VMAC.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMAC.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMAC.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMAC.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMAC.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMAC.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMAC.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMAC.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

SGPR[rd] $\leftarrow \sum$VGPR[rs] $\times$ VGPR[rt]

例外 :

Vector Integer Exception :

概要 :

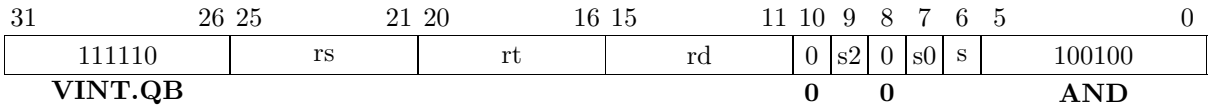
8 bit $\times$ 4 ベクトル積和演算．2 つのベクトル要素を乗算し，それを全て加算する．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sc2 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う．sync により，投機実行を抑制する．

VAND.QB

Vector And Quad Byte

ベクトル論理積

VECTOR



ニーモニック:

VAND.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VAND.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VAND.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VAND.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VAND.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VAND.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VAND.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VAND.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] and VGPR[rt]

例外 :

Vector Integer Exception :

概要 :

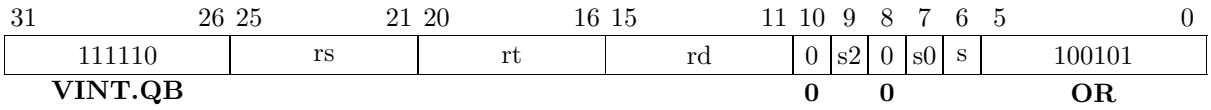
8 bit × 4 ベクトル論理積．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sc2 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う．sync により，投機実行を抑制する．

VOR.QB

Vector Or

ベクトル論理和

VECTOR



ニーモニック:

VOR.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VOR.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VOR.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VOR.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VOR.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VOR.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VOR.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VOR.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] or VGPR[rt]

例外 :

Vector Integer Exception :

概要 :

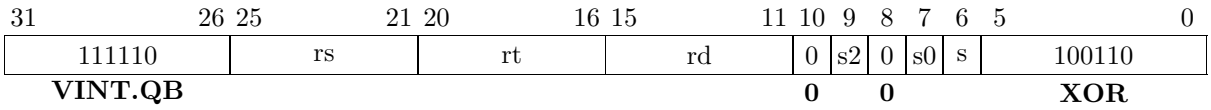
8 bit × 4 ベクトル論理和．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sc2 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う．sync により，投機実行を抑制する．

VXOR.QB

Vector Exclusive Or Quad Byte

ベクトル排他的論理和

VECTOR



ニーモニック:

VXOR.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VXOR.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VXOR.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VXOR.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VXOR.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VXOR.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VXOR.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VXOR.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能：

VGPR[rd] ← VGPR[rs] xor VGPR[rt]

例外：

Vector Integer Exception :

概要：

8 bit × 4 ベクトル排他的論理和. sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sc2 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う．sync により，投機実行を抑制する．

VNOR.QB

Vector Not Or Paried HalfWord

ベクトル否定論理和

VECTOR

31 26	25 21	20 16	15 11	10	9	8	7	6	5 0
111110	rs	rt	rd	0	s2	0	s0	s	100111
VINT.QB				**0**		**0**			**NOR**

ニーモニック:

VNOR.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VNOR.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VNOR.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VNOR.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VNOR.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VNOR.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VNOR.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VNOR.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] nor VGPR[rt]

例外 :

Vector Integer Exception :

概要 :

8 bit × 4 ベクトル否定論理和. sc0 の場合は VGPR[rt] の代わりに, スカラレジスタ (SGPR[rt]) を用いて演算を行う. sc2 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う. sync により, 投機実行を抑制する.

VSLLV.QB Vector Shift Left Logical Variable Quad Byte

ベクトル左論理シフト VECTOR

31–26	25–21	20–16	15–11	10	9–8	7	6	5–0
111110	rs	rt	rd	0	s2	0	s0	s	000100
VINT.QB				**0**		**0**		**SLLV**

ニーモニック:

VSLLV.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VSLLV.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VSLLV.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VSLLV.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VSLLV.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VSLLV.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VSLLV.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VSLLV.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rt] ≪ VGPR[rs]

例外 :

Vector Integer Exception :

概要 :

8 bit × 4 ベクトル左論理シフト. sc0 の場合は VGPR[rs] の代わりに, スカラレジスタ (SGPR[rs]) を用いて演算を行う. sc2 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う. sync により, 投機実行を抑制する.

VSRLV.QB

Vector Shift Right Logical Variable Quad Byte

ベクトル右論理シフト VECTOR

31 26	25 21	20 16	15 11	10	9	8	7	6	5 0
111110	rs	rt	rd	0	s2	0	s0	s	000110
VINT.QB				**0**		**0**			**SRLV**

ニーモニック:

VSRLV.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VSRLV.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VSRLV.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VSRLV.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VSRLV.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VSRLV.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VSRLV.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VSRLV.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rt] ≫ VGPR[rs]

例外 :

Vector Integer Exception :

概要 :

8 bit × 4 ベクトル右論理シフト. sc0 の場合は VGPR[rs] の代わりに, スカラレジスタ (SGPR[rs]) を用いて演算を行う. sc2 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う. sync により, 投機実行を抑制する.

VSRAV.QB Vector Shift Right Arithmetic Variable Quad Byte

ベクトル右算術シフト VECTOR

31 26	25 21	20 16	15 11	10	9	8	7	6	5 0
111110	rs	rt	rd	0	s2	0	s0	s	000111
VINT.QB				**0**		**0**			**SRAV**

ニーモニック:

VSRAV.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VSRAV.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VSRAV.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VSRAV.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VSRAV.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VSRAV.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VSRAV.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VSRAV.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rt] ≫ VGPR[rs]

例外 :

Vector Integer Exception :

概要 :

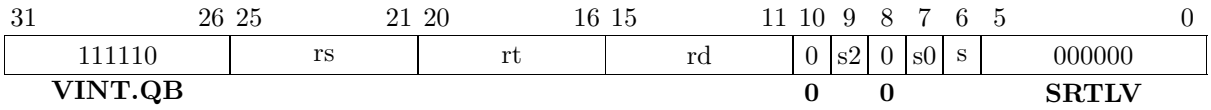
ベクトル右算術シフト．sc0 の場合は VGPR[rs] の代わりに，スカラレジスタ (SGPR[rs]) を用いて演算を行う．sc2 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う．sync により，投機実行を抑制する．

VRTLV.QB

Vector Rotate Left Variable Quad Byte

ベクトル左ローテーション

VECTOR



ニーモニック:

VRTLV.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VRTLV.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VRTLV.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VRTLV.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VRTLV.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VRTLV.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VRTLV.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VRTLV.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能：

VGPR[rd] ← VGPR[rt] <<< VGPR[rs]

例外：

Vector Integer Exception :

概要：

ベクトル左ローテーション．sc0 の場合は VGPR[rs] の代わりに，スカラレジスタ (SGPR[rs]) を用いて演算を行う．sc2 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う．sync により，投機実行を抑制する．

VRTRV.QB Vector Rotate Right Variable Quad Byte

ベクトル右ローテーション VECTOR

31 26	25 21	20 16	15 11	10	9	8	7	6	5 0
111110	rs	rt	rd	0	s2	0	s0	s	000010
VINT.QB				**0**		**0**			**SRTRV**

ニーモニック:

VRTRV.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VRTRV.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VRTRV.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VRTRV.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VRTRV.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VRTRV.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VRTRV.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VRTRV.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rt] >>> VGPR[rs]

例外 :

Vector Integer Exception :

概要 :

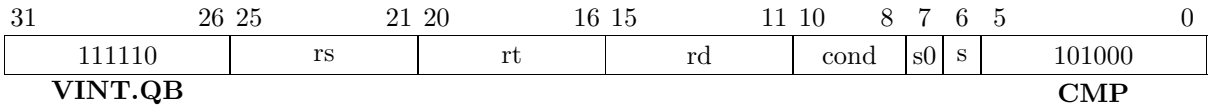
8 bit × 4 ベクトル右ローテーション．sc0 の場合は VGPR[rs] の代わりに，スカラレジスタ (SGPR[rs]) を用いて演算を行う．sc2 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う．sync により，投機実行を抑制する．

VCMP.QB

Vector Compare

ベクトル比較

VECTOR



ニーモニック:

VCMP.cond.QB.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMP.cond.QB.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMP.cond.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMP.cond.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← (VGPR[rs] cond VGPR[rt])

例外 :

Vector Integer Exception :

概要 :

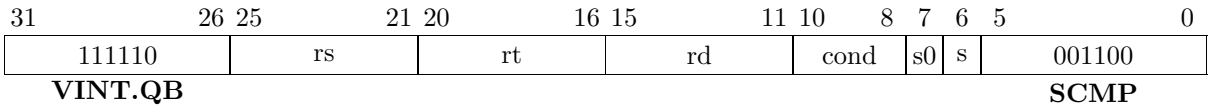
8 bit × 4 ベクトル比較命令．条件 (cond) により VGPR[rd] に 1 または 0 が入る．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VSCMP.QB

Vector Compare

ベクトル比較

VECTOR



ニーモニック:

VSCMP.cond.QB.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VSCMP.cond.QB.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VSCMP.cond.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VSCMP.cond.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← (VGPR[rs] cond VGPR[rt])

例外 :

Vector Integer Exception :

概要 :

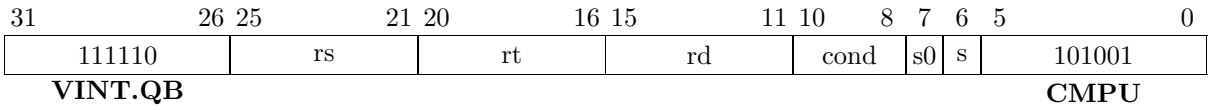
8 bit × 4 ベクトル比較命令．VGPR[rt] の下位 8 bit を用いて演算を行う．条件 (cond) により VGPR[rd] に 1 または 0 が入る．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VCMPU.QB

Vector Compare Unsigned Quad Byte

ベクトル比較

VECTOR



ニーモニック:

VCMPU.cond.QB.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMPU.cond.QB.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMPU.cond.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMPU.cond.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← (VGPR[rs] cond VGPR[rt])

例外 :

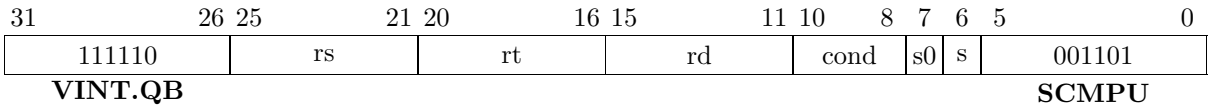
Vector Integer Exception :

概要 :

8 bit × 4 符号無しベクトル比較命令．条件 (cond) により VGPR[rd] に 1 または 0 が入る．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VSCMPU.QB Vector Compare Unsigned Quad Byte

ベクトル比較 VECTOR



ニーモニック:

VSCMPU.cond.QB.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VSCMPU.cond.QB.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VSCMPU.cond.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VSCMPU.cond.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← (VGPR[rs] cond VGPR[rt])

例外 :

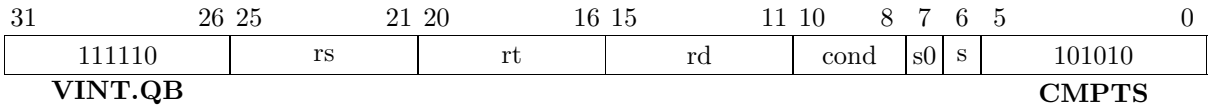
Vector Integer Exception :

概要 :

8 bit × 4 符号無しベクトル比較命令．VGPR[rt] の下位 8 bit を用いて演算を行う．条件 (cond) により VGPR[rd] に 1 または 0 が入る．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VCMPTS.QB Vector Compare to Scalar Register Quad Byte

ベクトル比較 VECTOR



ニーモニック:

VCMPTS.cond.QB.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMPTS.cond.QB.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMPTS.cond.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMPTS.cond.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

SGPR[rd] ← (VGPR[rs] cond VGPR[rt])

例外 :

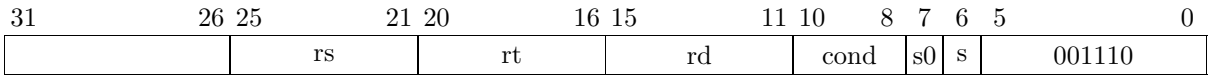
Vector Integer Exception :

概要 :

8 bit × 4 ベクトル比較命令．結果は各要素ごとに 1bit を割り当ててスカラレジスタに格納される．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VSCMPTS.QB Vector Compare Quad Byte to Scalar Register

ベクトル比較 VECTOR



VINT.QB **SCMPTS**

ニーモニック:

VSCMPTS.cond.QB.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VSCMPTS.cond.QB.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VSCMPTS.cond.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VSCMPTS.cond.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

SGPR[rd] ← (VGPR[rs] cond VGPR[rt])

例外 :

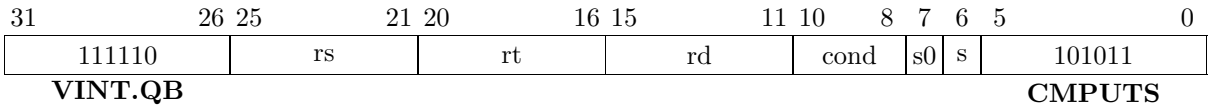
Vector Integer Exception :

概要 :

8 bit × 4 ベクトル比較命令．VGPR[rt] の下位 8 bit を用いて演算を行う．結果は各要素ごとに 1bit を割り当ててスカラレジスタに格納される．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VCMPUTS.QB Vector Compare Unsigned Quad Byte to Scalar Register

ベクトル比較 VECTOR



ニーモニック:

VCMPUTS.cond.QB.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMPUTS.cond.QB.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMPUTS.cond.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMPUTS.cond.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

SGPR[rd] ← (VGPR[rs] cond VGPR[rt])

例外 :

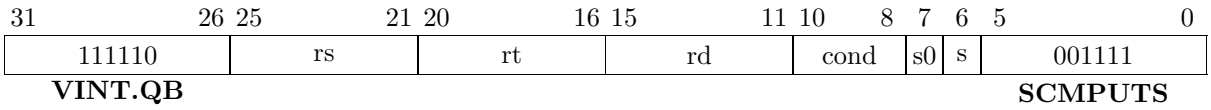
Vector Integer Exception :

概要 :

8 bit × 4 符号無しベクトル比較命令．結果は各要素ごとに 1bit を割り当ててスカラレジスタに格納される．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VSCMPUTS.QB Vector Compare Unsigned Quad Byte to Scalar Register

ベクトル比較 VECTOR



ニーモニック:

VSCMPUTS.cond.QB.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VSCMPUTS.cond.QB.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VSCMPUTS.cond.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VSCMPUTS.cond.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

SGPR[rd] ← (VGPR[rs] cond VGPR[rt])

例外 :

Vector Integer Exception :

概要 :

8 bit × 4 符号無しベクトル比較命令．VGPR[rt] の下位 8 bit を用いて演算を行う．結果は各要素ごとに 1bit を割り当ててスカラレジスタに格納される．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VADD.PH

Vector Add Paired HalfWord

ベクトル加算

VECTOR

31 26	25 21	20 16	15 11	10	9	8	7	6	5 0
110110	rs	rt	rd	0	s2	0	s0	s	100000
VINT.PH				**0**		**0**			**ADD**

ニーモニック:

VADD.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VADD.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VADD.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VADD.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VADD.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VADD.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VADD.PH.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VADD.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] + VGPR[rt]

例外 :

Vector Integer Exception :

概要 :

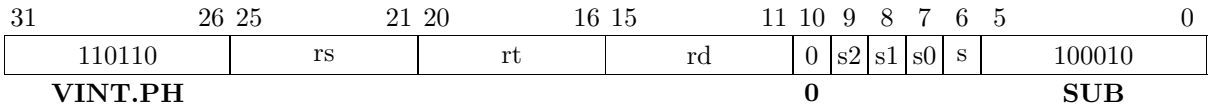
16 bit × 2 ベクトル加算．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sc2 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う．sync により，投機実行を抑制する．

VSUB.PH

Vector Subtract Paired HalfWord

ベクトル減算

VECTOR



ニーモニック:

VSUB.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) =0, scalar2(s2) = 0, sync(s) = 0)
VSUB.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) =0, scalar2(s2) = 0, sync(s) = 0)
VSUB.PH.sv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) =1, scalar2(s2) = 0, sync(s) = 0)
VSUB.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) =0, scalar2(s2) = 1, sync(s) = 0)
VSUB.PH.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) =0, scalar2(s2) = 0, sync(s) = 1)
VSUB.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) =0, scalar2(s2) = 1, sync(s) = 0)
VSUB.PH.sv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) =1, scalar2(s2) = 1, sync(s) = 0)
VSUB.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) =0, scalar2(s2) = 0, sync(s) = 1)
VSUB.PH.sv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) =1, scalar2(s2) = 0, sync(s) = 1)
VSUB.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) =0, scalar2(s2) = 1, sync(s) = 1)
VSUB.PH.sv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) =1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] - VGPR[rt]

例外 :

Vector Integer Exception :

概要 :

16 bit × 2 ベクトル減算．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sc1 の場合は VGPR[rs] の代わりに，スカラレジスタ (SGPR[rs]) を用いて演算を行う．sc2 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う．sync により，投機実行を抑制する．

VMULT.PH

Vector Multiply Paired HalfWord

ベクトル乗算

VECTOR

31 26	25 21	20 16	15 11	10	9	8	7	6	5 0
110110	rs	rt	rd	0	s2	0	s0	s	011000
VINT.PH				**0**		**0**			**MULT**

ニーモニック:

VMULT.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMULT.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMULT.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMULT.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMULT.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMULT.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMULT.PH.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMULT.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] × VGPR[rt]

例外 :

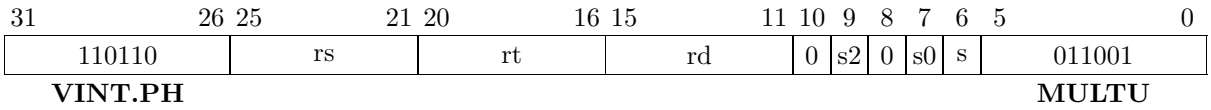
Vector Integer Exception :

概要 :

16 bit × 2 符号付きベクトル乗算．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sc2 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う．sync により，投機実行を抑制する．

VMULTU.PH Vector Multiply Unsigned

ベクトル乗算 VECTOR



ニーモニック:

VMULTU.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMULTU.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMULTU.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMULTU.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMULTU.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMULTU.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMULTU.PH.vv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMULTU.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] × VGPR[rt]

例外 :

Vector Integer Exception :

概要 :

16 bit × 2 符号無しベクトル乗算．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sc2 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う．sync により，投機実行を抑制する．

VMULTH.PH Vector Multiply Paired HalfWord on High Bit

ベクトル乗算 VECTOR

31 26	25 21	20 16	15 11	10	9	8	7	6	5 0
110110	rs	rt	rd	0	s2	0	s0	s	010000
VINT.PH				**0**		**0**			**MULRH**

ニーモニック:

VMULTH.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMULTH.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMULTH.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMULTH.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMULTH.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMULTH.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMULTH.PH.vv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMULTH.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] × VGPR[rt]

例外 :

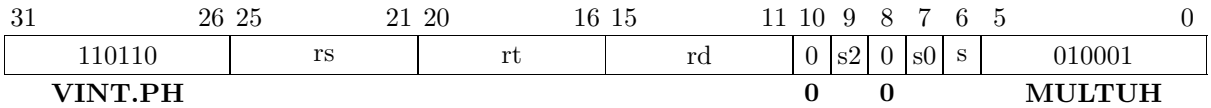
Vector Integer Exception :

概要 :

16 bit × 2 符号付きベクトル乗算．16 bit 毎に演算結果の上位 bit(32-16bit) が VGPR[rd] に格納される．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sc2 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う．sync により，投機実行を抑制する．

VMULTUH.PH Vector Multiply Unsigned Paired HalfWord on High Bit

ベクトル乗算 VECTOR



31–26	25–21	20–16	15–11	10	9	8	7	6	5–0
110110	rs	rt	rd	0	s2	0	s0	s	010001
VINT.PH				**0**		**0**			**MULTUH**

ニーモニック:

VMULTUH.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMULTUH.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMULTUH.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMULTUH.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMULTUH.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMULTUH.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMULTUH.PH.vv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMULTUH.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] × VGPR[rt]

例外 :

Vector Integer Exception :

概要 :

16 bit × 2 符号無しベクトル乗算．16 bit 毎に演算結果の上位 bit(32-16bit) が VGPR[rd] に格納される．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sc2 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う．sync により，投機実行を抑制する．

VMADD.PH Vector Multiply and Add Paired HalfWord

ベクトル積和演算 VECTOR

31 26	25 21	20 16	15 11	10	9	8	7	6	5 0
110110	rs	rt	rd	0	s2	0	s0	s	100001
VINT.PH				**0**		**0**			**MADD**

ニーモニック:

VMADD.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMADD.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMADD.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMADD.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMADD.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMADD.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMADD.PH.vv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMADD.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] × VGPR[rt] + VGPR[rd]

例外 :

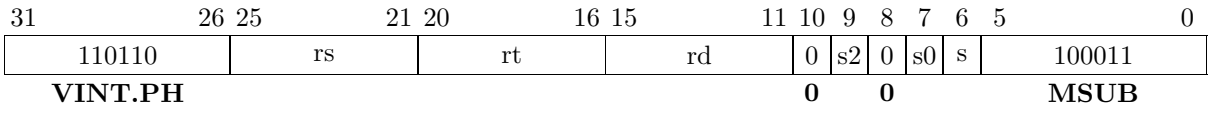
Vector Integer Exception :

概要 :

16 bit × 2 ベクトル積和演算．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sc2 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う．sync により，投機実行を抑制する．

VMSUB.PH Vector Multiply and Subtract Paired HalfWord

ベクトル積差演算 VECTOR



31–26	25–21	20–16	15–11	10	9	8	7	6	5–0
110110	rs	rt	rd	0	s2	0	s0	s	100011
VINT.PH				**0**		**0**			**MSUB**

ニーモニック:

VMSUB.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMSUB.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMSUB.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMSUB.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMSUB.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMSUB.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMSUB.PH.vv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMSUB.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] × VGPR[rt] - VGPR[rd]

例外 :

Vector Integer Exception :

概要 :

16 bit × 2 ベクトル積差演算．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sc2 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う．sync により，投機実行を抑制する．

VACC.PH Vector Accumulate Paired HalfWord

ベクトル累算 VECTOR

31 26	25 21	20 16	15 11	10 6	5 0
110110	rs	rt	rd	00000	001010
VINT.PH				**0**	**ACC**

ニーモニック:

VACC.PH rd, rs (sync(s) = 0)

VACC.PH.sy rd, rs (sync(s) = 1)

機能 :

SGPR[rd] ← $\sum$VGPR[rs]

例外 :

Vector Integer Exception :

概要 :

16 bit × 2 ベクトル累算．ベクトルの要素を全て加算する．sync により，投機実行を抑制する．

VMAC.PH Vector Multiply and Accumulate Paired HalfWord

ベクトル積和演算 VECTOR

31 26	25 21	20 16	15 11	10	9	8	7	6	5 0
110110	rs	rt	rd	0	s2	0	s0	s	001011
VINT.PH				**0**		**0**			**MAC**

ニーモニック:

VMAC.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMAC.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMAC.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMAC.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMAC.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMAC.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMAC.PH.vv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMAC.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

SGPR[rd] $\leftarrow \sum$VGPR[rs] $\times$ VGPR[rt]

例外 :

Vector Integer Exception :

概要 :

16 bit $\times$ 2 ベクトル積和演算．2 つのベクトル要素を乗算し，それを全て加算する．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sc2 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う．sync により，投機実行を抑制する．

VAND.PH

Vector And Paired HalfWord

ベクトル論理積

VECTOR

31 26	25 21	20 16	15 11	10	9	8	7	6	5 0
110110	rs	rt	rd	0	s2	0	s0	s	100100
VINT.PH				**0**		**0**			**AND**

ニーモニック:

VAND.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VAND.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VAND.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VAND.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VAND.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VAND.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VAND.PH.vv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VAND.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] and VGPR[rt]

例外 :

Vector Integer Exception :

概要 :

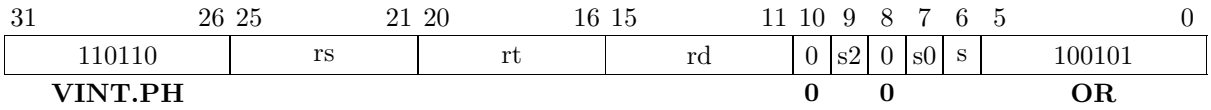
16 bit × 2 ベクトル論理積．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sc2 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う．sync により，投機実行を抑制する．

VOR.PH

Vector Or

ベクトル論理和

VECTOR



ニーモニック:

VOR.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VOR.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VOR.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VOR.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VOR.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VOR.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VOR.PH.vv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VOR.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] or VGPR[rt]

例外 :

Vector Integer Exception :

概要 :

16 bit × 2 ベクトル論理和．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sc2 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う．sync により，投機実行を抑制する．

VXOR.PH

Vector Exclusive Or Paired HalfWord

ベクトル排他的論理和 VECTOR

31 26	25 21	20 16	15 11	10	9	8	7	6	5 0
110110	rs	rt	rd	0	s2	0	s0	s	100110
VINT.PH				**0**		**0**			**XOR**

ニーモニック:

VXOR.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VXOR.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VXOR.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VXOR.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VXOR.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VXOR.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VXOR.PH.vv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VXOR.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] xor VGPR[rt]

例外 :

Vector Integer Exception :

概要 :

16 bit × 2 ベクトル排他的論理和．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sc2 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う．sync により，投機実行を抑制する．

VNOR.PH

Vector Not Or Paried HalfWord

ベクトル否定論理和

VECTOR

31 26	25 21	20 16	15 11	10	9	8	7	6	5 0
110110	rs	rt	rd	0	s2	0	s0	s	100111
VINT.PH				**0**		**0**			**NOR**

ニーモニック:

VNOR.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VNOR.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VNOR.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VNOR.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VNOR.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VNOR.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VNOR.PH.vv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VNOR.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rs] nor VGPR[rt]

例外 :

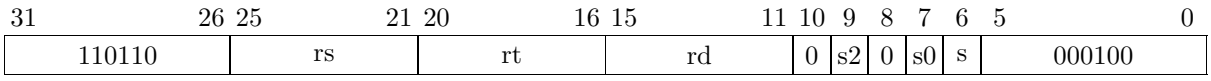
Vector Integer Exception :

概要 :

16 bit × 2 ベクトル否定論理和. sc0 の場合は VGPR[rt] の代わりに, スカラレジスタ (SGPR[rt]) を用いて演算を行う. sc2 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う. sync により, 投機実行を抑制する.

VSLLV.PH Vector Shift Left Logical Variable Paired HalfWord

ベクトル左論理シフト VECTOR



ニーモニック:

VSLLV.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VSLLV.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VSLLV.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VSLLV.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VSLLV.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VSLLV.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VSLLV.PH.vv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VSLLV.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rt] ≪ VGPR[rs]

例外 :

Vector Integer Exception :

概要 :

16 bit × 2 ベクトル左論理シフト．sc0 の場合は VGPR[rs] の代わりに，スカラレジスタ (SGPR[rs]) を用いて演算を行う．sc2 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う．sync により，投機実行を抑制する．

VSRLV.PH Vector Shift Right Logical Variable Paired HalfWord

ベクトル右論理シフト VECTOR

31 26	25 21	20 16	15 11	10	9	8	7	6	5 0
110110	rs	rt	rd	0	s2	0	s0	s	000110
VINT.PH				**0**		**0**			**SRLV**

ニーモニック:

VSRLV.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VSRLV.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VSRLV.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VSRLV.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VSRLV.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VSRLV.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VSRLV.PH.vv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VSRLV.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rt] ≫ VGPR[rs]

例外 :

Vector Integer Exception :

概要 :

16 bit × 2 ベクトル右論理シフト．sc0 の場合は VGPR[rs] の代わりに，スカラレジスタ (SGPR[rs]) を用いて演算を行う．sc2 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う．sync により，投機実行を抑制する．

VSRAV.PH Vector Shift Right Arithmetic Variable Paired HalfWord

ベクトル右算術シフト VECTOR

31 26	25 21	20 16	15 11	10	9	8	7	6	5 0
110110	rs	rt	rd	0	s2	0	s0	s	000111
VINT.PH				**0**		**0**			**SRAV**

ニーモニック:

VSRAV.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VSRAV.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VSRAV.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VSRAV.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VSRAV.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VSRAV.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VSRAV.PH.vv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VSRAV.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rt] ≫ VGPR[rs]

例外 :

Vector Integer Exception :

概要 :

ベクトル右算術シフト．sc0 の場合は VGPR[rs] の代わりに，スカラレジスタ (SGPR[rs]) を用いて演算を行う．sc2 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う．sync により，投機実行を抑制する．

VRTLV.PH

Vector Rotate Left Variable Paired HalfWord

ベクトル左ローテーション

VECTOR

31 26	25 21	20 16	15 11	10	9	8	7	6	5 0
110110	rs	rt	rd	0	s2	0	s0	s	000000
VINT.PH				**0**		**0**			**SRTLV**

ニーモニック:

VRTLV.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VRTLV.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VRTLV.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VRTLV.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VRTLV.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VRTLV.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VRTLV.PH.vv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VRTLV.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能：

VGPR[rd] ← VGPR[rt] <<< VGPR[rs]

例外：

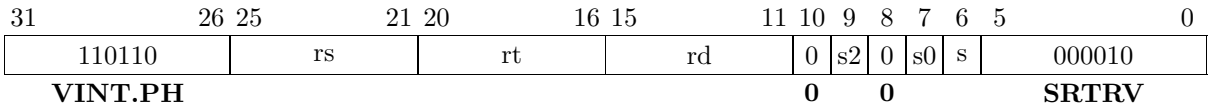
Vector Integer Exception :

概要：

ベクトル左ローテーション．sc0 の場合は VGPR[rs] の代わりに，スカラレジスタ (SGPR[rs]) を用いて演算を行う．sc2 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う．sync により，投機実行を抑制する．

VRTRV.PH Vector Rotate Right Variable Paired HalfWord

ベクトル右ローテーション VECTOR



ニーモニック:

VRTRV.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VRTRV.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VRTRV.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VRTRV.PH.vv,sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VRTRV.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VRTRV.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VRTRV.PH.vv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VRTRV.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← VGPR[rt] >>> VGPR[rs]

例外 :

Vector Integer Exception :

概要 :

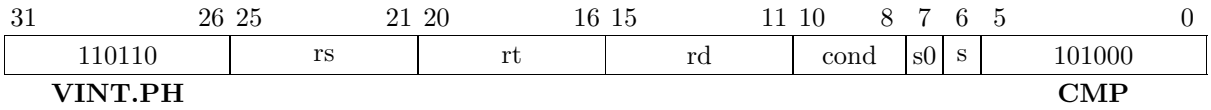
ベクトル右ローテーション．sc0 の場合は VGPR[rs] の代わりに，スカラレジスタ (SGPR[rs]) を用いて演算を行う．sc2 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う．sync により，投機実行を抑制する．

VCMP.PH

Vector Compare

ベクトル比較

VECTOR



ニーモニック:

VCMP.cond.PH.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMP.cond.PH.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMP.cond.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMP.cond.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← (VGPR[rs] cond VGPR[rt])

例外 :

Vector Integer Exception :

概要 :

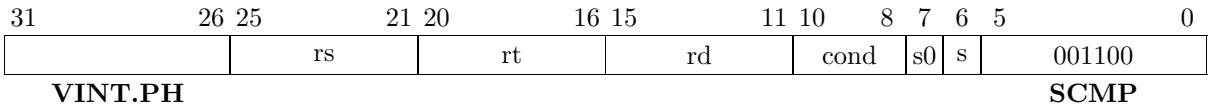
16 bit × 2 ベクトル比較命令．条件 (cond) により VGPR[rd] に 1 または 0 が入る．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VSCMP.PH

Vector Compare

ベクトル比較

VECTOR



ニーモニック:

VSCMP.cond.PH.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VSCMP.cond.PH.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VSCMP.cond.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VSCMP.cond.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← (VGPR[rs] cond VGPR[rt])

例外 :

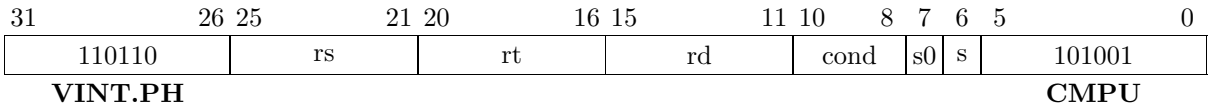
Vector Integer Exception :

概要 :

16 bit × 2 ベクトル比較命令．VGPR[rt] の下位 16 bit を用いて演算を行う．条件 (cond) により VGPR[rd] に 1 または 0 が入る．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VCMPU.PH　Vector Compare Unsigned Paired HalfWord

ベクトル比較　VECTOR



ニーモニック:

VCMPU.cond.PH.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMPU.cond.PH.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMPU.cond.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMPU.cond.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← (VGPR[rs] cond VGPR[rt])

例外 :

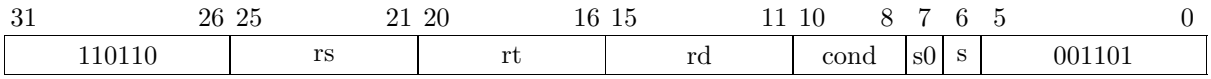
Vector Integer Exception :

概要 :

16 bit × 2 符号無しベクトル比較命令．条件 (cond) により VGPR[rd] に 1 または 0 が入る．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VSCMPU.PH Vector Compare Unsigned Paired HalfWord

ベクトル比較 VECTOR



VINT.PH **SCMPU**

ニーモニック:

VSCMPU.cond.PH.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VSCMPU.cond.PH.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VSCMPU.cond.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VSCMPU.cond.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VGPR[rd] ← (VGPR[rs] cond VGPR[rt])

例外 :

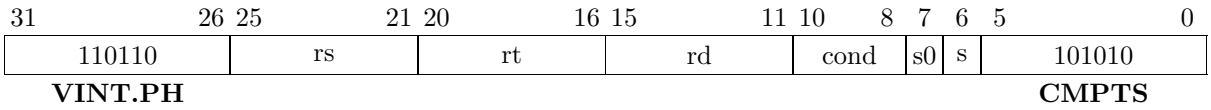
Vector Integer Exception :

概要 :

16 bit × 2 符号無しベクトル比較命令．VGPR[rt] の下位 16 bit を用いて演算を行う．条件 (cond) により VGPR[rd] に 1 または 0 が入る．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VCMPTS.PH Vector Compare to Scalar Register Paired HalfWord

ベクトル比較 VECTOR



ニーモニック:

VCMPTS.cond.PH.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMPTS.cond.PH.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMPTS.cond.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMPTS.cond.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

SGPR[rd] ← (VGPR[rs] cond VGPR[rt])

例外 :

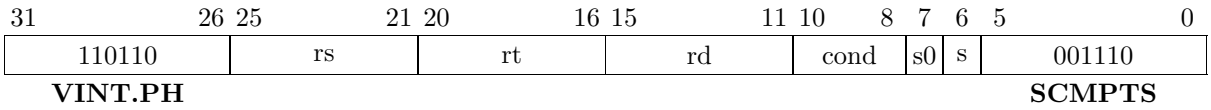
Vector Integer Exception :

概要 :

16 bit × 2 ベクトル比較命令．結果は各要素ごとに 1bit を割り当ててスカラレジスタに格納される．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VSCMPTS.PH Vector Compare Paired HalfWord to Scalar Register

ベクトル比較 VECTOR



ニーモニック:

VSCMPTS.cond.PH.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VSCMPTS.cond.PH.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VSCMPTS.cond.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VSCMPTS.cond.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

SGPR[rd] ← (VGPR[rs] cond VGPR[rt])

例外 :

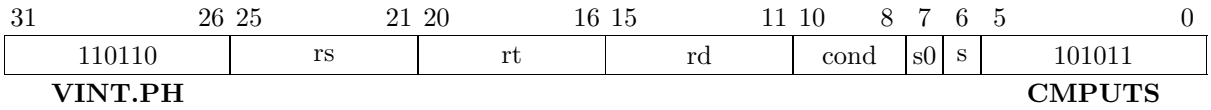
Vector Integer Exception :

概要 :

16 bit × 2 ベクトル比較命令．VGPR[rt] の下位 16 bit を用いて演算を行う．結果は各要素ごとに 1bit を割り当ててスカラレジスタに格納される．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VCMPUTS.PH Vector Compare Unsigned Paired HalfWord to Scalar Register

ベクトル比較 VECTOR



ニーモニック:

VCMPUTS.cond.PH.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMPUTS.cond.PH.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMPUTS.cond.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMPUTS.cond.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

SGPR[rd] ← (VGPR[rs] cond VGPR[rt])

例外 :

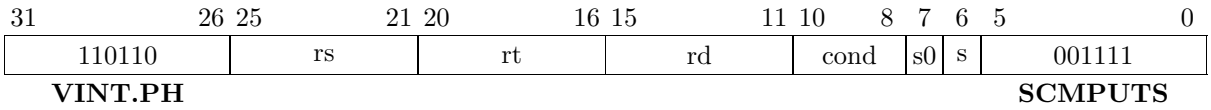
Vector Integer Exception :

概要 :

16 bit × 2 符号無しベクトル比較命令．結果は各要素ごとに 1bit を割り当ててスカラレジスタに格納される．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VSCMPUTS.PH Vector Compare Unsigned Paired HalfWord to Scalar Register

ベクトル比較 VECTOR



ニーモニック:

VSCMPUTS.cond.PH.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VSCMPUTS.cond.PH.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VSCMPUTS.cond.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VSCMPUTS.cond.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

SGPR[rd] ← (VGPR[rs] cond VGPR[rt])

例外 :

Vector Integer Exception :

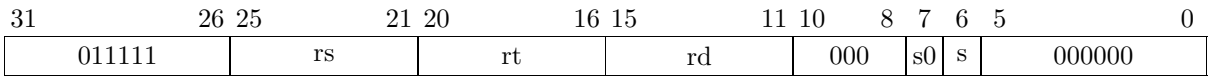
概要 :

16 bit × 2 符号無しベクトル比較命令．VGPR[rt] の下位 16 bit を用いて演算を行う．結果は各要素ごとに 1bit を割り当ててスカラレジスタに格納される．sc0 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

3.3.9 浮動小数点ベクトル命令

VADD.S — Vector Add Single

ベクトル加算 VECTOR



ニーモニック:

VADD.S.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VADD.S.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VADD.S.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VADD.S.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VFPR[rd] ← VFPR[rs] + VFPR[rt]

例外 :

Vector Floating Point Exception :

概要 :

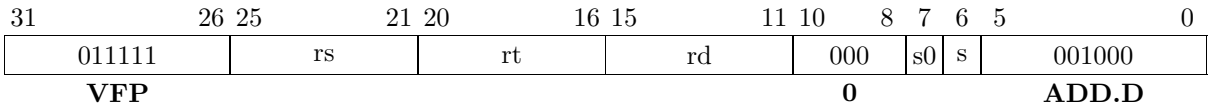
単精度ベクトル加算．sc0 の場合は VFPR[rt] の代わりに，スカラレジスタ (SFPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VADD.D

Vector Add Double

ベクトル加算

VECTOR



ニーモニック:

VADD.D.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VADD.D.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VADD.D.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VADD.D.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VFPR[rd] ← VFPR[rs] + VFPR[rt]

例外 :

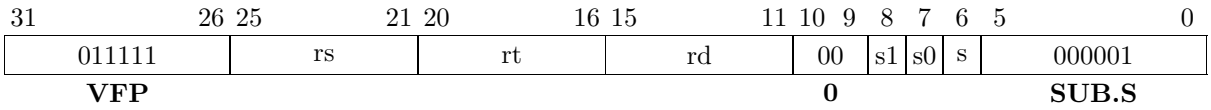
Vector Floating Point Exception :

概要 :

倍精度ベクトル加算．sc0 の場合は VFPR[rt] の代わりに，スカラレジスタ (SFPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VSUB.S Vector Subtract Single

ベクトル減算 VECTOR



ニーモニック:

VSUB.S.vv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 0)
VSUB.S.vs rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 0)
VSUB.S.sv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 0)
VSUB.S.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 1)
VSUB.S.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 1)
VSUB.S.sv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 1)

機能 :

VFPR[rd] ← VFPR[rs] − VFPR[rt]

例外 :

Vector Floating Point Exception :

概要 :

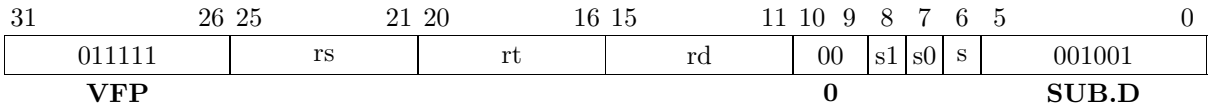
単精度ベクトル減算．sc0 の場合は VFPR[rt] の代わりに，スカラレジスタ (SFPR[rt]) を用いて演算を行う．sc1 の場合は VFPR[rs] の代わりに，スカラレジスタ (SFPR[rs]) を用いて演算を行う．sync により，投機実行を抑制する．

VSUB.D

Vector Subtract Double

ベクトル減算

VECTOR



ニーモニック:

VSUB.D.vv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 0)
VSUB.D.vs rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 0)
VSUB.D.sv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 0)
VSUB.D.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 1)
VSUB.D.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 1)
VSUB.D.sv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 1)

機能 :

VFPR[rd] ← VFPR[rs] − VFPR[rt]

例外 :

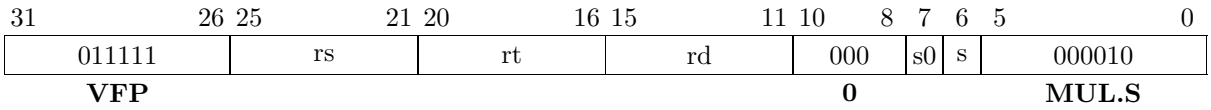
Vector Floating Point Exception :

概要 :

倍精度ベクトル減算．sc0 の場合は VFPR[rt] の代わりに，スカラレジスタ (SFPR[rt]) を用いて演算を行う．sc1 の場合は VFPR[rs] の代わりに，スカラレジスタ (SFPR[rs]) を用いて演算を行う．sync により，投機実行を抑制する．

VMUL.S Vector Multiply Single

ベクトル乗算 VECTOR



ニーモニック:

VMUL.S.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VMUL.S.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VMUL.S.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VMUL.S.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VFPR[rd] ← VFPR[rs] × VFPR[rt]

例外 :

Vector Floating Point Exception :

概要 :

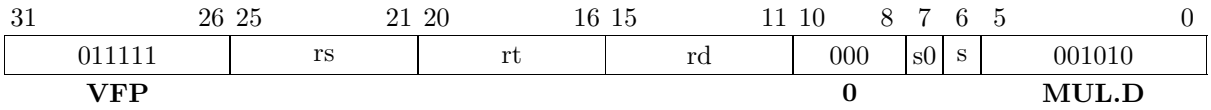
単精度ベクトル乗算．sc0 の場合は VFPR[rt] の代わりに，スカラレジスタ (SFPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VMUL.D

Vector Multiply Double

ベクトル乗算

VECTOR



ニーモニック:

VMUL.D.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VMUL.D.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VMUL.D.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VMUL.D.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VFPR[rd] ← VFPR[rs] × VFPR[rt]

例外 :

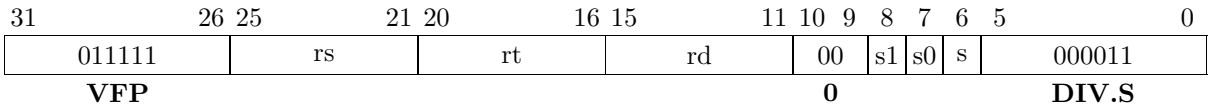
Vector Floating Point Exception :

概要 :

倍精度ベクトル乗算．sc0 の場合は VFPR[rt] の代わりに，スカラレジスタ (SFPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VDIV.S Vector Divide Single

ベクトル除算 VECTOR



ニーモニック:

VDIV.S.vv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 0)
VDIV.S.vs rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 0)
VDIV.S.sv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 0)
VDIV.S.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 1)
VDIV.S.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 1)
VDIV.S.sv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 1)

機能 :

VFPR[rd] ← VFPR[rs] ÷ VFPR[rt]

例外 :

Vector Floating Point Exception :

概要 :

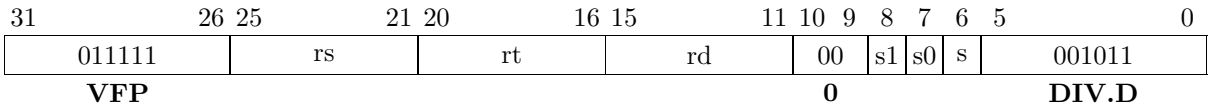
単精度ベクトル除算．sc0 の場合は VFPR[rt] の代わりに，スカラレジスタ (SFPR[rt]) を用いて演算を行う．sc1 の場合は VFPR[rs] の代わりに，スカラレジスタ (SFPR[rs]) を用いて演算を行う．sync により，投機実行を抑制する．

VDIV.D

Vector Divide Double

ベクトル除算

VECTOR



ニーモニック:

VDIV.D.vv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 0)
VDIV.D.vs rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 0)
VDIV.D.sv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 0)
VDIV.D.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 1)
VDIV.D.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 1)
VDIV.D.sv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 1)

機能 :

VFPR[rd] ← VFPR[rs] ÷ VFPR[rt]

例外 :

Vector Floating Point Exception :

概要 :

倍精度ベクトル除算．sc0 の場合は VFPR[rt] の代わりに，スカラレジスタ (SFPR[rt]) を用いて演算を行う．sc1 の場合は VFPR[rs] の代わりに，スカラレジスタ (SFPR[rs]) を用いて演算を行う．sync により，投機実行を抑制する．

VABS.S Vector Absolute Single

ベクトル絶対値演算 VECTOR

31 26	25 21	20 16	15 11	10 7	6	5 0
011111	rs	00000	rd	0000	s	000101
VFP		**0**		**0**		**ABS.S**

ニーモニック:

VABS.S rd, rs (sync(s) = 0)

VABS.S.sy rd, rs (sync(s) = 1)

機能 :

VFPR[rd] ← | VFPR[rs] |

例外 :

Vector Floating Point Exception :

概要 :

単精度ベクトル絶対値演算．sync により，投機実行を抑制する．

VABS.D Vector Absolute Double

ベクトル絶対値演算 VECTOR

31 26	25 21	20 16	15 11	10 7	6	5 0
011111	rs	00000	rd	0000	s	001101
VFP		**0**		**0**		**ABS.D**

ニーモニック:

VABS.D rd, rs (sync(s) = 0)

VABS.D.sy rd, rs (sync(s) = 1)

機能 :

VFPR[rd] ← | VFPR[rs] |

例外 :

Vector Floating Point Exception :

概要 :

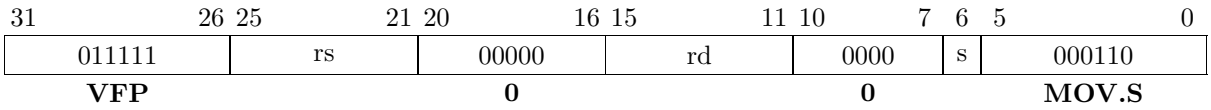
倍精度ベクトル絶対値演算．sync により，投機実行を抑制する．

VMOV.S

Vector Move Single

ベクトル転送

VECTOR



ニーモニック:

VMOV.S rd, rs　(sync(s) = 0)

VMOV.S.sy rd, rs　(sync(s) = 1)

機能 :

VFPR[rd] ← VFPR[rs]

例外 :

Vector Floating Point Exception :

概要 :

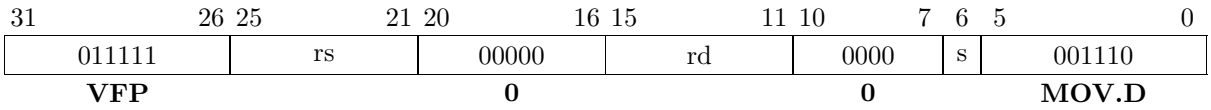
単精度ベクトル転送命令．sync により，投機実行を抑制する．

VMOV.D

Vector Move Double

ベクトル転送

VECTOR



ニーモニック:

VMOV.D rd, rs (sync(s) = 0)

VMOV.D.sy rd, rs (sync(s) = 1)

機能 :

VFPR[rd] ← VFPR[rs]

例外 :

Vector Floating Point Exception :

概要 :

倍精度ベクトル転送命令．sync により，投機実行を抑制する．

VNEG.S Vector Negate Single

ベクトル符号反転 VECTOR

31 26	25 21	20 16	15 11	10 7	6	5 0
011111	rs	00000	rd	0000	s	000111
VFP		**0**		**0**		**NEG.S**

ニーモニック:

VNEG.S rd, rs (sync(s) = 0)
VNEG.S.sy rd, rs (sync(s) = 1)

機能 :

VFPR[rd] ← −1 × VFPR[rs]

例外 :

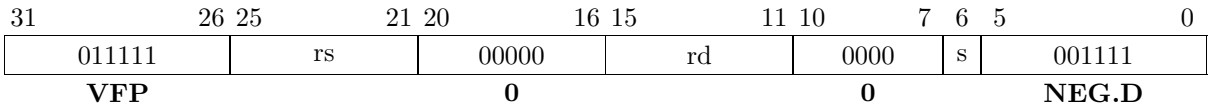
Vector Floating Point Exception :

概要 :

単精度ベクトル符号反転演算．sync により，投機実行を抑制する．

VNEG.D Vector Negate Double

ベクトル符号反転 VECTOR



31 26	25 21	20 16	15 11	10 7	6	5 0
011111	rs	00000	rd	0000	s	001111
VFP		**0**		**0**		**NEG.D**

ニーモニック:

VNEG.D rd, rs (sync(s) = 0)

VNEG.D.sy rd, rs (sync(s) = 1)

機能 :

VFPR[rd] ← −1 × VFPR[rs]

例外 :

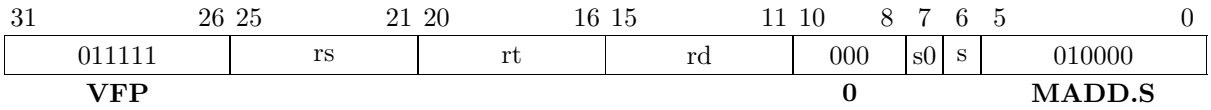
Vector Floating Point Exception :

概要 :

倍精度ベクトル符号反転演算．sync により，投機実行を抑制する．

VMADD.S Vector Multiply and Add Single

ベクトル積和演算 VECTOR



ニーモニック:

VMADD.S.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VMADD.S.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VMADD.S.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VMADD.S.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VFPR[rd] ← VFPR[rs] × VFPR[rt] + VFPR[rd]

例外 :

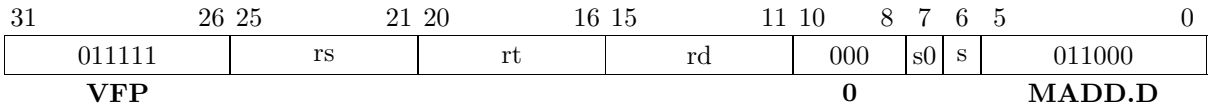
Vector Floating Point Exception :

概要 :

単精度ベクトル積和演算．sc0 の場合は VFPR[rt] の代わりに，スカラレジスタ (SFPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VMADD.D Vector Multiply and Add Double

ベクトル積和演算 VECTOR



ニーモニック:

VMADD.D.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VMADD.D.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VMADD.D.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VMADD.D.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VFPR[rd] ← VFPR[rs] × VFPR[rt] + VFPR[rd]

例外 :

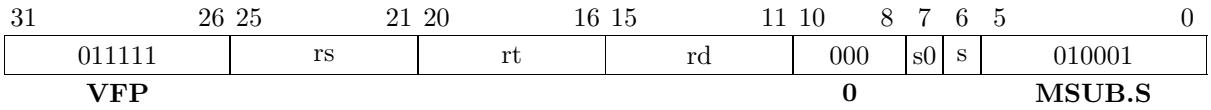
Vector Floating Point Exception :

概要 :

倍精度ベクトル積和演算．sc0 の場合は VFPR[rt] の代わりに，スカラレジスタ (SFPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VMSUB.S Vector Multiply and Subtract Single

ベクトル積差演算 VECTOR



ニーモニック:

VMSUB.S.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VMSUB.S.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VMSUB.S.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VMSUB.S.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VFPR[rd] ← VFPR[rs] × VFPR[rt] − VFPR[rd]

例外 :

Vector Floating Point Exception :

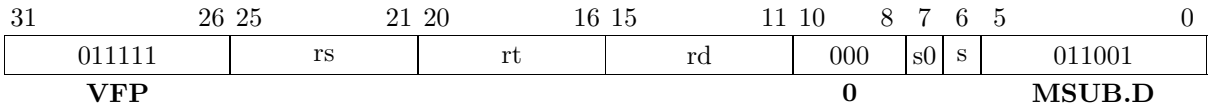
概要 :

単精度ベクトル積差演算．sc0 の場合は VFPR[rt] の代わりに，スカラレジスタ (SFPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VMSUB.D

Vector Multiply and Subtract Double

ベクトル積差演算 VECTOR



ニーモニック:

VMSUB.D.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VMSUB.D.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VMSUB.D.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VMSUB.D.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VFPR[rd] ← VFPR[rs] × VFPR[rt] − VFPR[rd]

例外 :

Vector Floating Point Exception :

概要 :

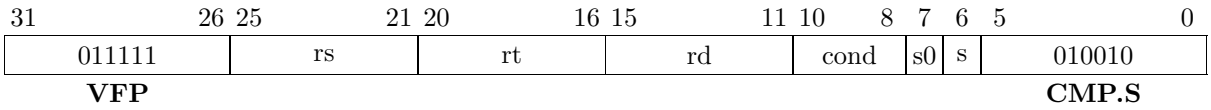
倍精度ベクトル積差演算．sc0 の場合は VFPR[rt] の代わりに，スカラレジスタ (SFPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VCMP.S

Vector Compare Single

ベクトル比較

VECTOR



ニーモニック:

VCMP.S.cond.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMP.S.cond.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMP.S.cond.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMP.S.cond.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VFPR[rd] ← VFPR[rs] cond VFPR[rt]

例外 :

Vector Floating Point Exception :

概要 :

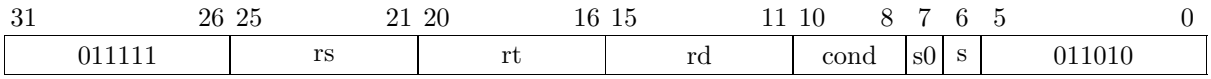
単精度ベクトル比較命令. 条件 (cond) により VFPR[rd] の値が決定する. sc0 の場合は VFPR[rt] の代わりに，スカラレジスタ (SFPR[rt]) を用いて演算を行う. sync により，投機実行を抑制する.

VCMP.D

Vector Compare Double

ベクトル比較

VECTOR



ニーモニック:

VCMP.D.cond.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMP.D.cond.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMP.D.cond.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMP.D.cond.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VFPR[rd] ← VFPR[rs] cond VFPR[rt]

例外 :

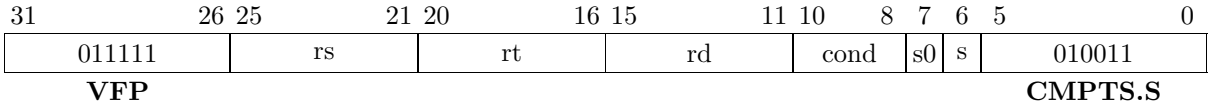
Vector Floating Point Exception :

概要 :

倍精度ベクトル比較命令. 条件 (cond) により VFPR[rd] の値が決定する. sc0 の場合は VFPR[rt] の代わりに，スカラレジスタ (SFPR[rt]) を用いて演算を行う. sync により，投機実行を抑制する.

VCMPTS.S Vector Compare Single to Scalar Register

ベクトル比較 VECTOR



ニーモニック:

VCMPTS.S.cond.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMPTS.S.cond.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMPTS.S.cond.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMPTS.S.cond.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

SFPR[rd] ← VFPR[rs] cond VFPR[rt]

例外 :

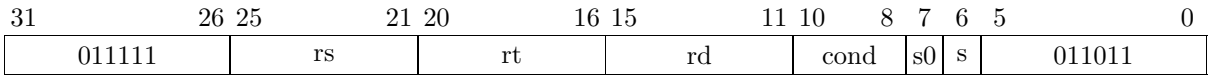
Vector Floating Point Exception :

概要 :

単精度ベクトル比較命令．結果は各要素ごとに 1bit を割り当ててスカラレジスタに格納される．sc0 の場合は VFPR[rt] の代わりに，スカラレジスタ (SFPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VCMPTS.D Vector Compare Double to Scalar Register

ベクトル比較 VECTOR



ニーモニック:

VCMPTS.D.cond.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMPTS.D.cond.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMPTS.D.cond.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMPTS.D.cond.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

SFPR[rd] ← VFPR[rs] cond VFPR[rt]

例外 :

Vector Floating Point Exception :

概要 :

倍精度ベクトル比較命令．結果は各要素ごとに 1bit を割り当ててスカラレジスタに格納される．sc0 の場合は VFPR[rt] の代わりに，スカラレジスタ (SFPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VCVT.S.D

Vector Convert to Single from Double

ベクトルフォーマット変換 VECTOR

31 26	25 21	20 16	15 11	10 7	6	5 0
011111	rs	00000	rd	0000	s	101000
VFP		**0**		**0**		**VCVT.S.D**

ニーモニック:

VCVT.S.D rd, rs (sync(s) = 0)

VCVT.S.D.sy rd, rs (sync(s) = 1)

機能 :

VFPR[rd] ← Double_to_Single(VFPR[rs])

例外 :

Vector Floating Point Exception :

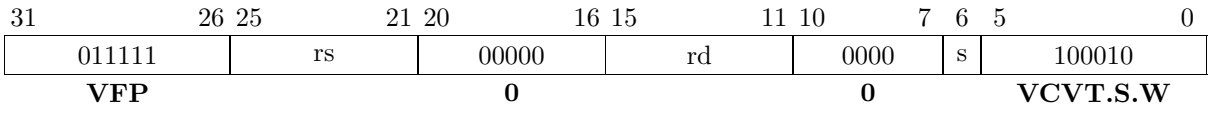
概要 :

倍精度フォーマットから単精度フォーマットへ変換する．sync により，投機実行を抑制する．

VCVT.S.W

Vector Convert to Single from Word

ベクトルフォーマット変換 VECTOR



31–26	25–21	20–16	15–11	10–7	6	5–0
011111	rs	00000	rd	0000	s	100010
VFP		**0**		**0**		**VCVT.S.W**

ニーモニック:

VCVT.S.W rd, rs (sync(s) = 0)

VCVT.S.W.sy rd, rs (sync(s) = 1)

機能 :

VFPR[rd] ← Word_to_Single(VFPR[rs])

例外 :

Vector Floating Point Exception :

概要 :

整数フォーマットから単精度フォーマットへ変換する．sync により，投機実行を抑制する．

VCVT.D.S

Vector Convert to Double from Single

ベクトルフォーマット変換 VECTOR

31 26	25 21	20 16	15 11	10 7	6	5 0
011111	rs	00000	rd	0000	s	100001
VFP		**0**		**0**		**VCVT.D.S**

ニーモニック:

VCVT.D.S rd, rs (sync(s) = 0)

VCVT.D.S.sy rd, rs (sync(s) = 1)

機能 :

VFPR[rd] ← Single_to_Double(VFPR[rs])

例外 :

Vector Floating Point Exception :

概要 :

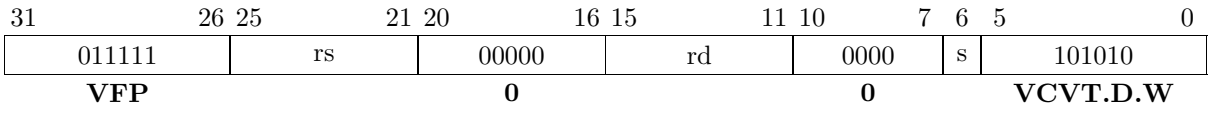
単精度フォーマットから倍精度フォーマットへ変換する．sync により，投機実行を抑制する．

VCVT.D.W

Vector Convert to Double from Word

ベクトルフォーマット変換

VECTOR



31–26	25–21	20–16	15–11	10–7	6	5–0
011111	rs	00000	rd	0000	s	101010
VFP		**0**		**0**		**VCVT.D.W**

ニーモニック:

VCVT.D.W rd, rs (sync(s) = 0)

VCVT.D.W.sy rd, rs (sync(s) = 1)

機能 :

VFPR[rd] ← Word_to_Double(VFPR[rs])

例外 :

Vector Floating Point Exception :

概要 :

整数フォーマットから倍精度フォーマットへ変換する．sync により，投機実行を抑制する．

VCVT.W.S

Vector Convert to Word from Single

ベクトルフォーマット変換 VECTOR

31 26	25 21	20 16	15 11	10 7	6	5 0
011111	rs	00000	rd	0000	s	100100
VFP		**0**		**0**		**VCVT.W.S**

ニーモニック:

VCVT.W.S rd, rs (sync(s) = 0)

VCVT.W.S.sy rd, rs (sync(s) = 1)

機能 :

VFPR[rd] ← Single_to_Word(VFPR[rs])

例外 :

Vector Floating Point Exception :

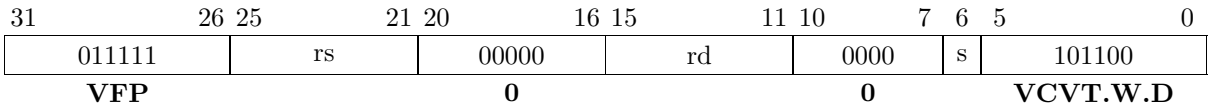
概要 :

単精度フォーマットから整数フォーマットへ変換する．sync により，投機実行を抑制する．

VCVT.W.D

Vector Convert to Word from Double

ベクトルフォーマット変換 VECTOR



31–26	25–21	20–16	15–11	10–7	6	5–0
011111	rs	00000	rd	0000	s	101100
VFP		**0**		**0**		**VCVT.W.D**

ニーモニック:

VCVT.W.D rd, rs (sync(s) = 0)

VCVT.W.D.sy rd, rs (sync(s) = 1)

機能 :

VFPR[rd] ← Double_to_Word(VFPR[rs])

例外 :

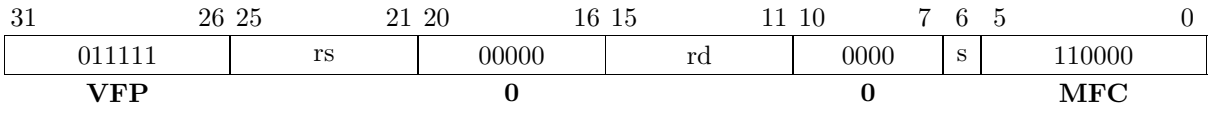
Vector Floating Point Exception :

概要 :

倍精度フォーマットから整数フォーマットへ変換する．sync により，投機実行を抑制する．

VFMFC Move from Vector Floating-Point Control Register

制御レジスタリード VECTOR



ニーモニック:

VFMFC rd, rs (sync(s) = 0)
VFMFC.sy rd, rs (sync(s) = 1)

機能 :

FPR[rd] ← VFCTRL[rs]

例外 :

Vector Floating Point Exception :

概要 :

浮動小数点ベクトル制御レジスタリード命令．rs で指定された制御レジスタの値を浮動小数点レジスタに格納する．sync により，投機実行を抑制する．

VFMTC Move to Vector Floating-Point Control Register

制御レジスタライト VECTOR

31 26	25 21	20 16	15 11	10 7	6	5 0
011111	rs	00000	rd	0000	s	110001
VFP		**0**		**0**		**MTC**

ニーモニック:

VFMTC rd, rs (sync(s) = 0)
VFMTC.sy rd, rs (sync(s) = 1)

機能 :

VFCTRL[rd] ← FPR[rs]

例外 :

Vector Floating Point Exception :

概要 :

浮動小数点ベクトル制御レジスタライト命令．rd で指定された制御レジスタに浮動小数点レジスタの値を格納する．sync により，投機実行を抑制する．

VFMFS　Move from Vector Floating-Point Scalar Register

浮動小数点スカラレジスタリード　VECTOR

31 26	25 21	20 16	15 11	10 7	6	5 0
011111	rs	00000	rd	0000	s	110010
VFP		**0**		**0**		**MFS**

ニーモニック:

VFMFS rd, rs　(sync(s) = 0)

VFMFS.sy rd, rs　(sync(s) = 1)

機能：

FPR[rd] ← SFPR[rs]

例外：

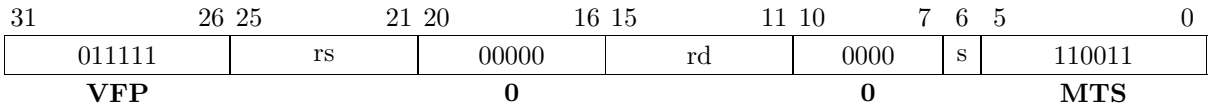
Vector Floating Point Exception :

概要：

浮動小数点スカラレジスタリード命令．rs で指定された浮動小数点スカラレジスタの値を浮動小数点レジスタに格納する．sync により，投機実行を抑制する．

VFMTS Move to Vector Floating-Point Scalar Register

浮動小数点スカラレジスタライト VECTOR



ニーモニック:

VFMTS rd, rs (sync(s) = 0)

VFMTS.sy rd, rs (sync(s) = 1)

機能 :

SFPR[rd] ← FPR[rs]

例外 :

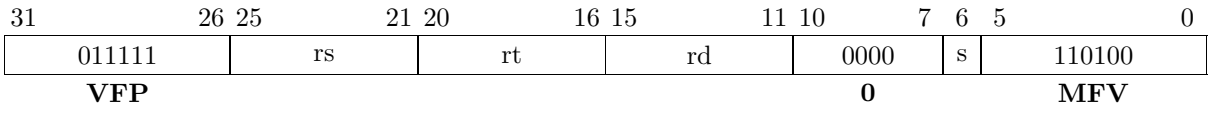
Vector Floating Point Exception :

概要 :

浮動小数点スカラレジスタライト命令．rd で指定された浮動小数点スカラレジスタに浮動小数点レジスタの値を格納する．sync により，投機実行を抑制する．

VFMFV Move from Vector Floating-Point Vector Register

浮動小数点ベクトルレジスタリード VECTOR



ニーモニック:

VFMFV rd, rs, rt (sync(s) = 0)
VFMFV.sy rd, rs, rt (sync(s) = 1)

機能 :

SFPR[rd] ← VFPR[rs][rt]

例外 :

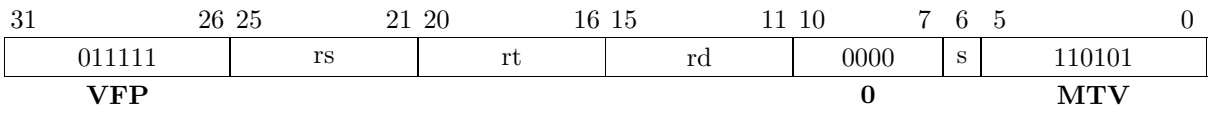
Vector Floating Point Exception :

概要 :

浮動小数点ベクトルレジスタリード命令．rs で指定された浮動小数点ベクトルレジスタの rt 番目の要素の値を浮動小数点レジスタに格納する．sync により，投機実行を抑制する．

VFMTV Move to Vector Floating-Point Vector Register

浮動小数点ベクトルレジスタライト VECTOR



ニーモニック:

VFMTV rd, rs, rt (sync(s) = 0)

VFMTV.sy rd, rs, rt (sync(s) = 1)

機能 :

SFPR[rd] ← VFPR[rs][rt]

例外 :

Vector Floating Point Exception :

概要 :

浮動小数点ベクトルレジスタライト命令．rd で指定された浮動小数点ベクトルレジスタの rt 番目の要素に浮動小数点レジスタの値を書き込む．sync により，投機実行を抑制する．

VFMTM　Move to Vector Floating-Point Mask Register

浮動小数点ベクトルマスクレジスタライト　VECTOR

31 26	25 21	20 7	6	5 0
011111	rs	00000000000000	s	011110
VFP		**0**		**MTM**

ニーモニック:

VFMTM rs　(sync(s) = 0)

VFMTM.sy rs　(sync(s) = 1)

機能 :

VFCTRL[*Mask Register*] ← VSFPR[rs]

例外 :

Vector Floating Point Exception :

概要 :

浮動小数点ベクトルマスクレジスタライト命令．rd で指定した浮動小数点スカラレジスタの値をマスクレジスタに格納する．sync により，投機実行を抑制する．

VFRSV Vector Floating-Point Register Reserve

浮動小数点ベクトルレジスタ予約 VECTOR

31 26	25 21	20 16	15 11	10 7	6	5 0
011111	rs	00000	rd	0000	s	110110
VFP		**0**		**0**		**RSV**

ニーモニック:

VFRSV rd, rs (sync(s) = 0)

VFRSV.sy rd, rs (sync(s) = 1)

機能 :

GPR[rd] ← if(成功) then 1 else 0

例外 :

None

概要 :

浮動小数点ベクトルレジスタ予約命令．GPR[rs] に予約するレジスタの構成を指定する．予約に成功した場合は GPR[rd] に 1 が，失敗した場合は 0 が格納される．sync により，投機実行を抑制する．

VFRLS

Vector Floating-Point Register Release

浮動小数点ベクトルレジスタ開放

VECTOR

31 26	25 16	15 11	10 7	6	5 0
011111	0000000000	rd	0000	s	110111
VFP	**0**		**0**		**RLS**

ニーモニック:

VFRLS rd	(sync(s) = 0)
VFRLS.sy rd	(sync(s) = 1)

機能：

GPR[rd] ← if(成功) then 1 else 0

例外：

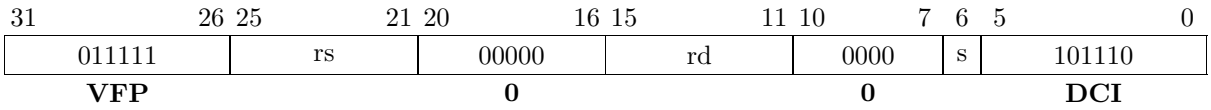
None

概要：

浮動小数点ベクトルレジスタ開放命令．開放に成功した場合は GPR[rd] に 1 が，失敗した場合は 0 が格納される．sync を 1 にすることにより，投機実行を抑制する．

VFDCI Vector Floating-Point Define Compound Instruction

浮動小数点ベクトル複合命令定義 VECTOR



31–26	25–21	20–16	15–11	10–7	6	5–0
011111	rs	00000	rd	0000	s	101110
VFP		**0**		**0**		**DCI**

ニーモニック:

VFDCI rd, rs (sync(s) = 0)

VFDCI.sy rd, rs (sync(s) = 1)

機能 :

VFCPD[rd] ← GPR[rs]

例外 :

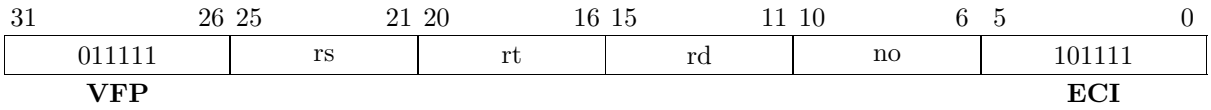
Vector Floating Point Exception :

概要 :

浮動小数点ベクトル複合命令の定義を行う．GPR[rs] で定義した命令を rd で指定した複合命令バッファのアドレスに格納する．sync により，投機実行を抑制する．

VFECI　Vector Floating-Point Execute Compound Instruction

浮動小数点ベクトル複合命令実行　VECTOR



ニーモニック:

VFDCI rd, rs, rt, no

機能 :

VFPR[rd] ← VFPR[rs] op VFPR[rt]

例外 :

Vector Floating Point Exception :

概要 :

浮動小数点ベクトル複合命令の実行を行う．no で指定した複合命令バッファのアドレスから命令を実行する．

VFLW Vector Floating-Point Load Word

浮動小数点ベクトルロード VECTOR

31 26	25 21	20 16	15 7	6	5 0
011111	base	rt	000000000	s	111010
VFP			**0**		**LW**

ニーモニック:

VFLW rt, base (sync(s) = 0)
VFLW.sy rt, base (sync(s) = 1)

機能 :

VFPR[rt] ← MEM[GPR[base]]

例外 :

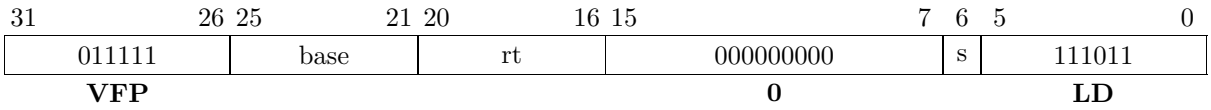
Vector Floating Point Exception :
D-TLB No Entry Matched :
D-TLB Protection Error :
Data Address Miss Align (Load) :

概要 :

メモリから浮動小数点ベクトルレジスタにロードする．sync により，投機実行を抑制する．

VFLD Vector Floating-Point Load Double

浮動小数点ベクトルロード VECTOR



ニーモニック:

VFLD rt, base (sync(s) = 0)
VFLD.sy rt, base (sync(s) = 1)

機能 :

VFPR[rt] ← MEM[GPR[base]]

例外 :

Vector Floating Point Exception :
D-TLB No Entry Matched :
D-TLB Protection Error :
Data Address Miss Align (Load) :

概要 :

メモリから浮動小数点ベクトルレジスタにロードする．sync により，投機実行を抑制する．

VFSW Vector Floating-Point Store Word

浮動小数点ベクトルストア VECTOR

31 26	25 21	20 16	15 7	6	5 0
011111	base	rt	000000000	s	111110
VFP			**0**		**SW**

ニーモニック:

VFSW rt, base (sync(s) = 0)
VFSW.sy rt, base (sync(s) = 1)

機能 :

MEM[GPR[base]] ← VFPR[rt]

例外 :

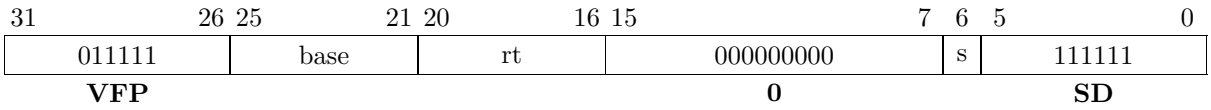
Vector Floating Point Exception :
D-TLB No Entry Matched :
D-TLB Protection Error :
Data Address Miss Align (Store) :

概要 :

浮動小数点ベクトルレジスタをメモリにストアする．sync を 1 にすることにより，投機実行を抑制する．

VFSD Vector Floating-Point Store Double

浮動小数点ベクトルストア VECTOR



ニーモニック:

VFSD rt, base (sync(s) = 0)
VFSD.sy rt, base (sync(s) = 1)

機能 :

MEM[GPR[base]] ← VFPR[rt]

例外 :

Vector Floating Point Exception :
D-TLB No Entry Matched :
D-TLB Protection Error :
Data Address Miss Align (Store) :

概要 :

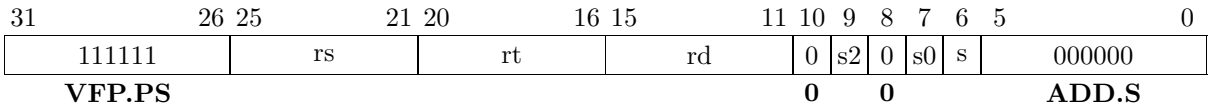
浮動小数点ベクトルレジスタをメモリにストアする．sync により，投機実行を抑制する．

VADD.PS

Vector Add Paired Single

ベクトル加算

VECTOR



ニーモニック:

VADD.PS.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VADD.PS.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VADD.PS.vv.lo32 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VADD.PS.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VADD.PS.vs.lo32 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VADD.PS.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VADD.PS.vv.lo32.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VADD.PS.vs.lo32.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能：

VFPR[rd] ← VFPR[rs] + VFPR[rt]

例外：

Vector Integer Exception :

概要：

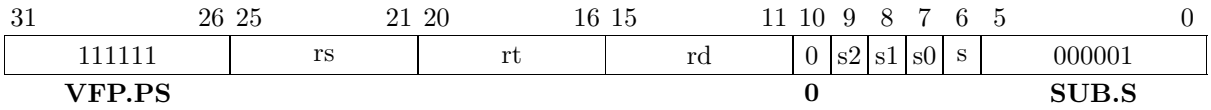
32bit × 2 単精度ベクトル加算．sc0 の場合は VFPR[rt] の代わりに，スカラレジスタ (SFPR[rt]) を用いて演算を行う．sc2 の場合 VFPR[rt] の下位 32bit を用いて演算を行う．sync により，投機実行を抑制する．

VSUB.PS

Vector Subtract Paired Single

ベクトル減算

VECTOR



ニーモニック:

VSUB.PS.vv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, scalar2(s2) = 0, sync(s) = 0)
VSUB.PS.vs rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, scalar2(s2) = 0, sync(s) = 0)
VSUB.PS.sv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, scalar2(s2) = 0, sync(s) = 0)
VSUB.PS.vv.lo32 rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, scalar2(s2) = 1, sync(s) = 0)
VSUB.PS.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, scalar2(s2) = 0, sync(s) = 1)
VSUB.PS.vs.lo32 rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, scalar2(s2) = 1, sync(s) = 0)
VSUB.PS.sv.lo32 rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, scalar2(s2) = 1, sync(s) = 0)
VSUB.PS.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, scalar2(s2) = 0, sync(s) = 1)
VSUB.PS.sv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, scalar2(s2) = 0, sync(s) = 1)
VSUB.PS.vv.lo32.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, scalar2(s2) = 1, sync(s) = 1)
VSUB.PS.vs.lo32.sy rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, scalar2(s2) = 1, sync(s) = 1)
VSUB.PS.sv.lo32.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VFPR[rd] ← VFPR[rs] - VFPR[rt]

例外 :

Vector Integer Exception :

概要 :

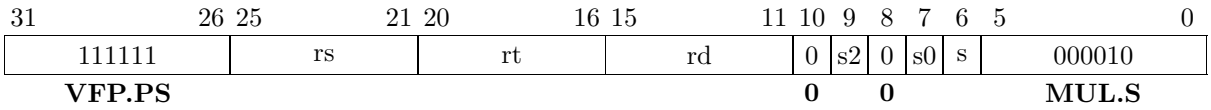
32bit × 2 単精度ベクトル減算. sc0 の場合は VFPR[rt] の代わりに，スカラレジスタ (SFPR[rt]) を用いて演算を行う．sc1 の場合は VFPR[rs] の代わりに，スカラレジスタ (SFPR[rs]) を用いて演算を行う．sc2 の場合 VFPR[rt] の下位 32bit を用いて演算を行う．sync により，投機実行を抑制する.

VMUL.PS

Vector Multiply Paired Single

ベクトル乗算

VECTOR



ニーモニック:

VMUL.PS.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMUL.PS.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMUL.PS.vv.lo32 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMUL.PS.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMUL.PS.vs.lo32 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMUL.PS.vs.sync rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMUL.PS.lo32.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMUL.PS.vs.lo32.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VFPR[rd] ← VFPR[rs] × VFPR[rt]

例外 :

Vector Integer Exception :

概要 :

32bit × 2 単精度ベクトル乗算．sc0 の場合は VFPR[rt] の代わりに，スカラレジスタ (SFPR[rt]) を用いて演算を行う．sc2 の場合 VFPR[rt] の下位 32bit を用いて演算を行う．sync により，投機実行を抑制する．

VABS.PS

Vector Absolute Paired Single

ベクトル絶対値演算

VECTOR

31 26	25 21	20 16	15 11	10 7	6	5 0
111111	rs	00000	rd	0000	s	000101
VFP.PS		**0**		**0**		**ABS.S**

ニーモニック:

VABS.PS rd, rs, rt　(sync(s) = 0)

VABS.PS.sy rd, rs, rt　(sync(s) = 1)

機能 :

VFPR[rd] ← |VFPR[rs]|

例外 :

Vector Integer Exception :

概要 :

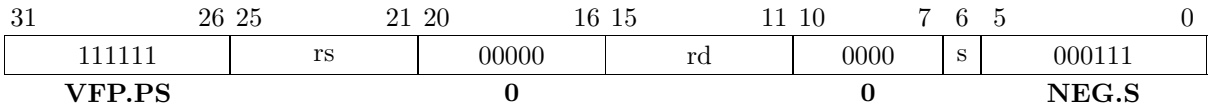
32bit × 2 単精度ベクトル絶対値演算．sync により，投機実行を抑制する．

VNEG.PS

Vector Negate Paired Single

ベクトル符号反転

VECTOR



ニーモニック:

VNEG.PS rd, rs, rt (sync(s) = 0)

VNEG.PS.sy rd, rs, rt (sync(s) = 1)

機能 :

VFPR[rd] ← -1 *times* VFPR[rs]

例外 :

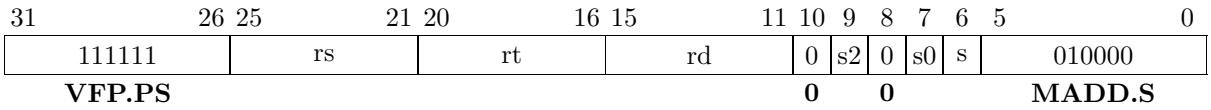
Vector Integer Exception :

概要 :

32bit × 2 単精度ベクトル符号反転演算．sync により，投機実行を抑制する．

VMADD.PS Vector Multiply and Add Paired Single

ベクトル積和演算 VECTOR



ニーモニック:

VMADD.PS.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMADD.PS.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMADD.PS.vv.lo32 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMADD.PS.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMADD.PS.vs.lo32 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMADD.PS.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMADD.PS.vv.lo32.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMADD.PS.vs.lo32.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VFPR[rd] ← VFPR[rs] *times* VFPR[rt] + VFPR[rd]

例外 :

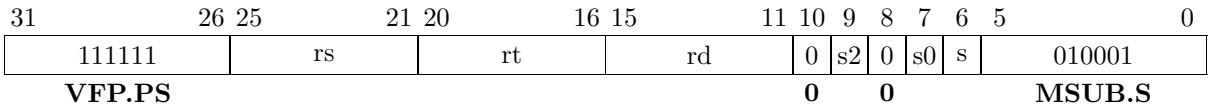
Vector Integer Exception :

概要 :

32bit × 2 単精度ベクトル積和演算．sc0 の場合は VFPR[rt] の代わりに，スカラレジスタ (SFPR[rt]) を用いて演算を行う．sc2 の場合は VFPR[rt] の下位 32bit を用いて演算を行う．sync により，投機実行を抑制する．

VMSUB.PS Vector Multiply and Subtract Paired Single

ベクトル積差演算 VECTOR



ニーモニック:

VMSUB.PS.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMSUB.PS.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMSUB.PS.vv.lo32 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMSUB.PS.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMSUB.PS.vs.lo32 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMSUB.PS.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMSUB.PS.vv.lo32.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMSUB.PS.vs.lo32.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能 :

VFPR[rd] ← VFPR[rs] × VFPR[rt] - VFPR[rd]

例外 :

Vector Integer Exception :

概要 :

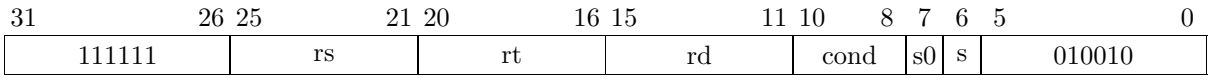
32bit × 単精度ベクトル積差演算. sc0 の場合は VFPR[rt] の代わりに, スカラレジスタ (SFPR[rt]) を用いて演算を行う. sc2 の場合は VFPR[rt] の下位 32bit を用いて演算を行う. sync により, 投機実行を抑制する.

VCMP.PS

Vector Compare Paired Single

ベクトル比較

VECTOR



VFP.PS **CMP.S**

ニーモニック:

VCMP.cond.PS.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMP.cond.PS.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMP.cond.PS.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMP.cond.PS.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VFPR[rd] ← VFPR[rs] cond VFPR[rt]

例外 :

Vector Integer Exception :

概要 :

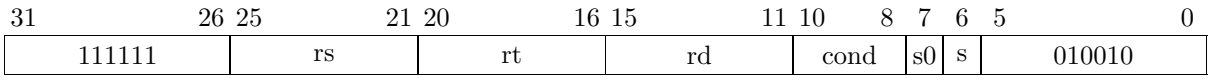
32bit × 2 単精度ベクトル比較命令．条件 (cond) により VFPR[rd] の値が決定する．sc0 の場合は VFPR[rt] の代わりに，スカラレジスタ (SFPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VSCMP.PS

Vector Compare Paired Single

ベクトル比較

VECTOR



VFP.PS **SCMP.S**

ニーモニック:

VSCMP.cond.PS.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VSCMP.cond.PS.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VSCMP.cond.PS.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VSCMP.cond.PS.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

VFPR[rd] ← VFPR[rs] cond VFPR[rt]

例外 :

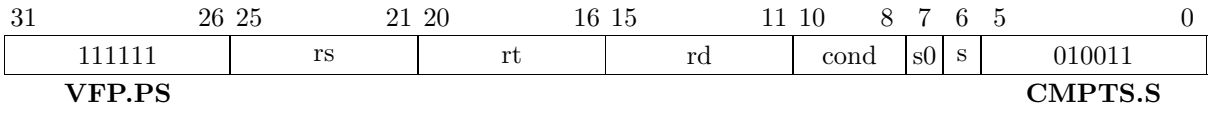
Vector Integer Exception :

概要 :

32bit × 2 単精度ベクトル比較命令. 条件 (cond) により VFPR[rd] の値が決定する. VFPR[rt] の下位 32bit を用いて演算を行う. sc0 の場合は VFPR[rt] の代わりに, スカラレジスタ (SFPR[rt]) を用いて演算を行う. sync により, 投機実行を抑制する.

VCMPTS.PS Vector Compare Paired Single to Scalar Register

ベクトル比較 VECTOR



ニーモニック:

VCMPTS.cond.PS.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMPTS.cond.PS.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMPTS.cond.PS.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMPTS.cond.PS.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

SFPR[rd] ← VFPR[rs] cond VFPR[rt]

例外 :

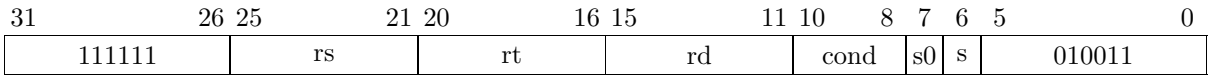
Vector Integer Exception :

概要 :

32bit × 2 単精度ベクトル比較命令．結果は各要素ごとに 1bit を割り当ててスカラレジスタに格納される．sc0 の場合は VFPR[rt] の代わりに，スカラレジスタ (SFPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

VSCMPTS.PS Vector Compare Paired Single to Scalar Register

ベクトル比較 VECTOR



VFP.PS **SCMPTS.S**

ニーモニック:

VSCMPTS.cond.PS.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VSCMPTS.cond.PS.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VSCMPTS.cond.PS.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VSCMPTS.cond.PS.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能 :

SFPR[rd] ← VFPR[rs] cond VFPR[rt]

例外 :

Vector Integer Exception :

概要 :

32bit × 2 単精度ベクトル比較命令．結果は各要素ごとに 1bit を割り当ててスカラレジスタに格納される．VFPR[rt] の下位 32bit を用いて演算を行う．sc0 の場合は VFPR[rt] の代わりに，スカラレジスタ (SFPR[rt]) を用いて演算を行う．sync により，投機実行を抑制する．

4

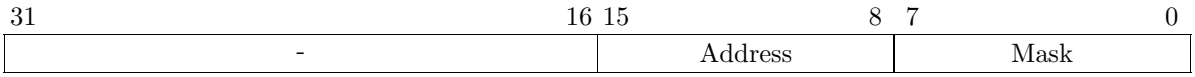
アドレスデコーダ

4.1 レジスタインターフェース

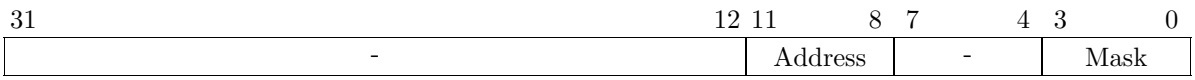
アドレスデコーダの各種設定レジスタはシステムレジスタとして定義されている．そのため各種レジスタを設定するためには mtc0 命令を用いる．

設定する値はモジュールにより異なる．

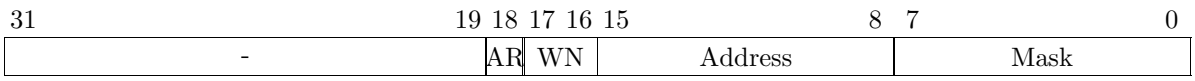
- 標準 (TYPE A)



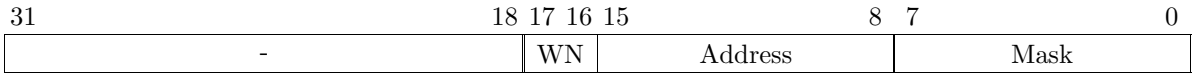
- I/O (TYPE B)



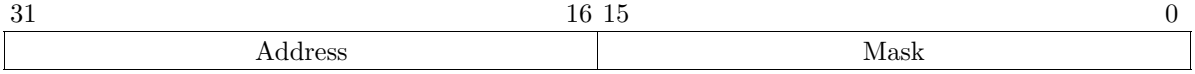
- 外部バス (TYPE C)



- リンク メモリ (TYPE D)



- I/O Base (TYPE E)



bit 名	機能
Address	ベースアドレス
Mask	マスク
WN	Word Number
AR	Auto Ready

4.2 アドレスマップ

接続されるモジュール	初期デコードアドレス	設定レジスタのアドレス	タイプ
ROM (EXT_0)	0x00000000 ～ 0x00ffffff	0xa0	TYPE C
EXT_1	0x20000000 ～ 0x20ffffff	0xa1	TYPE C
SDRAM IF0	0x80000000 ～ 0x87ffffff	0xa2	TYPE A
SDRAM IF1	0x88000000 ～ 0x8fffffff	0xa3	TYPE A
SDRAM IF2	0x90000000 ～ 0x97ffffff	0xa4	TYPE A
SDRAM IF3	0x98000000 ～ 0x9fffffff	0xa5	TYPE A
LINK SDRAM	0x04000000 ～ 0x04ffffff	0xa6	TYPE D
LINK DPM	0xc0000000 ～ 0xcfffffff	0xa7	TYPE D
DMAC0	0xffff0000 ～ 0xffff0fff	0xa8	TYPE B
DMAC1	0xffff1000 ～ 0xffff1fff	0xa9	TYPE B
DMAC2	0xffff2000 ～ 0xffff2fff	0xaa	TYPE B
PCI	0xffff3000 ～ 0xffff3fff	0xab	TYPE B
USB In	0xffff4000 ～ 0xffff4fff	0xac	TYPE B
USB Out	0xffff5000 ～ 0xffff5fff	0xad	TYPE B
UART	0xffff6000 ～ 0xffff6fff	0xae	TYPE B
PP	0xffff7000 ～ 0xffff7fff	0xaf	TYPE B
IEEE1394	0xffff8000 ～ 0xffff8fff	0xb0	TYPE B
LINK	0xfffe0000 ～ 0xfffeffff	0xb1	TYPE E
IRC	0xffff9000 ～ 0xffff9fff	0xb2	TYPE B
CLK Generator	0xffffa000 ～ 0xffffafff	0xb3	TYPE B
MDMAC	0xffffd000 ～ 0xffffdfff	0xb4	TYPE B
LINK SDRAM Mode	0xffffe000 ～ 0xffffefff	0xb5	TYPE B
SDRAM Mode	0xfffff000 ～ 0xffffffff	0xb6	TYPE B
I/O Base	0xffff0000 ～ 0xffffffff	0xb8	TYPE E

5

MMU

Responsive Multithreaded Processor のキャッシュシステムは命令キャッシュ，データキャッシュともに物理キャッシュなので MMU でのアドレス変換はプロセッシングコアとキャッシュの間で行う．また，アドレス空間上に，TLB によるアドレス変換が行なわれない領域は存在しない．

5.1 TLB エントリ

本 MMU における TLB エントリ数は命令 MMU, データ MMU ともに 64 エントリである．エントリへの設定方法は full associative 方式とし，設定を行うページ番号に関わらずどのエントリでも設定を行うことを可能である．

以下本 MMU における TLB エントリの機能の詳細と特徴について述べて行く．

表 5.1 に TLB エントリの設定項目の一覧を示す．TLB エントリは全部で 8byte であるが，設定に際しては 32bit の整数型データを用いて行うため便宜上エントリ 1 とエントリ 2 に分かれる．

また TLB エントリに指定した仮想アドレスとコンテキスト ID が一致したかどうかの判断はエントリ番号の大きな順に行われる．そのため複数のエントリが一致した場合には，よりエントリ番号が大きな TLB エントリの設定値を用いてアドレス変換が行われる．

TLB のエントリを設定した直後，そのエントリの LRU 情報はもっとも最近に参照されたものとして扱われる．

TLB エントリを初期化した場合，各フィールドの値は表 5.2 のようになる．また，TLB エントリの LRU 情報を初期化すると LRU の順序はエントリ 0 がもっとも最近アクセスされた tlb エントリとなり，エントリ 1,2,3,... 62, 63 と順にアクセスが古い，という状態になる．

VPN

この VPN フィールドには，アドレス変換を行う仮想アドレスがエントリを検索するために必要な仮想ページ番号を保持する．Responsive Multithreaded Processor は仮想アドレスに 32bit の信号を用い，最小ページサイズが 4KB であるため，TLB エントリには表 5.1 に示すように仮想アドレスの上位 20bit を保持する．VPN フィールドはエントリ 1 に属し，エントリ設定時には設定を行う仮想ページ番号を設定データの上位 20bit に指定する．

表 5.1: TLB エントリ一覧

フィールド名	エントリ番号	データ割り当て	機能
VPN	エントリ 1	[31:12]	仮想ページ番号 (Virtual Page Number)
LOCK	エントリ 1	[11]	エントリのロック
PROTECT	エントリ 1	[10:8]	保護情報
SHARE_TH	エントリ 1	[7:0]	エントリの有効情報と共有情報
PPN	エントリ 2	[31:12]	物理ページ番号 (Physical Page Number)
PSZ	エントリ 2	[11:10]	ページサイズ
GROUP	エントリ 2	[9:4]	コンテキストグループ番号
CACHE_LOCK	エントリ 2	[3]	該当ページのキャッシュでのロック
UNCACHE	エントリ 2	[2]	該当ページのキャッシュ不可
BURST	エントリ 2	[1:0]	内部バスアクセス時のバースト転送長

表 5.2: TLB の初期化時の値

フィールド名	初期化時の値
VPN	全 bit 0
LOCK	0 (ロックオフ)
PROTECT	000 (KER_R モード)
SHARE_TH	11111111 (全コンテキスト無効)
PPN	全 bit 0
PSZ	00 (4K Byte)
GROUP	全 bit 0
CACHE_LOCK	0 (ロックオフ)
UNCACHE	0 (無効)
BURST	11 (バーストなし)

LOCK

TLB ミスが起こると，そのミスを起こした仮想アドレスを変換するための新たな設定を TLB エントリに行う必要がある．全てのエントリがすでに使われていると，いずれかのエントリを選択して設定の入れ換えを行わなければならない．本 MMU では各 TLB エントリへのアクセス情報を記録した LRU 情報を用い，最もアクセスがなされていないエントリを入れ換えの対象とする．

この LOCK フィールドを設定する (1 にする) と，その TLB エントリを LRU 情報を用いた入れ換えの対象から外すことができる．ただし設定を行うエントリを直接指定した場合にはこの LOCK フィールドの設定は無効となる．

また，LOCK フィールドが設定された TLB エントリを使ってアドレス変換を行った場合，その TLB エントリはもっとも最近に参照されたものとして LRU 情報が更新される．

この LOCK フィールドはエントリ 1 に属する．

PROTECT

ページ単位でのメモリ領域の保護を行うため，この PROTECT フィールドにそのための保護情報を指定する．表 5.3 に指定可能な保護情報の一覧を示す．

尚 Responsive Multithreaded Processor では制限の厳しい順にカーネル・スーパバイザー・ユーザの 3 つのスレッドの動作モードが規定されているが，モードフィールドとして 2bit が利用可能であるため，本フィールドにはユーザモードよりも更に制限の緩い場合を設定できる．

この PROTECT フィールドはエントリ 1 に属する．

SHARE_TH

Responsive Multithreaded Processor は同時に最大 8 個のスレッドが動作するため，TLB エントリのミス率が高くなってしまう．ミス率を少しでも低く抑えるために，コンテキスト毎に TLB エントリに有効情報を持つようにする．Responsive Multithreaded Processor はスレッド ID(32bit) ではなくコンテキスト ID(3bit) を用いてスレッドの実行を制御している．特にコンテキストが有効かどうかは各コンテキストにつき 1bit の情報で与えられるので，TLB エントリにはそれに対応する bit を用意している．この SHARE_TH フィールドには，各コンテキストの有効情報を保持する．エントリの有効情報は仮想アドレスの比較に用いられるだけではなく，エントリの共有と入れ換えエントリの選択にも用いる．

入れ替えを行う TLB エントリは特に指定がなければ LRU 情報に基づいて選択されるが，その前に SHARE_TH フィールドを調べて無効なエントリが存在する場合はそれを入れ替えの対象とする．

また，有効であったコンテキストが無効化され，かつそのコンテキストの MMU でのアドレス変換が有効であった場合には，自動的にこの SHARE_TH フィールドは無効に設定される．

このフィールドはエントリ 1 に属する．

PPN

この PPN フィールドには，VPN フィールドの仮想ページ番号がマップされている物理ページ番号が保持される．Responsive Multithreaded Processor は物理アドレスに 32bit の信号を用い，最小ページサイズが 4KB であるため，TLB エントリで変換される物理アドレスは表 5.1 に示すように上位 20bit である．PPN フィールドはエントリ 2 に属し，エントリ設定時には設定を行う物理ページ番号を設定データの上位 20bit に指定する．

表 5.3: TLB エントリに指定可能なページ保護情報

保護モード	設定コード	保護の詳細
ALL_RW	111	全モードでの読み出しと書き込みを許可
ALL_R	110	全モードでの読み込み，ユーザモード以上での書き込みを許可
USR_RW	101	ユーザモード以上での読み出しと書き込みを許可
USR_R	100	ユーザモード以上の読み込み，スーパバイザーモード以上の書き込みを許可
SV_RW	011	スーパバイザーモード以上での読み出しと書き込みを許可
SV_R	010	スーパバイザーモード以上の読み込み，カーネルモードでの書き込みを許可
KER_RW	001	カーネルモードでの読み出しと書き込みを許可
KER_R	000	カーネルモードでの読み込みのみを許可

PSZ

Responsive Multithreaded Processor では，複数ページサイズのサポートをしている．TLB エントリでも複数のページサイズを用いることができるようにしており，表 5.4 に指定可能なページサイズを示す．

表 5.4: TLB エントリに指定可能なページサイズ

ページサイズ	設定コード
4K byte	00
64K byte	01
1M byte	10
16M byte	11

この PSZ フィールドはエントリ 2 に属する．

GROUP

Responsive Multithreaded Processor ではコンテキスト ID を用いた制御が行なわれるため，特に一度コンテキストキャッシュに退避されたスレッドが実行を再開する場合には，退避前の TLB エントリの設定値は全く使うことができない．これはコンテキスト単位で仮想アドレスを識別しているために，実行スレッドが切り替わる際にはどうしても無効化しなければならないからである．各々のスレッドのアドレスマップは通常独立であるから，エントリの無効化は問題にはならない．しかし共有メモリ領域のエントリでは，退避前までエントリを共有していたスレッドが，実行再開後は全く別のエントリに設定しなければならなくなる．

そこでそのような無駄を省くためにこの GROUP フィールドを用いる．各 TLB エントリはこのフィールドに設定された ID を用いて有効情報の変更が可能になっている (5.2)．そこで共有メモリ領域を持つスレッドは，その領域の TLB エントリに設定されたコンテキストグループ番号を知っていれば，実行を再開したコンテキスト番号をそのコンテキストグループに属する TLB エントリに通知するだけで，容易に TLB エントリの有効化を行なうことができる．

この GROUP フィールドはエントリ 2 に属する．

CACHE_LOCK

このフィールドを有効にすることで，該当ページのデータブロックをキャッシュ上にロックすることができる．機能は TLB エントリのロックフィールド (5.1) と同等である．

この CACHE_LOCK フィールドはエントリ 2 に属し，MMU が無効状態でのデフォルトの値は無効 (ロック不可) となる．

UNCACHE

この UNCACHE フィールドを有効 (1 と設定) にすると，そのページのブロックはキャッシュされない．キャッシュシステム内にはキャッシュメモリ本体以外にもデータが置かれるバッファがいくつかあるが，このフィールドが有効になっているページのデータは，書き込みデータのマージ機構 (6.1.4) や内部バス要求キューでの複数データヒット機構 (6.1.4)，victim buffer (6.1.3) が無効になる．

このフィールドはエントリ 2 に属し，MMU が無効状態でのデフォルトの値は有効 (キャッシュ不可) となる．

BURST

アクセスしたいデータがキャッシュミスとなると，内部バスへ要求を出すことになる．この BURST フィールドには，その場合の内部バスに対するバースト読み出しの転送長を指定する．設定可能な転送長を表 5.5 に示す．

表 5.5: TLB エントリで指定可能な内部バスのバースト転送長

転送長	設定コード	転送データ量
無し	11	32byte
2	10	64byte
4	01	128byte
8	00	256byte

このフィールドの値は書き込み要求には適応されない．また I/O など 32bit バスのデータ読み出しにも適応されない．I/O であるかどうかはアドレス空間を用いて識別する．

このフィールドはエントリ 2 に属し，MMU が無効状態でのデフォルトの値はバースト転送無しとなる．

5.2 MMU の制御

MMU のコントロールレジスタの一覧を表 5.7 に示す．

各レジスタは通常のアドレス空間や，プロセッシングコアのコントロールレジスタのアドレス空間とは異なる独自のアドレス空間にマッピングされている．そのため MMU のコントロールレジスタへのアクセスは表 5.6 に示す 4 つの専用命令を用いて行う．

表 5.6: MMU のコントロールレジスタアクセス用命令

命令	用途
MFIMM	命令用 MMU のコントロールレジスタの値を読み出す
MTIMM	命令用 MMU のコントロールレジスタに値を設定
MFDMM	データ用 MMU のコントロールレジスタの値を読み出す
MTDMM	データ用 MMU のコントロールレジスタに値を設定

コントロールレジスタの値の設定方法には，設定するデータをそのまま指定するものと，一定の形式に合わせて指定するものがある．後者の一定の形式は MMU のコントロールレジスタの設定のみならず，キャッシュコントローラのコントロールレジスタの設定にも用いられる場合がある (6.1.5)．そこでこの一定の形式のことを**共通設定形式** (図 5.1) と呼ぶことにする．共通設定形式では 1 を設定することで該当レジスタの機能を有効化することができる．またコンテキストグループ (5.1) に対してはこの共通設定形式を拡張した独自の形式 (図 5.4) を用いて設定を行う．

コントロールレジスタの設定のタイミングはプロセッサの実行状況によって全く異なるため，実行中のスレッドに対してページ設定以外の設定情報の変更 (MMU のオン・オフやエントリのフラッシュ) を行う場合は注意が必要である．

またコントロールレジスタの多くは設定要求 (書き込み要求) のみを規定しており，そのようなレジスタに対する読み出し要求には返戻値として 0 が返る．

S	select field	1に設定すると該当コンテキストを撰択
V	value field	1に設定すると有効化、0に設定すると無効化

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
context7		context6		context5		context4		context3		context2		context1		context0	
S	V	S	V	S	V	S	V	S	V	S	V	S	V	S	V

図 5.1: コントロールレジスタの共通設定形式

表 5.7: MMU のコントロールレジスタ一覧

アドレス [7:0]	レジスタ名	設定方法	機能
0x00	MMU_SPR_START	共通形式	アドレス変換の有効・無効
0x04	MMU_SPR_ALL_FLUSH	設定値無し	エントリの無効化と LRU 情報の初期化
0x08	MMU_SPR_TLB_FLUSH	直接指定	指定したエントリを無効化
0x0c	MMU_SPR_THREAD_FLUSH	共通形式	指定したコンテキストを無効化
0x10	MMU_SPR_GROUP_FLUSH	直接指定	指定したグループを全て無効化
0x14	MMU_SPR_LRU_FLUSH	直接指定	LRU 情報を初期化
0x18	MMU_SPR_MAX_LOCK	直接指定	エントリをロックできる最大数
0x1c	MMU_SPR_ENTRY1	直接指定	TLB エントリのエントリ 1 を設定
0x20	MMU_SPR_ENTRY2	直接指定	TLB エントリのエントリ 2 を設定
0x24	MMU_SPR_ENTRY_INDEX	設定値無し	指定した TLB エントリを設定
0x28	MMU_SPR_ENTRY_LRU	設定値無し	LRU 情報によりエントリを設定
0x2c	MMU_SPR_GROUP	特殊形式	指定したグループのエントリの有効化・無効化
0x30	MMU_SPR_EXP_ADDR	設定値無し	例外を発生したアドレス
0x34	MMU_SPR_EXP_LOG	設定値無し	発生した例外の詳細情報

MMU_SPR_START

MMU_SPR_START レジスタは MMU 機能の有効・無効を示す.

Responsive Multithreaded Processor はスレッド毎ではなくコンテキスト毎に制御を行うため, MMU_SPR_START レジスタもコンテキスト毎に用意されている. コンテキストの指定は書き込みデータの下位 8bit[7:0] で行う.

またこのレジスタの読み出し要求に対しては, データの下位 8bit[7:0] の上位から順番にコンテキスト番号 7 からコンテキスト番号 0 までの設定値が格納される.

設定には図 5.1 に示した共通設定形式を用いる.

MMU_SPR_ALL_FLUSH

このレジスタに対して書き込み要求を行うと, 全ての TLB エントリの設定データとエントリアクセスの LRU 情報を初期化する. 書き込むデータに制約はなく, 設定を行うと次クロックで自動的にクリアされる.

MMU_SPR_TLB_FLUSH

このレジスタに指定した番号の TLB エントリのみを初期化する．TLB エントリの指定は書き込みデータの下位 6bit[5:0] で行う．設定を行うと次クロックで自動的にクリアされる．

MMU_SPR_THREAD_FLUSH

各 TLB エントリにおいてこのレジスタに指定したコンテキストのみを無効化する．設定方法は図 5.1 に示した共通設定形式を用いる．設定を行うと次クロックで自動的にクリアされる．

MMU_SPR_GROUP_FLUSH

このレジスタに指定したコンテキストグループに属する TLB エントリを無効化する (5.1)．コンテキストグループの指定は書き込みデータの下位 6bit[5:0] で行う．設定を行うと次クロックで自動的にクリアされる．

MMU_SPR_LRU_FLUSH

このレジスタに書き込み要求を行うと，TLB エントリのアクセスに関する LRU 情報を初期化する．書き込むデータに制約はない．設定を行うと次クロックで自動的にクリアされる．

MMU_SPR_MAX_LOCK

このレジスタには TLB エントリをロックし，LRU 情報によるエントリの入れ換え対象計算から外すことのできるエントリ数を設定する．既定値は 16 エントリ，最小値は 0 エントリ，最大値は 63 エントリであり，このレジスタの値以上のエントリをロックすることは基本的にできない．しかしロックエントリ数の計算に 1 クロック要するため，エントリの設定命令が 2 クロック連続するとこのレジスタの値を越えてロックが設定される可能性がある．全 64 エントリがロックされてしまった場合，ページフォルト発生時にページテーブルを設定できなくなってしまう．そのため，64 エントリがロックされると例外を発生させる (5.3)．MMU_SPR_ENTRY_LRU でエントリ 1 の LOCK フィールドを 1 にした TLB エントリをセットする時に MMU_SPR_MAX_LOCK の値以上にセットしようとした場合，ロックはされないが TLB エントリのほかの内容はセットされる．例えば，MMU_SPR_MAX_LOCK が 16 で既にロックされている TLB エントリも 16 個存在する時に MMU_SPR_ENTRY_LRU でロックしたエントリをセットする場合，LOCK フィールドを 0 にして TLB エントリをセットすることになる．

このレジスタの読み出し要求に対しては，データの下位 6bit[5:0] に現在の設定値を格納する．

MMU_SPR_ENTRY1

各 TLB エントリが持つエントリフィールドのうち，このレジスタには仮想アドレス，エントリのロック指定，ページ保護情報，エントリ共有情報を設定する．設定形式を図 5.2 に示す．

MMU_SPR_ENTRY2

各 TLB エントリが持つエントリフィールドのうち，このレジスタには物理アドレス，ページサイズ，コンテキストグループ，該当ページのキャッシュでのロックの可否，該当ページのキャッシュの可否，該当ページのバスアクセス時のバースト転送長を指定する．設定形式を図 5.3 に示す．

仮想ページ番号 : VPN　ページ保護情報 : PRO
エントリロック : LCK　共有情報 : SHR

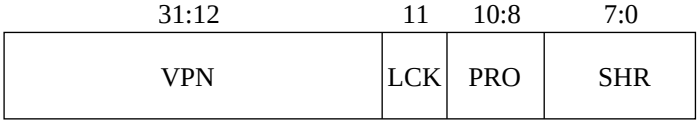


図 5.2: ENTRY1 の設定形式

物理ページ番号 : PPN　キャッシュロック : CLC
ページサイズ : PSZ　キャッシュ不可 : UNC
コンテキストグループ : GRP　バースト転送長 : BRT

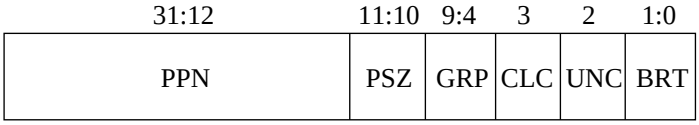


図 5.3: ENTRY2 の設定形式

MMU_SPR_ENTRY_INDEX

このレジスタに書き込み要求を行い TLB エントリ番号を指定することで，事前に設定しておいた MMU_SPR_ENTRY1 フィールドと MMU_SPR_ENTRY2 フィールドの値を，その指定された TLB エントリに設定する．TLB エントリの指定は書き込むデータの下位 6bit[5:0] で行う．

MMU_SPR_ENTRY_LRU

このレジスタに書き込み要求を行うことで，事前に設定しておいた MMU_SPR_ENTRY1 フィールドと MMU_SPR_ENTRY2 フィールドの値を，LRU 情報を元にして最もアクセスがなされていない TLB エントリに設定する．書き込むデータに制約はない．

MMU_SPR_GROUP

このレジスタに書き込み要求を行うことで，指定したコンテキストグループに所属する TLB エントリの，指定したコンテキストの有効化・無効化を行う (5.1)．コンテキストグループとコンテキストの指定形式を図 5.4 に示す．

MMU_SPR_EXP_ADDR

このレジスタはアドレス変換において該当する TLB エントリが存在しなかった場合に，その仮想アドレスを保持する．アドレスはコンテキスト毎に保持され，その値を読み出すにはデータの下位 3bit[2:0] にコンテキスト番号を指定する．

S	select field	1に設定すると該当コンテキストを撰択
V	value field	1に設定すると有効化、0に設定すると無効化

21 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
コンテキストグループ番号	context7	context6	context5	context4	context3	context2	context1	context0
	S V	S V	S V	S V	S V	S V	S V	S V

図 5.4: コンテキストグループの設定形式

このレジスタに対する書き込み要求は，実行したコンテキストに対応するレジスタの値がクリアされるだけである．

MMU_SPR_EXP_LOG

このレジスタにはページ保護の違反が確認された時にその違反コード (表 5.8) が保持される．

表 5.8: MMU のページ保護違反コード

コード名	違反コード	違反内容
MMU_EXP_NONE	000	違反無し
MMU_EXP_PRO_ALL_R	001	全モードでの書き込み制限違反
MMU_EXP_PRO_USR_RW	010	ユーザモード以上に限定されたページへのアクセス違反
MMU_EXP_PRO_USR_R	011	ユーザモード以上に限定されたページへの書き込み違反
MMU_EXP_PRO_SPV_RW	100	スーパバイザーモード以上に限定されたページへのアクセス違反
MMU_EXP_PRO_SPV_R	101	スーパバイザモード以上に限定されたページへの書き込み違反
MMU_EXP_PRO_KER_RW	110	カーネルモード以上に限定されたページへのアクセス違反
MMU_EXP_PRO_KER_R	111	カーネルモード以上に限定されたページへの書き込み違反

このレジスタの読み出し要求に対しては，4bit 目に違反発生の有無が (1 で保護違反発生)，下位 3bit[2:0] に違反コードが格納される．

またこのレジスタに対する書き込み要求は，実行したコンテキストに対応するレジスタの値がクリアされるだけである．

5.3 MMUが発生させる例外

本 MMU が発生させる例外を表 5.9 に示す．

命令用 MMU，データ用 MMU 共に発生させる例外の種類は同じであるが，命令とデータの区別をつけて例外を扱う．

また命令要求が発生させた例外とデータ要求が発生させた例外とではプロセッシングコアでの扱いが異なるが，MMU に対して設定を行うのはデータ要求であるため，命令用 MMU で発生した設定用の例外 (表 5.9 の例外の種類の設定) はデータ用 MMU の例外コードと一緒に扱われる．

表 5.9: MMU の発生させる例外

例外名	例外の種類	命令用 MMU のコード	データ用 MMU のコード
コントロールミス	設定	0x1	0x6
全エントリロック	設定	0x2	0x7
エントリミス	要求	0x3	0x8
モード違反	設定	0x4	0x9
ページ保護違反	要求	0x5	0xa

コントロールミス

どのコントロールレジスタにもマップされていないアドレスを用いてアクセスを行なった場合に発生する．このアドレスミスによってコントロールレジスタが影響を受けることはない．また MMU の状態が変化することもなく，ただ例外の発生を通知するだけである．この例外が発生したとき，コントロールレジスタである MMU_SPR_EXP_ADDR, MMU_SPR_EXP_LOG の値は変化しない．

全エントリロック

全てのエントリがロック指定された時に発生する．一度この例外が発生すると MMU はロックオーバー状態になる．この間もアドレス変換や TLB エントリの設定は可能であるが，LRU 情報による TLB エントリの設定は常に最も番号が若い TLB エントリ 0 に対してのみ行われる．ロックオーバー状態は TLB エントリのロック数が 63 以下になると自動的に解除される．この例外が発生したとき，コントロールレジスタである MMU_SPR_EXP_ADDR, MMU_SPR_EXP_LOG の値は変化しない．

エントリミス (TLB ミス)

命令要求やデータ要求によって指定された仮想アドレスとコンテキスト ID が，どのエントリの設定値とも一致しなかった場合に発生する．例外を起こした仮想アドレスをコントロールレジスタに保持するが (5.2)，MMU の状態は変化しない．例外発生後もアドレス変換や TLB エントリの設定は通常通り可能である．例外発生時，MMU_SPR_EXP_LOG の値は 0(違反なし) に設定される．

尚命令用 MMU で本例外が発生した場合，命令フェッチユニットの仕様により命令を返さなければ例外処理に進むことができないため，フェッチ命令幅の全てを No-op コードとして返す．

モード違反

MMU の設定を行うスレッドはカーネルモードがスーパバイザーモードでなければならない．この例外はユーザ以下のモードで MMU のコントロールレジスタに要求を行なった場合に発生する．本例外によるコントロールレジスタへの影響はなく，MMU の状態が変化することもない．この例外が発生したとき，コントロールレジスタである MMU_SPR_EXP_ADDR, MMU_SPR_EXP_LOG の値は変化しない．

ページ保護違反

命令要求やデータ要求によって指定された仮想アドレスとコンテキスト ID が一致した TLB エントリにおいて，その要求が設定された保護情報に反する場合に発生する．コントロールレジスタには例外コード (表 5.8) が格納されるが，MMU の状態は変化しない．

命令用 MMU でこの例外が発生した場合，エントリミスと同様に命令フェッチ幅の全てを No-op として返す.

6

CACHE

6.1 キャッシュシステム

6.1.1 概要

Responsive Multithreaded Processor のキャッシュシステムの特徴を以下に示す．またモジュール構成を図 6.1 に示す．本節ではこれらキャッシュシステムを構成する各要素について述べる．

- 32 KB 8-way set-associative 方式
- ブロックサイズ，ラインサイズともに 32byte
- Look Through
- ノンブロッキング
- 下位メモリとのデータ一貫性の維持はライトバック方式
- 書き込み要求ミスの処理はライトアロケート方式
- キャッシュポートは 1 ポート
- 物理タグでデータを保持
- 転送ブロック数が可変
- キャッシュのロックが可能
- 3 サイクルのアクセス遅延
- マルチタグ，シングルデータ方式
- 16 エントリの victim buffer
- 最大 16 個のキャッシュミスを同時に保持
- 入れ換えを行うブロックの選択方法は LRU と優先度の 2 通り
- バス待ちキューでの優先度による要求の追い越しが可能

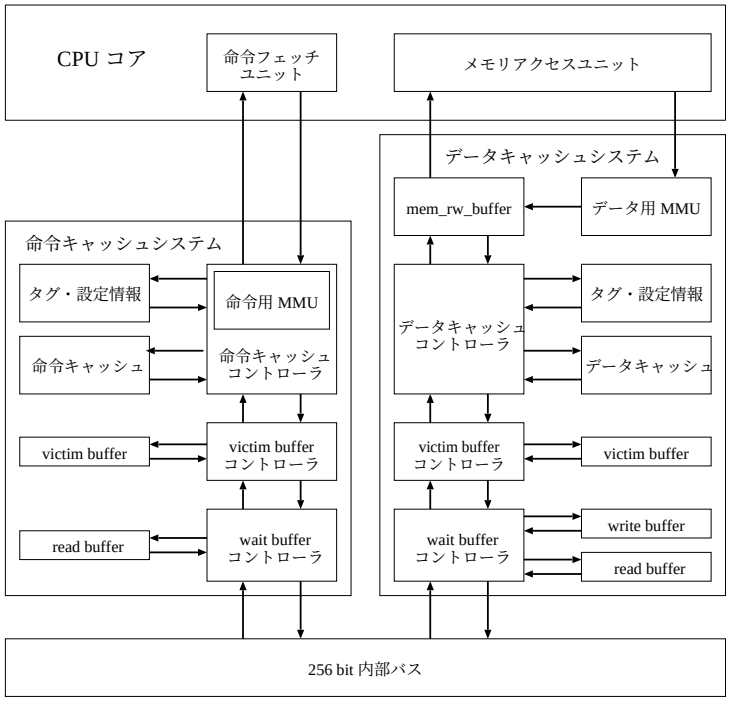


図 6.1: キャッシュシステムのモジュール構成

6.1.2 キャッシュ制御

キャッシュの制御は図 6.1 の命令キャッシュコントローラ，データキャッシュコントローラで行う．

キャッシュ要求がキャッシュミスを起こした場合には，先に victim buffer で保持されているデータと比較され (6.1.3)，そこでも該当データを発見できなければ wait buffer から内部バスへアクセスを行う (6.1.4)．

キャッシュでのデーター貫性の維持

命令，データの両キャッシュコントローラは，内部バスで発生する書き込み要求を常に監視する．そしてもしキャッシュしているデータが書き込みを受けた場合にそのデータを無効化する．

6.1.3 victim buffer

victim buffer では，キャッシュブロックの入れ換えに伴いキャッシュメモリを追い出されたデータを，full associative 方式でエントリに保持する．そしてキャッシュミスを起こした要求のアドレスを現在保持しているデータのタグと比較し，もし一致するデータがあれば該当データをキャッシュへと送り込む．

6.1.4 wait buffer

概要

wait buffer は内部バス要求キュー，read buffer とその管理機構からなるキャッシュコントローラの内部バスインタフェースであり，命令用とデータ用のそれぞれに分かれる．データキャッシュ用の wait buffer には更に write buffer とその管理機構が付随する．

内部バス要求

内部バス要求キューと write buffer は 16 エントリから成り，victim buffer から送られてくる様々な要求を順にエントリに格納して行く．

内部バス要求キューの要求順位入れ換え機能

内部バス要求キューの要求順位を入れ替える機構がある．その方法は読み出し要求を書き込み要求よりも優先して行う方法と，優先度が高いコンテキストの要求を優先して行う方法の 2 種類である．ただし，ライトアロケート方式を用いているため通常の書き込み要求は読み出し要求と同じくデータの読み出しを行うため，追い抜きの対象は write back 要求になっている．

書き込み要求のマージ機能

通常の書き込み要求のデータは 1 byte の文字型や 4 byte の整数型， 8 byte の倍精度浮動点小数型のデータであるため，1 キャッシュラインのデータ幅である 32 byte に対しては小さい．よって同じキャッシュラインに対するデータの書き込みは 1 つのエントリにまとめることができるようにしている．

ただし I/O への書き込み要求であった場合には，データのマージは行わない．

6.1.5 キャッシュのコントロールレジスタ

キャッシュのコントロールレジスタの一覧を表 6.1 に示す．これらのレジスタはプロセッシングコアのコントロールレジスタと同じアドレス空間にマッピングされており，それらと同じ命令 (表 6.2) を用いてアクセスする．尚これらのレジスタの設定方法は TLB のコントロールレジスタの設定 (5.2) と同じようにデータを直接指定するか，図 5.1 に示した共通設定形式を用いる．

表 6.1: キャッシュのコントロールレジスタ

レジスタ名	設定方法	機能	命令用アドレス [7:0]	データ用アドレス [7:0]
ON	共通形式	キャッシュの有効・無効	0x80	0x86
REP_MODE	直接指定	入れ換え方法の指定	0x81	0x87
ACC_SCHE	直接指定	要求の追い越し指定	0x82	0x88
LOCK	共通形式	ロックの有効・無効	0x83	0x89
RESET	直接指定	キャッシュのリセット	0x84	0x8a
FLUSH	共通形式	write back の指定	無し	0x8b
ALL_FLUSH	直接指定	全て write back	無し	0x8c

表 6.2: コントロールレジスタをアクセスする命令

命令	用途
MFC0	コントロールレジスタの値を読み出す
MTC0	コントロールレジスタに値を設定

ON

このレジスタを設定することで，コンテキスト毎にキャッシュの有効・無効を指定できる．
初期状態の設定値は全コンテキスト共に無効になっており，またコンテキストが無効化されるとそのコンテキストに対応するフィールドは自動的に無効となる．
このレジスタの設定には共通設定形式 (図 5.1) を用いる．またこのレジスタの読み出し要求に対しては，データの下位 8 bit [7:0] の上位から順番にコンテキスト番号 7 からコンテキスト番号 0 までの設定値が格納される．

REP_MODE

このレジスタにはキャッシュブロックの入れ換え方法を指定する．0 を設定すると LRU 情報に基づく方法，1 を設定するとオーナーコンテキストの優先度に基づく方法となる．
設定はデータの 1 bit 目で行われる．このレジスタの初期値は LRU を用いた方法である．またこのレジスタの読み出し要求に対しては，データの最下位 bit [0] に設定値が格納される．

ACC_SCHE

このレジスタには，6.1.4 で述べた優先度に従った内部バス要求キューの要求入れ換え機能の有効，無効を指定する．このレジスタに 1 を設定することでその機能を有効にできる．
設定は REP_MODE レジスタと同様にデータの 1 bit 目で行われ，初期値は無効である．またこのレジスタの読み出し要求に対しては，データの最下位 bit [0] に設定値が格納される．

LOCK

このレジスタには 5.1 で述べたコンテキスト毎のキャッシュロックの有効，無効を指定する．このレジスタと TLB エントリの CACHE_LOCK フィールドが設定されることで，該当コンテキストがキャッシュデータをロックすることが可能になる．どちらか一方の設定だけではキャッシュロックを行うことはできない．
一旦キャッシュをロックしてしまうと，そのコンテキストが有効である限りそのデータがキャッシュから追い出されることはない．これはキャッシュの入れ換えを優先度に従う方式で行っていても同様のため，低優先度のスレッドに対するロック許可や，ロック許可状態での高優先度のスレッドの実行を行う場合には注意が必要である．
初期状態の設定値は全コンテキスト共に無効になっており，またコンテキストが無効化されるとそのコンテキストに対応するフィールドは自動的に無効となる．
このレジスタの設定には共通設定形式 (図 5.1) を用いる．またこのレジスタの読み出し要求に対しては，データの下位 8 bit [7:0] の上位から順番にコンテキスト番号 7 からコンテキスト番号 0 までの設定値が格納される．

RESET

このレジスタに 1 を設定することで，キャッシュと victim buffer のエントリを全て無効化することができる．ただしデータ用のエントリに収められているデータでもその書き戻しは行わない．
このレジスタの読み出し要求に対しては，データの最下位 bit [0] に現在の状態を格納する．1 が読み出された場合は，現在キャッシュの無効化が行われていることを意味する．

FLUSH (データキャッシュコントローラのみ)

このレジスタを有効に設定することで，キャッシュデータと victim buffer のデータの下位メモリへの書き戻しを開始する．書き戻しが終了すると，自動的に無効状態になる．設定には共通設定形式 (図 5.1) を用い，コンテキスト単位で書き戻し要求を指定できるが，無効状態のコンテキストに対する指定でも書き戻しを行う．

またこのレジスタの読み出し要求に対しては，データの下位 8 bit [7:0] の上位から順番にコンテキスト番号 7 からコンテキスト番号 0 までの設定値が格納される．

ALL_FLUSH

このレジスタに書き込み要求を行うと，それだけで全コンテキストの書き戻しを開始する．指定するデータに制限はない．

7

システムレジスタ

システムレジスタは MFC0，MTC0 命令でアクセスする．アクセスしたいレジスタ番号を入れたレジスタを rd に指定する．

7.1 レジスタマップ

offset	31 24 23 16 15 8 7 0
0x00	Status Register (Thread0)
0x01	Status Register (Thread1)
0x02	Status Register (Thread2)
0x03	Status Register (Thread3)
0x04	Status Register (Thread4)
0x05	Status Register (Thread5)
0x06	Status Register (Thread6)
0x07	Status Register (Thread7)
0x08	Thread Table Register (Thread0)
0x09	Thread Table Register (Thread1)
0x0a	Thread Table Register (Thread2)
0x0b	Thread Table Register (Thread3)
0x0c	Thread Table Register (Thread4)
0x0d	Thread Table Register (Thread5)
0x0e	Thread Table Register (Thread6)
0x0f	Thread Table Register (Thread7)
0x10	Thread ID Register (Thread0)
0x11	Thread ID Register (Thread1)
0x12	Thread ID Register (Thread2)
0x13	Thread ID Register (Thread3)
0x14	Thread ID Register (Thread4)
0x15	Thread ID Register (Thread5)
0x16	Thread ID Register (Thread6)
0x17	Thread ID Register (Thread7)

offset	31 24 23 16 15 8 7 0
0x18	Instruction Count Register (Thread0)
0x19	Instruction Count Register (Thread1)
0x1a	Instruction Count Register (Thread2)
0x1b	Instruction Count Register (Thread3)
0x1c	Instruction Count Register (Thread4)
0x1d	Instruction Count Register (Thread5)
0x1e	Instruction Count Register (Thread6)
0x1f	Instruction Count Register (Thread7)
0x20	Count Register (Thread0)
0x21	Count Register (Thread1)
0x22	Count Register (Thread2)
0x23	Count Register (Thread3)
0x24	Count Register (Thread4)
0x25	Count Register (Thread5)
0x26	Count Register (Thread6)
0x27	Count Register (Thread7)
0x28	Compare Register (Thread0)
0x29	Compare Register (Thread1)
0x2a	Compare Register (Thread2)
0x2b	Compare Register (Thread3)
0x2c	Compare Register (Thread4)
0x2d	Compare Register (Thread5)
0x2e	Compare Register (Thread6)
0x2f	Compare Register (Thread7)
0x30	Floating-Point Control Register (Thread0)
0x31	Floating-Point Control Register (Thread1)
0x32	Floating-Point Control Register (Thread2)
0x33	Floating-Point Control Register (Thread3)
0x34	Floating-Point Control Register (Thread4)
0x35	Floating-Point Control Register (Thread5)
0x36	Floating-Point Control Register (Thread6)
0x37	Floating-Point Control Register (Thread7)
0x38	Issue Mode Register
0x39	CPU Count Register (Low)
0x3a	CPU Count Register (High)
0x3b ～ 0x47	MMU Register
0x48	Exception PC Register (Thread0)
0x49	Exception PC Register (Thread1)
0x4a	Exception PC Register (Thread2)
0x4b	Exception PC Register (Thread3)
0x4c	Exception PC Register (Thread4)
0x4d	Exception PC Register (Thread5)
0x4e	Exception PC Register (Thread6)
0x4f	Exception PC Register (Thread7)
0x50	Exception Cause Register (Thread0)
0x51	Exception Cause Register (Thread1)
0x52	Exception Cause Register (Thread2)
0x53	Exception Cause Register (Thread3)
0x54	Exception Cause Register (Thread4)
0x55	Exception Cause Register (Thread5)
0x56	Exception Cause Register (Thread6)
0x57	Exception Cause Register (Thread7)

offset	31 24 23 16 15 8 7 0
0x58	Interruption Wait Register (Thread0)
0x59	Interruption Wait Register (Thread1)
0x5a	Interruption Wait Register (Thread2)
0x5b	Interruption Wait Register (Thread3)
0x5c	Interruption Wait Register (Thread4)
0x5d	Interruption Wait Register (Thread5)
0x5e	Interruption Wait Register (Thread6)
0x5f	Interruption Wait Register (Thread7)
0x60	External Interruption Level Register (Thread0)
0x61	External Interruption Level Register (Thread1)
0x62	External Interruption Level Register (Thread2)
0x63	External Interruption Level Register (Thread3)
0x64	External Interruption Level Register (Thread4)
0x65	External Interruption Level Register (Thread5)
0x66	External Interruption Level Register (Thread6)
0x67	External Interruption Level Register (Thread7)
0x68 ～ 0x69	-
0x6a	Exception Base Address Register
0x6b ～ 0x6f	-
0x70 ～ 0x73	Event Link In Register
0x74 ～ 0x77	Event Link Out Register
0x80～0x84	Instruction Cache Control Register
0x86～0x8c	Data Cache Control Register
0x8e	ROM Status
0x8f	EXT Status
0x90	Multiplexer Arbitor Mode 256bit Bus
0x91	Multiplexer Arbitor Mode 32bit Bus
0x92	Multiplexer Watchdog Timer 256bit Bus Enable
0x93	Multiplexer Watchdog Timer 256bit Bus Mode
0x94	Multiplexer Watchdog Timer 256bit Bus Reset
0x95	Multiplexer Watchdog Timer 256bit Bus Count
0x96	Multiplexer Error Handler State 256bit Bus
0x97	Multiplexer Error Handler State 32bit Bus
0x98	Multiplexer Error Handler Instruction Cache
0x99	Multiplexer Error Handler Data Cache
0x9a	Multiplexer Error Handler DMAC0
0x9b	Multiplexer Error Handler DMAC1
0x9c	Multiplexer Error Handler DMAC2
0x9d	Multiplexer Error Handler PCI
0x9e	Multiplexer Error Handler Bus Interface Unit
0x9f	Multiplexer Error Handler MDMAC256
0xa0～0xb8	Address Decoder Control Register
0xb9	Multiplexer Watchdog Timer 32bit Bus Enable
0xba	Multiplexer Watchdog Timer 32bit Bus Mode
0xbb	Multiplexer Watchdog Timer 32bit Bus Reset
0xbc	Multiplexer Watchdog Timer 32bit Bus Count
0xbd	Multiplexer Error Handler MDMAC32
0xe0	Own Status Register
0xe1	Own Thread Table Register
0xe2	Own Thread ID Register
0xe3	Own Instruction Count Register
0xe4	Own Count Register

offset	31 — 24 23 — 16 15 — 8 7 — 0
0xe5	Own Compare Register
0xe6	Own Floating-Point Control Register
0xe7	Own Bad Virtual Address Register
0xe8	Own Exception PC Register
0xe9	Own Exception Cause Register
0xea	Own Interruption Wait Register
0xeb	Own External Interruption Level Register

7.1.1 Status Register

アドレス: 0x00 〜 0x07 (各スレッド毎)
スレッド毎の状態を示す．リセット後は 0x00000000 に初期化される．

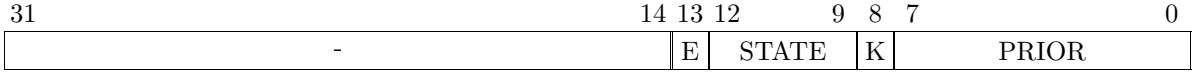
31 — 25	24	23	22	21 — 12	11 — 8	7	6	5	4 — 3	2	1	0
-	TS	PE	EV	-	IM	-	EB	C	MO	-	EL	IE

bit 名	機能
TS	Timer Start 1: タイマーをスタートする. 0: タイマーをストップする.
PE	Period 1: Timer Interrupt を周期的に発生する 0: One Shot
EV	Exception Vector Location. 1: Bootstrap ··· 例外発生時にブート時用の例外ベクタへ制御が移る. 0: Normal ··· 例外発生時に通常の例外ベクタへ制御が移る.
IM	Interruption Mask. 1 をセットすると，対応する種類の割り込みをマスクする. 11: Tiemr Interruption 10: Hardware Interruption 9: Software Interruption1 8: Software Interruption0
EBAE (EB)	Exception Base Address Enable 1: Variable ··· TBA(Table Base Address) を基準とした番地の例外ベクタを使用する. 0: Fixed ··· 固定番地の例外ベクタを使用する.
CRAM (C)	Control Register Access Mode 0: Precise ··· 直前の命令がコミットされるまで制御レジスタに対するアクセス命令の発行を待たせる.
MO	Mode Bit 00: Kernel Mode 01: Superviser Mode 10: User Mode
EL	Exception Level. 例外が発生すると 1 にセットされる. ERET 命令で 0 にセットされる.
IE	Interruption Mode 0: 全ての割り込みが無効 1: 全ての割り込みが有効

7.1.2 Thread Table Register

アドレス: 0x08 〜 0x0f (各スレッド毎)

スレッドの状態を示す. 基本的にスレッド制御命令によって変更を行う. 読み書き可能であるが，強制的に書き込みを行った場合の動作は保証しない. 0x08 意外は 0x00000000 に初期化される.



bit 名	機能
E	Thread Enable 0: そのコンテキストにアクティブスレッドが割り当てられていない. 1: そのコンテキストにアクティブスレッドが割り当てられている.
STATE	Thread State 0000: Invalid 0001: Run 0010: CMEM Access Ready 0011: CMEM Access Not Ready 0100: Backup Now 0101: Restore Now 0110: Backup Wait 0111: Restore Wait 1000: Copy or Swap Now 1001: Stop Wait
KEEP (K)	Keep Active Thread 0: 通常 1: スレッドをコンテキストキャッシュに退避する命令を無効にする.
PRIOR	Thread Priority. 256 Level.

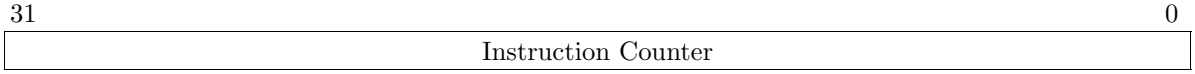
7.1.3 Thread ID Register

アドレス: 0x10 〜 0x17 (スレッド毎)

bit 名	機能
Thread ID	スレッドに対するアクセスの際のスレッドの指定に用いる.

7.1.4 Instruction Counter Register

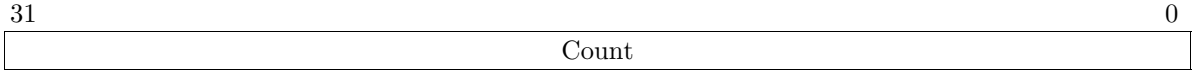
アドレス: 0x18 〜 0x1f (スレッド毎)



bit 名	機能
Instruction Counter	各スレッドが生成されてからコミットした命令の総数をカウントする.

7.1.5 Count Register

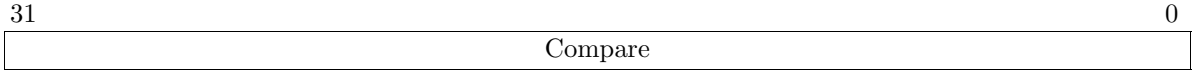
アドレス: 0x20 〜 0x27 (スレッド毎)



bit 名	機能
Count	毎クロックカウントアップされるカウンタ. Compare Register と等しくなると 0 にクリアされる.

7.1.6 Compare Register

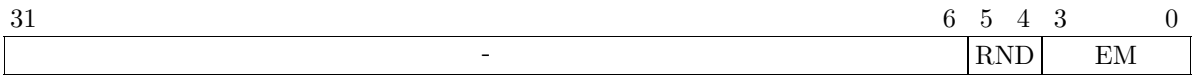
アドレス: 0x28 〜 0x2f (スレッド毎)



bit 名	機能
Compare	このレジスタに 0 以外の値がセットされており，かつ Count Register の値がこのレジスタの値と等しくなった時，タイマ割り込みを発生する．タイマ割り込みは Status Register の IM フィールドと IE フィールドで有効または無効に設定される.

7.1.7 Floating-Point Control Register

アドレス: 0x30 〜 0x37 (スレッド毎)



bit 名	機能
RND	Rounding Mode 00: Round to Nearest 01: Round to Zero 10: Round to Positive Infinity 11: Round to Negative Infinity
EM	Exception Mask 各ビットに 1 を立てることで、対応する例外をマスクすることができる。 3: Inexact Exception 2: Underflow Exception 1: Overflow Exception 0: Invalid Exception

7.1.8 Issue Mode Register

アドレス: 0x38
発行命令の選択方法を設定するレジスタ．0x00000000 に初期化される．

31 28	27 25	24 22	21 19	18 16	15 13	12 10	9 7	6 4	3 2	1 0
-	SA3	SA2	SA1	SA0	MA3	MA2	MA1	MA0	SP	PO

bit 名	機能
Sub Assign0, 1, 2, 3 (SA0, 1, 2, 3)	発行スロットにスレッドを割り当てる発行方式 (TH_ASSIGN) で，SUB_POLICY フィールドが SUB_FIX に設定されている状態で，スロット毎のメインで割り当てられているスレッドから命令を発行できない場合にこのフィールドで設定されたスレッドから命令を発行する．
Main Assign0, 1, 2, 3 (MA0, 1, 2, 3)	発行スロットにスレッドを割り当てる発行方式 (TH_ASSIGN) で，スロット毎にメインで割り当てるスレッドを指定する．
Sub Policy (SP)	発行命令選択ポリシーのサブポリシーを設定する． • 1INST_1TH 00: NORMAL 01: PRED_STOP ··· 次に発行すべき命令が分岐予測の結果として発行される命令であり，キャンセルされる可能性がある場合はそのスレッドの優先度を低下させる． 10: MINST_STOP ··· あるスレッドのリオーダバッファの半数以上のエントリが埋まっている場合はそのスレッドの優先度を低下させる． • TH_ASSIGN 00: SUB_PRIOR ··· スロットにメインで割り当てられているスレッドに発行できる命令がない場合は，スロットに割り当てられていないスレッドの中で最も優先度の高いスレッドから命令を発行する． 01: SUB_FIX ··· スロットにメインで割り当てられているスレッドに発行できる命令がない場合は，スロットに対しサブで割り当てられているスレッドから命令を発行する．サブのスレッドにも発行できる命令がない場合は，空きスロットとなる．
Policy (PO)	発行選択ポリシーを設定する． 00: 1INST_1TH ··· 毎クロックサイクル，1 スレッドから最大 1 命令だけを発行可能とするポリシー．4 スレッド以上のスレッドが実行されていないと，発行スロットに空きができてしまうことになる． 01: HIGHEST_FAST ··· 毎クロックサイクル，最高優先度のスレッドから発行できるだけの命令を発行し，余った発行スロットは次に高い優先度を持つスレッドに割り当てる．さらに余った場合は 3 番目に高い優先度を持つスレッドでも同様に行う． 10: TH_ASSIGN ··· 発行スロットごとに特定のスレッドを割り当てて，そのスレッドから命令を発行できない場合のみ他のスレッドの命令を発行する．

7.1.9 CPU Count Register

アドレス: 0x39, 0x3a

31	0
CPU Count	

bit 名	機能
CPU Count	64bit カウンタ．リセット時から毎クロック 1 ずつカウントアップしていく．0x39 が下位 32bit，0x3a が上位 32bit を示す．

7.1.10 MMU Register

アドレス: 0x3b 〜 0x47
MMU 関連の設定レジスタ．

7.1.11 Exception PC Register

アドレス: 0x48 〜 0x4f (スレッド毎)

31	0
Exception PC	

bit 名	機能
Exception PC	例外を生じた命令，もしくは例外が発生した時点で最後にコミットされた PC を保持するレジスタ．ERET 命令で例外処理からの戻り番地として参照する．

7.1.12 Exception Cause Register

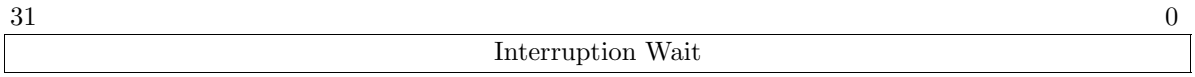
アドレス: 0x50 〜 0x57 (スレッド毎)
発生した例外の情報を保持するレジスタ．0x00000000 に初期化される．

31	30 … 17	16 … 12	11	10	9	8	7	6 … 2	1	0
D	-	HIRL	TI	HI	S1	S0	-	CODE	-	

bit 名	機能
Delay Bit (D)	例外を発生した命令が Delay Slot の命令である場合に 1 がセットされる.
Hardware Interruption Level (HIRL)	外部割込みのレベル.
TI	Timer Interruption Pending
HI	Hardware Interruption Pending
S1	Software Interruption 1 Pending
S0	Software Interruption 0 Pending
Exception Code (CODE)	最後に発生した例外のコードを保持する.

7.1.13 Interruption Wait Register (スレッド毎)

アドレス: 0x58 〜 0x5f



bit 名	機能
Interruption Wait	各ビットが割り込みレベル (IRL) に対応している. ビットが 1 ならそのスレッドは対応する IRL の外部割込みを受け付ける.

7.1.14 External Interruption Level Register (スレッド毎)

アドレス: 0x60 〜 0x67

bit 名	機能
External Interruption Level	最後に IRC から入力された外部割り込みの IRL を保持する.

7.1.15 Interruption Pending Register

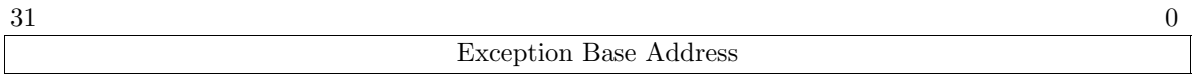
アドレス: 0x68
現在 (多分) 使用していない.

7.1.16 Interruption Clear Register

アドレス: 0x69
現在 (多分) 使用していない.

7.1.17 Exception Base Address Register

アドレス: 0x6a



bit 名	機能
Exception Base Address	例外ベクタのベースアドレスを保持する．Status Register の EBAE ビットを 1 に設定すると，例外発生時に EBA に例外の内容に従ったオフセットを加えた番地に制御が移る．

7.1.18 Event Link In Register

アドレス: 0x70 〜 0x73

7.1.19 Event Link Out Register

アドレス: 0x74 〜 0x77

7.1.20 Instruction Cache Control Register

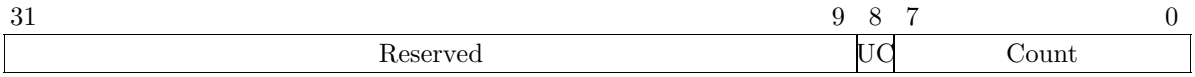
アドレス: 0x80〜0x84
6 章 (CACHE) のキャッシュのコントローロレジスタを参照。

7.1.21 Data Cache Control Register

アドレス: 0x86〜0x8c
6 章 (CACHE) のキャッシュのコントローロレジスタを参照。

7.1.22 ROM Status

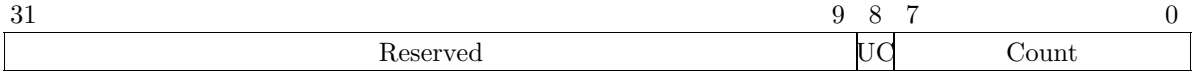
アドレス: 0x8e



bit 名	機能
Uncache (UC)	1 を指定すると ROM アクセス時に 32bit I/O バスに Uncache 信号を生成する (現在は使用されていない)。Read 時は反転した値が読み出される。
Count	ROM アクセス時の Auto Ready を返すカウントを指定する。

7.1.23 EXT Status

アドレス: 0x8f



bit 名	機能
Uncache (UC)	1 を指定すると外部バスアクセス時に 32bit I/O バスに Uncache 信号を生成する (現在は使用されていない)。Read 時は反転した値が読み出される。
Count	外部バスアクセス時の Auto Ready を返すカウントを指定する。

7.1.24 Multiplexer Arbitor Mode 256bit Bus

アドレス: 0x90

bit 名	機能
Mode (MO)	バスアービトレーションのモードを指定する。0 を指定すると固定優先度、1 を指定するとラウンドロビン方式でバスマスタにバス権を与える。

7.1.25　Multiplexer Arbitor Mode 32bit Bus

アドレス: 0x91

bit 名	機能
Mode (MO)	バスアービトレーションのモードを指定する。0 を指定すると固定優先度、1 を指定するとラウンドロビン方式でバスマスタにバス権を与える。

7.1.26　Multiplexer Watchdog Timer 256bit Bus Enable

アドレス: 0x92

bit 名	機能
Mode (MO)	バスアービトレーションのモードを指定する。0 を指定すると固定優先度、1 を指定するとラウンドロビン方式でバスマスタにバス権を与える。

7.1.27　Multiplexer Watchdog Timer 256bit Bus Mode

アドレス: 0x93
使用していない。

7.1.28　Multiplexer Watchdog Timer 256bit Bus Reset

アドレス: 0x94

7.1.29　Multiplexer Watchdog Timer 256bit Bus Count

アドレス: 0x95

7.1.30　Multiplexer Error Handler State 256bit Bus

アドレス: 0x96

7.1.31 Multiplexer Error Handler State 32bit Bus

アドレス: 0x97

7.1.32 Multiplexer Error Handler Instruction Cache

アドレス: 0x98

7.1.33 Multiplexer Error Handler Data Cache

アドレス: 0x99

7.1.34 Multiplexer Error Handler DMAC0

アドレス: 0x9a

7.1.35 Multiplexer Error Handler DMAC1

アドレス: 0x9b

7.1.36 Multiplexer Error Handler DMAC2

アドレス: 0x9c

7.1.37 Multiplexer Error Handler PCI

アドレス: 0x9d

7.1.38 Multiplexer Error Handler Bus Interface Unit

アドレス: 0x9e

7.1.39 Multiplexer Error Handler MDMAC256

アドレス: 0x9f

7.1.40 Address Decoder Control Register

アドレス: 0xa0〜0xb8
4 章 (アドレスデコーダ) のアドレスマップを参照。

7.1.41 Multiplexer Watchdog Timer 32bit Bus Enable

アドレス: 0xb9

7.1.42 Multiplexer Watchdog Timer 32bit Bus Mode

アドレス: 0xba

7.1.43 Multiplexer Watchdog Timer 32bit Bus Reset

アドレス: 0xbb

7.1.44 Multiplexer Watchdog Timer 32bit Bus Count

アドレス: 0xbc

7.1.45 Multiplexer Error Handler MDMAC32

アドレス: 0xbd

7.1.46 Own Status Register

アドレス: 0xe0

7.1.47 Own Thread Table Register

アドレス: 0xe1

7.1.48 Own Thread ID Register

アドレス: 0xe2

7.1.49 Own Instruction Count Register

アドレス: 0xe3

7.1.50 Own Count Register

アドレス: 0xe4

7.1.51 Own Compare Register

アドレス: 0xe5

7.1.52 Own Floating-Point Control Register

アドレス: 0xe6

7.1.53 Own Bad Virtual Address Register

アドレス: 0xe7

7.1.54 Own Exception PC Register

アドレス: 0xe8

7.1.55 Own Exception Cause Register

アドレス: 0xe9

7.1.56 Own Interruption Wait Register

アドレス: 0xea

7.1.57 Own External Interruption Level Register

アドレス: 0xe8

8

例外処理

8.1 割り込みコントローラ (IRC)

初期アドレス: 0xffff9000

8.1.1 レジスタマップ

offset	31							24 23							16 15							8 7							0
0x00	31ch							30ch							29ch							28ch							27ch
0x04	26ch							25ch							24ch							23ch							22ch
0x08	21ch							20ch							19ch							18ch							17ch
0x0c	16ch							15ch							14ch							13ch							12ch
0x10	11ch							10ch							9ch							8ch							7ch
0x14	6ch							5ch							4ch							3ch							2ch
0x18	1ch							0x00																					

offset	31 ... 1	0
0x08	Request Sense Register	0
0x0c	Request Clear Register	0
0x10	Mask Register	MI
0x14	26'h0	CL / IRL Latch
0x18	31'h0	Mode

8.1.2 Trigger Mode Register

オフセット: 0x00,0x04

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
31ch				30ch				29ch				28ch				27ch				26ch				25ch				24ch			

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
23ch				22ch				21ch				20ch				19ch				18ch				17ch				16ch			

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
15ch				14ch				13ch				12ch				11ch				10ch				9ch				8ch			

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
7ch				6ch				5ch				4ch				3ch				2ch				1ch				0x00			

<table>
<tr><th>bit 名</th><th>機能</th></tr>
<tr><td>Trigger</td><td>各チャネルのトリガモードの設定．offset 0x00 ではチャネル 31-17 まで，offset0x04 ではチャネル 16-1 のトリガモードの設定を行う．

<table>
<tr><th>bit</th><th>Trigger Mode</th></tr>
<tr><td>00</td><td>High Level</td></tr>
<tr><td>01</td><td>Low Level</td></tr>
<tr><td>10</td><td>Rise Edge</td></tr>
<tr><td>11</td><td>Fall Edge</td></tr>
</table>
</td></tr>
</table>

8.1.3 Request Sense Register

<table>
<tr><td>31</td><td></td><td>1</td><td>0</td></tr>
<tr><td colspan="3">Request Sense Register</td><td>0</td></tr>
</table>

bit 名	機能
Request Sense	Trigger Mode Register で設定されたトリガが端子 IRLIN,IRQIN に入力されると，その割り込みチャネルに対応したビットに 1 がセットされる．bit31 が IRQ31, bit1 が IRQ1 に対応．Read のみ

8.1.4 Request Clear Register

<table>
<tr><td>31</td><td></td><td>1</td><td>0</td></tr>
<tr><td colspan="3">Request Clear Register</td><td>0</td></tr>
</table>

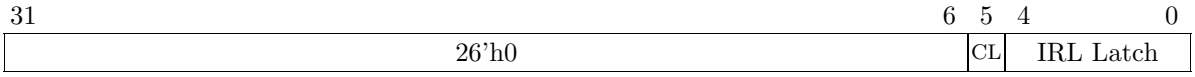
bit 名	機能
Request Clear	Request Clear Register の bit31-1 の中で 1 がセットされるとそれに対応する保持されていた割り込み要求が 0 になる．write のみ．

8.1.5 Mask Register

<table>
<tr><td>31</td><td></td><td>1</td><td>0</td></tr>
<tr><td colspan="3">Mask Register</td><td>MI</td></tr>
</table>

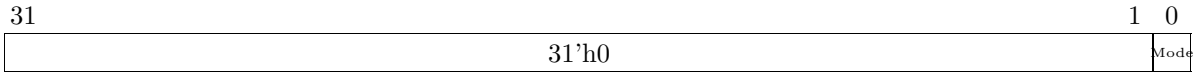
bit 名	機能
Mask	31-1 ビットが割り込みチャネルの 31-1 に対応し，1 をセットすることで割り込みをマスクできる．ただし，Mask が 1 の場合でも Request Sense Register はセットされる．
MI	0 ならば，IRLOUT に割り込みレベルラッチの内容を出力．1 ならばマスクし，IRLOUT には “L” を出力

8.1.6 IRL Latch/Clear



bit 名	機能
IRL Latch	割り込みレベルラッチの内容を出力
CL	1 を書き込むことで，割り込みレベルラッチの内容をクリアし，次の割り込みレベルをラッチする．

8.1.7 IRC Mode Register



bit 名	機能
Mode	1 ならば端子 IRLIN を IRQIN[31:27] として使用．0 ならば端子 IRLIN をそのまま IRLOUT に出力．

8.2 動作/使用方法

8.2.1 IRC

IRC モードレジスタが 1 に設定されると，入力端子 IRLIN[4:0](IRQIN[31:27]), IRQIN[26:1] に入力された割り込み信号はトリガモードレジスタに設定されたトリガモードに従ってその割り込みを保持します．保持された割り込みは MASK レジスタでマスクされていないもののうちでプライオリティが一番高いものがコード化されて割り込みレベルラッチ (IRL Latch) に保持されます．保持された IRL Latch のデータは MASK レジスタの MI ビットが (ビット 0) が 0 の場合，端子 IRLOUT[4:0] に出力されます．

IRC モードレジスタが 0 に設定されると，端子 IRLIN[4:0] に入力されたデータがそのまま IRLOUT[4:0] に出力されます．

表 8.1: 割り込みマップ

IRQ31	Bus Error
IRQ30	Address Error
IRQ29	Watch Dog Timer Error
IRQ28	Link
IRQ27	Pulse Counter 8
IRQ26	DMAC2
IRQ25	DMAC1
IRQ24	DMAC0
IRQ23	PCI
IRQ22	USB
IRQ21	IEEE1394
IRQ20	Pulse Counter 7
IRQ19	Pulse Counter 6
IRQ18	Pulse Counter 5
IRQ17	Pulse Counter 4
IRQ16	Pulse Counter 3
IRQ15	Pulse Counter 2
IRQ14	Pulse Counter 1
IRQ13	Pulse Counter 0
IRQ12	UART 1
IRQ11	UART 0
IRQ10	External IO 1
IRQ9	External IO 0
IRQ1-8	Low に固定

8.2.2 RMT 固有機能

- IRL Unit
  割り込みが入る度に実行中の全てのスレッドが割り込みハンドラを起動するのでは，非効率的であり，割り込みに対するレスポンス時間の増加を招いてしまいます．そのため，RMT では IRL Unit という IRL を各スレッドに割り当てる機構を持ちます．

  IRL Unit にはスレッド毎に Interruption Wait Register が用意されています．Interruption Wait Register の各ビットは IRL に対応しています．IRC が出力した IRL を受け取り，Interruption Wait Register の IRL に対応するビットが 1 のとき，割り込みを受け付け，Exception Unit に外部割り込みが発生したこととその IRL を通知します．

  Interruption Wait Register はコンテキストスイッチの際にレジスタセットと同様にバックアップ及びリストアされるために，コンテキストスイッチの度に設定する必要はありません．

- 割り込みによるスレッド起動
  外部イベントによるスレッド制御の際のレスポンス時間を短縮するために，停止状態にあるスレッド

(Status Register の STATE フィールドが 0010 or 0011) に対して外部割り込みがかかった場合はそのスレッドを実行状態にします.

ただし，この際に Interruption Wait Register の対象とする割り込みに対応するビットが 1 である必要があります.

8.2.3 例外処理プロセス

タイマ割り込み，ハードウェア割り込み (外部割り込み)，ソフトウェア割り込みが発生すると，Exception Cause Register の対応する Pending Register が 1 にセットされます．また，外部割り込みの場合は Exception Cause Register の HIRL に通知された IRL をセットします．これらの割り込みが通常通り発生するには Status Register の Interruption Mode(bit0) が 0, Interruption Mask(bit11-8) の対応するビットが 0 である必要があります.

これらの割り込みや RMTPU で例外が発生した場合，Status Register の Exception Level(bit1) が 0 ならば通常通り例外処理が発生します．Exception Unit では CPU core の exception 信号や Exception Cause Register の Pending Register を参照し，実際に例外処理を行います．例外処理の優先度は例外，タイマ割り込み，ハードウェア割り込み，ソフトウェア割り込みの順です.

例外処理が発生すると，次の動作が行われます.

- Status Register の Exception Level を 1 にセット
- その時点のモードを保持し，カーネルモードへ移行
- 例外を生じた命令，もしくは例外が発生した時点で最後にコミットされた PC を Exception PC Register に保持
- Exception Code Register の CODE フィールドに例外を識別するコード (表 8.2) を書き込む
- Status Register の Exception Base Address Enable が 1 ならば Exception Base Address Register の値に例外の内容に従ったオフセットの値を加えた番地に制御が移る
  Exception Base Address Enable が 0 ならば固定番地 (Exception Vector Location が 1 ならば 0xbfc00200,0 ならば 0x80000000) に制御が移る

外部割り込みが起こった際に，対象となるスレッドが停止状態にある場合は通常の例外処理と異なり，スレッドが実行状態へ移行する処理のみ行われます.

外部割り込みに対する例外処理ルーチンでは処理に対応した IRC に保持されている割り込み要求をクリアし，IRL Latch をクリアする必要があります．その結果次の保持されている割り込みを IRL Latch に保持することができ，Exception Cause Register の対応する Pending フィールドが更新されます．タイマ割り込みやソフトウェア割り込みの場合は明示的に Exception Cause Register の対応する Pending フィールドをクリアする必要があります.

例外処理の終了時には ERET 命令を実行することで次の動作が行われます.

- Status Rergister の Exception Level を 0 にセット
- モードを例外発生時のものに戻す
- Exception PC Register に格納された番地に制御が移る

表 8.2: Exception Code, Offset

種類	コード	オフセット
I-TLB SPR Address Miss	0x01	0x010
I-TLB All Entry Locked	0x02	0x020
I-TLB No Entry Matched	0x03	0x030
I-TLB Thread Mode Error	0x04	0x040
I-TLB Protection Error	0x05	0x050
D-TLB SPR Address Miss	0x06	0x060
D-TLB All Entry Locked	0x07	0x070
D-TLB No Entry Matched	0x08	0x080
D-TLB Thread Mode Error	0x09	0x090
D-TLB Protection Error	0x0a	0x0a0
Coprocessor Unusable	0x0b	0x0b0
Reserved (Invalid) Instruction	0x0c	0x0c0
Sytem Call	0x0d	0x0d0
Break Point	0x0e	0x0e0
Integer Overflow	0x0f	0x0f0
Divide By Zero	0x10	0x100
Trap	0x11	0x110
Data Address Miss Align (Load)	0x12	0x120
Data Address Miss Align (Store)	0x13	0x130
Floating Point Overflow	0x14	0x140
Floating Point Underflow	0x15	0x150
Floating Point Divide By 0	0x16	0x160
Floating Point Inexact	0x17	0x170
Floating Point Invalid Operation	0x18	0x180
Reserve	0x19	0x190
Vector Integer Exception	0x1a	0x1a0
Vector Floating Exception	0x1b	0x1b0
Timer Interruption	0x1c	0x1c0
Hardware Interruption	0x1d	0x1d0
Software Interruption	0x1e	0x1e0
Software Interruption	0x1f	0x1f0

9

クロックジェネレータ

9.1 接続図

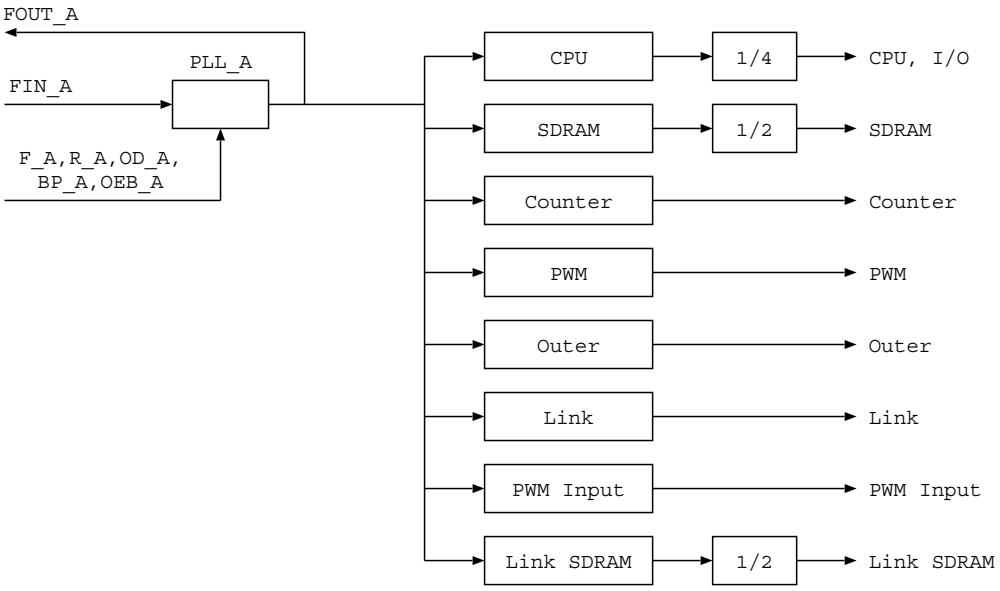


図 9.1: クロック生成部

ピン名	概要	デフォルト値
FIN_A	クロック入力	-
FOUT_A	PLL 出力	-
F_A	PLL の逓倍数を制御する	16
R_A	PLL の逓倍数を制御する	3
OD_A	PLL の逓倍数を制御する	0
PD_A	PLL Power Down Mode (1:Power Down)	0
BP_A	PLL Bypass Mode (1: Bypass)	0
OEB_A	PLL Output Enable (0:Enable)	0

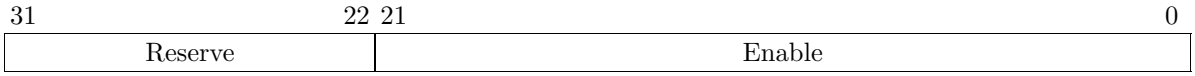
デフォルトでは FIN_A より 75MHz のクロックを入力し，PLL_A から 800MHz が出力される．これらを分周器で分周し各モジュールにクロックを供給する．

9.2　制御レジスタ

初期アドレス: 0xffffa000

9.2.1　Clock Enable

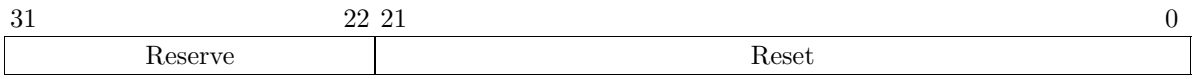
オフセット: 0x0000
各クロックを有効/無効にする．対応するビットを 1 で無効，0 で有効になる．初期値は全て 0.



bit 名	機能
Enable	21: Link Sdram, 20: PWM Input, 19: Link, 18: Outer, 17: PWM, 16: Counter, 15: SDRAM, 14: Vector Floating-Point, 13: Vector Integer, 12: Synchronize, 11: Floating-Point Reservation Station, 10: SIMD, 9: FPU, 8: Complex INT, 7: Context Cache, 6: PCI, 5: USB, 4: IEEE1394, 3: DMAC2, 2: DMAC1, 1: DMAC0, 0: CPU

9.2.2　Soft Reset

オフセット: 0x0004
各モジュールにリセットをかける．対応するビットを 0 にするとリセットがかかる．ビットを 1 に戻さない限りリセット状態が続く (CPU を除く).

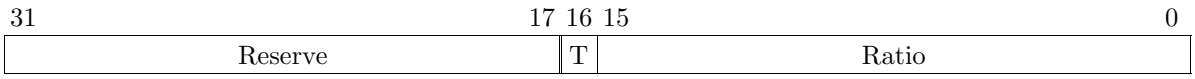


bit 名	機能
Reset	21: Link Sdram, 20: PWM Input, 19: Link, 18: Outer, 17: PWM, 16: Counter, 15: SDRAM, 14: Vector Floating-Point, 13: Vector Integer, 12: Synchronize, 11: Floating-Point Reservation Station, 10: SIMD, 9: FPU, 8: Complex INT, 7: Context Cache, 6: PCI, 5: USB, 4: IEEE1394, 3: DMAC2, 2: DMAC1, 1: DMAC0, 0: CPU

9.2.3 Divider Ratio

オフセット: 0x0008〜0x001c, 0x0028, 0x002c
各クロックの分周率を設定する．対応する分周器のアドレスと初期値は以下の通り．

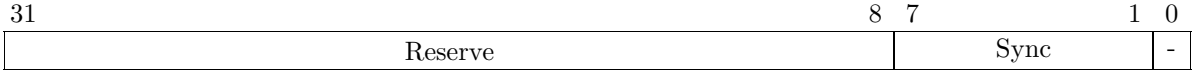
分周器	デフォルト値	アドレスオフセット
CPU	1/2	0x0008
SDRAM	1/4	0x000c
Counter	1/8	0x0010
PWM	1/512	0x0014
Outer	1/8	0x0018
Link	1/1	0x001c
PWM Input	1/512	0x0028
Link SDRAM	1/4	0x002c



bit 名	機能
T	分周せずにクロックをスルーする (1/1 指定時)
Ratio	クロックの分周率を指定する．指定した数値の半分 (小数点以下切捨て) でクロックが立ち下がり，指定した数値でクロックが立ち上がる．1 を指定した場合の動作は保証外．1/1 の場合は T ビットを 1 にすること．

9.2.4 Clock Synchronization

オフセット: 0x0020
1 を指定したクロックの立ち上りエッジを CPU のクロックにそろえる．次のクロックで自動的に値はリセットされる．



bit 名	機能
Sync	7: Link SDRAM, 6: PWM Input, 5: Link, 4: Outer, 3: PWM, 2: Counter, 1: SDRAM

9.2.5 All Reset

オフセット: 0x0024
このアドレスに書き込みを行うと全てにリセットをかける.

10

スレッド制御

Responsive Multithreaded Processor におけるスレッド制御方法について述べる．

10.1 スレッドの種類

RMT Processor のスレッドは 2 つに分類される．

- アクティブスレッド
- キャッシュスレッド

アクティブスレッドとはレジスタファイルやプログラムカウンタなどの資源が確保され，プロセッサ内ですぐにでも実行可能なスレッドを示す．キャッシュスレッドとはコンテキストキャッシュ内に保持されているスレッドを示す．*RMT Processor* が実行するスレッドはアクティブスレッドで実行状態にあるスレッドのみである．リセット時，スレッド ID が 0 のスレッドが優先度 0 でアドレス 0 から実行される．

10.2 スレッド制御命令

10.2.1 作成・削除

新しくスレッドを作成する場合は mkth 命令を用いる．また，アクティブスレッドをコピーして新しいスレッドを作成することも可能である．

- mkth

  新しくアクティブスレッドを作成する．rs でスレッド ID，rt でスタートアドレスを設定する．mkth 命令はアクティブスレッドを作成するだけで実行は開始しない．つまり mkth 命令で作成されたスレッドはストップ状態にある．実行を開始するためには runth 命令を使用する．スレッドの作成に成功すると rd に 1 が返り，失敗すると 0 が返る．

- delth

  アクティブスレッドを削除する．rs で削除するスレッドの ID を指定する．スレッドの削除に成功すると rd に 1 が返り，失敗すると 0 が返る．この命令が成功すると指定されたスレッドはプロセッサから削除される．

- cpthtoa

  アクティブスレッドを別のアクティブスレッドとしてコピーする．rs でコピー元のスレッド ID，rt で新たに作成するスレッドの ID を指定する．cpthtoa 命令はアクティブスレッドのコピーを新しいアクティブスレッドとして作成する．作成したコピーはストップ状態にあり，実行を開始するためには runth 命令を使用する．スレッドのコピーに成功すると rd に 1 が返り，失敗すると 0 が返る．

- cpthtom

  アクティブスレッドを別のキャッシュスレッドとしてコピーする．rs でコピー元のスレッド ID，rt で新たに作成するスレッドの ID を指定する．cpthtom 命令はアクティブスレッドのコピーを新しいキャッシュスレッドとして作成する．作成したコピーはコンテキストキャッシュ内にあるため，実行を開始するためには rstrth 命令などでアクティブスレッドにしなければならない．スレッドのコピーに成功すると rd に 1 が返り，失敗すると 0 が返る．

10.2.2　状態制御

アクティブスレッドは実行・停止のいずれかの状態にある．これらの状態は以下の命令を用いて制御する．

- runth

  停止状態のアクティブスレッドを実行状態にする．rs でスレッド ID を指定する．指定されたスレッドは実行状態になり，優先度に従って実行が開始される．実行開始に成功すると rd に 1 が返り，失敗すると 0 が返る．

- stopth

  実行状態のアクティブスレッドを停止状態にする．rs でスレッド ID を指定する．指定されたスレッドは停止状態になり，命令実行のスケジューリングからはずされる．再び実行するためには runth 命令を実行する．停止に成功すると rd に 1 が返り，失敗すると 0 が返る．

- stopslf

  自分自身を停止状態にする．停止に成功すると rd に 1 が返り，失敗すると 0 が返る．

- chgpr

  スレッドの優先度を変更する．rs で変更するスレッドの ID，rt で新しい優先度を指定する．優先度の変更に成功すると rd に 1 が返り，失敗すると 0 が返る．優先度が変更されると，つぎのクロックから新しい優先度で命令実行が制御される．

10.2.3　転送

RMT Processor はコンテキストキャッシュを持ち，コンテキストスイッチにおけるオーバヘッドを軽減している．以下にコンテキストキャッシュとの転送命令を示す．

- bkupth

  アクティブスレッドをコンテキストキャッシュに退避する．rs で退避するアクティブスレッドの ID を指定する．指定されたアクティブスレッドは実行を停止し，コンテキストキャッシュに退避される．退避に成功すると rd に 1 が返り，失敗すると 0 が返る．

- bkupslf

  自分自身をコンテキストキャッシュに退避する．退避に成功すると rd に 1 が返り，失敗すると 0 が返る．

- rstrth

  キャッシュスレッドをアクティブスレッドとして復帰する．rs で復帰するスレッドの ID を指定する．指定されたキャッシュスレッドはコンテキストキャッシュから読み込まれ，停止状態になる．復帰に成功すると rd に 1 が返り，失敗すると 0 が返る．

- swapth

  アクティブスレッドとキャッシュスレッドを入れ換える．rs で退避するアクティブスレッドの ID，rt で復帰するキャッシュスレッドの ID を指定する．指定されたアクティブスレッドは実行を停止し，コンテキストキャッシュに退避される．同時に指定されたキャッシュスレッドがコンテキストキャッシュから読み込まれ，実行状態になる．入れ換えに成功すると rd に 1 が返り，失敗すると 0 が返る．

- swapslf

  自分自身とキャッシュスレッドを入れ換える．rt で復帰するキャッシュスレッドの ID を指定する．自分自身の実行を停止し，コンテキストキャッシュに退避する．同時に指定されたキャッシュスレッドがコンテキストキャッシュから読み込まれ，実行状態になる．入れ換えに成功すると rd に 1 が返り，失敗すると 0 が返る．

10.3 状態遷移

RMT Processor におけるスレッドの状態遷移を図 10.1 に示す．

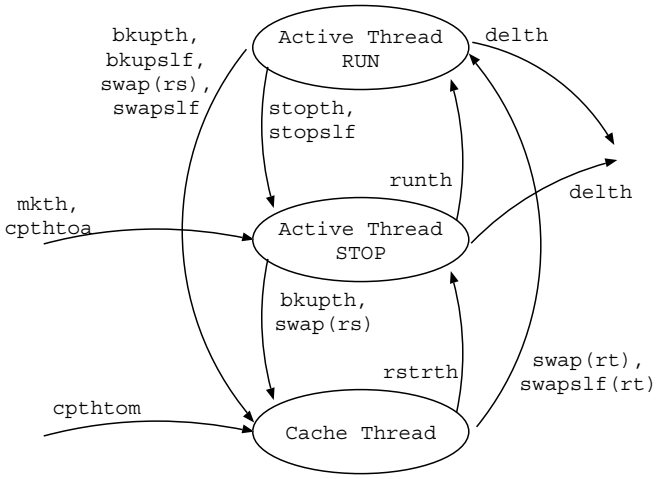


図 10.1: スレッドの状態遷移

図 10.1 において Active Thread RUN 状態のスレッドのみ優先度に従ってプロセッサで実行される．

11

同期

11.1 共有レジスタ

31 個 (レジスタ番号 32 は設定レジスタ) の共有レジスタ (64bit) を持ち，そのレジスタにロックをかけることで同期を取ることが可能です.

共有レジスタには次の 3bit とスレッド ID が割り当てられ，その共有レジスタの使用権利をどのスレッドが持つのか意味します.

- Full/Empty bit
- Exclusive / Producer-Consumer bit
- Barrier bit

Full/Empty bit が現在このレジスタにロックがかかっているかを示し，他 2bit はどの種類のロックがかかっているかを示します.

11.2 同期命令

共有レジスタには次の命令を用いてアクセス可能です.

- RGPEX, WGPEX, RFPEX, WFPEX
  Read 命令は対象となる共有レジスタの F/E bit が 0 のときに実行され，共有レジスタの値をデスティネーションレジスタに書き込みます．Read 命令が成功すると，対象のレジスタの F/E bit が 1 にし，自分のスレッド ID を書き込みます.

  Write 命令は対象の共有レジスタの F/E bit が 1 でなおかつ自分のスレッド ID が共有レジスタの権利を所持しているスレッド ID と等しい場合に実行されます．成功すると，ソースレジスタの値を共有レジスタに書き込み，F/E bit を 0 にすることでロックを開放します．ロックが掛かっていない場合には NOP となります.

- GPCO, GPPR, FPCO, FPPR
  Write 命令は対象となる共有レジスタの F/E bit が 0 のときに実行され，ソースレジスタの値を共有

レジスタに書き込みます．成功すると対象のレジスタの F/E bit を 1 にします．また，同時にスレッド ID を命令で指定し，その ID をロックをかけた対象の共有レジスタの権利者として書き込みます．

Read 命令は対象となる共有レジスタの F/E bit が 1 でなおかつ共有レジスタの権利を所持しているスレッド ID が自分のスレッド ID と一致する場合に実行され，共有レジスタの値をデスティネーションレジスタに読み出します．命令の終了時には F/E bit を 0 にします．

- RGPSH, WGPSH, RFPSH, WFPSH
  ロックをかけない共有レジスタアクセス命令です．対象の F/E bit が 1 のときは実行できません．

- BAR
  バリア命令です．共有レジスタは到着スレッド数を数えるのに使用されます．ソースレジスタは対象となるスレッド数を示します．初めて実行される場合，対象の共有レジスタの F/E bit が 0 のときに実行できます．成功すると共有レジスタの値を 1 にし，F/E bit を 1 にします．2 番目以降のバリア命令では共有レジスタの値を increment しソースレジスタの値と等しくなるまで，そのスレッドはストールします．ソースレジスタの値と等しくなると，全てのスレッドのストールを解除します．

共有レジスタアクセス命令の実行条件を表 11.1 に示します．

表 11.1: 共有レジスタアクセスの実行条件

命令種別	実行条件			実行後			注釈
	F/E	E/P	bar	F/E	E/P	bar	
EX_READ	E	x	x	F	E	0	
EX_WRITE	F	E	0	E	x	x	TH ID 一致，条件以外では NOP
CON_READ	F	P	0	E	x	x	対象 TH ID と一致
PRO_WRITE	E	x	x	F	P	0	
SH_READ	E	x	x	E	x	x	
SH_WRITE	E	x	x	E	x	x	
BARRIER	E	x	x	F	x	1	バリアに初めに到着する命令
	F	x	1	F	x	1	バリア待ち
				E	x	0	バリア解放

同期命令の失敗時にはデッドロックの回避のために，対象のスレッドのパイプライン中の命令を全て開放し，フェッチを止めることでストールさせます．対象のレジスタのロックが開放される (CON_READ では書き込みが起こる) とストールが解除され，再び実行されます．

また，同期命令の失敗時に次のスレッド ID を調べ，そのスレッドがコンテキストキャッシュ内に退避されている場合は同期命令を失敗したスレッドと入れ換えます．

- 対象の共有レジスタにロックが掛かっている場合
  ロックを獲得しているスレッド

- Read Consumer 命令が値が書き込まれておらず失敗した場合
  Read Consumer 命令で指定する相手スレッド

また，バリア命令により他の到着スレッドを待つ場合には，同じバリア命令を実行するグループのスレッドを調べます．その際，そのスレッドが属するグループを設定する命令として PBAR 命令があります．PBAR 命令はバリアに使用する共有レジスタを指定します．バリア待ちのスレッドは現在使用している共

有レジスタ番号と同じ値を PBAR 命令で設定されたスレッドがコンテキストキャッシュ内にあるか調べ, 存在する場合は自分と入れ換えます.

この同期命令失敗時のスレッド切替え機能は共有レジスタの 31 番に 0 以外の値を書き込むことで有効になります. default では無効化されています.

12

Vector Unit

12.1 概要

RMT Processor の Vector Unit のブロック図を図 12.1 に示す。Vector Integer Unit、Vector Floating Point Unit 共に大きく 3 つの部分から成る。

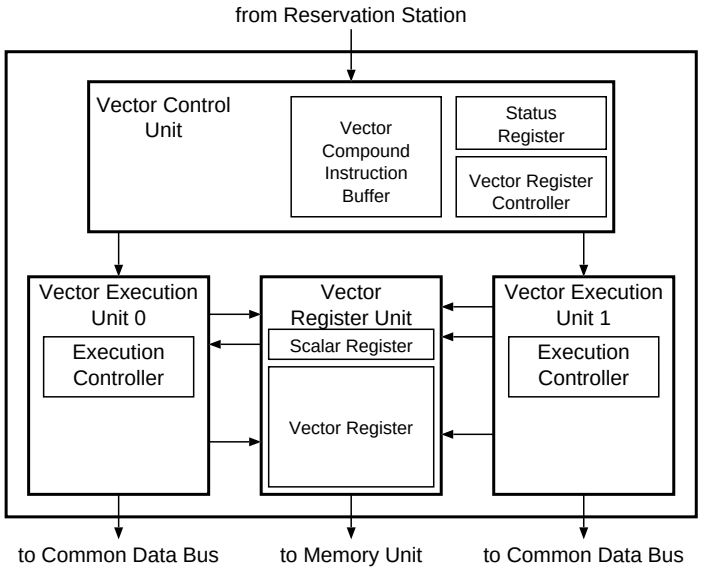


図 12.1: Vector Unit のブロック図

- Vector Control Unit

  演算ユニットの制御、命令発行を行い、後述する Vector Register の割り当て、解放を行う。Vector Length、Mask Bit などの Vector Unit による演算に必要な制御情報を管理する。

- Vector Register Unit

ベクトル演算を行うためのレジスタを持つ。このレジスタは Vector Execution Unit との接続ポートの他に Memory Unit との接続ポートを持ち、Memory とのデータ転送が行われる。RMT Processor は 512 個のレジスタを持つ。

- Vector Execution Unit

  Vector Register Unit からベクトル要素を取り出し、ベクトル演算を行い、結果を Vector Register Unit に格納する。

12.1.1 Vector Execution Unit

Vector Execution Unit のブロック図を図 12.2 に示す。

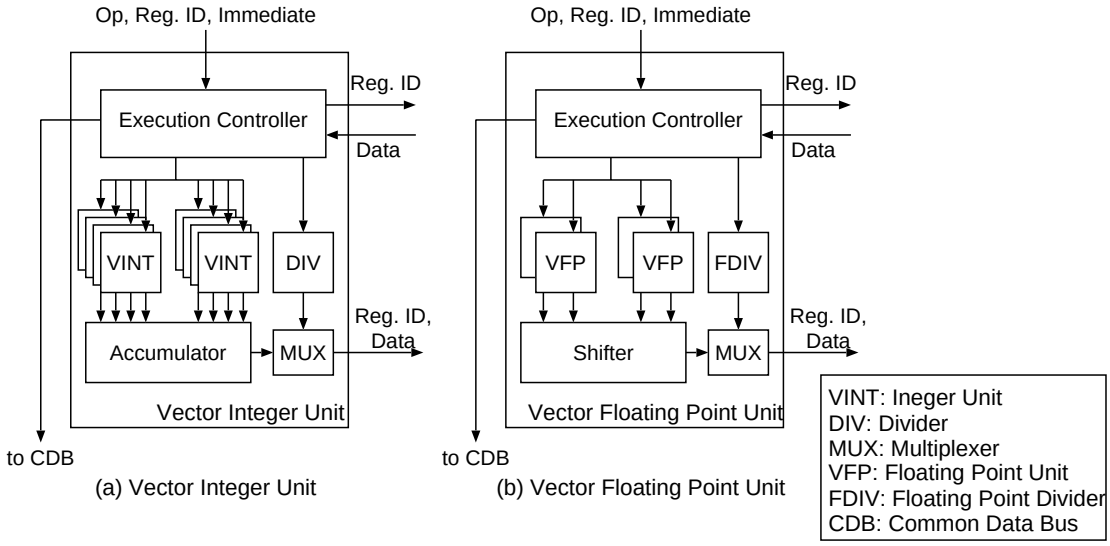


図 12.2: Vector Execution Unit のブロック図

Execution Controller は Vector Register Unit から必要なベクトルデータを取り出し、Vector Integer (Floating Point) Unit へ送る。

RMT Processor ではベクトル演算性能を向上させるために、Vector Integer Unit は 8 つ、Vector Floating Point Unit は 4 つの演算器を持つ。それぞれの演算器はパイプライン化され、1 クロックに 1 つの演算を開始する。

割り算は使用頻度が低いため、RMT Processor では Vector Divide Unit を 2 つある Vector Execution Unit のうちの片方のみ除算回路を持つ。

12.1.2 命令フォーマット

ベクトル演算命令のフォーマットは R-Type を拡張したものを用いる。Opecode フィールドには Vector Integer 命令用に 011110 (Word)、0x110110 (Paired HalfWord)、111110 (Quad Byte)、Vector Floating Point 命令用に 011111 (Double / Single)、111111 (Paired Single) を用いる。図 12.3 にベクトル演算命令のフォーマットを示す。

rs、rt フィールドはベクトル演算の Source Register、rd フィールドは Destination Register を指定する。function フィールドにはベクトル演算の種類を指定する。subfunc フィールドはベクトル演算命令により

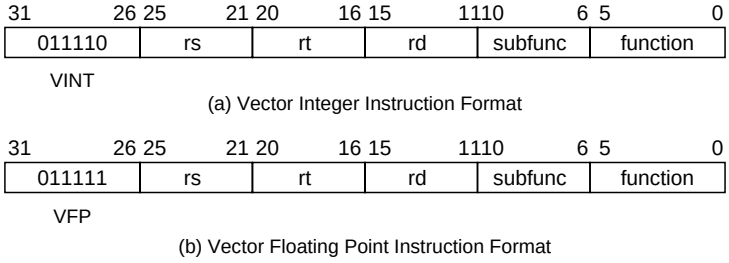


図 12.3: ベクトル演算命令フォーマット

用途が異なり、ベクトル - スカラ演算を行う際に用いる scalar bit や比較命令において比較条件を指定する cond bit、命令の順序制御を行う sync bit が含まれる。

12.2 Reserve/Release 命令

ベクトル演算を行うためには大きなベクトルレジスタが必要になる。これを各スレッドに持たせるとゲートサイズが大きくなり、また、ベクトル演算を行わないスレッドがある場合にはレジスタが無駄になる。よって 1 つのベクトルレジスタを用意し、それを複数のスレッドで共有して使用することによりベクトル演算を行う。ベクトルレジスタを必要な量だけ確保して使うことにより、複数のスレッドでベクトルレジスタを共有する。ベクトルレジスタの共有は、図 12.4 のようにベクトルレジスタを 4 つの領域に分け、128 エントリ、256 エントリ、512 エントリの固定サイズで確保する。スカラレジスタも同様に 4 つの領域に分け、確保したベクトルレジスタの大きさに応じて使用できるスカラレジスタの大きさが決定する。

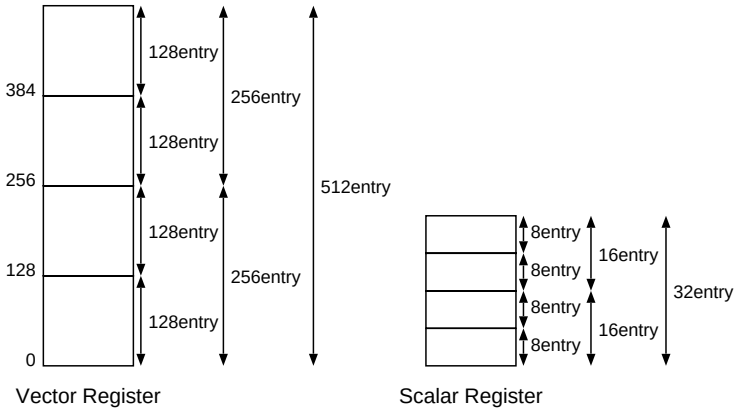


図 12.4: ベクトルレジスタのサイズ

確保したベクトルレジスタは、必要なベクトル長により分割される。ベクトル長は、8length、16length、32length、64length の中から選択する。図 12.5 に RMTProcessor で選択できるベクトルレジスタの構成を示す。

(a) は 128 個のベクトルレジスタを確保した場合の構成を示している。この場合、選択できる構成は、ベクトル長 8 のレジスタを 16 個、もしくはベクトル長 16 のレジスタを 8 個のどちらかとなる。(b) は 256 個のベクトルレジスタを確保した場合の構成で、ベクトル長 8 のレジスタを 32 個、ベクトル長 16 のレジスタを 16 個、ベクトル長 32 のレジスタを 8 個持つ構成の中から選択する。(c) は 512 個のベクトルレジ

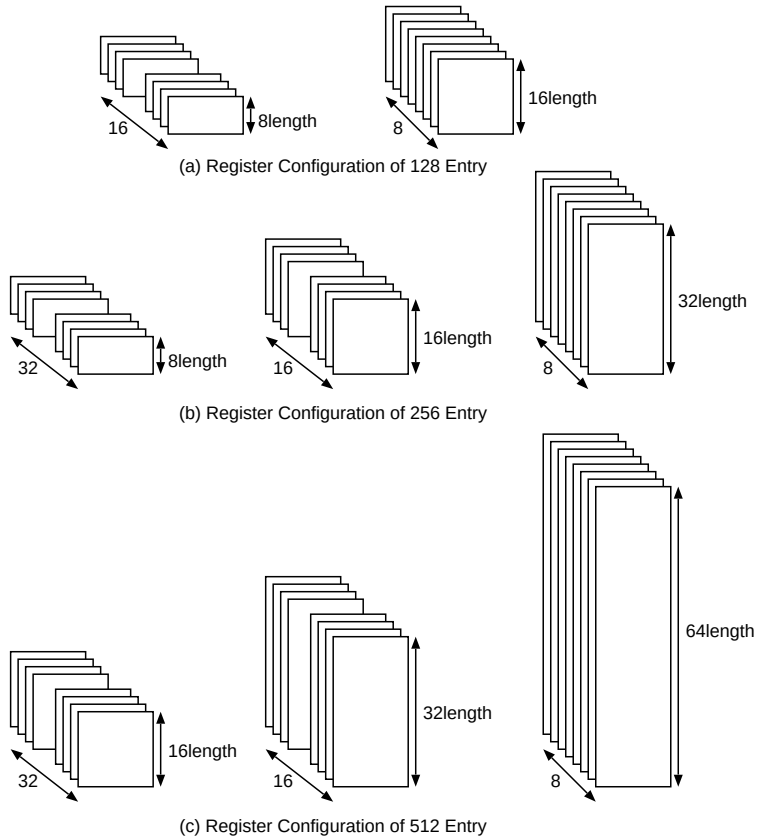


図 12.5: ベクトルレジスタの構成

スタを確保した場合で、ベクトル長 16 のレジスタを 32 個、ベクトル長 32 のレジスタを 16 個、ベクトル長 64 のレジスタを 8 個といった構成の中から一つを選択する。

Vector Unit で演算を行う場合、ベクトル演算を行う前にまず使用する分だけベクトルレジスタを確保する。ベクトルレジスタの確保は Vector Reserve 命令で行う。また Vector Unit で演算を行い、これ以上 Vector Unit を使用しなくなった場合は Vector Release 命令で確保していたベクトルレジスタを開放することにより、別のスレッドが新たに Vector Unit で演算を行うことが可能となる。Reserve、Release 命令を用いたプログラム例を図 12.6 に示す。

```
addu      $11, $0, 0x000A       # 256 Entry ( 32Depth x 8 ) Mode
virsv     $10, $11              # Reserve Instruction

== Vector Execution ==

virls     $10                   # Release Instruction
```

図 12.6: ベクトル演算のプログラム例

Vector Reserve 命令のオペランド (rs) で図 12.5 のどの構成でベクトルレジスタを使用するのかを指定する。指定する値 (Mode) は表 12.1 の中から選択する。値は上位 2bit が確保するレジスタの大きさ、下

位 2bit がベクトル長を示す。Vector Reserve 命令はベクトルレジスタの確保が成功すると rd に 1 を返す。rs で指定されたサイズのベクトルレジスタが確保できない場合は rd に 0 を返す。

表 12.1: Vector Register Mode の指定

(128 Entry)	
8 Depth × 16	0x4
16 Depth × 8	0x5
(256 Entry)	
8 Depth × 32	0x8
16 Depth × 16	0x9
32 Depth × 8	0xA
(512 Entry)	
16 Depth × 32	0xD
32 Depth × 16	0xE
64 Depth × 8	0xF

Vector Reserve 命令によりベクトルレジスタが確保されると、Vector Control Unit の Vector Register Controller 内にある Register Status Table に、確保したベクトルレジスタの情報を書きこむ。図 12.7 に Vector Status Table のフォーマットを示す。

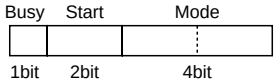


図 12.7: Vector Status Table

Busy Bit はそのスレッドがベクトルレジスタを確保しているかどうかを示す。Start Address は図 12.4 に示した 4 つに分割したベクトルレジスタのどの部分からベクトルレジスタを確保しているのかを示す。Mode は Vector Reserve 命令の rs で指定されたベクトルレジスタの構成を格納する。

Vector Release 命令を実行すると、確保していたベクトルレジスタを開放し、rd に 1 を返す。ベクトルレジスタを確保していない時に Vector Release 命令を実行すると rd に 0 を返す。

12.3 Status Register

Status Register はベクトル演算を行うために必要な以下の情報を各スレッドごとに保持する。

Status Register へのアクセスは vimfc, vfmfc (読み込み)、vimtc, vfmtc(書き込み) 命令を用いて行う。

12.4 複合演算命令

本ベクトル演算器では、ユーザが複合演算命令を定義し、複合演算実行命令 1 命令で定義された複合演算命令を処理することにより Vector Unit の使用率を向上させる。複合演算は Vector Control Unit の Compound Instruction Controller 内の Compound Instruction Buffer に定義する。Compound Instruction Buffer はベクトルレジスタやスカラレジスタと同じように 4 つに分割し、確保したベクトルレジスタと同

Address	Name	Description
0x00	Mask (Low)	ベクトルレジスタの要素 (下位) に対応し、1 を立てることにより演算をマスクする。マスクは最下位ビットが 1 番目の要素、最上位ビットが 32 番目の要素に対応する。
0x01 (Int)	Mask (High)	ベクトルレジスタの要素 (上位) に対応し、1 を立てることにより演算をマスクする。マスクは最下位ビットが 33 番目の要素、最上位ビットが 64 番目の要素に対応する。
0x01 (FP)	Rouding Mode	浮動小数点の丸めモードを指定する (0: Round to Nearest, 1: Round to 0, 2: Round to $+\infty$, 3: Round to $-\infty$)
0x02	Length	演算を行うベクトル長 (実際に指定するのはベクトル長 - 1) を指定する
0x03	Stride	Load / Store 時のアドレスのストライドを指定する。実際には各要素の間隔をワード数で指定する。0 を指定した場合は連続した番地からベクトル要素を読み込む。1 を指定すると 1 ワードおきに要素を読み込む。

じ領域を使うようにする。Compund Instruction Buffer 全体を 32 エントリであるため、ベクトルレジスタを 128 個確保した場合、使用できるエントリ数は 8 個、256 個確保した場合は、使用できるエントリ数は 16 個、512 個確保した場合は使用できるエントリ数は 32 個となる。

図 12.8 に Compound Instruction Buffer のフォーマットを示す。

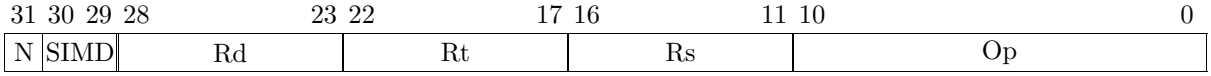


図 12.8: Compound Instruction Buffer のフォーマット

一つのエントリに一つの命令を定義し、それを複数合わせることにより複合演算命令を定義する。Next(N) bit は次のエントリに複合命令が続くことを示す。複合命令の最後の命令は Next Bit を 0 にする。Next Bit を 0 にして複合命令を区切ることにより、複数の複合命令を定義することができる。

SIMD フィールドには SIMD 演算を行う場合のビット幅を指定する。整数演算の場合、0x0 で 32bit 演算 (SIMD 演算を行わない)、0x1 で 16bit × 2 演算、0x2 で 8bit × 4 演算を行う。浮動小数点演算の場合、0x0 で通常の演算 (SIMD 演算を行わない)、0x1 で 32bit × 2 演算を行う。

Rs, Rt はソースレジスタ、Rd はデスティネーションレジスタを指定する。レジスタの指定は以下の通りである。

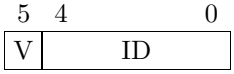


図 12.9: レジスタの指定

ベクトルレジスタを使用する場合、V ビットを 1 にする。スカラレジスタを使用する場合、V ビットを 0

にする。ID には使用するレジスタの ID を指定する。図 12.5 で指定した構成に従って ID の中で有効となるビット幅が決定する。例えば 8length × 16 個の構成では ID のうち下位 4 ビットが有効となる。16length × 32 個の構成では ID の 5 ビットが有効となる。

実際に使用される rs、rt、rd は次に述べる VIECI、VFECI 命令で指定された rs、rt、rd のオフセット値として用いられる。例えば VIECI 命令の rs が 1 で Compound Instruction Buffer の rs の ID が 3 の場合、実際に指定される Register ID は 1 + 3 の 4 となる。

operation には演算を行う命令を指定する。以下に整数演算の場合のフォーマットを示す。

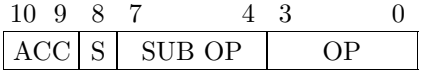


図 12.10: Operation のフォーマット (整数演算)

OP には以下を指定する。

- NOP (0x0)

  何も行わない。

- AND (0x1)

  論理積を計算する。

- OR (0x2)

  論理和を計算する。6 ビット目 (SUB OP) を 1 にすると NOR オペレーションとなる。

- XOR (0x3)

  排他的論理和を計算する。

- ADD (0x4)

  加算する。6 ビット目 (SUB OP) を 1 にすると減算となる。

- MULT (0x5)

  乗算する。4 ビット目 (SUB OP) を 1 にすると符号なし演算となる。6 ビット目 (SUB OP) を 1 にすると演算結果 (64bit 中) の上位 32 ビットを返す。

- SHIFT (0x6)

  シフト演算を行う。4 ビット目 (SUB OP) が 0 の場合は左シフト、1 の場合は右シフトとなる。6 ビット目 (SUB OP) が 1 の場合は算術シフトとなる。5 ビット目 (SUB OP) が 1 の場合はシフトではなくローテーションとなる。

- COMPARE (0x7)

  比較演算を行う。4 ビット目 (SUB OP) が 1 の場合はオペランドを符号無し数値として扱う。5-7 ビット目 (SUB OP) で比較条件を指定する。比較条件は 0x0: 常に偽、0x1: =、0x2: >=、0x3: >、0x4: 常に真、0x5: ≠、0x6: <、0x7: <= となる。

- THROUGH (0x8)

  rs の値を返す。

- MADD (0x9)

  Multiply and ADD 演算を行う。6 ビット目 (SUB OP) が 1 の場合減算となる。

- DIV (0xf)

  除算を行う。4 ビット目 (SUB OP) が 1 の場合、値を符号無しとして扱う。6 ビット目 (SUB OP) が 1 の場合、剰余演算となる。

S ビットを 1 にすると、SIMD 演算の場合にスカラ演算を行う。例えば 8bit × 4 演算の場合、S ビットを 1 にすると rt の下位 8bit を全てのフィールドで使用する。

ACC フィールドに 0x2 を指定すると、各ベクトル要素の演算結果を加算する。この場合、Rd はスカラレジスタを指定する必要がある。

以下に浮動小数点演算の場合のフォーマットを示す。

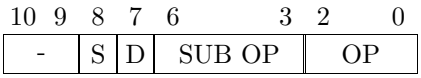


図 12.11: Operation のフォーマット (浮動小数点演算)

OP には以下を指定する。

- NOP (0x0)

  何も行わない。

- THROUGH (0x1)

  rs の値を返す。3 ビット目 (SUB OP) を 1 にすると符号反転を行なう。4 ビット目 (SUB OP) を 1 にすると絶対値を求める。

- ADD (0x2)

  加算する。4 ビット目 (SUB OP) を 1 にすると減算となる。

- MULT (0x3)

  乗算する。

- CONVERT (0x4)

  フォーマット変換を行う。4-3 ビット目 (SUB OP) で変換後のフォーマットを指定する。0x0: 単精度、0x1: 倍精度、0x2: 整数へ変換を行う。6 ビット目が 1 の場合ソースオペランドを整数値として扱う。

- COMPARE (0x5)

  比較を行う。5-3 ビット目 (SUB OP) で比較条件を指定する。0x0: False、0x1: Unorderd、0x2: Equal、0x3: Unorderd or Equal、0x4: Ordered or Less Than、0x5: Unordered or Less Than、0x6: Ordered or Less Than or Equal、0x7: Unorderded or Less Than or Equal。

- MADD (0x6)

  Multiply and Add 演算を行う。4 ビット目 (SUB OP) が 1 の場合減算となる。

- DIV (0x7)

  除算を行う。

D ビットが 1 の場合、オペランドを倍精度として扱う。D ビットが 0 の場合、オペランドを単精度として扱う。

S ビットを 1 にすると、SIMD 演算の場合にスカラ演算を行う。例えば 32bit × 2 演算の場合、S ビットを 1 にすると rt の下位 32bit を全てのフィールドで使用する。

複合演算命令は VIDCI、VFDCI 命令を用いて Compound Instruction Buffer に定義する。そして VIECI、VFECI 命令により複合命令の演算を開始する。それぞれの命令フォーマットを図 12.12 に示す。

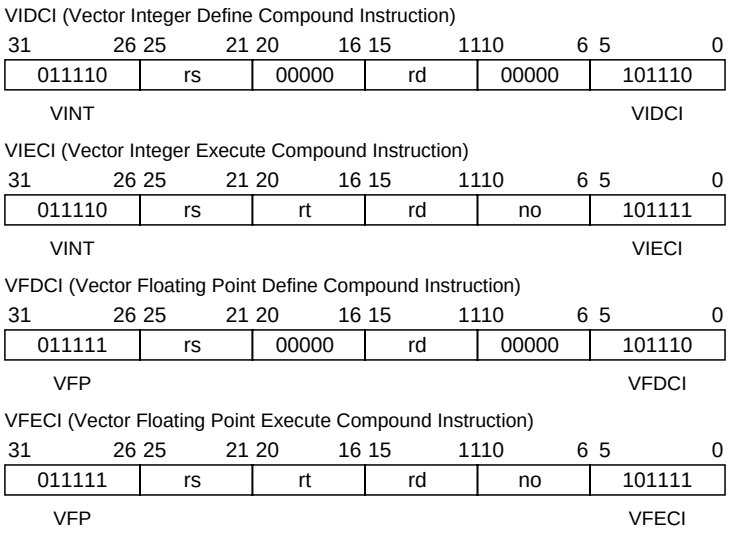


図 12.12: 複合演算命令のフォーマット

複合演算定義命令では rs に図 12.8 に従ったデータが入ったレジスタを指定し、rd に格納する Compound Instruction Buffer の ID を指定する。複合演算実行命令では rs、rt に Source Register、rd に Destination Register を指定し、no に実行を開始する Compound Instruction Buffer の位置を指定する。

複合演算実行命令が発行されると、Compound Instruction Controller は Compound Instruction Buffer から no に指定されたエントリの命令を読み出し、Register ID の変換を行ってから Vector Execution Unit へ読み出した命令を渡す。読み出した命令の Next Bit を見て 1 が立っていたら Compound Instruction Buffer の次のエントリから命令を読み出し演算を続ける。Next Bit が 0 ならばそこで複合演算を終了し、次の命令を受け付ける。

図 12.13 に複合演算命令の例を示す。例ではエントリの 0 から 1 でベクトルの加算をした後スカラレジスタの値で比較を行っている。また 2 から 9 で別の複合演算命令として、ベクトル変換命令を定義している。複合演算実行命令で no に 0 を指定するとベクトルの加算と比較を実行し、2 を指定するとベクトル変換命令を実行する。

	Next	Rd	Rt	Rs	Operation
0	1	V0	V0	V0	VADD
1	0	V1	S0	V1	VCMP
2	1	S0	V0	V0	VMAC
3	1	S1	V1	V1	VMAC
4	1	S2	V2	V2	VMAC
5	1	S3	V3	V3	VMAC
6	1	S4	V4	V4	VMAC
7	1	S5	V5	V5	VMAC
8	1	S6	V6	V6	VMAC
9	0	S7	V7	V7	VMAC

図 12.13: 複合演算の定義例

13

Responsive Link

13.1 概要

Responsive Link は，各種ロボット，自動車，プラント，ホームオートメーション等の種々の分散制御を実現するために必要なハードリアルタイム通信，及び，画像，音声等のマルチメディアデータを滑らかに伝送するために必要なソフトリアルタイム通信の両方を同時に可能にするように設計を行っている．特に，リアルタイムの理論をそのまま応用可能なように，パケットの追い越し機能を実現している．

Responsive Link は柔軟なリアルタイム通信を実現するために，

- 通信パケットに優先度を付け，高い優先度の通信パケットが低い優先度の通信パケットを通信ノード毎に追い越し
- ハードリアルタイム通信（データリンク）とソフトリアルタイム通信（イベントリンク）の分離
- 全く同じネットワークアドレス（送信元アドレス及び送信先アドレス）を持つ通信パケットの経路を優先度によって別の経路に設定することによって専用回線や迂回路を設け実時間通信を制御
- 通信パケットの優先度を通信ノード毎に付け替え可能にすることによってパケットの加減速を分散管理で制御
- ハードウェアによるフレーム単位のエラー訂正

という方法を組み合わせることによって，分散管理を用いて大規模かつ量子時間の小さい実時間通信を実現する．さらに，

- 通信速度を動的に変更可能
- トポロジーフリー，
- Hot-Plug&Play

等の様々な機能を実現する．

Responsive Link は国内では情報処理学会試行標準 (IPSJ-TS 2003:0006) として標準化されており，国際的にはでは ISO/IEC JTC1 SC25 WG4 において標準化作業が行われている．

13.2 *Responsive Link* のインタフェース

ソフトリアルタイム通信（以下，単にデータと呼ぶ）のデータサイズ（画像データ，音声データ等）は大きく，それに対してハードリアルタイム通信（以下，単にイベントと呼ぶ）のデータサイズ（制御コマンド，同期信号等）は非常に小さい．従って，従来型の 1 系統の通信路で全ての通信を行う方法では，同時に通信すべき通信データとして，大量のデータパケットと，ごくわずかではあるが分散リアルタイム制御用途には非常に重要なイベントパケットが同一種類のパケットとして存在する．データとイベントを，共有された同一の通信線を通して時分割に通信を行う従来方式ではイベント伝達の時間が正確にバウンドできないので，ハードリアルタイムシステムは実現困難であると考えられる．

また，複数のモジュールでひとつの通信チャネルを共有するシリアルバスでは，同時に何台のモジュールが通信するかによってバンド幅が動的に変化し時間をバウンドすることが困難であり，実効速度も出にくい．

さらに，リアルタイム通信におけるトレードオフとして，ソフトリアルタイム通信は主にバルク的なマルチメディアデータの通信等に用いられ，ハードリアルタイム通信は主に制御等に用いられるので，

- ソフトリアルタイム：バンド幅保証 ⇒
  スループットをできるだけ上げたい
- ハードリアルタイム：レイテンシ保証 ⇒
  レイテンシをできるだけ小さくしたい

という要求がある．しかしながらパケットサイズを大きくするとスループットは高くなるが，同時にレイテンシも長くなる．逆にパケットサイズを小さくするとレイテンシは短くなるが，オーバヘッドが大きくなりスループットが低くなる．

従って，*Responsive Link* では，データラインとイベントラインを分離し，かつ各ラインの結合形態を point-to-point の双方向シリアル通信として設計されている（図 13.1 参照）．以下，それぞれをデータリンク，イベントリンクと呼ぶ．データリンクではパケットサイズを固定長かつ大きめにしてソフトリアルタイム通信に使用し，イベントリンクではパケットサイズを固定長かつ小さめにしてハードリアルタイム通信に使用する．

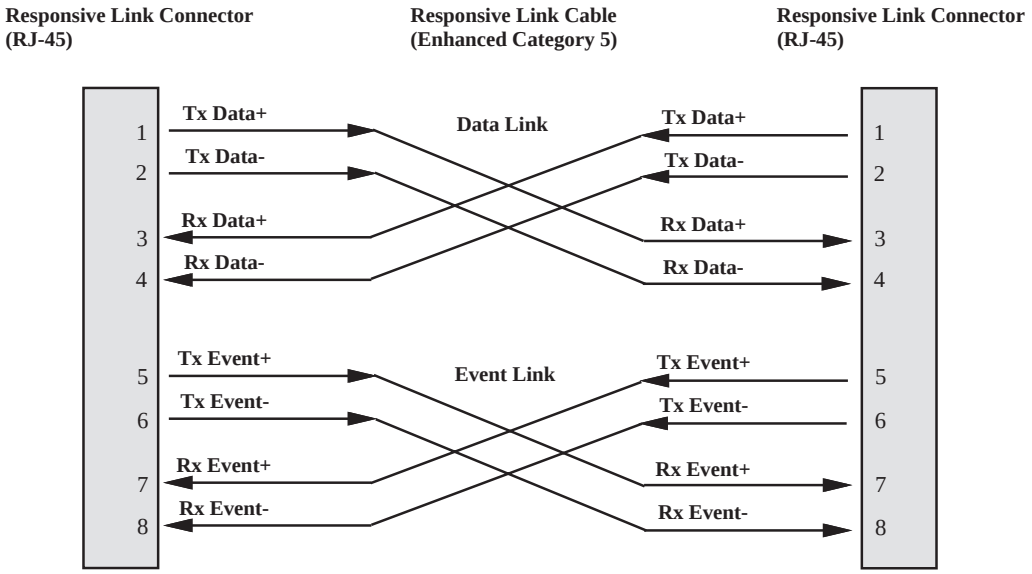


図 13.1: *Responsive Link* インタフェース

13.3 パケットフォーマット

図 13.2 に *Responsive Link* のパケットフォーマットを示す．通信パケットは，ヘッダ部，ペイロード部，トレイラ部から構成する．ヘッダ部は優先度付のネットワークアドレスから構成し，トレイラ部は制御情報とステータスから構成される．

通信パケットは固定長で，ハードリアルタイム通信用のイベントリンクのパケットサイズは 16 バイト（ペイロード：8 バイト）と小さく，ソフトリアルタイム通信用のデータリンクのパケットサイズは 64 バイト（ペイロード：56 バイト）と大きい．

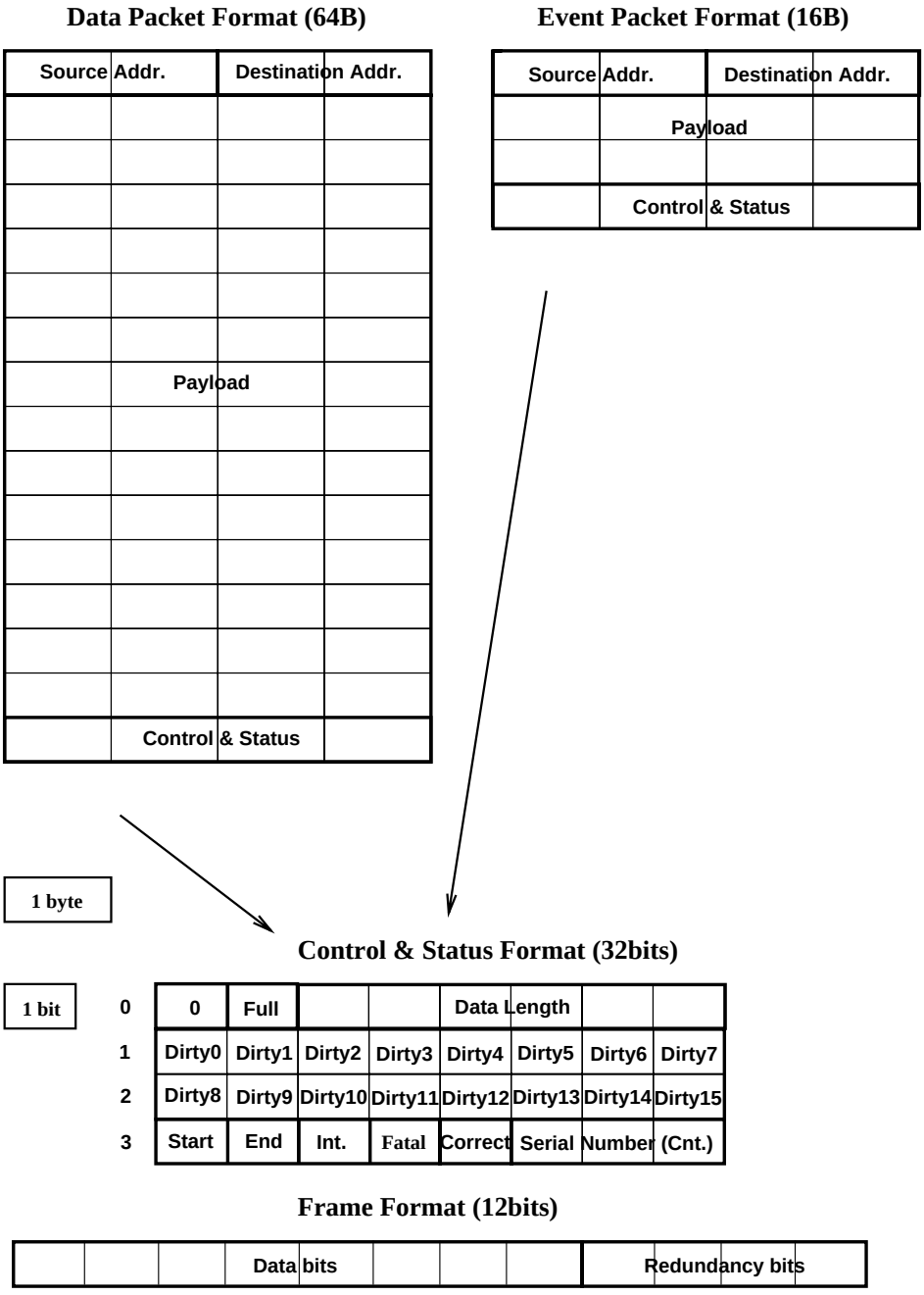


図 13.2: *Responsive Link* のパケットフォーマット

図 13.2 の通信パケットのヘッダ部に対して，図 13.3 に示すようにネットワークアドレスに優先度を付加する．256 レベル (8bit) の優先度を有し，優先度は 0 が一番低く，数字が大きくなるにしたがって高くなる．

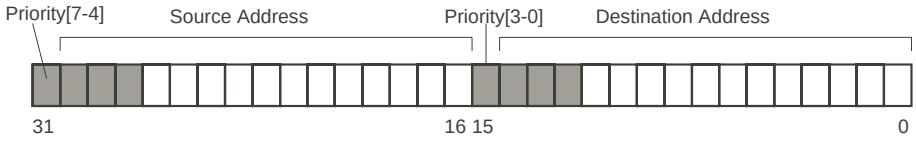


図 13.3: *Responsive Link* のヘッダフォーマット

Responsive Link の最大通信ノード数は，ネットワークアドレス長に制限され，優先度を使用しない場合，理論的には 2^{32} ノードとなる (図 13.3 参照)．*Responsive Link* の規格で推奨している使用法（ノード毎にノードアドレスを割り当て，12bit の送信元アドレス，12bit の送信先アドレス，8bit の優先度を用いてルーティングを行う）の場合には，$2^{12} = 4096$ ノードとなる．4096 よりノード数が大きなシステムを構築する際には，経路にアドレスを割り当てる（24bit のネットワークアドレスと 8bit の優先度を用いてルーティングを行う）ことにより $2^{24} = 16$M ノードまでのノード数をサポートする．

13.3.1 固定長（64B）のデータパケット

レスポンシブリンクのスイッチ部はカットスルー型のスイッチを採用している．データパケットは固定長 (64byte) で，パケットに優先度が付加されている．データパケットはアドレス（ソースとデスティネーション），ペイロード，ステータスから構成される．カットスルー型のスイッチなので，衝突が起きない限りデータはノードを経由して転送されるが，あるノードで衝突が起こった場合は，優先度の高いパケットが低いパケットを追い越すことができるようになっている．この機能によって従来までの集中管理型ではなく分散管理型のリアルタイム通信を実現している．

データパケットは，2byte の送信元アドレス・2byte の送信先アドレス・56byte のペイロード・4byte の制御・状態データの計 64byte より構成される．4byte の制御・状態データは以下のフォーマットをとる．

UD	ユーザ定義フラグ（任意に設定可能）
Full	ペイロード 56byte がすべて有効データで埋められているとき 1，それ以外は 0
Data Length	ペイロードの有効データ長．1 から 56 の値をとる．
Dirty0-15	パケットのどのワード（4byte）にエラーが存在するかを示すビット．パケットの 2 ワード目にエラーがある場合は Dirty1 が 1 となる．（ハードウェアによりセットされる）
Start	このパケットがスタートパケットであるとき 1，それ以外は 0
End	このパケットがエンドパケットであるとき 1，それ以外は 0
Int	このパケットを受け取る際に割り込みを生じるときは 1，それ以外は 0
Fatal	このパケットに致命的なエラーが存在するときは 1，それ以外は 0（ハードウェアによりセットされる）
Correct	このパケットの一部分にエラーが存在し，それが修復されたときは 1， それ以外は 0（ハードウェアによりセットされる）
Serial Number	パケットのシリアルナンバ．スタートパケットが 0，以降 0 から 7 までを繰り返す．

13.3.2 固定長（16B）のイベントパケット

イベントパケットも固定長 (16byte) で，送信元アドレス，送信先アドレス，ペイロード，ステータスから構成される．イベントの場合もノードで衝突がない限り，直接ノードを経由してルーティングされるが，衝突が生じた場合はデータの場合と同様に，優先順位に従ってパケットの追い越しを行なう．

4byte の制御・状態データは以下のフォーマットをとる．

UD	ユーザ定義フラグ（任意に設定可能）
Full	ペイロード 8byte がすべて有効データで埋められているとき 1，それ以外は 0
Data Length	ペイロードの有効データ長．1 から 8 の値をとる．
Dirty0-15	パケットのどのバイトにエラーが存在するかを示すビット．パケットの 2 バイト目にエラーがある場合は Dirty1 が 1 となる．（ハードウェアによりセットされる）
Start	このパケットがスタートパケットであるとき 1，それ以外は 0
End	このパケットがエンドパケットであるとき 1，それ以外は 0
Int	このパケットを受け取る際に割り込みを生じるときは 1，それ以外は 0
Fatal	このパケットに致命的なエラーが存在するときは 1，それ以外は 0（ハードウェアによりセットされる）
Correct	このパケットの一部分にエラーが存在し，それが修復されたときは 1， それ以外は 0（ハードウェアによりセットされる）
Serial Number	パケットのシリアルナンバ．スタートパケットが 0，以降 0 から 7 までを繰り返す．

13.3.3 優先度による追い越し機構

優先度を用いたパケットの追い越し機構を実現するために，追い越し用バッファと退避用外部記憶を有したネットワークスイッチを搭載している．図 13.4 は 5 入力 5 出力で一つの入力部当たり追い越し用バッファが 4 パケット分あるネットワークスイッチの構成を示している．（実際に RMTP に実装されている *Responsive Link* には 8 パケット分の追い越し用バッファが実装されている．）図 13.4 において，最後の数字はポート番号を示している．入力ポート (In0〜4) から入力された通信パケットは，通信ノードで衝突しない場合，そのまま出力ポート (Out0〜4) へ出力を行う．異なる入力ポートから入力された通信パケットが同じ出力ポートに出力を行なう場合，通信パケットに付加された優先度に従い，低い優先度の通信パケットを追い越し用バッファ（意味的には追い越され用バッファ）に貯めて出力を待たし，高い優先度の通信パケットを先に出力させる．高い優先度の通信パケットの出力の後に低い優先度の通信パケットを追い越し用バッファから出力ポートに出力し，優先度に従った通信パケットの追い越しを行う．

この際，内部のスイッチングは，ヘッダ部受信のオーバヘッド及びルーティングテーブルの参照時間を隠蔽するために図 13.4 のように 8bit パラレル（byte 単位）で行うように設計されている．

上記の通信パケットの追い越しを実現するために通信パケットの大きさと等しい追い越し用バッファを 8 本入力ポート側に搭載している．さらに，出力が待たされ続けている時に入力が入り続けバッファが溢れそうになった場合に，追い越し用バッファの内容を一時的に退避するための退避用外部記憶 (DDR SDRAM) を設けることができるようになっている．

図 13.5 は図 13.4 のネットワークスイッチのひとつの入力部の詳細を示している．図 13.5 において，最後の数字はポート番号を示している．通信パケットの追い越しを行うために，まず，入力ポート (In) から入力された通信パケットを，入力ポインタ (In-Pointer) で指し示されている追い越し用バッファ0 から追い越し用バッファ3 のうち使用されていない空バッファに書き込む．入力パケットのヘッダ部分は必ず全て受信し追い越し用バッファに書き込み，その受信されたヘッダを元に図 13.6 のようなルーティングテーブルを参照し出力ポート番号と優先度を得る．得られた出力ポート番号は図 13.5 のリンクストローブ (L0〜

L4) に書き込む．例えば L2 ビットが有効であればその入力パケットの出力先は出力ポート 2 であることを示す．

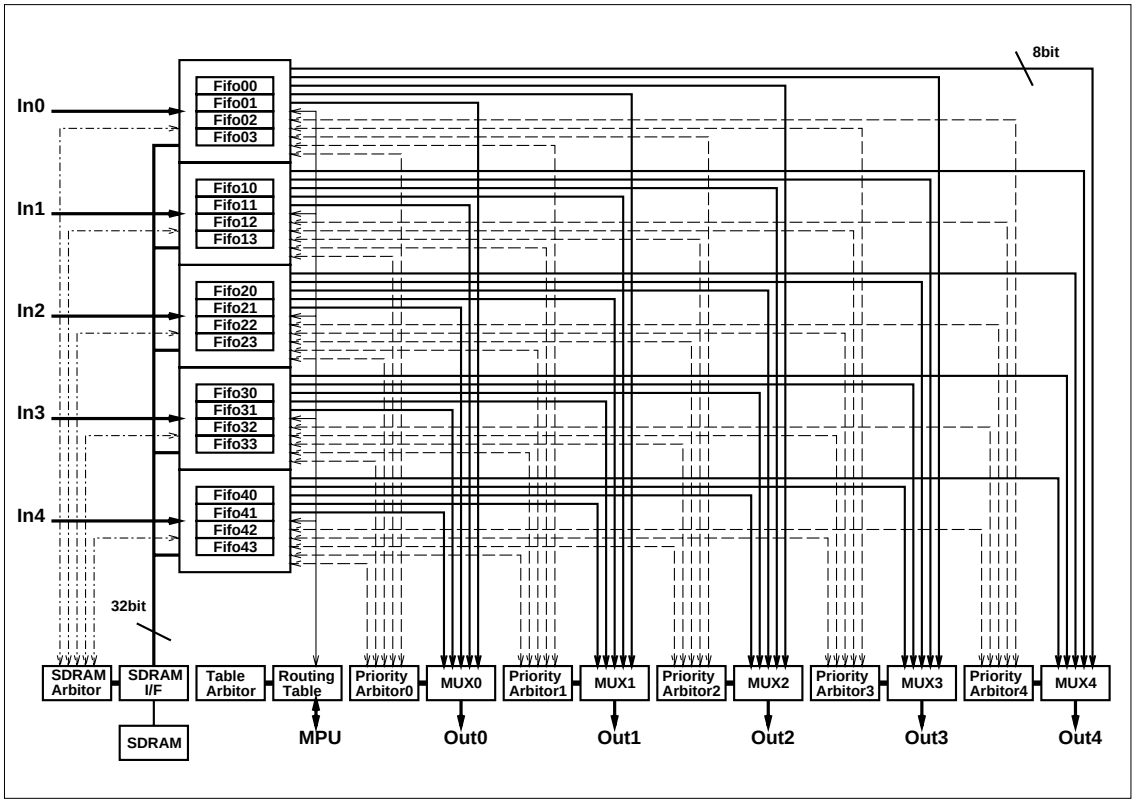


図 13.4: *Responsive Link* のネットワークスイッチ

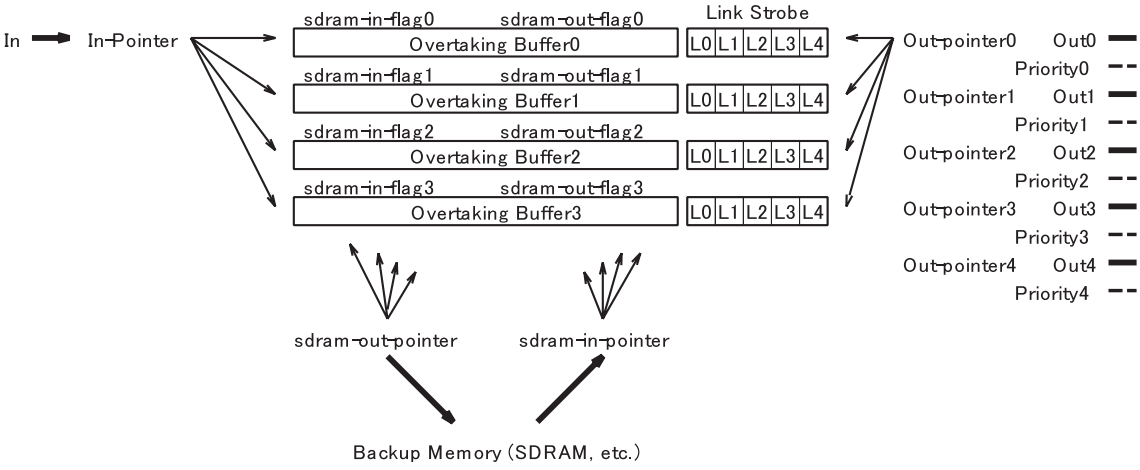


図 13.5: *Responsive Link* の追い越し用バッファ

図 13.5 において L0 から L4 までの複数ビットが有効であればマルチキャストを意味し，全て有効であればブロードキャストを意味する．入力部の出力側は出力ポート毎 (Out0〜Out4) にそれぞれ独立に各追

い越し用バッファのリンクストローブを参照し，自出力ポートのリンクストローブが有効な場合，出力側ポート側に配置された当該優先度調停器 (図 13.4 の Priority ArbitorN) に対して優先度と共に出力要求を行なう．図 13.5 の PriorityN は図 13.4 の Priority ArbitorN に接続されている．優先度調停器は，出力要求が一つの入力ポートからだけある場合はただちに出力許可を与え，出力要求が複数ある場合は優先度の一番高いものに出力許可を与えるようにする．一番優先度の高い要求が複数ある場合は，ラウンドロビン方式で出力許可を与える．

通信パケットの衝突がない場合や，衝突があってもその時点での最高優先度の通信パケットの場合は，ヘッダの受信とルーティングテーブル参照の遅延時間後に直ちに出力を開始する．入力部の各出力ポート側ではパケットの送信終了直後に対応するリンクストローブを無効にし，全てのリンクストローブが無効になったらそのバッファが空であることを意味する．

例えば，In-pointer が追い越し用バッファ1 を指している場合，入力ポート In から入力されたパケットは，まずヘッダ部が追い越し用バッファ1 に入る．次にそのヘッダを元にルーティングテーブルを引き，リンクストローブと優先度を得る．例えば，L1 と L3 が有効だった場合，Out-pointer1 と Out-pointer3 は共にその追い越し用バッファ1 を指し，Out1 と Out3 側が出力要求と共にその優先度をそれぞれ Priority1 と Priority3 に出力する．例えば，Out3 にすぐに出力可能であれば，出力許可が Priority Arbitor3 から与えられるので，直ちに追い越し用バッファ1 から Out3 に出力を開始する．出力が終われば，Out3 側が追い越し用バッファ1 の L3 をクリアする．また，Out1 には直ちに出力許可がおりなかったとすると，出力許可が得られるまで出力要求と優先度を Priorty1 に出力し続ける．ここで，Out1 への出力待ちの状態で，同じく Out1 へ出力したい高優先度パケットが新たに追い越し用バッファ2 に入ってきた場合，Out-pointer1 はより優先度の高いパケットの入っている追い越し用バッファ2 を指すようになり，その高優先度パケットの出力要求と優先度を Priority1 に出力するようになる．後から到着した高優先度パケットの出力が終わると，他に Out1 に出力したい高優先度パケットがない場合，Out-pointer1 は再び追い越し用バッファ1 を指して，同様に出力を継続しようとする．このように，同一系路上の先行する低優先度パケットが待たされている際にも，後続の高優先度パケットが追い越していくことを可能にする．

図 13.5 において，空バッファが少なくなっていき残り 1 本になってしまった場合，次の入力パケットは退避用外部記憶 (DDR SDRAM) に退避を行うようになっている．出力が進んで空バッファの残りが多くなり 2 本以上になると，退避用外部記憶に退避されていた入力パケットを優先度を考慮して追い越し用バッファに書き戻すことにより，出力を継続する．

また，退避用外部記憶が溢れそうになると，そのノードのプロセッシングコアに対して割り込みをかけられるようになっている．退避用外部記憶が溢れる場合は，アドミッションコントロールを行ってパケットの破棄を行ったり，送信元に送信データの一時停止を行うように制御する等の方法が考えられるが，そのプロトコル自身は *Responsive Link* の規格では定めていない．それらは上位のプロトコルで行うことになるので，上記割り込みをかける閾値を設定可能にするように設計している．

リアルタイム通信を実現するために，優先度によるパケットの追い越しをこのように再送を行なわななくてよいように設計されている．

13.4 フレームフォーマット

1byte は，図 13.2 の Frame Format ような冗長ビットを含めたフレームとしてシリアルに送受信される．詳細は低レベル通信の節を参照.

Data bits 8bit のデータ

Redundancy bits byte 毎に Redudancy bits（冗長ビット）を付加することで，CRC 等とは異なり，パケット全てを受信し終わらなくても byte 毎にエラー訂正が可能

13.5 ルーティング・テーブル

Responsive Link の経路制御は，図 13.6 に示すようなルーティングテーブル（経路制御表）を設定することによって行う．ルーティングテーブルは，*Responsive Link* コントローラ内に置き，そのノードのローカルプロセッサから読み書きできるようになっている．図 13.6 において，Reference 部はパケットのヘッダと同一であり，Referent 部に当該パケットに関する設定を行う．EE ビット及び DE ビットは，それぞれそのラインがイベントリンク用の設定かデータリンク用の設定かを示す．両方とも設定されていれば，両リンクとも同様の設定になる．L[4-0] は，前述のリンクストローブビットであり，出力ポート（複数可）を指し示す．

ルーティングテーブルの大きさ（エントリ数）は実装依存で有限となるため，非常に大きな分散システムを構築する際には溢れてしまう可能性がある．ルーティングテーブルに入りきらない大規模なシステムを構築する際には，ローカルノードプロセッサの主記憶上に完全なルーティングテーブルを用意し，*Responsive Link* コントローラ上のルーティングテーブルはキャッシュとして用いるようにする．つまり，TLB 付きの MMU とページテーブルを用いたメモリ管理と同様な管理手法を行うようにする．

そのために，ルーティングテーブルにヒットしないエントリがあった際には，ローカルノードのプロセッサに対して割り込みをかけると同時に，該当パケットを一時的に前述の退避用外部記憶に退避する．割り込みをかけられたプロセッサは，主記憶上の完全なルーティングテーブルをソフトウォークしてエントリを検索し，そのエントリを *Responsive Link* コントローラ上のルーティングテーブルの適切なエントリとスワップするようにする．（多くの場合，最近使用されていないエントリとスワップすると考えられるが，それは RT-OS のポリシ依存である．）*Responsive Link* コントローラ側は，イベントリンクとデータリンクそれぞれについて，LRU エントリアドレスが分かるように設計し，RT-OS に対してヒントを与えるようにする．その後，退避していたパケットを追い越しバッファに書き戻すことによって継続的にルーティングを実現する．

上記のような機構により，大規模な分散リアルタイムシステムが構築可能である．ただし，コントローラ内のルーティングテーブルに収まる範囲の規模でないと，厳密にハードリアルタイム性を維持するのは困難となる．

また，分散リアルタイムシステムの規模が大きくなればなるほど（つまりルーティングテーブルのサイズが大きくなればなるほど）通信のジッタは大きくなり，リアルタイム性の時間粒度も大きくなるが，近傍で激しく通信している経路をキャッシュに置き，そうでないものは主記憶上のルーティングテーブルに置く等の方法をとることにより，運用が可能であると考えられる．

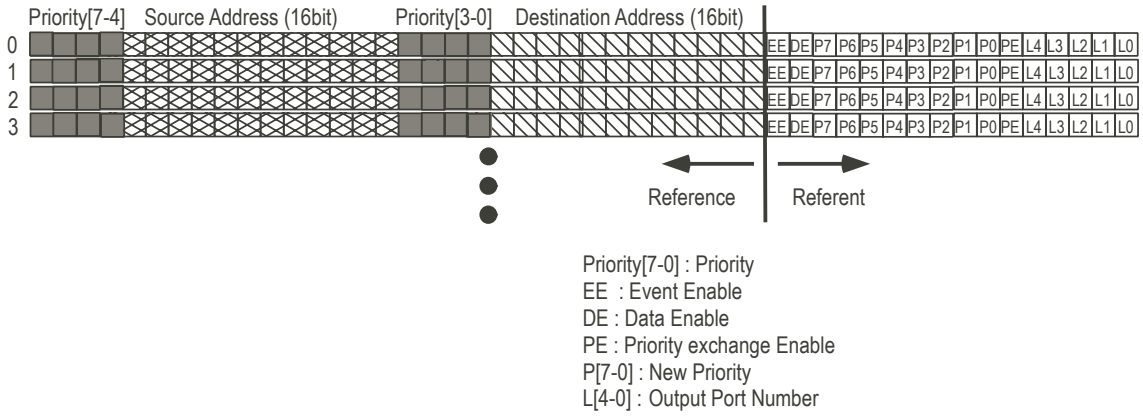


図 13.6: *Responsive Link* のルーティングテーブル

13.6 パケットの加減速制御

リアルタイム通信パケットの制御を外部から行うことができるようにするために，通信ノード毎にパケットの優先度の付け替えができるようにして，分散管理型でのリアルタイム通信の制御を実現している．

優先度の付け替えは，図 13.6 のルーティングテーブルを用いることによって行なう．図 13.6 において，ネットワークアドレスと優先度を元にルーティングテーブルを参照し出力ポート番号を決定する際に，優先度を付け替えないモード（図 13.6 の優先度付替ビット PE が無効）の場合は優先度はそのままであるが，優先度を付け替えるモード（優先度付替ビット PE が有効）の場合，出力ポートから出力する際に優先度 (Priority[7-0]) を新優先度 (P7〜P0) に置き換える．つまり，現ノードでの通信パケットの優先度は入力パケットのヘッダに付加されている優先度で決定され，その優先度に従って追い抜きやルーティングが決定されるが，次ノード以降での通信パケットの優先度を制御することができる．ルーティングテーブルの設定はソフトウェア（分散リアルタイムオペレーティングシステム等）で行ない，ルーティング（経路制御）自身はハードウェアで行なうようになっている．

このパケットの加減速制御機構により，例えば，リアルタイム通信の流量やレイテンシを監視するミドルウェアを用いて，リアルタイム通信の制御を可能とする．リアルタイム性の低い通信パケットがバルク的に流れていて，そのパケットが他のリアルタイム性の高いパケットの通信のリアルタイム性を阻害していたとしたら，通信監視ミドルウェアが当該パケットの優先度を下げることによって，リアルタイム性の制御を行うことができる．あるいは，あるノードでデッドラインミスが発生してしまった場合，その通信パケットの優先度を途中の経路で上げることにより（特にホットスポットで優先度を上げると効果的），次回からのデッドラインミスを防ぐことが可能となる．

13.7 優先度に従った経路制御

優先度に従って専用回線や迂回路を設けたり，データの流量の制御を行なうことができるように，全く同じネットワークアドレスを持つ通信パケットの経路を優先度によって別の経路に設定することができるようにしている．そのために，基本的にはネットワークアドレスと優先度の組でルーティングテーブルを参照する．

優先度ごとに必ずルーティングテーブルを設定しなければならないと煩雑であるので，デフォルトルートを設けることができる．ネットワークアドレスは同じであるが優先度が一致する組合わせ（経路）がルーティングテーブル上に無い場合には，最も優先度の低い優先度 0 の経路がデフォルト経路となるようになっている．つまり，

1. ネットワークアドレスと優先度の両方が一致すればその経路が第一優先
2. ネットワークアドレスは一致するが優先度が一致しない場合，優先度 0 の経路

となる．ここで，優先度 0 の経路はデフォルト経路となるので，途中で経路が消滅してしまわないようにルーティングテーブルに必ず登録する必要がある．

図 13.7 は，2 次元格子の交点に通信ノードがあるとし，全く同じ送信元から送信先に対して異なる優先度の通信パケットを同時に通信している状態を示す．例えば，優先度 0 のイベントリンクの経路上は別の通信ノードからの通信パケットも同じ経路を通って送信先に通るように設定しておき，優先度 3 の経路は送信元と送信先の優先度 3 の通信パケットしか通らないように設定しておくことにより，他の通信パケットと衝突が起きない専用回線を実現することができる．*Responsive Link* には優先度による追い越し機構があるが，衝突があると追い越しのために多少のオーバヘッドが生じてしまうので，このように優先度を用いてパケットの衝突が全くない専用回線を設定することにより，非常にレイテンシ及びジッタが小さいリアルタイム経路の実現を可能とする．また，優先度が異なる経路を複数設定することによってマルチリンクを実現し，バンド幅を広げることも同時に可能とする．

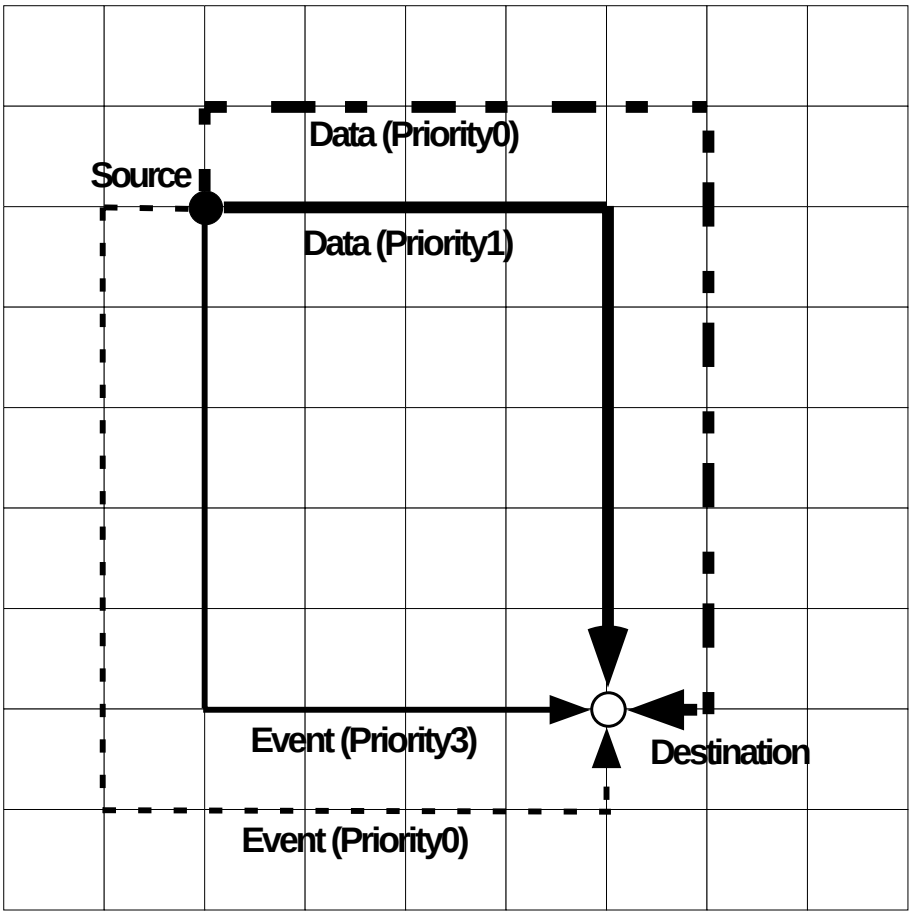


図 13.7: *Responsive Link* の優先度付経路

制御用の分散システムでは図 13.8 のような木構造を採る場合が多い．図 13.8 において通信ノード 0 から通信ノード 5 に通信する場合，優先度 0 の通信パケットは途中に通信ノード 1 と通信ノード 2 という中間ノードを経由して通信を行なうが，優先度 1 の通信パケットは通信ノード 0 から直接通信ノード 5 へ通信を行なうことができる．これは，例えばヒューマノイドロボットを開発した際に，当初は頭モジュール，肩モジュール，肘モジュール，指モジュールと接続しそれらの経路をホップして通信を行っていたが，設計後にどうしても頭モジュールと指モジュール間の通信レイテンシが間に合わないと判明した場合，後付で頭モジュールと指モジュールを直接接続し優先度を変えて通信することにより，容易に通信経路（この場合は専用回線）の増設を可能とする．この機能は，実システムを構築する際に手助けとなる．

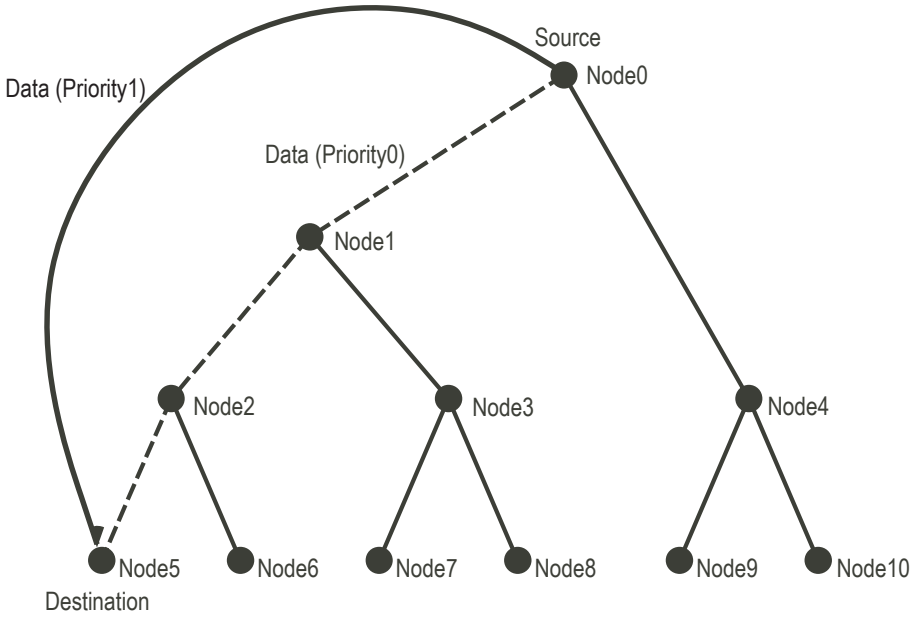


図 13.8: *Responsive Link* の優先度付木構造経路

13.8 低レベル通信

Responsive Link は分散制御用途であるので，必ずエラー訂正を行わなければならない．その際，できるだけエラー訂正によってリアルタイム性が損なわれないようにする必要がある．

ここで，パケット単位で CRC を付加しエラー訂正を行う方法では，パケット全体を受信しないとエラー訂正できない．その場合，ホップ毎にレイテンシが積算されていくので，リアルタイム通信用のエラー訂正としては好ましくない．そこで，レスポンシブリンクでは 1 ホップごとにフレーム（図 13.2 参照）単位でエラー訂正を行い，1 フレーム (8bit データ+4bit 冗長符号) につき 1bit のエラーであれば，再送することなしにハードウェアで誤り訂正を行うようにする．

13.8.1 CODEC

Responsive Link の CODEC は，8bit の情報ビット列に，誤り訂正用の 4bit の冗長ビット列を加えた 12bit を 1 フレームとして通信を行う．本 CODEC で行われる符合化は，以下のような流れとなる．

1. 巡回組織ハミング符合化（冗長ビット列を加える誤り訂正符合化）
2. Bit Stuffing（連続した 1 の符合に 0 を挿入）
3. NRZI 符合化

以下，各符合化について説明を行う．

13.8.2 巡回組織ハミング符号化

誤り訂正符合として，生成多項式が $x^4 + x + 1$ の巡回組織ハミング符合を採用する．この符合化では，8bit データの下位 (LSB) 側に 4bit の冗長ビット列を付加することで，12bit 中の任意の 1bit の誤りを受信

側で訂正することを可能にし，表 13.1 より誤りの位置を特定できる．送信時には，これら 12bit のビット列は，MSB 側から 1bit ずつ送信を行う．

表 13.1: シンドロームとエラーの位置

Syndrome	Error Position (4 redundancy bits)	Meaning
0000	00000000 0000	No error
0001	00000000 0001	Redundancy-bit error
0010	00000000 0010	Redundancy-bit error
0100	00000000 0100	Redundancy-bit error
1000	00000000 1000	Redundancy-bit error
0011	00000001 0000	0bit error
0110	00000010 0000	1bit error
1100	00000100 0000	2bit error
1011	00001000 0000	3bit error
0101	00010000 0000	4bit error
1010	00100000 0000	5bit error
0111	01000000 0000	6bit error
1110	10000000 0000	7bit error

13.8.3 Bit Stuffing

1 が長時間連続することによって引き起こされるリンクへの直流成分が発生や，受信側のビット同期への支障を回避するために，通信データ中に 5 つの連続した 1 が現れた場合には，その後ろに 0 を挿入する．

13.8.4 NRZI 符合化

最終的に送信される際に NRZI(Non Return to Zero Inverted) 符合化を行う．NRZI 符合化は，0 を送る場合にはリンクのデータビットを反転し，1 を送る場合にはデータビットの状態を前のまま保持する．

13.8.5 セットアップパターン

電源投入直後や，予期できないバースト的なリンクエラーなどの後は，送受信インタフェース間でフレーム同期がとれない場合がある．そのような場合，明示的にリンクの初期化を行うようにする．具体的には，以下に示すセットアップパターンを受信側に送信する．

セットアップパターン：000001111110

このパターンは，連続した 1 が 6 個以上は連続しないという bit stuffing の規則に反しているため，いかなる通常のパケットとも区別される．受信側では，このパターンを受信するとその後，最初に認識したフレームを，新しいパケットの第 1 フレームとして解釈する．

表 13.3: 通信速度とケーブル

Speed (Mbaud)	100	200	400
Maximum Length (m)	100	80	60
Recommendable Cable	Cat5e	Cat5e	Cat6

13.8.6 DPLL を用いたビット同期

受信側に DPLL(Digital Phase Lock Loop) 機構を設計し，受信用クロックの立ち上がりエッジに同期して受信信号をサンプリングする．1bit 転送あたりのサンプリング数はソフトウェアの設定によって可変 (4,8,16,32,64,128,256) にする．DPLL では設定された周期ごとに受信用クロックを生成し，受信信号のエッジを検出することにより，信号のエッジ間の中央で受信用クロックが立ち上がるように，受信用クロックの周期を微調整を行う．表 13.2 に DPLL のモードを示す.

モード名	p_mode2	p_mode1	p_mode0	d_clk 周期/1bit 転送
Mode2	1	1	1	2
Mode4	0	0	0	4
Mode8	0	0	1	8
Mode16	0	1	0	16
Mode32	0	1	1	32

表 13.2: DPLL モードの設定

13.8.7 エラーの取扱い

Responsive Link では，誤り訂正符合化によって 1[bit/frame] の誤りまでは自動的にエラー訂正を行うことができる．エラーの箇所を受信側で特定するために，図 13.2 のトレイラ部の Dirty ビットを立てる．具体的には，データリンクの場合ワード (4byte) 単位で，イベントリンクの場合バイト単位で，エラーのあった場所の Dirty ビットを立てる．エラーがハードウェアによって訂正されても，訂正しきれなくても Dirty ビットは立てるようにする．また，そのパケット中に 1 箇所でもエラー訂正が行われハードウェアで訂正しきれた場合，トレイラ部の Correct ビットを立てる．エラー訂正不可能だった場合，Fatal ビットを立てる．受信側のアプリケーションでは，これらを参考にし，例えば，受信データを本当に制御に使用してよいかどうか等を判断することを可能にする．

13.8.8 通信速度

Responsive Link の通信（変調）速度は，様々な環境（コンフィギュレーション，アプリケーション）を想定し，400, 200, 100, 50, 12.5, 6.25 [Mbaud] の範囲で段階的に可変とする．

表 13.3 に，通信速度と最大通信距離，推奨ケーブルの関係を示す．例えば，最大変調速度 400[Mbaud] で通信する場合，ケーブルには Category6 を使用し，最大通信距離は 60[m] 以内である．この場合，DPLL の基準周波数にはデューティ比が 1 対 1 の 800[MHz] のアップダウンエッジを使用し，サンプリング数 4 で DPLL を行うことによって実現する．

レスポンシブプロセッサは組み込み用途を想定しているので，消費電力が大きな問題となる．一般に通信速度（動作周波数）を速くすれば消費電力が大きくなり，遅くすれば小さくなる．通信速度の変更は，受信クロックを変更するのではなく DPLL のサンプリング数を変更することによって行う．従って，通信速度が遅い場合の通信は，通信速度が遅くなることによる安定性の増加と DPLL のサンプリング数が増加することによる安定性の増加という 2 重の恩恵を受ける．

13.9 メモリマップ

レスポンシブリンク部のアドレスマップは以下の通りである．

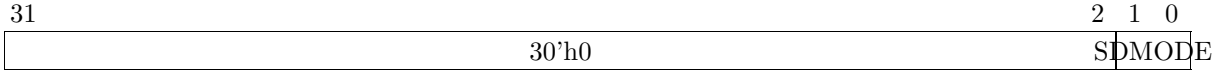
デコードアドレス	接続される I/O
0xFFFE_0xxx	レスポンシブリンク内部レジスタ
0xFFFE_1xxx	レスポンシブリンク用 IRC (r/w)
0xFFFE_2xxx	ルーティングテーブルアドレス部 (r/w)
0xFFFE_3xxx	ルーティングテーブルリンク部 (r/w)
0xC0xx_xxxx	イベント入力用 DPM (r)
0xC4xx_xxxx	イベント出力用 DPM (r/w)
0xC8xx_xxxx	データ入力用 DPM (r)
0xCCxx_xxxx	データ出力用 DPM (r/w)

初期アドレス: 0xfffe0000

13.10 レジスタマップ

13.10.1 SDRAM モードレジスタ

オフセット: 0x0000



Responsive Link は，パケット追い越し用に外付けの SDRAM を付けることができる．SDMODE(SDram MODE) レジスタは，パケット追い越し用外付け DDR SDRAM の有無と大きさを示す．外付け SDRAM を搭載しない場合は，内蔵の追い越しバッファ（各リンク 8 パケット分）のみで優先度付きパケットの追越を行う．

bit 名	機能
29'h0	0
SDMODE	Default 000 000 : 外付け SDRAM なし 001 : 外付け SDRAM あり，容量： 8MB 010 : 外付け SDRAM あり，容量： 16MB 011 : 外付け SDRAM あり，容量： 32MB 100 : 外付け SDRAM あり，容量： 64MB 101 : 外付け SDRAM あり，容量： 128MB 110 : 外付け SDRAM あり，容量： 256MB 111 : 外付け SDRAM あり，容量： 512MB

13.10.2 レスポンシブリンク速度設定レジスタ

オフセット: 0xFFFE_0004
属性 リード／ライト

31 28	27 25	24 22	21 19	18 16	15 12	11 9	8 6	5 3	2 0
-	Data4	Data3	Data2	Data1	-	Event4	Event3	Event2	Event1

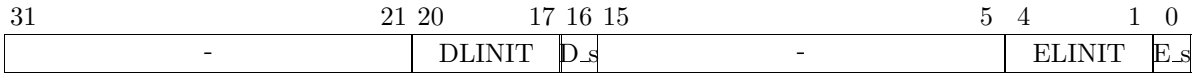
RSL(*Responsive Link* Speed): Default 000
本レジスタはレスポンシブリンクの変調速度を示す.
111 : 800 Mbaud
000 : 400 Mbaud
001 : 200 Mbaud
010 : 100 Mbaud
011 : 50 Mbaud

bit 名	機能
Data4	Data Link 4 用 RSL
Data3	Data Link 3 用 RSL
Data2	Data Link 2 用 RSL
Data1	Data Link 1 用 RSL
Event4	Event Link 4 用 RSL
Event3	Event Link 3 用 RSL
Event2	Event Link 2 用 RSL
Event1	Event Link 1 用 RSL

13.10.3 レスポンシブリンク初期化レジスタ

オフセット: 0xFFFE_0008

属性 リード／ライト



RLINIT(*Responsive Link* INITialization) レジスタはレスポンシブリンクのスイッチの初期化およびエンコーダ／デコーダ部分の初期化を行なう.

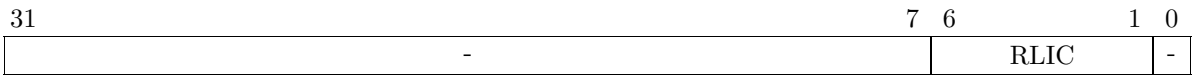
0: 通常動作

1: 初期化

bit 名	機能
DLINIT	Data link の各エンコーダ／デコーダの初期化 DLINIT[4]: RLINIT[20]: Data link4 の初期化 DLINIT[3]: RLINIT[19]: Data link3 の初期化 DLINIT[2]: RLINIT[18]: Data link2 の初期化 DLINIT[1]: RLINIT[17]: Data link1 の初期化
D_s	Data link switch の初期化
ELINIT	Event link の各エンコーダ／デコーダの初期化 ELINIT[4]: RLINIT[4]: Event link4 の初期化 ELINIT[3]: RLINIT[3]: Event link3 の初期化 ELINIT[2]: RLINIT[2]: Event link2 の初期化 ELINIT[1]: RLINIT[1]: Event link1 の初期化
E_s	Event link switch の初期化

13.10.4 レスポンシブリンク割り込みクリアレジスタ

オフセット: 0xFFFE_000C 属性 リード／ライト



RLIC(*Responsive Link* Irq Clear) レジスタはイベントリンクの割り込み要求のクリアを行なう.

Default 0

0: 通常動作

1: クリア

bit 名	機能
RLIC[1]	Data-Out EOP(End Of Packet) IRQ Clear: データパケットが DPM の設定した範囲から送信された場合に生じる割り込みのクリア
RLIC[2]	Event-Out EOP IRQ Clear: イベントパケットが DPM の設定した範囲から送信された場合に生じる割り込みのクリア
RLIC[3]	Data-In EOP IRQ Clear: データパケットが DPM の設定した範囲に受信された場合に生じる割り込みのクリア
RLIC[4]	Event-In EOP IRQ Clear: イベントパケットが DPM の設定した範囲に受信された場合に生じる割り込みのクリア
RLIC[5]	Data Packet-In IRQ Clear: 割り込みビットの設定されたデータパケットが到着した場合に生じる割り込みのクリア
RLIC[6]	Event Packet-In IRQ Clear: 割り込みビットの設定されたイベントパケットが到着した場合に生じる割り込みのクリア

13.10.5 レスポンシブリンク送信停止割り込みクリアレジスタ

オフセット: 0xFFFE_0010 属性 リード／ライト

31 21	20 16	15 5	4 0
-	DWIRQC	-	EWIRQC

Responsive Link はパケット追い越し用 SDRAM を使用している際には追い越し用 SDRAM が溢れそうになると送信停止割り込みを自動生成する．同様に，追い越し用 SDRAM を使用していない際には，追い越し用バッファが溢れそうになると送信停止割り込みを自動生成する．本 WIRQC(Wait IRQ Clear) レジスタはレスポンシブリンク送信停止割り込み要求のクリアを行なう．

Default 0

0: 通常動作

1: クリア

bit 名	機能
DWIRQC	Data link WIRQC DWIRQC[4]: WIRQC[20]: Data link4 DWIRQC[3]: WIRQC[19]: Data link3 DWIRQC[2]: WIRQC[18]: Data link2 DWIRQC[1]: WIRQC[17]: Data link1 DWIRQC[0]: WIRQC[16]: Data link0(CPU)
EWIRQC	Event link WIRQC EWIRQC[4]: WIRQC[4]: Event link4 EWIRQC[3]: WIRQC[3]: Event link3 EWIRQC[2]: WIRQC[2]: Event link2 EWIRQC[1]: WIRQC[1]: Event link1 EWIRQC[0]: WIRQC[0]: Event link0(CPU)

13.10.6 レスポンシブリンク継続割り込みクリアレジスタ

オフセット: 0xFFFE_0014 属性 リード／ライト

31 21	20 16	15 5	4 0
-	DCIC	-	ECIC

Responsive Link は，SDRAM に退避されたパケットがスイッチに書き戻された（再度送信された）際にレスポンシブリンク継続割り込み CI(Coutinuous Irq) を発生する．CIC(Continuous Irq Clear) レジスタはその割り込み要求 CI のクリアを行なう．

Default 0

0: 通常動作

1: クリア

bit 名	機能
DCIC	Data CIC DCIC[4]: CIC[20]: Data link4 DCIC[3]: CIC[19]: Data link3 DCIC[2]: CIC[18]: Data link2 DCIC[1]: CIC[17]: Data link1 DCIC[0]: CIC[16]: Data link0(CPU)
ECIC	Event CIC ECIC[4]: CIC[4]: Event link4 ECIC[3]: CIC[3]: Event link3 ECIC[2]: CIC[2]: Event link2 ECIC[1]: CIC[1]: Event link1 ECIC[0]: CIC[0]: Event link0(CPU)

13.10.7 レスポンシブリンク致命的エラー割り込みクリアレジスタ

オフセット: 0xFFFE_0018 属性 リード／ライト

31 21	20 16	15 5	4 0
-	DFIC	-	EFIC

Responisve Link は，各リンクの受信パケットにハードウェアで回復不可能なエラーがあった場合にレスポンシブリンク致命的エラー割り込み FI(Fatal Irq) を発生する．FIC(Fatal Irq Clear) レジスタは，その割り込み要求 FI のクリアを行なう．

Default 0

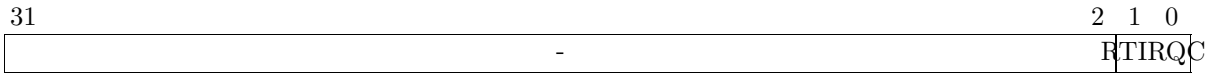
0: 通常動作

1: クリア

bit 名	機能
DFIC	Data FIC DFIC[4]: FIC[20]: Data link4 DFIC[3]: FIC[19]: Data link3 DFIC[2]: FIC[18]: Data link2 DFIC[1]: FIC[17]: Data link1 DFIC[0]: FIC[16]: Data link0(CPU)
EFIC	Event FIC EFIC[4]: FIC[4]: Event link4 EFIC[3]: FIC[3]: Event link3 EFIC[2]: FIC[2]: Event link2 EFIC[1]: FIC[1]: Event link1 EFIC[0]: FIC[0]: Event link0(CPU)

13.10.8 レスポンシブリンクルーティングテーブル割り込みクリアレジスタ

オフセット: 0xFFFE_001C 属性 リード／ライト



Responsive Link は，ルーティングテーブルにマッチするエントリが無かった場合にレスポンシブリンクルーティングテーブル割り込み (RTIRQ) を発生する．RTIRQC(Routing Table IRQ Clear) レジスタは，その割り込み要求 RTIRQ のクリアを行なう．

Default 0

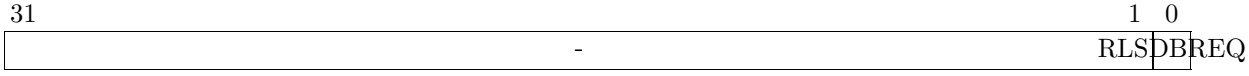
0: 通常動作 (r) ／割り込みクリア (w)

1: 割り込み状態 (r) ／割り込み発生（デバッグ用）(w)

bit 名	機能
RTIC[0]	Event Routing Table IRQ Clear
RTIC[1]	Data Routing Table IRQ Clear

13.10.9 レスポンシブリンク SDRAM バスリクエストレジスタ

オフセット: 0xFFFE_0020 属性 リード／ライト

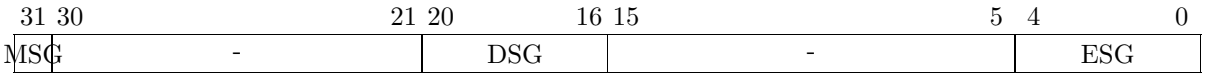


Responsive Link の追い越し用 SDRAM のバスには，*Responsive Link* とプロセッサバスの 2 つのバスマスタが接続されている．通常，プロセッサ側から追い越し用 SDRAM にアクセスする際には，データのトランザクション毎に，バス権の調停が行われている．プロセッサ側からバースト的に追い越し用 SDRAM をアクセスしたい場合には，本ビットを有効にすることで，追い越し用 SDRAM バスのバス権をプロセッサ側（プロセッサや DMAC 等）が常に得ることができる.（本ビットを設定しなくてもアクセス可能である.）*Responsive Link* 側が追い越し用 SDRAM バスを参照できなくなる (パケットの退避・復帰ができなくなる) という副作用がある.

bit 名	機能
RLSDBREQ	RLSDBREQ (*Responsive Link* SDram-Bus REQuest) : Default 1 本ビットはレスポンシブリンクの SDRAM バスへの明示的なバスリクエストを行なう. 0: バスリクエストイネーブル 1: バスリクエストディスエーブル

13.10.10 レスポンシブリンク SDRAM バスグラントレジスタ

オフセット: 0xFFFE_0024 属性 リード／ライト



RLSDBGRNT(*Responsive Link* SDram Bus GRaNT) レジスタは，追い越し用 SDRAM バスのバスグラント（どのバスマスタがバス権を有しているか）を示す.

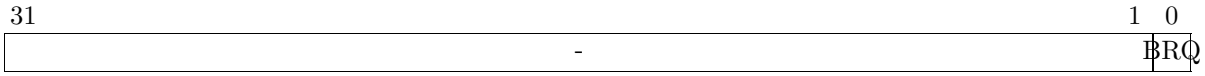
0: バス権獲得

1: バス権開放

bit 名	機能
MSG	Mpu Sdram bus Grant: MPU がバス権を得ている
DSG	Data link Sdram bus Grant: Data Link がバス権を得ている DSG[4]: RLSDBGRNT[20]: Data link4 DSG[3]: RLSDBGRNT[19]: Data link3 DSG[2]: RLSDBGRNT[18]: Data link2 DSG[1]: RLSDBGRNT[17]: Data link1 DSG[0]: RLSDBGRNT[16]: Data link0(CPU)
ES	Event link Sdram bus Grant: Event Link がバス権を得ている ESG[4]: RLSDBGRNT[4]: Event link4 ESG[3]: RLSDBGRNT[3]: Event link3 ESG[2]: RLSDBGRNT[2]: Event link2 ESG[1]: RLSDBGRNT[1]: Event link1 ESG[0]: RLSDBGRNT[0]: Event link0(CPU)

13.10.11 レスポンシブリンクルーティングテーブルバスリクエストレジスタ

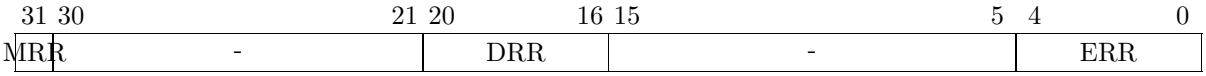
オフセット: 0xFFFE_0028 属性 ライト



Responsive Link のルーティングテーブルのバスには，*Responsive Link* とプロセッサバスの 2 つのバスマスタが接続されているが，デフォルトのバスマスタは *Responsive Link* である．プロセッサ側からルーティングテーブルをアクセスしたい場合には，本ビットを有効にすることで，ルーティングテーブルバスのバス権をプロセッサ側（プロセッサや DMAC 等）が得ることができる．*Responsive Link* 側がルーティングテーブルを参照できなくなる（パケットのルーティングができなくなる）という副作用がある．

bit 名	機能
BRQ	RLTBLBREQ (*Responsive Link* rouging TaBLe Bus REQuest): Default 1 本ビットはレスポンシブリンクのルーティングテーブルバスへのバスリクエストを行なう． 0: バスリクエストイネーブル 1: バスリクエストディスエーブル

オフセット: 0xFFFE_0028 属性 リード

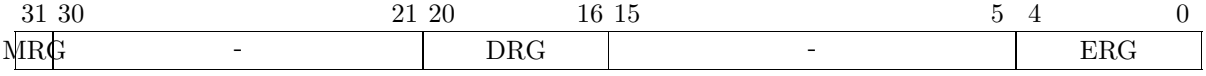


本ビットはプロセッサバス側からレスポンシブリンクのルーティングテーブルバスへのバスリクエストを示す.
0: バスリクエスト有
1: バスリクエスト無

bit 名	機能
MRR	Mpu Routing table bus Request
DRR	Data link Routing table bus Request DRR[4]: RLTBLBREQ[20]: Data link4 DRR[3]: RLTBLBREQ[19]: Data link3 DRR[2]: RLTBLBREQ[18]: Data link2 DRR[1]: RLTBLBREQ[17]: Data link1 DRR[0]: RLTBLBREQ[16]: Data link0(CPU)
ER	Event link Routing table bus Request ERR[4]: RLTBLBREQ[4]: Event link4 ERR[3]: RLTBLBREQ[3]: Event link3 ERR[2]: RLTBLBREQ[2]: Event link2 ERR[1]: RLTBLBREQ[1]: Event link1 ERR[0]: RLTBLBREQ[0]: Event link0(CPU)

13.10.12 レスポンシブリンクルーティングテーブルバスグラントレジスタ

オフセット: 0xFFFE_002C 属性 リード

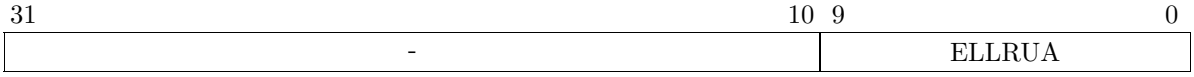


RLTBLBGRNT (*Responsive Link* routing TaBLe Bus GRaNT) レジスタは，レスポンシブリンクのルーティングテーブルバスのバスグラント（どのバスマスタがバス権を有しているか）を示す.
0: バス権獲得
1: バス権開放

bit 名	機能
MRG	Mpu Routing table bus Grant: MPU がバス権を得ている
DRG	Data link Routing table bus Grant: Data Link がバス権を得ている DRG[4]: RLTBLBGRNT[20]: Data link4 DRG[3]: RLTBLBGRNT[19]: Data link3 DRG[2]: RLTBLBGRNT[18]: Data link2 DRG[1]: RLTBLBGRNT[17]: Data link1 DRG[0]: RLTBLBGRNT[16]: Data link0(CPU)
ERG	Event link Routing table bus Grant: Event Link がバス権を得ている ERG[4]: RLTBLBGRNT[4]: Event link4 ERG[3]: RLTBLBGRNT[3]: Event link3 ERG[2]: RLTBLBGRNT[2]: Event link2 ERG[1]: RLTBLBGRNT[1]: Event link1 ERG[0]: RLTBLBGRNT[0]: Event link0(CPU)

13.10.13 イベントリンク LRU アドレスレジスタ

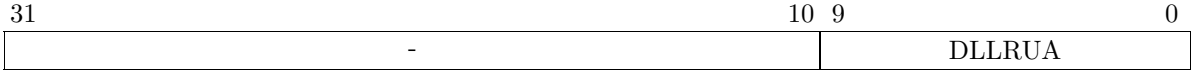
オフセット: 0xFFFE_0030 属性 リード



bit 名	機能
ELLRUA	ELLRUA (Event Link LRU Address) レジスタはイベントリンクのルーティングテーブル中で，最も近くに使用されたテーブルの格納されているアドレスを示す.

13.10.14 データリンク LRU アドレスレジスタ

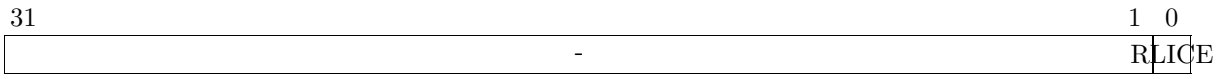
オフセット: 0xFFFE_0034 属性 リード



bit 名	機能
DLLRUA	DLLRUA (Data Link LRU Address) レジスタはデータリンクのルーティングテーブル中で，最も近くに使用されたテーブルの格納されているアドレスを示す.

13.10.15 レスポンシブリンク用割り込みコントローライネーブルレジスタ

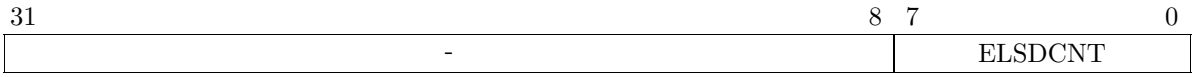
オフセット: 0xFFFE_0038 属性 リード



bit 名	機能
RLICE	RLICE (*Responsive Link* Interrupt Controller Enable) レジスタはレスポンシブリンク用割り込みコントローラ RLIRC のイネーブルビットを示す．1 のとき，RLIRC は出力を行っている．

13.10.16 イベントリンク用 SDRAM ループカウントレジスタ

オフセット: 0xFFFE_0040 属性 リード／ライト

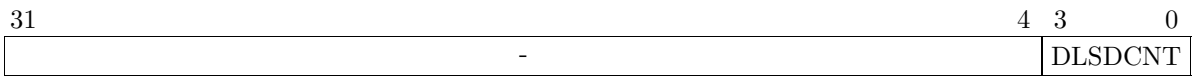


追い越し用 SDRAM に退避されたイベントパケットを *Responsive Link* イベントスイッチに再度送信してよいかどうかを調べる間隔を指定する．短すぎると消費電力が大きくなり，長すぎるとリアルタイム性が損なわれる．

bit 名	機能
ELSDCNT	ELSDCNT (Event Link SDram loop CouNTer) レジスタの設定により，追い越し用 SDRAM に退避されているイベントパケットをイベントスイッチに再送しようとするリトライの間隔を 1 パケット分の送信時間を単位として指定する．(1 - 40) Default: 32

13.10.17 データリンク用 SDRAM ループカウントレジスタ

オフセット: 0xFFFE_0044 属性 リード／ライト

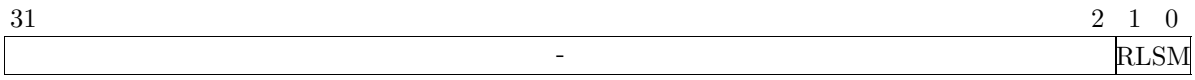


追い越し用 SDRAM に退避されたデータパケットを *Responsive Link* データスイッチに再度送信してよいかどうかを調べる間隔を指定する．短すぎると消費電力が大きくなり，長すぎるとリアルタイム性が損なわれる．

bit 名	機能
DLSDCNT	DLSDCNT (Data Link SDram loop CouNTer) レジスタの設定により，追い越し用 SDRAM に退避されているデータパケットをデータスイッチに再送しようとするリトライの間隔を 1 パケット分の送信時間を単位として指定する．(1 - 95) Default: 4

13.10.18 レスポンシブリンクスイッチモードレジスタ

オフセット: 0xFFFE_0048 属性 リード／ライト



RLSM(*Responsive Link* Switch Mode) レジスタの設定により，レスポンシブリンクのスイッチの動作を変更する．

0: Cut Through Mode レイテンシ的に有利であるがパケットの追い越しをしにくい

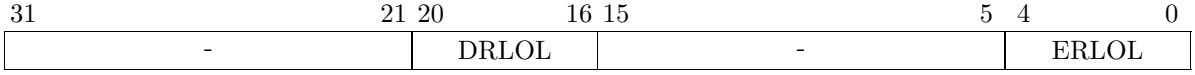
1: Store and Forward Mode レイテンシ的に不利であるがパケットの追越しをしやすい

Default: 0

bit 名	機能
RLSM[0]	Event Link Switch の設定
RSLM[1]	Data Link Switch の設定

13.10.19 レスポンシブリンク用オフラインレジスタ

オフセット: 0xFFFE_004c 属性 リード



Responsive Link は Plug&Play をサポートするために，リンクアップしていたリンクがリンクダウンするとオフライン割り込みを発生し，リンクダウンしていたリンクがリンクアップするとオンライン割り込みを発生する．

RLOL(*Responsive Link* OffLine) レジスタをリードすることにより，どのリンクがオフライン／オンラインかを調べることができる．

1: Offline

0: Online

bit 名	機能
DRLOL	Data link の RLOL レジスタ DRLOL[4]: RLOL[20]: Data link4 DRLOL[3]: RLOL[19]: Data link3 DRLOL[2]: RLOL[18]: Data link2 DRLOL[1]: RLOL[17]: Data link1 DRLOL[0]: RLOL[16]: Data link0(CPU)
ERLOL	Event link の RLOL レジスタ ERLOL[4]: RLOL[4]: Event link4 ERLOL[3]: RLOL[3]: Event link3 ERLOL[2]: RLOL[2]: Event link2 ERLOL[1]: RLOL[1]: Event link1 ERLOL[0]: RLOL[0]: Event link0(CPU)

オフセット: 0xFFFE_004c 属性 ライト

31	2	1	0
-			RLOL

本ビットの設定により，レスポンシブリンクのオフライン割り込み及びオンライン割り込みをクリアできる．

1: 割り込みクリアを行わない
0: 割り込みクリア

bit 名	機能
RLOL[0]	*Responsive Link* Down IRQ Clear: オフライン割り込みのクリア
RLOL[1]	*Responsive Link* Wakeup IRQ Clear: オンライン割り込みのクリア

13.11 DPM (Dual Port Memory)

Responsive Link とプロセッサは基本的に DPM を介してデータの送受信を行う．DPM はその名の通り 2port を有しており，片方はプロセッサバスに接続され，もう片方は *Responsive Link* の Link0 に接続されている．

Data in/out control register および，Event in/out control register を設定することでパケットの送信/受信方法を決定することができる．イベントパケットの送信および，受信には Event packet in/out 専用の DPM を用い，データパケットの送信および，受信には Data packet in/out 専用の DPM を使用する．

以下に Event in/out，Data in/out それぞれの DPM について説明する．

13.11.1 Event Output

図 13.9 にイベントリンク出力用 DPM の構成を示す．Event out control register（図 13.10 参照）に対して，開始アドレス From_Addr (byte address ではなく word address) と終了アドレス To_Addr (word address) を設定することにより，複数パケットを一度に送信できる．From_Addr と To_Addr は人間に分かりやすいようにこのような名前を付けられているが，実際には全く同じ機能のレジスタが二つ用意されている．From_Addr, To_Addr 共に，設定された word address − 1 のアドレスに DPM のプロセッサバス側からデータが書かれた瞬間に，DPM から Link0 に対して出力を開始する．

例えば，Mode0 を使用し，From_Addr を 0x00 に設定し To_Addr を 0x07 (byte address: 0x1c) に設定したとする．プロセッサバス側から DMAC もしくはプロセッサによって Payload0, Payload1 の順に DPM にデータが書かれたとすると，DPM のプロセッサ側から 0x06 番地にデータが書かれた瞬間に DPM から *Responsive Link* の Link0 に出力を開始する．（この場合，実際には From_Addr には意味がない．）

あるいは，Mode0 を使用し，From_Addr を 0x1f(byte address 0x3c) に設定し To_Addr を 0x2f(byte address: 0x7c) に設定し，さらに DMAC を continuous mode で使用すると，Payload0〜3 の領域と Payload4〜7 の領域を使用して，主記憶等に用意した DPM よりも大きな連続データをハードウェアのみで自動送信することができる．（DPM のアドレスデコードの範囲内では，シャドウアドレスでも CS が生成され DPM にアクセスできるように設計しているため．）

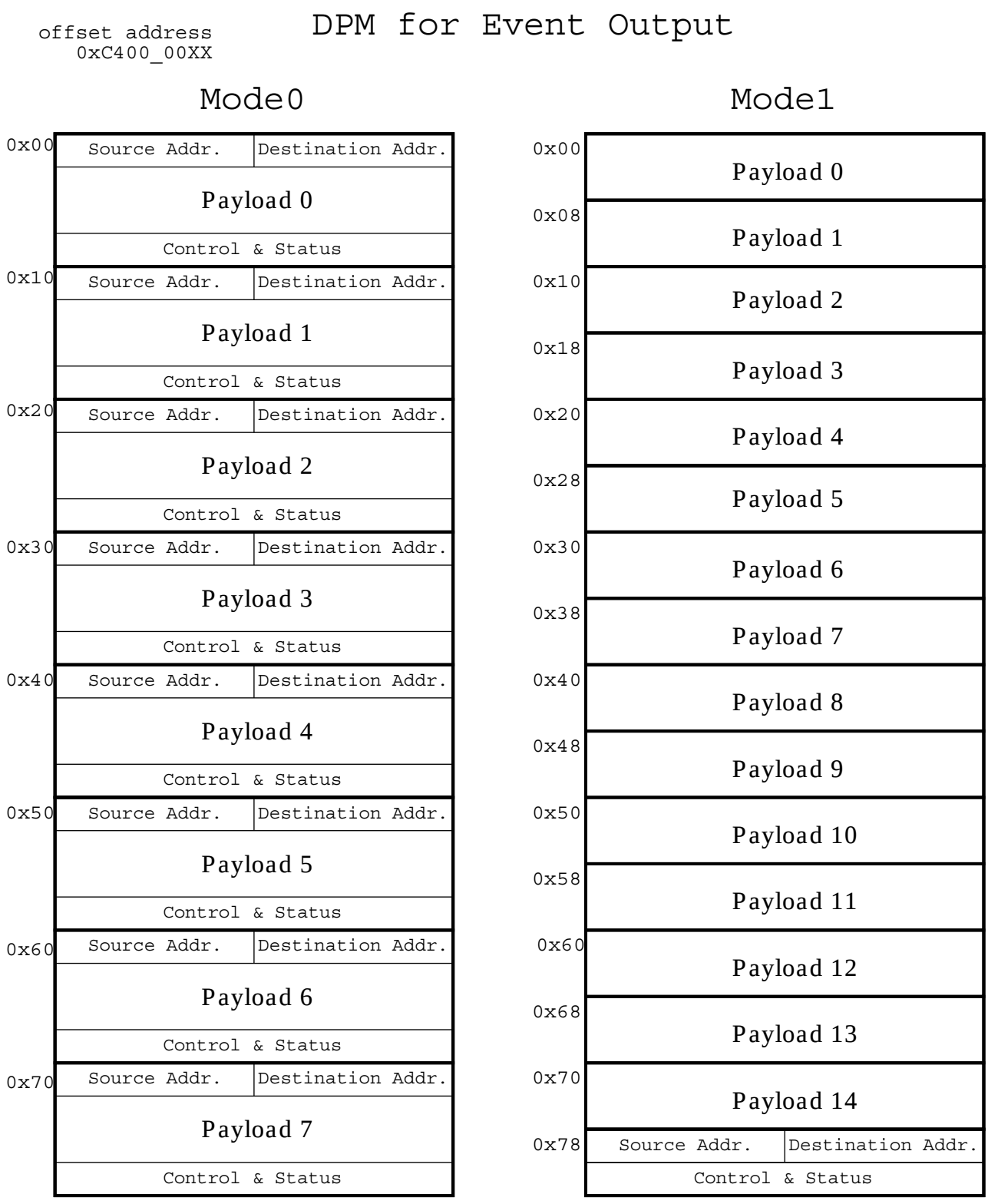


図 13.9: DPM for Event Output

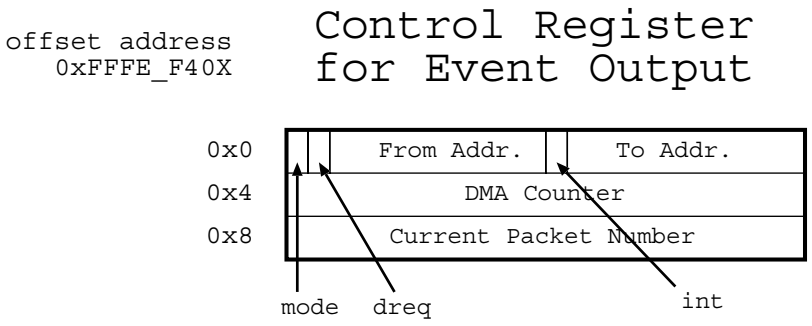


図 13.10: Event Out Control Register

DPM 制御レジスタ

DPM の制御レジスタ（図 13.10 参照）に以下を設定することで，送信の制御を行う．

制御レジスタ (r/w)

- Mode0: mode bit に 0 を設定．すべてのパケットに headr と trailer を付加する．
- Mode1: mode bit に 1 を設定．最後に共通の header と trailer を付加する (すべてのパケットの宛先が同一となる)．
- Int: 本ビットを 1 に設定すると，終了時に EOP(End Of Packet) 割り込みを生成する．
- Dreq: 本ビットを 1 に設定すると，DMA Counter に設定した回数分だけ DMA を行う．
- From_Addr: 設定された word address − 1 のアドレスに DPM のプロセッサバス側からデータが書かれた瞬間に DPM から Link0 に対して出力を開始する．
- To_Addr: 設定された word address − 1 のアドレスに DPM のプロセッサバス側からデータが書かれた瞬間に DPM から Link0 に対して出力を開始する．

DMA Counter (r/w) DMA の回数を指定する

Current Packet Number (r) 現在送信されているパケット番号（図 13.9 の payload 番号に相当）を示す

13.11.2 Event Input

図 13.11 にイベントリンク入力用 DPM の構成を示す．Event in control register（図 13.12 参照）に対して，開始アドレス From_Addr (byte address ではなく word address) と終了アドレス To_Addr (word address) を設定することにより，複数パケットを一度に受信できる．From_Addr と To_Addr は人間に分かりやすいようにこのような名前を付けられているが，実際には全く同じ機能のレジスタが二つ用意されている．From_Addr, To_Addr 共に，設定された word address − 1 のアドレスに DPM の *Responsive Link* 側からデータが書かれた瞬間に，DPM からプロセッサバス側に対して出力 (DMA 転送) を開始する (dreq bit が設定されている場合)．int bit の割り込みを利用して，ソフトウェアで受信することもできる．

例えば，Mode0 を使用し，From_Addr を 0x00 に設定し To_Addr を 0x07 (byte address: 0x1c) に設定したとする．*Responsive Link* 側から Payload0, Payload1 の順に DPM に受信データが書かれていく．*Responsive Link* 側から DPM の 0x06 番地にデータが書かれた瞬間に DPM からプロセッサバス側に出力（DMA 転送）を開始する．(この場合，実際には From_Addr には意味がない．)

あるいは，Mode0 を使用し，From_Addr を 0x1f(byte address 0x3c) に設定し To_Addr を 0x2f(byte address: 0x7c) に設定し，さらに DMAC を continuous mode で使用すると，Payload0〜3 の領域と Payload4〜7 の領域を使用して，主記憶等に用意した DPM よりも大きなメモリ領域（サイクリックバッファ等）に対して，受信データをハードウェアのみで連続的に自動受信することができる.（DPM のアドレスデコードの範囲内では，シャドウアドレスでも CS が生成され DPM にアクセスできるように設計しているため.）

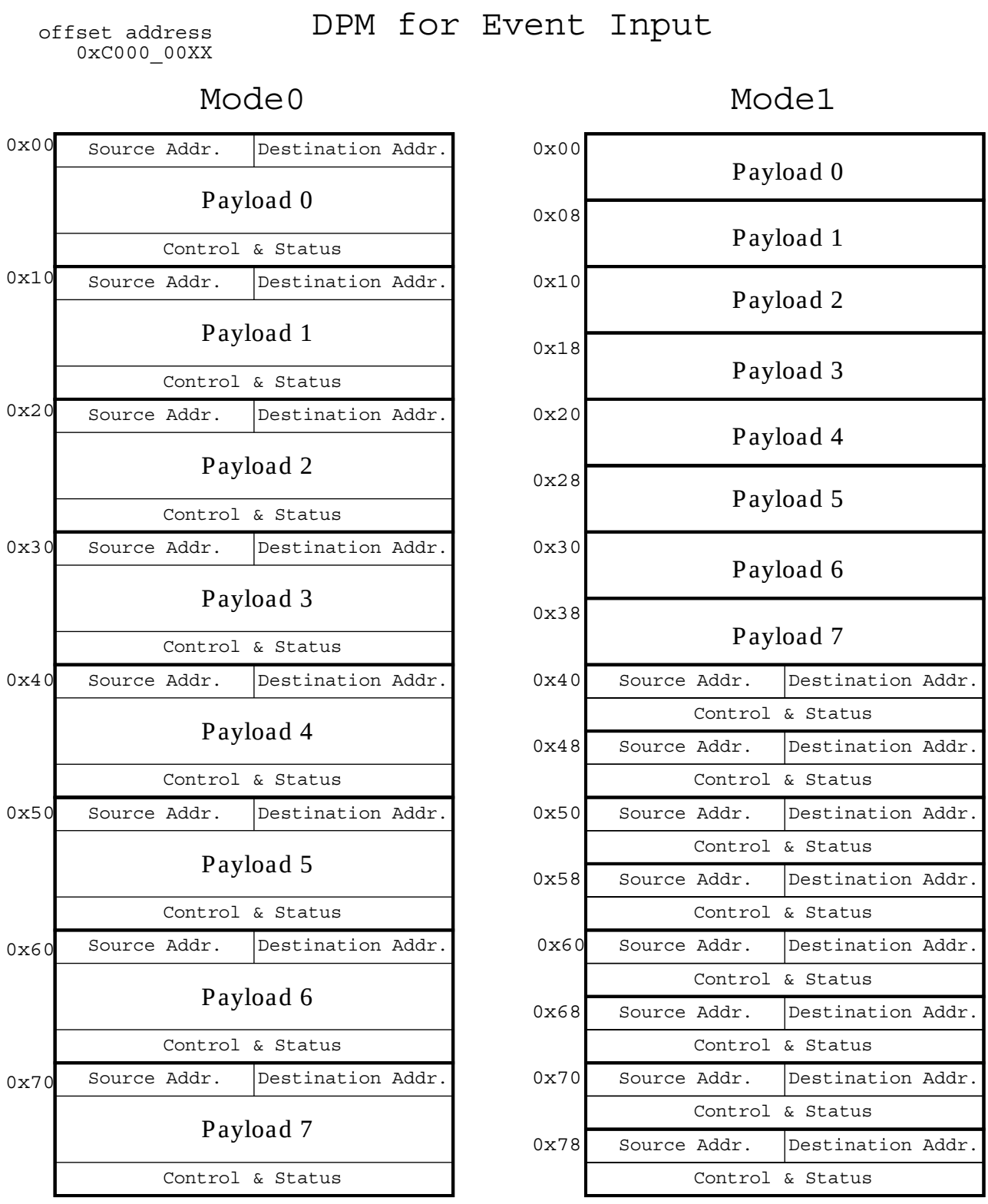


図 13.11: DPM for Event Input

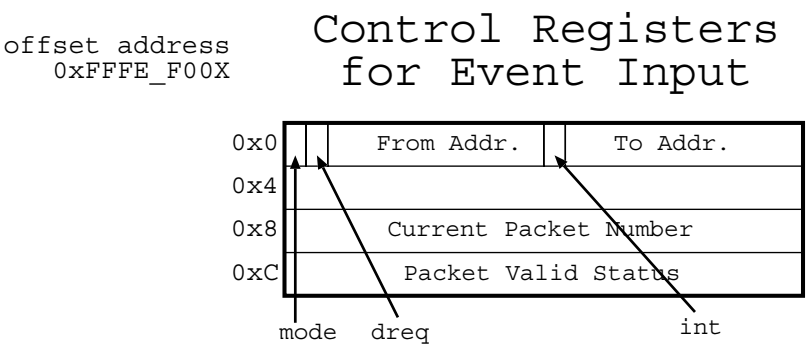


図 13.12: Event in control register

DPM 制御レジスタ

DPM の制御レジスタ（図 13.12 参照）に以下を設定することで，受信の制御を行う．

制御レジスタ (r/w)

- Mode0: mode bit に 0 を設定．すべてのパケットそれぞれに header と trailer が付加された状態で DPM に受信される．
- Mode1: mode bit に 1 を設定．ヘッダとペイロードを図 13.11 のように分離して受信．
- Int: 本ビットを 1 に設定すると，受信終了時にプロセッサに受信完了割り込みを発生する．
- Dreq: 本ビットを 1 に設定すると，From_Addr か To_Addr に設定した word address − 1 にパケットを受信した際に，DMA に対して DREQ を発生する．
- From_Addr: 設定された word address − 1 のアドレスに DPM の *Responsive Link* 側からデータが書かれた瞬間に DPM からプロセッサバス側に対して出力を開始する．
- To_Addr: 設定された word address − 1 のアドレスに DPM の *Responsive Link* 側からデータが書かれた瞬間に DPM からプロセッサバス側に対して出力を開始する．

Current Packet Number (r) 現在送信されているパケット番号（図 13.11 の payload 番号に相当）を示す

Packet Valid Status ハードウェアデバッグ用レジスタ

13.11.3 Data Output

図 13.13 にデータリンク出力用 DPM の構成を示す．Data out control register（図 13.14 参照）に対して，開始アドレス From_Addr (byte address ではなく word address) と終了アドレス To_Addr (word address) を設定することにより，複数パケットを一度に送信できる．From_Addr と To_Addr は人間に分かりやすいようにこのような名前を付けられているが，実際には全く同じ機能のレジスタが二つ用意されている．From_Addr, To_Addr 共に，設定された word address − 1 のアドレスに DPM のプロセッサバス側からデータが書かれた瞬間に，DPM から Link0 に対して出力を開始する．

例えば，Mode0 を使用し，From_Addr を 0x000 に設定し To_Addr を 0x01f (byte address: 0x07c) に設定したとする．プロセッサバス側から DMAC もしくはプロセッサによって Payload0, Payload1 の順に

DPM にデータが書かれたとすると，DPM のプロセッサ側から word address 0x01e 番地にデータが書かれた瞬間に DPM から *Responsive Link* の Link0 に出力を開始する.（この場合，実際には From_Addr には意味がない.）

あるいは，Mode0 を使用し，From_Addr を 0x0ff(byte address 0x3fc) に設定し To_Addr を 0x1ff(byte address: 0x7fc) に設定し，さらに DMAC を continuous mode で使用すると，Payload0〜15 の領域と Payload16〜31 の領域を使用して，主記憶等に用意した DPM よりも大きな連続データをハードウェアのみで自動送信することができる.（DPM のアドレスデコードの範囲内では，シャドウアドレスでも CS が生成され DPM にアクセスできるように設計しているため.）

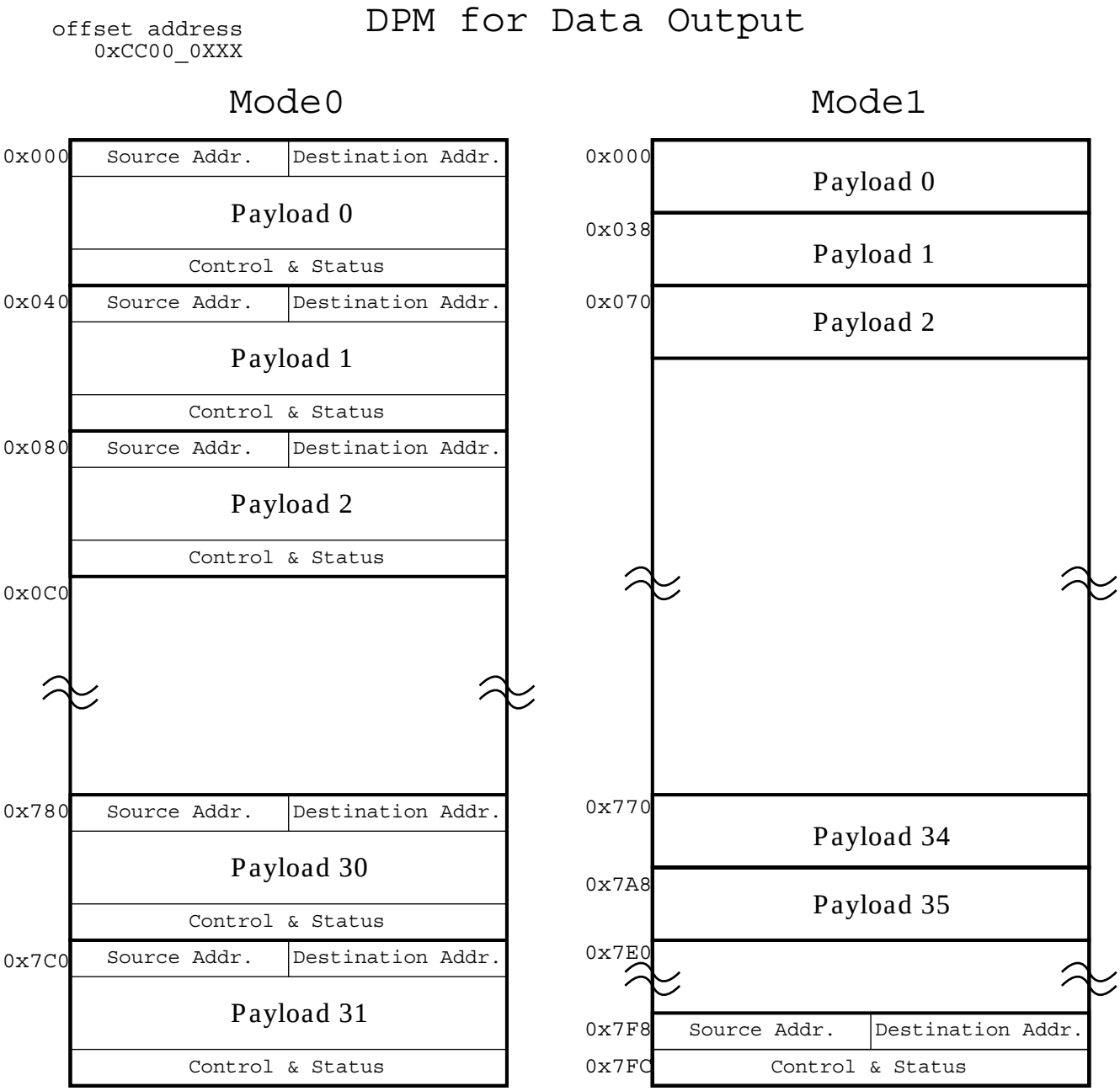


図 13.13: DPM for Data Output

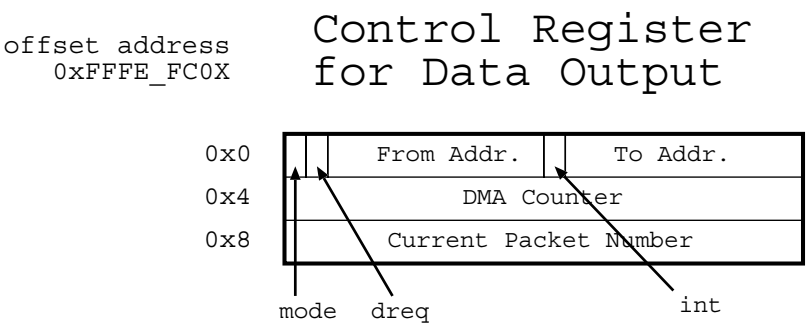


図 13.14: Data Out Control Register

DPM 制御レジスタ

DPM の制御レジスタ（図 13.14 参照）に以下を設定することで，送信の制御を行う．

制御レジスタ (r/w)

- Mode0: (r/w) mode bit に 0 を設定．すべてのパケットに headr と trailer を付加する．
- Mode1: (r/w) mode bit に 1 を設定．最後に共通の header と trailer を付加する (すべてのパケットの宛先が同一となる)．
- Int: (r/w) 本ビットを 1 に設定すると，終了時に EOP(End Of Packet) 割り込みを生成する．
- Dreq: (r/w) 本ビットを 1 に設定すると，DMA Counter に設定した回数分だけ DMA を行う．
- From_Addr: (r/w) 設定された word address − 1 のアドレスに DPM のプロセッサバス側からデータが書かれた瞬間に DPM から Link0 に対して出力を開始する．
- To_Addr: (r/w) 設定された word address − 1 のアドレスに DPM のプロセッサバス側からデータが書かれた瞬間に DPM から Link0 に対して出力を開始する．

DMA Counter (r/w) DMA の回数を指定する

Current Packet Number (r) 現在送信されているパケット番号（図 13.13 の payload 番号に相当）を示す

13.11.4 Data Input

図 13.15 にデータリンク入力用 DPM の構成を示す．Data in control register（図 13.16 参照）に対して，開始アドレス From_Addr (byte address ではなく word address) と終了アドレス To_Addr (word address) を設定することにより，複数パケットを一度に受信できる．From_Addr と To_Addr は人間に分かりやすいようにこのような名前を付けられているが，実際には全く同じ機能のレジスタが二つ用意されている．From_Addr, To_Addr 共に，設定された word address − 1 のアドレスに DPM の *Responsive Link* 側からデータが書かれた瞬間に，DPM からプロセッサバス側に対して出力 (DMA 転送) を開始する (dreq bit が設定されている場合)．int bit の割り込みを利用して，ソフトウェアで受信することもできる．

例えば，Mode0 を使用し，From_Addr を 0x000 に設定し To_Addr を 0x01f (byte address: 0x07c) に設定したとする．*Responsive Link* 側から Payload0, Payload1,... の順に DPM に受信データが書かれていく．

Responsive Link 側から DPM の word address 0x1e 番地にデータが書かれた瞬間に DPM からプロセッサバス側に出力（DMA 転送）を開始する.（この場合，実際には From_Addr には意味がない.）

あるいは，Mode0 を使用し，From_Addr を 0x0ff(byte address 0x3fc) に設定し To_Addr を 0x1ff(byte address: 0x7fc) に設定し，さらに DMAC を continuous mode で使用すると，Payload0〜15 の領域と Payload16〜31 の領域を使用して，主記憶等に用意した DPM よりも大きなメモリ領域（サイクリックバッファ等）に対して，受信データをハードウェアのみで連続的に自動受信することができる.（DPM のアドレスデコードの範囲内では，シャドウアドレスでも CS が生成され DPM にアクセスできるように設計しているため.）

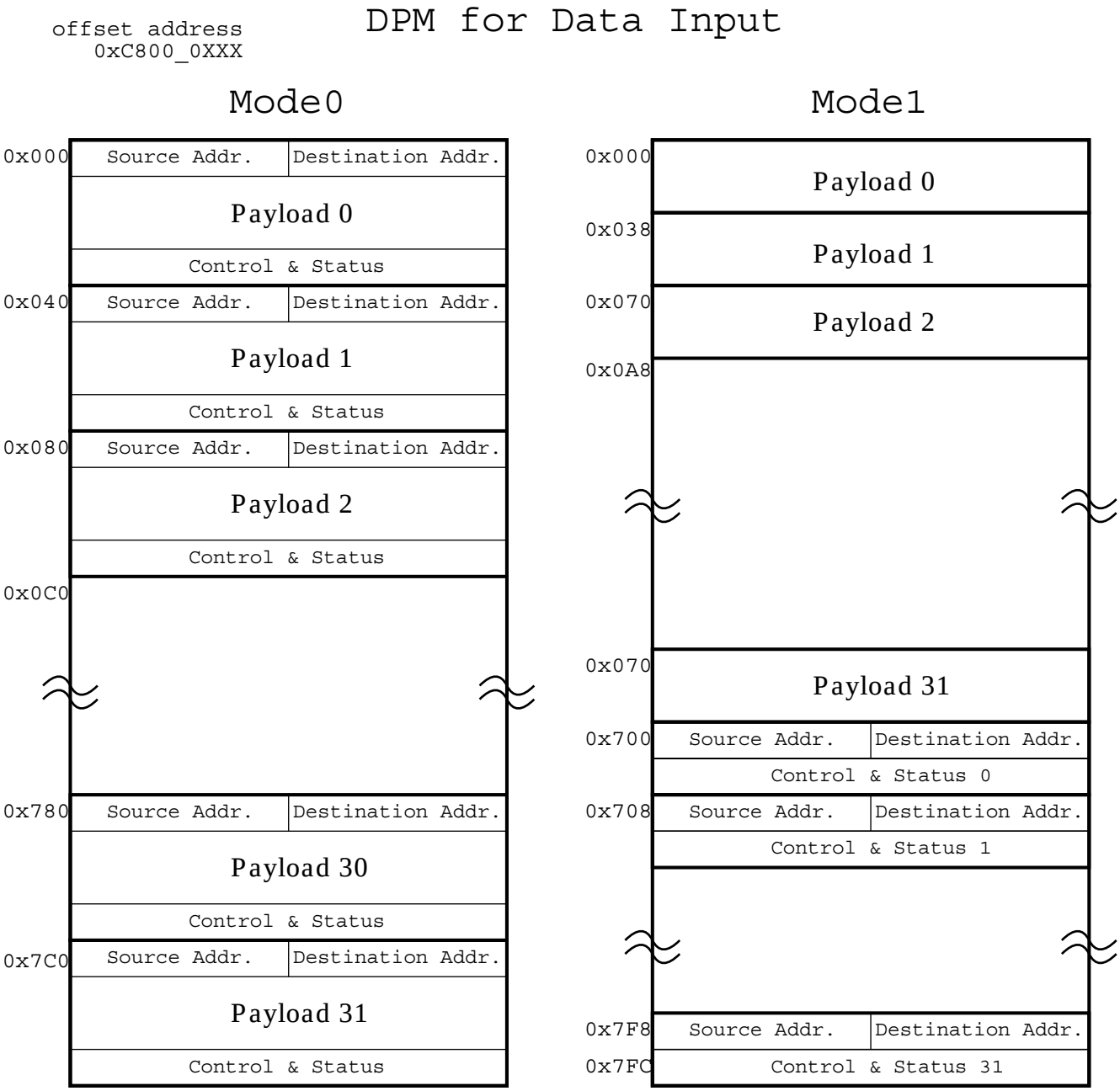


図 13.15: DPM for Data Input

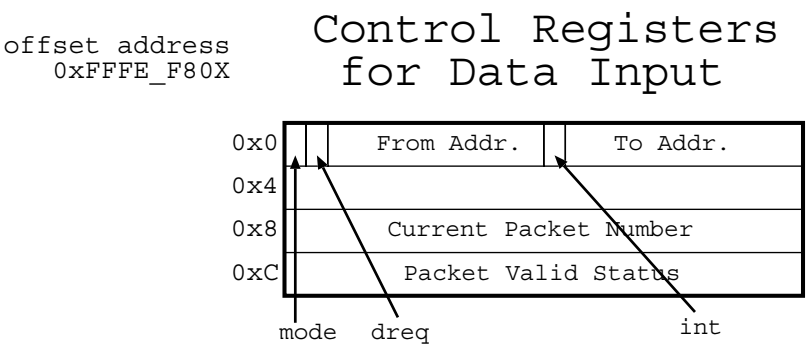


図 13.16: Data In Control Register

DPM 制御レジスタ

DPM の制御レジスタ（図 13.16 参照）に以下を設定することで，受信の制御を行う．

制御レジスタ (r/w)

- Mode0: mode bit に 0 を設定．すべてのパケットそれぞれに header と trailer が付加された状態で DPM に受信される．
- Mode1: mode bit に 1 を設定．ヘッダとペイロードを図 13.15 のように分離して受信．
- Int: 本ビットを 1 に設定すると，受信終了時にプロセッサに受信完了割り込みを発生する．
- Dreq: 本ビットを 1 に設定すると，From_Addr か To_Addr に設定した word address − 1 にパケットを受信した際に，DMA に対して DREQ を発生する．

Current Packet Number (r) 現在送信されているパケット番号（図 13.15 の payload 番号に相当）を示す

Packet Valid Status ハードウェアデバッグ用レジスタ

13.12 通信方法

13.12.1 手順

1. 通信速度の設定—*Responsive Link* 速度設定レジスタ
2. リンクの初期化—*Responsive Link* 初期化レジスタ
3. ルーティングテーブルのバスリクエスト—*Responsive Link* バスリクエストレジスタ
4. ルーティングテーブルの設定
5. ルーティングテーブルのバスリリース—*Responsive Link* バスリクエストレジスタ
6. DPM の設定—Event in/out control レジスタおよび，Data in/out control レジスタ
7. DPM にデータを書き込む → パケット送信

DMA を用いた送信

DPM の容量には当然限界がある．しかし，レスポンシブリンクでは，DMA と DPM が協調して動作することで，DPM の容量を越えるような大きなデータを一度に送信することが可能である．その際の手順は以下の通りである．ただし，総データ量は N packet 分であることを仮定する．

DPM の設定

1. N の約数のうち最大のものを f とする．ただし，データリンクでは fj36，イベントリンクでは fj15 であるとする．
2. DPM の DMA Counter を (N/f)-1 に設定する．
3. DPM の MODE1_HEADER 及び MODE1_TRAILER に宛先およびパケットの持つべき性質 (受信側での割込みなど) を設定する．
4. DPM のコントロールレジスタを mode 1,from(0),to(f*0xe+0xd),DREQ に設定する．(mode 0 で送信する場合，DMA の転送元にはパケットの形でデータが存在している必要がある．ただし，この場合手順 3 は必要無い．)

DMA の設定

1. DMA の送信元を送信したいデータの格納されているメモリの先頭アドレスに設定する．
2. DMA の送信先を送信用 DPM の先頭のアドレスにする．
3. DMA の送信先を送信用 DPM の先頭のアドレスにする．
4. DMA のコントロールレジスタは SAU, RL, MTM, ST を ON にする．(この ST による起動が DMA Counter の設定時に差し引いた 1 回に相当する)

13.12.2 相互通信の際の注意点

相互通信をする際に注意すべきは，二つのボードをつなげてから，まずそれぞれのボードにおいて「通信速度の設定」を行い，さらに，それぞれのボードにおいて「リンクの初期化」を行う．

当然，通信速度は同じでなければならない．また，リンクの初期化はボードをつなげてから行わないと相互通信ができないので注意すること．その他の設定は個別に各ボードで行う．(モニタでの相互通信には，リンクの初期化モジュールを作成し，ボードをつなげたあと，そのモジュールを実行することにより相互通信を行う．)

14

DMAC

- 32/16/8 bit I/F
- 入力チャネル：4
- 優先順位：固定優先度及びラウンドロビン
- Memory to memory 転送機能
- Bus sizing 機能 (8, 16bit I/O 用)
- Bus swapping 機能 (8, 16bit I/O 用)

14.1 レジスタマップ

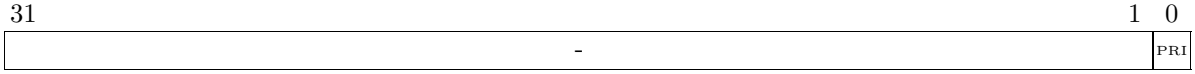
DMAC	初期アドレス
DMAC0	FFFF0000
DMAC1	FFFF1000
DMAC2	FFFF2000

offset	31	24	23	16	15	8	7	0
0x800	-							PRI
0x804	-							IC
0x40*(x)+0x04	PSA<31:0>							
0x40*(x)+0x08	MDA<31:0>							
0x40*(x)+0x18	LN<31:0>							
0x40*(x)+0x0c	ID<31:0>							

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x40*(x)+0x10	-																DAS	SAU	BM	RL	PCI	MTM	MR	32P	16P	8P	S16	S8	IER	IED	ST	
0x40*(x)+0x14	L0	L1	L2	L3	-																										ER	ED

14.1.1 DMA 制御レジスタ

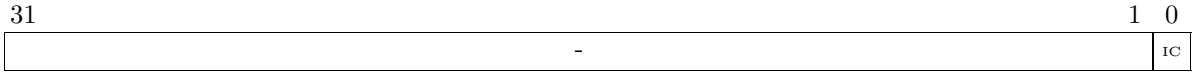
ライト／リード
オフセット: 0x800



bit 名	機能
PRI	PRIority :Default 0 本ビットは DMA チャネルのプライオリティを示す 0:プライオリティはラウンドロビン 1:プライオリティは ch0>ch1>ch2>ch3

14.1.2 DMA 割り込みクリアレジスタ

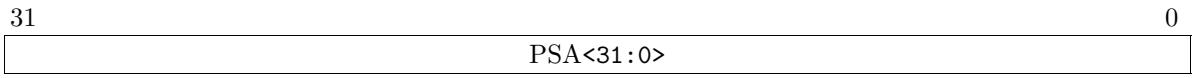
オフセット: 0x804



bit 名	機能
IC	Interrupt Clear 本ビットは DMA 割り込みのクリアを行う． 0:割り込みクリア

14.1.3 ポート／ソースアドレスレジスタ

オフセット: 0x40*(x) +0x04



bit 名	機能
PSA<31:0>	Port/Source Address :Default X チャネル x の DMA に対し，本ビットはメモリから I/O への転送の時（MODE レジスタの MTM ビットが０）ポートアドレスを示し，メモリからメモリへの転送の時（MODE レジスタの MTM ビットが１）ソースアドレスを示す.

14.1.4 メモリ／デスティネーションアドレスレジスタ

オフセット: 0x40*(x) +0x08

31	0
MDA<31:0>	

bit 名	機能
MDA<31:0>	Memory/Destination Address :Default X チャネル x の DMA に対し，本ビットはメモリから I/O への転送の時（MODE レジスタの MTM ビットが０）メモリアドレスを示し，メモリからメモリへの転送の時（MODE レジスタの MTM ビットが１）デスティネーションアドレスを示す.

14.1.5 転送レングスレジスタ

オフセット: 0x40*(x) +0x18

31	0
LN<31:0>	

bit 名	機能
LN<31:0>	transfer LeNgth :Default X チャネル x の DMA に対し，本レジスタは転送レングスを示す．単位はバイトである.

14.1.6 データバッファレジスタ

オフセット: 0x40*(x) +0x0c

31	0
ID<31:0>	

bit 名	機能
ID<31:0>	Internal Data :Default X チャネル x の DMA に対し，DMA 転送時一度内部のデータバッファにてメモリにライトするデータを組み立てるが，そのデータバッファの値が読める．どのロケーションに有効なデータがあるかはステータスレジスタを見る必要がある.

14.1.7 転送モード制御レジスタ

オフセット: 0x40*(x) +0x10

31	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-		DAS	SAU	BM	RL	PCI	MTM	MR	32P	16P	8P	S16	S8	IER	IED	ST

ライト／リード

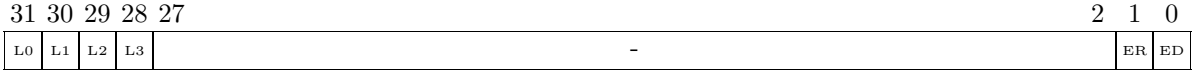
bit 名	機能
DAS	Destination Address Update :Default X 0:メモリアドレスレジスタで設定したアドレスが次の転送にも使用される． 1:メモリアドレスレジスタの値は，最後に転送を行ったアドレスより１ワード先を示す.
SAU	Source Address Update :Default X 0:ポートアドレスレジスタで設定したアドレスが次の転送にも使用される． 1:ポートアドレスレジスタの値は，最後に転送を行ったアドレスより１ワード先を示す.
BM	Burst Mode :Default X 0:バースト転送しない． 1:バースト転送する.
RL	Responsive Link :Default X 1:レスポンシブリンク用 DPM に対する DMA 転送を行う.
PCI	PCI :Default X 1:PCI に対して DMA 転送を行う.
MTM	Memory To Memory transfer :Default X 0:ポートアドレスレジスタで設定した I/O とメモリ間の DMA 転送であることを示す．転送方向は MR ビットにて指定する． 1:ソースアドレスからデスティネーションアドレスへ，レングスレジスタで設定したバイト数のデータを DMA 転送を行う．アドレスカウンタは UP 方向のみのカウントとする．また，4 バイトバウンダリでない転送領域及びレングスの DMA 転送は Memory To Memory ではサポートしない.
MR	MR Memory Read :Default X 0:I/O からメモリへの転送であることを示す． 1:メモリから I/O への転送であることを示す.
32P	32bit I/O Port :Default X 0:don ’t care 1:MTM ビットが 0 の時 32bit の I/O ポートとの転送であることを示す．この時，ポートアドレスのビット 1，0 は無視される.
16P	16P 16bit I/O Port :Default X 0:don ’t care 1:MTM ビットが 0 の時 16bit の I/O ポートとの転送であることを示す．この時，ポートアドレスのビット 0 は無視され，ビット１によりどのデータバスに接続されるか（D31-16 or D15-0）を示す.
8P	8P 8bit I/O Port :Default X 0:don ’t care 1:MTM ビットが 0 の時 8bit の I/O ポートとの転送であることを示す．この時，ポートアドレスのビット 1，0 によりどのデータバスに接続されるか（D31-24 or D23-16 or D15-8 or D7-0）を示す.
S16	Swap at 16bit :Default X 0:don ’t care 1: 16bit 単位でデータのスワップを行う. 31 | A | B | C | D | 0 → 31 | C | D | A | B | 0
S8	Swap at 8bit :Default X 0:don ’t care 1: 8bit 単位でデータのスワップを行う. 31 | A | B | C | D | 0 → 31 | B | A | D | C | 0 S16=1,S8=1 をセットすると以下のようにスワップされる. 31 | A | B | C | D | 0 → 31 | D | C | B | A | 0
IER	Interrupt enable of ER-bit :Default 0 0:割込みを発生しない． 1:割込み発生を許可する.
IED	Interrupt enable of ED-bit :Default 0 0:割込みを発生しない． 1:割込み発生を許可する.
ST	Start :Default 0 0:DMA 転送を停止させる．0 をライト後 DMAC は初期化される．1:DMA 転送を起動する.

本レジスタで設定できるモードは次ページの通りであり，それ以外の設定では動作の保証はしない.

転送モード		スワップなし (S16=0,S8=0)	スワップあり (S16=0,S8=1)	スワップあり (S16=1,S8=0)	リトルエンディアン (S16=1,S8=1)
メモリ (32bit)	メモリ (32bit)	○	×	×	○
メモリ (32bit)	I/O 32bit(D31-0)	○	○	○	○
	I/O 16bit(D31-16)	○	×	×	○
	I/O 16bit(D15-0)	○	×	×	○
	I/O 8bit(D31-24)	○	×	×	○
	I/O 8bit(D23-16)	○	×	×	○
	I/O 8bit(D15-8)	○	×	×	○
	I/O 8bit(D7-0)	○	×	×	○
メモリ (D31-16)	I/O8bit(D31-24)	○	×	×	○

14.1.8 ステータスレジスタ

オフセット: 0x40*(x)+0x14 ライト／リード



bit 名	機能
L0	Location 0 :Default 0 0:内部データレジスタの D31-24 は有効なデータでない． 1:内部データレジスタの D31-24 は有効なデータである．
L1	Location 1 :Default 0 0:内部データレジスタの D23-16 は有効なデータでない． 1:内部データレジスタの D23-16 は有効なデータである．
L2	Location 2 :Default 0 0:内部データレジスタの D15-8 は有効なデータでない． 1:内部データレジスタの D15-8 は有効なデータである．
L3	Location 3 :Default 0 0:内部データレジスタの D7-0 は有効なデータでない． 1:内部データレジスタの D7-0 は有効なデータである．
ER	Error :Default 0 0:don ’t care 1:DMA 転送中にエラーが発生して DMA 転送が停止したことを示す．本ビットは 0 をライトするとクリアされる．
ED	END :Default 0 0:don ’t care 1:DMA 転送が終了すると 1 に設定される．本ビットは 0 をライトするとクリアされる．

15

バスサイジング機能付きDMA

15.1 本DMAの特徴

256 bit ⇔ 32 bit のバスサイジングをしながら転送する。

15.2 制御レジスタ

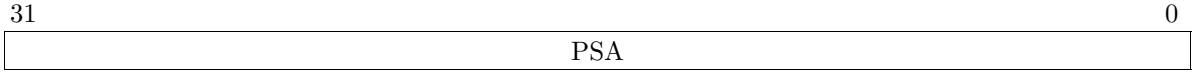
表 15.1: 制御レジスター一覧

アドレス	レジスタ名	用途
0xFFFFD000	PSA	転送元アドレスを指定 (32 bit)
0xFFFFD004	MDA	転送先アドレスを指定 (32 bit)
0xFFFFD008	LENGTH	転送データ数を指定 (byte)
0xFFFFD00C	MODE	転送モード指定、転送開始指定

15.3 制御レジスタ詳細

15.3.1 PSA レジスタ

アドレス: 0xFFFFD000
DMA 転送先のアドレスを 32 bit で指定する。Read/Write 可能。



15.3.2　MDA レジスタ

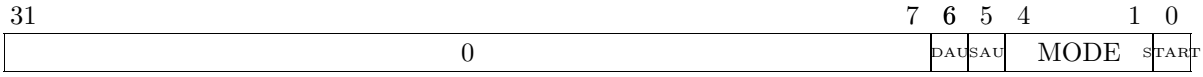
アドレス: 0xFFFFD004
DMA 転送元のアドレスを 32 bit で指定する。Read/Write 可能。

15.3.3　LENGTH レジスタ

アドレス: 0xFFFFD008
DMA 転送するデータ数を byte 単位で指定する。Read/Write 可能。

15.3.4　MODE レジスタ

アドレス: 0xFFFFD00C



<table>
<tr><td>bit 名</td><td>機能</td></tr>
<tr><td>DAU</td><td>DMA 転送開始時に転送先として設定されているアドレス。Write のみ可能。

<table>
<tr><td>設定値</td><td>動作</td></tr>
<tr><td>0</td><td>MDA に設定したアドレス</td></tr>
<tr><td>1</td><td>前回の転送先の続き</td></tr>
</table></td></tr>
<tr><td>SAU</td><td>DMA 転送開始時に転送元として設定されているアドレス。Write のみ可能。

<table>
<tr><td>設定値</td><td>動作</td></tr>
<tr><td>0</td><td>MDA に設定したアドレス</td></tr>
<tr><td>1</td><td>前回の転送元の続き</td></tr>
</table></td></tr>
<tr><td>MODE</td><td>DMA 転送元と転送先の対象を指定する。Read/Write 可能。

<table>
<tr><td>設定値</td><td>転送先</td><td>転送元</td></tr>
<tr><td>0000</td><td>I/O</td><td>I/O</td></tr>
<tr><td>0100</td><td>Memory</td><td>I/O</td></tr>
<tr><td>0001</td><td>I/O</td><td>Memory</td></tr>
<tr><td>0101</td><td>Memory</td><td>Memory</td></tr>
<tr><td>1100</td><td>SDRAM</td><td>I/O</td></tr>
<tr><td>1101</td><td>SDRAM</td><td>Memory</td></tr>
<tr><td>0011</td><td>I/O</td><td>SDRAM</td></tr>
<tr><td>0111</td><td>Memory</td><td>SDRAM</td></tr>
<tr><td>1111</td><td>SDRAM</td><td>SDRAM</td></tr>
</table>

SDRAM : DDR SDRAM I/F (256 bit bus)
Memory : 32 bit bus に接続されたメモリ
I/O : 32 bit bus に接続された I/O
</td></tr>
<tr><td>START</td><td>DMA の転送の開始を指定する。Read/Write 可能。

<table>
<tr><td>設定値</td><td>動作</td></tr>
<tr><td>0</td><td>DMA 転送終了状態</td></tr>
<tr><td>1</td><td>DMA 転送開始/転送中</td></tr>
</table></td></tr>
</table>

15.4 注意事項

- データ転送に指定するアドレスは 8word(32Byte) アラインにそろえる必要がある。

16

パルスカウンタ

16.1 パルスカウンタ概要

- 位相（2 入力）による Up-Down Counter（いわゆるマウスカウンタ）
- bit 幅：32bit
- パルスカウント機能：カウント数がコンペアレジスタにあらかじめ設定されている数になるとパルス（割り込み）を発生
- 上記パルス発生の許可レジスタ及びステータスレジスタ
- 外部からのトリガ入力によってその時点でのカウントレジスタ値を別のカウント保持レジスタにコピーし値を保持
- 外部入力
- チャネル数：9

16.2 レジスタインタフェース

16.2.1 パルスカウンタ制御レジスタ

アドレス	パルスカウンタ制御レジスタ
0xFFFF7000	パルスカウンタ [0]
0xFFFF7020	パルスカウンタ [1]
0xFFFF7040	パルスカウンタ [2]
0xFFFF7060	パルスカウンタ [3]
0xFFFF7080	パルスカウンタ [4]
0xFFFF70a0	パルスカウンタ [5]
0xFFFF70c0	パルスカウンタ [6]
0xFFFF70e0	パルスカウンタ [7]
0xFFFF7100	パルスカウンタ [8]

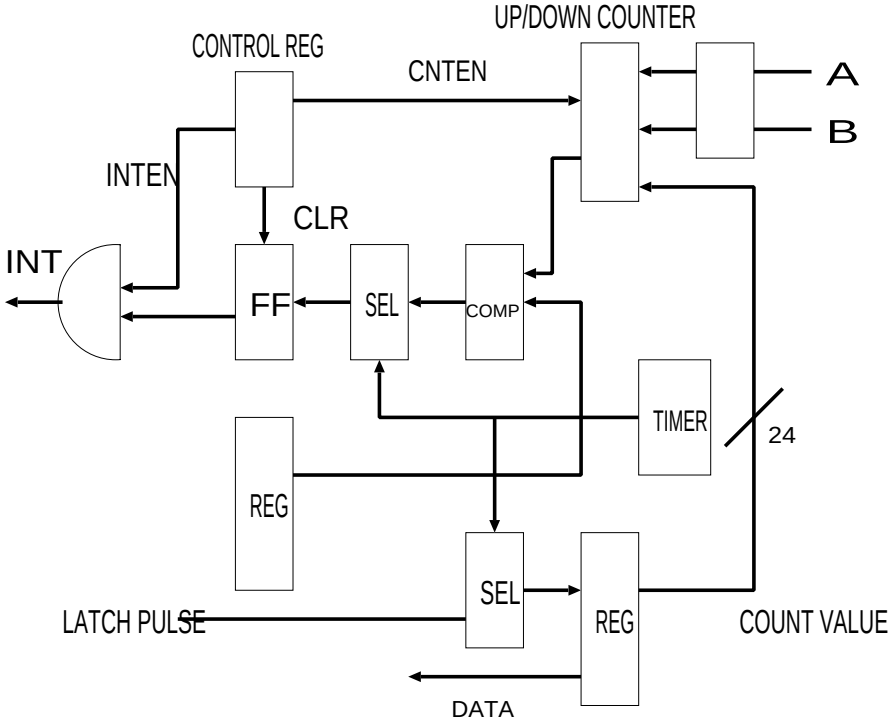


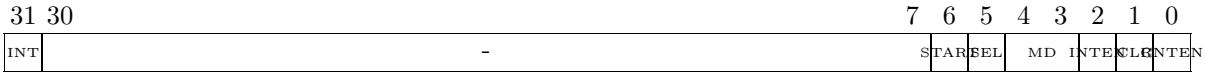
図 16.1: パルスカウンタブロック図

ライト時

31	7	6	5	4	3	2	1	0
-	START	SEL	MD			INTEN	CLR	CNTEN

bit 名	機能
START	START :Default 0 0：内部タイマをリセットして、停止させる。1：内部タイマを起動させる。
SEL	Select :Default 0 カウンタの割り込み動作モードの選択を行う。0：カウンタの値がコンペアデータレジスタに設定された値と同じになった時に割込みを発生させる。1：内部タイマにより設定された値によって、周期的に割込みを発生させる。
MD<4:3>	Mode :Default 0 00：1 逓倍でカウントアップする。01：2 逓倍でカウントアップする。10,11：4 逓倍でカウントアップする。
INTEN	Interrupt Enable :Default 0 0:割り込み禁止 1:割り込み許可
CLR	counter CLear :Default 1 0:カウンタをクリアする 1:don ’t care
CNTEN	Count Enable :Default 0 0:カウンタを停止する 1:カウンタを起動する

リード時

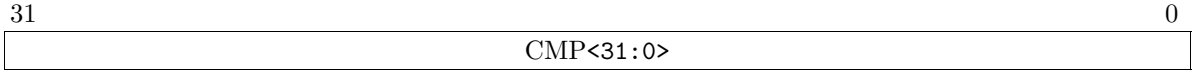


bit 名	機能
INT	Interrupt :Default 0 0:割込み発生なし 1:割込み発生本レジスタのビットは本レジスタをリードするとクリアされる。
START	START :Default 0 0：内部タイマをリセットして、停止させる。1：内部タイマを起動させる。
SEL	Select :Default 0 カウンタの割り込み動作モードの選択を行う。0：カウンタの値がコンペアデータレジスタに設定された値と同じになった時に割込みを発生させる。1：内部タイマにより設定された値によって、周期的に割込みを発生させる。
MD<4:3>	Mode :Default 0 00：1 逓倍でカウントアップする。01：2 逓倍でカウントアップする。10,11：4 逓倍でカウントアップする。
INTEN	Interrupt Enable :Default 0 0:割り込み禁止 1:割り込み許可
CLR	counter CLear :Default 1 0:カウンタをクリアする 1:don ’t care
CNTEN	Count Enable :Default 0 0:カウンタを停止する 1:カウンタを起動する

16.2.2 コンペアデータレジスタ

アドレス	コンペアデータレジスタ
0xFFFF7004	パルスカウンタ [0]
0xFFFF7024	パルスカウンタ [1]
0xFFFF7044	パルスカウンタ [2]
0xFFFF7064	パルスカウンタ [3]
0xFFFF7084	パルスカウンタ [4]
0xFFFF70A4	パルスカウンタ [5]
0xFFFF70C4	パルスカウンタ [6]
0xFFFF70E4	パルスカウンタ [7]
0xFFFF7104	パルスカウンタ [8]

リード／ライト時



bit 名	機能
CMP<31:0>	Compare Data :Default X カウンタ値と比較す比較データを格納する。SEL bit が 0 の場合、カウンタがこの値と等しくなると割込みを発生する。

16.2.3 カウンタレジスタ

アドレス	カウンタレジスタ
0xFFFF7008	パルスカウンタ [0]
0xFFFF7028	パルスカウンタ [1]
0xFFFF7048	パルスカウンタ [2]
0xFFFF7068	パルスカウンタ [3]
0xFFFF7088	パルスカウンタ [4]
0xFFFF70A8	パルスカウンタ [5]
0xFFFF70C8	パルスカウンタ [6]
0xFFFF70E8	パルスカウンタ [7]
0xFFFF7108	パルスカウンタ [8]

リード時

31	0
CNT<31:0>	

bit 名	機能
CNT<31:0>	Count Data :Default X ラッチパルスが入力された時にカウンタの値が本レジスタにラッチされる。

16.2.4 タイマレジスタ

アドレス	タイマレジスタ
0xFFFF700C	パルスカウンタ [0]
0xFFFF702C	パルスカウンタ [1]
0xFFFF704C	パルスカウンタ [2]
0xFFFF706C	パルスカウンタ [3]
0xFFFF708C	パルスカウンタ [4]
0xFFFF70AC	パルスカウンタ [5]
0xFFFF70CC	パルスカウンタ [6]
0xFFFF70EC	パルスカウンタ [7]
0xFFFF710C	パルスカウンタ [8]

リード／ライト時

31	0
TIMER<31:0>	

bit 名	機能
TIMER<31:0>	Timer Data :Default X 周期割り込みに使用するタイマ値を設定する。カウンタクロックをカウントし本タイマ値と等しくなると、SEL bit が 1 の場合、割り込みを発生させる。

17

PWM発生器

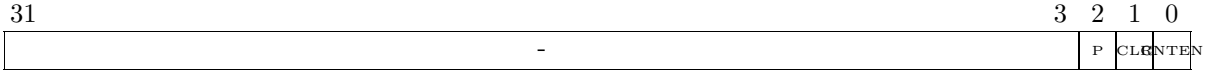
17.1 PWM発生器概要

- Counter 兼 Timer
- Up-Down Counter
- Bit 幅：32bit
- PWM 出力：内部レジスタの設定によってデューティ比の異なる矩形波を出力
- 外部入力可能
- チャネル数：9

17.2 PWMコントロールレジスタ

アドレス	CTRL レジスタ
0xFFFF7200	PWMCTRL[0]
0xFFFF7220	PWMCTRL[1]
0xFFFF7240	PWMCTRL[2]
0xFFFF7260	PWMCTRL[3]
0xFFFF7280	PWMCTRL[4]
0xFFFF72a0	PWMCTRL[5]
0xFFFF72c0	PWMCTRL[6]
0xFFFF72e0	PWMCTRL[7]
0xFFFF7300	PWMCTRL[8]

リード／ライト



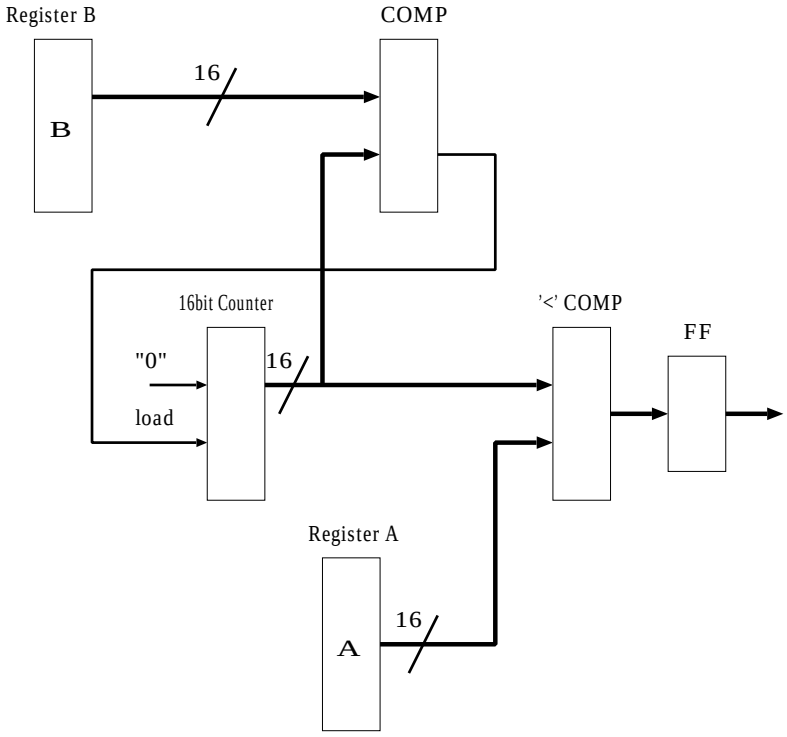


図 17.1: PWM 発生器ブロック図

bit 名	機能
P	Positive :Default 0 0:カウンタ値がタイミングレジスタ値よりも小さい時 PWM 出力は 0 となる 1:カウンタ値がタイミングレジスタ値よりも小さい時 PWM 出力は 1 となる
CLR	Counter clear :Default 0 0:don ’t care 1:カウンタをクリアする
P	Count Enable :Default 0 0:カウンタを停止する 1:カウンタを起動する

17.3 PWM サイクルレジスタ

アドレス	PWM サイクルレジスタ
0xFFFF7204	CY[0]
0xFFFF7224	CY[1]
0xFFFF7244	CY[2]
0xFFFF7264	CY[3]
0xFFFF7284	CY[4]
0xFFFF72a4	CY[5]
0xFFFF72c4	CY[6]
0xFFFF72e4	CY[7]
0xFFFF7304	CY[8]

リード／ライト

31	0
CY<31:0>	

bit 名	機能
CY<31:0>	Cycle :Default X サイクル時間を決定するレジスタである．カウンタ値が本レジスタ値と同じになったらカウンタは値‘ 0 ’をロードする．

17.4 PWM 出力反転制御レジスタ

アドレス	PWM 出力反転制御レジスタ
0xFFFF7208	RV[0]
0xFFFF7228	RV[1]
0xFFFF7248	RV[2]
0xFFFF7268	RV[3]
0xFFFF7288	RV[4]
0xFFFF72a8	RV[5]
0xFFFF72c8	RV[6]
0xFFFF72e8	RV[7]
0xFFFF7308	RV[8]

リード／ライト

31	0
RV<31:0>	

bit 名	機能
RV<31:0>	ReVerse timing :Default X PWM 出力を反転する時間を決定するレジスタである．カウンタ値が本レジスタ値と同じになったら PWM 出力は反転する．

18

DDR SDRAM I/F

- 主記憶
  - 32/128 bit I/F のいずれかを選択可能
- Link SDRAM
  - 32 bit I/F
- 2/2.5/3 の CAS Latency に対応
- tWTR(Internal Write to Read Command Delay) が 1 の DDR チップにのみ対応
- 設定レジスタはワードアクセスのみ有効

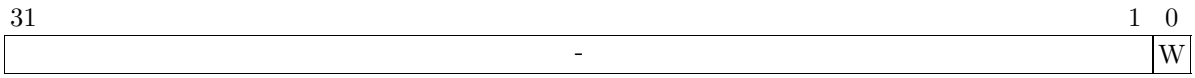
18.1 レジスタマップ

DDR SDRAM I/F	初期アドレス
主記憶 I/F	FFFFF000
Link SDRAM	FFFFE000

offset	31 – 24	23 – 16	15 – 8	7 – 0
0x0	-			State | S
0x4	-	CS	-	RAS | - | CAS
0x8	-			EMRS
0xC	-	MRS2	-	MRS1
0x10	-			
0x14	-	RFC | - | RP	-	RCD | - | MRD
0x18	-	RASmax	-	RASmin
0x1C	-			REFRESH
0x20	-			W

18.1.1 主記憶 I/F 幅設定レジスタ

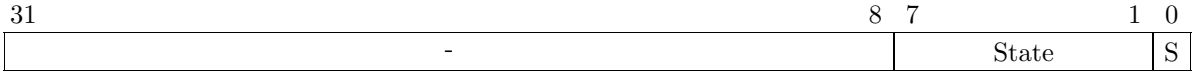
オフセット: 0x20



bit 名	機能
W	Width :Default 0 本ビットで主記憶 I/F のビット幅を設定する。 本レジスタは主記憶 I/F でのみ有効である。 0: 32 bit 1: 128 bit

18.1.2 I/F 起動レジスタ

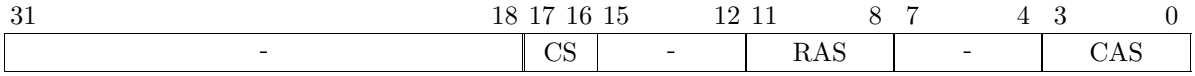
オフセット: 0x0



bit 名	機能
State	State (Read Only) 本ビットは I/F の内部状態を示す。
S	Start :Default 1 本ビットで I/F の起動/停止を設定する。 0: I/F 起動 1: I/F 停止

18.1.3 メモリモジュール設定レジスタ

オフセット: 0x4



bit 名	機能
State	CS: Default 2(主記憶 128 bit I/F) 2(主記憶 32 bit I/F) 1(Link SDRAM I/F) 本ビットは各 I/F の CS 出力信号の接続本数を設定する。
RAS	Row Address Width: Default 12(主記憶 128 bit I/F) 13(主記憶 32 bit I/F) 13(Link SDRAM I/F) 本ビットは各 I/F に接続されている DDR チップの Row Address 幅を設定する。
CAS	Column Address Width: Default 10(主記憶 128 bit I/F) 9(主記憶 32 bit I/F) 9(Link SDRAM I/F) 本ビットは各 I/F に接続されている DDR チップの Column Address 幅を設定する。

18.1.4 EMRS 設定レジスタ

オフセット: 0x8

31	12	11	0
-		EMRS	

bit 名	機能
EMRS	Extended Mode Register Set: Default 0 本レジスタは I/F の起動時に、DDR チップの Extended Mode Reigister Set に書き込む値を設定する。

18.1.5 MRS 設定レジスタ

オフセット: 0xC

31	28	27	16	15	12	11	0
-		MRS2		-		MRS1	

bit 名	機能
MRS2	Mode Register Set 2: Default 0x21 本レジスタは I/F の起動時に、DDR チップの Mode Reigister Set に二度目に書き込む値を設定する。
MRS1	Mode Register Set 1: Default 0x121 本レジスタは I/F の起動時に、DDR チップの Mode Reigister Set に最初に書き込む値を設定する。

18.1.6 DDR 設定レジスタ 1

オフセット: 0x14

31 28	27 24	23 20	19 16	15 12	11 8	7 4	3 0
-	RFC	-	RP	-	RCD	-	MRD

bit 名	機能
RFC	tRFC: Default 9 本レジスタは、I/F に接続されている DDR チップの tRFC 値をサイクル単位で指定する。サイクルの周期は DDR チップに与えているクロックと同じである。
RP	tRP: Default 2 本レジスタは、I/F に接続されている DDR チップの tRP 値をサイクル単位で指定する。サイクルの周期は DDR チップに与えているクロックと同じである。
RCD	tRCD: Default 1 本レジスタは、I/F に接続されている DDR チップの tRCD 値をサイクル単位で指定する。サイクルの周期は DDR チップに与えているクロックと同じである。
MRD	tMRD: Default 1 本レジスタは、I/F に接続されている DDR チップの tMRD 値をサイクル単位で指定する。サイクルの周期は DDR チップに与えているクロックと同じである。

18.1.7 DDR 設定レジスタ 2

オフセット: 0x18

31 30 29	16 15 14 13	0
-	RASmax	-	RASmin

bit 名	機能
RASmax	tRAS max: Default 0x2328 本レジスタは、I/F に接続されている DDR チップの tRASmax 値をサイクル単位で指定する。サイクルの周期は DDR チップに与えているクロックと同じである。
RASmin	tRAS min: Default 6 本レジスタは、I/F に接続されている DDR チップの tRASmin 値をサイクル単位で指定する。サイクルの周期は DDR チップに与えているクロックと同じである。

18.1.8 リフレッシュインターバル設定レジスタ

オフセット: 0x18

31	16	15	0
-		REFRESH	

bit 名	機能
REFRESH	REFRESH: Default 0x48a8 本レジスタは、I/F に接続されている DDR チップのリフレッシュ周期をサイクル単位で指定する。サイクルの周期は DDR チップに与えているクロックと同じである。

19

PCI I/F

初期アドレス: 0xffff3000

19.1 アドレスマップ

19.1.1 Local Bus

PCI I/F

offset	31 — 24	23 — 16	15 — 8	7 — 0
0x00	0x0000		te, me, ee, lte, lts, lme, lms, 0	pe, ps, le, ls, id, 0
0x04	data	4'b0000, dreq, bst	rest	data	4'b0000, dreq, bst	rest
0x08	MailboxA_Higher (PCI → Local)			
0x0c	MailboxA_Lower (PCI → Local)			
0x10	MailboxB (Local → PCI)			
0x14	Reserved			
0x18	Local AD	Mode	Local AD	
0x1c	Reserved			
0x20	Local Bus Acess Port(Local-BAP)			
0x24	Reserved			
0x28	PCI Bus Acess Port (PCI-BAP)			
0x2c	Reserved			
0x30	Current Local AD			
0x34	Reserved			
0x38	Reserved			
0x3c	Reserved			

DMA(Channel0)

offset	31 24	23 16	15 8	7 0
0x40	B/C AR	Mode	Base/Current Address Register(B/C AR)	
0x44	B/C DCR	00	Base/Current Data Count Register(B/C DCR)	
0x48	dsr	00000	dsmr	dmr
0x4c	00000000	dcr	dmcr	damr
0x50	16'h0000		fdcr | 00	fdcr
0x54	rqpr	00 rqpr	wqpr | 00	wqpr
0x58	fsr	mrber	fsmr	8'h00
0x5c	8'h00	fcr	fmcr	famr

DMA(Channel0)

offset	31 24	23 16	15 8	7 0
0xc0	B/C AR	Mode	Base/Current Address Register(B/C AR)	
0xc4	B/C DCR	00	Base/Current Data Count Register(B/C DCR)	
0xc8	dsr	00000	dsmr	dmr
0xcc	00000000	dcr	dmcr	damr
0xd0	16'h0000		fdcr | 00	fdcr
0xd4	rqpr	00 rqpr	wqpr | 00	wqpr
0xd8	fsr	mrber	fsmr	8'h00
0xdc	8'h00	fcr	fmcr	famr

PCI Configration Register

offset	31 24	23 16	15 8	7 0
0x100	VendorID		DeviceID	
0x104	Command		Status	
0x108	RevisionID		ClassCode	
0x10c	CacheLineSize	LatencyTimer	HeaderType	BIST
0x110	Base Address Register			
0x114	Rerserved			
0x118	Rerserved			
0x11c	Rerserved			
0x120	Rerserved			
0x124	Rerserved			
0x12c	SubsystemVendorID		SubsystemID	
0x130	Expansion ROM Base Address			
0x134	Cap_Ptr	Reserved		
0x138	Reserved			
0x13c	InterruptLine	InterruptPin	Min_Gnt	Max_Lat

19.1.2 PCI Bus

PCI I/F

offset	31	24	23	16	15	8	7	0
0x00	pe / ps / le / ls / id / 0 / te / me / ee / lte / lts / lme / lms / 0				0x0000			
0x04	dreq / bst / rest		data		4'b0000	dreq / bst / rest		data / 4'b0000
0x08	MailboxA_Lower (PCI → Local)							
0x0c	MailboxA_Higher (PCI → Local)							
0x10	MailboxB (Local → PCI)							
0x14	Reserved							
0x18	Local AD							Mode
0x1c	Reserved							
0x20	Local Bus Acess Port(Local-BAP)							
0x24	Reserved							
0x28	PCI Bus Acess Port (PCI-BAP)							
0x2c	Reserved							
0x30	Current Local AD							
0x34	Reserved							
0x38	Reserved							
0x3c	Reserved							

DMA(Channel0)

offset	31	24	23	16	15	8	7	0
0x40	Base/Current Address Register(B/C AR)							Mode
0x44	Reserved							
0x48	Base/Current Data Count Register(B/C DCR)							00
0x4c	Reserved							
0x50	dmr		dsmr		00000	cdce / cau / mas32		dsr
0x54	Reserved							
0x58	damr		dmcr		dcr			00000000
0x5c	Reserved							
0x60	fdcr			00	16'h0000			
0x64	Reserved							
0x68	wqpr			00	rqpr			00
0x6c	Reserved							
0x70	8'h00		fsmr		mrber			fsr
0x74	Reserved							
0x78	famr		fmcr		fcr			8'h00
0x7c	Reserved							

DMA(Channel1)

offset	31 24	23 16	15 8	7 0
0xc0	Base/Current Address Register(B/C AR)			Mode
0xc4	Reserved			
0xc8	Base/Current Data Count Register(B/C DCR)			00
0xcc	Reserved			
0xd0	dmr	dsmr	00000	cdce / cau / mas32 / dsr
0xd4	Reserved			
0xd8	damr	dmcr	dcr	00000000
0xdc	Reserved			
0xe0	fdcr	00	16'h0000	
0xe4	Reserved			
0xe8	wqpr	00	rqpr	00
0xec	Reserved			
0xf0	8'h00	fsmr	mrber	fsr
0xf4	Reserved			
0xf8	famr	fmcr	fcr	8'h00
0xfc	Reserved			

19.2 PCI I/F レジスタマップ

PCI BUS 側は little endian．8bit の memory space.

19.2.1 割り込み制御レジスタ

オフセット: 0x0000(Local)

31 16	15	14	13	12	11	10	9	8	7	6	5	4 3	2 1	0
0x0000	te	me	ee	lte	lts	lme	lms	0	pe	ps	le	ls	id	0

オフセット: 0x0000(PCI)

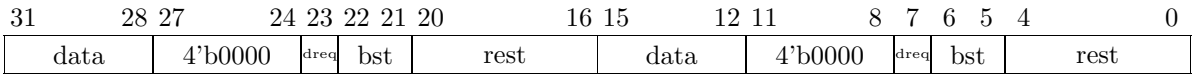
31	30	29	28 27	26 25	24	23	22	21	20	19	18	17	16	15 0
pe	ps	le	ls	id	0	te	me	ee	lte	lts	lme	lms	0	0x0000

bit 名	機能
te	Target abort interrupt Enable (Default:'b1) Local:R, PCI R/W Master 動作時に Target Abort を受信した際の PCI BUS への割り込み制御 1:割り込み許可 0:不許可
me	Master Abort Interrupt Enable (Default:'b0) Local:R, PCI R/W Master 動作時に Master Abort を受信した際の PCI Bus への割り込み制御 1: 割り込み許可 0:不許可
ee	End of Process Interrupt Enable (Default:'b0) Local: R/W, PCI: R DMA レジスタの CDCR が"0"になったときの Local Bus への割り込み制御 1: 割り込み許可 0:不許可
lte	Target Abort Interrupt Enable for Local side (Default:'b0) Local: R/W, PCI: R Master 動作時に Target Abort を受信した際の Local Bus への割り込み制御 1: 割り込み許可 0:不許可
lts	Target Abort Interrupt Status for Local side (Default:'b0) Local: R/W, PCI: R Master 動作の Target Abort 受信用の割り込みレジスタ (Local Bus 側) 受信時に"1"にセットされ，Local Bus 側から"1"を書込むことでリセット
lme	Master Abort Interrupt Enable for Master side (Default:'b0) Local: R/W, PCI: R Master 動作時に Master Abort を受信した際の Local Bus への割り込み制御 1: 割り込み許可 0:不許可
lms	Master Abort Interrupt Status for Master side (Default:'b0) Local: R/W, PCI: R Master 動作の Master Abort 受信用の割り込みレジスタ (Local Bus 側) 受信時に"1"にセットされ，Local Bus 側から"1"を書込むことでリセット

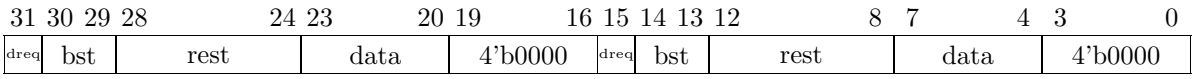
bit 名	機能
pe	PCI Bus Interrupt Enable (Default:'b1) Local: R, PCI: R/W MailboxB への書込み時の PCI Bus への割り込み制御. 1: 割り込み許可 0: 不許可
ps	PCI Bus Interrupt Status (Default:'b0) Local: R, PCI: R/W DoorbellB 用の割り込みレジスタ，割り込み発生時に"1"にセット PCI BUS から"1"を書込むことでリセット
le	Local Bus Interrupt Enable (Default:'b1) Local: R/W, PCI: R MailboxA への書込み時の LocalBus への割り込み制御．High,Low 一括で行なう. 1: 割り込み許可 0: 不許可
ls	Local Bus Interrupt Status (Default:'b00) Local: R/W, PCI: R DoorbellA 用の割り込みレジスタ，割り込み発生時に"1"にセット MailBoxA(Lower) - LIS(Lower),bit3 MailBoxA(Higher) - LIS(Higher), bit4 Local Bus から"1"を書込むことでそれぞれリセット．割り込み解除には双方リセットする必要あり.
id	ID0,ID1 (Default:'b00) Local: R/W, PCI: R LocalBus から任意の値を書込み可，PCI から読み込み可. bit3: ID1, bit2: ID0

19.2.2 プログラム制御レジスタ

オフセット: 0x0004(Local)



オフセット: 0x0004(PCI)



Channel1: 0x0007, 0x0006
Channel0: 0x0005, 0x0004

bit 名	機能
data	FIFO data number for Master Transaction Local: R, PCI: R Master 転送用の FIFO 内のデータ数をバイト単位で出力
dreq	DMA Request Local: R, PCI: R PCI I/F からの dreq_を出力
bst	Burst Number Local: R, PCI: R PCI I/F からの burst_ack_を出力 'b00: 8, 'b01: 4, 'b10: 2, 'b00: single
rest	Remaining Number of Master Transaction Local: R, PCI: R Master 転送時の残り転送数 'b10000 - 8 以上, 'b01000 - 4-7 'b00100 - 2-3, 'b00010 - 1 'b00001 - 0

19.2.3 MailboxA

オフセット: 0x000c(Local), 0x0008(PCI)

31	0
MailboxA_Lower (PCI → Local)	

オフセット: 0x0008(Local), 0x000c(PCI)

31	0
MailboxA_Higher (PCI → Local)	

bit 名	機能
MailboxA	MailBoxA (PCI → Local)(Default: 64'h0) Local: R, PCI: R/W PCI Bus から任意の値を書くことができ，Local Bus から読み出すことが可能 32Bit PCI - 'h0c のみ, 64Bit PCI - 'h08, 'h0c

19.2.4 MailboxB

オフセット: 0x0010(Local), 0x0010(PCI)

31	0
MailboxB (Local → PCI)	

bit 名	機能
MailboxB	MailBoxB (Local → PCI)(Default: 64'h0) Local: R/W, PCI: R Local Bus から任意の値を書くことができ，PCI Bus から読み出すことが可能

19.2.5 Local AD

オフセット: 0x0018(Local)

31	26	25	24	23	0
Local AD		Mode		Local AD	

オフセット: 0x0018(PCI)

31	2	1	0
Local AD			Mode

bit 名	機能
Local AD	Address for Local Bus (Default: 32'h0) Local: R, PCI: R/W Target Transaction 時に Local Bus の address bus に出力するアドレスを指定．転送開始時にこの値を Current Local AD に読み込む．実際に出力する値は [Local AD, 2'b00].
Mode	Address Update Mode in Target Transaction (Default: 2'b0) Local: R, PCI: R/W Local AD の更新モードを指定. 2'b00: リニアインクリメント, その他: 固定

19.2.6 Local Bus Access Port

オフセット: 0x0020(Local), 0x0020(PCI)

31	0
Local Bus Acess Port(Local-BAP)	

bit 名	機能
Local-BAP	Local Bus Access Port Local:不可, PCI: R/W PCI Bus から Local Bus へのアクセスポート．このポートへアクセスすると Local Bus の権利を取得して transaction を行う．

19.2.7 PCI Bus Access Port

オフセット: 0x0028, 0x00a8(Local), 0x0028, 0x00a8(PCI)

31	0
PCI Bus Acess Port (PCI-BAP)	

bit 名	機能
PCI-BAP	PCI Bus Access Port Local: R/W, PCI: 不可 Local Bus から PCI Bus へのアクセスポート．このポートへアクセスすると Master Transaction 用の FIFO との転送を行う． 'h28: Channel0, 'ha8: Channel1

19.2.8 Current Local AD

オフセット: 0x0030(Local), 0x0030(PCI)

31	0
Current Local AD	

bit 名	機能
C-Local AD	Current Local AD Local: R, PCI: R Target Transaction 時に Local Bus 転送に用いられている現在のアドレスを出力．転送開始時に Local AD に変更がある場合は Local AD の値を取り込む．

19.3 Master Transaction 用 DMA レジスタマップ

Channel ごとに独立して用意．bit7 = Channel No.

19.3.1 Address Register

オフセット: 0x0040, 0x00c0(Local)

31 — 26	25 — 24	23 — 0
B/C AR	Mode	Base/Current Address Register(B/C AR)

オフセット: 0x0040, 0x00c0(PCI)

31 — 2	1 — 0
Base/Current Address Register(B/C AR)	Mode

bit 名	機能
B/C AR	Base/Current Address Register (Default: 30'h0) Local: R/W, PCI: R/W PCI Master 動作時に PCI Bus に出力するアドレスの指定．実際に出力されるアドレスは [AR,2'b00]．このポートへの書き込みは BAR と CAR を同時に変更するので，転送中に明示的に BAR を変更することは推奨しない．
Mode	Address Update Mode in Master Transaction (Default: 2'b00) Local: R/W, PCI: R/W Master Transaction 時のアドレス更新モードの指定 2'b00:リニアインクリメント, その他: 固定

19.3.2 Data Count Register

オフセット: 0x0044, 0x00c4(Local)

31 — 26	25 — 24	23 — 0
B/C DCR	00	Base/Current Data Count Register(B/C DCR)

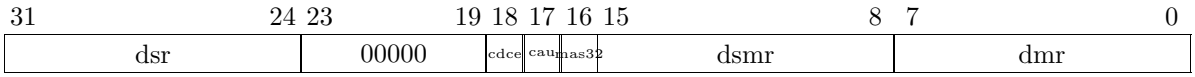
オフセット: 0x0048, 0x00c8(PCI)

31 — 2	1 — 0
Base/Current Data Count Register(B/C DCR)	00

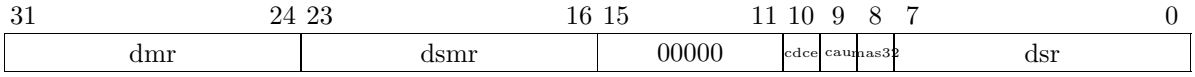
bit 名	機能
B/C DCR	Base/Current Data Count Register (Default: 30'h0) Local R/W, PCI R/W Master Transaction 時の転送バイト数の指定．64bit PCI では 8, 32bit PCI では 4 ずつ減少．Byte Enable に関わらず一定．このポートへの書き込みは BDCR,CDCR 同時に値を変更するので，転送中に値を明示的に変更することは推奨しない．

19.3.3 DMA Control Register

オフセット: 0x0048, 0x00c8(Local)



オフセット: 0x0050, 0x00d0(PCI)



bit 名	機能
dsr	DMA Status Register (Default: 8'h10) Local: R, PCI: R DMA の状態出力ポート. [4]: pci_req_の値を出力．Core への Master 転送要求時に"0". [0]: tc(terminal count) の値を出力．cdcr が 0 のとき"1".
cdce	Circulate Data Count Enable (Default: 1'b0) Local: R/W, PCI: R/W CDCR が 0 に Local Bus 側の DMA が転送を終了した際に自動的に転送を開始するかを指定．開始する際には CDCR には BDCR の値がロードされる. 1'b1:自動開始, 1'b0: 待機
cau	Current Address Update (Default: 1'b0) Local: R/W, PCI: R/W CDCE=1 の場合の自動転送開始時における CAR の設定方法の指定. 1'b1:最後の転送の次のアドレス (アドレス更新モードが固定ならば変更なし) 1'b0:BAR
mas32	Master 32Bit Mode (Default: 1'b0) Local: R/W, PCI: R/W Core-I/F 間の転送モードの指定．PCI-BUS 幅と同じ値の設定を推奨. PCI-32bit, 64Bit Mode の場合は Disconnect や Retry の際にデータが消える恐れあり.
dsmr	DMA Single Mask Register (Default: 8'h04) Local: R/W, PCI: R/W Master Transaction における PCI DMA 転送開始シグナル (pci_req_) のマスク. DSMR,DAMR の両方のマスクを解除する必要がある．bit[2] がマスク.
dmr	DMA Mode Register (Default:8'h00) Local: R/W, PCI R/W Master Transaction 時のコマンドの指定．下位 4bit が app_cmd に出力.

19.3.4 DMA Stop/Reset Register

オフセット: 0x004c, 0x00cc (Local)

31 24	23 16	15 8	7 0
00000000	dcr	dmcr	damr

オフセット: 0x0058, 0x00d8 (PCI)

31 24	23 16	15 8	7 0
damr	dmcr	dcr	00000000

bit 名	機能
dcr	DMA Clear Register Local: W, PCI: W ソフトウェアで DMA を初期化するポート．このポートへ書き込むと DMA(内部レジスタと status) が初期化される．
dmcr	DMA Mask Clear Register Local: W, PCI: W このポートへ書き込むと DMA Mask(DSMR,DAMR) がクリアされる．
damr	DMA All Mask Register (Default:8'h01) Local: R/W, PCI: R/W Channel0,1 双方に効果のあるマスクレジスタ．どちらの Channel のポートへ書き込んでも同じ意味．

19.3.5 FIFO Data Register

オフセット: 0x0050, 0x00d0(Local)

31 16	15 10	9 8	7 0
16'h0000	fdcr	00	fdcr

オフセット: 0x0060, 0x00e0(PCI)

31 18	17 16	15 0
fdcr	00	16'h0000

bit 名	機能
fdcr	FIFO Data Count Register Local: R, PCI: R Master Transaction 用 FIFO のデータ数 (byte count)．

19.3.6 FIFO Request Paremter Register

オフセット: 0x0054,0x00d4(Local)

31 26	25 24	23 16	15 10	9 8	7 0
rqpr	00	rqpr	wqpr	00	wqpr

オフセット: 0x0068,0x00e8(PCI)

31 18	17 16	15 2	1 0
wqpr	00	rqpr	00

bit 名	機能
wqpr	Write Request Parameter Register (Default: 14'h0) Local: R/W, PCI: R/W PCI bus に write request を出す閾値の設定. byte count で設定し，FIFO のデータ数が上回ると要求を出す.
rqpr	Read Request Parameter Register (Default: 14'0) Local: R/W, PCI: R/W PCI bus に read request を出す閾値の設定. byte count で設定し，FIFO の空きデータ数が上回ると要求を出す.

19.3.7 FIFO Control Register

オフセット: 0x0058, 0x00d8(Local)

31 24	23 16	15 8	7 0
fsr	mrber	fsmr	8'h00

オフセット: 0x0070, 0x00f0(PCI)

31 24	23 16	15 8	7 0
8'h00	fsmr	mrber	fsr

bit 名	機能
fsr	FIFO Status Register Local: R, PCI: R FIFO の Status 出力ポート. [7] - Full, [6] - Empty, [5] - R/W_, [4] - MREQ_, [3] - MACK_, [0] - EOP
mrber	Master Read Byte Enable Register (Default: 8'h00) Local: R, PCI: R/W Master Read Transaction 時に出力する Byte Enable の設定.
fsmr	FIFO Single Mask Register (Default: 8'h04) Local: R/W, PCI: R/W 対応する Channel の pci_req_の mask register．bit[2] がマスク．DMA と同様に FSMR と FAMR の両方を解除する必要がある.

19.3.8 FIFO Stop/Reset Register

オフセット: 0x005c, 0x00dc(Local)

31 24	23 16	15 8	7 0
8'h00	fcr	fmcr	famr

オフセット: 0x0078, 0x00f8(Local)

31 24	23 16	15 8	7 0
famr	fmcr	fcr	8'h00

bit 名	機能
fcr	FIFO Clear Register Local: W, PCI: W このポートへの書き込みは FIFO 内のデータをクリアする．内部レジスタはクリアされない.
fmcr	FIFO Mask Clear Register Local: W, PCI: W このポートへの書き込みは対応する Channel の FSMR,FAMR を解除する.
famr	FIFO All Mask Register (Default: 8'h01) Local: R/W, PCI: R/W 両 Channel にマスクをかける．bit[0] がマスク.

19.4 動作/使用方法

19.4.1 Target Transaction (PCI → Local)

PCI Bus 側から Local BUS をアクセスする際には転送前に Local AD を設定する必要があります．Local AD の設定後，Local Bus Access Port にアクセスすると，PCI I/F が Local Bus の権利を要求し，CPU bus のバスマスタとして動作します．Local Bus 側へ出力されるアドレスは Local AD です．各転送ごとにアドレスは更新され，そのモードは Local AD の下位 2bit で決定されます.

データの転送の際には一時的に PCI I/F 内にある FIFO にデータが格納されますが，転送自体は Local Bus の権利を獲得してから行なわれます．Local Bus の権利を獲得するのに時間がかかるので，PCI Bus の仕様にある 16clock rule は無視し，初めのデータ転送に限り 255clock で retry をかけます．次のデータ転送からは 8clock rule を守ります.

- Target Write Transaction
  Local Bus の権利を取り，FIFO に空きがある場合のみ trdy_がアサートされます．FIFO に空きが無くなった場合は trdy_をディアサートし続けます．ただし，8clock たった場合は disconnect します．Local Bus 側への転送は FIFO にデータが格納された時点で開始します.

- Target Read Transaction
  Local Bus の権利を取るとすぐに Local Bus の転送を始めます．この転送は必ず Burst Access になり，複数データを一度に取ってこようとします．FIFO にデータが格納されると trdy_がアサートされます．FIFO にデータがない場合は 8clock たった時点で disconnect されます．常にデータを先に取りに行くので，transaction 終了時に FIFO はリセットされます．

- target initiated termination
  termination が起こる要因は次の 2 つです．stop がアサートされるとともに configuration register の status register がセットされます．

  1. Clock Rule (Time up) 初めのデータについては 255clock(仕様では 16clock)，2 つ目以降では 8clock 経過した場合に disconnet します．この場合は PCI 仕様に従ってただちに転送を再開してください．
  2. Local Bus Error Local Bus 側の転送で Error が起きた場合 (Address が間違っているなど) に target abort が発生します．master への通知は PCI 仕様に従って起こるため，master 側で対処を行なってください．ただし，te をセットし，PCI Bus に割り込みをかけてブリッジ側で対処することも可能です．

19.4.2 Master Transaction(Local → PCI)

CPU Bus から PCI Bus にアクセスする場合は Local Bus の DMA を使用します (しなくても可)．PCI DMA のレジスタの設定の前には pci_req_と dreq_にマスクをかけます．かならず動作停止時にレジスタの変更を行なってください．設定するのは PCI Bus 上での相手アドレス (BAR)，転送データ数 (BDCR), 転送モード (DMR) が必須です．レジスタ設定後に mask を解除して Master 転送を開始します．

- Write
  BDCR が 1 以上で，FIFO に 1 つでも空きがあると dreq_がアサートされます．dreq_がアサートされている状態で，PCI Bus Access Port へ data が書込まれると FIFO に格納されます．

  FIFO のデータ数が wqpr で設定された閾値を上回ると pci_req_がアサートされます．PCI Bus の権利を獲得すると PCI Master となり，転送を始めます．1 回の転送は FIFO が空になるか，CDCR が 0 になるまで行なわれます．FIFO が空になって転送が中断した場合は再度 FIFO のデータ数が，wqpr を越えるか CDCR と一致するまで pci_req_をアサートしません．

- Read
  BDCR が 1 以上で，FIFO の空き数が rqpr で設定された閾値を上回ると pci_req_がアサートされます．PCI Bus の権利を獲得すると PCI Master になり転送を始めます．この転送は FIFO が一杯になるか，CDCR が 0 になるまで行なわれます．FIFO が一杯になって転送が中断した場合は再び空き領域が rqpr の閾値を越すか，CDCR と一致するまで pci_req_をアサートしません．FIFO にデータが 1 つでもあると dreq_がアサートされ，dreq_がアサートされている状態で，PCI Bus Access Port へ read access が来ると，FIFO から data が読み込まれます．

- Master Initiated Termination
  Master 動作時の Termination は通常終了以外は Master Abort のみです．PCI 仕様通りに Configuration Reigster の Master Abort 受信 bit がセットされます．Master での対処を行なう場合は，lme をセットして Local Bus 側へ割り込みがかかるようにしてください．

20

IEEE1394

20.1 概要

- IEEE1394-1995 及び P1394a に準じた送信時のパッキング，受信時のアンパッキング
- サイクルマスターのサポート
- 32-bitCRC によるパリティ生成とエラー検出
- PHY チップとの DC 接続のインターフェイスをサポート
- 100/200/400Mb/sec の 3 スピードのサポート
- Isochronous 転送時，各サイクルにおける転送数の制御
- Isochronous パケットの，送信時はヘッダー自動挿入，受信時はヘッダー自動分離及びルーティング
- バスタイムレジスタのサポート
- CycleSync 出力機能対応
- Isochronous パケットマルチスピードコンカチネーション機能対応
- Asynchronous Stream パケット（ IsochronousCycle 外の Isochronous パケット ）のサポート
- Host I/F
  - ホストインターフェースは，SPARC lite バスライクな同期バス
  - バッファ状態に応じて DMA 転送要求を行い高速にデータ転送を行うこ
- Apri Block
  - Contorl
  - Internal Reg
  - Buffer Manager

- Link Block
  - LinkTx Asynchronous Isochronous の両送信バッファからデータを読出し IEEE1394 で定義される各パケッフォーマットにして PHY インターフェースへパケットを送出する．ノードがルートのときは，サイクルスタートパケットの送出も行う．
  - Link Rx Phy インターフェースからのパケットを受信し，そのパケットが自ノード宛かどうか判断する． Asnchronous パケットであれば， Asynchronous バッファへ書き込む． Isochronous パケットであれば，Isochronous バッファへ書き込む．
  - Cycle Timer アイソクロナスサイクルスタートパケットの送出管理を行う．
  - Cycle Manager アイソクロナスサイクルスタートパケットの監視を行う．
  - ＣＲＣ巡回冗長チェック用コードの生成とチェックを行う．
  - Receive Acknowledge 受信パケットに対するアクノリッジを生成する．
  - Phy Interface Phy チップを直接接続可能なインターフェースを有している．接続対象チップは，100, 200, 400Mbps のいずれのチップも可能である．本コアでは，DC 接続をサポートする．

20.2 レジスター覧

20.2.1 レジスタ内容

Version Register

初期値：0000_0001h

offset	31 24	23 16	15 8	7 0
00h	Version		Revision	

本レジスタは，RTL のバージョン，レビジョンナンバーを示すリードオンリーレジスタである．

Control Register

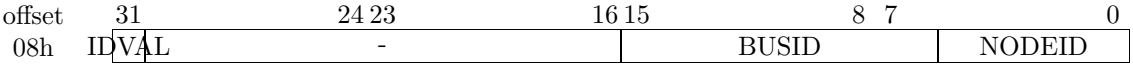
初期値：0000_0000h

offset	31 24	23	16	15	8	7	0
04h	-	SRO	MAS	TIM	-	REN	TEN

本レジスタは，チップの各動作のコンフィグレーション，イネーブル等の設定を行う．通常，電源投入直後にこのレジスタの設定を行い，本コアのコンフィグレーションを決めておく．

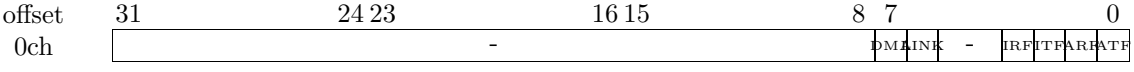
bit 名	機能
TEN<0>	Transmitter Enable ビット（RW - 初期値：0b） • 0 = トランスミッターをディスイネーブルにする • 1 = トランスミッターをイネーブルにする 本レジスタのトランスミッターをイネーブルにするか否かを設定するイネーブル時は以下の送信を行う． • Asyncronous パケット • CycleMaster ビットがイネーブル時でのサイクルスタートパケット • サイクルスタート時での Isochronous パケット
REN<1>	Receiver Enable ビット（RW - 初期値：0b） • 0 = レシーバーをディスイネーブルにする 但し，Self_ID パケットは受信する • 1 = レシーバーをイネーブルにする 本コアのレシーバーをイネーブルにするか否かを設定する．イネーブル時は以下の受信を行う． • 他のノードからこのノードにアドレスされた Asynchronous パケット • 指定したチャンネルの Isochronous パケット
TIMEN<16>	Cycle Timer Enable ビット（RW - 初期値：0b） • 0 = サイクルタイマーをディスイネーブルにする • 1 = サイクルタイマーをイネーブルにする 本コア 内部のサイクルタイマーをイネーブルにするか否かを設定する．
MASTER<17>	Cycle Master ビット（RW - 初期値：0b） • 0 = 他のルートノードからのサイクルスタートパケットを受信し，サイクルタイマーの管理を行う． • 1 = 自ノードがルートであり，かつこのビットが'1' の時，本コアのサイクルタイマーがキャリーするたびに，サイクルスタートパケットを生成する．
SRC<18>	Cycle Sourcer ビット（RW - 初期値：0b） • 0 = PHY チップから供給されるクロックであるマスタークロックの 49.152MHz を 2 分周して 24.576MHz でサイクルタイマーをカウントして Isochronous サイクルを管理する． • 1 = CYCLEIN 端子から入力される信号の立ち上がりでサイクルタイマーを更新して Isochronous サイクルを管理する．Isochronous の時間管理を行っている，内部のサイクルタイマーの更新元を設定する．

Node Identification Register



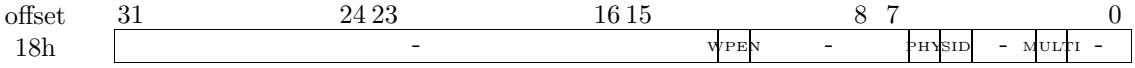
bit 名	機能
NODEID<5:0>	Node Number ビット（RW - 初期値：3Fh）この値は IEEE1212 空間で定義される 6-bit のノードナンバーを設定する．送信時 IEEE 1394 パケットフォーマットのヘッダー内のソース領域にこの値を使用する．また，受信時は受信するパケットのデスティネーションのノードナンバーを見て，この値と一致する場合は受信，そうでない場合はリジェクトする．通常はバスリセット後の Self Identification フェイズの終了後，PHY チップからノードナンバーを読み出し，このレジスタに設定する．
BUSID<15:6>	Bus Number ビット（RW - 初期値：3FFh）この値は IEEE1212 空間で定義される 10-bit のバスナンバーを設定する．送信時 IEEE 1394 パケットフォーマットのヘッダー領域のソース領域にこの値を使用する．また，受信時は受信するパケットのデスティネーションのバスナンバーを見て，この値と一致する場合は受信，そうでない場合はリジェクトする．
IDVAL<31>	ID Valid ビット（RW- 初期値：00b）0 = BusNumber の値が'3FFH' でかつ NodeNumber の値が'3Fh' にアドレスされたパケットのみを受信する．それ以外のパケットはリジェクトする．1 =以下の条件で上記レジスタで設定された，IEEE1212 アドレス空間にアドレスされたパケットのみを受信します．ブロードキャストパケットもまた受信する． • BusNumber と NodeNumber の両方がレジスタ設定値と一致 • BusNumber がレジスタ設定と一致しかつ NodeNumber の値が'3FH' • BusNumber の値が'3FFH' でかつ NodeNumber がレジスタ設定と一致 • BusNumber の値が'3FFH' でかつ NodeNumber の値が'3FH' バスリセット状態が起こるとこのレジスタは自動的に'0' にクリアされる．通常，バスリセット後の Self Identification フェイズの終了後にノードナンバーが決定するため，その値をホストが NodeNumber レジスタに設定後，このビットをセットする．

Reset Register



bit 名	機能
ATF<0>	Reset ATF ビット（RW - 初期値：0b） • 0 = 通常状態 • 1 = Asynchronous 送信用バッファ領域を初期化 Asynchronous 送信用バッファ領域のみを初期状態に戻す．この時，その領域にあるデータはすべて失われる．また，このバッファのステータスフラグもすべて初期状態に戻る．'1' をセットすると，その後内部で初期化動作が完了すると自動的にこのビットは'0' にセットされる．
ARF<1>	Reset ARF ビット（RW - 初期値：0b） • 0 = 通常状態 • 1 = Asynchronous 受信用バッファ領域を初期化 Asynchronous 受信用バッファ領域のみを初期状態に戻す．この時，その領域にあるデータはすべて失われる．また，このバッファのステータスフラグもすべて初期状態に戻る．'1' をセットすると，その後内部で初期状態が完了すると自動的にこのビットは'0' にセットされる．
ITF<2>	Reset ARF ビット（RW - 初期値：0b） • 0 = 通常状態 • 1 = Isochronous 送信用バッファ領域を初期化 Isochronous 送信用バッファ領域のみを初期状態に戻す．この時，その領域にあるデータはすべて失われる．また，このバッファのステータスフラグもすべて初期状態に戻る．'1' をセットすると，その後内部で初期状態が完了すると自動的にこのビットは'0' にセットされる．
IRF<3>	Reset IRF ビット（RW - 初期値：0b） • 0 = 通常状態 • 1 = Isochronous 受信用バッファ領域を初期化 Isochronous Configuration レジスタで指定された Isochronous のチャンネルの受信用バッファ領域のみを初期状態に戻す．この時，その領域にあるデータはすべて失われる．また，このバッファのステータスフラグもすべて初期状態に戻る．'1' をセットすると，その後内部で初期状態が完了すると自動的にこのビットは'0' にセットされる．
LINK<6>	Reset Link Core ビット（RW - 初期値：0b） • 0 = 通常状態 • 1 = Link Core をリセットする． Link Core をリセットする．すべての動作をアボートする．'1' をセットすると，その後内部で初期状態が完了すると自動的にこのビットは'0' にセットされる．
DMA<7>	Reset DMA ビット（RW - 初期値：0b） • 0 = 通常状態 • 1 = DMA 制御をリセットする． DMA 制御をリセットし，DMA 転送可能な状態に設定する．DMA は Quadlet 単位で転送を完了しなければならない．その Quadlet 単位の DMA 転送ポインタをこのビットでクリアし，Quadlet 単位での先頭位置にポインタをセットする．'1' をセットすると，その後内部で初期状態が完了すると自動的にこのビットは'0' にセットされる．

Packet Control Register



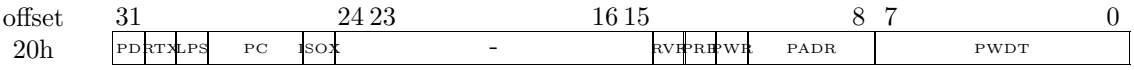
bit 名	機能
MULTI<2>	Multi Speed Concatination On ビット（RW - 初期値：0b） • 0 =Isochronous パケットのマルチスピードコンカチネーション送信を禁止する. • 1 =Isochronous パケットのマルチスピードコンカチネーション送信を許可する. Isochronous パケットのマルチスピードコンカチネーション送信を可能にするか否かを設定する.
SID<5>	Recive Self ID ビット（RW - 初期値：0b） • 0 = SelfID パケットをバッファへ挿入しない. • 1 = SiefID パケットをバッファへ入力する. バスリセット後の Self ID フェイズ中に受信される Self ID パケットを，受信 Async 用のバッファ領域に入れるか入れないかを，このビットで設定する.
PHY<6>	Recive Phy Packet ビット（RW - 初期値：0b） • 0 = Phy Packet をバッファへ挿入しない. • 1 = PhyPacket をバッファへ入力する. 受信した PHY コントロールパケットを，受信 Async 用のバッファ領域に入れるか入れないかを，このビットで設定する．PingPacket を受信した PHY が 4 ポート以上を有する場合，その Self ID Packet は，本デバイスの ARF バッファには，格納されない．また，PHY Control Packet の反転データが異なっていた場合でも，その PHY Controll Packet は，ARF バッファには，格納されない.
WPEN<12>	Write Request Ack-Pending ビット（RW - 初期値：0b） • 0 = Write Requst Packet に対する Ack コードで正常受信の場合 Ack-Complete を返します. • 1 = Write Requst Packet に対する Ack コードで正常受信の場合 Ack-Pending を返します. 通常，Write Request Packet を受信した場合，正常に受信したら，Ack コードは ACK-Complete を返し，バッファーの容量不足等で正常に受信できなかった場合は ACK-Busy を返す．このビットを'1' にセットすることで正常に受信したら，Ack コードは ACK-Pendig を返す．つまり，Write Request の Split Transaction を実行することになる．ホストは Write Requst の処理が完了したら，Write Responce Packet を送信しなければならない.

Diagnostic Status Register

offset	31	24	23	16	15	8	7	0
1ch	-	ITB	ATB	-	RXAS	TXAC	-	TXB -

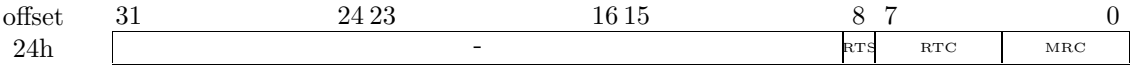
bit 名	機能
TXB<2>	Busy State ビット（R - 初期値：0b）
TXAC<11:8>	AT Ack ビット（R - 初期値：0000b） • 0000 = No Ack • 0001 = ack_complete • 0010 = ack_pending • 0011 = リザーブ • 0100 = ack_busy_X • 0101 = ack_busy_A • 0110 = ack_busy_B • 0111〜1100= リザーブ • 1101 = ack_data_error • 1110 = ack_type_error • 1111 = 予約 送信時，トランスミッターから送信されたパケットに対してのデスティネーションノードから返送されたアクノリッジ（Ack コード）の内容がこのレジスタに反映される．反映されるタイミングは送信したい Asynchronous バッファ内のパケットの処理中を示す，ビジーフラグがネゲートされたときである．次のパケット送信でそのビジーがネゲートされるまで，この値は保持される．
RXAS<13:12>	Ack Status ビット（R - 初期値：00b） • 00 = 正常に受信 • 01 = パリティーエラー • 10 = パケットロスト（規定時間内でのアクノリッジパケットがこなかった） • 11 = 予約 送信した Asynchronous パケットに対してデスティネーションノードから返送されてきたアクノリッジパケットのステータス状態を示す．
ATB<24>	AT Busy ビット（R - 初期値：0b） • 0 = ATGo の発行可能を示す． • 1 = ATGo が発行できない状態を示す．現在直前に発行された ATGo によるパケット処理中を示す． Asynchronous 送信時，ATGo の発行でアサートし，そのアクノリッジの返送を ATAck レジスタに設定したらネゲートする．ホストはこのビットがアサート中は次の ATGo を発行できない．また発行しても無視される．あるパケット送信がリトライ動作になった場合，そのリトライが終了するまで，このビットはネゲートされない．
ITB<25>	IT Busy ビット（R - 初期値：0b） • 0 = ITGo の発行可能を示します． • 1 = ITGo が発行できない状態を示します．現在直前に発行された ITGo によるパケット処理中を示します． Isochronous 送信でかつ IsoMode がノーマル時，ITGo の発行でアサートし，パケットの送信終了でネゲートする．ホストはこのビットがアサート中は次の ITGo を発行できない．また発行しても無視される．

Phy Control Register



bit 名	機能
PWDT<7:0>	Register Data ビット（RW - 初期値：00h）ライト要求で PHY へ転送されるデータを格納する．また，リード要求で PHY から転送されたデータが格納される．このレジスタの内容を読み出す場合，RegData の内容は直前のリード要求にて PHY から読み出された値が読み込まれる．つまり，ホストから書き込んだ値をこのレジスタから直接読み出すことはできない．読み出したい場合は PHY にリード要求して読み出す必要がある．
PADR<11:8>	Register Address ビット（RW - 初期値：00h）ライト要求でアクセスしたい PHY のレジスタのアドレス値を設定する．リード要求で PHY から転送されたレジスタアドレスが格納される．
PWR<12>	Write Register ビット（RW - 初期値：0b） • 0 = 通常状態 • 1 = ライト要求発行 PHY のレジスタへのライト要求を発行します．そのライト要求を行った後このビットをクリアする．
PRR<13>	Read Register ビット（RW - 初期値：0b） • 0 = 通常状態 • 1 = リード要求発行 PHY のレジスタへのリード要求を発行する．そのリード要求を行った後このビットをクリアする．
RVF<14>	Register Data Received ビット（R - 初期値：0b） • 0 = 通常状態 • 1 = リード要求発行後，RegData に Phy からのデータが格納されたことを示す． PHY のレジスタへのリード要求を発行後，RegData に Phy からのデータが格納されると’1’ が設定される．その後に一度このレジスタを読み出すと’0’ にクリアされる．

AT Rretries Register



bit 名	機能
MRC<3:0>	Maxmum Retry Count ビット（RW - 初期値：01h）デスティネーションノードからのビジーのアクノリッジに対して，最大リトライを何回おこなうかを，このレジスタで設定する．このカウント値を参照するリトライフェイズはシングルフェイズの時でである．この設定値内にリトライフェイズが終了しない場合は，本コア が施行するリトライフェイズを終了する．その後，ATF バッファ内のパケットデータはフラッシュされる．設定できる最大値は 15 回である．また”0000” を設定すると本コア は自動的にシングルリトライフェイズを行う．この場合，ビジーアクノリッジに対してパケットデータをフラッシュする．また，リトライフェイズ中にエラーのアックコードが返送されたとき，その時点でリトライを中断しバッファ内をフラッシュしてフラグ（AckErr）を立てて終了する．
RTC<7:4>	Retry Count ビット（R - 初期値：00h）本コアがリトライ中に，その現在のリトライ回数を示しす．
RTS<8>	Retry Stop ビット（RW - 初期値：0b）本コア が自動的にリトライフェイズに入り，そのリミット値にまだ到達せずリトライ中のときにそのリトライの強制終了を行うビットである．’1’ を設定すると，リトライフェイズが完了後，自動でクリアされる． • 0：通常状態 • 1：強制終了

Cycle Timeer Register

offset	31	24 23	16 15	8 7	0
28h	SECOND	CPOUNT		OFFSET	

bit 名	機能
OFFSET<12:0>	Cycle Offset ビット（RW - 初期値：00h）この領域は 24.576MHz のクロックでカウントアップする．Modulo3072 で動作する．
COUNT<24:13>	Cycle Count ビット（RW - 初期値：00h）この領域は CycleField レジスタがキャリーしたときにカウントアップし，Isochronous サイクルをカウントする．Modulo8000 で動作する．
SECOND<31:25>	Cycle Seconds ビット（RW - 初期値：00h）この領域は CycleCount レジスタがキャリーしたときにカウントアップし，秒をカウントする．Modulo128 で動作する．

Isochrounous Configuration Register

offset	31	24 23	16 15	8 7	0
30h	IRTAG	IRCHN	-		

bit 名	機能
IRCHN<29:24>	Channel ビット（RW - 初期値：00h）オートでの送信モードの場合，Isochronous パケットのチャンネルを指定する．ここで設定されたチャンネルナンバーをパケットヘッダー内に挿入して送信する．受信モードでは両モードとも受信したい Isochronous チャンネルを設定する．設定範囲は 0 から 63 である．
IRTAG<31:30>	Tag ビット（RW - 初期値：00h）オートモードでの送信の場合，Isochronous パケットの Tag を指定する．設定範囲は 0 から 3 である．ノーマル及びオートモードでの受信では，受信したい Tag を設定する．以下に IsoMode レジスタの内容とこのコンフィギュレーションレジスタ群との関係を示す．

ATF Data Register

offset	31 24	23 16	15 8	7 0
40h	ATF data			

bit 名	機能
ATF Data<31:0>	ATF Data ビット（W - 初期値：xxxx_xxxxh）Asynchronous パケット送信データ書き込み用レジスタ．内部の Asynchronous 送信用バッファ内にデータが書かれる．

ARF Data Register

offset	31 24	23 16	15 8	7 0
44h	ARF data			

bit 名	機能
ARF Data<31:0>	ARF Data ビット（R - 初期値：xxxx_xxxxh）Asynchronous パケット受信データ読みだし用レジスタ．内部の Asynchronous 受信用バッファからデータが読み出される．

ITF Data Register

offset	31 24	23 16	15 8	7 0
48h	ITF data			

bit 名	機能
ITF Data<31:0>	ITF Data ビット（W - 初期値：xxxx_xxxxh）Isochronous パケット送信データ書き込み用レジスタ．内部の Isochronous 送信用バッファ内にデータが書かれる．

IRF Data Register

offset	31 24	23 16	15 8	7 0
4ch	IRF data			

bit 名	機能
IRF Data<31:0>	IRF Data ビット（R - 初期値：xxxx_xxxxh）Isochronous パケット受信データ読みだし用レジスタ．内部の Isochronous 受信用バッファからデータが読み出される．

Buffer Status and Control Register

offset	31 24	23 16	15	8	7 0
50h	-	FFCNT	-	BSD BDB BMB -	IRF IRE ITF ITE ARF ARE ATF ATE

bit 名	機能
ATE<0>	ATF Empty ビット（R - 初期値：1b） • 0 = バッファが空ではない状態を示す. • 1 = バッファが空の状態を示す. ATF Data レジスタからアクセスする Asynchronous 送信バッファが空であることを示す.
ATF<1>	ATF Full ビット（R - 初期値：0b） • 0 = バッファが一杯ではない状態を示す. • 1 = バッファが一杯の状態を示す. ATF Data レジスタからアクセスする Asynchronous 送信バッファが一杯であることを示す.
ARE<2>	ARF Empty ビット（R - 初期値：1b） • 0 = バッファが空ではない状態を示します. • 1 = バッファが空の状態を示します. ARF Data レジスタからアクセスする Asynchronous 受信バッファが空であることを示す.
ARF<3>	ARF Full ビット（R - 初期値：0b） • 0 = バッファが空ではない状態を示す. • 1 = バッファが空の状態を示す. ARF Data レジスタからアクセスする Asynchronous 受信バッファが一杯であることを示す.
ITE<4>	ITF/IRF Empty ビット（R - 初期値：1b） • 0 = バッファ内が全て空ではないことを示す. • 1 = バッファ内がすべて空であることを示す. ITF Data レジスタからアクセスする Isochronous 送信バッファが空であることを示す.
ITF<5>	ARF Full ビット（R - 初期値：0b） • 0 = バッファが空ではない状態を示す. • 1 = バッファが空の状態を示す. ITF Data レジスタからアクセスする Isochronous 送信バッファが一杯であることを示す.

bit 名	機能
IRE<6>	ARF Empty ビット（R - 初期値：1b） • 0 = バッファが空ではない状態を示す. • 1 = バッファが空の状態を示す. IRF Data レジスタからアクセスする Isochronous 受信バッファが空であることを示す.
IRF<7>	ARF Full ビット（R - 初期値：0b） • 0 = バッファが空ではない状態を示す. • 1 = バッファが空の状態を示す. IRF Data レジスタからアクセスする Isochronous 受信バッファが一杯であることを示す.
BME<11>	Burst Mode Enable ビット（RW - 初期値：0b） • 0 = BMACK 信号が随時ノンアクティブのままである. • 1 = BMREQ 信号に対して BMACK を送出しバースト転送を行う. BMACK 信号を有効にするためのビットである.
BDE<12>	Dreq Enable ビット（RW - 初期値：0b） • 0 = DREQ 信号が随時ノンアクティブのままである. • 1 = DREQ 信号に SelectDreq で選ばれた内容のステータスとして DREQ 信号をアクティブにする. DREQ 信号を有効にするためのビットである.
BSD<14:13>	Select Dreq ビット（RW - 初期値：00b） • 00 = ATF Data レジスタでアクセスするバッファの，ATFFull ビットと同等のステータスとして DREQ 信号に反映される. • 01 = ARF Data レジスタでアクセスするバッファの，ARFEmpty ビットと同等のステータスとして DREQ 信号に反映される. • 10 = ITF Data レジスタでアクセスするバッファの，ITFFull ビットと同等のステータスとして DREQ 信号に反映される. • 11 = IRF Data レジスタでアクセスするバッファの，IRFEmpty ビットと同等のステータスとして DREQ 信号に反映される. チップ内部のどのバッファの状態を DREQ 信号に反映させたいかを選択するレジスタである.
FFCNT<25:16>	Fifo count ビット（R - 初期値：00b）BSC_SELDREQ で選択された受信 Fifo のカウント値を Quadlet 単位で示す.

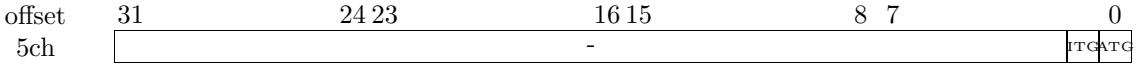
Interrupt Register

bit 名	機能
INTRTYEXP<0>	（RW - 初期値：0b）
INTARFFLU<1>	（RW - 初期値：0b）
INTIRFFLU<2>	（RW - 初期値：0b）
INTARXEND<3>	（RW - 初期値：0b）
INTIRXEND<4>	（RW - 初期値：0b）
INTATXEND<5>	（RW - 初期値：0b）
INTITXEND<6>	（RW - 初期値：0b）
INTPHYINT<7>	（RW - 初期値：0b）
INTBRSTF<8>	（RW - 初期値：0b）
INTBUSRST<9>	（RW - 初期値：0b）
INTPHYRCV<10>	（RW - 初期値：0b）
INTACKERR<11>	（RW - 初期値：0b）
INTTCERR<12>	（RW - 初期値：0b）
INTHDERR<13>	（RW - 初期値：0b）
INTSENTRJ<14>	（RW - 初期値：0b）
INTCYSEC<15>	（RW - 初期値：0b）
INTCYSTART<16>	（RW - 初期値：0b）
INTCYDON<17>	（RW - 初期値：0b）
INTCYLOS<18>	（RW - 初期値：0b）
INTCYARBFL<19>	（RW - 初期値：0b）
INTCMDRST<20>	（RW - 初期値：0b）
PHY_CNA<30>	（RW - 初期値：0b）
PHY_LKON<31>	（RW - 初期値：0b）

Interrupt Mask Register

Interrupt レジスタ内の各割込発生要因を INT# 信号に反映させたくない場合，このレジスタでマスクする．このレジスタの並びは Interrupt レジスタと同様である．各ビットとも'1' の設定でマスクされる．なお，Interrupt レジスタで未定義の部分に当たる Mask ビットは必ずマスク（ "1 " を設定）すること．

TGo Register



bit 名	機能
ATGo<0>	AT Go ビット（RW - 初期値：0b）Asynchronous パケットの送信開始を，このレジスタへ'1' を書くことにより本コア へ知らせる.
ITGo<1>	IT Go ビット（RW - 初期値：0b）Isochronous パケットの送信開始を，このレジスタへ'1' を書くことにより本コア へ知らせる.

21

Universal Asynchronous Receiver/Transmitter

初期アドレス: Channel0:0xffff6000 Channel1:0xffff6080

21.1 アドレスマップ

offset	31 24 23 16 15 8	7 0
0x0000		RB
0x0000		THR
0x0000		DL1
0x0004		IER
0x0004		DL2
0x0008		IIR
0x0008		FCR
0x000c		LCR
0x00010		MCR
0x0014		LSR
0x0018		MSR

21.1.1 Receiver Buffer (RB) / Transmitter Holding Register (THR)

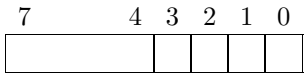
オフセット: 0x0000

7	0

bit 名	機能
7-0	送信 FIFO の入力および受信 FIFO の出力.

21.1.2 Interrupt Enable Register (IER)

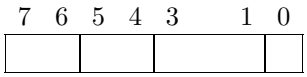
オフセット: 0x0004



bit 名	機能
0	Received Data availble interrupt. ‘ 0 ’ - Disabled. ‘ 1 ’ - Enabled.
1	Transmitter Holding Register empty interrupt. ‘ 0 ’ - Disabled. ‘ 1 ’ - Enabled.
2	Receiver Line Status Interrupt. ‘ 0 ’ - Disabled. ‘ 1 ’ - Enabled.
3	Modem Status Interrupt. ‘ 0 ’ - Disabled. ‘ 1 ’ - Enabled.
7-4	Reserved. Should be logic ‘ 0 ’.

21.1.3 Interrupt Identification Register (IIR)

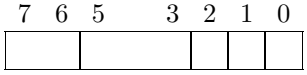
オフセット: 0x0008



<table>
<tr><th>bit 名</th><th>機能</th></tr>
<tr><td>0</td><td>When this is ‘ 0 ’, an interrupt is pending. When this is ‘ 1 ’, no interrupt is pending.</td></tr>
<tr><td>3-1</td><td>The following table displays the list of possible interrupts along with the bits they enable, priority, and their source and reset control.
<table>
<tr><th></th><th>Prio-rity</th><th>Interrupt Type</th><th>Interrupt Source</th><th>Interrupt Reset Control</th></tr>
<tr><td>011</td><td>1th</td><td>Receiver Line Status</td><td>Parity, Overrun or Framing errors or Break Interrupt</td><td>Reading the Line Status Register</td></tr>
<tr><td>010</td><td>2nd</td><td>Receiver Data available</td><td>FIFO trigger level reached</td><td>FIFO drops below trigger level</td></tr>
<tr><td>110</td><td>2nd</td><td>Timeout Indication</td><td>There’s at least 1 character in the FIFO but no character has been input to the FIFO or read from it for the last 4 char times.</td><td>Reading from the FIFO (Receiver Buffer Register)</td></tr>
<tr><td>001</td><td>3rd</td><td>Transmitter Holding Register empty</td><td>Transmitter Holding Register Empty</td><td>Writing to the Transmitter Holding Register or reading the IIR</td></tr>
<tr><td>000</td><td>4th</td><td>Modem Status</td><td>CTS, DSR, RI or DCD</td><td>Reading the Modem Status Register</td></tr>
</table></td></tr>
<tr><td>5-4</td><td>Reserved. Should be logic ‘ 0 ’.</td></tr>
<tr><td>7-6</td><td>Reserved. Should be logic ‘ 1 ’ for compatibility reason.</td></tr>
</table>

21.1.4 FIFO Control Register (FCR)

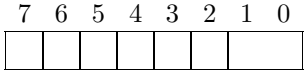
オフセット: 0x0008



bit 名	機能
0	Ignored(Used to enable FIFOs in NS16550D). Since this UART only supports FIFO mode, this bit is ignored.
1	Writing a ‘ 1 ’ to bit 1 clears the Receiver FIFO and resets its logic. But it doesn ’ t clear the shift register, i.e. receiving of the current character continues.
2	Writing a ‘ 1 ’ to bit 2 clears the Transmitter FIFO and resets its logic. The shift register is not clreared, i.e. transmitting of the current character continues.
5-3	Ignored.
7-6	7-6 Define the Receiver FIFO Interrupt trigger level. ‘ 00 ’ - 1 bytes ‘ 01 ’ - 4 bytes ‘ 10 ’ - 8 bytes ‘ 11 ’ - 16 bytes

21.1.5 Line Control Register (LCR)

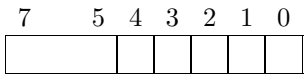
オフセット: 0x000c



bit 名	機能
1-0	Select number of bits in each character. ‘ 00 ’ - 5 bits ‘ 01 ’ - 6 bits ‘ 10 ’ - 7 bits ‘ 11 ’ - 8 bits
2	Specify the number of generated stop bits. ‘ 0 ’ - 1 stop bit. ‘ 0 ’ - 1.5 stop bits when 5-bit character length selected and 2 bits otherwise. Note that the receiver always checks the first stop bit only.
3	Parity Enable. ‘ 0 ’ - No parity ‘ 1 ’ - Parity bit is generated on each outgoing character and is checked on each incoming one.
4	Even Parity select. ‘ 0 ’- Odd number of ‘ 1 ’ is transmitted and checked in each word (data and parity combined). In other words, if the data has an even number of ‘ 1 ’ in it, then the parity bit is ‘ 1 ’. ‘ 1 ’ - Even number of ‘ 1 ’ is transmitted in each word.
5	Stick Parity bit. ‘ 0 ’ - Stick Parity disabled. ‘ 1 ’ - If bits 3 and 4 are logic ‘ 1 ’, the parity bit is transmitted and checked as logic ‘ 0 ’. If bit 3 is ‘ 1 ’ and bit 4 is ‘ 0 ’ then the parity bit is transmitted and checked as ‘ 1 ’.
6	Break Control bit. ‘ 1 ’ - The srial out is forced into logic ‘ 0 ’ (break state). ‘ 0 ’ - Break is disabled.
7	Divisor Latch Access bit. ‘ 1 ’ - The divisor latches can be accessed. ‘ 0 ’ - The normal registers are accessed.

21.1.6 Modem Control Register (MCR)

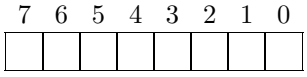
オフセット: 0x0010



bit 名	機能
0	Data Terminal Ready (DTR) signal control. ‘ 0 ’ - DTR is ‘ 1 ’ ‘ 1 ’ - DTR is ‘ 0 ’
1	Request To Send (RTS) signal control ‘ 0 ’ - RTS is ‘ 1 ’ ‘ 1 ’ - RTS is ‘ 0 ’
2	Out1. In loopback mode, connected Ring Indicator (RI) signal input.
3	Out2. In loopback mode, connected to Data Carrier Detect (DCD) input.
4	Loopback mode. ‘ 0 ’ - normal operation. ‘ 1 ’ - loopback mode. When in loopback mode, the Serial Output Signal (STX_PAD_O) is set to logic ‘ 1‘ . The signal of the transmitter shift register is internally connected to the input of the receiver shift register. The following connections are made: DTR → DSR RTS → CTS Out1 → RI Out2 → DCD
7-5	Ignored.

21.1.7 Line Status Register (LSR)

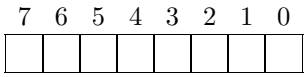
オフセット: 0x0014



bit 名	機能
0	Data Ready (DR) indicator. ‘ 0 ’ - No characters in the FIFO. ‘ 1 ’ - At least one character has been received and is in the FIFO.
1	Overrun Error (OE) INDICATOR. ‘ 1 ’ - If the FIFO is full and another character has been received in the receiver shift register. If another character is starting to arrive, it will overwrite the data in the shift register but the FIFO will remain intact. The bit is cleared upon reading from the register. Generates Receiver Line Status interrupt. ‘ 0 ’ - No overrun state.
2	Parity Error (PE) indicator. ‘ 1 ’ - The character that is currently at the top of the FIFO has been received with parity error. The bit is cleared upon reading from the register. Generate Receiver Line Status interrupt. ‘ 0 ’ - No parity error in the current character.
3	Framing Error (FE) indicator. ‘ 1 ’ - The received character at the top of the FIFO did not have a valid stop bit. The UART core tries re-synchronizing by assuming that the bit received was a start bit. Of course, generally, it might be that all the following data is corrupt. The bit is cleared upon reading from the register. Generates Receiver Line Status interrupt. ‘ 0 ’ - No framing error in the current character.
4	Break Interrupt (BI) indicator. ‘ 1 ’- A break condition has been reached in the current character. The break occurs when the line is held in logic 0 for a time of one character (start bit + data + parity + stop bit). In that case, one zero character enters the FIFO and the UART waits for a valid start bit to receive next character. The bit is cleared upon reading from the register. Generates Receiver Line Status interrupt. ‘ 0 ’ - No break condition in the current character.
5	Transmit FIFO is empty. ‘ 1 ’ - The transmitter FIFO is empty. Generates Transmitter Holding Register Empty interrupt. The bit is cleared in the following cases: The LSR has been read, the IIR has been read or data has been written to the transmitter FIFO. ‘ 0 ’ - Otherwise.
6	Transmitter Empty indicator. ‘ 1 ’ - Both the transmitter FIFO and transmitter shift register are empty. The bit is cleared upon reading from the register or upon writing data to the transmit FIFO. ‘ 0 ’ - Otherwise.
7	‘ 1 ’- At least one parity error, framing error or break indications have been received and are inside the FIFO. The bit is cleared upon reading from the register. ‘ 0 ’ - Otherwise.

21.1.8 Modem Status Register (MSR)

オフセット: 0x0018

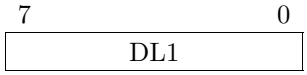


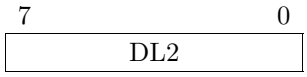
bit 名	機能
0	Delta Clear To Send (DCTS) indicator. ‘ 1 ’ - The CTS line has changed its state.
1	Delta Data Set Ready (DDSR) indicator. ‘ 1 ’ - The DSR line has changed its state.
2	Trailing Edge of Ring Indicator (TERI) detector. The RI line has changed its state from low to high state.
3	Delta Data Carrier Detect (DDCD) indicator. ‘ 1 ’ - The DCD line has changed its state.
4	Complement of the CTS input or equals to RTS in loopback mode.
5	Complement of the DSR input or equals to DTR in loopback mode.
6	Complement of the RI input or equals to Out1 in loopback mode.
7	Complement of the DCD input or equals to Out2 in loopback mode.

21.1.9 Divisor Latches (DL)

オフセット: 0x0000(DL1), 0x0004(DL2)

The divisor latches can be accessed by setting the 7th bit of LCR to ‘ 1 ’. You should restore this bit to ‘ 0 ’ after setting the divisor latches in order to restore access to the other registers that occupy the same addresses.





bit 名	機能
DL1, DL2	The 2 bytes form one 16-bit register, which is internally accessed as a single number. You should therefore set all 2 bytes of the register to ensure normal operation. The register is set to the default value of 0 on reset, which disables all serial I/O operations in order to ensure explicit setup of the register in the software. The value set should be equal to (system clock speed) / (16 times desired baud rate). The internal counter starts to work when the LSB of DL is written, so when setting the divisor, write the MSB first and the LSB last.

21.2 動作/使用方法

This UART core is very similar in operation to the standard 16550 UART chip with the main exception being that only the FIFO mode is supported. The scratch register is removed, as it serves no purpose.

21.2.1 Initialization

Upon reset the core performs the following tasks:

- The receiver and transmitter FIFOs are cleared.
- The receiver and transmitter shift registers are cleared.
- The Divisor Latch register is set to 0.
- The Line Control Register is set to communication of 8 bits of data, no parity, 1 stop bit.
- All interrupts are disabled in the Interrupt Enable Register.

For proper operation, perform the following:

- Set the Line Control Register to the desired line control parameters. Set bit 7 to ‘ 1 ’ to allow access to the Divisor Latches.
- Set the Divisor Latches, MSB first, LSB next.
- Set bit 7 of LCR to 0 to disable access to Divisor Latches. At this time the transmission engine starts working and data can be sent and received.
- Set the FIFO trigger level. Generally, higher trigger level values produce less interrupt to the system, so setting it to 14 bytes is recommended if the system responds fast enough.
- Enable desired interrupts by setting appropriate bits in the Interrupt Enable register.

Remember that (Input Clock Speed)/(Divisor Latch value) = 16 × the communication baud rate. Since the protocol is asynchronous and the sampling of the bits is performed in the perceived middle of the bit time, it is highly immune to small differences in the clocks of the sending and receiving sides, yet no such assumption should be made calculating the Divisor Latch values.

22

更新履歴

2006 年 3 月 30 日

PCI インターフェースの Mailbox のアドレスマップ (ローカル側オフセット) を修正。

2006 年 3 月 30 日

更新履歴の追加

2006 年 6 月 13 日

ベクトル演算器の仕様を追加。

2006 年 7 月 11 日

ベクトル命令のニーモニックを修正。

2006 年 7 月 13 日

ベクトルシフト命令のオペランドの位置を修正。

2006 年 7 月 19 日

ベクトル制御レジスタの説明を追加。

2006 年 8 月 25 日

概要の I/O 部分を修正。全体のブロック図を修正。