Space *Responsive Multithreaded Processor* Version 1 Specification

The 6th Edition.

November 5, 2021

Yamasaki Lab.,

Department of Information and Computer Science,
Keio University

— NY0030 —

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

Contents

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 (UART232C, 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．

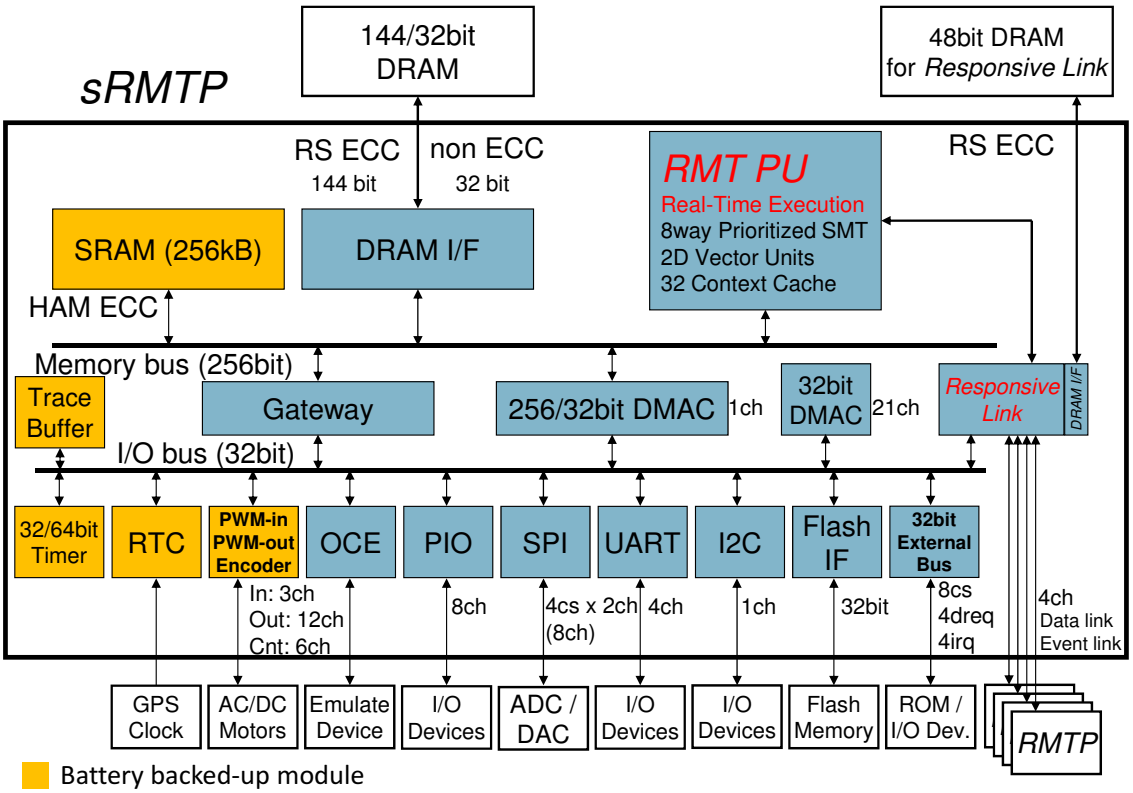


Figure 1.1: *RMT Processor* のブロック図

以下に *RMT Processor* が持つ I/O を示す．

- *Responsive Link* (4ch)
- Trace Buffer

- 32/64bit Timer
- Real Time Clock Unit
- PWM Input (3ch)
- PWM Output (12ch)
- Pulse Counter (6ch)
- On-Chip Emulator (1ch)
- Parallel I/O Unit (8ch)
- Serial Peripheral Interface (4cs × 2ch)
- UART (4ch)
- I2C (1ch)
- NOR Flash I/F (1ch)
- External Bus I/F (8cs, 4dreq, 4irq)
- 256/32bit DMA Controller (1ch)
- 32bit DMA Controller (4ch × 5 + 1)
- Hamming ECC SRAM (256kB)
- 144bit ReedSolomon DDR SDRAM I/F (1ch)
- 48bit ReedSolomon DDR SDRAM I/F for RL(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．

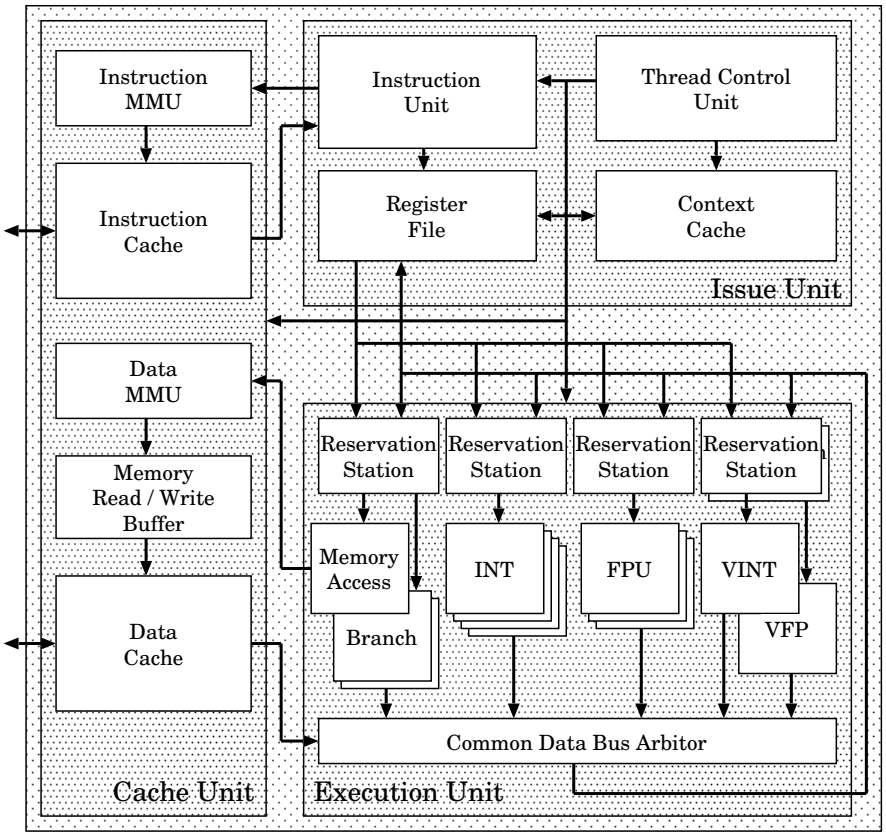


Figure 1.2: *RMT PU* のブロック図

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

Figure 1.3: スレッドテーブルのフォーマット

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．

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

Table 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 (GPR) 数	32bit × 32entry × 8set (1set/thread)
整数リネームレジスタ数	32bit × 64entry
浮動小数点レジスタ (FPR) 数	64bit × 8entry × 8set (1set/thred)
浮動小数点リネームレジスタ数	64bit × 64entry

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，表 1.3 には各演算ユニットごとにコンフリクトが発生しない場合のレイテンシ (命令がリザベーションステーションからディスパッチされ、データが使用可能となるまでのクロックサイクル) を示す．表 1.3 中の vlen はプログラマの指定したベクトル長を意味し、表 13.2 で示されるベクトルユニット内の制御レジスタにより設定を行うことができる。また、ストア命令に関しては書き込み先がレジスタではなくメモリであり、ロード命令以外からのアクセスはできないため、レイテンシは省略した。

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

Table 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 × 2unit）
浮動小数点ベクトル演算器	1（4FPU × 2unit）
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

Table 1.3: 命令演算ユニットのレイテンシ

整数演算器	1 cycle
整数演算器 (Divide)	3 cycle
浮動小数点演算器	2 cycle
浮動小数点演算器 (Divide)	13 cycle
64bit 整数演算器	2 cycle
整数ベクトル演算器	⌈ vlen/8 ⌉
整数ベクトル演算器 (Divide)	6 + vlen
浮動小数点ベクトル演算器	⌈ vlen/4 ⌉
浮動小数点ベクトル演算器 (Divide)	3 + vlen × 13
メモリアクセスユニット (Load, Cache hit)	8 cycle
メモリアクセスユニット (Load, SRAM)	22 cycle
メモリアクセスユニット (Load, SDRAM)	35 cycle
メモリアクセスユニット (Store)	-

とができる．

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 1.4 にキャッシュユニットの概要を示す．

Table 1.4: 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

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 assignments

PIN assignments of SRMTP is shown below.

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
CORNER	LEFT	CORNER_TL	pnl_iocrnr	NGPIO CORNER
1	LEFT	hiz_pad_LINK_PAD_HIZ_GEN_3__link_hiz_		NGPIO
	input	link_hiz_		
2	LEFT	hiz_pad_LINK_PAD_HIZ_GEN_4__link_hiz_		NGPIO
	input	link_hiz_		
3	LEFT	vdd_vop_ngpio_l7	pnl_vop	NGPIO
4	LEFT	ext_iopad0_data31		NGPIO
	inout	bus_data		
5	LEFT	vss_gcs_ngpio_l30	pnl_gcs	NGPIO
6	LEFT	ext_iopad0_data30		NGPIO
	inout	bus_data		
7	LEFT	ext_iopad0_data29		NGPIO
	inout	bus_data		
8	LEFT	vdd_vc_ngpio_l14	pnl_vc	NGPIO
9	LEFT	ext_iopad0_data28		NGPIO
	inout	bus_data		
10	LEFT	vss_go_ngpio_l8	pnl_go	NGPIO

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
11	LEFT	ext_iopad0_data27		NGPIO
	inout	bus_data		
12	LEFT	vss_gcs_ngpio_l29	pnl_gcs	NGPIO
13	LEFT	ext_iopad0_data26		NGPIO
	inout	bus_data		
14	LEFT	ext_iopad0_data25		NGPIO
	inout	bus_data		
15	LEFT	vss_gcs_ngpio_l28	pnl_gcs	NGPIO
16	LEFT	ext_iopad0_data24		NGPIO
	inout	bus_data		
17	LEFT	vdd_vc_ngpio_l13	pnl_vc	NGPIO
18	LEFT	ext_iopad0_data23		NGPIO
	inout	bus_data		
19	LEFT	vss_go_ngpio_l7	pnl_go	NGPIO
20	LEFT	ext_iopad0_data22		NGPIO
	inout	bus_data		
21	LEFT	vss_gcs_ngpio_l27	pnl_gcs	NGPIO
22	LEFT	ext_iopad0_data21		NGPIO
	inout	bus_data		
23	LEFT	ext_iopad0_data20		NGPIO
	inout	bus_data		
24	LEFT	vdd_vop_ngpio_l6	pnl_vop	NGPIO
25	LEFT	ext_iopad0_data19		NGPIO
	inout	bus_data		
26	LEFT	vss_gcs_ngpio_l26	pnl_gcs	NGPIO
27	LEFT	ext_iopad0_data18		NGPIO
	inout	bus_data		
28	LEFT	ext_iopad0_data17		NGPIO
	inout	bus_data		
29	LEFT	vdd_vc_ngpio_l12	pnl_vc	NGPIO
30	LEFT	ext_iopad0_data16		NGPIO
	inout	bus_data		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
31	LEFT	vss_gcs_ngpio_l25	pnl_gcs	NGPIO
32	LEFT	ext_iopad0_data15		NGPIO
	inout	bus_data		
33	LEFT	ext_iopad0_data14		NGPIO
	inout	bus_data		
34	LEFT	ext_iopad0_data13		NGPIO
	inout	bus_data		
35	LEFT	vdd_vc_ngpio_l11	pnl_vc	NGPIO
36	LEFT	ext_iopad0_data12		NGPIO
	inout	bus_data		
37	LEFT	ext_iopad0_data11		NGPIO
	inout	bus_data		
38	LEFT	ext_iopad0_data10		NGPIO
	inout	bus_data		
39	LEFT	vss_gcs_ngpio_l24	pnl_gcs	NGPIO
40	LEFT	ext_iopad0_data9		NGPIO
	inout	bus_data		
41	LEFT	vss_go_ngpio_l6	pnl_go	NGPIO
42	LEFT	ext_iopad0_data8		NGPIO
	inout	bus_data		
43	LEFT	ext_iopad0_data7		NGPIO
	inout	bus_data		
44	LEFT	vss_gcs_ngpio_l23	pnl_gcs	NGPIO
45	LEFT	ext_iopad0_data6		NGPIO
	inout	bus_data		
46	LEFT	vdd_vop_ngpio_l5	pnl_vop	NGPIO
47	LEFT	ext_iopad0_data5		NGPIO
	inout	bus_data		
48	LEFT	ext_iopad0_data4		NGPIO
	inout	bus_data		
49	LEFT	ext_iopad0_data3		NGPIO
	inout	bus_data		
50	LEFT	ext_iopad0_data2		NGPIO
	inout	bus_data		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
51	LEFT	ext_iopad0_data1		NGPIO
	inout	bus_data		
52	LEFT	vss_gcs_ngpio_l22	pnl_gcs	NGPIO
53	LEFT	ext_iopad0_data0		NGPIO
	inout	bus_data		
54	LEFT	vdd_vc_ngpio_l10	pnl_vc	NGPIO
55	LEFT	hiz_pad_ebiu_hiz_		NGPIO
	input	ebiu_hiz_		
56	LEFT	vss_gcs_ngpio_l21	pnl_gcs	NGPIO
57	LEFT	ext_iopad0_data_dir		NGPIO
	output	bus_dir		
58	LEFT	vdd_vc_ngpio_l9	pnl_vc	NGPIO
59	LEFT	ext_iopad0_ie		NGPIO
	output	bus_ie_		
60	LEFT	ext_iopad0_oe		NGPIO
	output	bus_oe_		
61	LEFT	ext_iopad0_addr31		NGPIO
	inout	bus_addr		
62	LEFT	vss_gcs_ngpio_l20	pnl_gcs	NGPIO
63	LEFT	ext_iopad0_addr30		NGPIO
	inout	bus_addr		
64	LEFT	vss_go_ngpio_l5	pnl_go	NGPIO
65	LEFT	ext_iopad0_addr29		NGPIO
	inout	bus_addr		
66	LEFT	vss_gcs_ngpio_l19	pnl_gcs	NGPIO
67	LEFT	ext_iopad0_addr28		NGPIO
	inout	bus_addr		
68	LEFT	vdd_vop_ngpio_l4	pnl_vop	NGPIO
69	LEFT	ext_iopad0_addr27		NGPIO
	inout	bus_addr		
70	LEFT	vss_gcs_ngpio_l18	pnl_gcs	NGPIO

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
71	LEFT	ext_iopad0_addr26		NGPIO
	inout	bus_addr		
72	LEFT	ext_iopad0_addr25		NGPIO
	inout	bus_addr		
73	LEFT	ext_iopad0_addr24		NGPIO
	inout	bus_addr		
74	LEFT	vdd_vc_ngpio_l8	pnl_vc	NGPIO
75	LEFT	ext_iopad0_addr23		NGPIO
	inout	bus_addr		
76	LEFT	vss_gcs_ngpio_l17	pnl_gcs	NGPIO
77	LEFT	ext_iopad0_addr22		NGPIO
	inout	bus_addr		
78	LEFT	vdd_vc_ngpio_l7	pnl_vc	NGPIO
79	LEFT	ext_iopad0_addr21		NGPIO
	inout	bus_addr		
80	LEFT	vss_gcs_ngpio_l16	pnl_gcs	NGPIO
81	LEFT	ext_iopad0_addr20		NGPIO
	inout	bus_addr		
82	LEFT	ext_iopad0_addr19		NGPIO
	inout	bus_addr		
83	LEFT	ext_iopad0_addr18		NGPIO
	inout	bus_addr		
84	LEFT	ext_iopad0_addr17		NGPIO
	inout	bus_addr		
85	LEFT	ext_iopad0_addr16		NGPIO
	inout	bus_addr		
86	LEFT	vss_go_ngpio_l4	pnl_go	NGPIO
87	LEFT	ext_iopad0_addr15		NGPIO
	inout	bus_addr		
88	LEFT	ext_iopad0_addr14		NGPIO
	inout	bus_addr		
89	LEFT	ext_iopad0_addr13		NGPIO
	inout	bus_addr		
90	LEFT	vss_gcs_ngpio_l15	pnl_gcs	NGPIO

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
91	LEFT	ext_iopad0_addr12		NGPIO
	inout	bus_addr		
92	LEFT	vdd_vop_ngpio_l3	pnl_vop	NGPIO
93	LEFT	ext_iopad0_addr11		NGPIO
	inout	bus_addr		
94	LEFT	vss_gcs_ngpio_l14	pnl_gcs	NGPIO
95	LEFT	ext_iopad0_addr10		NGPIO
	inout	bus_addr		
96	LEFT	ext_iopad0_addr9		NGPIO
	inout	bus_addr		
97	LEFT	ext_iopad0_addr8		NGPIO
	inout	bus_addr		
98	LEFT	vdd_vc_ngpio_l6	pnl_vc	NGPIO
99	LEFT	ext_iopad0_addr7		NGPIO
	inout	bus_addr		
100	LEFT	vss_gcs_ngpio_l13	pnl_gcs	NGPIO
101	LEFT	ext_iopad0_addr6		NGPIO
	inout	bus_addr		
102	LEFT	vdd_vc_ngpio_l5	pnl_vc	NGPIO
103	LEFT	ext_iopad0_addr5		NGPIO
	inout	bus_addr		
104	LEFT	vss_gcs_ngpio_l12	pnl_gcs	NGPIO
105	LEFT	ext_iopad0_addr4		NGPIO
	inout	bus_addr		
106	LEFT	vss_go_ngpio_l3	pnl_go	NGPIO
107	LEFT	ext_iopad0_addr3		NGPIO
	inout	bus_addr		
108	LEFT	ext_iopad0_addr2		NGPIO
	inout	bus_addr		
109	LEFT	ext_iopad0_chip_select_7__cs		NGPIO
	output	bus_cs_		
110	LEFT	vss_gcs_ngpio_l11	pnl_gcs	NGPIO

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
111	LEFT	ext_iopad0_chip_select_6__cs		NGPIO
	output	bus_cs_		
112	LEFT	ext_iopad0_chip_select_5__cs		NGPIO
	output	bus_cs_		
113	LEFT	ext_iopad0_chip_select_4__cs		NGPIO
	output	bus_cs_		
114	LEFT	vdd_vop_ngpio_l2	pnl_vop	NGPIO
115	LEFT	ext_iopad0_chip_select_3__cs		NGPIO
	output	bus_cs_		
116	LEFT	vss_gcs_ngpio_l10	pnl_gcs	NGPIO
117	LEFT	ext_iopad0_chip_select_2__cs		NGPIO
	output	bus_cs_		
118	LEFT	vdd_vc_ngpio_l4	pnl_vc	NGPIO
119	LEFT	ext_iopad0_chip_select_1__cs		NGPIO
	output	bus_cs_		
120	LEFT	ext_iopad0_chip_select_0__cs		NGPIO
	output	bus_cs_		
121	LEFT	ext_iopad0_as		NGPIO
	inout	bus_as_		
122	LEFT	vss_gcs_ngpio_l9	pnl_gcs	NGPIO
123	LEFT	ext_iopad0_rw		NGPIO
	inout	bus_rw_		
124	LEFT	ext_iopad0_be3		NGPIO
	inout	bus_be_		
125	LEFT	ext_iopad0_be2		NGPIO
	inout	bus_be_		
126	LEFT	vdd_vc_ngpio_l3	pnl_vc	NGPIO
127	LEFT	ext_iopad0_be1		NGPIO
	inout	bus_be_		
128	LEFT	vss_gcs_ngpio_l8	pnl_gcs	NGPIO
129	LEFT	ext_iopad0_be0		NGPIO
	inout	bus_be_		
130	LEFT	ext_iopad0_ready		NGPIO
	inout	bus_ready_		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
131	LEFT	vss_go_ngpio_l2	pnl_go	NGPIO
132	LEFT	ext_iopad0_err		NGPIO
	inout	bus_err_		
133	LEFT	ext_iopad0_burst1		NGPIO
	inout	bus_burst		
134	LEFT	vss_gcs_ngpio_l7	pnl_gcs	NGPIO
135	LEFT	ext_iopad0_burst0		NGPIO
	inout	bus_burst		
136	LEFT	vdd_vop_ngpio_l1	pnl_vop	NGPIO
137	LEFT	ext_iopad0_br_ack1		NGPIO
	inout	bus_br_ack		
138	LEFT	ext_iopad0_br_ack0		NGPIO
	inout	bus_br_ack		
139	LEFT	ext_iopad0_dma_req_3__dreq		NGPIO
	input	bus_dreq_		
140	LEFT	vss_gcs_ngpio_l6	pnl_gcs	NGPIO
141	LEFT	ext_iopad0_dma_req_2__dreq		NGPIO
	input	bus_dreq_		
142	LEFT	ext_iopad0_dma_req_1__dreq		NGPIO
	input	bus_dreq_		
143	LEFT	ext_iopad0_dma_req_0__dreq		NGPIO
	input	bus_dreq_		
144	LEFT	vdd_vc_ngpio_l2	pnl_vc	NGPIO
145	LEFT	ext_iopad0_dack3		NGPIO
	output	bus_dack_		
146	LEFT	vss_gcs_ngpio_l5	pnl_gcs	NGPIO
147	LEFT	ext_iopad0_dack2		NGPIO
	output	bus_dack_		
148	LEFT	ext_iopad0_dack1		NGPIO
	output	bus_dack_		
149	LEFT	vdd_vc_ngpio_l1	pnl_vc	NGPIO
150	LEFT	ext_iopad0_dack0		NGPIO
	output	bus_dack_		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
151	LEFT	ext_iopad0_bus_req_3__req		NGPIO
	input	bus_req_		
152	LEFT	ext_iopad0_bus_req_2__req		NGPIO
	input	bus_req_		
153	LEFT	ext_iopad0_bus_req_1__req		NGPIO
	input	bus_req_		
154	LEFT	vss_gcs_ngpio_l4	pnl_gcs	NGPIO
155	LEFT	ext_iopad0_bus_req_0__req		NGPIO
	input	bus_req_		
156	LEFT	vss_go_ngpio_l1	pnl_go	NGPIO
157	LEFT	ext_iopad0_bgrnt_3__grant		NGPIO
	output	bus_grant_		
158	LEFT	ext_iopad0_bgrnt_2__grant		NGPIO
	output	bus_grant_		
159	LEFT	ext_iopad0_bgrnt_1__grant		NGPIO
	output	bus_grant_		
160	LEFT	ext_iopad0_bgrnt_0__grant		NGPIO
	output	bus_grant_		
161	LEFT	vss_gcs_ngpio_l3	pnl_gcs	NGPIO
162	LEFT	pnl_filler_4g_l0	pnl_filler_4g	FILLER
163	LEFT	pnl_filler_2g_l0	pnl_filler_2g	FILLER
164	LEFT	vdd_vop_ngpio_l0	pnl_vop	NGPIO
165	LEFT	ext_iopad0_birq_3__irq		NGPIO
	input	bus_irq		
166	LEFT	ext_iopad0_birq_2__irq		NGPIO
	input	bus_irq		
167	LEFT	vss_gcs_ngpio_l2	pnl_gcs	NGPIO
168	LEFT	ext_iopad0_birq_1__irq		NGPIO
	input	bus_irq		
169	LEFT	ext_iopad0_birq_0__irq		NGPIO
	input	bus_irq		
170	LEFT	vdd_vc_ngpio_l0	pnl_vc	NGPIO

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
171	LEFT	ext_iopad0_cs_toggle_		NGPIO
	input	bus_cs_toggle_		
172	LEFT	ext_iopad0_auto_rdy_en		NGPIO
	input	bus_auto_ready_en_		
173	LEFT	ext_iopad0_flash_byte		NGPIO
	output	flash_byte_		
174	LEFT	vss_gcs_ngpio_l1	pnl_gcs	NGPIO
175	LEFT	ext_iopad0_bit16		NGPIO
	input	bus_16_		
176	LEFT	ext_iopad0_bit8		NGPIO
	input	bus_8_		
177	LEFT	vss_gcs_ngpio_l0	pnl_gcs	NGPIO
178	LEFT	ext_iopad0_FLASH_RDY__0__flash_rdy_		NGPIO
	input	flash_ready_		
179	LEFT	ext_iopad0_FLASH_RDY__1__flash_rdy_		NGPIO
	input	flash_ready_		
180	LEFT	ext_iopad0_clk_out		NGPIO
	output	ext_clk_out		
181	LEFT	vss_go_ngpio_l0	pnl_go	NGPIO
182	LEFT	pnl_filler_16g_l0	pnl_filler_16g	FILLER
183	LEFT	pnl_filler_8g_l0	pnl_filler_8g	FILLER
184	LEFT	pnl_filler_4g_dummy		DUMMY(deleated for DRC)
185	LEFT	pnl_filler_std2sstl_bkp_l0	pnl_filler_std2sstl	BREAKER
186	LEFT	vss_go_bkp_l5	pnl_go	NGPIO BKP
187	LEFT	rtc_pad_rtc_clk_i		NGPIO BKP
	input	rtc_clk		
188	LEFT	vss_gcs_bkp_l16	pnl_gcs	NGPIO BKP
189	LEFT	rtc_pad_rtc_hold_i_		NGPIO BKP
	input	rtc_hold_		
190	LEFT	rtc_pad_rtc_reset_i_		NGPIO BKP
	input	rtc_reset_		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
191	LEFT	pp_iopad0_PWMIN_5__pwm_in_pad		NGPIO BKP
	input	pp_pwm_in		
192	LEFT	vdd_vc_bkp_l8	pnl_vc	NGPIO BKP
193	LEFT	pp_iopad0_PWMIN_4__pwm_in_pad		NGPIO BKP
	input	pp_pwm_in		
194	LEFT	vss_gcs_bkp_l15	pnl_gcs	NGPIO BKP
195	LEFT	pp_iopad0_PWMIN_3__pwm_in_pad		NGPIO BKP
	input	pp_pwm_in		
196	LEFT	pp_iopad0_PWMIN_2__pwm_in_pad		NGPIO BKP
	input	pp_pwm_in		
197	LEFT	pp_iopad0_PWMIN_1__pwm_in_pad		NGPIO BKP
	input	pp_pwm_in		
198	LEFT	vdd_vc_bkp_l7	pnl_vc	NGPIO BKP
199	LEFT	pp_iopad0_PWMIN_0__pwm_in_pad		NGPIO BKP
	input	pp_pwm_in		
200	LEFT	vss_gcs_bkp_l14	pnl_gcs	NGPIO BKP
201	LEFT	pp_iopad0_PWM_11__pwm_out_pad		NGPIO BKP
	output	pp_pwm_out		
202	LEFT	vdd_vop_bkp_l4	pnl_vop	NGPIO BKP
203	LEFT	pp_iopad0_PWM_10__pwm_out_pad		NGPIO BKP
	output	pp_pwm_out		
204	LEFT	pp_iopad0_PWM_9__pwm_out_pad		NGPIO BKP
	output	pp_pwm_out		
205	LEFT	pp_iopad0_PWM_8__pwm_out_pad		NGPIO BKP
	output	pp_pwm_out		
206	LEFT	vss_gcs_bkp_l13	pnl_gcs	NGPIO BKP
207	LEFT	pp_iopad0_PWM_7__pwm_out_pad		NGPIO BKP
	output	pp_pwm_out		
208	LEFT	vss_go_bkp_l4	pnl_go	NGPIO BKP
209	LEFT	pp_iopad0_PWM_6__pwm_out_pad		NGPIO BKP
	output	pp_pwm_out		
210	LEFT	hiz_pad_pwmout_hiz_		NGPIO BKP
	input	pwmout_hiz_		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
211	LEFT	pp_iopad0_PWM_5__pwm_out_pad		NGPIO BKP
	output	pp_pwm_out		
212	LEFT	vss_gcs_bkp_l12	pnl_gcs	NGPIO BKP
213	LEFT	pp_iopad0_PWM_4__pwm_out_pad		NGPIO BKP
	output	pp_pwm_out		
214	LEFT	pp_iopad0_PWM_3__pwm_out_pad		NGPIO BKP
	output	pp_pwm_out		
215	LEFT	pp_iopad0_PWM_2__pwm_out_pad		NGPIO BKP
	output	pp_pwm_out		
216	LEFT	vdd_vc_bkp_l6	pnl_vc	NGPIO BKP
217	LEFT	pp_iopad0_PWM_1__pwm_out_pad		NGPIO BKP
	output	pp_pwm_out		
218	LEFT	vss_gcs_bkp_l11	pnl_gcs	NGPIO BKP
219	LEFT	pp_iopad0_PWM_0__pwm_out_pad		NGPIO BKP
	output	pp_pwm_out		
220	LEFT	pp_iopad0_PLSCNT_5__pz_pad		NGPIO BKP
	input	pp_pz		
221	LEFT	pp_iopad0_PLSCNT_5__pb_pad		NGPIO BKP
	input	pp_pb		
222	LEFT	vdd_vc_bkp_l5	pnl_vc	NGPIO BKP
223	LEFT	vss_gcs_bkp_l10	pnl_gcs	NGPIO BKP
224	LEFT	pp_iopad0_PLSCNT_5__pa_pad		NGPIO BKP
	input	pp_pa		
225	LEFT	vdd_vop_bkp_l3	pnl_vop	NGPIO BKP
226	LEFT	pp_iopad0_PLSCNT_4__pz_pad		NGPIO BKP
	input	pp_pz		
227	LEFT	vss_gcs_bkp_l9	pnl_gcs	NGPIO BKP
228	LEFT	pp_iopad0_PLSCNT_4__pb_pad		NGPIO BKP
	input	pp_pb		
229	LEFT	pp_iopad0_PLSCNT_4__pa_pad		NGPIO BKP
	input	pp_pa		
230	LEFT	vss_go_bkp_l3	pnl_go	NGPIO BKP

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
231	LEFT	pp_iopad0_PLSCNT_3__pz_pad		NGPIO BKP
	input	pp_pz		
232	LEFT	pp_iopad0_PLSCNT_3__pb_pad		NGPIO BKP
	input	pp_pb		
233	LEFT	vss_gcs_bkp_l8	pnl_gcs	NGPIO BKP
234	LEFT	pp_iopad0_PLSCNT_3__pa_pad		NGPIO BKP
	input	pp_pa		
235	LEFT	vdd_vc_bkp_l4	pnl_vc	NGPIO BKP
236	LEFT	pp_iopad0_PLSCNT_2__pz_pad		NGPIO BKP
	input	pp_pz		
237	LEFT	pp_iopad0_PLSCNT_2__pb_pad		NGPIO BKP
	input	pp_pb		
238	LEFT	vss_gcs_bkp_l7	pnl_gcs	NGPIO BKP
239	LEFT	pp_iopad0_PLSCNT_2__pa_pad		NGPIO BKP
	input	pp_pa		
240	LEFT	pp_iopad0_PLSCNT_1__pz_pad		NGPIO BKP
	input	pp_pz		
241	LEFT	pp_iopad0_PLSCNT_1__pb_pad		NGPIO BKP
	input	pp_pb		
242	LEFT	vdd_vc_bkp_l3	pnl_vc	NGPIO BKP
243	LEFT	pp_iopad0_PLSCNT_1__pa_pad		NGPIO BKP
	input	pp_pa		
244	LEFT	vss_gcs_bkp_l6	pnl_gcs	NGPIO BKP
245	LEFT	pp_iopad0_PLSCNT_0__pz_pad		NGPIO BKP
	input	pp_pz		
246	LEFT	pp_iopad0_PLSCNT_0__pb_pad		NGPIO BKP
	input	pp_pb		
247	LEFT	pp_iopad0_PLSCNT_0__pa_pad		NGPIO BKP
	input	pp_pa		
248	LEFT	vdd_vop_bkp_l2	pnl_vop	NGPIO BKP
249	LEFT	clk_iopad0_EM_7__em_reset_in_pad		NGPIO BKP
	input	em_reset		
250	LEFT	vss_gcs_bkp_l5	pnl_gcs	NGPIO BKP

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
251	LEFT	clk_iopad0_EM_6__em_reset_in_pad		NGPIO BKP
	input	em_reset		
252	LEFT	vss_go_bkp_l2	pnl_go	NGPIO BKP
253	LEFT	clk_iopad0_EM_5__em_reset_in_pad		NGPIO BKP
	input	em_reset		
254	LEFT	clk_iopad0_EM_4__em_reset_in_pad		NGPIO BKP
	input	em_reset		
255	LEFT	vss_gcs_bkp_l4	pnl_gcs	NGPIO BKP
256	LEFT	vdd_vc_bkp_l2	pnl_vc	NGPIO BKP
257	LEFT	clk_iopad0_EM_3__em_reset_in_pad		NGPIO BKP
	input	em_reset		
258	LEFT	vss_gcs_bkp_l3	pnl_gcs	NGPIO BKP
259	LEFT	clk_iopad0_EM_2__em_reset_in_pad		NGPIO BKP
	input	em_reset		
260	LEFT	vdd_vc_bkp_l1	pnl_vc	NGPIO BKP
261	LEFT	clk_iopad0_EM_1__em_reset_in_pad		NGPIO BKP
	input	em_reset		
262	LEFT	clk_iopad0_EM_0__em_reset_in_pad		NGPIO BKP
	input	em_reset		
263	LEFT	vss_gcs_bkp_l2	pnl_gcs	NGPIO BKP
264	LEFT	clk_iopad0_clk_outer		NGPIO BKP
	output	clk_outer_p,n		
265	LEFT	vss_gcs_bkp_l1	pnl_gcs	NGPIO BKP
266	LEFT	hiz_pad_clk_hiz_		NGPIO BKP
	input	clk_hiz_		
267	LEFT	vdd_vop_bkp_l1	pnl_vop	NGPIO BKP
268	LEFT	clk_iopad0_reset_outer		NGPIO BKP
	output	reset_outer_		
269	LEFT	vdd_vc_bkp_l0	pnl_vc	NGPIO BKP
270	LEFT	clk_iopad0_reset_in		NGPIO BKP
	input	reset_in_		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
271	LEFT	vss_gcs_bkp_l0	pnl_gcs	NGPIO BKP
272	LEFT	clk_iopad0_clk_sel_iopad		NGPIO BKP
	input	clk_sel		
273	LEFT	vss_go_bkp_l1	pnl_go	NGPIO BKP
274	LEFT	clk_iopad0_clk_reset		NGPIO BKP
	input	clk_reset_		
275	LEFT	clk_iopad0_F0		NGPIO BKP
	input	F		
276	LEFT	clk_iopad0_F1		NGPIO BKP
	input	F		
277	LEFT	clk_iopad0_F2		NGPIO BKP
	input	F		
278	LEFT	clk_iopad0_F3		NGPIO BKP
	input	F		
279	LEFT	clk_iopad0_F4		NGPIO BKP
	input	F		
280	LEFT	clk_iopad0_F5		NGPIO BKP
	input	F		
281	LEFT	clk_iopad0_FIN		NGPIO BKP
	input	FIN		
282	LEFT	vdd_vop_bkp_l0	pnl_vop	NGPIO BKP
283	LEFT	clk_iopad0_BP		NGPIO BKP
	input	BP		
284	LEFT	clk_iopad0_R0		NGPIO BKP
	input	R		
285	LEFT	clk_iopad0_R1		NGPIO BKP
	input	R		
286	LEFT	clk_iopad0_R2		NGPIO BKP
	input	R		
287	LEFT	clk_iopad0_R3		NGPIO BKP
	input	R		
288	LEFT	clk_iopad0_OEB		NGPIO BKP
	input	OEB		
289	LEFT	vss_go_bkp_l0	pnl_go	NGPIO BKP
290	LEFT	clk_iopad0_OD		NGPIO BKP
	input	OD		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
291	LEFT	clk_iopad0_PD		NGPIO BKP
	input	PD		
292	LEFT	pnl_filler_std2sstl_bkp_l1	pnl_filler_std2sstl	NGPIO BKP BREAKER
293	LEFT	clk_iopad0_PRCUT2P1	PRCUT2P	PLL Analog Breaker
294	LEFT	clk_iopad0_PVSS2P	PVSS2P	PLL Analog
295	LEFT	clk_iopad0_PVDD2P	PVDD2P	PLL Analog
296	LEFT	clk_iopad0_PVDD1P1	PVDD1P	PLL Analog
297	LEFT	clk_iopad0_PVDD1P0	PVDD1P	PLL Analog
298	LEFT	clk_iopad0_PVSS1PC0	PVSS1PC	PLL Analog
299	LEFT	clk_iopad0_PVSS1P1	PVSS1P	PLL Analog
300	LEFT	clk_iopad0_PVSS1P0	PVSS1P	PLL Analog
301	LEFT	clk_iopad0_PVDD1PC0	PVDD1PC	PLL Analog
302	LEFT	clk_iopad0_PRCUT2P0	PRCUT2P	PLL Analog Breaker
CORNER	BOTTOM	CORNER_BL	PCORNERDGZ	PLL Analog Corner
BREAKER	BOTTOM	pnl_filler_std2sstl_b0	pnl_filler_std2sstl	NGPIO Breaker
303	BOTTOM	uart_iopad0_uart3_dcd_pad		NGPIO
	input	uart3_dcd_pad_in		
304	BOTTOM	uart_iopad0_uart3_srx_pad		NGPIO
	input	uart3_srx_pad_in		
305	BOTTOM	uart_iopad0_uart3_stx_pad		NGPIO
	output	uart3_stx_pad_out		
306	BOTTOM	uart_iopad0_uart3_dtr_pad		NGPIO
	output	uart3_dtr_pad_out		
307	BOTTOM	vss_go_ngpio_b6	pnl_go	NGPIO
308	BOTTOM	hiz_pad_UART_PAD_HIZ_GEN_3__uart_hiz_		NGPIO
	input	uart_hiz_		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
309	BOTTOM	uart_iopad0_uart3_dsr_pad		NGPIO
	input	uart3_dsr_pad_in		
310	BOTTOM	uart_iopad0_uart3_rts_pad		NGPIO
	output	uart3_rts_pad_out		
311	BOTTOM	uart_iopad0_uart3_cts_pad		NGPIO
	input	uart3_cts_pad_in		
312	BOTTOM	vdd_vop_ngpio_b5	pnl_vop	NGPIO
313	BOTTOM	uart_iopad0_uart3_ri_pad		NGPIO
	input	uart3_ri_pad_in		
314	BOTTOM	uart_iopad0_uart2_dcd_pad		NGPIO
	input	uart2_dcd_pad_in		
315	BOTTOM	uart_iopad0_uart2_srx_pad		NGPIO
	input	uart2_srx_pad_in		
316	BOTTOM	uart_iopad0_uart2_stx_pad		NGPIO
	output	uart2_stx_pad_out		
317	BOTTOM	vss_go_ngpio_b5	pnl_go	NGPIO
318	BOTTOM	uart_iopad0_uart2_dtr_pad		NGPIO
	output	uart2_dtr_pad_out		
319	BOTTOM	hiz_pad_UART_PAD_HIZ_GEN_2__uart_hiz_		NGPIO
	input	uart_hiz_		
320	BOTTOM	uart_iopad0_uart2_dsr_pad		NGPIO
	input	uart2_dsr_pad_in		
321	BOTTOM	uart_iopad0_uart2_rts_pad		NGPIO
	output	uart2_rts_pad_out		
322	BOTTOM	vdd_vop_ngpio_b4	pnl_vop	NGPIO
323	BOTTOM	uart_iopad0_uart2_cts_pad		NGPIO
	input	uart2_cts_pad_in		
324	BOTTOM	uart_iopad0_uart2_ri_pad		NGPIO
	input	uart2_ri_pad_in		
325	BOTTOM	uart_iopad0_uart1_dcd_pad		NGPIO
	input	uart1_dcd_pad_in		
326	BOTTOM	uart_iopad0_uart1_srx_pad		NGPIO
	input	uart1_srx_pad_in		
327	BOTTOM	vss_gcs_ngpio_b16	pnl_gcs	NGPIO
328	BOTTOM	uart_iopad0_uart1_stx_pad		NGPIO
	output	uart1_stx_pad_out		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
329	BOTTOM	vdd_vc_ngpio_b10	pnl_vc	NGPIO
330	BOTTOM	uart_iopad0_uart1_dtr_pad		NGPIO
	output	uart1_dtr_pad_out		
331	BOTTOM	vss_gcs_ngpio_b15	pnl_gcs	NGPIO
332	BOTTOM	hiz_pad_UART_PAD_HIZ_GEN_1__uart_hiz_		NGPIO
	input	uart_hiz_		
333	BOTTOM	vdd_vc_ngpio_b9	pnl_vc	NGPIO
334	BOTTOM	uart_iopad0_uart1_dsr_pad		NGPIO
	input	uart1_dsr_pad_in		
335	BOTTOM	uart_iopad0_uart1_rts_pad		NGPIO
	output	uart1_rts_pad_out		
336	BOTTOM	vss_gcs_ngpio_b14	pnl_gcs	NGPIO
337	BOTTOM	uart_iopad0_uart1_cts_pad		NGPIO
	input	uart1_cts_pad_in		
338	BOTTOM	uart_iopad0_uart1_ri_pad		NGPIO
	input	uart1_ri_pad_in		
339	BOTTOM	vss_go_ngpio_b4	pnl_go	NGPIO
340	BOTTOM	uart_iopad0_uart0_dcd_pad		NGPIO
	input	uart0_dcd_pad_in		
341	BOTTOM	vss_gcs_ngpio_b13	pnl_gcs	NGPIO
342	BOTTOM	uart_iopad0_uart0_srx_pad		NGPIO
	input	uart0_srx_pad_in		
343	BOTTOM	vdd_vop_ngpio_b3	pnl_vop	NGPIO
344	BOTTOM	uart_iopad0_uart0_stx_pad		NGPIO
	output	uart0_stx_pad_out		
345	BOTTOM	vss_gcs_ngpio_b12	pnl_gcs	NGPIO
346	BOTTOM	uart_iopad0_uart0_dtr_pad		NGPIO
	output	uart0_dtr_pad_out		
347	BOTTOM	vdd_vc_ngpio_b8	pnl_vc	NGPIO
348	BOTTOM	hiz_pad_UART_PAD_HIZ_GEN_0__uart_hiz_		NGPIO
	input	uart_hiz_		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
349	BOTTOM	uart_iopad0_uart0_dsr_pad		NGPIO
	input	uart0_dsr_pad_in		
350	BOTTOM	uart_iopad0_uart0_rts_pad		NGPIO
	output	uart0_rts_pad_out		
351	BOTTOM	vss_gcs_ngpio_b11	pnl_gcs	NGPIO
352	BOTTOM	uart_iopad0_uart0_cts_pad		NGPIO
	input	uart0_cts_pad_in		
353	BOTTOM	vdd_vc_ngpio_b7	pnl_vc	NGPIO
354	BOTTOM	uart_iopad0_uart0_ri_pad		NGPIO
	input	uart0_ri_pad_in		
355	BOTTOM	vss_gcs_ngpio_b10	pnl_gcs	NGPIO
356	BOTTOM	spi_pad1_spi_miso_i		NGPIO
	input	spi_miso1		
357	BOTTOM	vss_go_ngpio_b3	pnl_go	NGPIO
358	BOTTOM	spi_pad1_spi_mosi_o		NGPIO
	output	spi_mosi1		
359	BOTTOM	spi_pad1_spi_sck_o		NGPIO
	output	spi_sck1		
360	BOTTOM	vss_gcs_ngpio_b9	pnl_gcs	NGPIO
361	BOTTOM	hiz_pad_SPI_PAD_HIZ_GEN_1__spi_hiz_		NGPIO
	input	spi_hiz_		
362	BOTTOM	spi_pad1_spi_ss_3_o_		NGPIO
	output	spi_ss1_		
363	BOTTOM	vdd_vop_ngpio_b2	pnl_vop	NGPIO
364	BOTTOM	spi_pad1_spi_ss_2_o_		NGPIO
	output	spi_ss1_		
365	BOTTOM	vss_gcs_ngpio_b8	pnl_gcs	NGPIO
366	BOTTOM	spi_pad1_spi_ss_1_o_		NGPIO
	output	spi_ss1_		
367	BOTTOM	vdd_vc_ngpio_b6	pnl_vc	NGPIO
368	BOTTOM	spi_pad1_spi_ss_0_o_		NGPIO
	output	spi_ss1_		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
369	BOTTOM	vss_gcs_ngpio_b7	pnl_gcs	NGPIO
370	BOTTOM	spi_pad0_spi_miso_i		NGPIO
	input	spi_miso0		
371	BOTTOM	vdd_vc_ngpio_b5	pnl_vc	NGPIO
372	BOTTOM	spi_pad0_spi_mosi_o		NGPIO
	output	spi_mosi0		
373	BOTTOM	spi_pad0_spi_sck_o		NGPIO
	output	spi_sck0		
374	BOTTOM	hiz_pad_SPI_PAD_HIZ_GEN_0__spi_hiz_		NGPIO
	input	spi_hiz_		
375	BOTTOM	vss_gcs_ngpio_b6	pnl_gcs	NGPIO
376	BOTTOM	spi_pad0_spi_ss_3_o_		NGPIO
	output	spi_ss0_		
377	BOTTOM	vss_go_ngpio_b2	pnl_go	NGPIO
378	BOTTOM	spi_pad0_spi_ss_2_o_		NGPIO
	output	spi_ss0_		
379	BOTTOM	vdd_vc_ngpio_b4	pnl_vc	NGPIO
380	BOTTOM	spi_pad0_spi_ss_1_o_		NGPIO
	output	spi_ss0_		
381	BOTTOM	vdd_vop_ngpio_b1	pnl_vop	NGPIO
382	BOTTOM	spi_pad0_spi_ss_0_o_		NGPIO
	output	spi_ss0_		
383	BOTTOM	vss_gcs_ngpio_b5	pnl_gcs	NGPIO
384	BOTTOM	oce_pad_oce_reload_o_		NGPIO
	output	oce_reload_		
385	BOTTOM	oce_pad_oce_ss_i_		NGPIO
	input	oce_ss_		
386	BOTTOM	hiz_pad_oce_hiz_		NGPIO
	input	oce_hiz_		
387	BOTTOM	vdd_vc_ngpio_b3	pnl_vc	NGPIO
388	BOTTOM	oce_pad_oce_sck_i		NGPIO
	input	oce_sck		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
389	BOTTOM	vss_gcs_ngpio_b4	pnl_gcs	NGPIO
390	BOTTOM	oce_pad_oce_miso_o		NGPIO
	output	oce_miso		
391	BOTTOM	vdd_vc_ngpio_b2	pnl_vc	NGPIO
392	BOTTOM	oce_pad_oce_mosi_i		NGPIO
	input	oce_mosi		
393	BOTTOM	vss_gcs_ngpio_b3	pnl_gcs	NGPIO
394	BOTTOM	i2c_pad_i2c_sda_pad		NGPIO
	inout	i2c_sda		
395	BOTTOM	vss_go_ngpio_b1	pnl_go	NGPIO
396	BOTTOM	hiz_pad_i2c_hiz_		NGPIO
	input	i2c_hiz_		
397	BOTTOM	i2c_pad_i2c_scl_pad		NGPIO
	inout	i2c_scl		
398	BOTTOM	clk_iopad0_pll_fin_sel_iopad		NGPIO
	input	pll_fin_sel		
399	BOTTOM	vss_gcs_ngpio_b2	pnl_gcs	NGPIO
400	BOTTOM	clk_iopad0_lvds_fout_sel_iopad		NGPIO
	input	lvds_fout_sel		
401	BOTTOM	vdd_vop_ngpio_b0	pnl_vop	NGPIO
402	BOTTOM	pio_pad_pio_data_7_io		NGPIO
	inout	pio_data		
403	BOTTOM	pio_pad_pio_data_6_io		NGPIO
	inout	pio_data		
404	BOTTOM	pio_pad_pio_data_5_io		NGPIO
	inout	pio_data		
405	BOTTOM	vdd_vc_ngpio_b1	pnl_vc	NGPIO
406	BOTTOM	hiz_pad_pio_hiz_		NGPIO
	inout	pio_hiz_		
407	BOTTOM	pio_pad_pio_data_4_io		NGPIO
	inout	pio_data		
408	BOTTOM	pio_pad_pio_data_3_io		NGPIO
	inout	pio_data		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
409	BOTTOM	vss_gcs_ngpio_b1	pnl_gcs	NGPIO
410	BOTTOM	pio_pad_pio_data_2_io		NGPIO
	inout	pio_data		
411	BOTTOM	pio_pad_pio_data_1_io		NGPIO
	inout	pio_data		
412	BOTTOM	vdd_vc_ngpio_b0	pnl_vc	NGPIO
413	BOTTOM	pio_pad_pio_data_0_io		NGPIO
	inout	pio_data		
414	BOTTOM	vss_gcs_ngpio_b0	pnl_gcs	NGPIO
415	BOTTOM	hiz_pad_sdram_hiz_		NGPIO
	input	sdram_hiz_		
416	BOTTOM	hiz_pad_link_sdram_hiz_		NGPIO
	input	link_sdram_hiz_		
417	BOTTOM	hiz_pad_soft_hiz_en_		NGPIO
	input	soft_hiz_en_		
418	BOTTOM	vss_go_ngpio_b0	pnl_go	NGPIO
419	BOTTOM	pnl_filler_std2sstl_b1	pnl_filler_std2sstl	BREAKER
420	BOTTOM	pnl_filler_sstl_16g_b0	pnl_filler_sstl_16g	FILLER
421	BOTTOM	pnl_filler_sstl_8g_b0	pnl_filler_sstl_8g	FILLER
422	BOTTOM	pnl_filler_sstl_4g_b0	pnl_filler_sstl_4g	FILLER
423	BOTTOM	vss_go_sstl_b12	pnl_sstl_go	SSTL
424	BOTTOM	sdram_iopad0_DQ_143__dq		SSTL
	inout	sdram_dq		
425	BOTTOM	sdram_iopad0_DQ_142__dq		SSTL
	inout	sdram_dq		
426	BOTTOM	vss_gcs_sstl_b26	pnl_sstl_gcs	SSTL
427	BOTTOM	sdram_iopad0_DQ_141__dq		SSTL
	inout	sdram_dq		
428	BOTTOM	sdram_iopad0_DQ_140__dq		SSTL
	inout	sdram_dq		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
429	BOTTOM	vdd_vq_sstl_b12	pnl_sstl_vq	SSTL
430	BOTTOM	sdram_iopad0_DQ_139__dq		SSTL
	inout	sdram_dq		
431	BOTTOM	vss_gcs_sstl_b25	pnl_sstl_gcs	SSTL
432	BOTTOM	sdram_iopad0_DQ_138__dq		SSTL
	inout	sdram_dq		
433	BOTTOM	vdd_vc_sstl_b12	pnl_sstl_vc	SSTL
434	BOTTOM	sdram_iopad0_DQ_137__dq		SSTL
	inout	sdram_dq		
435	BOTTOM	vss_gcs_sstl_b24	pnl_sstl_gcs	SSTL
436	BOTTOM	sdram_iopad0_DQ_136__dq		SSTL
	inout	sdram_dq		
437	BOTTOM	sdram_iopad0_DQM_17__dqm		SSTL
	output	sdram_dqm		
438	BOTTOM	sdram_iopad0_DQS_17__dqs		SSTL
	inout	sdram_dqs		
439	BOTTOM	vss_go_sstl_b11	pnl_sstl_go	SSTL
440	BOTTOM	sdram_iopad0_DQ_135__dq		SSTL
	inout	sdram_dq		
441	BOTTOM	sdram_iopad0_DQ_134__dq		SSTL
	inout	sdram_dq		
442	BOTTOM	vss_gcs_sstl_b23	pnl_sstl_gcs	SSTL
443	BOTTOM	sdram_iopad0_DQ_133__dq		SSTL
	inout	sdram_dq		
444	BOTTOM	vdd_vq_sstl_b11	pnl_sstl_vq	SSTL
445	BOTTOM	sdram_iopad0_DQ_132__dq		SSTL
	inout	sdram_dq		
446	BOTTOM	vss_gcs_sstl_b22	pnl_sstl_gcs	SSTL
447	BOTTOM	sdram_iopad0_DQ_131__dq		SSTL
	inout	sdram_dq		
448	BOTTOM	vss_go_sstl_b10	pnl_sstl_go	SSTL

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
449	BOTTOM	sdram_iopad0_CLK_2__clk		SSTL
	output	sdram_clk		
450	BOTTOM	vss_gcs_sstl_b21	pnl_sstl_gcs	SSTL
451	BOTTOM	sdram_iopad0_CLK_2__clk_		SSTL
	output	sdram_clk_		
452	BOTTOM	vdd_vp_sstl_b5	pnl_sstl_vp	SSTL
453	BOTTOM	sdram_iopad0_DQ_130__dq		SSTL
	inout	sdram_dq		
454	BOTTOM	sdram_iopad0_DQ_129__dq		SSTL
	inout	sdram_dq		
455	BOTTOM	vss_gcs_sstl_b20	pnl_sstl_gcs	SSTL
456	BOTTOM	sdram_iopad0_DQ_128__dq		SSTL
	inout	sdram_dq		
457	BOTTOM	sdram_iopad0_DQM_16__dqm		SSTL
	output	sdram_dqm		
458	BOTTOM	vdd_vq_sstl_b10	pnl_sstl_vq	SSTL
459	BOTTOM	sdram_iopad0_DQS_16__dqs		SSTL
	inout	sdram_dqs		
460	BOTTOM	vdd_vc_sstl_b11	pnl_sstl_vc	SSTL
461	BOTTOM	sdram_iopad0_DQ_127__dq		SSTL
	inout	sdram_dq		
462	BOTTOM	vss_go_sstl_b9	pnl_sstl_go	SSTL
463	BOTTOM	sdram_iopad0_DQ_126__dq		SSTL
	inout	sdram_dq		
464	BOTTOM	vss_gcs_sstl_b19	pnl_sstl_gcs	SSTL
465	BOTTOM	sdram_iopad0_DQ_125__dq		SSTL
	inout	sdram_dq		
466	BOTTOM	vdd_vc_sstl_b10	pnl_sstl_vc	SSTL
467	BOTTOM	sdram_iopad0_DQ_124__dq		SSTL
	inout	sdram_dq		
468	BOTTOM	sdram_iopad0_DQ_123__dq		SSTL
	inout	sdram_dq		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
469	BOTTOM	vss_gcs_sstl_b18	pnl_sstl_gcs	SSTL
470	BOTTOM	sdram_iopad0_DQ_122__dq		SSTL
	inout	sdram_dq		
471	BOTTOM	sdram_iopad0_DQ_121__dq		SSTL
	inout	sdram_dq		
472	BOTTOM	vdd_vq_sstl_b9	pnl_sstl_vq	SSTL
473	BOTTOM	sdram_iopad0_DQ_120__dq		SSTL
	inout	sdram_dq		
474	BOTTOM	vss_gcs_sstl_b17	pnl_sstl_gcs	SSTL
475	BOTTOM	sdram_iopad0_DQM_15__dqm		SSTL
	output	sdram_dqm		
476	BOTTOM	sdram_iopad0_DQS_15__dqs		SSTL
	inout	sdram_dqs		
477	BOTTOM	sdram_iopad0_DQ_119__dq		SSTL
	inout	sdram_dq		
478	BOTTOM	vss_go_sstl_b8	pnl_sstl_go	SSTL
479	BOTTOM	sdram_iopad0_DQ_118__dq		SSTL
	inout	sdram_dq		
480	BOTTOM	vdd_vp_sstl_b4	pnl_sstl_vp	SSTL
481	BOTTOM	sdram_iopad0_DQ_117__dq		SSTL
	inout	sdram_dq		
482	BOTTOM	vss_gcs_sstl_b16	pnl_sstl_gcs	SSTL
483	BOTTOM	sdram_iopad0_DQ_116__dq		SSTL
	inout	sdram_dq		
484	BOTTOM	vdd_vc_sstl_b9	pnl_sstl_vc	SSTL
485	BOTTOM	sdram_iopad0_DQ_115__dq		SSTL
	inout	sdram_dq		
486	BOTTOM	vdd_vq_sstl_b8	pnl_sstl_vq	SSTL
487	BOTTOM	sdram_iopad0_DQ_114__dq		SSTL
	inout	sdram_dq		
488	BOTTOM	vss_gcs_sstl_b15	pnl_sstl_gcs	SSTL

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
489	BOTTOM	sdram_iopad0_DQ_113__dq		SSTL
	inout	sdram_dq		
490	BOTTOM	vss_go_sstl_b7	pnl_sstl_go	SSTL
491	BOTTOM	sdram_iopad0_DQ_112__dq		SSTL
	inout	sdram_dq		
492	BOTTOM	sdram_iopad0_DQM_14__dqm		SSTL
	output	sdram_dqm		
493	BOTTOM	sstl_vref_b1	pnl_sstl_vref	SSTL
494	BOTTOM	sdram_iopad0_DQS_14__dqs		SSTL
	inout	sdram_dqs		
495	BOTTOM	sdram_iopad0_DQ_111__dq		SSTL
	inout	sdram_dq		
496	BOTTOM	sdram_iopad0_DQ_110__dq		SSTL
	inout	sdram_dq		
497	BOTTOM	sdram_iopad0_DQ_109__dq		SSTL
	inout	sdram_dq		
498	BOTTOM	vss_gcs_sstl_b14	pnl_sstl_gcs	SSTL
499	BOTTOM	sdram_iopad0_DQ_108__dq		SSTL
	inout	sdram_dq		
500	BOTTOM	vdd_vq_sstl_b7	pnl_sstl_vq	SSTL
501	BOTTOM	sdram_iopad0_DQ_107__dq		SSTL
	inout	sdram_dq		
502	BOTTOM	vdd_vp_sstl_b3	pnl_sstl_vp	SSTL
503	BOTTOM	sdram_iopad0_DQ_106__dq		SSTL
	inout	sdram_dq		
504	BOTTOM	sdram_iopad0_DQ_105__dq		SSTL
	inout	sdram_dq		
505	BOTTOM	vss_go_sstl_b6	pnl_sstl_go	SSTL
506	BOTTOM	sdram_iopad0_DQ_104__dq		SSTL
	inout	sdram_dq		
507	BOTTOM	vss_gcs_sstl_b13	pnl_sstl_gcs	SSTL
508	BOTTOM	sdram_iopad0_DQM_13__dqm		SSTL
	output	sdram_dqm		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
509	BOTTOM	vdd_vc_sstl_b8	pnl_sstl_vc	SSTL
510	BOTTOM	sdram_iopad0_DQS_13__dqs		SSTL
	inout	sdram_dqs		
511	BOTTOM	sdram_iopad0_DQ_103__dq		SSTL
	inout	sdram_dq		
512	BOTTOM	vss_gcs_sstl_b12	pnl_sstl_gcs	SSTL
513	BOTTOM	sdram_iopad0_DQ_102__dq		SSTL
	inout	sdram_dq		
514	BOTTOM	vdd_vq_sstl_b6	pnl_sstl_vq	SSTL
515	BOTTOM	sdram_iopad0_DQ_101__dq		SSTL
	inout	sdram_dq		
516	BOTTOM	vdd_vc_sstl_b7	pnl_sstl_vc	SSTL
517	BOTTOM	sdram_iopad0_DQ_100__dq		SSTL
	inout	sdram_dq		
518	BOTTOM	sdram_iopad0_DQ_99__dq		SSTL
	inout	sdram_dq		
519	BOTTOM	sdram_iopad0_DQ_98__dq		SSTL
	inout	sdram_dq		
520	BOTTOM	sdram_iopad0_DQ_97__dq		SSTL
	inout	sdram_dq		
521	BOTTOM	vss_go_sstl_b5	pnl_sstl_go	SSTL
522	BOTTOM	sdram_iopad0_DQ_96__dq		SSTL
	inout	sdram_dq		
523	BOTTOM	vss_gcs_sstl_b11	pnl_sstl_gcs	SSTL
524	BOTTOM	sdram_iopad0_DQM_12__dqm		SSTL
	output	sdram_dqm		
525	BOTTOM	vdd_vc_sstl_b6	pnl_sstl_vc	SSTL
526	BOTTOM	sdram_iopad0_DQS_12__dqs		SSTL
	inout	sdram_dqs		
527	BOTTOM	vss_gcs_sstl_b10	pnl_sstl_gcs	SSTL
528	BOTTOM	sdram_iopad0_DQ_95__dq		SSTL
	inout	sdram_dq		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
529	BOTTOM	vdd_vq_sstl_b5	pnl_sstl_vq	SSTL
530	BOTTOM	sdram_iopad0_DQ_94__dq		SSTL
	inout	sdram_dq		
531	BOTTOM	vdd_vp_sstl_b2	pnl_sstl_vp	SSTL
532	BOTTOM	sdram_iopad0_DQ_93__dq		SSTL
	inout	sdram_dq		
533	BOTTOM	vss_go_sstl_b4	pnl_sstl_go	SSTL
534	BOTTOM	sdram_iopad0_DQ_92__dq		SSTL
	inout	sdram_dq		
535	BOTTOM	sdram_iopad0_DQ_91__dq		SSTL
	inout	sdram_dq		
536	BOTTOM	vss_gcs_sstl_b9	pnl_sstl_gcs	SSTL
537	BOTTOM	sdram_iopad0_DQ_90__dq		SSTL
	inout	sdram_dq		
538	BOTTOM	vdd_vc_sstl_b5	pnl_sstl_vc	SSTL
539	BOTTOM	sdram_iopad0_DQ_89__dq		SSTL
	inout	sdram_dq		
540	BOTTOM	vss_gcs_sstl_b8	pnl_sstl_gcs	SSTL
541	BOTTOM	sdram_iopad0_DQ_88__dq		SSTL
	inout	sdram_dq		
542	BOTTOM	vdd_vq_sstl_b4	pnl_sstl_vq	SSTL
543	BOTTOM	sdram_iopad0_DQM_11__dqm		SSTL
	output	sdram_dqm		
544	BOTTOM	vdd_vc_sstl_b4	pnl_sstl_vc	SSTL
545	BOTTOM	sdram_iopad0_DQS_11__dqs		SSTL
	inout	sdram_dqs		
546	BOTTOM	sdram_iopad0_DQ_87__dq		SSTL
	inout	sdram_dq		
547	BOTTOM	sdram_iopad0_DQ_86__dq		SSTL
	inout	sdram_dq		
548	BOTTOM	sdram_iopad0_DQ_85__dq		SSTL
	inout	sdram_dq		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
549	BOTTOM	vss_go_sstl_b3	pnl_sstl_go	SSTL
550	BOTTOM	sdram_iopad0_DQ_84__dq		SSTL
	inout	sdram_dq		
551	BOTTOM	vss_gcs_sstl_b7	pnl_sstl_gcs	SSTL
552	BOTTOM	sdram_iopad0_DQ_83__dq		SSTL
	inout	sdram_dq		
553	BOTTOM	vdd_vc_sstl_b3	pnl_sstl_vc	SSTL
554	BOTTOM	sdram_iopad0_DQ_82__dq		SSTL
	inout	sdram_dq		
555	BOTTOM	vss_gcs_sstl_b6	pnl_sstl_gcs	SSTL
556	BOTTOM	sdram_iopad0_DQ_81__dq		SSTL
	inout	sdram_dq		
557	BOTTOM	vdd_vq_sstl_b3	pnl_sstl_vq	SSTL
558	BOTTOM	sdram_iopad0_DQ_80__dq		SSTL
	inout	sdram_dq		
559	BOTTOM	sdram_iopad0_DQM_10__dqm		SSTL
	output	sdram_dqm		
560	BOTTOM	vss_go_sstl_b2	pnl_sstl_go	SSTL
561	BOTTOM	sdram_iopad0_DQS_10__dqs		SSTL
	inout	sdram_dqs		
562	BOTTOM	vdd_vp_sstl_b1	pnl_sstl_vp	SSTL
563	BOTTOM	sdram_iopad0_DQ_79__dq		SSTL
	inout	sdram_dq		
564	BOTTOM	sdram_iopad0_DQ_78__dq		SSTL
	inout	sdram_dq		
565	BOTTOM	vss_gcs_sstl_b5	pnl_sstl_gcs	SSTL
566	BOTTOM	sdram_iopad0_DQ_77__dq		SSTL
	inout	sdram_dq		
567	BOTTOM	vdd_vc_sstl_b2	pnl_sstl_vc	SSTL
568	BOTTOM	sdram_iopad0_DQ_76__dq		SSTL
	inout	sdram_dq		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
569	BOTTOM	vss_gcs_sstl_b4	pnl_sstl_gcs	SSTL
570	BOTTOM	sdram_iopad0_DQ_75__dq		SSTL
	inout	sdram_dq		
571	BOTTOM	vdd_vq_sstl_b2	pnl_sstl_vq	SSTL
572	BOTTOM	sdram_iopad0_DQ_74__dq		SSTL
	inout	sdram_dq		
573	BOTTOM	vdd_vc_sstl_b1	pnl_sstl_vc	SSTL
574	BOTTOM	sdram_iopad0_DQ_73__dq		SSTL
	inout	sdram_dq		
575	BOTTOM	sdram_iopad0_DQ_72__dq		SSTL
	inout	sdram_dq		
576	BOTTOM	vss_go_sstl_b1	pnl_sstl_go	SSTL
577	BOTTOM	sdram_iopad0_DQM_9__dqm		SSTL
	output	sdram_dqm		
578	BOTTOM	sdram_iopad0_DQS_9__dqs		SSTL
	inout	sdram_dqs		
579	BOTTOM	vss_gcs_sstl_b3	pnl_sstl_gcs	SSTL
580	BOTTOM	sdram_iopad0_DQ_71__dq		SSTL
	inout	sdram_dq		
581	BOTTOM	vdd_vq_sstl_b1	pnl_sstl_vq	SSTL
582	BOTTOM	sdram_iopad0_DQ_70__dq		SSTL
	inout	sdram_dq		
583	BOTTOM	sdram_iopad0_DQ_69__dq		SSTL
	inout	sdram_dq		
584	BOTTOM	sdram_iopad0_DQ_68__dq		SSTL
	inout	sdram_dq		
585	BOTTOM	vss_go_sstl_b0	pnl_sstl_go	SSTL
586	BOTTOM	sdram_iopad0_oe_		SSTL
	output	sdram_oe_		
587	BOTTOM	vdd_vp_sstl_b0	pnl_sstl_vp	SSTL
588	BOTTOM	sdram_iopad0_dir		SSTL
	output	sdram_dir		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
589	BOTTOM	vss_gcs_sstl_b2	pnl_sstl_gcs	SSTL
590	BOTTOM	sdram_iopad0_DQ_67__dq		SSTL
	inout	sdram_dq		
591	BOTTOM	vdd_vc_sstl_b0	pnl_sstl_vc	SSTL
592	BOTTOM	sdram_iopad0_DQ_66__dq		SSTL
	inout	sdram_dq		
593	BOTTOM	vss_gcs_sstl_b1	pnl_sstl_gcs	SSTL
594	BOTTOM	sdram_iopad0_DQ_65__dq		SSTL
	inout	sdram_dq		
595	BOTTOM	vdd_vq_sstl_b0	pnl_sstl_vq	SSTL
596	BOTTOM	sdram_iopad0_DQ_64__dq		SSTL
	inout	sdram_dq		
597	BOTTOM	vss_gcs_sstl_b0	pnl_sstl_gcs	SSTL
598	BOTTOM	sdram_iopad0_DQM_8__dqm		SSTL
	output	sdram_dqm		
599	BOTTOM	sdram_iopad0_DQS_8__dqs		SSTL
	inout	sdram_dqs		
600	BOTTOM	sstl_vref_b0	pnl_sstl_vref	SSTL
601	BOTTOM	sdram_iopad0_CS__1__cs_		SSTL
	output	sdram_cs_		
602	BOTTOM	sdram_iopad0_CS__0__cs_		SSTL
	output	sdram_cs_		
CORNER	RIGHT	CORNER_BR	pnl_iocrnr_hs	SSTL Corner
603	RIGHT	sdram_iopad0_cas_		SSTL
	output	sdram_cas_		
604	RIGHT	sdram_iopad0_we_		SSTL
	output	sdram_we		
605	RIGHT	vss_go_sstl_r0	pnl_sstl_go	SSTL
606	RIGHT	sdram_iopad0_ras_		SSTL
	output	sdram_ras_		
607	RIGHT	vdd_vq_sstl_r0	pnl_sstl_vq	SSTL

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
608	RIGHT	sdram_iopad0_BA_0__ba		SSTL
	output	sdram_ba		
609	RIGHT	vdd_vp_sstl_r0	pnl_sstl_vp	SSTL
610	RIGHT	sdram_iopad0_BA_1__ba		SSTL
	output	sdram_ba		
611	RIGHT	sdram_iopad0_cke		SSTL
	output	sdram_cke		
612	RIGHT	vss_gcs_sstl_r0	pnl_sstl_gcs	SSTL
613	RIGHT	sdram_iopad0_ADDR_0__addr		SSTL
	output	sdram_a		
614	RIGHT	vdd_vc_sstl_r0	pnl_sstl_vc	SSTL
615	RIGHT	sdram_iopad0_ADDR_1__addr		SSTL
	output	sdram_a		
616	RIGHT	vss_gcs_sstl_r1	pnl_sstl_gcs	SSTL
617	RIGHT	sdram_iopad0_ADDR_2__addr		SSTL
	output	sdram_a		
618	RIGHT	sdram_iopad0_ADDR_3__addr		SSTL
	output	sdram_a		
619	RIGHT	vss_go_sstl_r1	pnl_sstl_go	SSTL
620	RIGHT	sdram_iopad0_ADDR_4__addr		SSTL
	output	sdram_a		
621	RIGHT	vss_gcs_sstl_r2	pnl_sstl_gcs	SSTL
622	RIGHT	sdram_iopad0_ADDR_5__addr		SSTL
	output	sdram_a		
623	RIGHT	vdd_vq_sstl_r1	pnl_sstl_vq	SSTL
624	RIGHT	sdram_iopad0_ADDR_6__addr		SSTL
	output	sdram_a		
625	RIGHT	sdram_iopad0_ADDR_7__addr		SSTL
	output	sdram_a		
626	RIGHT	vss_gcs_sstl_r3	pnl_sstl_gcs	SSTL
627	RIGHT	sdram_iopad0_ADDR_8__addr		SSTL
	output	sdram_a		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
628	RIGHT	vdd_vc_sstl_r1	pnl_sstl_vc	SSTL
629	RIGHT	sdram_iopad0_ADDR_9__addr		SSTL
	output	sdram_a		
630	RIGHT	vss_gcs_sstl_r4	pnl_sstl_gcs	SSTL
631	RIGHT	sdram_iopad0_ADDR_10__addr		SSTL
	output	sdram_a		
632	RIGHT	vss_go_sstl_r2	pnl_sstl_go	SSTL
633	RIGHT	sdram_iopad0_ADDR_11__addr		SSTL
	output	sdram_a		
634	RIGHT	vdd_vc_sstl_r2	pnl_sstl_vc	SSTL
635	RIGHT	sdram_iopad0_ADDR_12__addr		SSTL
	output	sdram_a		
636	RIGHT	vdd_vq_sstl_r2	pnl_sstl_vq	SSTL
637	RIGHT	sdram_iopad0_DQ_63__dq		SSTL
	inout	sdram_dq		
638	RIGHT	vdd_vp_sstl_r1	pnl_sstl_vp	SSTL
639	RIGHT	sdram_iopad0_DQ_62__dq		SSTL
	inout	sdram_dq		
640	RIGHT	vss_gcs_sstl_r5	pnl_sstl_gcs	SSTL
641	RIGHT	sdram_iopad0_DQ_61__dq		SSTL
	inout	sdram_dq		
642	RIGHT	sdram_iopad0_DQ_60__dq		SSTL
	inout	sdram_dq		
643	RIGHT	vdd_vc_sstl_r3	pnl_sstl_vc	SSTL
644	RIGHT	sdram_iopad0_DQ_59__dq		SSTL
	inout	sdram_dq		
645	RIGHT	vss_gcs_sstl_r6	pnl_sstl_gcs	SSTL
646	RIGHT	sdram_iopad0_DQ_58__dq		SSTL
	inout	sdram_dq		
647	RIGHT	vss_go_sstl_r3	pnl_sstl_go	SSTL

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
648	RIGHT	sdram_iopad0_DQ_57__dq		SSTL
	inout	sdram_dq		
649	RIGHT	vdd_vc_sstl_r4	pnl_sstl_vc	SSTL
650	RIGHT	sdram_iopad0_DQ_56__dq		SSTL
	inout	sdram_dq		
651	RIGHT	vdd_vq_sstl_r3	pnl_sstl_vq	SSTL
652	RIGHT	sdram_iopad0_DQM_7__dqm		SSTL
	output	sdram_dqm		
653	RIGHT	sdram_iopad0_DQS_7__dqs		SSTL
	inout	sdram_dqs		
654	RIGHT	sdram_iopad0_DQ_55__dq		SSTL
	inout	sdram_dq		
655	RIGHT	sdram_iopad0_DQ_54__dq		SSTL
	inout	sdram_dq		
656	RIGHT	vss_gcs_sstl_r7	pnl_sstl_gcs	SSTL
657	RIGHT	sdram_iopad0_DQ_53__dq		SSTL
	inout	sdram_dq		
658	RIGHT	vdd_vc_sstl_r5	pnl_sstl_vc	SSTL
659	RIGHT	sdram_iopad0_DQ_52__dq		SSTL
	inout	sdram_dq		
660	RIGHT	vss_gcs_sstl_r8	pnl_sstl_gcs	SSTL
661	RIGHT	sdram_iopad0_DQ_51__dq		SSTL
	inout	sdram_dq		
662	RIGHT	vss_go_sstl_r4	pnl_sstl_go	SSTL
663	RIGHT	sdram_iopad0_DQ_50__dq		SSTL
	inout	sdram_dq		
664	RIGHT	vss_gcs_sstl_r9	pnl_sstl_gcs	SSTL
665	RIGHT	sdram_iopad0_DQ_49__dq		SSTL
	inout	sdram_dq		
666	RIGHT	sdram_iopad0_DQ_48__dq		SSTL
	inout	sdram_dq		
667	RIGHT	vdd_vq_sstl_r4	pnl_sstl_vq	SSTL

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
668	RIGHT	sdram_iopad0_DQM_6__dqm		SSTL
	output	sdram_dqm		
669	RIGHT	vdd_vp_sstl_r2	pnl_sstl_vp	SSTL
670	RIGHT	sdram_iopad0_DQS_6__dqs		SSTL
	inout	sdram_dqs		
671	RIGHT	sdram_iopad0_DQ_47__dq		SSTL
	inout	sdram_dq		
672	RIGHT	vss_gcs_sstl_r10	pnl_sstl_gcs	SSTL
673	RIGHT	sdram_iopad0_DQ_46__dq		SSTL
	inout	sdram_dq		
674	RIGHT	vss_go_sstl_r5	pnl_sstl_go	SSTL
675	RIGHT	sdram_iopad0_DQ_45__dq		SSTL
	inout	sdram_dq		
676	RIGHT	vdd_vc_sstl_r6	pnl_sstl_vc	SSTL
677	RIGHT	sdram_iopad0_DQ_44__dq		SSTL
	inout	sdram_dq		
678	RIGHT	vdd_vq_sstl_r5	pnl_sstl_vq	SSTL
679	RIGHT	sdram_iopad0_DQ_43__dq		SSTL
	inout	sdram_dq		
680	RIGHT	vss_gcs_sstl_r11	pnl_sstl_gcs	SSTL
681	RIGHT	sdram_iopad0_DQ_42__dq		SSTL
	inout	sdram_dq		
682	RIGHT	sdram_iopad0_DQ_41__dq		SSTL
	inout	sdram_dq		
683	RIGHT	vdd_vc_sstl_r7	pnl_sstl_vc	SSTL
684	RIGHT	sdram_iopad0_DQ_40__dq		SSTL
	inout	sdram_dq		
685	RIGHT	sdram_iopad0_DQM_5__dqm		SSTL
	output	sdram_dqm		
686	RIGHT	vss_go_sstl_r6	pnl_sstl_go	SSTL
687	RIGHT	sdram_iopad0_DQS_5__dqs		SSTL
	inout	sdram_dqs		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
688	RIGHT	vdd_vq_sstl_r6	pnl_sstl_vq	SSTL
689	RIGHT	sdram_iopad0_DQ_39__dq		SSTL
	inout	sdram_dq		
690	RIGHT	sdram_iopad0_DQ_38__dq		SSTL
	inout	sdram_dq		
691	RIGHT	sdram_iopad0_DQ_37__dq		SSTL
	inout	sdram_dq		
692	RIGHT	vss_gcs_sstl_r12	pnl_sstl_gcs	SSTL
693	RIGHT	sdram_iopad0_CLK_1__clk		SSTL
	output	sdram_clk		
694	RIGHT	vss_go_sstl_r7	pnl_sstl_go	SSTL
695	RIGHT	sdram_iopad0_CLK_1__clk_		SSTL
	output	sdram_clk_		
696	RIGHT	vss_gcs_sstl_r13	pnl_sstl_gcs	SSTL
697	RIGHT	sdram_iopad0_DQ_36__dq		SSTL
	inout	sdram_dq		
698	RIGHT	vdd_vq_sstl_r7	pnl_sstl_vq	SSTL
699	RIGHT	sdram_iopad0_DQ_35__dq		SSTL
	inout	sdram_dq		
700	RIGHT	vdd_vp_sstl_r3	pnl_sstl_vp	SSTL
701	RIGHT	sdram_iopad0_DQ_34__dq		SSTL
	inout	sdram_dq		
702	RIGHT	vss_gcs_sstl_r14	pnl_sstl_gcs	SSTL
703	RIGHT	sdram_iopad0_DQ_33__dq		SSTL
	inout	sdram_dq		
704	RIGHT	vss_go_sstl_r8	pnl_sstl_go	SSTL
705	RIGHT	sdram_iopad0_DQ_32__dq		SSTL
	inout	sdram_dq		
706	RIGHT	sdram_iopad0_DQM_4__dqm		SSTL
	output	sdram_dqm		
707	RIGHT	sstl_vref_r0	pnl_sstl_vref	SSTL

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
708	RIGHT	sdram_iopad0_DQS_4__dqs		SSTL
	inout	sdram_dqs		
709	RIGHT	sdram_iopad0_DQ_31__dq		SSTL
	inout	sdram_dq		
710	RIGHT	vdd_vq_sstl_r8	pnl_sstl_vq	SSTL
711	RIGHT	sdram_iopad0_DQ_30__dq		SSTL
	inout	sdram_dq		
712	RIGHT	vss_gcs_sstl_r15	pnl_sstl_gcs	SSTL
713	RIGHT	sdram_iopad0_CLK_0__clk		SSTL
	output	sdram_clk		
714	RIGHT	vss_go_sstl_r9	pnl_sstl_go	SSTL
715	RIGHT	sdram_iopad0_CLK_0__clk_		SSTL
	output	sdram_clk_		
716	RIGHT	vss_gcs_sstl_r16	pnl_sstl_gcs	SSTL
717	RIGHT	sdram_iopad0_DQ_29__dq		SSTL
	inout	sdram_dq		
718	RIGHT	sdram_iopad0_DQ_28__dq		SSTL
	inout	sdram_dq		
719	RIGHT	vdd_vq_sstl_r9	pnl_sstl_vq	SSTL
720	RIGHT	sdram_iopad0_DQ_27__dq		SSTL
	inout	sdram_dq		
721	RIGHT	vdd_vp_sstl_r4	pnl_sstl_vp	SSTL
722	RIGHT	sdram_iopad0_DQ_26__dq		SSTL
	inout	sdram_dq		
723	RIGHT	vdd_vc_sstl_r8	pnl_sstl_vc	SSTL
724	RIGHT	sdram_iopad0_DQ_25__dq		SSTL
	inout	sdram_dq		
725	RIGHT	sdram_iopad0_DQ_24__dq		SSTL
	inout	sdram_dq		
726	RIGHT	vss_gcs_sstl_r17	pnl_sstl_gcs	SSTL
727	RIGHT	sdram_iopad0_DQM_3__dqm		SSTL
	output	sdram_dqm		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
728	RIGHT	vss_go_sstl_r10	pnl_sstl_go	SSTL
729	RIGHT	sdram_iopad0_DQS_3__dqs		SSTL
	inout	sdram_dqs		
730	RIGHT	vss_gcs_sstl_r18	pnl_sstl_gcs	SSTL
731	RIGHT	sdram_iopad0_DQ_23__dq		SSTL
	inout	sdram_dq		
732	RIGHT	sdram_iopad0_DQ_22__dq		SSTL
	inout	sdram_dq		
733	RIGHT	sdram_iopad0_DQ_21__dq		SSTL
	inout	sdram_dq		
734	RIGHT	sdram_iopad0_DQ_20__dq		SSTL
	inout	sdram_dq		
735	RIGHT	vdd_vq_sstl_r10	pnl_sstl_vq	SSTL
736	RIGHT	sdram_iopad0_DQ_19__dq		SSTL
	inout	sdram_dq		
737	RIGHT	vss_gcs_sstl_r19	pnl_sstl_gcs	SSTL
738	RIGHT	sdram_iopad0_DQ_18__dq		SSTL
	inout	sdram_dq		
739	RIGHT	vdd_vc_sstl_r9	pnl_sstl_vc	SSTL
740	RIGHT	sdram_iopad0_DQ_17__dq		SSTL
	inout	sdram_dq		
741	RIGHT	vss_gcs_sstl_r20	pnl_sstl_gcs	SSTL
742	RIGHT	sdram_iopad0_DQ_16__dq		SSTL
	inout	sdram_dq		
743	RIGHT	vdd_vc_sstl_r10	pnl_sstl_vc	SSTL
744	RIGHT	sdram_iopad0_DQM_2__dqm		SSTL
	output	sdram_dqm		
745	RIGHT	vdd_vp_sstl_r5	pnl_sstl_vp	SSTL
746	RIGHT	sdram_iopad0_DQS_2__dqs		SSTL
	inout	sdram_dqs		
747	RIGHT	vss_go_sstl_r11	pnl_sstl_go	SSTL

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
748	RIGHT	sdram_iopad0_DQ_15__dq		SSTL
	inout	sdram_dq		
749	RIGHT	sdram_iopad0_DQ_14__dq		SSTL
	inout	sdram_dq		
750	RIGHT	vss_gcs_sstl_r21	pnl_sstl_gcs	SSTL
751	RIGHT	sdram_iopad0_DQ_13__dq		SSTL
	inout	sdram_dq		
752	RIGHT	vdd_vq_sstl_r11	pnl_sstl_vq	SSTL
753	RIGHT	sdram_iopad0_DQ_12__dq		SSTL
	inout	sdram_dq		
754	RIGHT	vss_gcs_sstl_r22	pnl_sstl_gcs	SSTL
755	RIGHT	sdram_iopad0_DQ_11__dq		SSTL
	inout	sdram_dq		
756	RIGHT	vdd_vc_sstl_r11	pnl_sstl_vc	SSTL
757	RIGHT	sdram_iopad0_DQ_10__dq		SSTL
	inout	sdram_dq		
758	RIGHT	vss_gcs_sstl_r23	pnl_sstl_gcs	SSTL
759	RIGHT	sdram_iopad0_DQ_9__dq		SSTL
	inout	sdram_dq		
760	RIGHT	sdram_iopad0_DQ_8__dq		SSTL
	inout	sdram_dq		
761	RIGHT	sdram_iopad0_DQM_1__dqm		SSTL
	output	sdram_dqm		
762	RIGHT	sdram_iopad0_DQS_1__dqs		SSTL
	inout	sdram_dqs		
763	RIGHT	vdd_vc_sstl_r12	pnl_sstl_vc	SSTL
764	RIGHT	sdram_iopad0_DQ_7__dq		SSTL
	inout	sdram_dq		
765	RIGHT	vss_gcs_sstl_r24	pnl_sstl_gcs	SSTL
766	RIGHT	sdram_iopad0_DQ_6__dq		SSTL
	inout	sdram_dq		
767	RIGHT	vss_go_sstl_r12	pnl_sstl_go	SSTL

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
768	RIGHT	sdram_iopad0_DQ_5__dq		SSTL
	inout	sdram_dq		
769	RIGHT	vdd_vc_sstl_r13	pnl_sstl_vc	SSTL
770	RIGHT	sdram_iopad0_DQ_4__dq		SSTL
	inout	sdram_dq		
771	RIGHT	vdd_vq_sstl_r12	pnl_sstl_vq	SSTL
772	RIGHT	sdram_iopad0_DQ_3__dq		SSTL
	inout	sdram_dq		
773	RIGHT	sdram_iopad0_DQ_2__dq		SSTL
	inout	sdram_dq		
774	RIGHT	vdd_vc_sstl_r14	pnl_sstl_vc	SSTL
775	RIGHT	sdram_iopad0_DQ_1__dq		SSTL
	inout	sdram_dq		
776	RIGHT	vdd_vp_sstl_r6	pnl_sstl_vp	SSTL
777	RIGHT	sdram_iopad0_DQ_0__dq		SSTL
	inout	sdram_dq		
778	RIGHT	sdram_iopad0_DQM_0__dqm		SSTL
	output	sdram_dqm		
779	RIGHT	vss_gcs_sstl_r25	pnl_sstl_gcs	SSTL
780	RIGHT	sdram_iopad0_DQS_0__dqs		SSTL
	inout	sdram_dqs		
781	RIGHT	vss_go_sstl_r13	pnl_sstl_go	SSTL
782	RIGHT	link_sdram_iopad0_dq47		SSTL
	inout	sdram32_dq		
783	RIGHT	vdd_vc_sstl_r15	pnl_sstl_vc	SSTL
784	RIGHT	link_sdram_iopad0_dq46		SSTL
	inout	sdram32_dq		
785	RIGHT	vdd_vq_sstl_r13	pnl_sstl_vq	SSTL
786	RIGHT	link_sdram_iopad0_dq45		SSTL
	inout	sdram32_dq		
787	RIGHT	vss_gcs_sstl_r26	pnl_sstl_gcs	SSTL

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
788	RIGHT	link_sdram_iopad0_dq44		SSTL
	inout	sdram32_dq		
789	RIGHT	link_sdram_iopad0_dq43		SSTL
	inout	sdram32_dq		
790	RIGHT	vss_go_sstl_r14	pnl_sstl_go	SSTL
791	RIGHT	link_sdram_iopad0_dq42		SSTL
	inout	sdram32_dq		
792	RIGHT	link_sdram_iopad0_dq41		SSTL
	inout	sdram32_dq		
793	RIGHT	vdd_vc_sstl_r16	pnl_sstl_vc	SSTL
794	RIGHT	link_sdram_iopad0_dq40		SSTL
	inout	sdram32_dq		
795	RIGHT	vdd_vq_sstl_r14	pnl_sstl_vq	SSTL
796	RIGHT	link_sdram_iopad0_dq39		SSTL
	inout	sdram32_dq		
797	RIGHT	link_sdram_iopad0_dq38		SSTL
	inout	sdram32_dq		
798	RIGHT	link_sdram_iopad0_dq37		SSTL
	inout	sdram32_dq		
799	RIGHT	vss_gcs_sstl_r27	pnl_sstl_gcs	SSTL
800	RIGHT	link_sdram_iopad0_dq36		SSTL
	inout	sdram32_dq		
801	RIGHT	vdd_vp_sstl_r7	pnl_sstl_vp	SSTL
802	RIGHT	link_sdram_iopad0_dq35		SSTL
	inout	sdram32_dq		
803	RIGHT	vss_gcs_sstl_r28	pnl_sstl_gcs	SSTL
804	RIGHT	link_sdram_iopad0_dq34		SSTL
	inout	sdram32_dq		
805	RIGHT	vss_go_sstl_r15	pnl_sstl_go	SSTL
806	RIGHT	link_sdram_iopad0_dq33		SSTL
	inout	sdram32_dq		
807	RIGHT	vdd_vc_sstl_r17	pnl_sstl_vc	SSTL

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
808	RIGHT	link_sdram_iopad0_dq32		SSTL
	inout	sdram32_dq		
809	RIGHT	vdd_vq_sstl_r15	pnl_sstl_vq	SSTL
810	RIGHT	link_sdram_iopad0_dqm5		SSTL
	output	sdram32_dqm		
811	RIGHT	vss_gcs_sstl_r29	pnl_sstl_gcs	SSTL
812	RIGHT	link_sdram_iopad0_dqm4		SSTL
	output	sdram32_dqm		
813	RIGHT	link_sdram_iopad0_dqs5		SSTL
	inout	sdram32_dqs		
814	RIGHT	sstl_vref_r1	pnl_sstl_vref	SSTL
815	RIGHT	link_sdram_iopad0_dqs4		SSTL
	inout	sdram32_dqs		
816	RIGHT	link_sdram_iopad0_oe		SSTL
	output	sdram32_oe_		
817	RIGHT	vss_gcs_sstl_r30	pnl_sstl_gcs	SSTL
818	RIGHT	link_sdram_iopad0_dir		SSTL
	output	sdram32_dir_		
819	RIGHT	vdd_vc_sstl_r18	pnl_sstl_vc	SSTL
820	RIGHT	link_sdram_iopad0_addr12		SSTL
	output	sdram32_a		
821	RIGHT	vss_gcs_sstl_r31	pnl_sstl_gcs	SSTL
822	RIGHT	link_sdram_iopad0_clk2_		SSTL
	output	sdram32_clk2_		
823	RIGHT	vss_go_sstl_r16	pnl_sstl_go	SSTL
824	RIGHT	link_sdram_iopad0_clk2		SSTL
	output	sdram32_clk2		
825	RIGHT	vss_gcs_sstl_r32	pnl_sstl_gcs	SSTL
826	RIGHT	link_sdram_iopad0_addr11		SSTL
	output	sdram32_a		
827	RIGHT	vdd_vq_sstl_r16	pnl_sstl_vq	SSTL

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
828	RIGHT	link_sdram_iopad0_addr10		SSTL
	output	sdram32_a		
829	RIGHT	link_sdram_iopad0_addr09		SSTL
	output	sdram32_a		
830	RIGHT	vdd_vp_sstl_r8	pnl_sstl_vp	SSTL
831	RIGHT	link_sdram_iopad0_addr08		SSTL
	output	sdram32_a		
832	RIGHT	link_sdram_iopad0_addr07		SSTL
	output	sdram32_a		
833	RIGHT	vss_gcs_sstl_r33	pnl_sstl_gcs	SSTL
834	RIGHT	link_sdram_iopad0_addr06		SSTL
	output	sdram32_a		
835	RIGHT	vdd_vc_sstl_r19	pnl_sstl_vc	SSTL
836	RIGHT	link_sdram_iopad0_addr05		SSTL
	output	sdram32_a		
837	RIGHT	vss_gcs_sstl_r34	pnl_sstl_gcs	SSTL
838	RIGHT	link_sdram_iopad0_addr04		SSTL
	output	sdram32_a		
839	RIGHT	link_sdram_iopad0_addr03		SSTL
	output	sdram32_a		
840	RIGHT	link_sdram_iopad0_addr02		SSTL
	output	sdram32_a		
841	RIGHT	link_sdram_iopad0_addr01		SSTL
	output	sdram32_a		
842	RIGHT	vss_go_sstl_r17	pnl_sstl_go	SSTL
843	RIGHT	link_sdram_iopad0_addr00		SSTL
	output	sdram32_a		
844	RIGHT	vdd_vc_sstl_r20	pnl_sstl_vc	SSTL
845	RIGHT	link_sdram_iopad0_bank0		SSTL
	output	sdram32_ba		
846	RIGHT	vdd_vq_sstl_r17	pnl_sstl_vq	SSTL
847	RIGHT	link_sdram_iopad0_bank1		SSTL
	output	sdram32_ba		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
848	RIGHT	vss_gcs_sstl_r35	pnl_sstl_gcs	SSTL
849	RIGHT	link_sdram_iopad0_cs1		SSTL
	output	sdram32_cs_		
850	RIGHT	vdd_vc_sstl_r21	pnl_sstl_vc	SSTL
851	RIGHT	link_sdram_iopad0_cs0		SSTL
	output	sdram32_cs_		
852	RIGHT	vdd_vp_sstl_r9	pnl_sstl_vp	SSTL
853	RIGHT	link_sdram_iopad0_cke		SSTL
	output	sdram32_cke		
854	RIGHT	vss_gcs_sstl_r36	pnl_sstl_gcs	SSTL
855	RIGHT	link_sdram_iopad0_ras		SSTL
	output	sdram32_ras_		
856	RIGHT	link_sdram_iopad0_cas		SSTL
	output	sdram32_cas_		
857	RIGHT	vdd_vc_sstl_r22	pnl_sstl_vc	SSTL
858	RIGHT	link_sdram_iopad0_we		SSTL
	output	sdram32_we_		
859	RIGHT	vss_gcs_sstl_r37	pnl_sstl_gcs	SSTL
860	RIGHT	link_sdram_iopad0_dqm3		SSTL
	output	sdram32_dqm		
861	RIGHT	vss_go_sstl_r18	pnl_sstl_go	SSTL
862	RIGHT	link_sdram_iopad0_dqm1		SSTL
	output	sdram32_dqm		
863	RIGHT	vdd_vc_sstl_r23	pnl_sstl_vc	SSTL
864	RIGHT	link_sdram_iopad0_dqm2		SSTL
	output	sdram32_dqm		
865	RIGHT	vdd_vq_sstl_r18	pnl_sstl_vq	SSTL
866	RIGHT	link_sdram_iopad0_dqm0		SSTL
	output	sdram32_dqm		
867	RIGHT	vss_gcs_sstl_r38	pnl_sstl_gcs	SSTL

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
868	RIGHT	link_sdram_iopad0_dqs3		SSTL
	inout	sdram32_dqs		
869	RIGHT	link_sdram_iopad0_dqs2		SSTL
	inout	sdram32_dqs		
870	RIGHT	vss_go_sstl_r19	pnl_sstl_go	SSTL
871	RIGHT	link_sdram_iopad0_clk		SSTL
	output	sdram32_clk		
872	RIGHT	vss_gcs_sstl_r39	pnl_sstl_gcs	SSTL
873	RIGHT	link_sdram_iopad0_clk_		SSTL
	output	sdram32_clk_		
874	RIGHT	vss_gcs_sstl_r40	pnl_sstl_gcs	SSTL
875	RIGHT	link_sdram_iopad0_dqs1		SSTL
	inout	sdram32_dqs		
876	RIGHT	vdd_vq_sstl_r19	pnl_sstl_vq	SSTL
877	RIGHT	link_sdram_iopad0_dqs0		SSTL
	inout	sdram32_dqs		
878	RIGHT	vss_go_sstl_r20	pnl_sstl_go	SSTL
879	RIGHT	link_sdram_iopad0_dq31		SSTL
	inout	sdram32_dq		
880	RIGHT	link_sdram_iopad0_dq30		SSTL
	inout	sdram32_dq		
881	RIGHT	vdd_vq_sstl_r20	pnl_sstl_vq	SSTL
882	RIGHT	link_sdram_iopad0_dq29		SSTL
	inout	sdram32_dq		
883	RIGHT	vdd_vp_sstl_r10	pnl_sstl_vp	SSTL
884	RIGHT	link_sdram_iopad0_dq28		SSTL
	inout	sdram32_dq		
885	RIGHT	link_sdram_iopad0_dq27		SSTL
	inout	sdram32_dq		
886	RIGHT	vss_gcs_sstl_r41	pnl_sstl_gcs	SSTL
887	RIGHT	link_sdram_iopad0_dq26		SSTL
	inout	sdram32_dq		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
888	RIGHT	vss_go_sstl_r21	pnl_sstl_go	SSTL
889	RIGHT	link_sdram_iopad0_dq25		SSTL
	inout	sdram32_dq		
890	RIGHT	vdd_vc_sstl_r24	pnl_sstl_vc	SSTL
891	RIGHT	link_sdram_iopad0_dq24		SSTL
	inout	sdram32_dq		
892	RIGHT	vdd_vq_sstl_r21	pnl_sstl_vq	SSTL
893	RIGHT	link_sdram_iopad0_dq23		SSTL
	inout	sdram32_dq		
894	RIGHT	vss_gcs_sstl_r42	pnl_sstl_gcs	SSTL
895	RIGHT	link_sdram_iopad0_dq22		SSTL
	inout	sdram32_dq		
896	RIGHT	link_sdram_iopad0_dq21		SSTL
	inout	sdram32_dq		
897	RIGHT	vss_go_sstl_r22	pnl_sstl_go	SSTL
898	RIGHT	link_sdram_iopad0_dq20		SSTL
	inout	sdram32_dq		
899	RIGHT	link_sdram_iopad0_dq19		SSTL
	inout	sdram32_dq		
900	RIGHT	pnl_filler_sstl_4g_r0	pnl_filler_sstl_4g	FILLER
901	RIGHT	pnl_filler_sstl_2g_r0	pnl_filler_sstl_2g	FILLER
CORNER	TOP	CORNER_TR	pnl_iocrnr_hs	SSTL Corner
902	TOP	pnl_filler_sstl_8g_t0	pnl_filler_sstl_8g	FILLER
903	TOP	pnl_filler_sstl_4g_t0	pnl_filler_sstl_4g	FILLER
904	TOP	pnl_filler_sstl_1g_t0	pnl_filler_sstl_1g	FILLER
905	TOP	vss_gcs_sstl_t5	pnl_sstl_gcs	SSTL
906	TOP	link_sdram_iopad0_dq18		SSTL
	inout	sdram32_dq		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
907	TOP	vdd_vc_sstl_t2	pnl_sstl_vc	SSTL
908	TOP	link_sdram_iopad0_dq17		SSTL
	inout	sdram32_dq		
909	TOP	link_sdram_iopad0_dq16		SSTL
	inout	sdram32_dq		
910	TOP	link_sdram_iopad0_dq15		SSTL
	inout	sdram32_dq		
911	TOP	vss_gcs_sstl_t4	pnl_sstl_gcs	SSTL
912	TOP	link_sdram_iopad0_dq14		SSTL
	inout	sdram32_dq		
913	TOP	vdd_vp_sstl_t0	pnl_sstl_vp	SSTL
914	TOP	link_sdram_iopad0_dq13		SSTL
	inout	sdram32_dq		
915	TOP	vss_gcs_sstl_t3	pnl_sstl_gcs	SSTL
916	TOP	link_sdram_iopad0_dq12		SSTL
	inout	sdram32_dq		
917	TOP	vdd_vc_sstl_t1	pnl_sstl_vc	SSTL
918	TOP	link_sdram_iopad0_dq11		SSTL
	inout	sdram32_dq		
919	TOP	vdd_vq_sstl_t1	pnl_sstl_vq	SSTL
920	TOP	link_sdram_iopad0_dq10		SSTL
	inout	sdram32_dq		
921	TOP	vss_gcs_sstl_t2	pnl_sstl_gcs	SSTL
922	TOP	link_sdram_iopad0_dq09		SSTL
	inout	sdram32_dq		
923	TOP	vss_go_sstl_t1	pnl_sstl_go	SSTL
924	TOP	link_sdram_iopad0_dq08		SSTL
	inout	sdram32_dq		
925	TOP	link_sdram_iopad0_dq07		SSTL
	inout	sdram32_dq		
926	TOP	sstl_vref_t0	pnl_sstl_vref	SSTL

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
927	TOP	link_sdram_iopad0_dq06		SSTL
	inout	sdram32_dq		
928	TOP	link_sdram_iopad0_dq05		SSTL
	inout	sdram32_dq		
929	TOP	vss_gcs_sstl_t1	pnl_sstl_gcs	SSTL
930	TOP	link_sdram_iopad0_dq04		SSTL
	inout	sdram32_dq		
931	TOP	vdd_vc_sstl_t0	pnl_sstl_vc	SSTL
932	TOP	link_sdram_iopad0_dq03		SSTL
	inout	sdram32_dq		
933	TOP	vdd_vq_sstl_t0	pnl_sstl_vq	SSTL
934	TOP	link_sdram_iopad0_dq02		SSTL
	inout	sdram32_dq		
935	TOP	vss_gcs_sstl_t0	pnl_sstl_gcs	SSTL
936	TOP	link_sdram_iopad0_dq01		SSTL
	inout	sdram32_dq		
937	TOP	link_sdram_iopad0_dq00		SSTL
	inout	sdram32_dq		
938	TOP	vss_go_sstl_t0	pnl_sstl_go	SSTL
939	TOP	pnl_filler_std2sstl_t0	pnl_filler_std2sstl	SSTL Breaker
940	TOP	pnl_filler_lvds_brk_t0	pnl_filler_lvds_brk_3g	LVDS Breaker
941	TOP	lvds_vref_t5	pnl_vref_lvds	LVDS
942	TOP	vss_go_lvds_t10	pnl_go_lvds	LVDS
943	TOP	link_iopad0_data_s_out_iopad1		LVDS
	output	data_s_out		
944	TOP	vdd_vop_lvds_t11	pnl_vop_lvds	LVDS
945	TOP	link_iopad0_data_p_out1_iopad1		LVDS
	output	data_p_out1		
946	TOP	vss_gcs_lvds_t19	pnl_gcs_lvds	LVDS

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
947	TOP	link_iopad0_data_p_out1_iopad2		LVDS
	output	data_p_out1		
948	TOP	link_iopad0_data_p_out1_iopad3		LVDS
	output	data_p_out1		
949	TOP	vss_gcs_lvds_t18	pnl_gcs_lvds	LVDS
950	TOP	link_iopad0_data_s_in_iopad1		LVDS
	input	data_s_in		
951	TOP	vdd_vop_lvds_t10	pnl_vop_lvds	LVDS
952	TOP	link_iopad0_data_p_in1_iopad1		LVDS
	input	data_p_in1		
953	TOP	link_iopad0_data_p_in1_iopad2		LVDS
	input	data_p_in1		
954	TOP	vdd_vc_lvds_t4	pnl_vc_lvds	LVDS
955	TOP	link_iopad0_data_p_in1_iopad3		LVDS
	input	data_p_in1		
956	TOP	vss_go_lvds_t9	pnl_go_lvds	LVDS
957	TOP	link_iopad0_data_s_out_iopad2		LVDS
	output	data_s_out		
958	TOP	link_iopad0_data_p_out2_iopad1		LVDS
	output	data_p_out2		
959	TOP	vss_gcs_lvds_t17	pnl_gcs_lvds	LVDS
960	TOP	link_iopad0_data_p_out2_iopad2		LVDS
	output	data_p_out2		
961	TOP	link_iopad0_data_p_out2_iopad3		LVDS
	output	data_p_out2		
962	TOP	vss_gcs_lvds_t16	pnl_gcs_lvds	LVDS
963	TOP	link_iopad0_data_s_in_iopad2		LVDS
	input	data_s_in		
964	TOP	link_iopad0_data_p_in2_iopad1		LVDS
	input	data_p_in2		
965	TOP	vdd_vop_lvds_t9	pnl_vop_lvds	LVDS
966	TOP	link_iopad0_data_p_in2_iopad2		LVDS
	input	data_p_in2		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
967	TOP	link_iopad0_data_p_in2_iopad3		LVDS
	input	data_p_in2		
968	TOP	lvds_vref_t4	pnl_lvds_vref	LVDS
969	TOP	vss_gcs_lvds_t15	pnl_gcs_lvds	LVDS
970	TOP	link_iopad0_data_s_out_iopad3		LVDS
	output	data_s_out		
971	TOP	vss_go_lvds_t8	pnl_go_lvds	LVDS
972	TOP	link_iopad0_data_p_out3_iopad1		LVDS
	output	data_p_out3		
973	TOP	vss_gcs_lvds_t14	pnl_gcs_lvds	LVDS
974	TOP	link_iopad0_data_p_out3_iopad2		LVDS
	output	data_p_out3		
975	TOP	link_iopad0_data_p_out3_iopad3		LVDS
	output	data_p_out3		
976	TOP	vss_gcs_lvds_t13	pnl_gcs_lvds	LVDS
977	TOP	link_iopad0_data_s_in_iopad3		LVDS
	input	data_s_in		
978	TOP	vss_go_lvds_t7	pnl_go_lvds	LVDS
979	TOP	link_iopad0_data_p_in3_iopad1		LVDS
	input	data_p_in3		
980	TOP	link_iopad0_data_p_in3_iopad2		LVDS
	input	data_p_in3		
981	TOP	vdd_vc_lvds_t3	pnl_vc_lvds	LVDS
982	TOP	link_iopad0_data_p_in3_iopad3		LVDS
	input	data_p_in3		
983	TOP	vdd_vop_lvds_t8	pnl_vop_lvds	LVDS
984	TOP	link_iopad0_data_s_out_iopad4		LVDS
	output	data_s_out		
985	TOP	link_iopad0_data_p_out4_iopad1		LVDS
	output	data_p_out4		
986	TOP	vss_gcs_lvds_t12	pnl_gcs_lvds	LVDS

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
987	TOP	link_iopad0_data_p_out4_iopad2		LVDS
	output	data_p_out4		
988	TOP	link_iopad0_data_p_out4_iopad3		LVDS
	output	data_p_out4		
989	TOP	vss_gcs_lvds_t11	pnl_gcs_lvds	LVDS
990	TOP	link_iopad0_data_s_in_iopad4		LVDS
	input	data_s_in		
991	TOP	link_iopad0_data_p_in4_iopad1		LVDS
	input	data_p_in4		
992	TOP	vss_go_lvds_t6	pnl_go_lvds	LVDS
993	TOP	link_iopad0_data_p_in4_iopad2		LVDS
	input	data_p_in4		
994	TOP	link_iopad0_data_p_in4_iopad3		LVDS
	input	data_p_in4		
995	TOP	lvds_vref_t3	pnl_lvds_vref	LVDS
996	TOP	vss_gcs_lvds_t10	pnl_gcs_lvds	LVDS
997	TOP	link_iopad0_ext_rl_clk_out_pad1		LVDS
	output	ext_rl_clk_out		
998	TOP	vdd_vop_lvds_t7	pnl_vop_lvds	LVDS
999	TOP	link_iopad0_ext_rl_clk_out_pad2		LVDS
	output	ext_rl_clk_out		
1000	TOP	vss_gcs_lvds_t9	pnl_gcs_lvds	LVDS
1001	TOP	link_iopad0_ext_rl_clk_out_pad3		LVDS
	output	ext_rl_clk_out		
1002	TOP	link_iopad0_ext_rl_clk_out_pad4		LVDS
	output	ext_rl_clk_out		
1003	TOP	vss_gcs_lvds_t8	pnl_gcs_lvds	LVDS
1004	TOP	link_iopad0_ext_rl_clk_in_pad1		LVDS
	input	ext_rl_clk_in		
1005	TOP	vdd_vop_lvds_t6	pnl_vop_lvds	LVDS
1006	TOP	link_iopad0_ext_rl_clk_in_pad2		LVDS
	input	ext_rl_clk_in		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
1007	TOP	link_iopad0_ext_rl_clk_in_pad3		LVDS
	input	ext_rl_clk_in		
1008	TOP	vdd_vc_lvds_t2	pnl_vc_lvds	LVDS
1009	TOP	link_iopad0_ext_rl_clk_in_pad4		LVDS
	input	ext_rl_clk_in		
1010	TOP	vss_go_lvds_t5	pnl_go_lvds	LVDS
1011	TOP	link_iopad0_event_s_out_iopad1		LVDS
	output	event_s_out		
1012	TOP	link_iopad0_event_p_out1_iopad1		LVDS
	output	event_p_out1		
1013	TOP	vss_gcs_lvds_t7	pnl_gcs_lvds	LVDS
1014	TOP	link_iopad0_event_p_out1_iopad2		LVDS
	output	event_p_out1		
1015	TOP	link_iopad0_event_p_out1_iopad3		LVDS
	output	event_p_out1		
1016	TOP	vss_gcs_lvds_t6	pnl_gcs_lvds	LVDS
1017	TOP	link_iopad0_event_s_in_iopad1		LVDS
	input	event_s_in		
1018	TOP	link_iopad0_event_p_in1_iopad1		LVDS
	input	event_p_in1		
1019	TOP	vdd_vop_lvds_t5	pnl_vop_lvds	LVDS
1020	TOP	link_iopad0_event_p_in1_iopad2		LVDS
	input	event_p_in1		
1021	TOP	link_iopad0_event_p_in1_iopad3		LVDS
	input	event_p_in1		
1022	TOP	lvds_vref_t2	pnl_lvds_vref	LVDS
1023	TOP	vss_go_lvds_t4	pnl_go_lvds	LVDS
1024	TOP	link_iopad0_event_s_out_iopad2		LVDS
	output	event_s_out		
1025	TOP	vdd_vop_lvds_t4	pnl_vop_lvds	LVDS
1026	TOP	link_iopad0_event_p_out2_iopad1		LVDS
	output	event_p_out2		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
1027	TOP	vss_gcs_lvds_t5	pnl_gcs_lvds	LVDS
1028	TOP	link_iopad0_event_p_out2_iopad2		LVDS
	output	event_p_out2		
1029	TOP	link_iopad0_event_p_out2_iopad3		LVDS
	output	event_p_out2		
1030	TOP	vss_gcs_lvds_t4	pnl_gcs_lvds	LVDS
1031	TOP	link_iopad0_event_s_in_iopad2		LVDS
	input	event_s_in		
1032	TOP	vdd_vop_lvds_t3	pnl_vop_lvds	LVDS
1033	TOP	link_iopad0_event_p_in2_iopad1		LVDS
	input	event_p_in2		
1034	TOP	link_iopad0_event_p_in2_iopad2		LVDS
	input	event_p_in2		
1035	TOP	vdd_vc_lvds_t1	pnl_vc_lvds	LVDS
1036	TOP	link_iopad0_event_p_in2_iopad3		LVDS
	input	event_p_in2		
1037	TOP	vss_go_lvds_t3	pnl_go_lvds	LVDS
1038	TOP	link_iopad0_event_s_out_iopad3		LVDS
	output	event_s_out		
1039	TOP	link_iopad0_event_p_out3_iopad1		LVDS
	output	event_p_out3		
1040	TOP	vss_gcs_lvds_t3	pnl_gcs_lvds	LVDS
1041	TOP	link_iopad0_event_p_out3_iopad2		LVDS
	output	event_p_out3		
1042	TOP	link_iopad0_event_p_out3_iopad3		LVDS
	output	event_p_out3		
1043	TOP	vss_gcs_lvds_t2	pnl_gcs_lvds	LVDS
1044	TOP	link_iopad0_event_s_in_iopad3		LVDS
	input	event_s_in		
1045	TOP	link_iopad0_event_p_in3_iopad1		LVDS
	input	event_p_in3		
1046	TOP	vdd_vop_lvds_t2	pnl_vop_lvds	LVDS

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
1047	TOP	link_iopad0_event_p_in3_iopad2		LVDS
	input	event_p_in3		
1048	TOP	link_iopad0_event_p_in3_iopad3		LVDS
	input	event_p_in3		
1049	TOP	lvds_vref_t1	pnl_lvds_vref	LVDS
1050	TOP	vss_go_lvds_t2	pnl_go_lvds	LVDS
1051	TOP	link_iopad0_event_s_out_iopad4		LVDS
	output	event_s_out		
1052	TOP	vdd_vop_lvds_t1	pnl_vop_lvds	LVDS
1053	TOP	link_iopad0_event_p_out4_iopad1		LVDS
	output	event_p_out4		
1054	TOP	vss_gcs_lvds_t1	pnl_gcs_lvds	LVDS
1055	TOP	link_iopad0_event_p_out4_iopad2		LVDS
	output	event_p_out4		
1056	TOP	link_iopad0_event_p_out4_iopad3		LVDS
	output	event_p_out4		
1057	TOP	vss_gcs_lvds_t0	pnl_gcs_lvds	LVDS
1058	TOP	link_iopad0_event_s_in_iopad4		LVDS
	input	event_s_in		
1059	TOP	vdd_vop_lvds_t0	pnl_vop_lvds	LVDS
1060	TOP	link_iopad0_event_p_in4_iopad1		LVDS
	input	event_p_in4		
1061	TOP	link_iopad0_event_p_in4_iopad2		LVDS
	input	event_p_in4		
1062	TOP	vdd_vc_lvds_t0	pnl_vc_lvds	LVDS
1063	TOP	link_iopad0_event_p_in4_iopad3		LVDS
	input	event_p_in4		
1064	TOP	vss_go_lvds_t1	pnl_go_lvds	LVDS
1065	TOP	clk_iopad0_lvds_FOUT		LVDS
	output	FOUT		
1066	TOP	clk_iopad0_lvds_clk_outer		LVDS
	output	clk_outer		

Pin No.	Side	Name	Physical Cell	IOPAD Type
	In/Out	Remarks		
1067	TOP	clk_iopad0_lvds_clk_in_iopad		LVDS
	input	clk_in_p,n		
1068	TOP	vss_go_lvds_t0	pnl_go_lvds	LVDS
1069	TOP	lvds_vref_t0	pnl_lvds_vref	LVDS
1070	TOP	pnl_filler_lvds_brk_t1	pnl_filler_lvds_brk_3g	LVDS Breaker
1071	TOP	pnl_filler_std2sstl_t1	pnl_filler_std2sstl	SSTL Breaker
1072	TOP	vss_go_ngpio_t0	pnl_go_ngpio	NGPIO
1073	TOP	hiz_pad_LINK_PAD_HIZ_GEN_1__link_hiz_		NGPIO
	input	link_hiz_		
1074	TOP	vss_gcs_ngpio_t0	pnl_gcs_ngpio	NGPIO
1075	TOP	hiz_pad_LINK_PAD_HIZ_GEN_2__link_hiz_		NGPIO
	input	link_hiz_		
1076	TOP	pnl_filler_8g_t1	pnl_filler_8g	FILLER
1077	TOP	pnl_filler_4g_t1	pnl_filler_4g	FILLER
1078	TOP	pnl_filler_2g_t0	pnl_filler_2g	FILLER

Pull-up resistance and pull-down resistance of each cells are shown in the table below. (Cells which don't have resistors are not shown.)

Table 2.1: Pull-Up / Down Resistance value

Master Cell	R[Ω]	I[A]	E[V]	Pull-Up / Down
pnl_it2pu8	82500	0.00004	3.3	Pull-Up
pnl_it2pd8	55000	0.00006	3.3	Pull-Down
pnl_tf12it0pu8	82500	0.00004	3.3	Pull-Up
pnl_tf12it0pd8	55000	0.00006	3.3	Pull-Down

3

Instruction Set

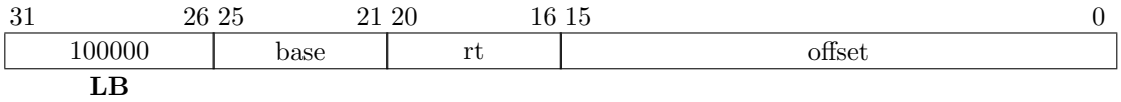
3.1 Instructions compatible with MIPS ISA

Responsive Multithreaded Processor supports instructions in MIPS ISA. Supported MIPS compatible instructions are shown below.

3.1.1 Load / Store Instruction

LB — **Load Byte**

8bit Load — MIPS I



Mnemonic:

LB rt, offset(base)

Function :

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

Exception :

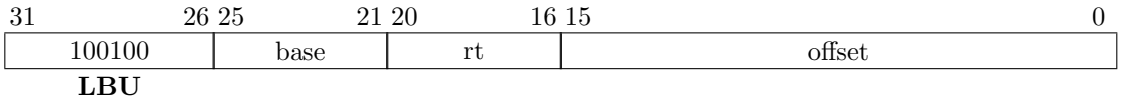
D-TLB No Entry Matched
D-TLB Protection Error

Overview :

Load a byte from memory as a 32-bit signed value.

LBU — Load Byte Unsigned

8bit Unsigned Load — MIPS I



Mnemonic:

LBU rt, offset(base)

Function :

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

Exception :

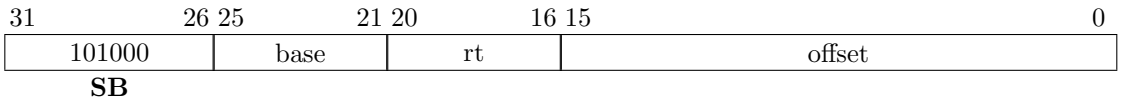
D-TLB No Entry Matched
D-TLB Protection Error

Overview :

Load a byte from memory as 32-bit unsigned value.

SB — Store Byte

8bit Store — MIPS I



Mnemonic:

SB rt, offset(base)

Function :

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

Exception :

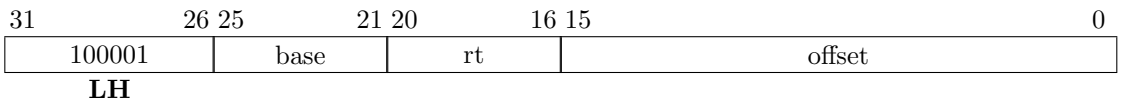
D-TLB No Entry Matched
D-TLB Protection Error

Overview :

The least-significant byte is stored to memory as signed value.

LH — Load Halfword

16bit Load — MIPS I



Mnemonic:

LH rt, offset(base)

Function :

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

Exception :

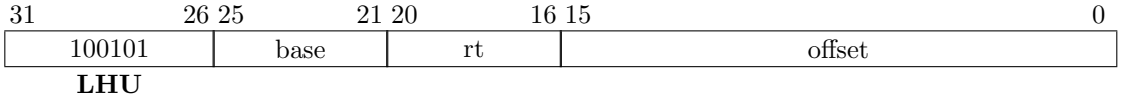
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Load)

Overview :

Load half word from memory. Loaded value is sign-extended and placed into GPR[rt].

LHU — Load Halfword Unsigned

16bit Unsigned Load — MIPS I



Mnemonic:

LHU rt, offset(base)

Function :

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

Exception :

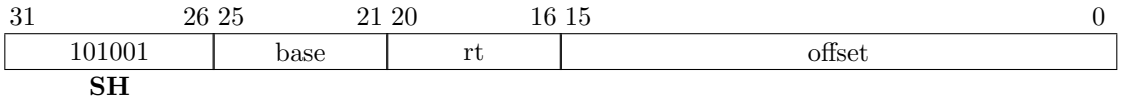
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Load)

Overview :

Load a half word from memory. Loaded value is zero-extended and placed into GPR[rt].

SH — Store Halfword

16bit Store — MIPS I



Mnemonic:

SH rt, offset(base)

Function :

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

Exception :

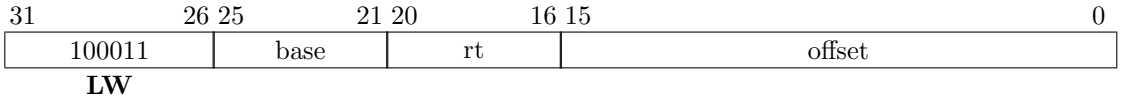
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Store)

Overview :

Store a half word.

LW — Load Word

32bit Load — MIPS I



Mnemonic:

LW rt, offset(base)

Function :

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

Exception :

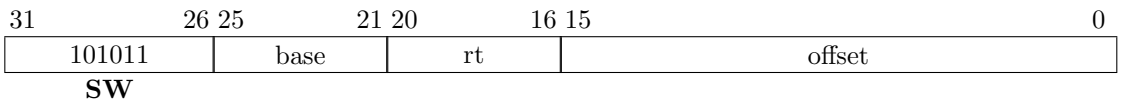
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Load)

Overview :

Load a word from memory.

SW — Store Word

32bit Load — MIPS I



Mnemonic:

SW rt, offset(base)

Function :

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

Exception :

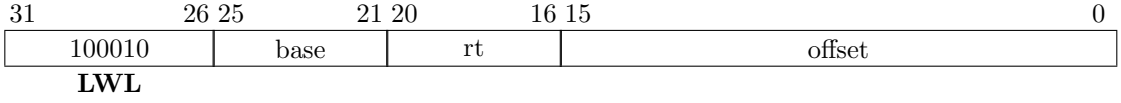
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Store)

Overview :

Store 1 word to memory.

LWL — Load Word Left

32bit unaligned load left — MIPS I



Mnemonic:

LWL rt, offset(base)

Function :

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

Exception :

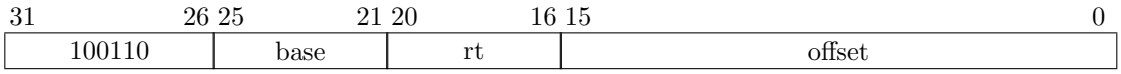
D-TLB No Entry Matched
D-TLB Protection Error

Overview :

Load a word from an unaligned memory address. Using with LWR, unaligned 1 word can be loaded to a GPR.

LWR — Load Word Right

32bit unaligned load right — MIPS I



LWR

Mnemonic:

LWR rt, offset(base)

Function :

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

Exception :

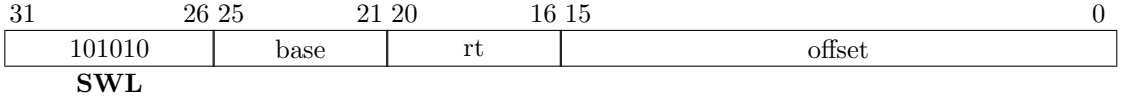
D-TLB No Entry Matched
D-TLB Protection Error

Overview :

Load a word from an analigned memory address. Using with LWL, unaligned 1 word can be loaded to a GPR.

SWL — Store Word Left

32bit unaligned store — MIPS I



SWL

Mnemonic:

SWL rt, offset(base)

Function :

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

Exception :

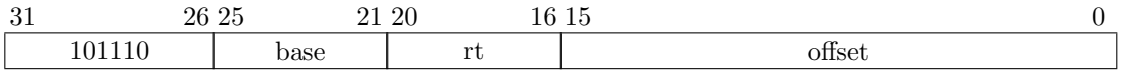
D-TLB No Entry Matched
D-TLB Protection Error

Overview :

Store the most-significant part of a word to an unaligned memory address.

SWR — Store Word Right

32bit unaligned store — MIPS I



Mnemonic:

SWR rt, offset(base)

Function :

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

Exception :

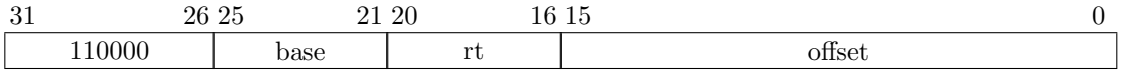
D-TLB No Entry Matched
D-TLB Protection Error

Overview :

Store the least-significant part of a word to an unaligned memory address.

LL — Load Linked Word

32-bit load for atomic read-modify-write — MIPS II



Mnemonic:

LL rt, offset(base)

Function :

GPR[rt] ← MEM.WORD[GPR[base] + sign_extend(offset)]
LL bit ← 1

Exception :

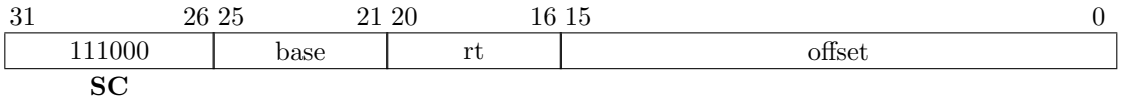
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Load)

Overview :

Load for Atomic Read-Modify-Write

SC Store Conditional Word

32-bit store for atomic read-modify-write MIPS II



Mnemonic:

SC rt, offset(base)

Function :

```
if LL Bit = 1 then
    MEM.WORD[GPR[base] + sign_extend(offset)] ← GPR[rt]
    GPR[rt] ← 1
else
    GPR[rt] ← 0
endif
```

Exception :

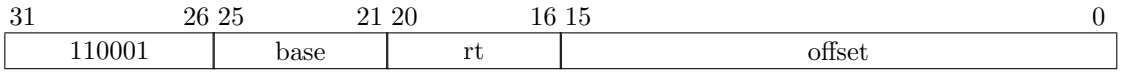
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Load)

Overview :

Perform a store operation for an Atomic Read-Modify-Write. It returns 1 if Atomic Read-Modify-Write succeeds, otherwise returns 0.

LWC1 — Load Word to Floating Point

Load Word to a FP register — MIPS I



LWC1

Mnemonic:

LWC1 ft, offset(base)

Function :

FPR[ft] ← MEM.WORD[GPR[base] + sign_extend(offset)]

Exception :

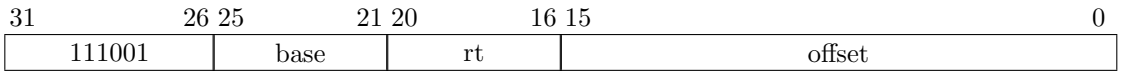
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Load)

Overview :

Perform a load operation from memory to a floating-point register.

SWC1 — Store Word from Floating Point

Store Word from FP register — MIPS I



SWC1

Mnemonic:

SWC1 ft, offset(base)

Function :

MEM.WORD[GPR[base] + sign_extend(offset)] ← FPR[ft]

Exception :

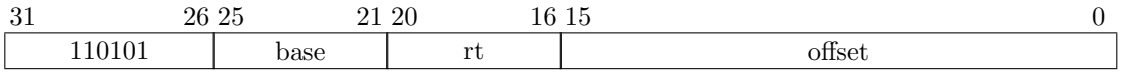
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Store)

Overview :

Perform a store operation to an memory from a floating-point register.

LDC1 Load Doubleword to Floating Point

Load operation for FP registers MIPS I



LDC1

Mnemonic:

LDC1 ft, offset(base)

Function :

FPR[ft] ← MEM.DWORD[GPR[base] + sign_extend(offset)]

Exception :

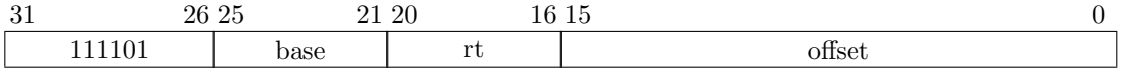
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Load)

Overview :

Perform a load doubleword operation from memory to a FPR.

SDC1 Store Doubleword from Floating Point

64-bit store for FPRs MIPS I



SDC1

Mnemonic:

SDC1 ft, offset(base)

Function :

MEM.DWORD[GPR[base] + sign_extend(offset)] ← FPR[ft]

Exception :

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

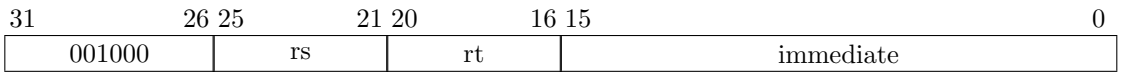
Overview :

Perform a 64-bit store operation to memory.

3.1.2 Computational Instructions

ADDI — Add Immediate Word

Add Immediate — MIPS I



ADDI

Mnemonic:

ADDI rt, rs, immediate

Function :

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

Exception :

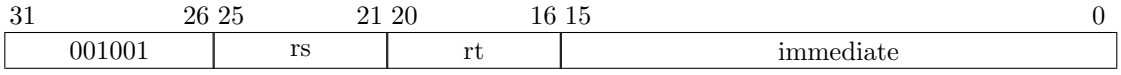
Overflow

Overview :

Add a constant to a 32-bit register.

ADDIU — Add Immediate Unsigned Word

Unsigned Add Immediate — MIPS I



ADDIU

Mnemonic:

ADDIU rt, rs, immediate

Function :

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

Exception :

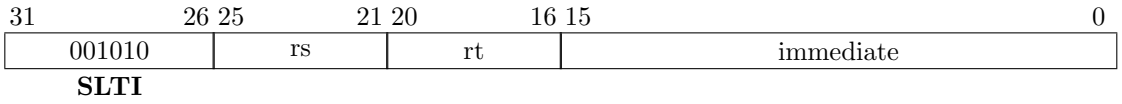
None

Overview :

Add a constant to a 32-bit register. No Integer Overflow exception occurs under any circumstances.

SLTI Set on Less Than Immediate

To record the result of a less-than comparison with a constant. MIPS I



Mnemonic:

SLTI rt, rs, immediate

Function :

```
if GPR[rs] < sign_extend(immediate) then
    GPR[rt] ← 1
else
    GPR[rt] ← 0
endif
```

Exception :

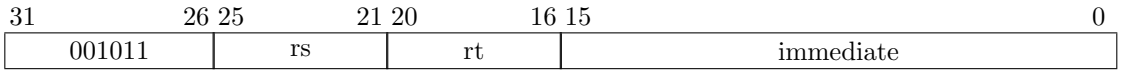
None

Overview :

Compare register value to constant.

SLTIU — Set on Less Than Immediate Unsigned

To record the result of an unsigned less-than comparison with a constant. MIPS I



SLTIU

Mnemonic:

SLTIU rt, rs, immediate

Function :

```
if GPR[rs] < sign_extend(immediate) then
    GPR[rt] ← 1
else
    GPR[rt] ← 0
endif
```

Exception :

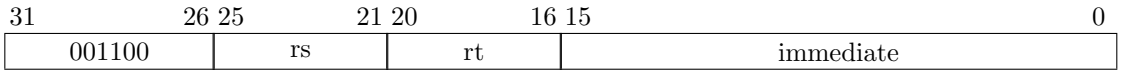
None

Overview :

Compare register value and constant value as unsigned integers.

ANDI — And Immediate

Logical AND Immediate MIPS I



ANDI

Mnemonic:

ANDI rt, rs, immediate

Function :

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

Exception :

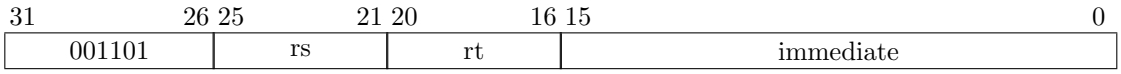
None

Overview :

Perform a bitwise AND operation with a constant.

ORI — Or Immediate

Logical OR Immediate — MIPS I



ORI

Mnemonic:

ORI rt, rs, immediate

Function :

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

Exception :

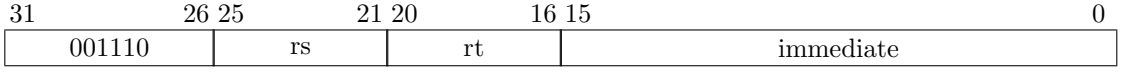
None

Overview :

Perform a bitwise OR operation with a constant.

XORI — Exclusive Or Immediate

Logical Exclusive OR Immediate. — MIPS I



XORI

Mnemonic:

XORI rt, rs, immediate

Function :

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

Exception :

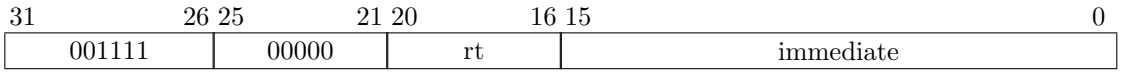
None

Overview :

Perform a bitwise XOR with a constant.

LUI — Load Upper Immediate

Load Upper Immediate MIPS I



Mnemonic:

LUI rt, immediate

Function :

GPR[rt] ← { immediate, 0000000000000000 }

Exception :

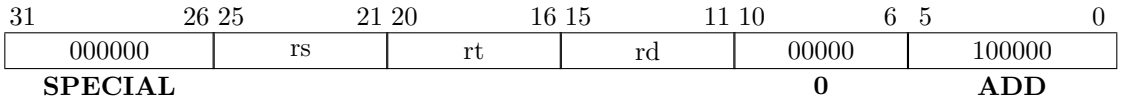
None

Overview :

Load a constant value into the upper half of a word.

ADD — Add Word

Add a word MIPS I



Mnemonic:

ADD rd, rs, rt

Function :

GPR[rd] ← GPR[rs] + GPR[rt]

Exception :

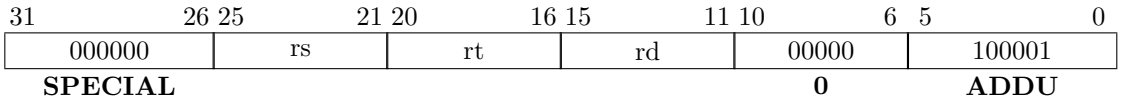
Overflow

Overview :

Peform an add operation.

ADDU — Add Unsigned Word

Unsigned Add MIPS I



Mnemonic:

ADDU rd, rs, rt

Function :

GPR[rd] ← GPR[rs] + GPR[rt]

Exception :

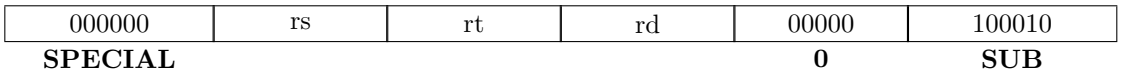
None

Overview :

Perform an add operation. No Integer Overflow occurs under any circumstances.

SUB — Subtract Word

Subtraction MIPS I



Mnemonic:

SUB rd, rs, rt

Function :

GPR[rd] ← GPR[rs] − GPR[rt]

Exception :

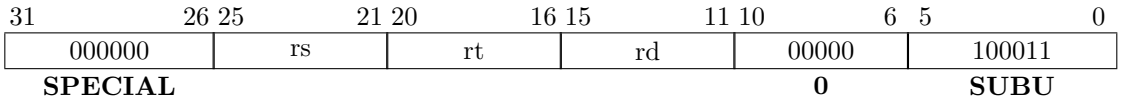
Overflow

Overview :

Perform a subtract operation.

SUBU — Subtract Unsigned Word

Unsigned Subtraction — MIPS I



Mnemonic:

SUBU rd, rs, rt

Function :

GPR[rd] ← GPR[rs] − GPR[rt]

Exception :

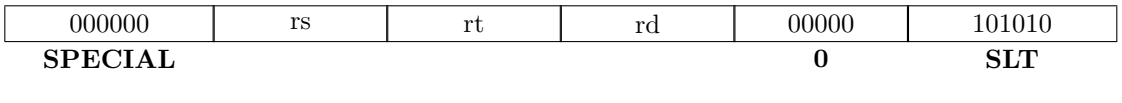
None

Overview :

Perform a subtract operation. No Integer Overflow occurs under any circumstances.

SLT — Set on Less Than

Set on Less Than — MIPS I



Mnemonic:

SLT rd, rs, rt

Function :

if GPR[rs] < GPR[rt] then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif

Exception :

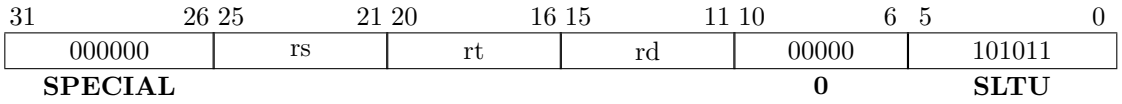
None

Overview :

Compare register values and record the result.

SLTU — Set on Less Than Unsigned

Unsigned Set on Less Than — MIPS I



Mnemonic:

SLTU rd, rs, rt

Function :

if GPR[rs] < GPR[rt] then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif

Exception :

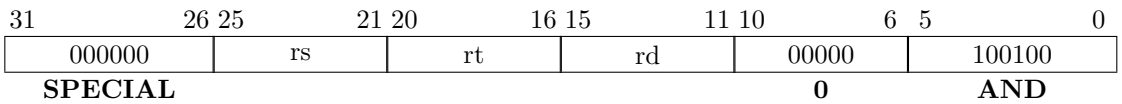
None

Overview :

Compare register values as unsigned integers and record the result.

AND — And

Logical AND — MIPS I



Mnemonic:

AND rd, rs, rt

Function :

GPR[rd] ← GPR[rs] **and** GPR[rt]

Exception :

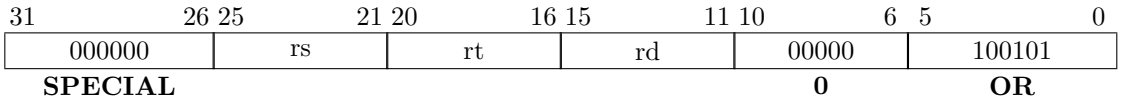
None

Overview :

Perform a logical and operation.

OR — Or

Logical OR — MIPS I



Mnemonic:

OR rd, rs, rt

Function :

GPR[rd] ← GPR[rs] **or** GPR[rt]

Exception :

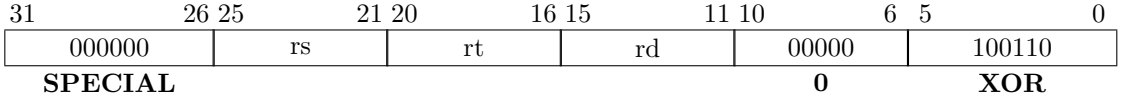
None

Overview :

Perform a logical or operation.

XOR — Exclusive Or

Logical Exclusive OR — MIPS I



Mnemonic:

XOR rd, rs, rt

Function :

GPR[rd] ← GPR[rs] **xor** GPR[rt]

Exception :

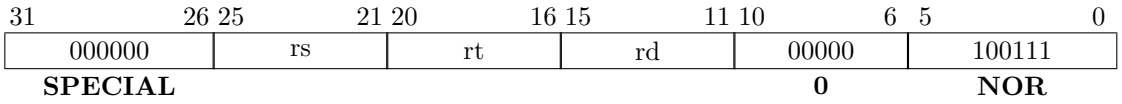
None

Overview :

Perform a logical exclusive or operation.

NOR — Not Or

Logical NOT OR — MIPS I



Mnemonic:

NOR rd, rs, rt

Function :

GPR[rd] ← GPR[rs] **nor** GPR[rt]

Exception :

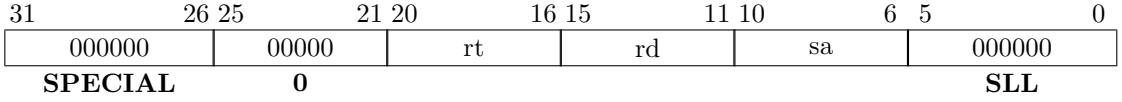
None

Overview :

Perform a logical NOR operation.

SLL — Shift Word Left Logical

Logical Shift Left — MIPS I



Mnemonic:

SLL rd, rt, sa

Function :

GPR[rd] ← GPR[rt] << sa

Exception :

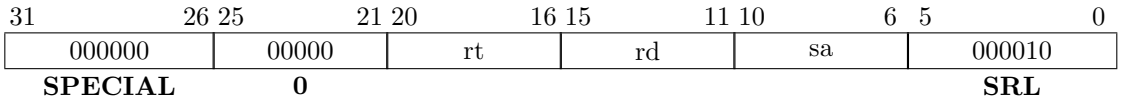
None

Overview :

Perform a logical shift-left operation.

SRL — Shift Word Right Logical

Logical Shift Right — MIPS I



Mnemonic:

SRL rd, rt, sa

Function :

GPR[rd] ← GPR[rt] >> sa

Exception :

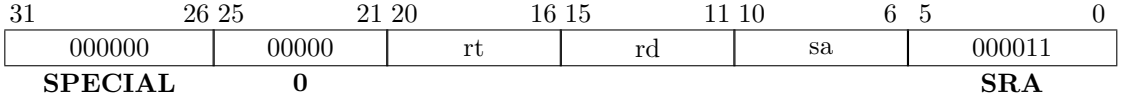
None

Overview :

Perform a logical shift-right operation.

SRA — Shift Word Right Arithmetic

Arithmetic Shift Right — MIPS I



Mnemonic:

SRA rd, rt, sa

Function :

GPR[rd] ← GPR[rt] >> sa

Exception :

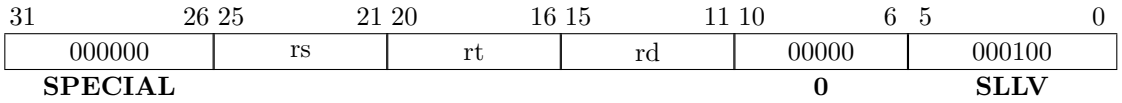
None

Overview :

Perform an arithmetic shift-right.

SLLV — Shift Word Left Logical Variable

Arithmetic Shift Left — MIPS I



Mnemonic:

SLLV rd, rt, rs

Function :

GPR[rd] ← GPR[rt] << GPR[rs]

Exception :

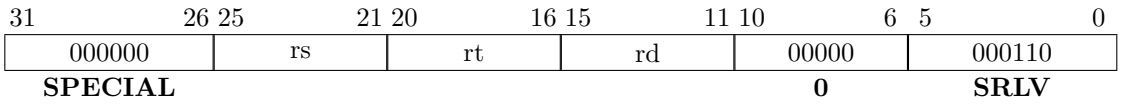
None

Overview :

Perform an arithmetic shift-left.

SRLV — Shift Word Right Logical Variable

Logical Shift Right Variable — MIPS I



Mnemonic:

SRLV rd, rt, rs

Function :

GPR[rd] ← GPR[rt] >> GPR[rs]

Exception :

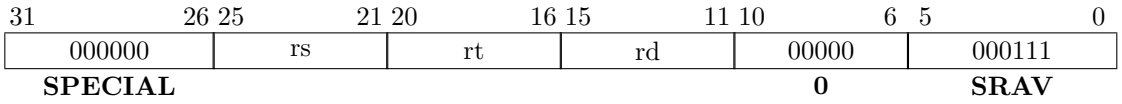
None

Overview :

Perform a logical shift-right variable.

SRAV — Shift Word Right Arithmetic Variable

Arithmetic Shift Right Variable — MIPS I



Mnemonic:

SRAV rd, rt, rs

Function :

GPR[rd] ← GPR[rt] >> GPR[rs]

Exception :

None

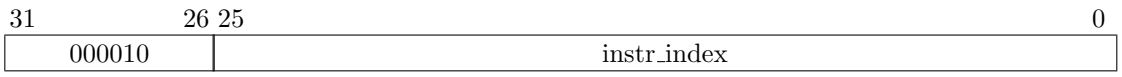
Overview :

Perform an arithmetic shift-right variable.

3.1.3 Jump / Branch Instructions

J — Jump

Jump — MIPS I



Mnemonic:

J target

Function :

pc ← { pc[31:28], instr_index, 00 }

Exception :

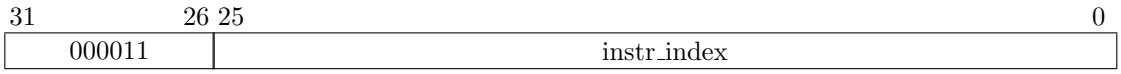
None

Overview :

Branch within the current 256MB aligned region.

JAL — Jump and Link

To procedure call within the current 256MB aligned region. MIPS I



Mnemonic:

JAL target

Function :

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

Exception :

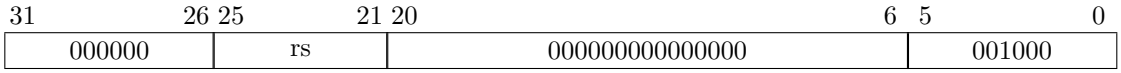
None

Overview :

Place the return address link in GPR 31. The return link is the address of the second instruction following the branch, where execution would continue after a procedure call.

JR — Jump Register

To branch to an instruction address in a register. MIPS I



Mnemonic:

JR rs

Function :

pc ← GPR[rs]

Exception :

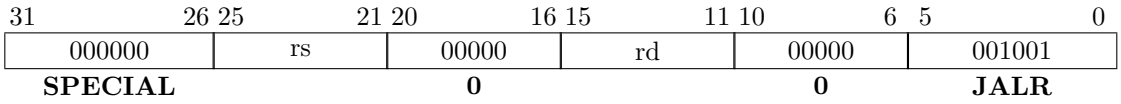
None

Overview :

Jump to the effective target address in GPR rs. Execute the instruction following the jump, in the branch delay slot, before jumping.

JALR — Jump and Link Register

To procedure call to an instruction address in a register. MIPS I



Mnemonic:

JALR rs (rd = 31 implied)
JALR rd, rs

Function :

pc ← GPR[rs]
GPR[rd] ← pc + 8

Exception :

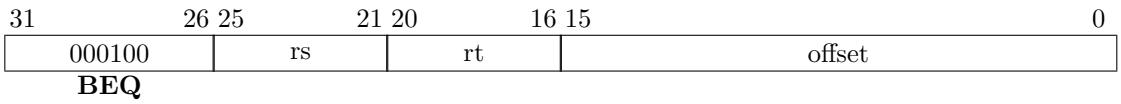
None

Overview :

Place the return address link in GPR rd. The return link is the address of the second instruction following the branch, where execution would continue after a procedure call.

BEQ — Branch on Equal

To conditional branch. MIPS I



Mnemonic:

BEQ rs, rt, offset

Function :

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

Exception :

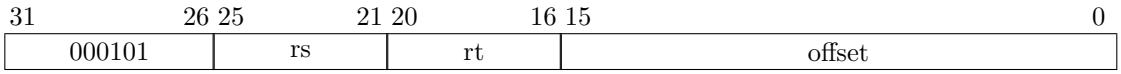
None

Overview :

Compare GPR[rs] and GPR[rt] then do a PC-relative conditional branch if equal.

BNE — Branch on Not Equal

To conditional branch. MIPS I



BNE

Mnemonic:

BNE rs, rt, offset

Function :

if GPR[rs] ≠ GPR[rt] then
branch

Exception :

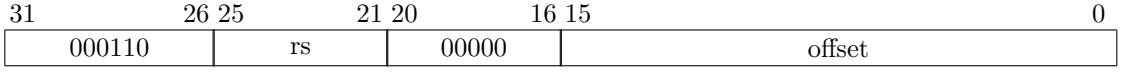
None

Overview :

Compare GPR[rs] and GPR[rt] then do a PC-relative conditional branch if not equal.

BLEZ — Branch on Less Than or Equal to Zero

To conditional branch. MIPS I



BLEZ **0**

Mnemonic:

BLEZ rs, offset

Function :

if GPR[rs] ≤ 0 then
branch

Exception :

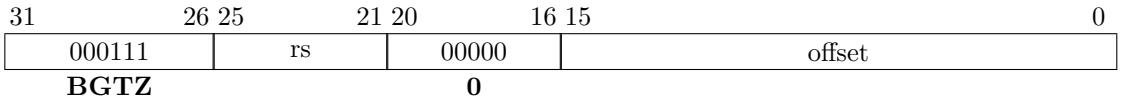
None

Overview :

Test if a GPR is less than or equal to zero, then do a PC-relative conditional branch.

BGTZ — Branch on Greater Than Zero

To conditional branch. MIPS I



Mnemonic:

BGTZ rs, offset

Function :

if GPR[rs] > 0 then
branch

Exception :

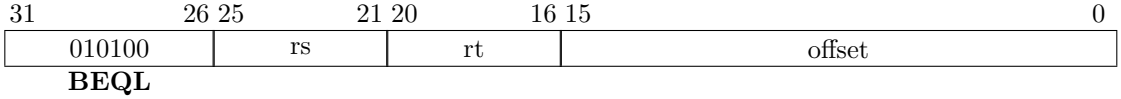
None

Overview :

Test if a GPR is greater than zero, then do a PC-relative conditional branch.

BEQL — Branch on Equal Likely

To conditional branch. MIPS II



Mnemonic:

BEQL rs, rt, offset

Function :

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

Exception :

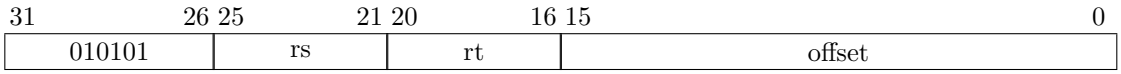
None

Overview :

To compare GPRs then do a PC-relative conditional branch; execute the delay slot only if the branch is taken.

BNEL — Branch on Not Equal Likely

To conditional branch. MIPS II



BNEL

Mnemonic:

BNEL rs, rt, offset

Function :

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

Exception :

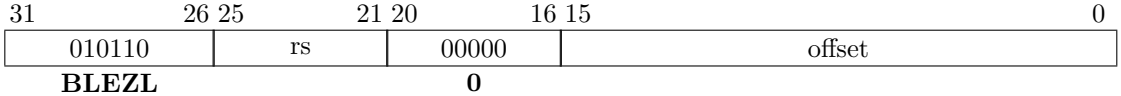
None

Overview :

To compare GPRs then do a PC-relative conditional branch; execute the delay slot only if the branch is taken.

BLEZL — Branch on Less Than or Equal to Zero Likely

To conditional branch. MIPS II



BLEZL **0**

Mnemonic:

BLEZL rs, offset

Function :

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

Exception :

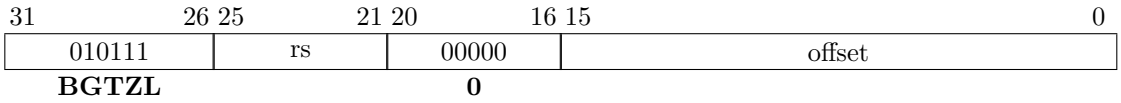
None

Overview :

To test a GPR then do a PC-relative conditional branch; execute the delay slot only if the branch is taken.

BGTZL — Branch on Greater Than to Zero Likely

To conditional branch. MIPS II



Mnemonic:

BGTZL rs, offset

Function :

if GPR[rs] > 0 then
branch_likely

Exception :

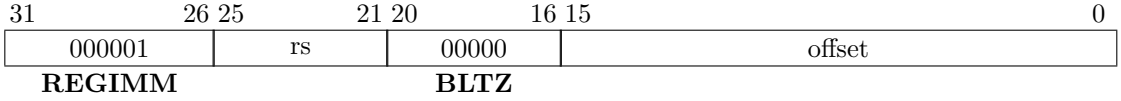
None

Overview :

To test a GPR then do a PC-relative conditional branch; execute the delay slot only if the branch is taken.

BLTZ — Branch on Less Than Zero

To conditional branch. MIPS I



Mnemonic:

BLTZ rs, offset

Function :

if GPR[rs] < 0 then
branch

Exception :

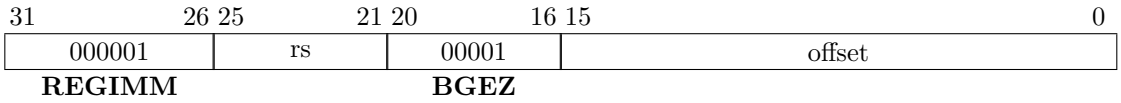
None

Overview :

To test a GPR then do a PC-relative conditional branch.

BGEZ — Branch on Greater Than or Equal to Zero

To conditional branch. MIPS I



Mnemonic:

BGEZ rs, offset

Function :

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

Exception :

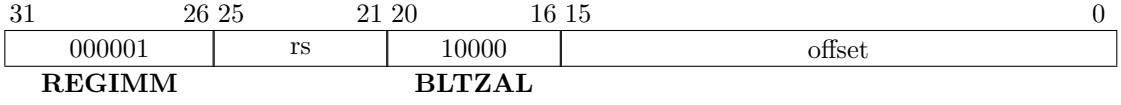
None

Overview :

To test a GPR then do a PC-relative conditional branch.

BLTZAL — Branch on Less Than Zero and Link

To conditional procedure call. MIPS I



Mnemonic:

BLTZAL rs, offset

Function :

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

Exception :

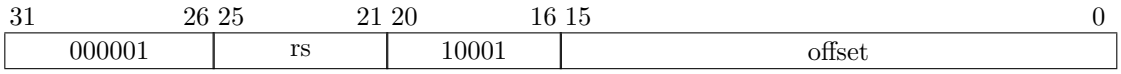
None

Overview :

To test a GPR then do a PC-relative conditional procedure call.

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

To conditional procedure call. MIPS I



Mnemonic:

BGEZAL rs, offset

Function :

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

Exception :

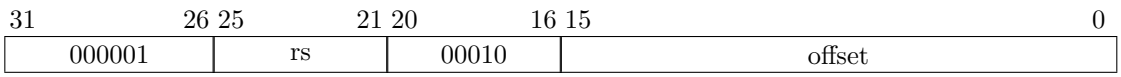
None

Overview :

To test a GPR then do a PC-relative conditional procedure call.

BLTZL — Branch on Less Than Zero Likely

To conditional branch. MIPS II



Mnemonic:

BLTZL rs, offset

Function :

if GPR[rs] $<$ 0 then
branch_likely

Exception :

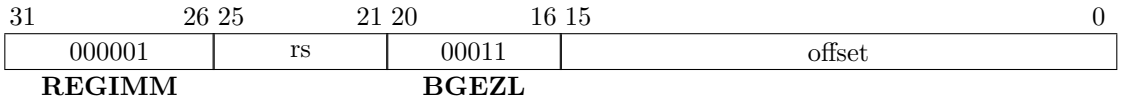
None

Overview :

To test a PC-relative conditional branch; execute the delay slot only if the branch is taken.

BGEZL — Branch on Greater Than or Equal to Zero Likely

To conditional branch. MIPS II



Mnemonic:

BGEZL rs, offset

Function :

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

Exception :

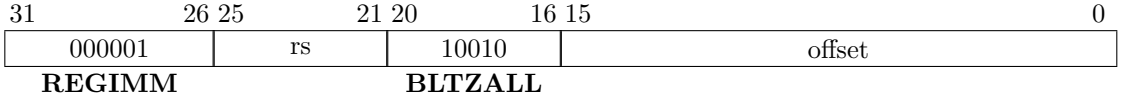
None

Overview :

To test a PC-relative conditional branch; execute the delay slot only if the branch is taken.

BLTZALL — Branch on Less Than Zero and Link Likely

To conditional procedure call. MIPS II



Mnemonic:

BLTZALL rs, offset

Function :

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

Exception :

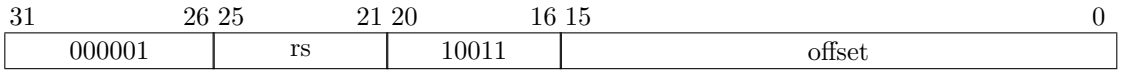
None

Overview :

Place the return address link in GPR 31. The return link is the address of the second instruction following the branch (not the branch itself), where execution would continue after a procedure call.

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

To conditional procedure call. MIPS II



Mnemonic:

BGEZALL rs, offset

Function :

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

Exception :

None

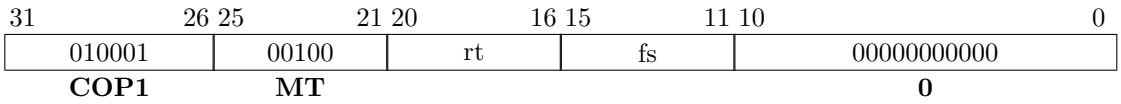
Overview :

Place the return address link in GPR 31. The return link is the address of the second instruction following the branch (not the branch itself), where execution would continue after a procedure call.

3.1.4 Floating-Point Instructions

MTC1 Move Word to Floating Point

Move a word to FPR MIPS I



Mnemonic:

MTC1 rt, fs

Function :

FPR[fs] $\leftarrow$ GPR[rt]

Exception :

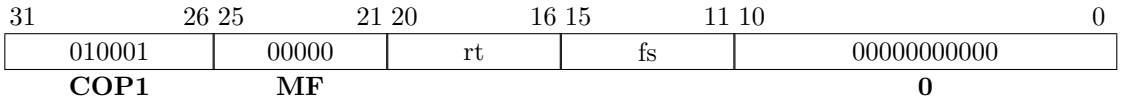
None

Overview :

Move a word to FPR from GPR.

MFC1 Move Word from Floating Point

Move a word from FPR MIPS I



Mnemonic:

MFC1 rt, fs

Function :

GPR[rt] ← FPR[fs]

Exception :

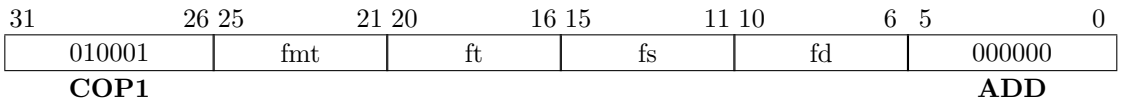
None

Overview :

Move a word from FPR to GPR.

ADD.fmt Floating Point Add

FP Add MIPS I



Mnemonic:

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

Function :

FPR[fd] ← FPR[fs] + FPR[ft]

Exception :

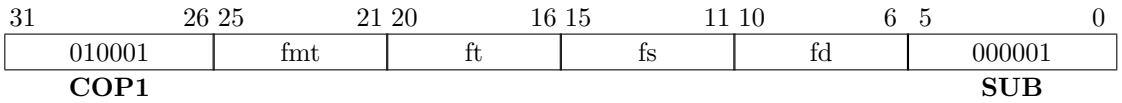
Floating Point Invalid Operation
Floating Point Inexact
Floating Point Overflow
Floating Point Underflow

Overview :

Perform an add operation.

SUB.fmt — Floating Point Subtract

FP Sub — MIPS I



Mnemonic:

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

Function :

FPR[fd] ← FPR[fs] − FPR[ft]

Exception :

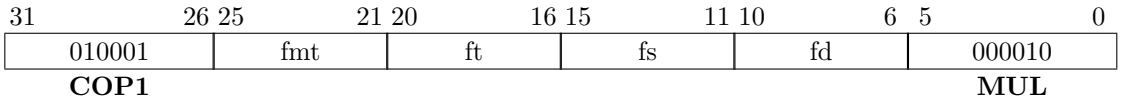
Floating Point Invalid Operation
Floating Point Inexact
Floating Point Overflow
Floating Point Underflow

Overview :

Perform a subtract operation.

MUL.fmt — Floating Point Multiply

To multiply floating-point values. MIPS I



Mnemonic:

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

Function :

FPR[fd] ← FPR[fs] × FPR[ft]

Exception :

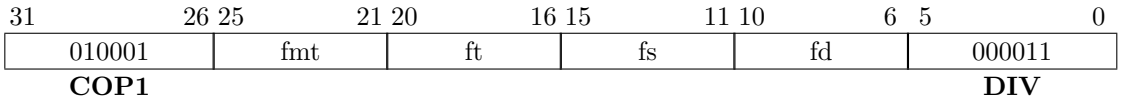
Floating Point Invalid Operation
Floating Point Inexact
Floating Point Overflow
Floating Point Underflow

Overview :

Perform a multiply operation.

DIV.fmt — Floating Point Divide

To divide a floating-point value. MIPS I



Mnemonic:

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

Function :

FPR[fd] ← FPR[fs] / FPR[ft]

Exception :

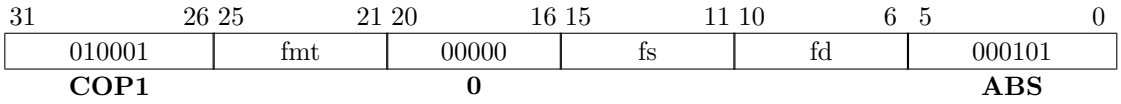
Floating Point Invalid Operation
Floating Point Inexact
Floating Point Overflow
Floating Point Underflow
Floating Point Divide By 0

Overview :

Perform a divide operation.

ABS.fmt Floating Point Absolute Value

To compute the absolute value of FP value. MIPS I



Mnemonic:

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

Function :

FPR[fd] ← abs(FPR[fs])

Exception :

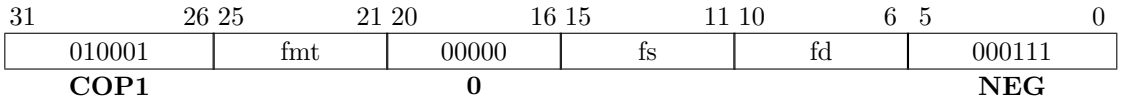
Floating Point Invalid Operation

Overview :

The absolute value of the value in FPR fs is placed in FPR fd. The operand and result are values in format fmt.

NEG.fmt — Floating Point Negate

To negate an FP value. MIPS I



Mnemonic:

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

Function :

FPR[fd] ← −(FPR[fs])

Exception :

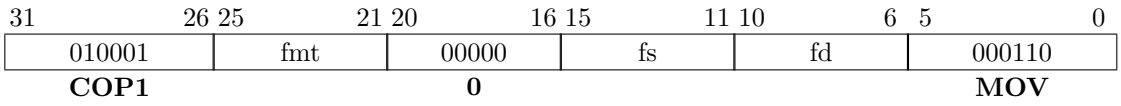
Floating Point Invalid Operation

Overview :

The value in FPR fs is negated and placed into FPR fd. The value is negated by changing the sign bit value. The operand and result are values in format fmt.

MOV.fmt — Floating Point Move

To move an FP value between FPRs. MIPS I



Mnemonic:

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

Function :

FPR[fd] ← FPR[fs]

Exception :

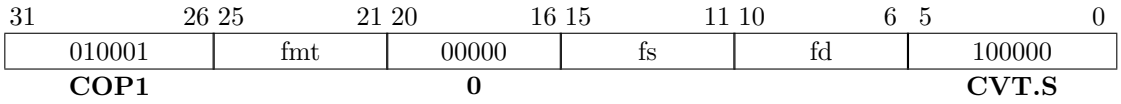
None

Overview :

The value in FPR fs is placed into FPR fd. The source and destination are values in format fmt.

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

To convert an FP or fixed-point value to single FP. MIPS I



Mnemonic:

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

Function :

FPR[fd] ← convert_and_round(FPR[fs])

Exception :

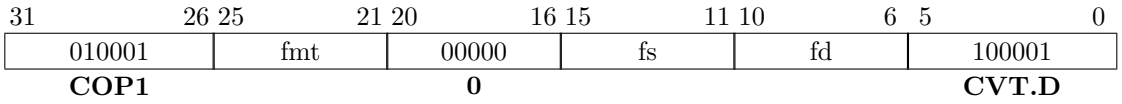
Floating Point Invalid Operation
Floating Point Inexact
Floating Point Overflow
Floating Point Underflow

Overview :

The value in FPR fs in format fmt is converted to a value in single floating-point format rounded according to the current rounding mode in FCSR. The result is placed in FPR fd.

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

To convert an FP or fixed-point value to double FP. MIPS I



Mnemonic:

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

Function :

FPR[fd] ← convert_and_round(FPR[fs])

Exception :

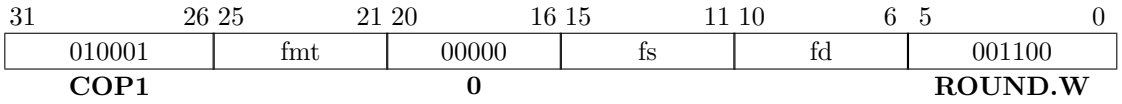
Floating Point Invalid Operation
Floating Point Inexact

Overview :

The value in FPR fs in format fmt is converted to a value in double floating-point format rounded according to the current rounding mode in FCSR. The result is placed in FPR fd.

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

To convert an FP value to 32-bit fixed-point, rounding to nearest. MIPS II



Mnemonic:

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

Function :

FPR[fd] ← convert_and_round(FPR[fs])

Exception :

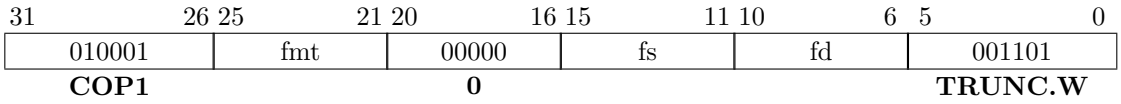
Floating Point Invalid Operation
Floating Point Inexact
Floating Point Overflow

Overview :

The value in FPR fs in format fmt, is converted to a value in 32-bit word fixed-point format rounding to nearest/even (rounding mode 0). The result is placed in FPR fd.

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

To convert an FP value to 32-bit fixed-point, rounding toward zero. MIPS II



Mnemonic:

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

Function :

FPR[fd] ← convert_and_round(FPR[fs])

Exception :

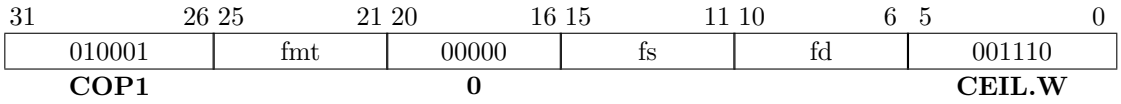
Floating Point Invalid Operation
Floating Point Inexact
Floating Point Overflow

Overview :

The value in FPR fs in format fmt, is converted to a value in 32-bit word fixed-point format using rounding toward zero (rounding mode 1)). The result is placed in FPR fd.

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

To convert an FP value to 32-bit fixed-point, rounding up. MIPS II



Mnemonic:

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

Function :

FPR[fd] ← convert_and_round(FPR[fs])

Exception :

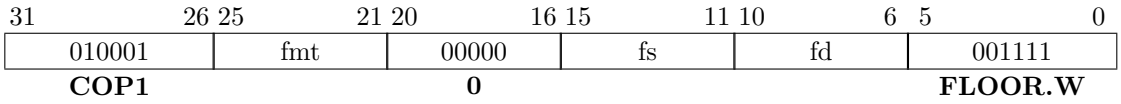
Floating Point Invalid Operation
Floating Point Inexact
Floating Point Overflow

Overview :

The value in FPR fs in format fmt, is converted to a value in 32-bit word fixed-point format rounding toward $+\infty$ (rounding mode 2). The result is placed in FPR fd.

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

To convert an FP value to 32-bit fixed-point, rounding down. MIPS II



Mnemonic:

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

Function :

FPR[fd] ← convert_and_round(FPR[fs])

Exception :

Floating Point Invalid Operation
Floating Point Inexact
Floating Point Overflow

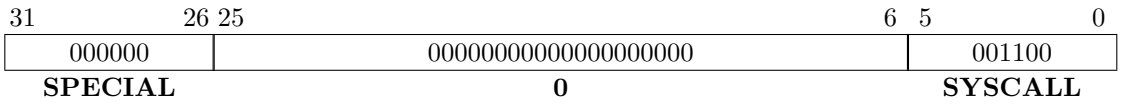
Overview :

The value in FPR fs in format fmt, is converted to a value in 32-bit word fixed-point format rounding toward $-\infty$ (rounding mode 3). The result is placed in FPR fd.

3.1.5 Miscellaneous Instructions

SYSCALL — System Call

To system call. MIPS I



Mnemonic:

SYSCALL

Function :

exception(system_call)

Exception :

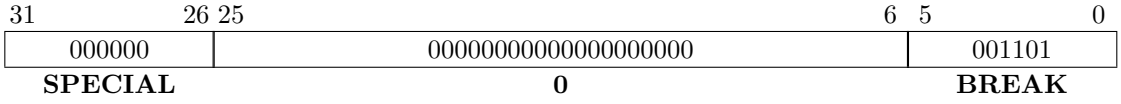
System Call

Overview :

Cause a System Call exception.

BREAK — Breakpoint

Breakpoint MIPS I



Mnemonic:

BREAK

Function :

exception(breakpoint)

Exception :

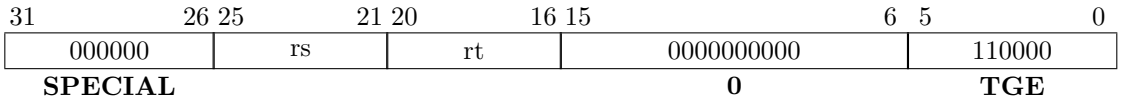
Break Point

Overview :

Cause a breakpoint exception.

TGE — Trap if Greater or Equal

To compare GPRs and do a conditional Trap. MIPS II



Mnemonic:

TGE rs, rt

Function :

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

Exception :

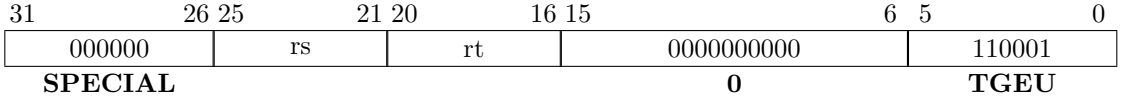
Trap

Overview :

Compare the contents of GPR rs and GPR rt as signed integers; if GPR rs is greater than or equal to GPR rt then take a Trap exception.

TGEU — Trap if Greater or Equal Unsigned

To compare GPRs and do a conditional Trap. MIPS II



Mnemonic:

TGEU rs, rt

Function :

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

Exception :

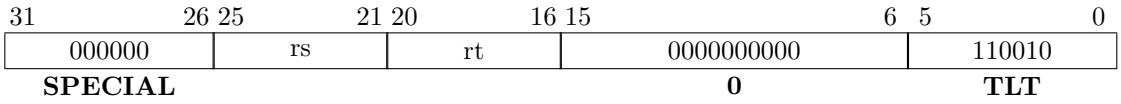
Trap

Overview :

Compare the contents of GPR rs and GPR rt as unsigned integers; if GPR rs is greater than or equal to GPR rt then take a Trap exception.

TLT — Trap if Less Than

To compare GPRs and do a conditional Trap. MIPS II



Mnemonic:

TLT rs, rt

Function :

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

Exception :

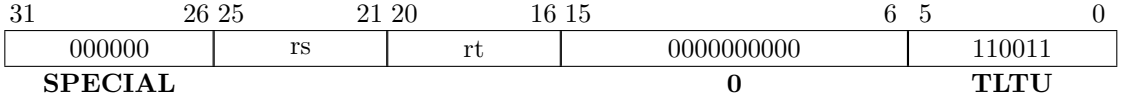
Trap

Overview :

Compare the contents of GPR rs and GPR rt as signed integers; if GPR rs is less than GPR rt then take a Trap exception.

TLTU — Trap if Less Than Unsigned

To compare GPRs and do a conditional Trap. MIPS II



Mnemonic:

TLTU rs, rt

Function :

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

Exception :

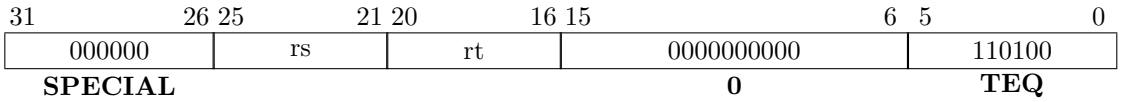
Trap

Overview :

Compare the contents of GPR rs and GPR rt as unsigned integers; if GPR rs is less than GPR rt then take a Trap exception.

TEQ — Trap if Equal

To compare GPRs and do a conditional Trap. MIPS II



Mnemonic:

TEQ rs, rt

Function :

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

Exception :

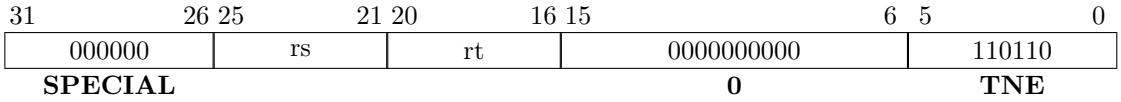
Trap

Overview :

Compare the contents of GPR[rs] and GPR[rt] as signed integers; if GPR[rs] is equal to GPR[rt] then take a Trap exception.

TNE — Trap if Not Equal

To conditional trap. MIPS II



Mnemonic:

TEQ rs, rt

Function :

```
if GPR[rs] ≠ GPR[rt] then
    exception(trap)
```

Exception :

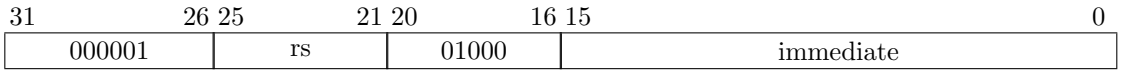
Trap

Overview :

If GPR[rs] is not equal to GPR[rt], then take a Trap exception.

TGEI — Trap if Greater or Equal Immediate

To conditional trap. MIPS II



Mnemonic:

TGEI rs, immediate

Function :

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

Exception :

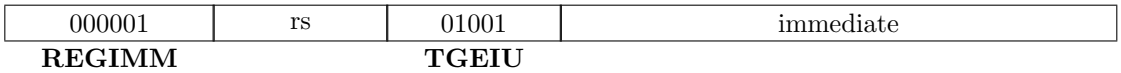
Trap

Overview :

If GPR[rs] is greather or equal to immediate value, then take a Trap exception.

TGEIU — Trap if Greater or Equal Immediate Unsigned

To conditional trap. MIPS II



Mnemonic:

TGEIU rs, immediate

Function :

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

Exception :

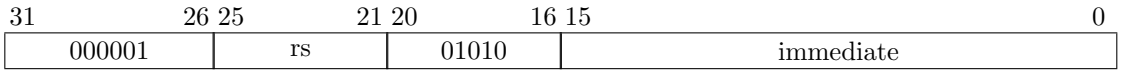
Trap

Overview :

If GPR[rs] is greater or equal to immediate value, then take a Trap exception. Values are treated as unsigned.

TLTI — Trap if Less Than Immediate

To conditional trap. MIPS II



Mnemonic:

TLTI rs, immediate

Function :

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

Exception :

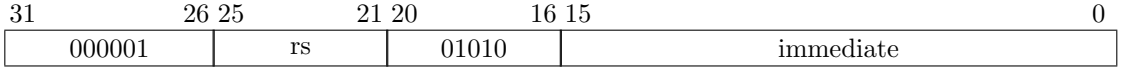
Trap

Overview :

If GPR[rs] is less than immediate valeu, then take a Trap exception.

TLTIU — Trap if Less Than Immediate Unsigned

To conditional trap. MIPS II



Mnemonic:

TLTIU rs, immediate

Function :

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

Exception :

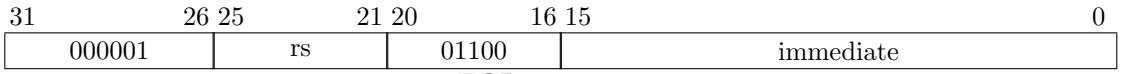
Trap

Overview :

If GPR[rs] is less than immediate value, then take a Trap exception. Values are treated as unsigned.

TEQI — Trap if Equal Immediate

To conditional trap. MIPS II



Mnemonic:

TEQI rs, immediate

Function :

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

Exception :

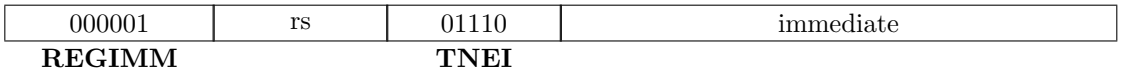
Trap

Overview :

If GPR[rs] is equal to immediate value, then take a Trap exception.

TNEI — Trap if Not Equal Immediate

To conditional trap. MIPS II



Mnemonic:

TNEI rs, immediate

Function :

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

Exception :

Trap

Overview :

If GPR[rs] is not equal to immediate value, then take a Trap exception.

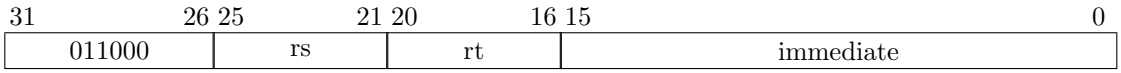
3.2 Instructions which are not compatible with MIPS ISA

Responsive Multithreaded Processor instructions which aren't compatible with MIPS ISA is shown below.

3.2.1 Computational Instructions

DADDI — Doubleword Add Immediate

64bit Add Immediate — MIPS III modified



DADDI

Mnemonic:

DADDI rt, rs, immediate

Function :

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

Exception :

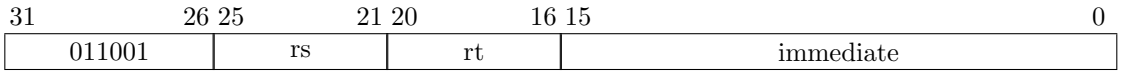
Overflow

Overview :

Add 64-bit integers. In *Responsive Multithreaded Processor*, this operation is executed on floating point registers.

DADDIU — Doubleword Add Immediate Unsigned

64bit Add Immediate — MIPS III modified



Mnemonic:

DADDIU rt, rs, immediate

Function :

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

Exception :

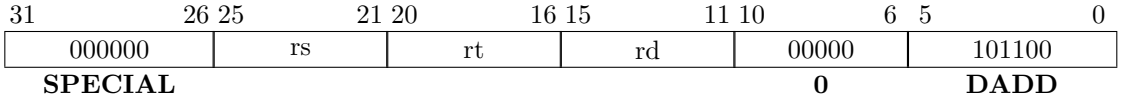
None

Overview :

Add 64-bit integers. This instruction doesn't trap on overflow. In *Responsive Multithreaded Processor*, this operation is executed on floating point registers.

DADD — Doubleword Add

64bit Addition — MIPS III modified



Mnemonic:

DADD rd, rs, rt

Function :

FPR[rd] ← FPR[rs] + FPR[rt]

Exception :

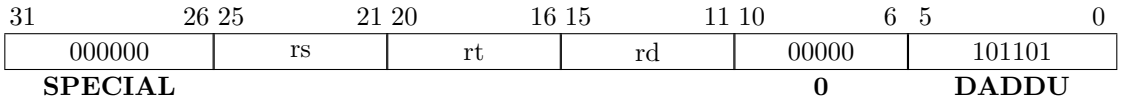
Overflow

Overview :

Add 64-bit integers. In *Responsive Multithreaded Processor*, this operations is executed on floating point registers.

DADDU — Doubleword Add Unsigned

64bit Addition — MIPS III modified



Mnemonic:

DADDU rd, rs, rt

Function :

FPR[rd] ← FPR[rs] + FPR[rt]

Exception :

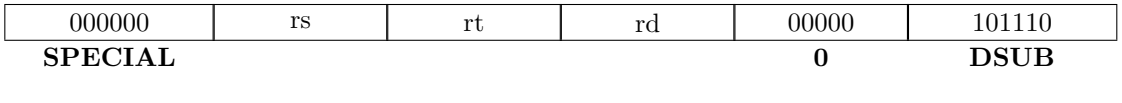
None

Overview :

Add 64-bit integers. This instruction doesn't trap on overflow. In *Responsive Multithreaded Processor*, this operation is executed on floating point registers.

DSUB — Doubleword Subtract

64bit Subtraction — MIPS III modified



Mnemonic:

DSUB rd, rs, rt

Function :

FPR[rd] ← FPR[rs] − FPR[rt]

Exception :

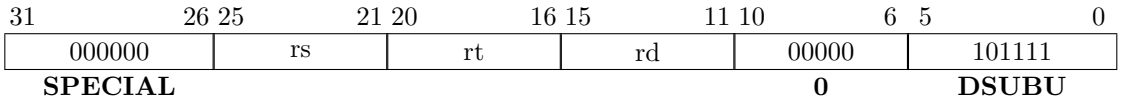
Overflow

Overview :

Perform a subtraction. In *Responsive Multithreaded Processor*, this operations is executed on floating point registers.

DSUBU — Doubleword Subtract Unsigned

64bit Subtraction — MIPS III modified



Mnemonic:

DSUBU rd, rs, rt

Function :

FPR[rd] ← FPR[rs] − FPR[rt]

Exception :

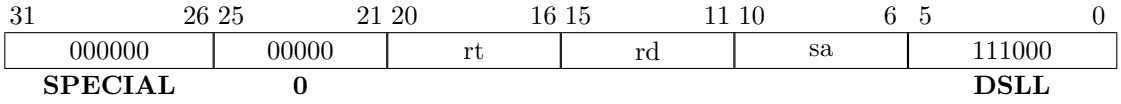
None

Overview :

Perform a subtraction. This instruction doesn't trap on overflow. In *Responsive Multithreaded Processor*, this operation is executed on floating point registers.

DSLL — Doubleword Shift Left Logical

64bit Shift Left Logical — MIPS III modified



Mnemonic:

DSLL rd, rt, sa

Function :

FPR[rd] ← FPR[rt] ≪ sa

Exception :

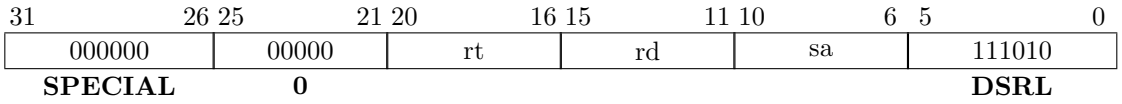
None

Overview :

Perform a logical shift-left. In *Responsive Multithreaded Processor*, this operation is executed on floating point registers.

DSRL — Doubleword Shift Right Logical

64bit Shift Right Logical — MIPS III modified



Mnemonic:

DSRL rd, rt, sa

Function :

FPR[rd] ← FPR[rt] ≫ sa

Exception :

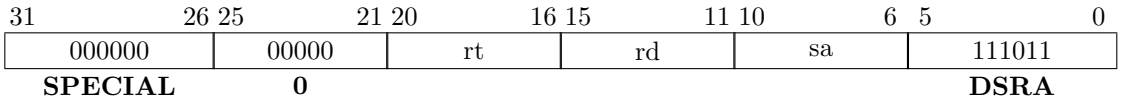
None

Overview :

Perform a logical shift-right. In *Responsive Multithreaded Processor*, this operation is executed on floating point registers.

DSRA — Doubleword Shift Right Arithmetic

64bit Shift Right Arithmetic — MIPS III modified



Mnemonic:

DSRA rd, rt, sa

Function :

FPR[rd] ← FPR[rt] ≫ sa

Exception :

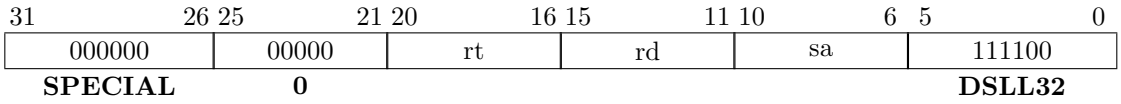
None

Overview :

Perform a arithmetic shift-right. In *Responsive Multithreaded Processor*, this operation is executed on floating point registers.

DSLL32 — Doubleword Shift Left Logical plus 32

64bit Shift Left Logical — MIPS III modified



Mnemonic:

DSLL32 rd, rt, sa

Function :

FPR[rd] ← FPR[rt] ≪ (sa + 32)

Exception :

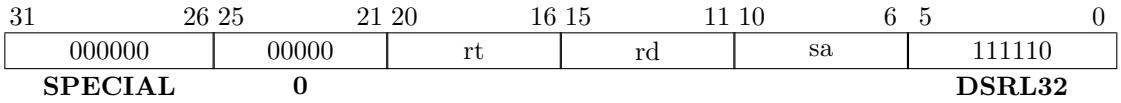
None

Overview :

Perform a logical shift-left. In *Responsive Multithreaded Processor*, this operation is executed on floating point registers.

DSRL32 — Doubleword Shift Right Logical plus 32

64bit Shift Right Logical — MIPS III modified



Mnemonic:

DSRL32 rd, rt, sa

Function :

FPR[rd] ← FPR[rt] ≫ (sa + 32)

Exception :

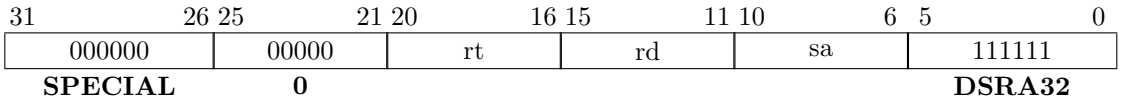
None

Overview :

Perform a logical shift-right. In *Responsive Multithreaded Processor*, this operation is executed on floating point registers.

DSRA32 Doubleword Shift Right Arithmetic plus 32

64bit Shift Right Arithmetic MIPS III modified



Mnemonic:

DSRA32 rd, rt, sa

Function :

FPR[rd] ← FPR[rt] ≫ (sa + 32)

Exception :

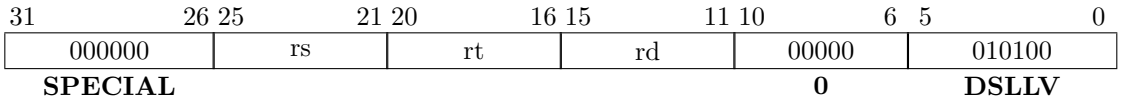
None

Overview :

Perform a arithmetic shift-right. In *Responsive Multithreaded Processor*, this operation is executed on floating point registers.

DSLLV Doubleword Shift Left Logical Variable

64bit Shift Left Logical MIPS III modified



Mnemonic:

DSLLV rd, rt, rs

Function :

FPR[rd] ← FPR[rt] ≪ FPR[rs]

Exception :

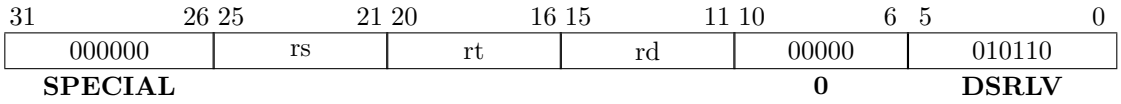
None

Overview :

Perform a logical shift-left. In *Responsive Multithreaded Processor*, this operation is executed on floating point registers.

DSRLV — Doubleword Shift Right Logical Variable

64bit Shift Right Logical — MIPS III modified



Mnemonic:

DSRLV rd, rt, rs

Function :

FPR[rd] ← FPR[rt] ≫ FPR[rs]

Exception :

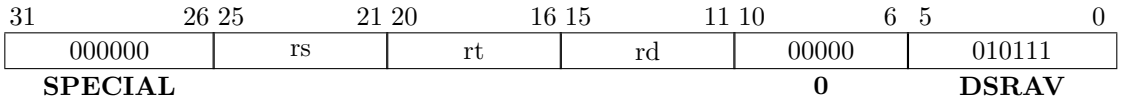
None

Overview :

Perform a logical shift-right. In *Responsive Multithreaded Processor*, this operation is executed on floating point registers.

DSRAV — Doubleword Shift Right Arithmetic Variable

64bit Shift Right Arithmetic — MIPS III modified



Mnemonic:

DSRAV rd, rt, rs

Function :

FPR[rd] ← FPR[rt] ≫ FPR[rs]

Exception :

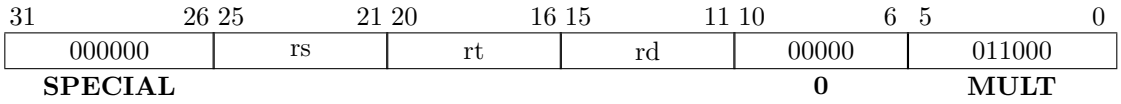
None

Overview :

Perform a arithmetic shift-right. In *Responsive Multithreaded Processor*, this operation is executed on floating point registers.

MULT — Multiply Word

Multiply signed integers — MIPS I modified



Mnemonic:

MULT rd, rs, rt

Function :

GPR[rd] ← GPR[rs] × GPR[rt]

Exception :

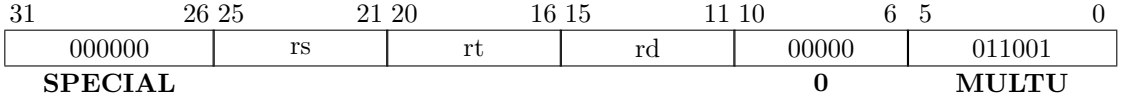
None

Overview :

The 32-bit word in GPR[rt] is multiplied by the 32-bit value in GPR[rs]]. In *Responsive Multithreaded Processor*, this instruction has 3 operands, and the low-order 32-bit word of the result is placed into GPR[rd].

MULTU — Multiply Word Unsigned

Multiply unsigned integers — MIPS I modified



Mnemonic:

MULTU rd, rs, rt

Function :

GPR[rd] ← GPR[rs] × GPR[rt]

Exception :

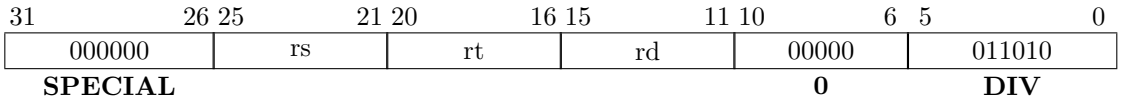
None

Overview :

Perform a unsigned multiplication. In *Responsive Multithreaded Processor*, this instruction has 3 operands, and the low-order 32-bit word of the result is placed into GPR[rd].

DIV — Divide Word

Signed division — MIPS I modified



Mnemonic:

DIV rd, rs, rt

Function :

GPR[rd] ← GPR[rs] ÷ GPR[rt]

Exception :

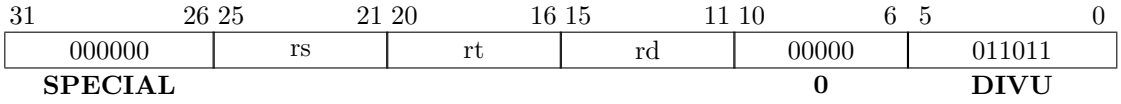
Divide by Zero

Overview :

Perform a signed division. In *Responsive Multithreaded Processor*, this instruction has 3 operands, and the quotient is placed into GPR[rd].

DIVU — Divide Word Unsigned

Unsigned division — MIPS I modified



Mnemonic:

DIVU rd, rs, rt

Function :

GPR[rd] ← GPR[rs] ÷ GPR[rt]

Exception :

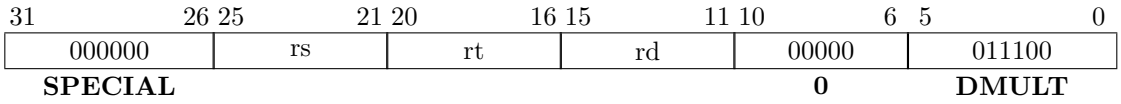
Divide by Zero

Overview :

Perform a unsigned division. In *Responsive Multithreaded Processor*, this instruction has 3 operands, and the quotient is placed into GPR[rd].

DMULT — Doubleword Multiply

Signed 64-bit Multiplication — MIPS III modified



Mnemonic:

DMULT rd, rs, rt

Function :

FPR[rd] ← FPR[rs] × FPR[rt]

Exception :

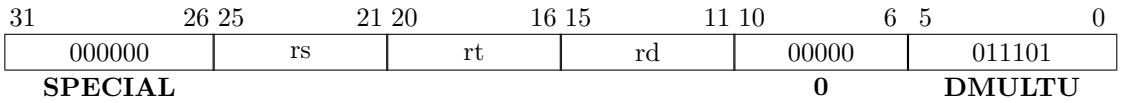
None

Overview :

Perform a multiplication. In *Responsive Multithreaded Processor*, this instruction has 3 operands, and the low-order 64-bit word of the result is placed into FPR[rd].

DMULTU — Doubleword Multiply Unsigned

Unsigned 64-bit Multiplication — MIPS III modified



Mnemonic:

DMULTU rd, rs, rt

Function :

FPR[rd] ← FPR[rs] × FPR[rt]

Exception :

None

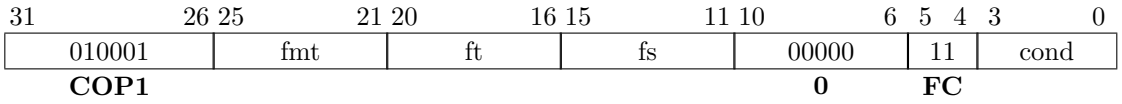
Overview :

Perform a unsigned multiprication. In *Responsive Multithreaded Processor*, this instruction has 3 operands, and the low-order 64-bit word of the result is placed into FPR[rd].

3.2.2 Floating-Point Instructions

C.cond.fmt — Floating-Point Compare

Floating-Point Compare MIPS I modified



Mnemonic:

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

Function :

FPR[7] ← FPR[fs] conpare_cond FPR[ft]

Exception :

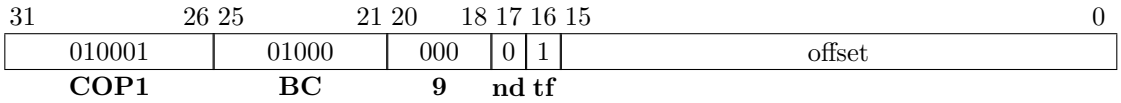
Floating Point Invalid

Overview :

浮動小数点の比較を行う．*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に結果が格納される．比較条件 (cond) には，eq, le, lt, ult, f, sf, un, ueq, seq, ngt, ngl, ngle, ole, ule, olt, nge のどれか指定する．

BC1T — Branch on FP True

Branch on FP True — MIPS I modified



Mnemonic:

BC1T offset

Function :

if FPR[7] = 1 then
branch

Exception :

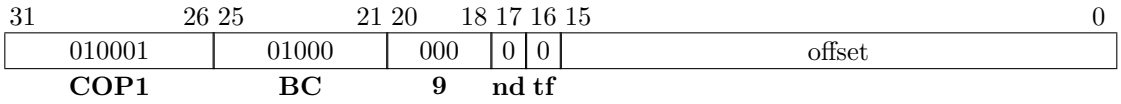
None

Overview :

If FPR[7] is true (value '1'), then branch to effective target address. In *Responsive Multithreaded Processor*, this instruction tests FPR[7], not status register.

BC1F — Branch on FP False

Branch on FP False — MIPS I modified



Mnemonic:

BC1F offset

Function :

if FPR[7] = 0 then
branch

Exception :

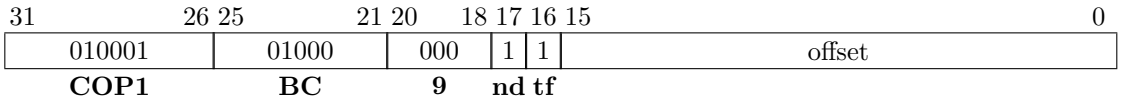
None

Overview :

If FPR[7] is false (value '0'), then branch to effective target address. In *Responsive Multithreaded Processor*, this instruction tests FPR[7], not status register.

BC1TL — Branch on FP True Likely

Branch on FP True Likely — MIPS II modified



Mnemonic:

BC1TL offset

Function :

if FPR[7] = 1 then
branch_likely

Exception :

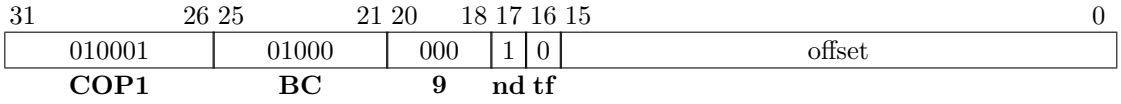
None

Overview :

If FPR[7] is true (value '1'), then branch to effective target address. In *Responsive Multithreaded Processor*, this instruction tests FPR[7], not status register.

BC1FL — Branch on FP False Likely

Branch on FP False Likely — MIPS II modified



Mnemonic:

BC1FL offset

Function :

if FPR[7] = 0 then
branch_likely

Exception :

None

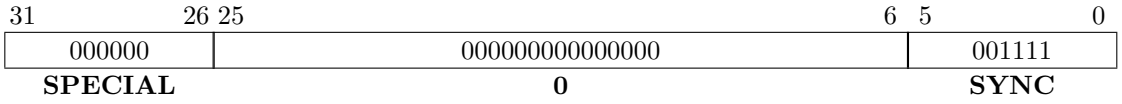
Overview :

If FPR[7] is false (value '0'), then branch to effective target address. In *Responsive Multithreaded Processor*, this instruction tests FPR[7], not status register.

3.2.3 Other Instructions

SYNC — **Synchronize Operation**

To order instruction executions — MIPS II modified



Mnemonic:

SYNC

Function :

synchronize_operation_order()

Exception :

None

Overview :

In *Responsive Multithreaded Processor*, instructions are executed Out-of-Order. SYNC guarantees instruction execution order before and after its execution. In other words, instructions after SYNC cannot be executed before SYNC. In addition, SYNC cannot be executed speculatively, it can be used to control speculative execution.

3.2.4 Unsupported MIPS II Instructions

*Responsive Multithreaded Processor*is basically MIPS II ISA compatible, but some of the MIPS II instructions are not supported. The unsupported MIPS II instructions are shown below.

Mnemonic	Description	
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

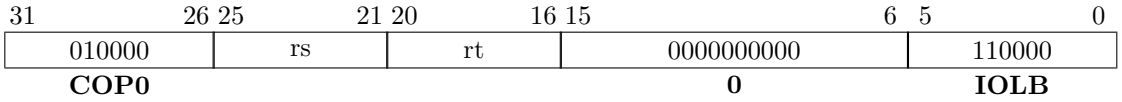
3.3 Responsive Multithreaded Processor Specific Instructions

Responsive Multithreaded Processor specific instructions are shown below.

3.3.1 Load / Store Instruction

IOLB — Load Byte for I/O

Load Instruction for I/O RESPII



Mnemonic:

IOLB rt, rs

Function :

GPR[rt] ← sign_extend(MEM.BYTE[GPR[rs]])

Exception :

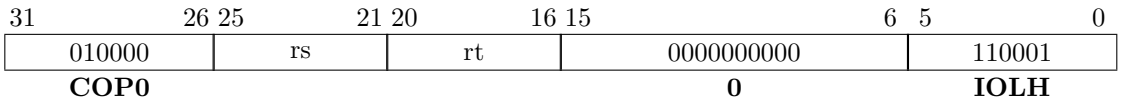
D-TLB No Entry Matched
D-TLB Protection Error

Overview :

Load a Byte from specified address. Because this instruction is not speculative execution, it is used for modules whose status changes after load like I/O devices.

IOLH Load Half Word for I/O

Load Instruction for I/O RESPII



Mnemonic:

IOLH rt, rs

Function :

GPR[rt] ← sign_extend(MEM.HWORD[GPR[rs]])

Exception :

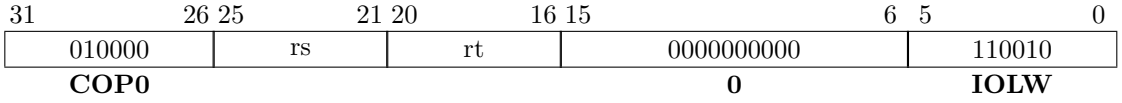
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Load)

Overview :

Load a Half Word from specified address. Because this instruction is not speculative execution, it is used for modules whose status changes after load like I/O devices.

IOLW — Load Word for I/O

Load Instruction for I/O — RESPII



Mnemonic:

IOLW rt, rs

Function :

GPR[rt] ← sign_extend(MEM.WORD[GPR[rs]])

Exception :

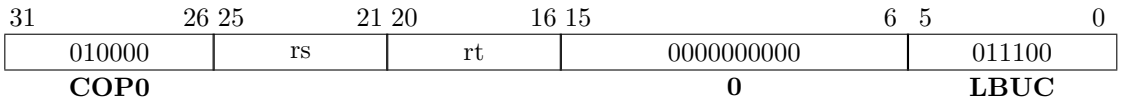
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Load)

Overview :

Load a Word from specified address. Because this instruction is not speculative execution, it is used for modules whose status changes after load like I/O devices.

LBUC — Uncache Load Byte

Uncache Load Instruction — RESPII



Mnemonic:

LBUC rt, rs

Function :

GPR[rt] ← sign_extend(MEM.BYTE[GPR[rs]])

Exception :

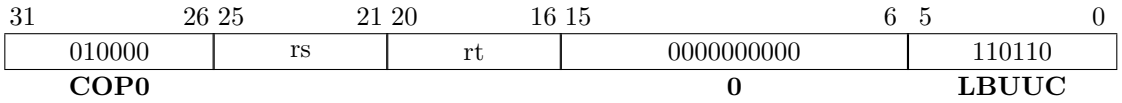
D-TLB No Entry Matched
D-TLB Protection Error

Overview :

Uncached load a Byte from specified address.

LBUUC — Load Byte Unsigned

Uncache Load Instruction RESPII



Mnemonic:

LBUC rt, rs

Function :

GPR[rt] ← zero_extend(MEM.BYTE[GPR[rs]])

Exception :

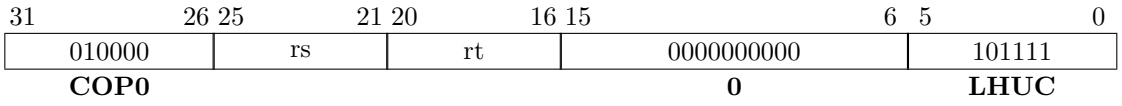
D-TLB No Entry Matched
D-TLB Protection Error

Overview :

Uncached load a Byte from specified address.

LHUC — Uncache Load Half Word

Uncache Load Instruction RESPII



Mnemonic:

LHUC rt, rs

Function :

GPR[rt] ← sign_extend(MEM.HWORD[GPR[rs]])

Exception :

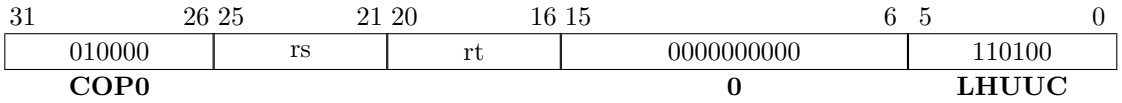
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Load)

Overview :

Uncached load a Half Word from specified address.

LHUUC — Uncache Load Half Word Unsigned

Uncache Load Instruction RESPII



Mnemonic:

LHUUC rt, rs

Function :

GPR[rt] ← zero_extend(MEM.HWORD[GPR[rs]])

Exception :

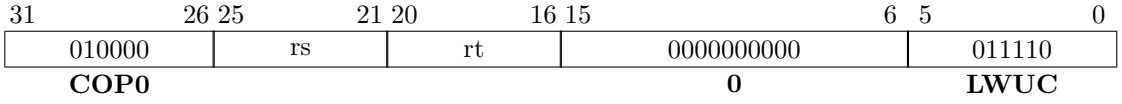
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Load)

Overview :

Uncached load a Half Word from specified address.

LWUC — Uncache Load Word

Uncache Load Instruction RESPII



Mnemonic:

LWUC rt, rs

Function :

GPR[rt] ← sign_extend(MEM.WORD[GPR[rs]])

Exception :

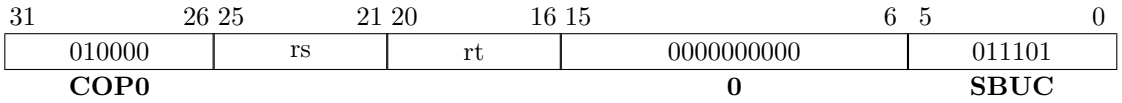
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Load)

Overview :

Uncached load a Word from specified address.

SBUC — Uncache Store Byte

Uncache Store Instruction RESPII



Mnemonic:

SBUC rt, rs

Function :

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

Exception :

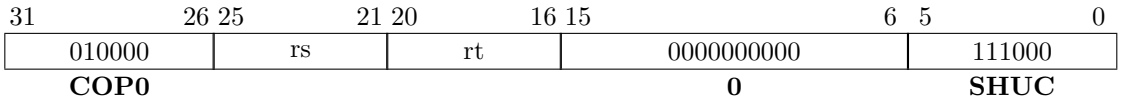
D-TLB No Entry Matched
D-TLB Protection Error

Overview :

Uncached store a Byte to specified address.

SHUC — Uncache Store Half Word

Uncache Store Instruction RESPII



Mnemonic:

SHUC rt, rs

Function :

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

Exception :

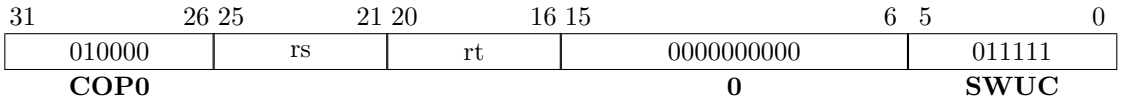
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Store)

Overview :

Uncached store a Half Word to specified address.

SWUC — Uncache Store Word

Uncache Store Instruction — RESPII



Mnemonic:

SWUC rt, rs

Function :

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

Exception :

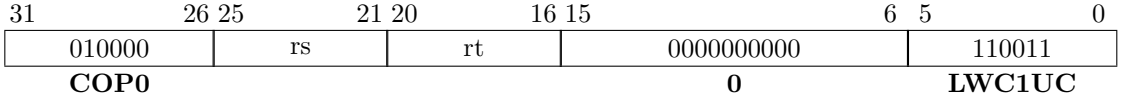
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Store)

Overview :

Uncached store a Word to specified address.

LWC1UC — Uncache Load Word to Floating Point

Uncache Load Word to a FP register — MIPS I



Mnemonic:

LWC1UC rs, ft

Function :

FPR[ft] ← MEM.WORD[GPR[rs]]

Exception :

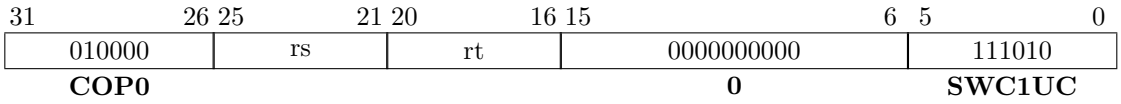
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Load)

Overview :

Perform a load operation from memory to a floating-point register.

SWC1UC — Uncache Store Word from Floating Point

Uncache Store Word from FP register — MIPS I



Mnemonic:

SWC1UC rs, ft

Function :

MEM.WORD[GPR[rs]] ← FPR[ft]

Exception :

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

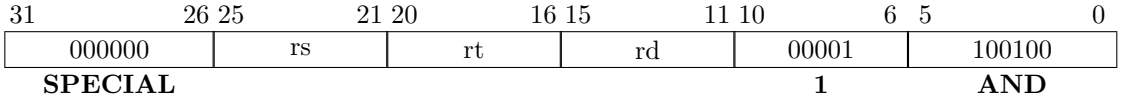
Overview :

Perform a store operation to an memory from a floating-point register.

3.3.2 Arithmetic Instructions

DAND — Doubleword And

64bit Logical AND — RESPII



Mnemonic:

DAND rd, rs, rt

Function :

FPR[rd] ← FPR[rs] **and** FPR[rt]

Exception :

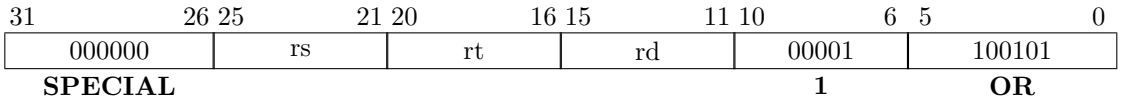
None

Overview :

Perform a 64-bit logical AND operation on FPRs.

DOR — Doubleword Or

64bit Logical OR — RESPII



Mnemonic:

DOR rd, rs, rt

Function :

FPR[rd] ← FPR[rs] **or** FPR[rt]

Exception :

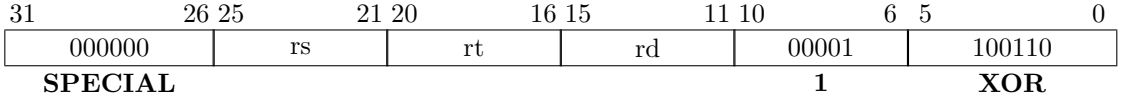
None

Overview :

Perform a 64-bit logical OR operation on FPRs.

DXOR — Doubleword Exclusive Or

64bit logical Exclusive OR — RESPII



Mnemonic:

DXOR rd, rs, rt

Function :

FPR[rd] ← FPR[rs] **xor** FPR[rt]

Exception :

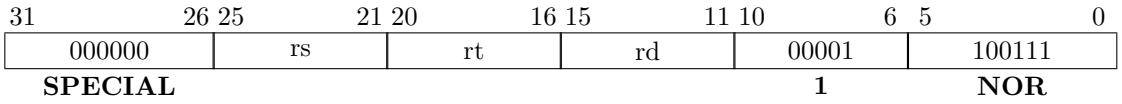
None

Overview :

Perform a 64-bit logical XOR operation on FPRs.

DNOR — Doubleword Not Or

64bit logical NOT OR — RESPII



Mnemonic:

DNOR rd, rs, rt

Function :

FPR[rd] ← FPR[rs] **nor** FPR[rt]

Exception :

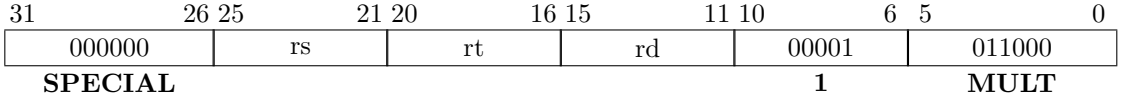
None

Overview :

Perform a 64-bit logical NOT OR on FPRs.

MULTH — Multiply Word on High Bit

Signed Multiplication — RESPII



Mnemonic:

MULTH rd, rs, rt

Function :

GPR[rd] ← GPR[rs] × GPR[rt]

Exception :

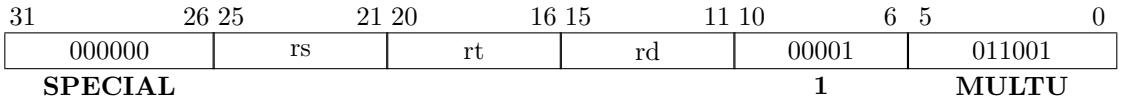
None

Overview :

Perform a multiplication. The high-order 32-bit of the result is placed into GPR[rd].

MULTUH — Multiply Word Unsigned on High Bit

Unsigned Multiplication RESPII



Mnemonic:

MULTUH rd, rs, rt

Function :

GPR[rd] ← GPR[rs] × GPR[rt]

Exception :

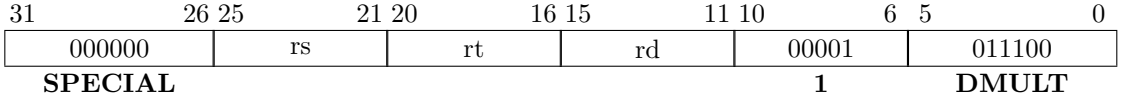
None

Overview :

Perform a unsigned multiplication. The high-order 32-bit of the result is placed into GPR[rd].

DMULTH — Doubleword Multiply on High Bit

Signed 64-bit Multiplication RESPII



Mnemonic:

DMULTH rd, rs, rt

Function :

FPR[rd] ← FPR[rs] × FPR[rt]

Exception :

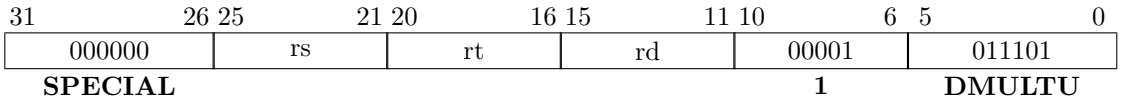
None

Overview :

Perform a multiplication on FPR. The high-order 64-bit of the result is placed into destination register.

DMULTUH — Doubleword Multiply Unsigned on High Bit

Unsigned 64-bit Multiply RESPII



Mnemonic:

DMULTUH rd, rs, rt

Function :

FPR[rd] ← FPR[rs] × FPR[rt]

Exception :

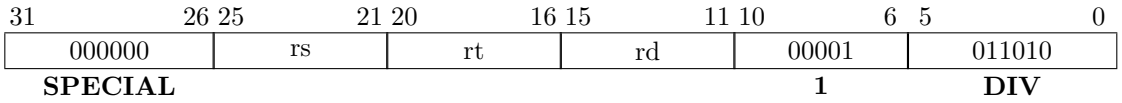
None

Overview :

Perform a unsigned multiply operation on FPRs. The high-order 64-bit of the result is placed into destination register.

REM — Reminder Word

Signed Reminder RESPII



Mnemonic:

REM rd, rs, rt

Function :

GPR[rd] ← GPR[rs] ÷ GPR[rt]

Exception :

Divide by Zero

Overview :

Perform a divide operation. The reminder of the division is placed into destination register.

REMU — Reminder Word Unsigned

Unsigned Reminder RESPII

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

Mnemonic:

REMU rd, rs, rt

Function :

GPR[rd] ← GPR[rs] ÷ GPR[rt]

Exception :

Divide by Zero

Overview :

Perform a unsigned divide operation. The reminder of the operation is placed into destination register.

DSLT — Doubleword Set on Less Than

Signed 64-bit compare RESPII

31 26	25 21	20 16	15 11	10 6	5 0
000000	rs	rt	rd	00001	101010
SPECIAL				**1**	**SLT**

Mnemonic:

DSLT rd, rs, rt

Function :

if FPR[rs] < FPR[rt] then
    FPR[rd] ← 1
else
    FPR[rd] ← 0
endif

Exception :

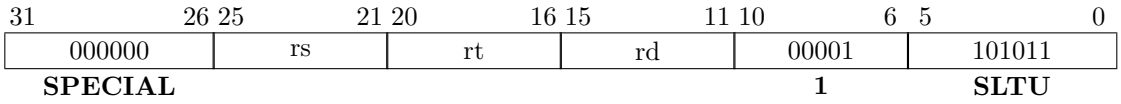
None

Overview :

Perform a compare operation on Floating-Point registers.

DSLTU — Doubleword Set on Less Than Unsigned

Unsigned 64-bit comparison RESPII



Mnemonic:

DSLTU rd, rs, rt

Function :

if FPR[rs] < FPR[rt] then
    FPR[rd] ← 1
else
    FPR[rd] ← 0
endif

Exception :

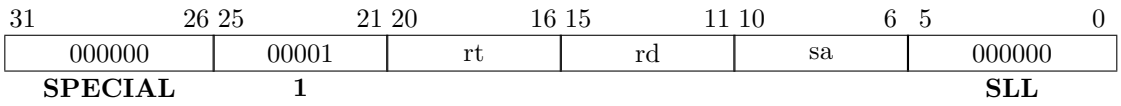
None

Overview :

Perform a unsigned compare operation on FPRs.

RTL — Rotate Left

Rotate Left RESPII



Mnemonic:

RTL rd, rt, sa

Function :

GPR[rd] ← GPR[rt] <<< sa

Exception :

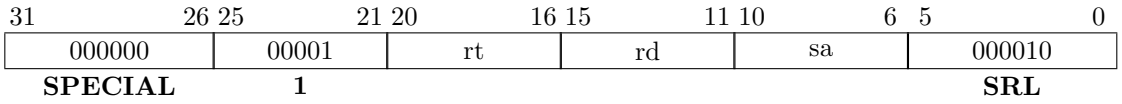
None

Overview :

Perform a left rotate operation.

RTR — Rotate Right

Rotate Right RESPII



Mnemonic:

RTR rd, rt, sa

Function :

GPR[rd] ← GPR[rt] >>> sa

Exception :

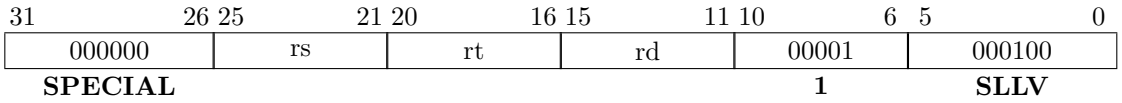
None

Overview :

Perform a right rotate operation.

RTLV — Rotate Left Variable

Rotate Left Variable RESPII



Mnemonic:

RTLV rd, rt, rs

Function :

GPR[rd] ← GPR[rt] <<< GPR[rs]

Exception :

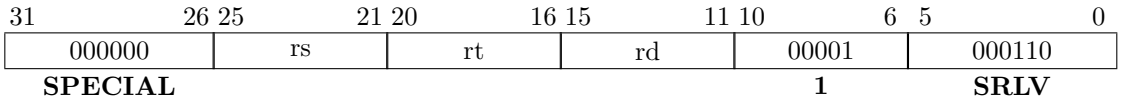
None

Overview :

Perform a left rotate operation.

RTRV — Rotate Right Variable

Rotate Right Variable RESPII



Mnemonic:

RTRV rd, rt, rs

Function :

GPR[rd] ← GPR[rt] >>> GPR[rs]

Exception :

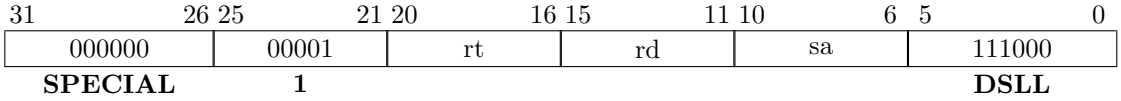
None

Overview :

Perform a right rotate operation.

DRTL — Doubleword Rotate Left

64bit Rotate Left RESPII



Mnemonic:

DRTL rd, rt, sa

Function :

FPR[rd] ← FPR[rt] <<< sa

Exception :

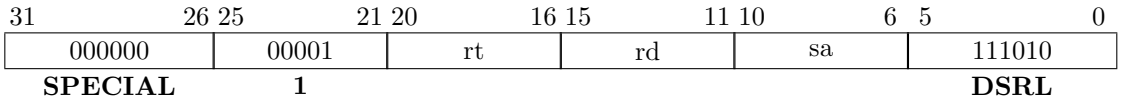
None

Overview :

Perform a 64-bit left rotate operation on FPRs.

DRTR — Doubleword Rotate Right

64bit Rotate Right RESPII



Mnemonic:

DRTR rd, rt, sa

Function :

FPR[rd] ← FPR[rt] >>> sa

Exception :

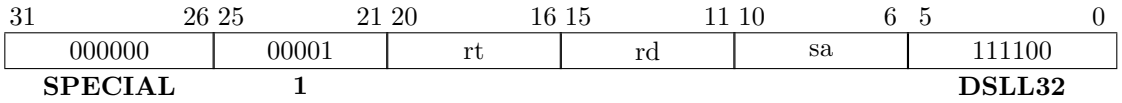
None

Overview :

Perform a right rotate operation on FPRs.

DRTL32 — Doubleword Rotate Left plus 32

64bit Rotate Left RESPII



Mnemonic:

DRTL rd, rt, sa

Function :

FPR[rd] ← FPR[rt] <<< (sa + 32)

Exception :

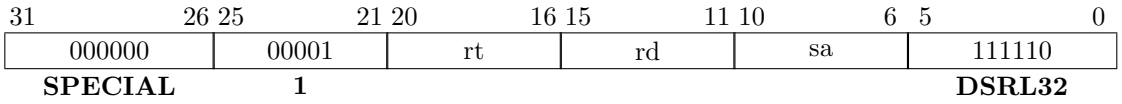
None

Overview :

Perform a left rotate operation on FPRs.

DRTR32 Doubleword Rotate Right plus 32

64bit Rotate Right RESPII



Mnemonic:

DRTR32 rd, rt, sa

Function :

FPR[rd] ← FPR[rt] >>> (sa + 32)

Exception :

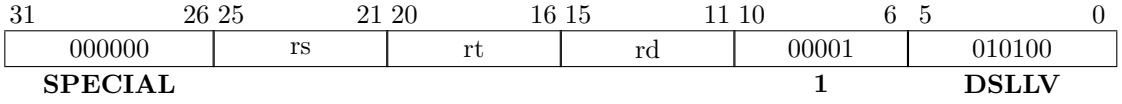
None

Overview :

Perform a right rotate operation on FPRs.

DRTLV Doubleword Rotate Left Variable

64bit Rotate Left Variable RESPII



Mnemonic:

DRTLV rd, rt, rs

Function :

FPR[rd] ← FPR[rt] <<< FPR[rs]

Exception :

None

Overview :

Perform a left rotate operation on FPRs.

DRTRV — Doubleword Rotate Right Variable

64bit Rotate Right RESPII

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

Mnemonic:

DRTRV rd, rt, rs

Function :

FPR[rd] ← FPR[rt] >>> FPR[rs]

Exception :

None

Overview :

Perform a right rotate operation on FPRs.

3.3.3 Data Transfer Instructions

MTC1H — Move Word to Floating Point on High bit

Data transfer between registers. RESPII

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

Mnemonic:

MTC1H rt, fs

Function :

FPR[fs] ← {GPR[rt], 0^{32}}

Exception :

None

Overview :

Transfer a GPR value to high-order 32-bit of a floating-point register.

MFC1H — Move Word from Floating Point on High bit

Data transfer between registers RESPII

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

Mnemonic:

MFC1H rt, fs

Function :

GPR[rt] ← high_32bit(FPR[fs])

Exception :

None

Overview :

Transfer high-order 32-bit of a floating-point register to GPR[rt].

3.3.4 System Control Instruction

MFC0 — Move from System Control Register

Move from System Control Register SYSTEM (Privilege Instruction)

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

Mnemonic:

MFC0 rt, rd

Function :

GPR[rt] ← SYSTEM[GPR[rd]]

Exception :

Coprocessor Unusable

Overview :

Move a word from system control register to GPR. The Address of the system control register is contained to GPR[rd].

MTC0 — Move to System Control Register

Move to System Control Register — SYSTEM (Privilege Instruction)

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

Mnemonic:

MTC0 rt, rd

Function :

SYSTEM[GPR[rd]] ← GPR[rt]

Exception :

Coprocessor Unusable

Overview :

Move a word to system control register from GPR. The Address of the system control register is contained to GPR[rd].

MFIMM — Move from Instruction MMU Control Register

Move from IMMU Control Register — SYSTEM (Privilege Instruction)

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

Mnemonic:

MFIMM rt, rd

Function :

GPR[rt] ← IMMU[GPR[rd]]

Exception :

Coprocessor Unusable

Overview :

Move a word from IMMU control register to GPR. The Address of the control register is contained to GPR[rd].

MTIMM Move to Instruction MMU Control Register

Move to IMMU Control Register SYSTEM (Privilege Instruction)

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

Mnemonic:

MTIMM rt, rd

Function :

IMMU[GPR[rd]] ← GPR[rt]

Exception :

Coprocessor Unusable

Overview :

Move a word to IMMU control register from GPR. The Address of the control register is contained to GPR[rd].

MFDMM Move from Data MMU Control Register

Move from DMMU Control Register SYSTEM

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

Mnemonic:

MFDMM rt, rd

Function :

GPR[rt] ← DMMU[GPR[rd]]

Exception :

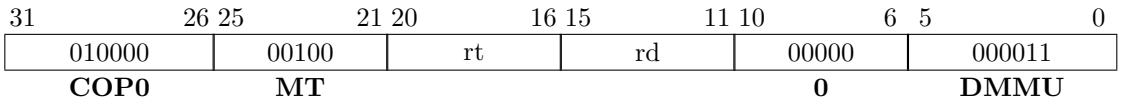
Coprocessor Unusable

Overview :

Move a word from DMMU control register to GPR. The Address of the control register is contained to GPR[rd].

MTDMM — Move to Data MMU Control Register

Move to DMMU Control Register — SYSTEM (Privilege Instruction)



Mnemonic:

MTDMM rt, rd

Function :

DMMU[GPR[rd]] ← GPR[rt]

Exception :

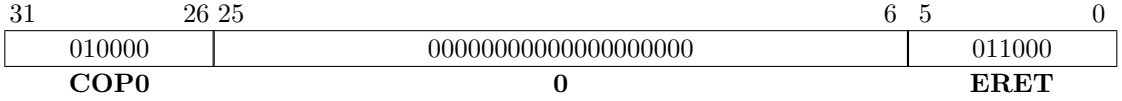
Coprocessor Unusable

Overview :

Move a word to DMMU control register from GPR. The Address of the control register is contained to GPR[rd].

ERET — Exception Return

Exception Return — SYSTEM



Mnemonic:

ERET

Function :

Exception Return

Exception :

None

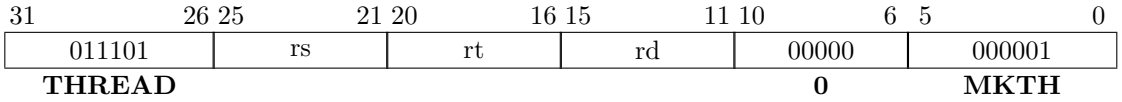
Overview :

Return from Exception.

3.3.5 Thread Control Instructio

MKTH — Make Thread

Make Thread THREAD



Mnemonic:

MKTH rd, rs, rt

Function :

make_thread(GPR[rs], GPR[rt])
if success_thread_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif

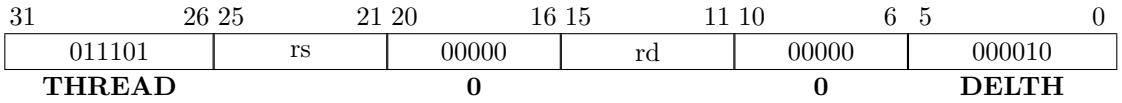
Exception :

Overview :

Make a thread. The Thread ID to make is contained in GPR[rs] and the start address of it is contained in GPR[rt]. If succeed in making the thread, GPR[rd] is set to one. It gets zero otherwise.

DELTH — Delete Thread

Delete Thread THREAD



Mnemonic:

DELTH rd, rs

Function :

```
delete_thread(GPR[rs])
if success_thread_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif
```

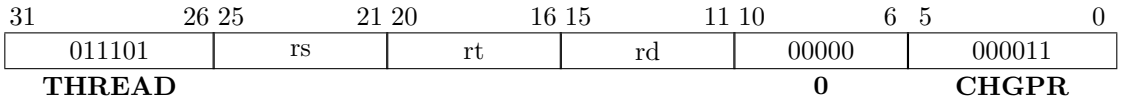
Exception :

Overview :

Delete a thread. The Thread ID to delete is contained in GPR[rs]. If succeed in deleting the thread, GPR[rd] is set to one. It gets zero otherwise.

CHGPR Change Priority

Change Priority THREAD



Mnemonic:

CHGPR rd, rs, rt

Function :

change_priority(GPR[rs] ,GPR[rd])
if success_thread_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif

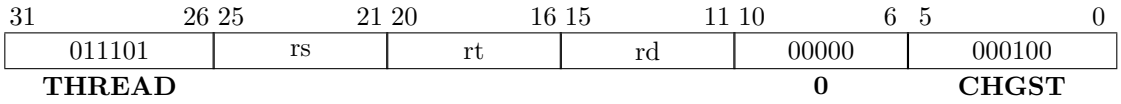
Exception :

Overview :

Change the priority of a thread. The Thread ID to change the priority is contained in GPR[rs] and the new priority is contained in GPR[rt]. If succeed in changing the priority, GPR[rd] is set to one. It gets zero otherwise.

CHGST — Change Status

Change Status THREAD



Mnemonic:

CHGST rd, rs, rt

Function :

change_status(GPR[rs], GPR[rt])
if success_thread_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif

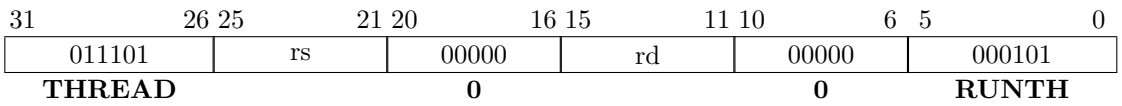
Exception :

Overview :

Change the status of a thread. The Thread ID to change the status is contained in GPR[rs] and the new status is contained in GPR[rt]. If succeed in changing the status, GPR[rd] is set to one. It gets zero otherwise.

RUNTH Run Thread

Run Thread THREAD



Mnemonic:

RUNTH rd, rs

Function :

run_thread(GPR[rs])
if success_thread_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif

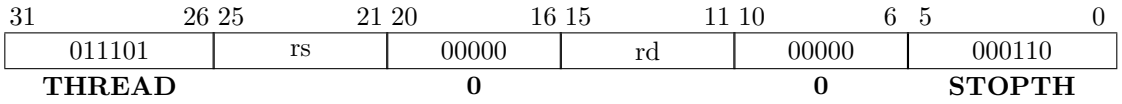
Exception :

Overview :

Run a thread. The Thread ID to run is contained in GPR[rs]. If succeed in running the thread, GPR[rd] is set to one. It gets zero otherwise.

STOPTH — Stop Thread

Stop Thread THREAD



Mnemonic:

STOPTH rd, rs

Function :

stop_thread(GPR[rs])
if success_thread_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif

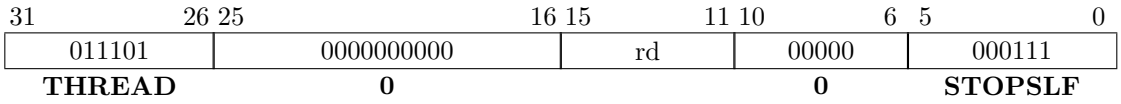
Exception :

Overview :

Stop a thread. The Thread ID to stop is contained in GPR[rs]. If succeed in stopping the thread, GPR[rd] is set to one. It gets zero otherwise.

STOPSLF Stop Myself

Stop Myself THREAD



Mnemonic:

STOPSLF rd

Function :

```
stop_myself()
if success_thread_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif
```

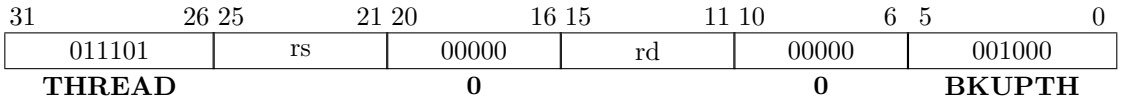
Exception :

Overview :

Stop a thread oneself. If succeed in stopping the thread, GPR[rd] is set to one. It gets zero otherwise.

BKUPTH — Backup Thread

Backup Thread THREAD



Mnemonic:

BKUPTH rd, rs

Function :

backup_thread(GPR[rs])
if success_thread_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif

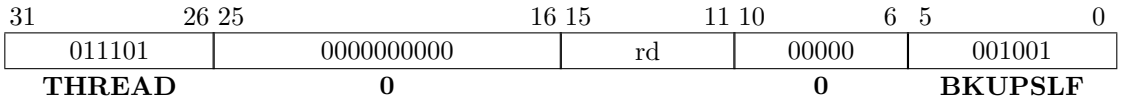
Exception :

Overview :

Backup a thread to the context cache. The Thread ID to backup is contained in GPR[rs]. If succeed in backup the thread, GPR[rd] is set to one. It gets zero otherwise.

BKUPSLF Backup Myself

Backup Myself THREAD



Mnemonic:

BKUPSLF rd

Function :

```
backup_myself()
if success_thread_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif
```

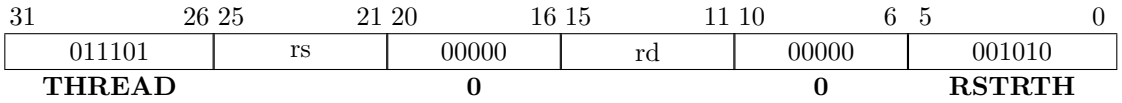
Exception :

Overview :

Backup oneself to the context cache. If succeed in backup the thread, GPR[rd] is set to one. It gets zero otherwise.

RSTRTH Restore Thread

Restore Thread THREAD



Mnemonic:

RSTRTH rd, rs

Function :

restore_thread(GPR[rs])
if success_thread_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif

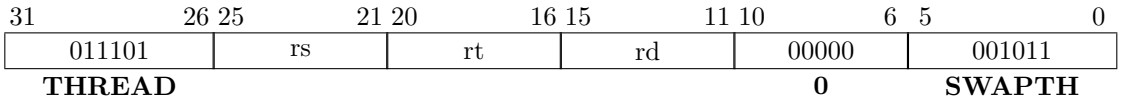
Exception :

Overview :

Restore a cached thread from the context cache. The Thread ID to restore is contained in GPR[rs]. If succeed in restoring the thread, GPR[rd] is set to one. It gets zero otherwise.

SWAPTH Swap Thread

Swap Thread THREAD



Mnemonic:

SWAPTH rd, rs, rt

Function :

swap_thread(GPR[rs], GPR[rt])
if success_thread_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif

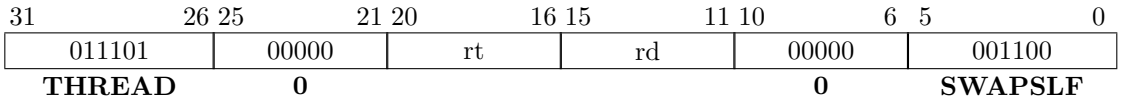
Exception :

Overview :

Swap an active thread for a cached thread. The Thread ID to backup is contained in GPR[rs] and the Thread ID to restore is contained in GPR[rt]. If succeed in swapping the threads, GPR[rd] is set to one. It gets zero otherwise.

SWAPSLF — Swap Myself

Swap Myself THREAD



Mnemonic:

SWAPSLF rd, rt

Function :

swap_myself(GPR[rt])
if success_thread_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif

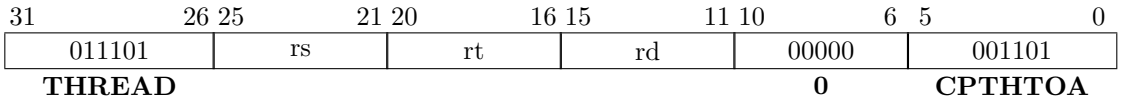
Exception :

Overview :

Swap oneself for a cached thread. The Thread ID to restore is contained in GPR[rt]. If succeed in swapping the threads, GPR[rd] is set to one. It gets zero otherwise.

CPTHTOA — Copy Thread to Active Thread

Copy Thread to Active Thread THREAD



Mnemonic:

CPTHTOA rd, rs, rt

Function :

```
copy_to_active(GPR[rs], GPR[rt])
if success_thread_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif
```

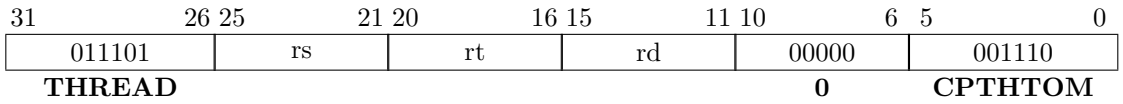
Exception :

Overview :

Copy a thread to an active thread. The Thread ID of the copy source active thread is contained in GPR[rs]. and the Thread ID of the copy destination active thread is contained in GPR[rt]. If succeed in copying the threads, GPR[rd] is set to one. It gets zero otherwise.

CPTHTOM Copy Thread to Cache Thread (Memory)

Copy Thread to Cache Thread (Memory) THREAD



Mnemonic:

CPTHTOM rd, rs, rt

Function :

copy_to_meory(GPR[rs], GPR[rt])
if success_thread_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif

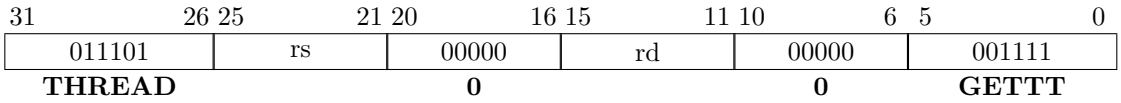
Exception :

Overview :

Copy a thread as a cached thread. The Thread ID of the copy source active thread is contained in GPR[rs]. and the Thread ID of the copy destination cached thread is contained in GPR[rt]. If succeed in copying the threads, GPR[rd] is set to one. It gets zero otherwise.

GETTT — Get Thread Table

Get Thread Table THREAD



Mnemonic:

GETTT rd, rs

Function :

GPR[rd] ← ThreadTable of GPR[rs]

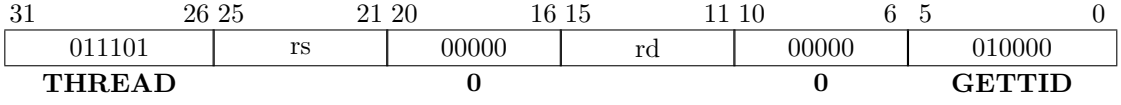
Exception :

Overview :

Get the value of a Thread Table. The Thread ID to get the Thread Table is contained in GPR[rs]. The value of the thread table is placed into GPR[rd].

GETTID — Get Thread ID

Get Thread ID THREAD



Mnemonic:

GETTID rd, rs

Function :

GPR[rd] ← ThreadID of GPR[rs]

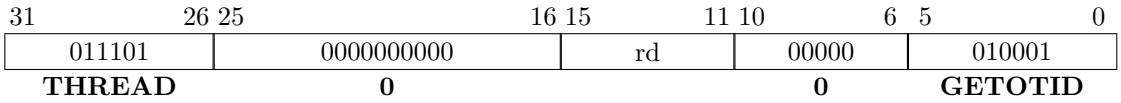
Exception :

Overview :

Get a thread's ID which is contained in a logical core whose cid is equal to GPR[rs]. (cid must be in 0 to 7, hardware only decode lower 3bit of GRP[rs].) The Thread ID is placed into GPR[rd]. If there isn't valid thread in the context, 0xffff_ffff is placed into GPR[rd]. (software cannot distinguish there is active thread which tid is 0xffff_ffff with there is no active thread.)

GETOTID — Get Own Thread ID

Get Own Thread ID THREAD



Mnemonic:

GETOTID rd

Function :

GPR[rd] ← Own ThreadID

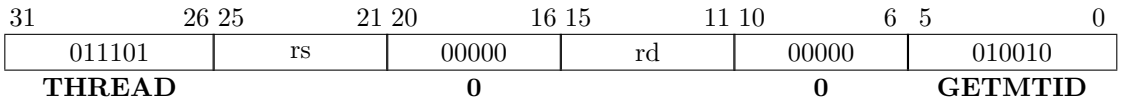
Exception :

Overview :

Get Own Thread ID. The Thread ID is placed into GPR[rd].

GETMTID — Get Cache Thread ID (Get Memory Thread ID)

Get Thread ID THREAD



Mnemonic:

GETMTID rd, rs

Function :

GPR[rd] ← ThreadID of GPR[rs]

Exception :

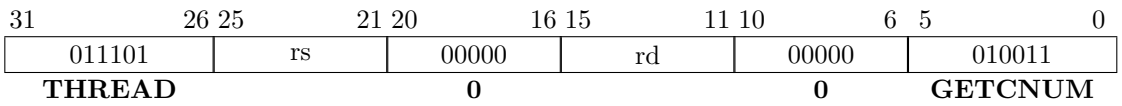
Overview :

Get a thread's id which is contained in a context cache entry whose cid is equal to GPR[rs]. (cid must be in 0 to 32, hardware only decode lower 5bit of GRP[rs].) The thread's id is placed into GPR[rd]. If there isn't valid thread in the entry, 0xffff_ffff is placed into GPR[rd]. (software cannot distinguish that there is active thread which thread id is 0xffff_ffffk with that there is no valid thread.)

If the 8 bit is equal to 1, the thread is in active thread and the 0 - 2 bit represents context ID, and if the 6 bit is equal to 0, the thread is in context cache and the 0 - 4 bit represents context ID.

GETCNUM Get Context ID Number

Get Context ID THREAD



Mnemonic:

GETCNUM rd, rs

Function :

GPR[rd] ← ContextID of GPR[rs]

Exception :

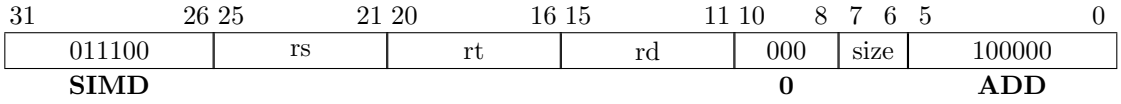
Overview :

Get a Context ID from the Thread ID. The Thread ID to get the Context ID is contained in GPR[rs]. The Context ID is placed into GPR[rd].

3.3.6 SIMD Arithmetic Instruction

SADD.size — SIMD Add

SIMD Add — SIMD



Mnemonic:

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

Function :

FPR[rd] ← FPR[rs] + FPR[rt]

Exception :

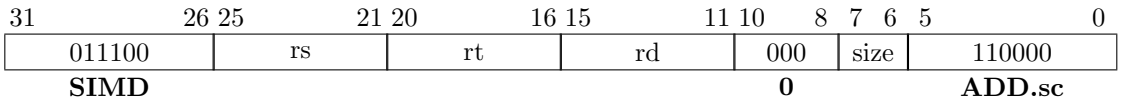
None

Overview :

64-bit integer SIMD Add by using FPR.
SADD.8: Add two packed 8-bit vaules.
SADD.16: Add two packed 16-bit vaules.
SADD.32: Add two packed 32-bit vaules.

SADD.size.sc SIMD Add Scalar

SIMD Add SIMD



Mnemonic:

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

Function :

FPR[rd] ← FPR[rs] + FPR[rt]

Exception :

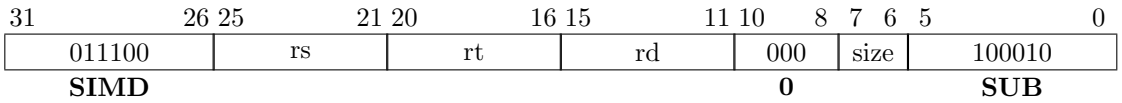
None

Overview :

64-bit integer SIMD Add by using FPR. This instruction copies the value of least significant field to each field within FPR[rt] and operates.
SADD.8.sc: Add two packed 8-bit vaules.
SADD.16.sc: Add two packed 16-bit vaules.
SADD.32.sc: Add two packed 32-bit vaules.

SSUB.size — SIMD Subtract

SIMD Subtract SIMD



Mnemonic:

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

Function :

FPR[rd] ← FPR[rs] − FPR[rt]

Exception :

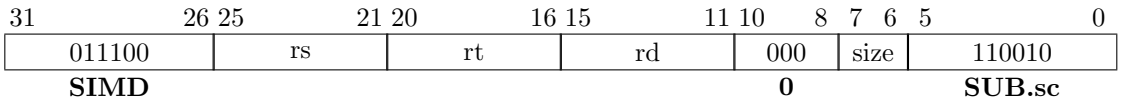
None

Overview :

64-bit integer SIMD subtract by using FPR.
SSUB.8: Subtract two packed 8-bit vaules.
SSUB.16: Subtract two packed 16-bit vaules.
SSUB.32: Subtract two packed 32-bit vaules.

SSUB.size.sc SIMD Subtract Scalar

SIMD Subtract SIMD



Mnemonic:

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

Function :

FPR[rd] ← FPR[rs] − FPR[rt]

Exception :

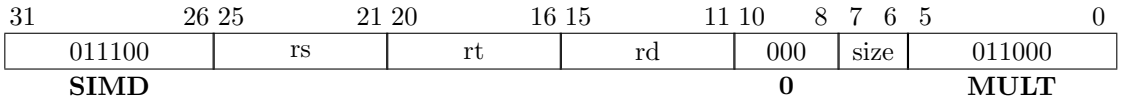
None

Overview :

64-bit integer SIMD subtract by using FPR.
This instruction copies the value of least significant field to each field within FPR[rt] and operates.
SSUB.8.sc: Subtract two packed 8-bit vaules.
SSUB.16.sc: Subtract two packed 16-bit vaules.
SSUB.32.sc: Subtract two packed 32-bit vaules.

SMULT.size — SIMD Multiply

Signed SIMD Multiply — SIMD



Mnemonic:

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

Function :

FPR[rd] ← FPR[rs] × FPR[rt]

Exception :

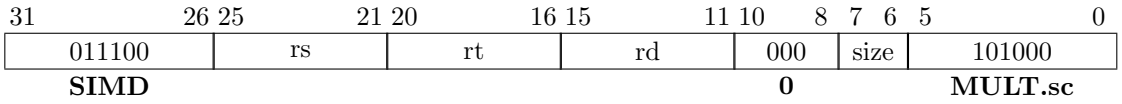
None

Overview :

64-bit integer SIMD Multiply by using FPR.
SMULT.8: Multiply two packed 8-bit vaules.
SMULT.16: Multiply two packed 16-bit vaules.
SMULT.32: Multiply two packed 32-bit vaules.

SMULT.size.sc SIMD Multiply Scalar

Singned SIMD Multiply SIMD



Mnemonic:

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

Function :

FPR[rd] ← FPR[rs] × FPR[rt]

Exception :

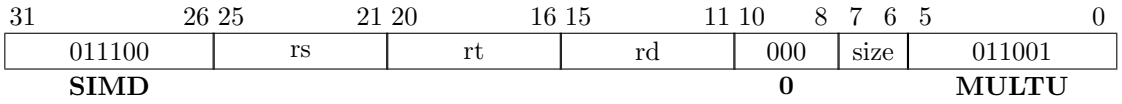
None

Overview :

64-bit integer SIMD Multiply by using FPR.
This instruction copies the value of least significant field to each field within FPR[rt] and operates.
SMULT.8.sc: Multiply two packed 8-bit vaules.
SMULT.16.sc: Multiply two packed 16-bit vaules.
SMULT.32.sc: Multiply two packed 32-bit vaules.

SMULTU.size — SIMD Multiply Unsigned

Unsigned SIMD Multiply SIMD



Mnemonic:

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

Function :

FPR[rd] ← FPR[rs] × FPR[rt]

Exception :

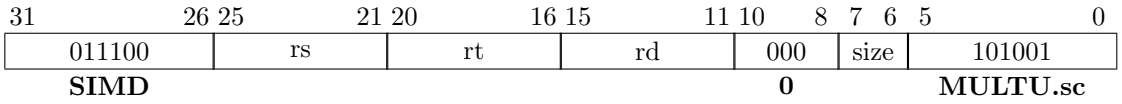
None

Overview :

64-bit integer Unsigned SIMD Multiply by using FPR.
SMULT.8: Multiply two packed 8-bit vaules.
SMULT.16: Multiply two packed 16-bit vaules.
SMULT.32: Multiply two packed 32-bit vaules.

SMULTU.size.sc SIMD Multiply Unsigned Scalar

Unsigned SIMD Multiply SIMD



Mnemonic:

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

Function :

FPR[rd] ← FPR[rs] × FPR[rt]

Exception :

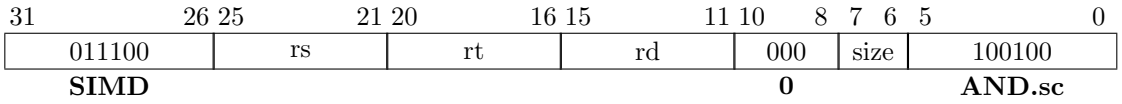
None

Overview :

64-bit integer Unsigned SIMD Multiply by using FPR.
This instruction copies the value of least significant field to each field within FPR[rt] and operates.
SMULT.8.sc: Multiply two packed 8-bit vaules.
SMULT.16.sc: Multiply two packed 16-bit vaules.
SMULT.32.sc: Multiply two packed 32-bit vaules.

SAND.size.sc SIMD And Scalar

SIMD Logical AND SIMD



Mnemonic:

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

Function :

FPR[rd] ← FPR[rs] **and** FPR[rt]

Exception :

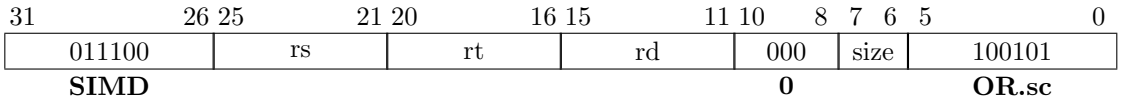
None

Overview :

64-bit integer SIMD Logical AND by using FPR.
This instruction copies the value of least significant field to each field within FPR[rt] and operates.
SAND.8.sc: Logical AND operation of two packed 8-bit vaules.
SAND.16.sc: Logical AND operation of two packed 16-bit vaules.
SAND.32.sc: Logical AND operation of two packed 32-bit vaules.

SOR.size.sc SIMD Or Scalar

SIMD Logical OR SIMD



Mnemonic:

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

Function :

FPR[rd] ← FPR[rs] **or** FPR[rt]

Exception :

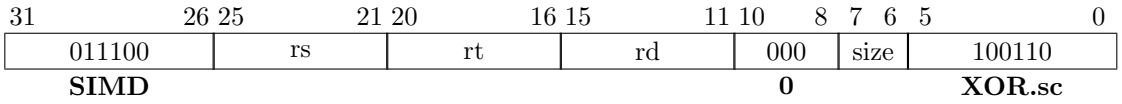
None

Overview :

64-bit integer SIMD Logical OR by using FPR.
This instruction copies the value of least significant field to each field within FPR[rt] and operates.
SOR.8.sc: Logical OR operation of two packed 8-bit vaules.
SOR.16.sc: Logical OR operation of two packed 16-bit vaules.
SOR.32.sc: Logical OR operation of two packed 32-bit vaules.

SXOR.size.sc SIMD Exclusive Or Scalar

SIMD Logical Exclusive OR SIMD



Mnemonic:

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

Function :

FPR[rd] ← FPR[rs] **xor** FPR[rt]

Exception :

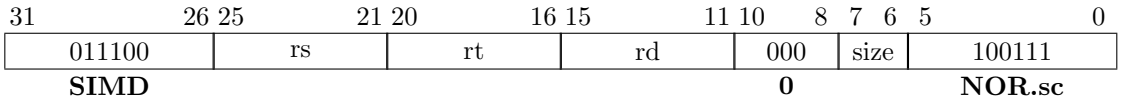
None

Overview :

64-bit integer SIMD Logical Exclusive OR by using FPR.
This instruction copies the value of least significant field to each field within FPR[rt] and operates.
SXOR.8.sc Logical Exclusive OR operation of two packed 8-bit vaules.
SXOR.16.sc Logical Exclusive OR operation of two packed 16-bit vaules.
SXOR.32.sc Logical Exclusive OR operation of two packed 32-bit vaules.

SNOR.size.sc SIMD Not Or Scalar

SIMD Logical NOT OR SIMD



Mnemonic:

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

Function :

FPR[rd] ← FPR[rs] **nor** FPR[rt]

Exception :

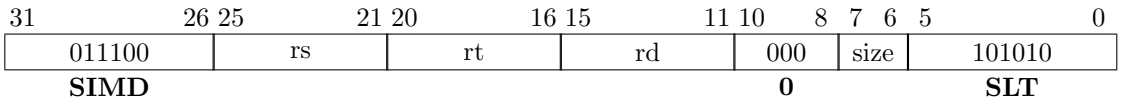
None

Overview :

64-bit integer SIMD Logical NOT OR by using FPR.
This instruction copies the value of least significant field to each field within FPR[rt] and operates.
SNOR.8.sc Logical NOT OR operation of two packed 8-bit vaules.
SNOR.16.sc Logical NOT OR operation of two packed 16-bit vaules.
SNOR.32.sc Logical NOT OR operation of two packed 32-bit vaules.

SSLT.size — SIMD Set on Less Than

SIMD Set on Less Than SIMD



Mnemonic:

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

Function :

```
if FPR[rs] < FPR[rt] then
    FPR[rd] ← 1
else
    FPR[rd] ← 0
endif
```

Exception :

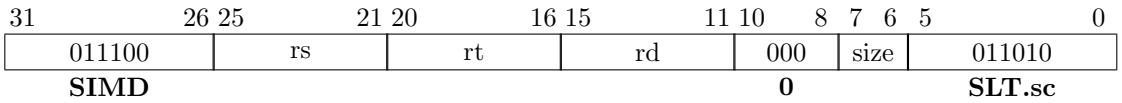
None

Overview :

SIMD Compare by using FPR.
SSLT.8: Compare two packed 8-bit vaules.
SSLT.16: Compare two packed 16-bit vaules.
SSLT.32: Compare two packed 32-bit vaules.

SSLT.size.sc SIMD Set on Less Than Scalar

SIMD Set on Less Than SIMD



Mnemonic:

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

Function :

```
if FPR[rs] < FPR[rt] then
    FPR[rd] ← 1
else
    FPR[rd] ← 0
endif
```

Exception :

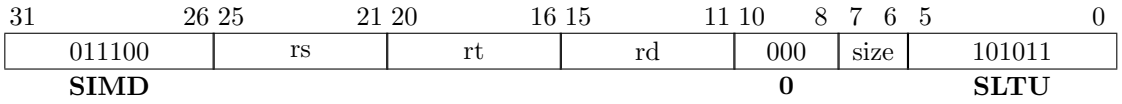
None

Overview :

SIMD Compare by using FPR.
This instruction copies the value of least significant field to each field within FPR[rt] and operates.
SSLT.8.sc: Compare two packed 8-bit vaules.
SSLT.16.sc: Compare two packed 16-bit vaules.
SSLT.32.sc: Compare two packed 32-bit vaules.

SSLTU.size — SIMD Set on Less Than Unsigned

Unsigned SIMD Set on Less Than SIMD



Mnemonic:

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

Function :

```
if FPR[rs] < FPR[rt] then
    FPR[rd] ← 1
else
    FPR[rd] ← 0
endif
```

Exception :

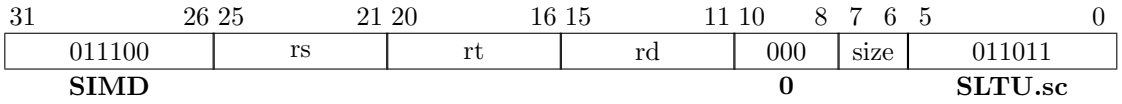
None

Overview :

SIMD Compare as unsigned values by using FPR.
SSLTU.8: Compare two packed 8-bit vaules.
SSLTU.16: Compare two packed 16-bit vaules.
SSLTU.32: Compare two packed 32-bit vaules.

SSLTU.size.sc SIMD Set on Less Than Unsigned Scalar

Unsigned SIMD Set on Less Than SIMD



Mnemonic:

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

Function :

```
if FPR[rs] < FPR[rt] then
    FPR[rd] ← 1
else
    FPR[rd] ← 0
endif
```

Exception :

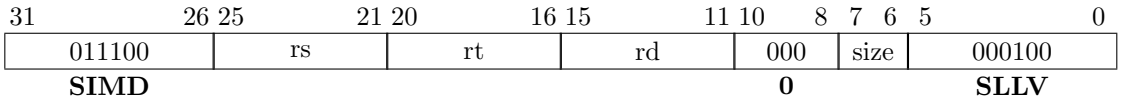
None

Overview :

SIMD Compare as unsigned values by using FPR.
This instruction copies the value of least significant field to each field within FPR[rt] and operates.
SSLTU.8.sc: Compare two packed 8-bit vaules.
SSLTU.16.sc: Compare two packed 16-bit vaules.
SSLTU.32.sc: Compare two packed 32-bit vaules.

SSLLV.size — SIMD Shift Left Logical Variable

SIMD Shift Left Logical SIMD



Mnemonic:

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

Function :

FPR[rd] ← FPR[rt] ≪ FPR[rs]

Exception :

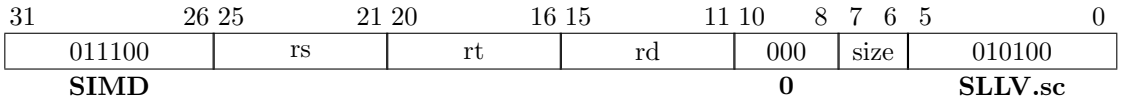
None

Overview :

SIMD Shift Left Logical by using FPR.
SSLLV.8: Shift Left Logical packed 8-bit vaule within FPR[rt].
SSLLV.16.sc: Shift Left Logical packed 16-bit vaule within FPR[rt].
SSLLV.32.sc: Shift Left Logical packed 32-bit vaule within FPR[rt].

SSLLV.size.sc SIMD Shift Left Logical Variable Scalar

SIMD Shift Left Logical SIMD



Mnemonic:

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

Function :

FPR[rd] ← FPR[rt] ≪ FPR[rs]

Exception :

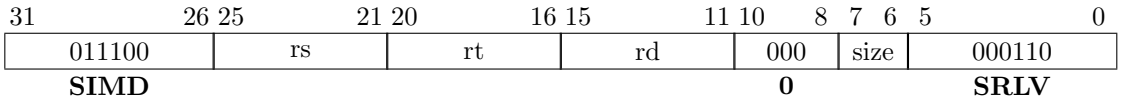
None

Overview :

SIMD Shift Left Logical by using FPR.
This instruction copies the value of least significant field to each field within FPR[rt] and operates.
SSLLV.8.sc: Shift Left Logical packed 8-bit vaule within FPR[rt].
SSLLV.16.sc: Shift Left Logical packed 16-bit vaule within FPR[rt].
SSLLV.32.sc: Shift Left Logical packed 32-bit vaule within FPR[rt].

SSRLV.size — SIMD Shift Right Logical Variable

SIMD Shift Right Logical SIMD



Mnemonic:

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

Function :

FPR[rd] ← FPR[rt] ≫ FPR[rs]

Exception :

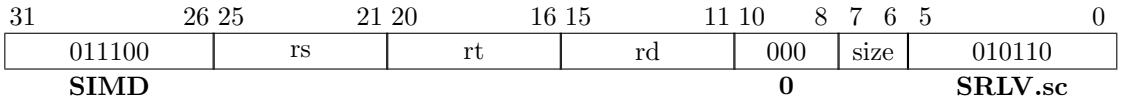
None

Overview :

SIMD Shift Right Logical by using FPR.
SSRLV.8.sc: Shift Right Logical packed 8-bit vaule within FPR[rt].
SSRLV.16.sc: Shift Right Logical packed 16-bit vaule within FPR[rt].
SSRLV.32.sc: Shift Right Logical packed 32-bit vaule within FPR[rt].

SSRLV.size.sc SIMD Shift Right Logical Variable Scalar

SIMD Shift Right Logical SIMD



Mnemonic:

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

Function :

FPR[rd] ← FPR[rt] ≫ FPR[rs]

Exception :

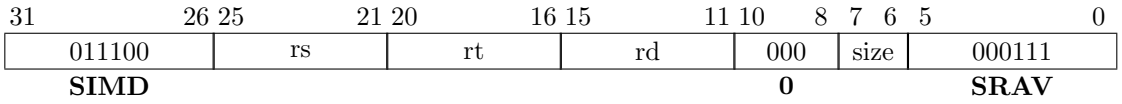
None

Overview :

SIMD Shift Right Logical by using FPR.
This instruction copies the value of least significant field to each field within FPR[rt] and operates.
SSRLV.8.sc: Shift Right Logical packed 8-bit vaule within FPR[rt].
SSRLV.16.sc: Shift Right Logical packed 16-bit vaule within FPR[rt].
SSRLV.32.sc: Shift Right Logical packed 32-bit vaule within FPR[rt].

SSRAV.size — SIMD Shift Right Arithmetic Variable

SIMD Shift Right Arithmetic SIMD



Mnemonic:

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

Function :

FPR[rd] ← FPR[rt] ≫ FPR[rs]

Exception :

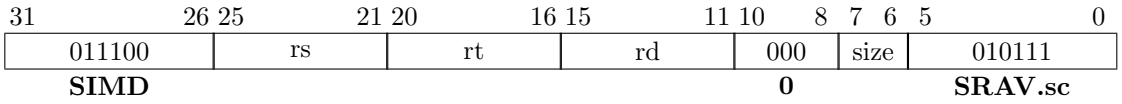
None

Overview :

SIMD Shift Right Arithmetic by using FPR.
SSRAV.8.sc: Shift Right Arithmetic packed 8-bit vaule within FPR[rt].
SSRAV.16.sc: Shift Right Arithmetic packed 16-bit vaule within FPR[rt].
SSRAV.32.sc: Shift Right Arithmetic packed 32-bit vaule within FPR[rt].

SSRAV.size.sc SIMD Shift Right Arithmetic Variable Scalar

SIMD Shift Right Arithmetic SIMD



Mnemonic:

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

Function :

FPR[rd] ← FPR[rt] ≫ FPR[rs]

Exception :

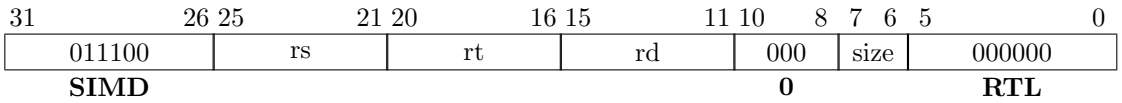
None

Overview :

SIMD Shift Right Arithmetic by using FPR.
This instruction copies the value of least significant field to each field within FPR[rt] and operates.
SSRAV.8.sc: Shift Right Arithmetic packed 8-bit vaule within FPR[rt].
SSRAV.16.sc: Shift Right Arithmetic packed 16-bit vaule within FPR[rt].
SSRAV.32.sc: Shift Right Arithmetic packed 32-bit vaule within FPR[rt].

SRTLV.size SIMD Rotate Left Variable

SIMD Rotate Left SIMD



Mnemonic:

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

Function :

FPR[rd] ← FPR[rt] <<< FPR[rs]

Exception :

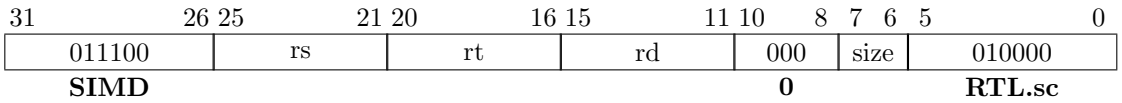
None

Overview :

SIMD Rotate Left by using FPR.
SRTLV.8.sc: Rotate Left packed 8-bit vaule within FPR[rt].
SRTLV.16.sc: Rotate Left packed 16-bit vaule within FPR[rt].
SRTLV.32.sc: Rotate Left packed 32-bit vaule within FPR[rt].

SRTLV.size.sc SIMD Rotate Left Variable Scalar

SIMD Rotate Left SIMD



Mnemonic:

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

Function :

FPR[rd] ← FPR[rt] <<< FPR[rs]

Exception :

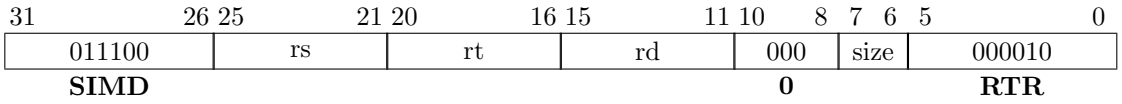
None

Overview :

SIMD Rotate Left by using FPR.
This instruction copies the value of least significant field to each field within FPR[rt] and operates.
SRTLV.8.sc: Rotate Left packed 8-bit vaule within FPR[rt].
SRTLV.16.sc: Rotate Left packed 16-bit vaule within FPR[rt].
SRTLV.32.sc: Rotate Left packed 32-bit vaule within FPR[rt].

SRTRV.size — SIMD Rotate Right Variable

SIMD Rotate Right SIMD



Mnemonic:

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

Function :

FPR[rd] ← FPR[rt] >>> FPR[rs]

Exception :

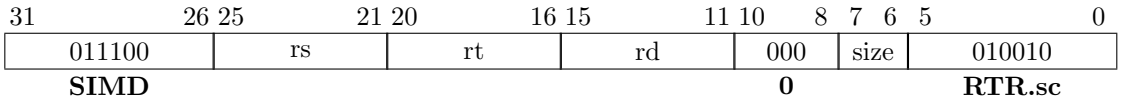
None

Overview :

SIMD Rotate Right by using FPR.
SRTRV.8.sc: Rotate Right packed 8-bit vaule within FPR[rt].
SRTRV.16.sc: Rotate Right packed 16-bit vaule within FPR[rt].
SRTRV.32.sc: Rotate Right packed 32-bit vaule within FPR[rt].

SRTRV.size.sc SIMD Rotate Right Variable Scalar

SIMD Rotate Right SIMD



Mnemonic:

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

Function :

FPR[rd] ← FPR[rt] >>> FPR[rs]

Exception :

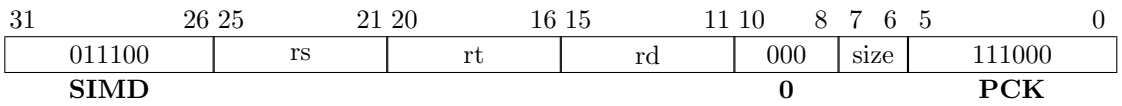
None

Overview :

SIMD Rotate Right by using FPR.
This instruction copies the value of least significant field to each field within FPR[rt] and operates.
SRTRV.8.sc: Rotate Right packed 8-bit vaule within FPR[rt].
SRTRV.16: Rotate Right packed 16-bit vaule within FPR[rt].
SRTRV.32: Rotate Right packed 32-bit vaule within FPR[rt].

PCK — Pack Data

Pack Data SIMD



Mnemonic:

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

Function :

FPR[rd] ← pack(FPR[rs], FPR[rt])

Exception :

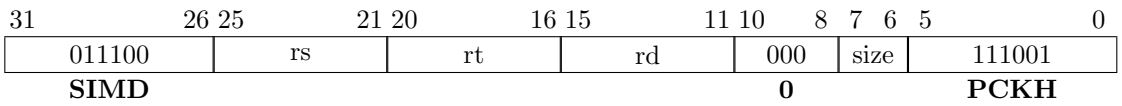
None

Overview :

Pack data in FPR[rs] and FPR[rd] to FPR[rd]. It splits the data into fields with specified size and coordinates lower half values in each field.

PCKH — Pack Data on High Bit

Pack Data SIMD



Mnemonic:

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

Function :

FPR[rd] ← packh(FPR[rs], FPR[rt])

Exception :

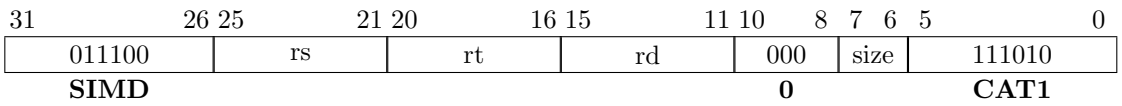
None

Overview :

Pack data in FPR[rs] and FPR[rd] to FPR[rd]. It splits the data into fields with specified size and coordinates upper half values in each field.

CAT1 Concatenate Data Type1

Concatenate Data SIMD



Mnemonic:

CAT1.8 rd, rs, rt (size = 01)
CAT1.16 rd, rs, rt (size = 10)
CAT1.32 rd, rs, rt (size = 11)

Function :

FPR[rd] ← cat1(FPR[rs], FPR[rt])

Exception :

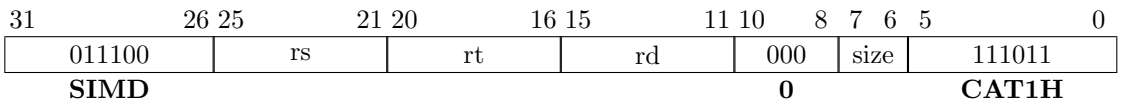
None

Overview :

Concatenate data in FPR[rs] and FPR[rd] to FPR[rd]. It splits the data into fields with specified size and concatinates them.
CAT1.8: FPR[rs].byte3, FPR[rt].byte3, FPR[rs].byte2, FPR[rt].byte2, ...
CAT1.16: FPR[rs].half1, FPR[rt].half1, FPR[rs].half0, FPR[rt].half0
CAT1.32: FPR[rs].word0, FPR[rt].word0

CAT1H — Concatenate Data Type1 on High Bit

Concatenate Data SIMD



Mnemonic:

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

Function :

FPR[rd] ← cat1h(FPR[rs], FPR[rt])

Exception :

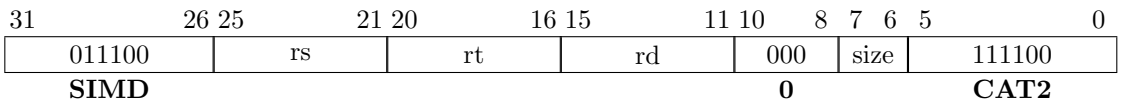
None

Overview :

Concatenate data in FPR[rs] and FPR[rd] to FPR[rd]. It splits the data into fields with specified size and concatinates them.
CAT1H.8: FPR[rs].byte7, FPR[rt].byte7, FPR[rs].byte6, FPR[rt].byte6, ...
CAT1H.16: FPR[rs].half3, FPR[rt].half3, FPR[rs].half2, FPR[rt].half2
CAT1H.32: FPR[rs].word1, FPR[rt].word1

CAT2 — Concatenate Data Type2

Concatenate Data SIMD



Mnemonic:

CAT2.8 rd, rs, rt (size = 01)
CAT2.16 rd, rs, rt (size = 10)
CAT2.32 rd, rs, rt (size = 11)

Function :

FPR[rd] ← cat2(FPR[rs], FPR[rt])

Exception :

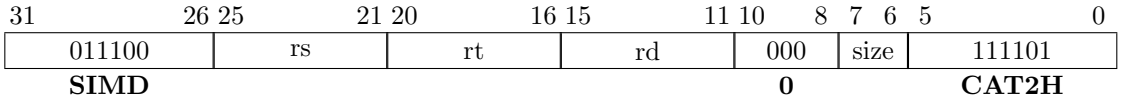
None

Overview :

Concatenate lower word in FPR[rs] and FPR[rd] to FPR[rd].

CAT2H — Concatenate Data Type2 on High Bit

Concatenate Data SIMD



Mnemonic:

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

Function :

FPR[rd] ← cat2h(FPR[rs], FPR[rt])

Exception :

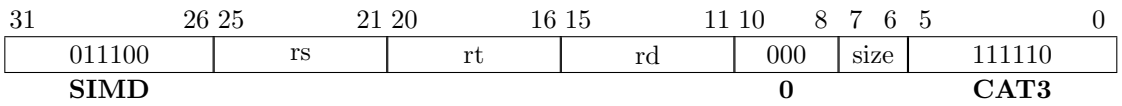
None

Overview :

Concatenate higher word in FPR[rs] and FPR[rd] to FPR[rd].

CAT3 Concatenate Data Type3

Concatenate Data SIMD



Mnemonic:

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

Function :

FPR[rd] ← cat3(FPR[rs], FPR[rt])

Exception :

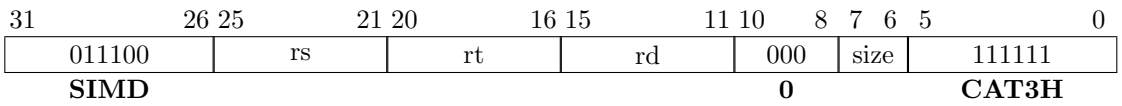
None

Overview :

Concatenate data in FPR[rs] and FPR[rd] to FPR[rd]. It splits the data into fields with specified size and concatinates them.
CAT3.8: FPR[rs].byte6, FPR[rs].byte4, ..., FPR[rt].byte6, FPR[rt].byte4, ...
CAT3.16: FPR[rs].half2, FPR[rs].half2, FPR[rt].half0, FPR[rt].half0
CAT3.32: FPR[rs].word0, FPR[rt].word0

CAT3H — Concatenate Data Type3 on High Bit

Concatenate Data SIMD



Mnemonic:

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

Function :

FPR[rd] ← cat3h(FPR[rs], FPR[rt])

Exception :

None

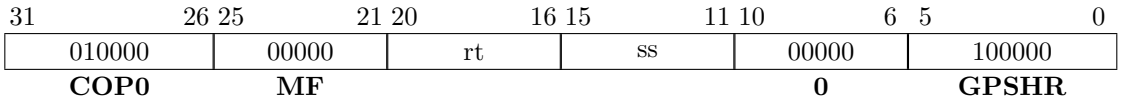
Overview :

Concatenate data in FPR[rs] and FPR[rd] to FPR[rd]. It splits the data into fields with specified size and concatinates them.
CAT3H.8: FPR[rs].byte7, FPR[rs].byte5, ..., FPR[rt].byte7, FPR[rt].byte5, ...
CAT3H.16: FPR[rs].half3, FPR[rs].half3, FPR[rt].half1, FPR[rt].half1
CAT3H.32: FPR[rs].word1, FPR[rt].word1

3.3.7 同期命令

RGPSH — Read Shared(General Purpose Register)

Synchronization Instruction SYNC



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

Mnemonic:

RGPSH rt, ss

Function :

GPR[rt] ← SHARE[ss]

Exception :

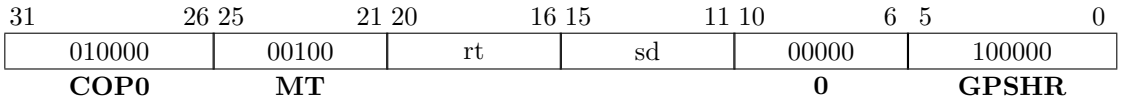
None

Overview :

共有レジスタから GPR に値を読み込む.

WGPSH — Write Shared(General Purpose Register)

Synchronization Instruction SYNC



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

Mnemonic:

WGPSH rt, sd

Function :

SHARE[sd] ← GPR[rt]

Exception :

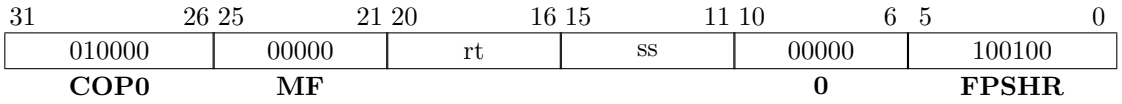
None

Overview :

GPR から共有レジスタに値を書き込む.

RFPSH — Read Shared(Floating-Point Register)

Synchronization Instruction — SYNC



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

Mnemonic:

RFPSH rt, ss

Function :

FPR[rt] ← SHARE[ss]

Exception :

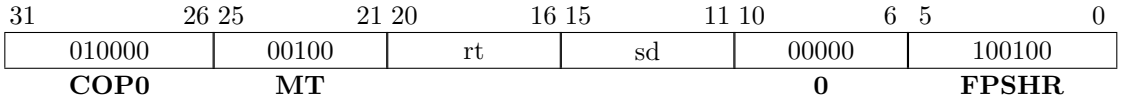
None

Overview :

共有レジスタから FPR に値を読み込む.

WFPSH — Write Shared(Floating-Point Register)

Synchronization Instruction — SYNC



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

Mnemonic:

WFPSH rt, sd

Function :

SHARE[sd] ← FPR[rt]

Exception :

None

Overview :

FPR から共有レジスタに値を書き込む.

RGPEX Read Exclusive(General Purpose Register)

Synchronization Instruction SYNC

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

Mnemonic:

RGPEX rt, ss

Function :

GPR[rt] ← SHARE[ss]

Exception :

None

Overview :

共有レジスタから GPR に ロックを確保しつつ値を読み込む.

WGPEX Write Exclusive(General Purpose Register)

Synchronization Instruction SYNC

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

Mnemonic:

WGPEX rt, sd

Function :

SHARE[sd] ← GPR[rt]

Exception :

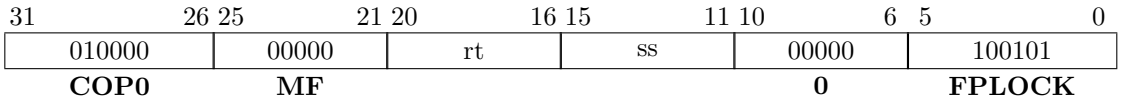
None

Overview :

GPR から共有レジスタに ロックを解放しつつ値を書き込む.

RFPEX — Read Exclusive(Floating-Point Register)

Synchronization Instruction — SYNC



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

Mnemonic:

RFPEX rt, ss

Function :

FPR[rt] ← SHARE[ss]

Exception :

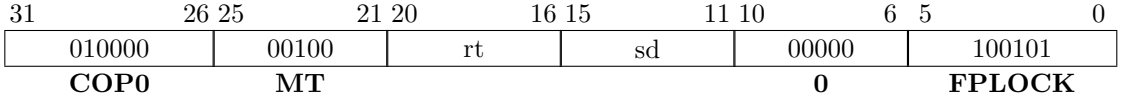
None

Overview :

共有レジスタから FPR にロックを確保しつつ値を読み込む.

WFPEX — Write Exclusive(Floating-Point Register)

Synchronization Instruction — SYNC



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

Mnemonic:

WFPEX rt, sd

Function :

SHARE[sd] ← FPR[rt]

Exception :

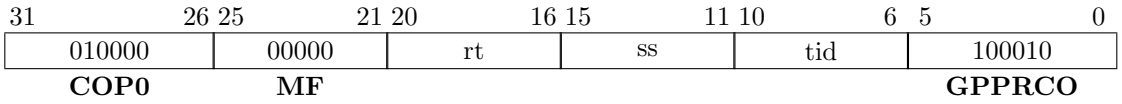
None

Overview :

FPR から共有レジスタにロックを解放しつつ値を書き込む.

GPCO Read Consumer(General Purpose Register)

Synchronization Instruction SYNC



Mnemonic:

GPCO rt, ss, tid

Function :

GPR[rt] ← SHARE[ss]

Exception :

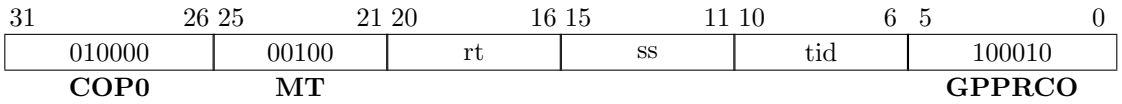
None

Overview :

共有レジスタから GPR にロックを解放しつつ値を読み込む．tid GPPR (FPPR) によりロックを確保したスレッドの ID を指定する．

GPPR Write Producer(General Purpose Register)

Synchronization Instruction SYNC



Mnemonic:

GPPR rt, sd, tid

Function :

SHARE[sd] ← GPR[rt]

Exception :

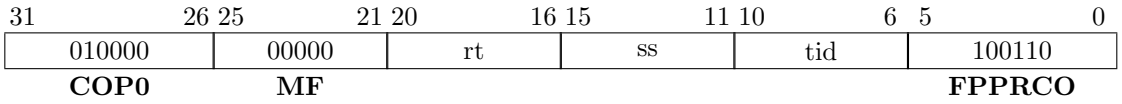
None

Overview :

GPR から共有レジスタにロックを確保しつつ値を書き込む．tid にはロックを与えるスレッドの ID を指定する．

FPCO — Read Consumer(Floating-Point Register)

Synchronization Instruction SYNC



Mnemonic:

FPCO rt, ss, tid

Function :

FPR[rt] ← SHARE[ss]

Exception :

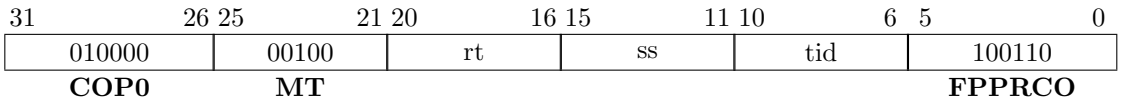
None

Overview :

共有レジスタから FPR にロックを解放しつつ値を読み込む．tid GPPR (FPPR) によりロックを確保したスレッドの ID を指定する．

FPPR — Write Producer(Floating-Point Register)

Synchronization Instruction SYNC



Mnemonic:

FPPR rt, sd, tid

Function :

SHARE[sd] ← FPR[rt]

Exception :

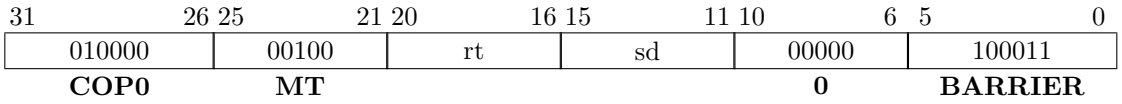
None

Overview :

FPR から共有レジスタにロックを確保しつつ値を書き込む．tid にはロックを与えるスレッドの ID を指定する．

BAR Barrier

Synchronization Instruction SYNC



Mnemonic:

BAR rt, sd

Function :

SHARE[sd] ← SHARE[sd] + 1

Exception :

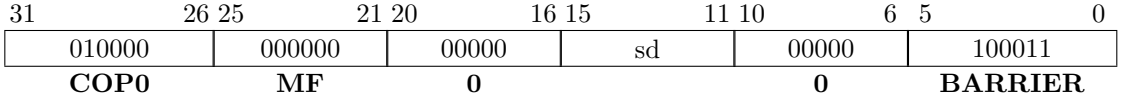
None

Overview :

共有レジスタを使用してバリア同期を行う. rt には到着を待つスレッド数を指定する.

PBAR Pre Barrier

Synchronization Instruction SYNC



Mnemonic:

PBAR sd

Function :

—

Exception :

None

Overview :

バリア同期を行うグループを指定する.

SEMLOCK Semaphore Lock(General Purpose Register)

Synchronization Instruction SYNC

31 26	25 21	20 16	15 11	10 6	5 0
010000	001000	rt	ss	00000	101001
COP0	**MF**			**0**	

Mnemonic:

SEMLOCK rt, ss

Function :

get binary semaphore lock(GPR[ss])
if succeed in getting lock then
    GPR[rt] ← 1
else
    stop thread execution
endif

Exception :

None

Overview :

バイナリセマフォレジスタのロックを確保する.

SEMREL Semaphore Release(General Purpose Register)

Synchronization Instruction SYNC

31 26	25 21	20 16	15 11	10 6	5 0
010000	001000	rt	ss	00000	101010
COP0	**MF**			**0**	

Mnemonic:

SEMREL rt, ss

Function :

release binary semaphore lock(GPR[ss])
if succeed in releasing lock then
    GPR[rt] ← 1
else
    GPR[rt] ← 0
endif

Exception :

None

Overview :

確保したセマフォレジスタのロックを開放する.

SEMTRY Semaphore Try(General Purpose Register)

Synchronization Instruction SYNC

31 26	25 21	20 16	15 11	10 6	5 0
010000	001000	rt	ss	00000	101011
COP0	**MF**			**0**	

Mnemonic:

SEMTRY rt, ss

Function :

get lock binary semaphore(GPR[ss])
if succeed in getting lock then
    GPR[rt] ← 1
else
    GPR[rt] ← 0
endif

Exception :

None

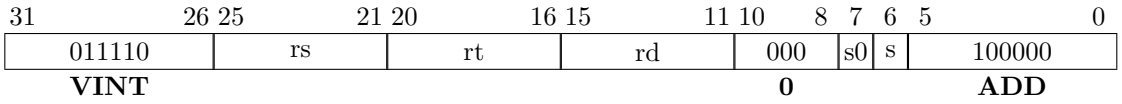
Overview :

セマフォレジスタを用いてロックを確保する.

3.3.8 Integer Vector Instructions

VADD — Vector Add

Vector Add — VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rs] + VGPR[rt]

Exception :

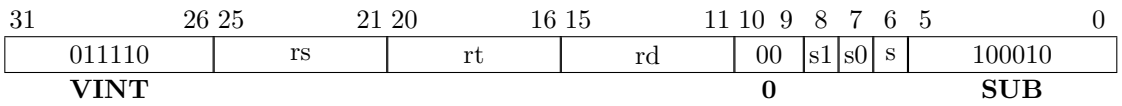
Vector Integer Exception

Overview :

Vector Add. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution.

VSUB — Vector Subtract

Vector Subtract VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rs] − VGPR[rt]

Exception :

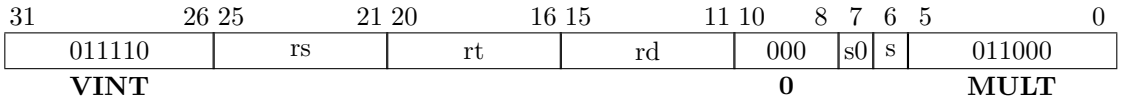
Vector Integer Exception

Overview :

Vector Subtract. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s1 is 1, execute operations by scalar register (SGPR[rs]) insted of VGPR[rs]. When sync is 1, suppress the speculative execution.

VMULT — Vector Multiply

Vector Multiply VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rs] × VGPR[rt]

Exception :

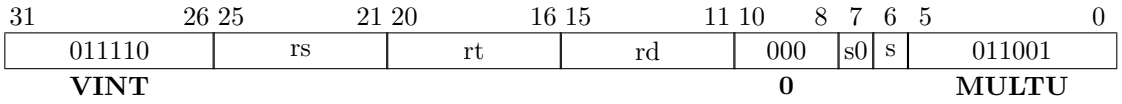
Vector Integer Exception

Overview :

Signed Vector Multiply. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. Whens sync is 1, suppress the speculative execution.

VMULTU — Vector Multiply Unsigned

Vector Multiply VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rs] × VGPR[rt]

Exception :

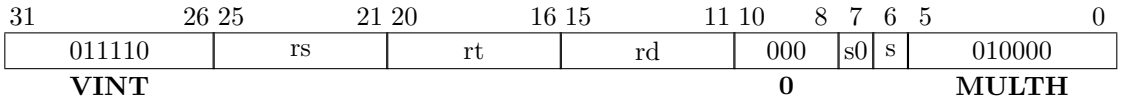
Vector Integer Exception

Overview :

Unsigned Vector Multiply. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution.

VMULTH — Vector Multiply on High Bit

Vector Multiply VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rs] × VGPR[rt]

Exception :

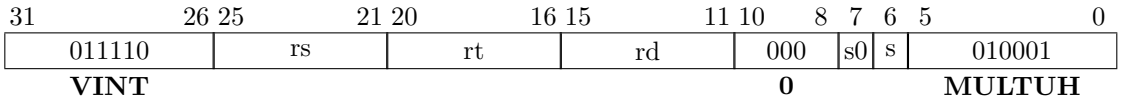
Vector Integer Exception

Overview :

Signed Vector Multiply. Upper word bit (63-32bit) of execution result is stored in VGPR[rd]. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution.

VMULTUH — Vector Multiply Unsigned on High Bit

Vector Multiply VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rs] × VGPR[rt]

Exception :

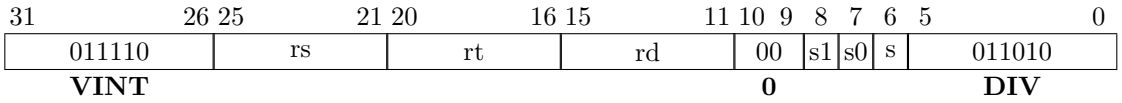
Vector Integer Exception

Overview :

Unsigned Vector Multiply. Upper word bit (63-32bit) of execution result is stored in VGPR[rd]. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution.

VDIV Vector Divide

Vector Divide VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rs] ÷ VGPR[rt]

Exception :

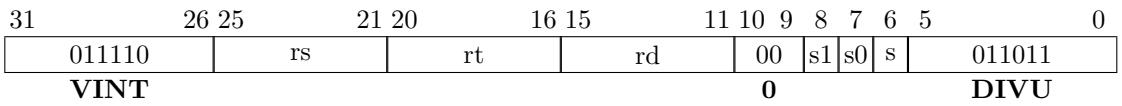
Vector Integer Exception

Overview :

Signed Vector Divide. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s1 is 1, execute oeprations by scalar register (SGPR[rs]) insted of VGPR[rs]. When sync is 1, suppress the speculative execution.

VDIVU — Vector Divide Unsigned

Vector Divede — VECTOR



Mnemonic:

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)

Function :

VGPR[rd] $\leftarrow$ VGPR[rs] $\div$ VGPR[rt]

Exception :

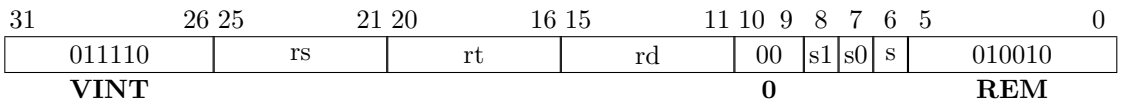
Vector Integer Exception

Overview :

Unsiged Vector Divide. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s1 is 1, execute oeprations by scalar register (SGPR[rs]) insted of VGPR[rs]. When sync is 1, suppress the speculative execution.

VREM Vector Reminder

Vector Reminder VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rs] ÷ VGPR[rt]

Exception :

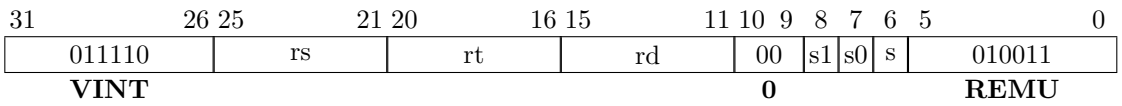
Vector Integer Exception

Overview :

Signed Vector Reminder. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s1 is 1, execute oeprations by scalar register (SGPR[rs]) insted of VGPR[rs]. When sync is 1, suppress the speculative execution.

VREMU — Vector Reminder Unsigned

Vector Reminder VECTOR



Mnemonic:

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)

Function :

VGPR[rd] $\leftarrow$ VGPR[rs] $\div$ VGPR[rt]

Exception :

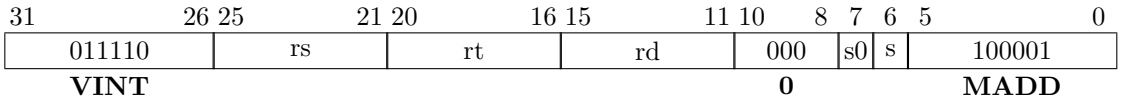
Vector Integer Exception

Overview :

Unsigned Vector Reminder. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s1 is 1, execute oeprations by scalar register (SGPR[rs]) insted of VGPR[rs]. When sync is 1, suppress the speculative execution.

VMADD — Vector Multiply and Add

Vector Multiply and Add — VECTOR



Mnemonic:

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)

Function :

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

Exception :

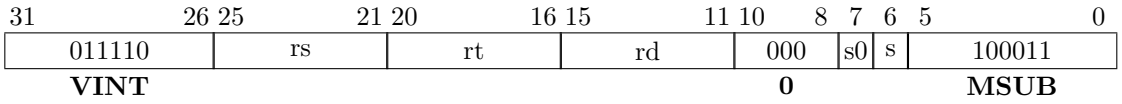
Vector Integer Exception

Overview :

Vector Multiply and Add. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution.

VMSUB — Vector Multiply and Subtract

Vector Multiply and Subtract VECTOR



Mnemonic:

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)

Function :

VGPR[rd] $\leftarrow$ VGPR[rs] $\times$ VGPR[rt] $-$ VGPR[rd]

Exception :

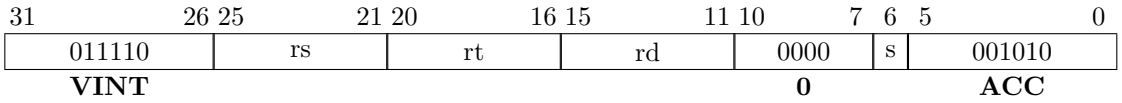
Vector Integer Exception

Overview :

Vector Multiply and Subtract. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution.

VACC Vector Accumulate

Vector Accumulate VECTOR



Mnemonic:

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

Function :

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

Exception :

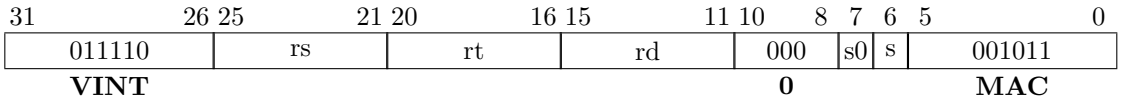
Vector Integer Exception

Overview :

Vector Accumulate. Add all elements of a vector. When sync is 1, suppress the speculative execution.

VMAC — Vector Multiply and Accumulate

Vector Multiply and Accumulate VECTOR



Mnemonic:

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)

Function :

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

Exception :

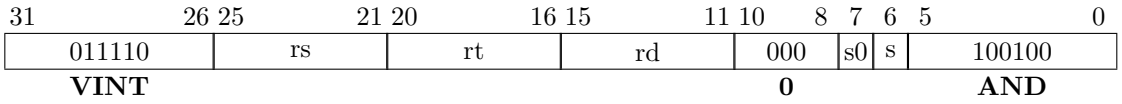
Vector Integer Exception

Overview :

Vector Multiply and Accumulate. Multiply 2 elements of vector and Add all results of them. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution.

VAND Vector And

ベクトル論理積 VECTOR



Mnemonic:

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.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

Function :

VGPR[rd] ← VGPR[rs] **and** VGPR[rt]

Exception :

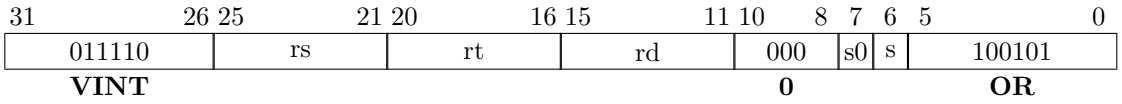
Vector Integer Exception

Overview :

Vector And. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution.

VOR — Vector Or

ベクトル論理和 VECTOR



Mnemonic:

VOR 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)

Function :

VGPR[rd] ← VGPR[rs] **or** VGPR[rt]

Exception :

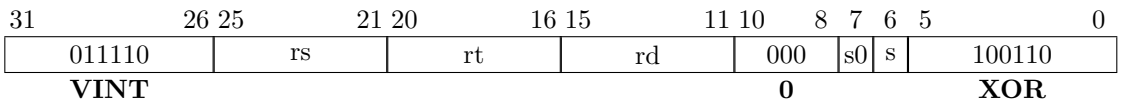
Vector Integer Exception

Overview :

Vector Or. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution.

VXOR Vector Exclusive Or

Vector Exclusive Or VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rs] **xor** VGPR[rt]

Exception :

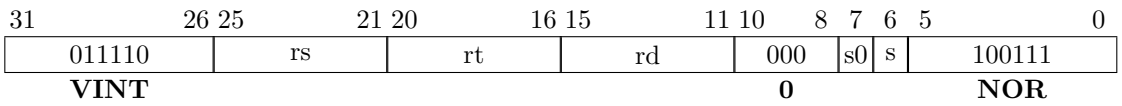
Vector Integer Exception

Overview :

Vector Exclusive Or. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution.

VNOR — Vector Not Or

Vector Not Or VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rs] **nor** VGPR[rt]

Exception :

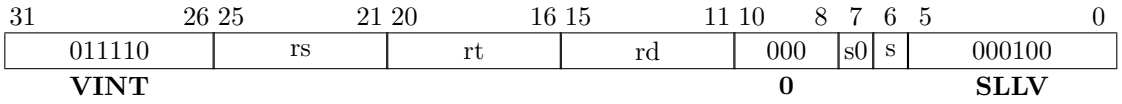
Vector Integer Exception

Overview :

Vector Not Or. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution.

VSLLV — Vector Shift Left Logical Variable

Vector Shift Left Logical Variable VECTOR



Mnemonic:

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.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

Function :

VGPR[rd] $\leftarrow$ VGPR[rt] $\ll$ VGPR[rs]

Exception :

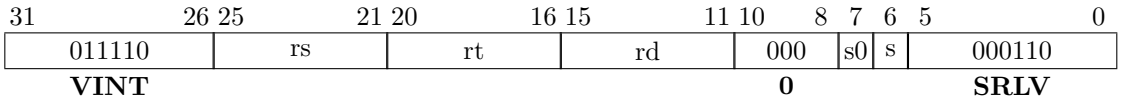
Vector Integer Exception

Overview :

Vector Shift Left Logical Variable. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution.

VSRLV — Vector Shift Right Logical Variable

Vector Shift Right Logical Variable VECTOR



Mnemonic:

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)

Function :

VGPR[rd] $\leftarrow$ VGPR[rt] $\gg$ VGPR[rs]

Exception :

Vector Integer Exception

Overview :

Vector Shift Right Logical Variable. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution.

VSRAV — Vector Shift Right Arithmetic Variable

Vector Shift Right Arithmetic Variable 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**

Mnemonic:

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)

Function :

VGPR[rd] $\leftarrow$ VGPR[rt] $\gg$ VGPR[rs]

Exception :

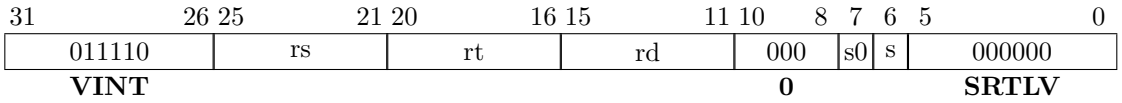
Vector Integer Exception

Overview :

Vector Shift Right Arithmetic Variable. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution.

VRTLV — Vector Rotate Left Variable

Vector Rotate Left Variable VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rt] <<< VGPR[rs]

Exception :

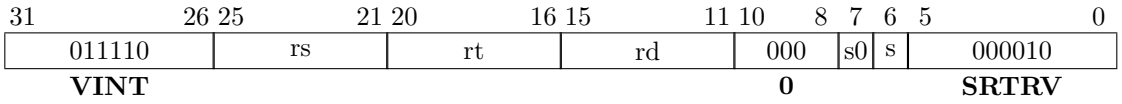
Vector Integer Exception

Overview :

Vector Rotate Left Variable. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution.

VRTRV Vector Rotate Right Variable

Vector Rotate Right Variable VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rt] >>> VGPR[rs]

Exception :

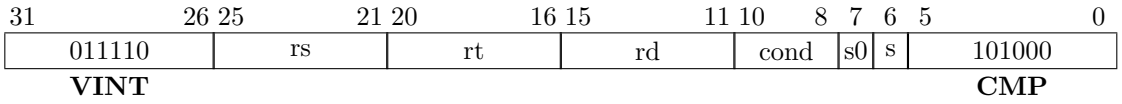
Vector Integer Exception

Overview :

Vector Rotate Right Variable. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution.

VCMP — Vector Compare

Vector Compare VECTOR



Mnemonic:

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)

Function :

```
if VGPR[rs] cond VGPR[rt] then
    VGPR[rd] ← 1
else
    VGPR[rd] ← 0
endif
```

Exception :

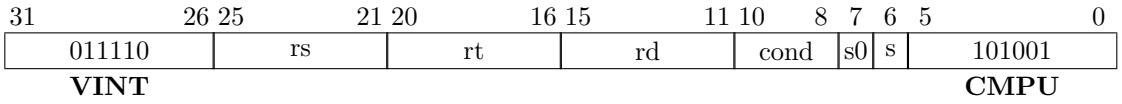
Vector Integer Exception

Overview :

Vector Compare. Set 1 or 0 to VGPR[rd] depend on cond. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution. When set up, specifiy eq(=), gt(>), lt(<), ne($\neq$), le($\leq$) or ge($\geq$) to cond.

VCMPU — Vector Compare Unsigned

Vector Compare VECTOR



Mnemonic:

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)

Function :

```
if VGR[rs] cond VGPR[rt] then
    VGPR[rd] ← 1
else
    VREG[rd] ← 0
endif
```

Exception :

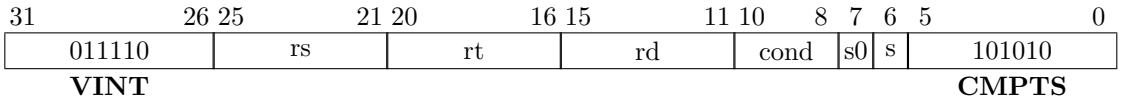
Vector Integer Exception

Overview :

Vector Compare Unsigned. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution. When set up, specifiy eq(=), gt(>), lt(<), ne($\neq$), le($\leq$) or ge($\geq$) to cond.

VCMPTS — Vector Compare to Scalar Register

Vector Copare VECTOR



Mnemonic:

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)

Function :

```
if VGPR[rs] cond VGPR[rt] then
    SGPR[rd] ← 1
else
    SGPR[rd] ← 0
endif
```

Exception :

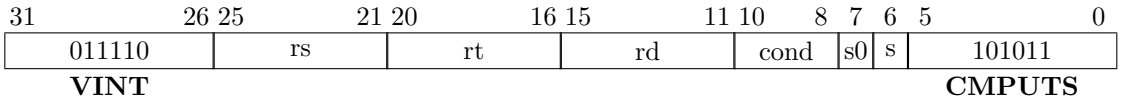
Vector Integer Exception

Overview :

Vector Compare to Scalar Register. Results of each elements are store in each bit of scalar register. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution. When set up, specifiy eq(=), gt(>), lt(<), ne($\neq$), le($\leq$) or ge($\geq$) to cond.

VCMPUTS Vector Compare Unsigned to Scalar Register

Vector Compare VECTOR



Mnemonic:

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)

Function :

```
if VGPR[rs] cond VGPR[rt] then
    SGPR[rd] ← 1
else
    SGPR[rd] ← 0
endif
```

Exception :

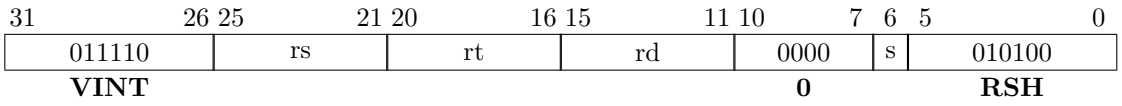
Vector Integer Exception

Overview :

Vector Compare Unsigned to Scalar Register. Results of each elements are stored in each bit of scalar register. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution. When set up, specifiy eq(=), gt(>), lt(<), ne($\neq$), le($\leq$) or ge($\geq$) to cond.

VIRSH — Vector Register Shift

Vector Register Shift — VECTOR (After MRMTP2)



Mnemonic:

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

Function :

VREG[rd][i] ← VREG[rs][i−SREG[rt]]

Exception :

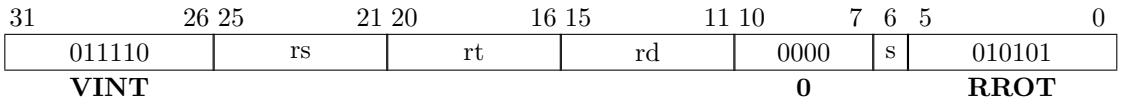
Vector Integer Exception

Overview :

Vector Register Shift. Shift elements of vector with SREG[rt]. Shift value is available also negative number. If elements number of source register is negative number or over vector length, set 0 to destination register. When sync is 1, suppress the speculative execution.

VIRROT — Vector Register Rotation

Vector Register Rotation — VECTOR (After MRMTP2)



Mnemonic:

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

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

Function :

VREG[rd][i] ← VREG[rs][(i−SREG[rt]) mod LENGTH]

Exception :

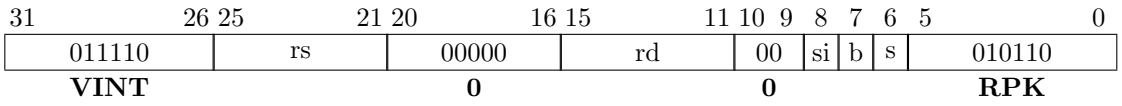
Vector Integer Exception

Overview :

Vector Register Rotation. Rotate elements of vector with SREG[rt]. Rotate value is available also negative number. When sync is 1, suppress the speculative execution.

VIRPK — Vector Register Packing

Vector Register Packing — VECTOR (After MRMTP2)



Mnemonic:

VIRPK rd, rs	(sync(s) = 0, high(h) = 0, byte(b) = 0)
VIRPK.sy rd, rs	(sync(s) = 1, high(h) = 0, byte(b) = 0)
VIRPK.b rd, rs	(sync(s) = 0, high(h) = 0, byte(b) = 1)
VIRPK.b.sy rd, rs	(sync(s) = 1, high(h) = 0, byte(b) = 1)
VIRPK.h rd, rs	(sync(s) = 0, high(h) = 1, byte(b) = 0)
VIRPK.h.sy rd, rs	(sync(s) = 1, high(h) = 1, byte(b) = 0)
VIRPK.b.h rd, rs	(sync(s) = 0, high(h) = 1, byte(b) = 1)
VIRPK.b.h.sy rd, rs	(sync(s) = 1, high(h) = 1, byte(b) = 1)

Function :

VREG[rd][i/2] ← { VREG[rs][i]$_{15...0}$, VREG[rs][i+1]$_{15...0}$} (high = 0, byte = 0)
VREG[rd][i/2] ← { VREG[rs][i]$_{31...16}$, VREG[rs][i+1]$_{31...16}$} (high = 1, byte = 0)
VREG[rd][i/2] ← { VREG[rs][i]$_{23...16}$, VREG[rs][i+1]$_{23...16}$, VREG[rs][i]$_{7...0}$, VREG[rs][i+1]$_{7...0}$} (high = 0, byte = 1)
VREG[rd][i/2] ← { VREG[rs][i]$_{31...24}$, VREG[rs][i+1]$_{31...24}$, VREG[rs][i]$_{15...8}$, VREG[rs][i+1]$_{15...8}$} (high = 1, byte = 1)

Exception :

Vector Integer Exception

Overview :

Vector Register Packing. Pack elements of vector i and i + 1 to destination register i. Result of this instruction, length of destination register become a half. When sync is 1, suppress the speculative execution.

VIRUPK Vector Register Unpacking

Vector Register Unpacking VECTOR (After MRMTP2)

31 26	25 21	20 16	15 11	10 9	8	7	6	5 0
011110	rs	00000	rd	00	h	b	s	010111
VINT		**0**		**0**				**RUPK**

Mnemonic:

VIRUPK rd, rs	(sync(s) = 0, signed(si) = 0, byte(b) = 0)
VIRUPK.sy rd, rs	(sync(s) = 1, signed(si) = 0, byte(b) = 0)
VIRUPK.b rd, rs	(sync(s) = 0, signed(si) = 0, byte(b) = 1)
VIRUPK.b.sy rd, rs	(sync(s) = 1, signed(si) = 0, byte(b) = 1)
VIRUPK.s rd, rs	(sync(s) = 0, signed(si) = 1, byte(b) = 0)
VIRUPK.s.sy rd, rs	(sync(s) = 1, signed(si) = 1, byte(b) = 0)
VIRUPK.b.s rd, rs	(sync(s) = 0, signed(si) = 1, byte(b) = 1)
VIRUPK.b.s.sy rd, rs	(sync(s) = 1, signed(si) = 1, byte(b) = 1)

Function :

VREG[rd][i*2] ← zero_ext(VREG[rs][i]$_{31...16}$),
VREG[rd][i*2+1] ← zero_ext(VREG[rs][i]$_{15...0}$) (signed = 0, byte = 0)
VREG[rd][i*2] ← sign_ext(VREG[rs][i]$_{31...16}$),
VREG[rd][i*2+1] ← sign_ext(VREG[rs][i]$_{15...0}$) (signed = 1, byte = 0)
VREG[rd][i*2] ← { zero_ext(VREG[rs][i]$_{31...24}$), zero_ext(VREG[rs][i]$_{15...8}$)},
VREG[rd][i*2+1] ← { zero_ext(VREG[rs][i]$_{23...16}$), zero_ext(VREG[rs][i]$_{7...0}$)} (signed = 0, byte = 1)
VREG[rd][i*2] ← { sign_ext(VREG[rs][i]$_{31...24}$), sign_ext(VREG[rs][i]$_{15...8}$)},
VREG[rd][i*2+1] ← { sign_ext(VREG[rs][i]$_{23...16}$), sign_ext(VREG[rs][i]$_{7...0}$)} (signed = 1, byte = 1)

Exception :

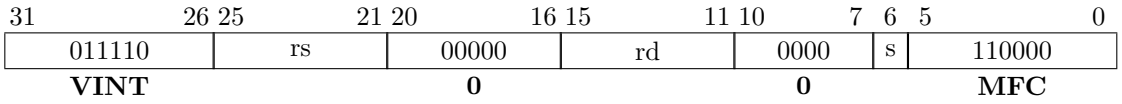
Vector Integer Exception

Overview :

Vector Register Unpacking. Unpack an element of vector i to destination register 2 * i and 2 * i + 1. Result of this instruction, length of destination register become a double. When sync is 1, suppress the speculative execution.

VIMFC — Move from Vector Integer Control Register

Control Register Read — 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**

Mnemonic:

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

Function :

GPR[rd] ← VICTRL[rs]

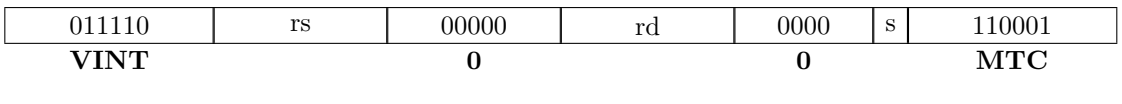
Exception :

Overview :

Move from Vector Integer Control Register. Save the value from control register which is assigned by rs to GPR. When sync is 1, suppress the speculative execution.

VIMTC — Move to Vector Integer Control Register

Control Register Write — 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**

Mnemonic:

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

Function :

VICTRL[rd] ← GPR[rs]

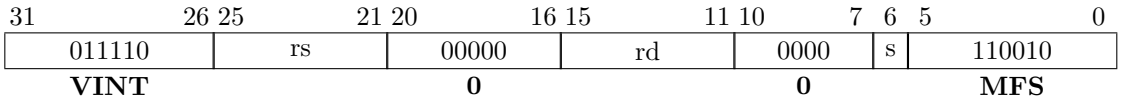
Exception :

Overview :

Move to Vector Integer Control Register. Save the value from GPR to control register which is assigned by rd. When sync is 1, suppress the speculative execution.

VIMFS Move from Vector Integer Scalar Register

Integer Scalar Register Read VECTOR



Mnemonic:

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

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

Function :

GPR[rd] ← SGPR[rs]

Exception :

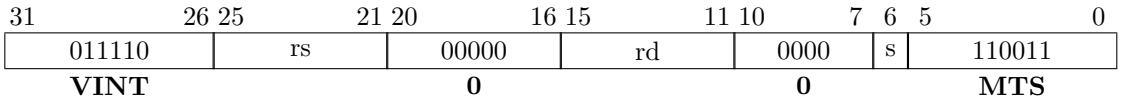
Vector Integer Exception

Overview :

Move from Vector Integer Scalar Register. Save the value from integer scalar register which is assigned by rs to GPR. When sync is 1, suppress the speculative execution.

VIMTS — Move to Vector Integer Scalar Register

Integer Scalar Register Write — VECTOR



Mnemonic:

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

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

Function :

SGPR[rd] ← GPR[rs]

Exception :

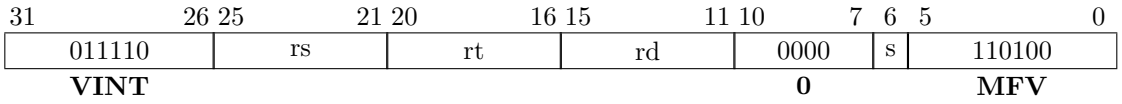
Vector Integer Exception

Overview :

Move to Vector Integer Scalar Register. Save the value from GPR to integer scalar register which is assigned by rd. When sync is 1, suppress the speculative execution.

VIMFV Move from Vector Integer Vector Register

Integer Vector Register Read VECTOR



Mnemonic:

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

Function :

SGPR[rd] ← VGPR[rs][rt]

Exception :

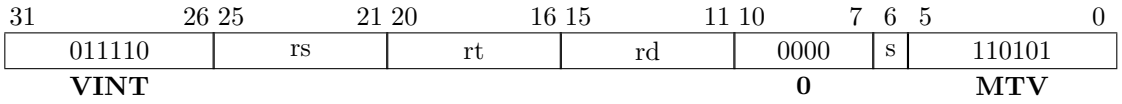
Vector Integer Exception

Overview :

Move from Vector Integer Vector Register. Save the value from rt th element of vector register which is assigned by rs to integer scalar register. When sync is 1, suppress the speculative execution.

VIMTV Move to Vector Integer Vector Register

Integer Vector Register Write VECTOR



Mnemonic:

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

Function :

VGPR[rd][rt] ← SGPR[rs]

Exception :

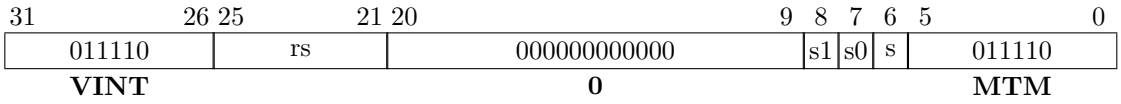
Vector Integer Exception

Overview :

Move to Vector Integer Vector Register. Save the value from integer scalar register from rt th element of vector register which is assigned by rs. When sync is 1, suppress the speculative execution.

VIMTM Move to Vector Integer Mask Register

Integer Vector Mask Register Write VECTOR



Mnemonic:

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)

Function :

VICTRL[*Mask Regieter*] ← SGPR[rs]

Exception :

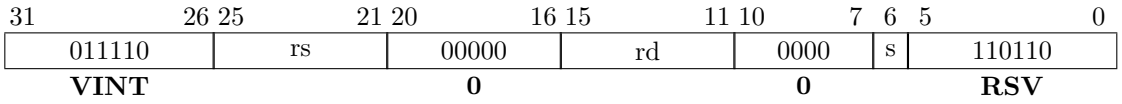
Vector Integer Exception

Overview :

Move to Vector Integer Mask Register. Save the value from integer scalar register which is assigned by rs to mask register. When s0 is 1, save the value to lower half of mask register. When s1 is 1, save the value to upper half of mask register. When sync is 1, suppress the speculative execution.

VIRSV Vector Integer Register Reserve

Vector Integer Register Reserve VECTOR



Mnemonic:

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

Function :

```
reserve_vector_register(GPR[rs])
if success_reserve_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif
```

Exception :

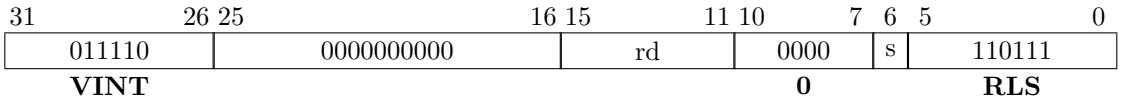
None

Overview :

Vector Integer Register Reserve. Set the value which assign construction of register reserve to GPR[rs]. When reservation is success, set 1 to GPR[rd], otherwise set 0. When sync is 1, suppress the speculative execution.

VIRLS Vector Integer Register Release

Vector Integer Register Release VECTOR



Mnemonic:

VIRLS rd (sync(s) = 0)

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

Function :

```
release_vector_register()
if success_release_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif
```

Exception :

None

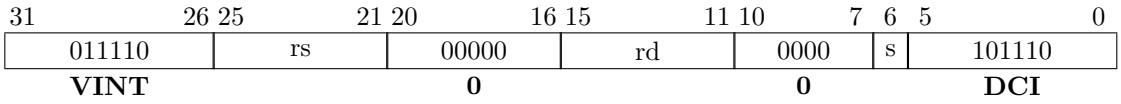
Overview :

Vector Integer Register Release. When release is success, set 1 to GPR[rd], otherwise set 0. When sync is 1, suppress the speculative execution.

i

VIDCI Vector Integer Define Compound Instruction

Vector Integer Define Compound Instruction VECTOR



Mnemonic:

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

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

Function :

VICPD[rd] ← GPR[rs]

Exception :

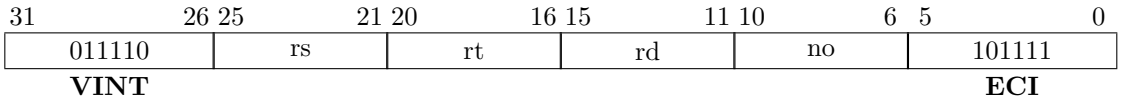
Vector Integer Exception

Overview :

Vector Integer Define Compound Instruction. Store instruction in GPR[rs] to entry which is assigned by rd of compound instruction buffer When sync is 1, suppress the speculative execution.

VIECI Vector Integer Execute Compound Instruction

Vector Integer Execute Compound Instruction VECTOR



Mnemonic:

VIECI rd, rs, rt, no

Function :

VGPR[rd] ← VGPR[rs] op VGPR[rt]

Exception :

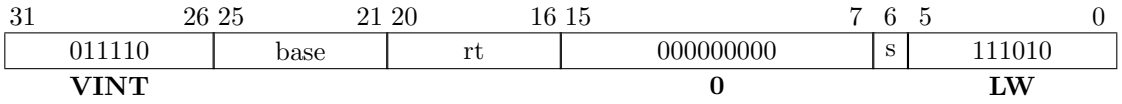
Vector Integer Exception

Overview :

Vector Integer Execute Compound Instruction. Execute instructions on compound instruction buffer from the address which is assigned by no.

VILW — Vector Integer Load Word

Vector Integer Load Word VECTOR



Mnemonic:

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

Function :

VGPR[rt] ← MEM.WORD[GPR[base]]

Exception :

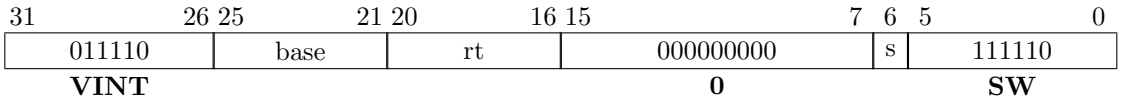
Vector Integer Exception
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Load)

Overview :

Vector Integer Load Word. Load from memory to vector integer register. When sync is 1, suppress the speculative execution.

VISW — Vector Integer Store Word

Vector Integer Store Word VECTOR



Mnemonic:

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

Function :

MEM.WORD[GPR[base]] ← VGPR[rt]

Exception :

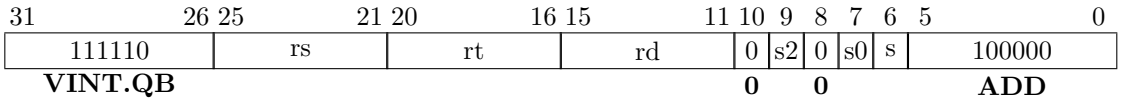
Vector Integer Exception
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Store)

Overview :

Vectro Integer Store Word. Store from vector integer register to memory. When sync is 1, suppress the speculative execution.

VADD.QB Vector Add Quad Byte

Vector Add Quad Byte VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rs] + VGPR[rt]

Exception :

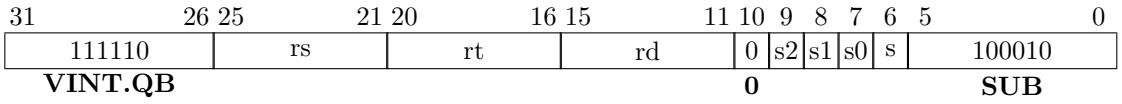
Vector Integer Exception

Overview :

8 bit * 4 Vector Add. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VSUB.QB Vector Subtract Quad Byte

Vector Subtract Quad Byte VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rs] - VGPR[rt]

Exception :

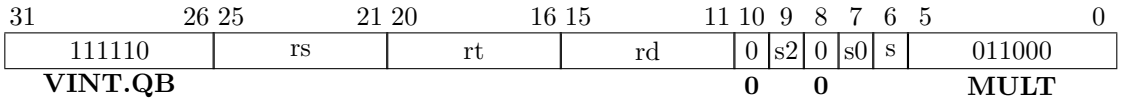
Vector Integer Exception

Overview :

8 bit * 4 Vector Subtract. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VMULT.QB — Vector Multiply Quad Byte

Vector Multiply Quad Byte VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rs] × VGPR[rt]

Exception :

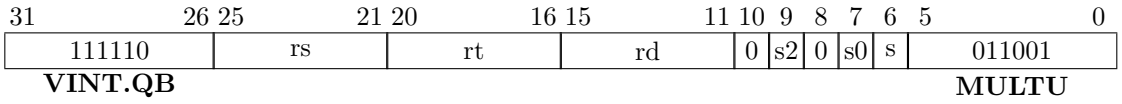
Vector Integer Exception

Overview :

8 bit * 4 Vector Multiply. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VMULTU.QB Vector Multiply Unsigned

Vector Multiply Unsigned VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rs] × VGPR[rt]

Exception :

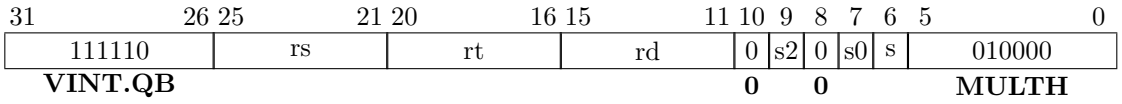
Vector Integer Exception

Overview :

8 bit * 4 Vector Multiply Unsigned. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VMULTH.QB — Vector Multiply Quad Byte on High Bit

Vector Multiply Quad Byte VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rs] × VGPR[rt]

Exception :

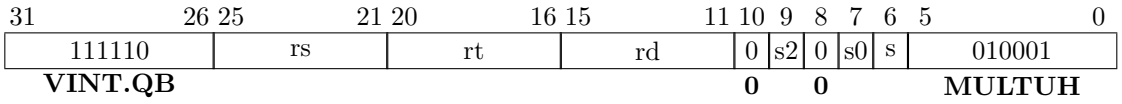
Vector Integer Exception

Overview :

8 bit * 4 Vector Multiply. Store upper half of results of execution to VGPR[rd]. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VMULTUH.QB Vector Multiply Unsigned Quad Byte on High Bit

Vector Multiply Unsigned Quad Byte VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rs] × VGPR[rt]

Exception :

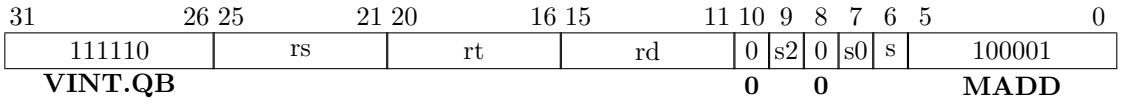
Vector Integer Exception

Overview :

8 bit * 4 Vector Multiply Unsigned. Store upper half of results of execution to VGPR[rd]. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VMADD.QB — Vector Multiply and Add Quad Byte

Vector Multiply and Add Quad Byte VECTOR



Mnemonic:

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)

Function :

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

Exception :

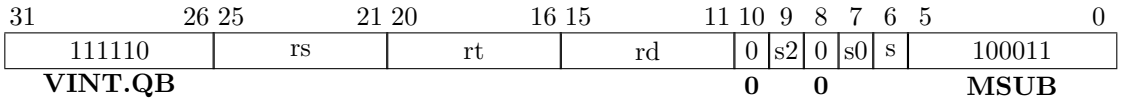
Vector Integer Exception

Overview :

8 bit * 4 Vector Multiply and Add. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VMSUB.QB Vector Multiply and Subtract Quad Byte

Vector Multiply and Subtract Quad Byte VECTOR



Mnemonic:

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)

Function :

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

Exception :

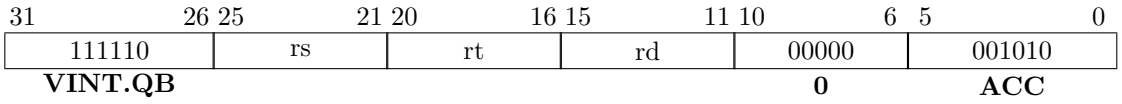
Vector Integer Exception

Overview :

8 bit * 4 Vector Multiply and Subtract. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VACC.QB Vector Accumulate Quad Byte

Vector Accumulate Quad Byte VECTOR



Mnemonic:

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

Function :

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

Exception :

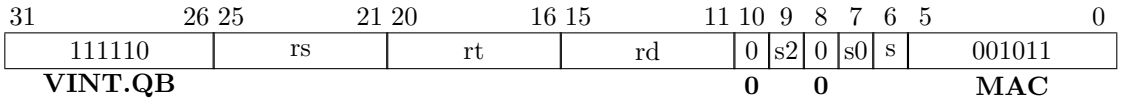
Vector Integer Exception

Overview :

8 bit * 4 Vector Accumulate. Add all elements of vector. When sync is 1, suppress the speculative execution.

VMAC.QB Vector Multiply and Accumulate Quad Byte

Vector Multiply and Accumulate Quad Byte VECTOR



Mnemonic:

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)

Function :

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

Exception :

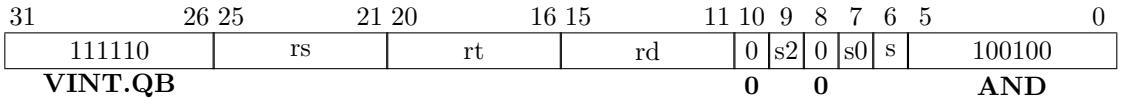
Vector Integer Exception

Overview :

8 bit * 4 Vector Multiply and Accumulate. Multiply elements of 2 vector and add all elements of these result. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VAND.QB — Vector And Quad Byte

Vector And Quad Byte VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rs] **and** VGPR[rt]

Exception :

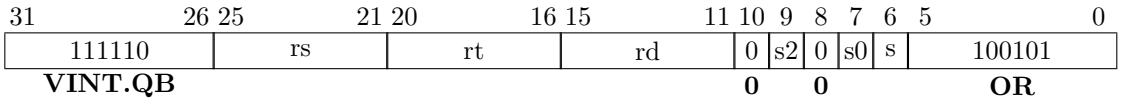
Vector Integer Exception

Overview :

8 bit * 4 Vector And. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VOR.QB — Vector Or Quad Byte

Vector Or Quad Byte VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rs] **or** VGPR[rt]

Exception :

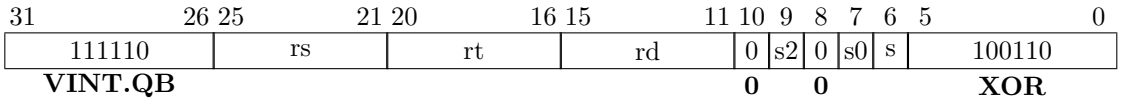
Vector Integer Exception

Overview :

8 bit * 4 Vector Or. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VXOR.QB — Vector Exclusive Or Quad Byte

Vector Exclusive Or Quad Byte VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rs] **xor** VGPR[rt]

Exception :

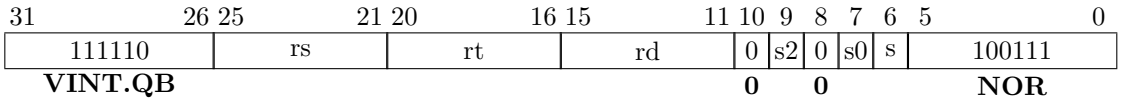
Vector Integer Exception

Overview :

8 bit * 4 Vector Exclusive Or. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VNOR.QB Vector Not Or Paried HalfWord

Vector Not Or Quad Byte VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rs] **nor** VGPR[rt]

Exception :

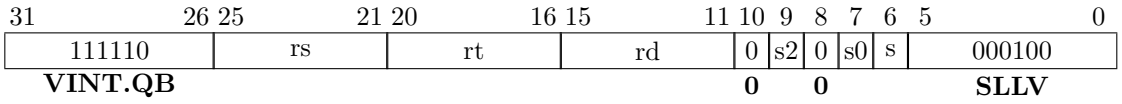
Vector Integer Exception

Overview :

8 bit * 4 Vector Not Or. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VSLLV.QB Vector Shift Left Logical Variable Quad Byte

Vector Shift Left Logical Variable Quad Byte VECTOR



Mnemonic:

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)

Function :

8 bit * 4 Vector Shift Left Logical Variable. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

Exception :

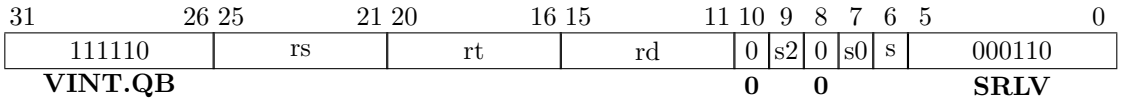
Vector Integer Exception

Overview :

8 bit * 4 Vector Not Or. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VSRLV.QB Vector Shift Right Logical Variable Quad Byte

Vector Shift Right Logical Variable Quad Byte VECTOR



Mnemonic:

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)

Function :

8 bit * 4 Vector Shift Right Logical Variable. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

Exception :

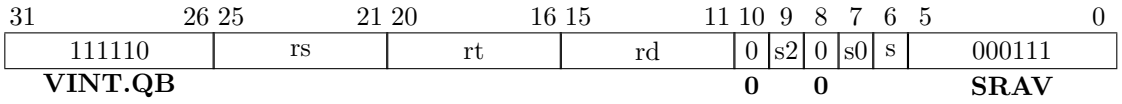
Vector Integer Exception

Overview :

8 bit * 4 Vector Not Or. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VSRAV.QB Vector Shift Right Arithmetic Variable Quad Byte

Vector Shift Right Arithmetic Variable Quad Byte VECTOR



Mnemonic:

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)

Function :

8 bit * 4 Vector Shift Right Arithmetic Variable. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

Exception :

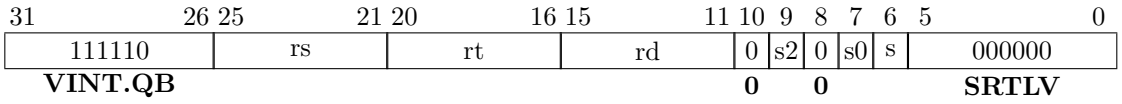
Vector Integer Exception

Overview :

8 bit * 4 Vector Not Or. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VRTLV.QB Vector Rotate Left Variable Quad Byte

Vector Rotate Left Variable Quad Byte VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rt] <<< VGPR[rs]

Exception :

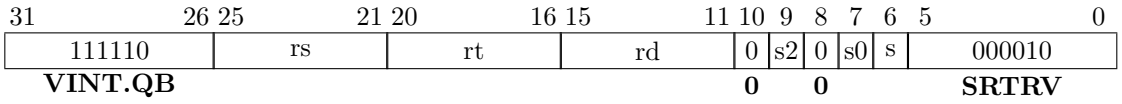
Vector Integer Exception

Overview :

8 bit * 4 Vector Rotate Left Variable. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VRTRV.QB Vector Rotate Right Variable Quad Byte

Vector Rotate Right Variable Quad Byte VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rt] >>> VGPR[rs]

Exception :

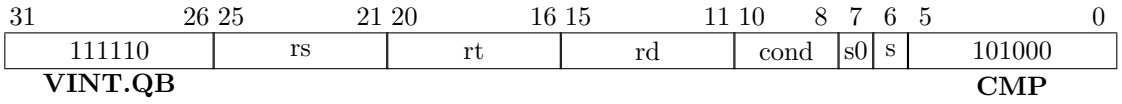
Vector Integer Exception

Overview :

8 bit * 4 Vector Rotate Right Variable. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VCMP.QB Vector Compare Quad Byte

Vector Compare Quad Byte VECTOR



Mnemonic:

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)

Function :

```
if VGPR[rs] cond VGPR[rt] then
    VGPR[rd] ← 1
else
    VGPR[rd] ← 0
endif
```

Exception :

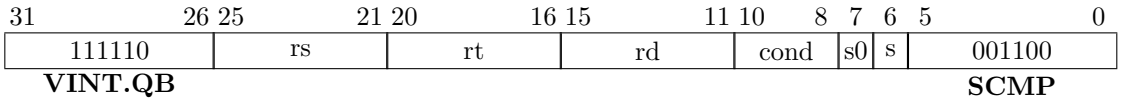
Vector Integer Exception

Overview :

8 bit * 4 Vector Compare. Set 1 or 0 to VGPR[rd] depend on cond. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution. When set up, specifiy eq(=), gt(>), lt(<), ne(≠), le(≤) or ge(≥) to cond.

VSCMP.QB — Vector Compare Quad Byte

Vector Compare Quad Byte VECTOR



Mnemonic:

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)

Function :

```
if VGPR[rs] cond VGPR[rt] then
    VGPR[rd] ← 1
else
    VGPR[rd] ← 0
endif
```

Exception :

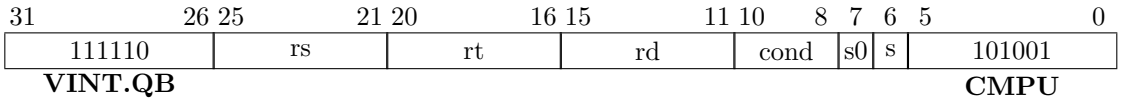
Vector Integer Exception

Overview :

8 bit * 4 Vector Compare. Execute oepration using lower half of VGPR[rt]. Set 1 or 0 to VGPR[rd] depend on cond. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution. When set up, specifiy eq(=), gt(>), lt(<), ne($\neq$), le($\leq$) or ge($\geq$) to cond.

VCMPU.QB Vector Compare Unsigned Quad Byte

Vector Compare Unsigned Quad Byte VECTOR



Mnemonic:

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)

Function :

```
if VGPR[rs] cond VGPR[rt] then
    VGPR[rd] ← 1
else
    VGPR[rd] ← 0
endif
```

Exception :

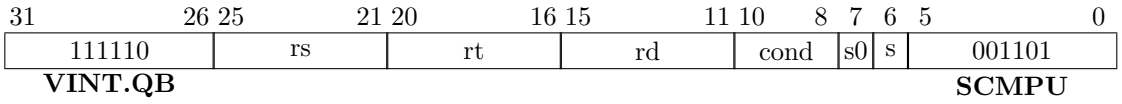
Vector Integer Exception

Overview :

8 bit * 4 Vector Compare Unsigned. Set 1 or 0 to VGPR[rd] depend on cond. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution. When set up, specifiy eq(=), gt(>), lt(<), ne($\neq$), le($\leq$) or ge($\geq$) to cond.

VSCMPU.QB Vector Compare Unsigned Quad Byte

Vector Compare Unsigned Quad Byte VECTOR



Mnemonic:

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)

Function :

```
if VGPR[rs] cond VGPR[rt] then
    VGPR[rd] ← 1
else
    VGPR[rd] ← 0
endif
```

Exception :

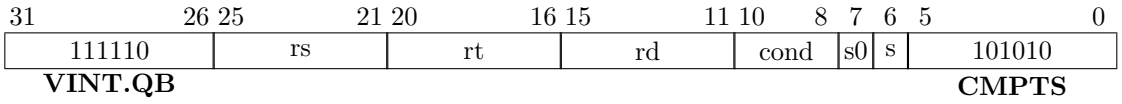
Vector Integer Exception

Overview :

8 bit * 4 Vector Compare Unsigned. Execute oepration using lower half of VGPR[rt]. Set 1 or 0 to VGPR[rd] depend on cond. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution. When set up, specifiy eq(=), gt(>), lt(<), ne($\neq$), le($\leq$) or ge($\geq$) to cond.

VCMPTS.QB Vector Compare to Scalar Register Quad Byte

Vector Compare to Scalar Register Quad Byte VECTOR



Mnemonic:

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)

Function :

```
if VGPR[rs] cond VGPR[rt] then
    SGPR[rd] ← 1
else
    SGPR[rd] ← 0
endif
```

Exception :

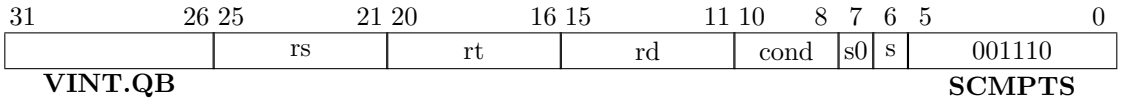
Vector Integer Exception

Overview :

8 bit * 4 Vector Compare to Scalar Register. Results of each elements are store in each bit of scalar register. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution. When set up, specifiy eq(=), gt(>), lt(<), ne($\neq$), le($\leq$) or ge($\geq$) to cond.

VSCMPTS.QB Vector Compare Quad Byte to Scalar Register

Vector Compare Quad Byte to Scalar Register VECTOR



Mnemonic:

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)

Function :

```
if VGPR[rs] cond VGPR[rt] then
    SGPR[rd] ← 1
else
    SGPR[rd] ← 0
endif
```

Exception :

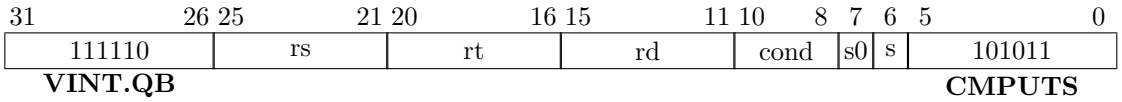
Vector Integer Exception

Overview :

8 bit * 4 Vector Compare to Scalar Register. Execute oepration using lower half of VGPR[rt]. Results of each elements are store in each bit of scalar register. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution. When set up, specifiy eq(=), gt(>), lt(<), ne($\neq$), le($\leq$) or ge($\geq$) to cond.

VCMPUTS.QB Vector Compare Unsigned Quad Byte to Scalar Register

Vector Compare Unsigned Quad Byte to Scalar Register VECTOR



Mnemonic:

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)

Function :

```
if VGPR[rs] cond VGPR[rt] then
    SGPR[rd] ← 1
else
    SGPR[rd] ← 0
endif
```

Exception :

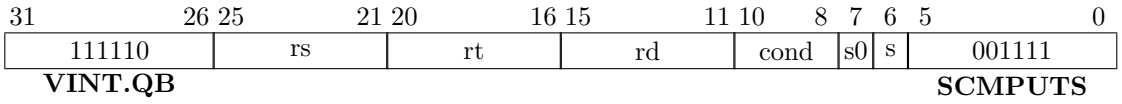
Vector Integer Exception

Overview :

8 bit * 4 Vector Compare Unsigned to Scalar Register. Results of each elements are store in each bit of scalar register. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution. When set up, specifiy eq(=), gt(>), lt(<), ne($\neq$), le($\leq$) or ge($\geq$) to cond.

VSCMPUTS.QB Vector Compare Unsigned Quad Byte to Scalar Register

Vector Compare Unsigned Quad Byte to Scalar Register VECTOR



Mnemonic:

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)

Function :

```
if VGPR[rs] cond VGPR[rt] then
    SGPR[rd] ← 1
else
    SGPR[rd] ← 0
endif
```

Exception :

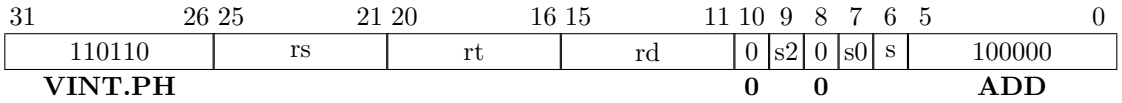
Vector Integer Exception

Overview :

8 bit * 4 Vector Compare Unsigned to Scalar Register. Execute oepration using lower half of VGPR[rt]. Results of each elements are store in each bit of scalar register. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution. When set up, specifiy eq(=), gt(>), lt(<), ne($\neq$), le($\leq$) or ge($\geq$) to cond.

VADD.PH Vector Add Paired HalfWord

Vector Add Paired HalfWord VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rs] + VGPR[rt]

Exception :

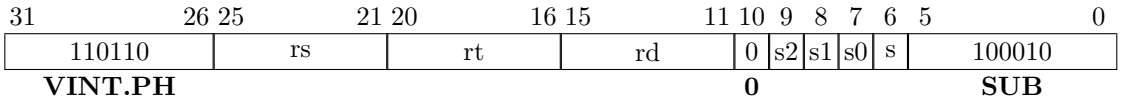
Vector Integer Exception

Overview :

16 bit * 2 Vector Add. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VSUB.PH Vector Subtract Paired HalfWord

Vector Subtract Paired HalfWord VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rs] - VGPR[rt]

Exception :

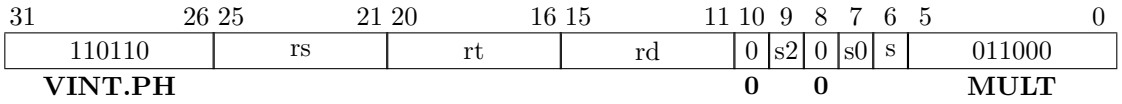
Vector Integer Exception

Overview :

16 bit * 2 Vector Subtract. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VMULT.PH — Vector Multiply Paired HalfWord

Vector Multiply Paired HalfWord VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rs] × VGPR[rt]

Exception :

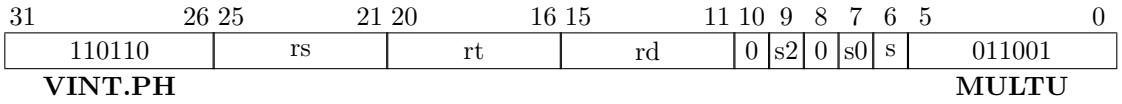
Vector Integer Exception

Overview :

16 bit * 2 Vector Multiply. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VMULTU.PH — Vector Multiply Unsigned

Vector Multiply Paired HalfWord — VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rs] × VGPR[rt]

Exception :

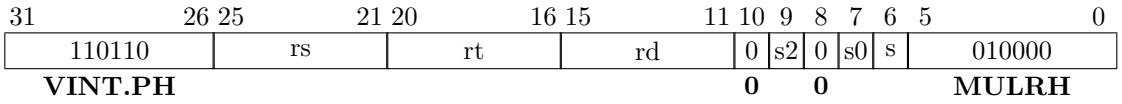
Vector Integer Exception

Overview :

16 bit * 2 Vector Multiply Unsigned. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VMULTH.PH Vector Multiply Paired HalfWord on High Bit

Vector Multiply Paired HalfWord VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rs] × VGPR[rt]

Exception :

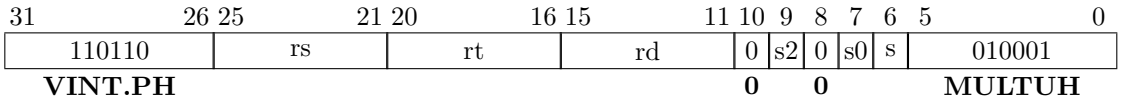
Vector Integer Exception

Overview :

16 bit * 2 Vector Multiply Signed. Store upper half of results of execution to VGPR[rd]. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VMULTUH.PH Vector Multiply Unsigned Paired HalfWord on High Bit

Vector Multiply Paired HalfWord VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rs] × VGPR[rt]

Exception :

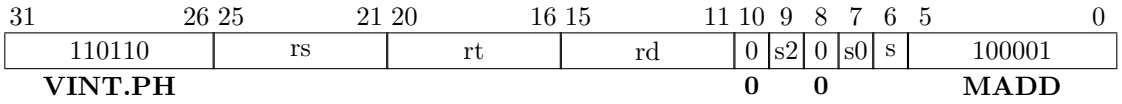
Vector Integer Exception

Overview :

16 bit * 2 Vector Multiply Unsigned. Store upper half of results of execution to VGPR[rd]. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VMADD.PH Vector Multiply and Add Paired HalfWord

Vector Multiply and Add Paired HalfWord VECTOR



Mnemonic:

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)

Function :

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

Exception :

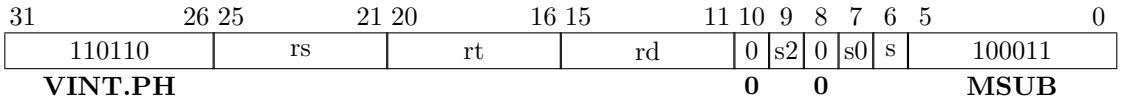
Vector Integer Exception

Overview :

16 bit * 2 Vector Multiply and Add. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VMSUB.PH Vector Multiply and Subtract Paired HalfWord

Vector Multiply and Subtract Paired HalfWord VECTOR



Mnemonic:

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)

Function :

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

Exception :

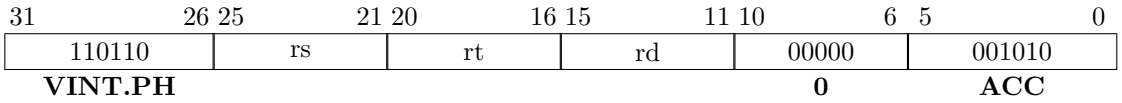
Vector Integer Exception

Overview :

16 bit * 2 Vector Multiply and Subtract. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VACC.PH Vector Accumulate Paired HalfWord

Vector Accumulate Paired HalfWord VECTOR



Mnemonic:

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

Function :

SGPR[rd] ← ∑VGPR[rs]

Exception :

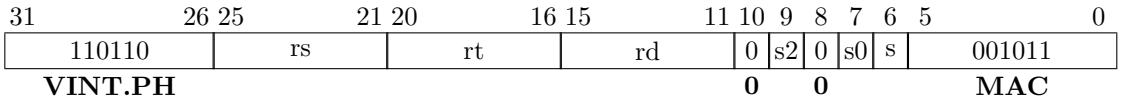
Vector Integer Exception

Overview :

16 bit * 2 Vector Accumulate. Add all elements of vector. When sync is 1, suppress the speculative execution.

VMAC.PH Vector Multiply and Accumulate Paired HalfWord

Vector Multiply and Accumulate Paired HalfWord VECTOR



Mnemonic:

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)

Function :

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

Exception :

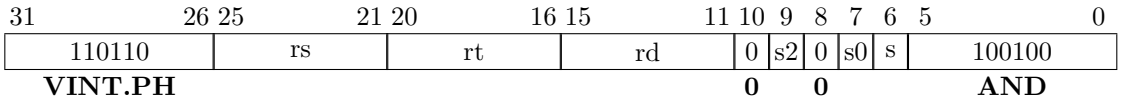
Vector Integer Exception

Overview :

16 bit * 2 Vector Multiply and Accumulate. Multiply elements of 2 vector and add all elements of these result. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VAND.PH Vector And Paired HalfWord

Vector And Paired HalfWord VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rs] **and** VGPR[rt]

Exception :

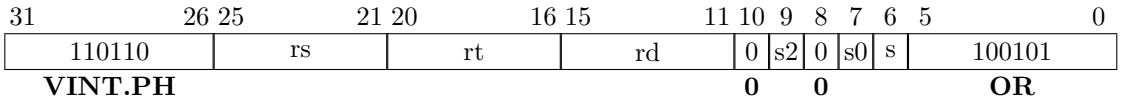
Vector Integer Exception

Overview :

16 bit * 2 Vector And. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VOR.PH — Vector Or Paired HalfWord

Vector Or Paired HalfWord VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rs] **or** VGPR[rt]

Exception :

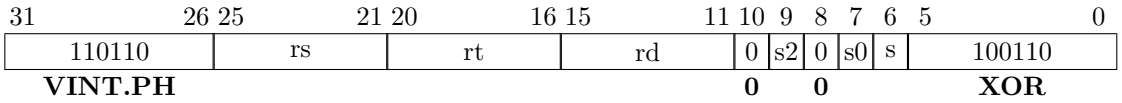
Vector Integer Exception

Overview :

16 bit * 2 Vector Or. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VXOR.PH Vector Exclusive Or Paired HalfWord

Vector Exclusive Or Paired HalfWord VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rs] **xor** VGPR[rt]

Exception :

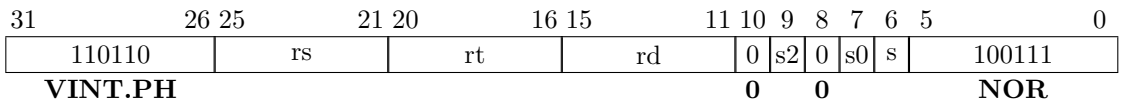
Vector Integer Exception

Overview :

16 bit * 2 Vector Exclusive Or. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VNOR.PH Vector Not Or Paried HalfWord

Vector Not Or Paired HalfWord VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rs] **nor** VGPR[rt]

Exception :

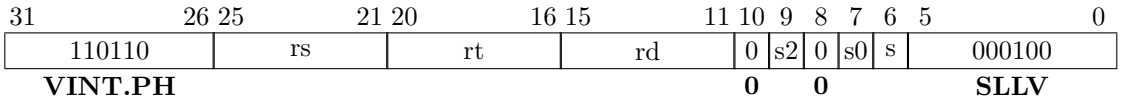
Vector Integer Exception

Overview :

16 bit * 2 Vector Not Or. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VSLLV.PH Vector Shift Left Logical Variable Paired HalfWord

Vector Shift Left Logical Variable Paired HalfWord VECTOR



Mnemonic:

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)

Function :

16 bit * 2 Vector Shift Left Logical Variable. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

Exception :

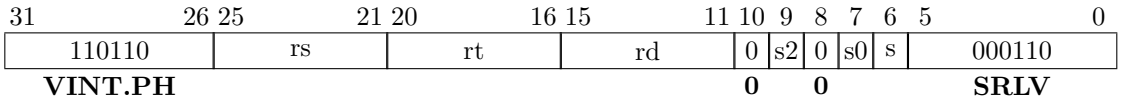
Vector Integer Exception

Overview :

16 bit * 2 Vector Not Or. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VSRLV.PH Vector Shift Right Logical Variable Paired HalfWord

Vector Shift Right Logical Variable Paried HalfWord VECTOR



Mnemonic:

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)

Function :

VGPR[rd] $\leftarrow$ VGPR[rt] $\gg$ VGPR[rs]

Exception :

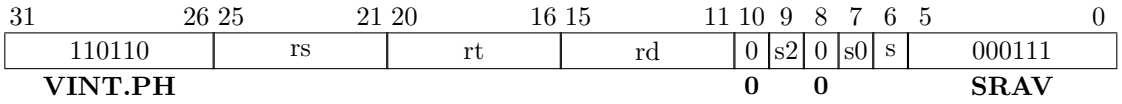
Vector Integer Exception

Overview :

16 bit * 2 Vector Shift Right Logical Variable. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VSRAV.PH Vector Shift Right Arithmetic Variable Paired HalfWord

Vector Shift Right Arithmetic Variable Paired HalfWord VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rt] ≫ VGPR[rs]

Exception :

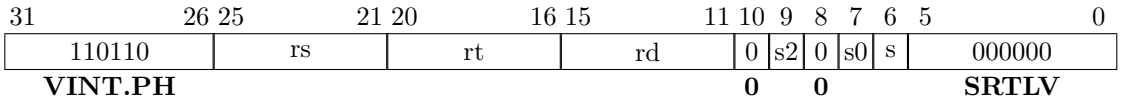
Vector Integer Exception

Overview :

16 bit * 2 Vector Shift Right Arithmetic Variable. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VRTLV.PH Vector Rotate Left Variable Paired HalfWord

Vector Rotate Left Variable Paired HalfWord VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rt] <<< VGPR[rs]

Exception :

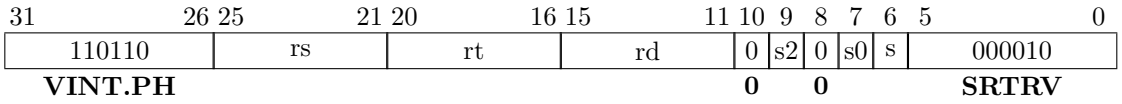
Vector Integer Exception

Overview :

16 bit * 2 Vector Rotate Left Variable. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VRTRV.PH Vector Rotate Right Variable Paired HalfWord

Vector Rotate Right Variable Paired HalfWord VECTOR



Mnemonic:

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)

Function :

VGPR[rd] ← VGPR[rt] >>> VGPR[rs]

Exception :

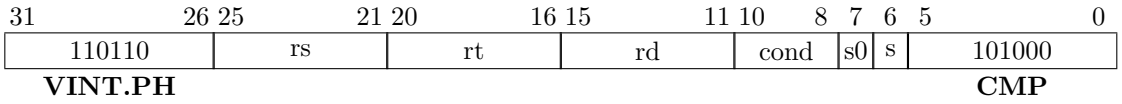
Vector Integer Exception

Overview :

16 bit * 2 Vector Rotate Right Variable. When s0 is 1, execute operations by scalar register (SGPR[rt]) insted of VGPR[rt]. When s2 is 1, execute operations by lower half of VGPR[rt]. When sync is 1, suppress the speculative execution.

VCMP.PH — Vector Compare

Vector Compare Paired HalfWord — VECTOR



Mnemonic:

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)

Function :

```
if VGPR[rs] cond VGPR[rt] then
    VGPR[rd] ← 1
else
    VGPR[rd] ← 0
endif
```

Exception :

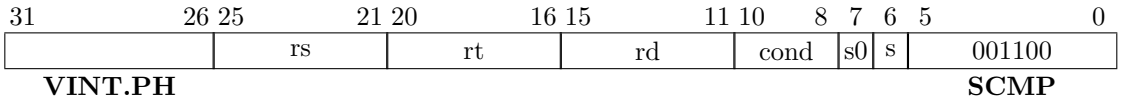
Vector Integer Exception

Overview :

16 bit * 2 Vector Compare. Set 1 or 0 to VGPR[rd] depend on cond. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution. When set up, specifiy eq(=), gt(>), lt(<), ne($\neq$), le($\leq$) or ge($\geq$) to cond.

VSCMP.PH Vector Compare

Vector Compare Paired HalfWord VECTOR



Mnemonic:

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)

Function :

```
if VGPR[rs] cond VGPR[rt] then
    VGPR[rd] ← 1
else
    VGPR[rd] ← 0
endif
```

Exception :

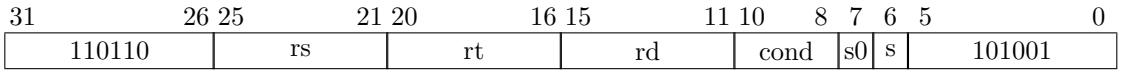
Vector Integer Exception

Overview :

16 bit * 2 Vector Compare. Execute oepration using lower half of VGPR[rt]. Set 1 or 0 to VGPR[rd] depend on cond. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution. When set up, specifiy eq(=), gt(>), lt(<), ne($\neq$), le($\leq$) or ge($\geq$) to cond.

VCMPU.PH Vector Compare Unsigned Paired HalfWord

Vector Compare Unsigned Paired HalfWord VECTOR



Mnemonic:

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)

Function :

```
if VGPR[rs] cond VGPR[rt] then
    VGPR[rd] ← 1
else
    VGPR[rd] ← 0
endif
```

Exception :

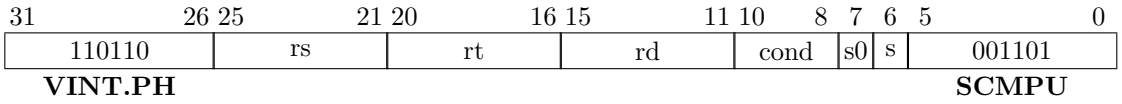
Vector Integer Exception

Overview :

16 bit * 2 Vector Compare Unsigned. Set 1 or 0 to VGPR[rd] depend on cond. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution. When set up, specifiy eq(=), gt(>), lt(<), ne($\neq$), le($\leq$) or ge($\geq$) to cond.

VSCMPU.PH Vector Compare Unsigned Paired HalfWord

Vector Compare Unsigned Paired HalfWord VECTOR



Mnemonic:

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)

Function :

```
if VGPR[rs] cond VGPR[rt] then
    VGPR[rd] ← 1
else
    VGRP[rd] ← 0
endif
```

Exception :

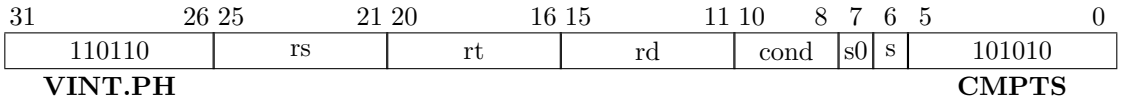
Vector Integer Exception

Overview :

16 bit * 2 Vector Compare Unsigned. Execute oepration using lower half of VGPR[rt]. Set 1 or 0 to VGPR[rd] depend on cond. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution. When set up, specifiy eq(=), gt(>), lt(<), ne(≠), le(≤) or ge(≥) to cond.

VCMPTS.PH Vector Compare to Scalar Register Paired HalfWord

Vector Compare to Scalar Register Paired HalfWord VECTOR



Mnemonic:

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)

Function :

```
if VGPR[rs] cond VGPR[rt] then
    SGPR[rd] ← 1
else
    SGPR[rd] ← 0
endif
```

Exception :

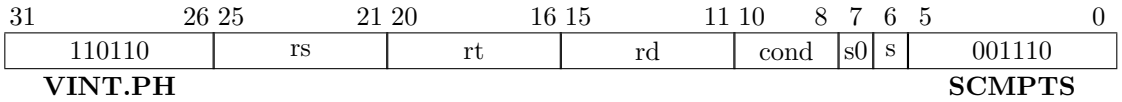
Vector Integer Exception

Overview :

16 bit * 2 Vector Compare to Scalar Register. Results of each elements are store in each bit of scalar register. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution. When set up, specifiy eq(=), gt(>), lt(<), ne($\neq$), le($\leq$) or ge($\geq$) to cond.

VSCMPTS.PH Vector Compare Paired HalfWord to Scalar Register

Vector Compare Paired HalfWord to Scalar Register VECTOR



Mnemonic:

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)

Function :

```
if VGPR[rs] cond VGPR[rt] then
    SGPR[rd] ← 1
else
    SGPR[rd] ← 0
endif
```

Exception :

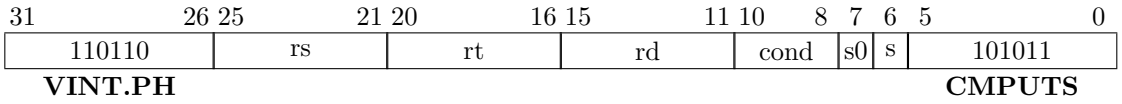
Vector Integer Exception

Overview :

16 bit * 2 Vector Compare to Scalar Register. Execute oepration using lower half of VGPR[rt]. Results of each elements are store in each bit of scalar register. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution. When set up, specifiy eq(=), gt(>), lt(<), ne(≠), le(≤) or ge(≥) to cond.

VCMPUTS.PH Vector Compare Unsigned Paired HalfWord to Scalar Register

Vector Compare Unsigned Paired HalfWord to Scalar Register VECTOR



Mnemonic:

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)

Function :

```
if VGPR[rs] cond VGPR[rt] then
    SGPR[rd] ← 1
else
    SGPR[rd] ← 0
endif
```

Exception :

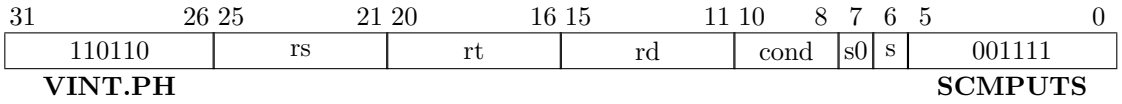
Vector Integer Exception

Overview :

16 bit * 2 Vector Compare to Scalar Register Unsigned. Results of each elements are store in each bit of scalar register. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution. When set up, specifiy eq(=), gt(>), lt(<), ne($\neq$), le($\leq$) or ge($\geq$) to cond.

VSCMPUTS.PH Vector Compare Unsigned Paired HalfWord to Scalar Register

Vector Compare Unsigned Paired HalfWord to Scalar Register VECTOR



Mnemonic:

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)

Function :

```
if VGPR[rs] cond VGPR[rt] then
    SGPR[rd] ← 1
else
    SGPR[rd] ← 0
endif
```

Exception :

Vector Integer Exception

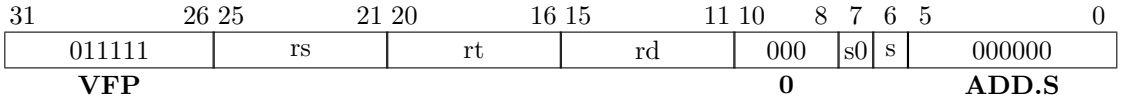
Overview :

16 bit * 2 Vector Compare to Scalar Register Unsigned. Execute oepration using lower half of VGPR[rt]. Results of each elements are store in each bit of scalar register. When s0 is 1, execute oeprations by scalar register (SGPR[rt]) insted of VGPR[rt]. When sync is 1, suppress the speculative execution. When set up, specifiy eq(=), gt(>), lt(<), ne(≠), le(≤) or ge(≥) to cond.

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



Mnemonic:

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)

Function :

VFPR[rd] ← VFPR[rs] + VFPR[rt]

Exception :

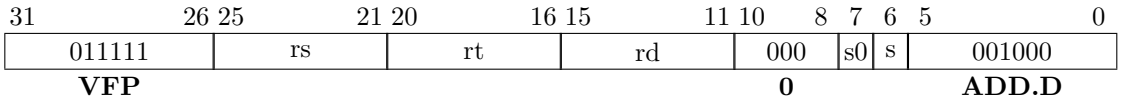
Vector Floating Point Exception

Overview :

単精度ベクトル加算．s0 が 1 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 が 1 の場合は，投機実行を抑制する．

VADD.D Vector Add Double

ベクトル加算 VECTOR



Mnemonic:

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)

Function :

VFPR[rd] ← VFPR[rs] + VFPR[rt]

Exception :

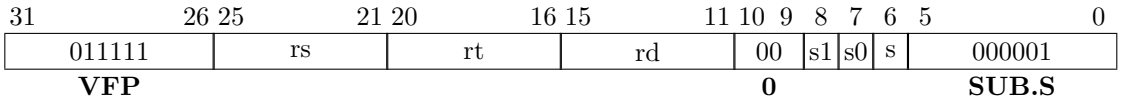
Vector Floating Point Exception

Overview :

倍精度ベクトル加算．s0 が 1 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 が 1 の場合は，投機実行を抑制する．

VSUB.S — Vector Subtract Single

ベクトル減算 VECTOR



Mnemonic:

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)

Function :

VFPR[rd] ← VFPR[rs] − VFPR[rt]

Exception :

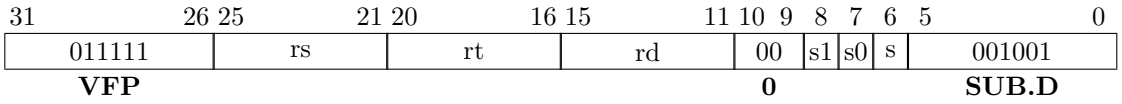
Vector Floating Point Exception

Overview :

単精度ベクトル減算．s0 が 1 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．s1 が 1 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 が 1 の場合は，投機実行を抑制する．

VSUB.D Vector Subtract Double

ベクトル減算 VECTOR



Mnemonic:

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)

Function :

VFPR[rd] ← VFPR[rs] − VFPR[rt]

Exception :

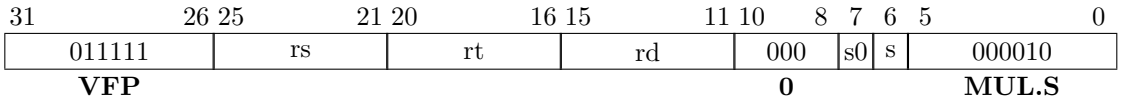
Vector Floating Point Exception

Overview :

倍精度ベクトル減算．s0 が 1 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．s1 が 1 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 が 1 の場合，投機実行を抑制する．

VMUL.S — Vector Multiply Single

ベクトル乗算 VECTOR



Mnemonic:

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)

Function :

VFPR[rd] ← VFPR[rs] × VFPR[rt]

Exception :

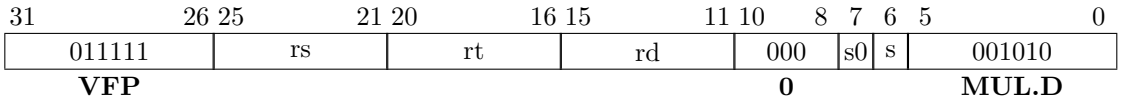
Vector Floating Point Exception

Overview :

単精度ベクトル乗算．s0 が 1 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 が 1 の場合，投機実行を抑制する．

VMUL.D Vector Multiply Double

ベクトル乗算 VECTOR



Mnemonic:

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)

Function :

VFPR[rd] ← VFPR[rs] × VFPR[rt]

Exception :

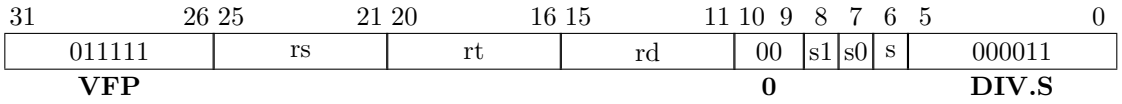
Vector Floating Point Exception

Overview :

倍精度ベクトル乗算．s0 が 1 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 が 1 の場合，投機実行を抑制する．

VDIV.S — Vector Divide Single

ベクトル除算 VECTOR



Mnemonic:

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)

Function :

VFPR[rd] ← VFPR[rs] ÷ VFPR[rt]

Exception :

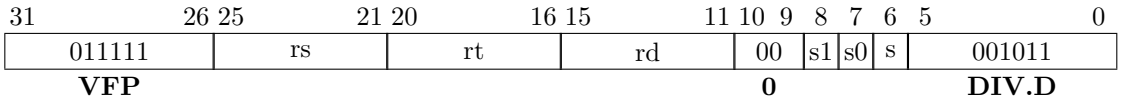
Vector Floating Point Exception

Overview :

単精度ベクトル除算．s0 が 1 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．s1 が 1 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 が 1 の場合は，投機実行を抑制する．

VDIV.D Vector Divide Double

ベクトル除算 VECTOR



Mnemonic:

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)

Function :

VFPR[rd] ← VFPR[rs] ÷ VFPR[rt]

Exception :

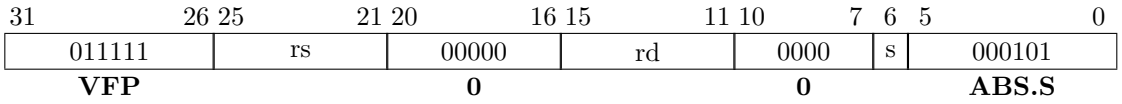
Vector Floating Point Exception

Overview :

倍精度ベクトル除算．s0 が 1 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．s1 が 1 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 が 1 の場合は，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**

Mnemonic:

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

Function :

VFPR[rd] ← | VFPR[rs] |

Exception :

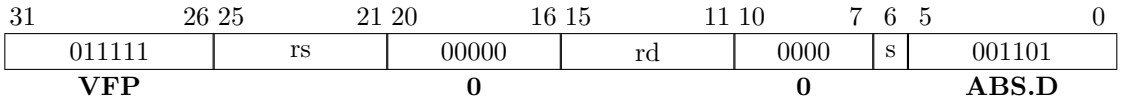
Vector Floating Point Exception

Overview :

単精度ベクトル絶対値演算．sync が 1 の場合は，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**

Mnemonic:

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

Function :

VFPR[rd] ← | VFPR[rs] |

Exception :

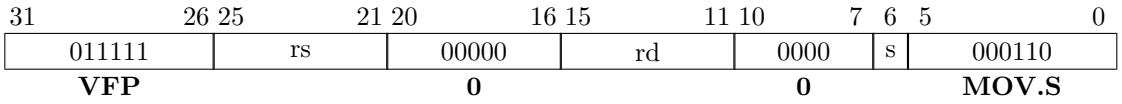
Vector Floating Point Exception

Overview :

倍精度ベクトル絶対値演算．sync が 1 の場合は，投機実行を抑制する．

VMOV.S Vector Move Single

ベクトル転送 VECTOR



Mnemonic:

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

Function :

VFPR[rd] ← VFPR[rs]

Exception :

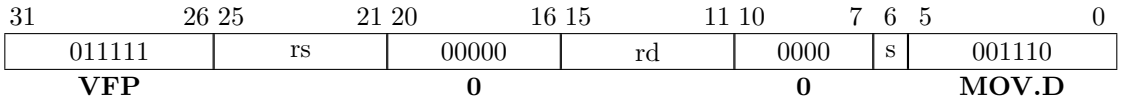
Vector Floating Point Exception

Overview :

単精度ベクトル転送命令．sync が 1 の場合は，投機実行を抑制する．

VMOV.D Vector Move Double

ベクトル転送 VECTOR



Mnemonic:

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

Function :

VFPR[rd] ← VFPR[rs]

Exception :

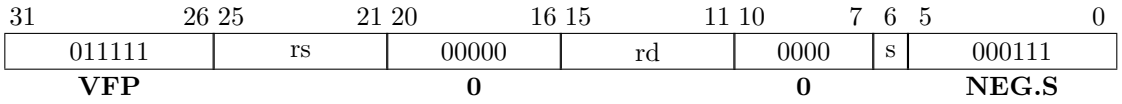
Vector Floating Point Exception

Overview :

倍精度ベクトル転送命令．sync が 1 の場合は，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**

Mnemonic:

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

Function :

VFPR[rd] ← −1 × VFPR[rs]

Exception :

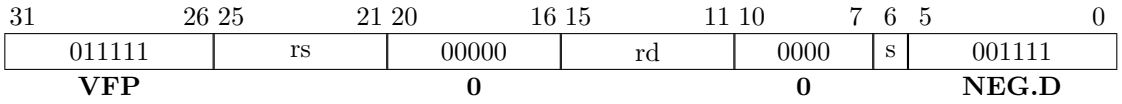
Vector Floating Point Exception

Overview :

単精度ベクトル符号反転演算．sync が 1 の場合は，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**

Mnemonic:

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

Function :

VFPR[rd] ← −1 × VFPR[rs]

Exception :

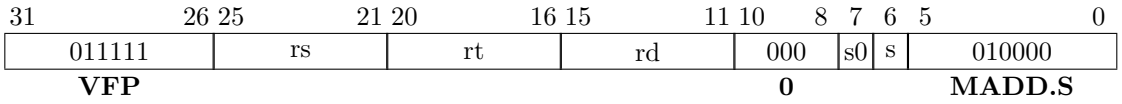
Vector Floating Point Exception

Overview :

倍精度ベクトル符号反転演算．sync が 1 の場合は，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



Mnemonic:

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)

Function :

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

Exception :

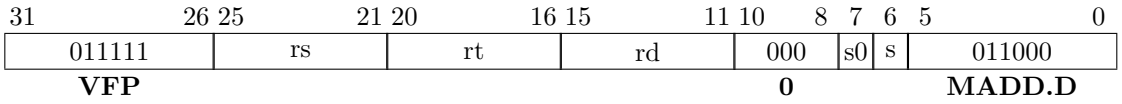
Vector Floating Point Exception

Overview :

単精度ベクトル積和演算．s0 が 1 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 が 1 の場合は，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



Mnemonic:

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)

Function :

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

Exception :

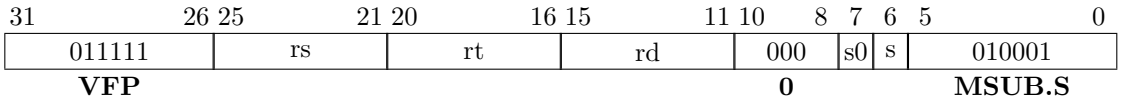
Vector Floating Point Exception

Overview :

倍精度ベクトル積和演算．s0 が 1 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 が 1 の場合は，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



Mnemonic:

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)

Function :

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

Exception :

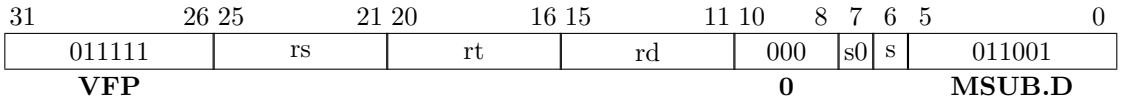
Vector Floating Point Exception

Overview :

単精度ベクトル積差演算．s0 が 1 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 が 1 の場合は，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



Mnemonic:

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)

Function :

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

Exception :

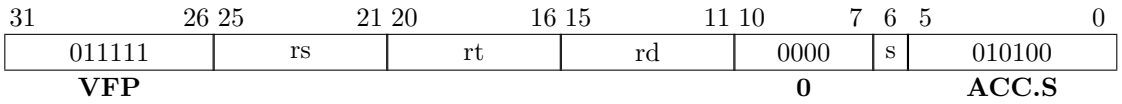
Vector Floating Point Exception

Overview :

倍精度ベクトル積差演算．s0 が 1 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 が 1 の場合は，投機実行を抑制する．

VACC.S Vector Accumulate Single

ベクトル累算 (単精度) VECTOR



Mnemonic:

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

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

Function :

SFPR[rd] $\leftarrow \sum$ VFPR[rs]

Exception :

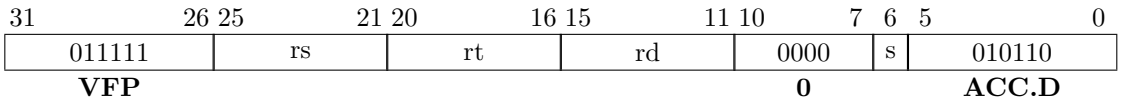
Vector Floating Point Exception

Overview :

32bit 単精度ベクトル累算．ベクトルの要素を全て加算する．sync が 1 の場合は，投機実行を抑制する．

VACC.D Vector Accumulate Double

ベクトル累算 (倍精度) VECTOR



Mnemonic:

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

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

Function :

SFPR[rd] $\leftarrow \sum$ VFPR[rs]

Exception :

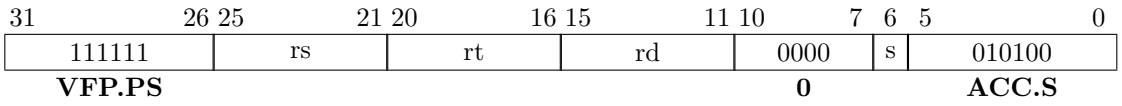
Vector Floating Point Exception

Overview :

64bit 倍精度ベクトル累算．ベクトルの要素を全て加算する．sync が 1 の場合は，投機実行を抑制する．

VACC.PS Vector Accumulate Paired Single

ベクトル累算 (SIMD) VECTOR



Mnemonic:

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

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

Function :

SFPR[rd] $\leftarrow \sum$ VFPR[rs]

Exception :

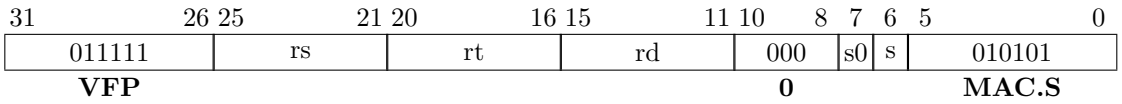
Vector Floating Point Exception

Overview :

32bit×2 単精度ベクトル累算．ベクトルの要素を全て加算する．sync が 1 の場合は，投機実行を抑制する．

VMAC.S Vector Multiply and Accumulate Single

ベクトル積和演算 (単精度) VECTOR



Mnemonic:

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

Function :

SFPR[rd] $\leftarrow \sum$ VFPR[rs] $\times$ VFPR[rt]

Exception :

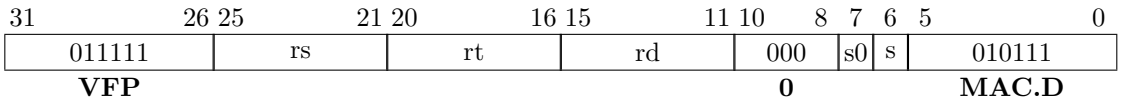
Vector Integer Exception

Overview :

32bit 単精度ベクトル積和演算．2 つのベクトル要素を乗算し，それを全て加算する．s0 が 1 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 が 1 の場合は，投機実行を抑制する．

VMAC.D Vector Multiply and Accumulate Double

ベクトル積和演算 (倍精度) VECTOR



Mnemonic:

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

Function :

SFPR[rd] $\leftarrow \sum$ VFPR[rs] $\times$ VFPR[rt]

Exception :

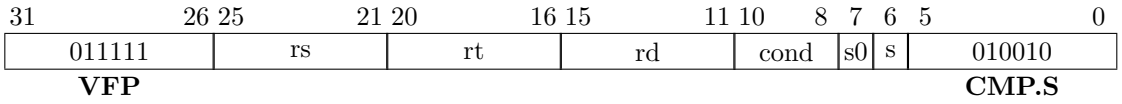
Vector Floating Point Exception

Overview :

64bit 倍精度ベクトル積和演算．2 つのベクトル要素を乗算し，それを全て加算する．s0 が 1 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 が 1 の場合は，投機実行を抑制する．

VCMP.S — Vector Compare Single

ベクトル比較 VECTOR



Mnemonic:

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)

Function :

```
if VFPR[rs] cond VFPR[rt] then
    VFPR[rd] ← 1
else
    VFPR[rd] ← 0
endif
```

Exception :

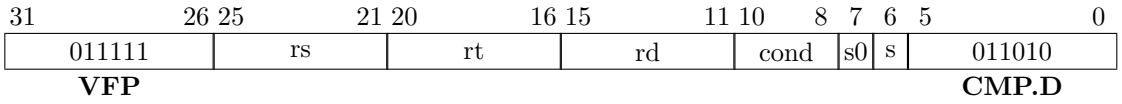
Vector Floating Point Exception

Overview :

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．s0 が 1 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 が 1 の場合は，投機実行を抑制する．比較条件 (cond) には，eq(=)，gt(>)，lt(<)，ne(≠)，le(≤)，ge(≥) のどれかを指定する．

VCMP.D Vector Compare Double

ベクトル比較 VECTOR



Mnemonic:

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)

Function :

```
if VFPR[rs] cond VFPR[rt] then      VFPR[rd] ← 1
else
    VFPR[rd] ← 0
endif
```

Exception :

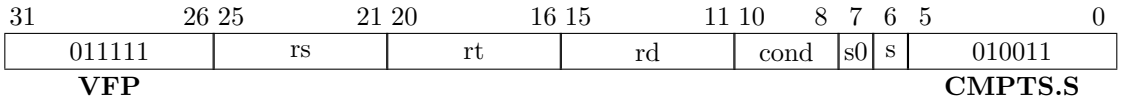
Vector Floating Point Exception

Overview :

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．s0 が 1 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 が 1 の場合は，投機実行を抑制する．比較条件 (cond) には，eq(=)，gt(>)，lt(<)，ne(≠)，le(≤)，ge(≥) のどれかを指定する．

VCMPTS.S Vector Compare Single to Scalar Register

ベクトル比較 VECTOR



Mnemonic:

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)

Function :

```
if VFPR[rs] cond VFPR[rt] then
    SFPR[rd] ← 1
else
    SFRP[rd] ← 0
endif
```

Exception :

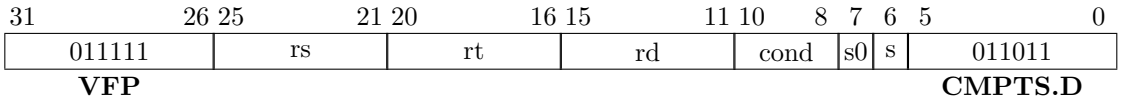
Vector Floating Point Exception

Overview :

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．s0 が 1 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 が 1 の場合は，投機実行を抑制する．比較条件 (cond) には，eq(=)，gt(>)，lt(<)，ne(≠)，le(≤)，ge(≥) のどれかを指定する．

VCMPTS.D Vector Compare Double to Scalar Register

ベクトル比較 VECTOR



Mnemonic:

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)

Function :

```
if VFPR[rs] cond VFPR[rt] then
    SFPR[rd] ← 1
else
    SFPR[rd] ← 0
endif
```

Exception :

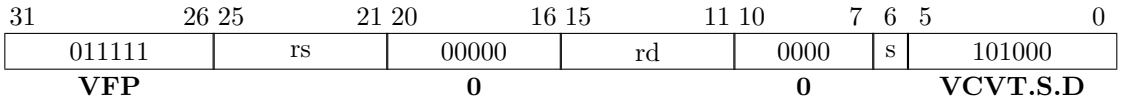
Vector Floating Point Exception

Overview :

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．s0 が 1 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 が 1 の場合は，投機実行を抑制する．比較条件 (cond) には，eq(=)，gt(>)，lt(<)，ne(≠)，le(≤)，ge(≥) のどれかを指定する．

VCVT.S.D — Vector Convert to Single from Double

ベクトルフォーマット変換 VECTOR



Mnemonic:

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

Function :

VFPR[rd] ← Double_to_Single(VFPR[rs])

Exception :

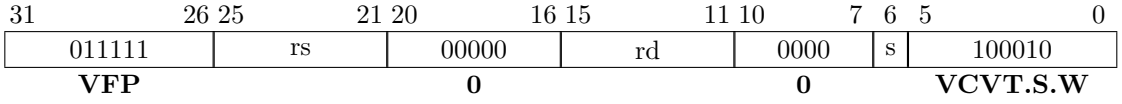
Vector Floating Point Exception

Overview :

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 が 1 の場合は，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



Mnemonic:

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

Function :

VFPR[rd] ← Word_to_Single(VFPR[rs])

Exception :

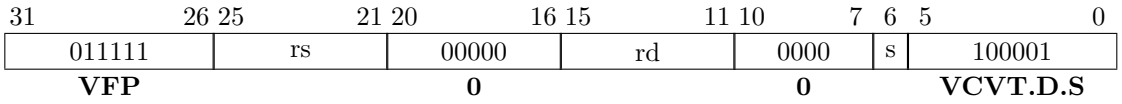
Vector Floating Point Exception

Overview :

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 が 1 の場合は，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



Mnemonic:

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

Function :

VFPR[rd] ← Single_to_Double(VFPR[rs])

Exception :

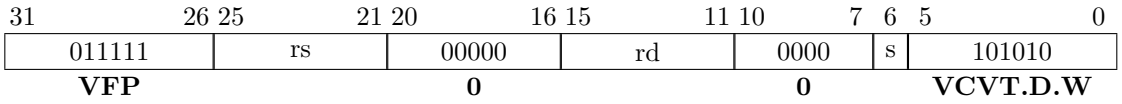
Vector Floating Point Exception

Overview :

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 が 1 の場合は，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



Mnemonic:

VCVT.D.W rd, rs (sync(s) = 0)
VCVT.D.W.sy rd, rs (sync(s) = 1)

Function :

VFPR[rd] ← Word_to_Double(VFPR[rs])

Exception :

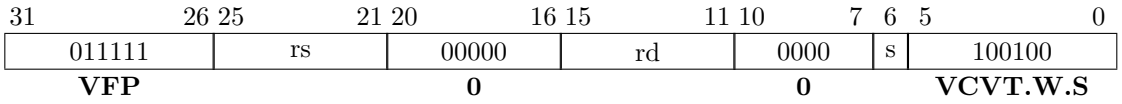
Vector Floating Point Exception

Overview :

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 が 1 の場合は，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**

Mnemonic:

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

Function :

VFPR[rd] ← Single_to_Word(VFPR[rs])

Exception :

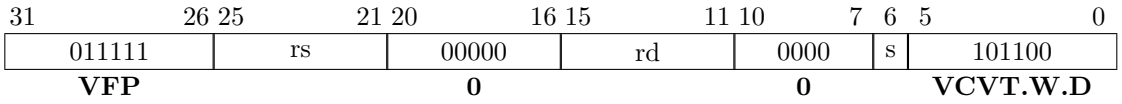
Vector Floating Point Exception

Overview :

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 が 1 の場合は，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**

Mnemonic:

VCVT.W.D rd, rs (sync(s) = 0)
VCVT.W.D.sy rd, rs (sync(s) = 1)

Function :

VFPR[rd] ← Double_to_Word(VFPR[rs])

Exception :

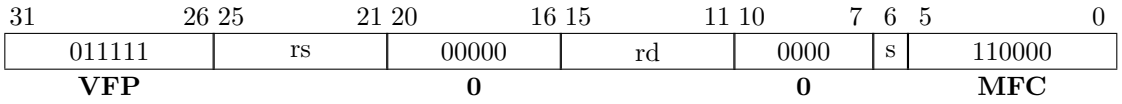
Vector Floating Point Exception

Overview :

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 が 1 の場合は，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



Mnemonic:

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

Function :

FPR[rd] ← VFCTRL[rs]

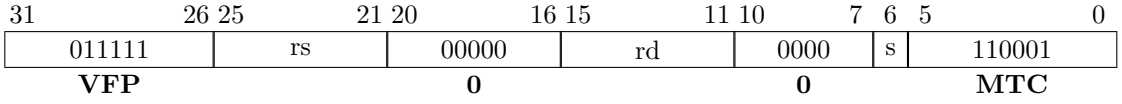
Exception :

Overview :

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 が 1 の場合は，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



Mnemonic:

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

Function :

VFCTRL[rd] ← FPR[rs]

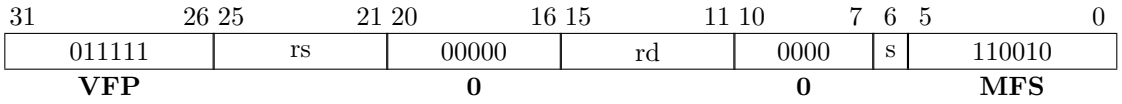
Exception :

Overview :

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 が 1 の場合は，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



Mnemonic:

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

Function :

FPR[rd] ← SFPR[rs]

Exception :

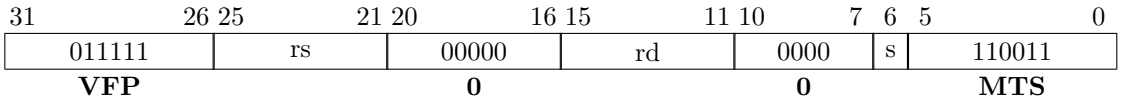
Vector Floating Point Exception

Overview :

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 が 1 の場は，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



Mnemonic:

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

Function :

SFPR[rd] ← FPR[rs]

Exception :

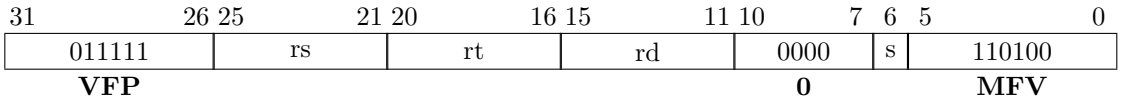
Vector Floating Point Exception

Overview :

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 が 1 の場合は，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



Mnemonic:

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

Function :

SFPR[rd] ← VFPR[rs][rt]

Exception :

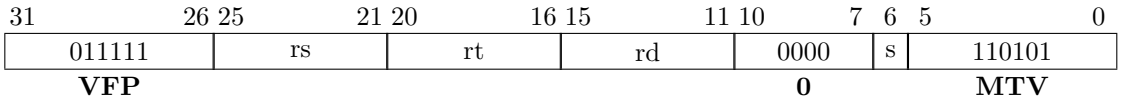
Vector Floating Point Exception

Overview :

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 が 1 の場合は，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



Mnemonic:

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

Function :

SFPR[rd] ← VFPR[rs][rt]

Exception :

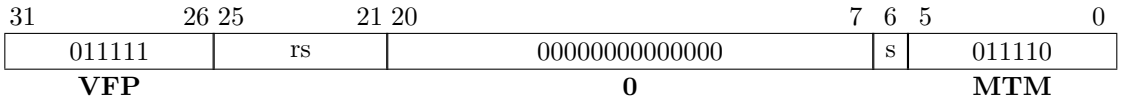
Vector Floating Point Exception

Overview :

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 が 1 の場合は，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



Mnemonic:

VFMTM rs (sync(s) = 0)

VFMTM.sy rs (sync(s) = 1)

Function :

VFCTRL[*Mask Register*] ← VSFPR[rs]

Exception :

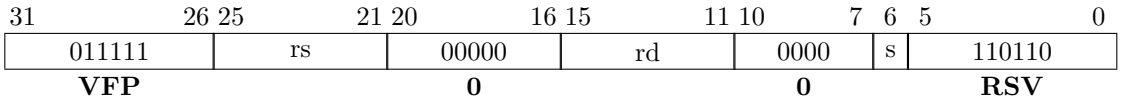
Vector Floating Point Exception

Overview :

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 が 1 の場合は，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



Mnemonic:

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

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

Function :

```
reserve_vector_register(GPR[rs])
if success_reserve_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif
```

Exception :

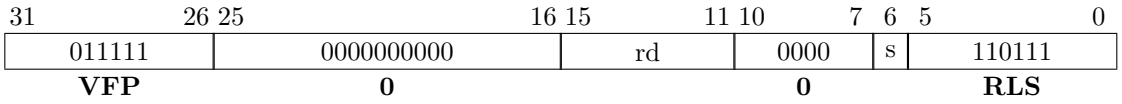
None

Overview :

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 が 1 の場合は，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



Mnemonic:

VFRLS rd (sync(s) = 0)

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

Function :

```
release_vector_register()
if success_release_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif
```

Exception :

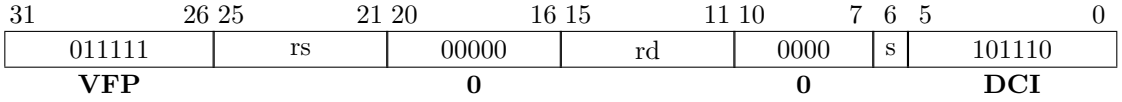
None

Overview :

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 が 1 の場合，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



Mnemonic:

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

Function :

VFCPD[rd] ← GPR[rs]

Exception :

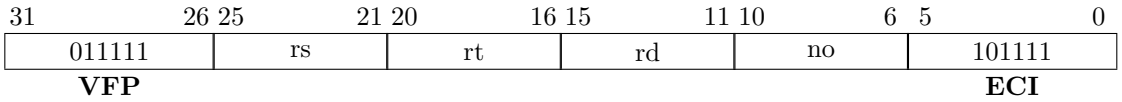
Vector Floating Point Exception

Overview :

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 が 1 の場合，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



Mnemonic:

VFDCI rd, rs, rt, no

Function :

VFPR[rd] ← VFPR[rs] op VFPR[rt]

Exception :

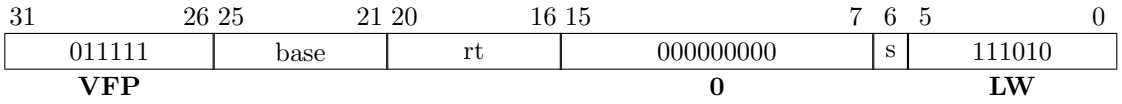
Vector Floating Point Exception

Overview :

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



Mnemonic:

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

Function :

VFPR[rt] ← MEM.WORD[GPR[base]]

Exception :

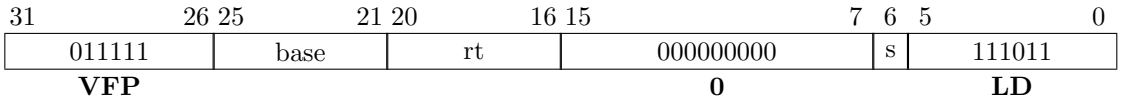
Vector Floating Point Exception
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Load)

Overview :

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 が 1 の場合，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



Mnemonic:

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

Function :

VFPR[rt] ← MEM.DWORD[GPR[base]]

Exception :

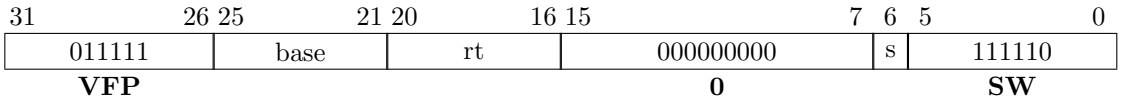
Vector Floating Point Exception
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Load)

Overview :

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 が 1 の場合，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



Mnemonic:

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

Function :

MEM.WORD[GPR[base]] ← VFPR[rt]

Exception :

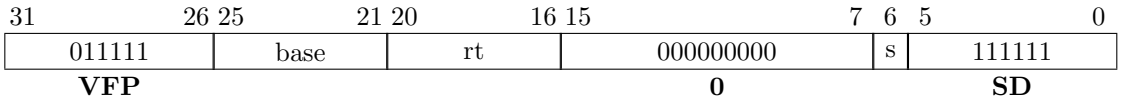
Vector Floating Point Exception
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Store)

Overview :

浮動小数点ベクトルレジスタからメモリにストアする．sync が 1 の場合，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



Mnemonic:

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

Function :

MEM.DWORD[GPR[base]] ← VFPR[rt]

Exception :

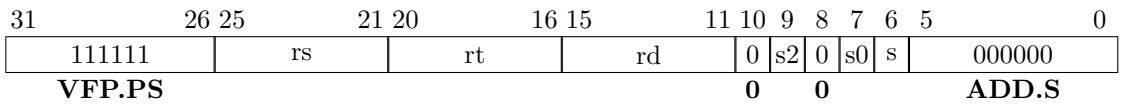
Vector Floating Point Exception
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Store)

Overview :

浮動小数点ベクトルレジスタからメモリにストアする．sync が 1 の場合，投機実行を抑制する．

VADD.PS — Vector Add Paired Single

Vector Add — VECTOR



Mnemonic:

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)

Function :

VFPR[rd] ← VFPR[rs] + VFPR[rt]

Exception :

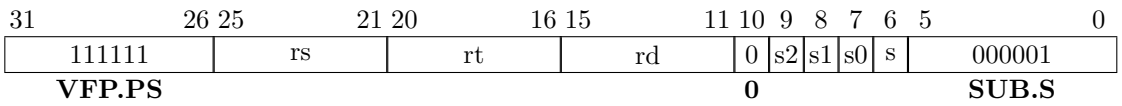
Vector Integer Exception

Overview :

32bitx2, Single-Precision Vector Add. When s0 is 1, execute operation by scalar register (SFPR[rt]) instead of VFPR[rt]. When s2 is 1, execute operation by lower 32bit of VFPR[rt]. When sync is 1, suppress the speculative execution.

VSUB.PS — Vector Subtract Paired Single

Vector Substract VECTOR



Mnemonic:

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)

Function :

VFPR[rd] ← VFPR[rs] - VFPR[rt]

Exception :

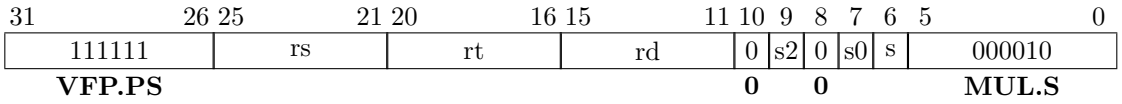
Vector Integer Exception

Overview :

32bitx2, Single-Precision Vector Substract. When s0 is 1, execute operation by scalar register (SFPR[rt]) instead of VFPR[rt]. When s1 is 1, execute operation by scalar register (SFPR[rs]) instead of VFPR[rs]. When s2 is 1, execute operation by lower 32bit of VFPR[rt]. When sync is 1, suppress the speculative execution.

VMUL.PS — Vector Multiply Paired Single

Vector Multiply VECTOR



Mnemonic:

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)

Function :

VFPR[rd] $\leftarrow$ VFPR[rs] $\times$ VFPR[rt]

Exception :

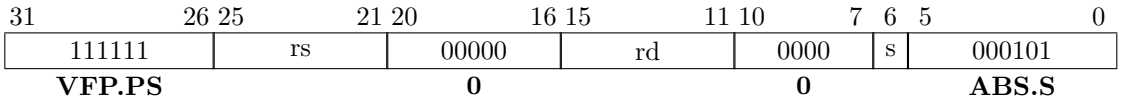
Vector Integer Exception

Overview :

32bitx2, Single-Precision Vector Multiply. When s0 is 1, execute operation by scalar register (SFPR[rt]) instead of VFPR[rt]. When s2 is 1, execute operation by lower 32bit of VFPR[rt]. When sync is 1, suppress the speculative execution.

VABS.PS Vector Absolute Paired Single

Vector Absolute Operation VECTOR



Mnemonic:

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

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

Function :

VFPR[rd] ← |VFPR[rs]|

Exception :

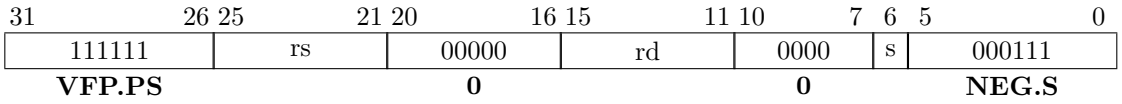
Vector Integer Exception

Overview :

32bitx2, Single-Precision Vector Absolute Operation. When sync is 1, suppress the speculative execution.

VNEG.PS — Vector Negate Paired Single

Vector Negate VECTOR



Mnemonic:

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

Function :

VFPR[rd] ← -1 *times* VFPR[rs]

Exception :

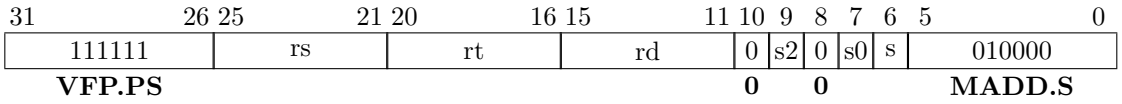
Vector Integer Exception

Overview :

32bitx2, Single-Precision Vector Negate Operataion. When sync is 1, suppress the speculative execution.

VMADD.PS Vector Multiply and Add Paired Single

Vector Multiply and Add Operation VECTOR



Mnemonic:

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)

Function :

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

Exception :

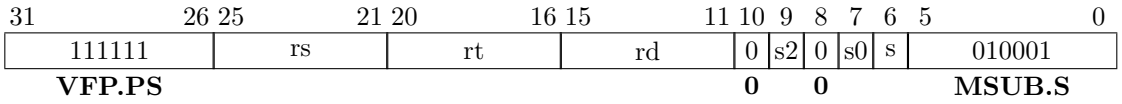
Vector Integer Exception

Overview :

32bitx2, Single-Precision Vector Multiply and Add Operation. When s0 is 1, execute operation by scalar register (SFPR[rt]) instead of VFPR[rt]. When s2 is 1, execute operation by lower 32bit of VFPR[rt]. When sync is 1, suppress the speculative execution.

VMSUB.PS Vector Multiply and Subtract Paired Single

Vector Multiply and Substract VECTOR



Mnemonic:

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)

Function :

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

Exception :

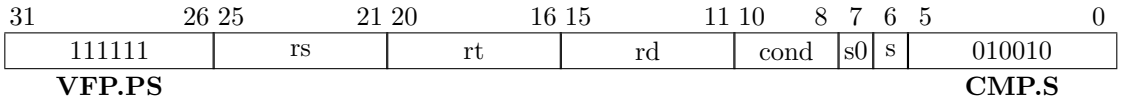
Vector Integer Exception

Overview :

32bitx2, Single-Precision Vector Multiply and Substract Operation. When s0 is 1, execute operation by scalar register (SFPR[rt]) instead of VFPR[rt]. When s2 is 1, execute operation by lower 32bit of VFPR[rt]. When sync is 1, suppress the speculative execution.

VCMP.PS Vector Compare Paired Single

Vector Comparison VECTOR



Mnemonic:

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)

Function :

```
if VFPR[rs] cond VFPR[rt] then
    VFPR[rd] ← 1
else
    VFPR[rd] ← 0
endif
```

Exception :

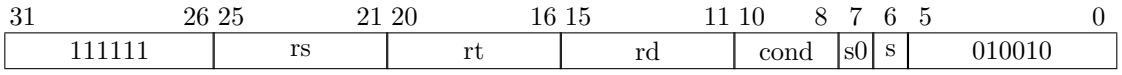
Vector Integer Exception

Overview :

32bitx2, Single-Precision Vector Compare Instruction. Determinate value of VFPR[rd] by condition. When s0 is 1, execute operation by scalar register (SFPR[rt]) instead of VFPR[rt]. When sync is 1, suppress the speculative execution. When set up relation conditions, specifiy which of eq(=), gt(>), lt(<), ne($\neq$), le($\leq$), ge($\geq$).

VSCMP.PS — Vector Compare Paired Single

Vector Comparison VECTOR



Mnemonic:

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)

Function :

```
if VFPR[rs] cond VFPR[rt] then
    VFPR[rd] ← 1
else
    VFPR[rd] ← 0
endif
```

Exception :

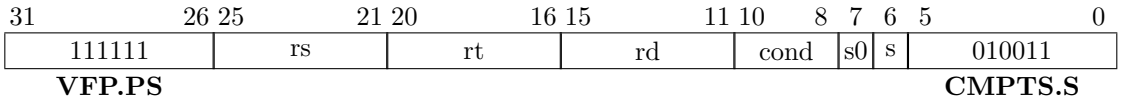
Vector Integer Exception

Overview :

32bitx2, Single-Precision Vector Compare Instruction. Determinate value of VFPR[rd] by condition. Execute operation by lower 32bit of VFPR[rt]. When s0 is 1, execute operation by scalar register (SFPR[rt]) instead of VFPR[rt]. When sync is 1, suppress the speculative execution. When set up relation conditions, specifiy which of eq(=), gt(>), lt(<), ne($\neq$), le($\leq$), ge($\geq$).

VCMPTS.PS Vector Compare Paired Single to Scalar Register

Vector Comparison VECTOR



Mnemonic:

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)

Function :

```
if VFPR[rs] cond VFPR[rt] then
    SFPR[rd] ← 1
else
    SFRP[rd] ← 0
endif
```

Exception :

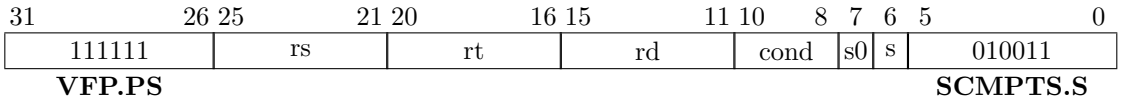
Vector Integer Exception

Overview :

32bitx2, Single-Precision Vector Compare Instruction. Operation result is assigned 1bit for each elements, and stored scalar register. When s0 is 1, execute operation by scalar register (SFPR[rt]) instead of VFPR[rt]. When sync is 1, suppress the speculative execution. When set up relation conditions, specifiy which of eq(=), gt(>), lt(<), ne($\neq$), le($\leq$), ge($\geq$).

VSCMPTS.PS Vector Compare Paired Single to Scalar Register

Vector Comparison VECTOR



Mnemonic:

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)

Function :

```
if VFPR[rs] cond VFPR[rt] then
    SFPR[rd] ← 1
else
    SFPR[rd] ← 0
endif
```

Exception :

Vector Integer Exception

Overview :

32bitx2, Single-Precision Vector Compare Instruction. Operation result is assigned 1bit for each elements, and stored scalar register. Execute operation by lower 32bit of VFPR[rt]. When s0 is 1, execute operation by scalar register (SFPR[rt]) instead of VFPR[rt]. When sync is 1, suppress the speculative execution. When set up relation conditions, specifiy which of eq(=), gt(>), lt(<), ne($\neq$), le($\leq$), ge($\geq$).

4

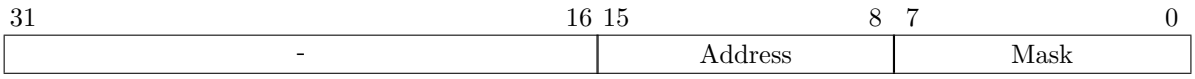
Address Decoder

4.1 Register Interface

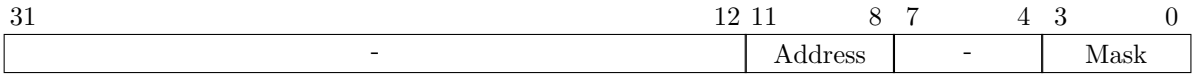
Each of the address decoder configuration registers are defined as a system register. Therefore, to configure, mtc0 instruction is used.

Values are different by a module.

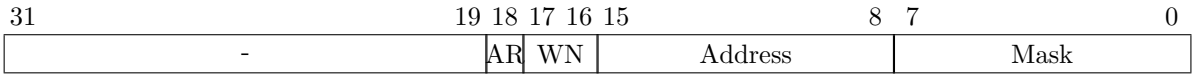
- Standard (TYPE A)



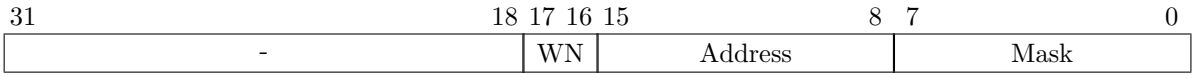
- I/O (TYPE B)



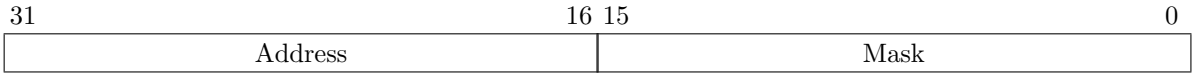
- External Bus (TYPE C)



- Link Memory (TYPE D)



- I/O Base (TYPE E)



Field Name	Function
Address	Base Address
Mask	Mask
WN	Word Number
AR	Auto Ready

4.2 Address Mapping

Connected Module	Initial Decode Address	Configuration Register Address	Type
ROM (EXT_0)	0x00000000 ~ 0x00100000	0x100	TYPE C
LINK SDRAM	0x04000000 ~ 0x04ffffff	0x138	TYPE D
LINK ECC-SDRAM (LSB 8bit valid)	0x05000000 ~ 0x05ffffff	0x139	TYPE D
Trace Buffer	0x10000000 ~ 0x1fffffff	0x153	TYPE A
EXT_1 (FLASH IF)	0x40000000 ~ 0x4fffffff	0x111	TYPE C
EXT_2	0x21000000 ~ 0x21ffffff	0x112	TYPE C
EXT_3	0x22000000 ~ 0x22ffffff	0x113	TYPE C
EXT_4	0x23000000 ~ 0x23ffffff	0x114	TYPE C
EXT_5	0x24000000 ~ 0x24ffffff	0x114	TYPE C
EXT_6	0x25000000 ~ 0x25ffffff	0x114	TYPE C
EXT_7	0x26000000 ~ 0x26ffffff	0x114	TYPE C
FLASH IF Control	0x27000000 ~ 0x27ffffff	NA	TYPE C
SDRAM IF0	0x80000000 ~ 0x87ffffff	0x130	TYPE A
SDRAM IF1	0x88000000 ~ 0x8fffffff	0x131	TYPE A
SDRAM IF2	0x90000000 ~ 0x97ffffff	0x132	TYPE A
SRAM	0x98000000 ~ 0x9fffffff	0x101	TYPE A
LINK DPM	0xc0000000 ~ 0xcfffffff	0x141	TYPE D
LINK	0xfffe0000 ~ 0xfffeffff	0x140	TYPE E
DMAC0	0xffff0000 ~ 0xffff0fff	0x120	TYPE B
DMAC1	0xffff1000 ~ 0xffff1fff	0x121	TYPE B
DMAC2	0xffff2000 ~ 0xffff2fff	0x122	TYPE B
DMAC3	0xffff3000 ~ 0xffff3fff	0x123	TYPE B
DMAC4	0xffff4000 ~ 0xffff4fff	0x124	TYPE B
DMAC DIAG	0xffff5000 ~ 0xffff5fff	0x125	TYPE B
UART0~3	0xffff6000 ~ 0xffff61ff	0x155	TYPE B
PULSE COUNTER0~5	0xffff7000 ~ 0xffff71ff	0x156	TYPE B
PWMOUT0~11	0xffff7200 ~ 0xffff73ff	0x156	TYPE B
PWMIN0~5	0xffff7400 ~ 0xffff75ff	0x156	TYPE B
32bit Timer0~3	0xffff7800 ~ 0xffff79ff	0x156	TYPE B
64bit Timer0~3	0xffff7a00 ~ 0xffff7bff	0x156	TYPE B
SRAM ECC CONTROLER	0xffff7c00 ~ 0xffff7fff	NA	TYPE B
IRC	0xffff9000 ~ 0xffff93ff	0x151	TYPE B
SUB IRC0	0xffff9400 ~ 0xffff97ff	0x151	TYPE B
SUB IRC1	0xffff9800 ~ 0xffff9bff	0x151	TYPE B
CLK Generator	0xffffa000 ~ 0xffffa1ff	0x150	TYPE B
MICRO RESET	0xffffa200 ~ 0xffffa3ff	NA	TYPE B
HIZ CONTROLER	0xffffa400 ~ 0xffffa5ff	NA	TYPE B
SPI	0xffffb000 ~ 0xffffb7ff	0x157	TYPE B
Parallel I/O	0xffffc000 ~ 0xffffc7ff	0x154	TYPE B
I2C	0xffffc800 ~ 0xffffcfff	0x158	TYPE B
256bit DMAC	0xffffd000 ~ 0xffffd7ff	0x12c	TYPE B
LINK SDRAM ECC CONTROLER	0xffffd800 ~ 0xffffdfff	0x142	TYPE B
LINK SDRAM Mode	0xffffe000 ~ 0xffffe7ff	NA	TYPE B
LINK SDRAM (ECC) Mode	0xffffe800 ~ 0xffffefff	NA	TYPE B
SDRAM Mode	0xfffff000 ~ 0xfffff1ff	NA	TYPE B
ECC SDRAM Mode	0xfffff200 ~ 0xfffff3ff	NA	TYPE B
ECC SDRAM CONTROLER	0xfffff400 ~ 0xfffff5ff	NA	TYPE B
RTC	0xfffff600 ~ 0xfffff7ff	0x152	TYPE B

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．

Table 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

Table 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 (全コンテキスト有効)
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．

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．

0 から 7 ビット目に，それぞれコンテキスト番号 0 から 7 が対応する．

この SHARE_TH フィールドはビットを反転して格納される．例えば，SHARE_TH フィールドに 0x0f と設定すると実際に格納される値は 0xf0 となる．

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．

Table 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_RO	111	全モードでの読み込のみを許可
ALL_R	110	全モードでの読み込みと書き込みを許可
USR_RW	101	全モードでの読み込みと書き込みを許可
USR_R	100	全モードでの読み込み，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込みと書き込みを許可
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．

Table 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．

RMTP の仕様上 BURST 無しで使用すること．

Table 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．

Table 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．

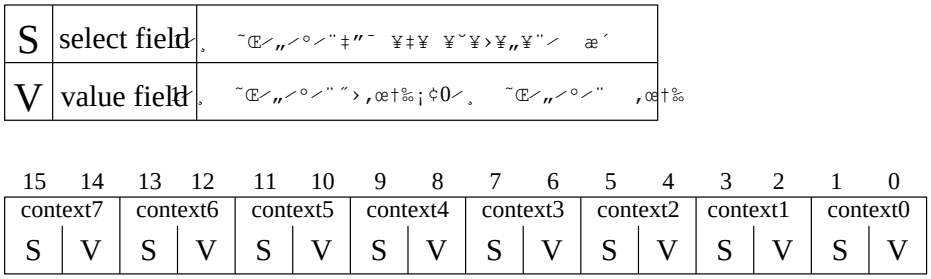


Figure 5.1: コントロールレジスタの共通設定形式

Table 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
0x38	MMU_READ_TLB_ADDR	設定値無し	TLB の読みたいエントリのアドレス
0x3c	MMU_READ_TLB_DATA	設定値無し	TLB の読みたいエントリのデータ
0x40	MMU_GLOBAL_BIT_LOW	直接指定	グローバルビットの下位 32bit
0x44	MMU_GLOBAL_BIT_HIGH	直接指定	グローバルビットの上位 32bit

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．

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．

† ` ¥ ¡…¥,¨ „ : VPN ¥ ¡…¥,° , °: PRO
¥¤¥ ¥¨¥Œ¥ ¥^¥Ω LCK ¶ƒ˝› ° : SHR

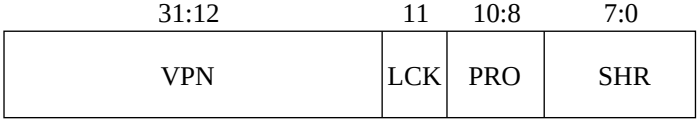


Figure 5.2: ENTRY1 の設定形式

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．

°“˝ ¥ ¡…¥,¨ „ : PPN ¥›¥ª¥^¥•¥ ¥ ¥^¥fCLC
¥ ¡…¥,¥ ¥⁄¥” : PSZ ¥›¥ª¥^¥•¥ †˜ UNC
¥‡¥ ¥˜¥›¥„¥¨¥ ¥°¡…¥GRP ¥–¡…¥„¥¨¯ ` ¯; BRT

31:12	11:10	9:4	3	2	1:0
PPN	PSZ	GRP	CLC	UNC	BRT

Figure 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．

S	select field	˜Œ⁄„⁄°⁄¨‡”¯ ¥‡¥ ¥˜¥›¥„¥¨⁄ æ´
V	value field	˜Œ⁄„⁄°⁄¨˝›‚œ†‰¡¢0⁄ ˜Œ⁄„⁄°⁄¨ ‚œ†‰

21 16	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

Figure 5.4: コンテキストグループの設定形式

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．

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．

Table 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_RO	001	全モードでの読み出し制限違反
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，5bit 目 [4] 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．

MMU_READ_TLB_ADDR

MMU_READ_TLB_DATA

この 2 つのレジスタを用いて TLB の中身を読むことができる．まず読みたい TLB のエントリを MMU_READ_TLB_ADDR レジスタに書き込み，次に MMU_READ_TLB_DATA レジスタを読み出す．MMU_READ_TLB_ADDR レジスタに書き込む値は 6:1 bit 目にエントリの番号，0bit 目に Entry1 ならば 0，Entry2 ならば 1 を指定する．ただし，Entry1 の SHARE_TH のビットは反転した値が読みだされる．

MMU_GLOBAL_BIT_LOW

MMU_GLOBAL_BIT_HIGH

グローバルビットを指定することで，SHARE_TH ビットに関係なく TLB エントリの内容は全てのスレッドで有効となる．つまりグローバルビットがセットされているエントリは全てのスレッドで共通のアドレス空間となり，コンテキストスイッチが発生した際もエントリの無効化は発生しない．これはカーネル空間などの共有領域や，TLB ミスが発生してはいけない領域などでの利用を想定している．グローバルビットを使用する際には，同時にそのエントリをロックすることを強く推奨する．グローバルビットはエントリ番号に紐付けされているため，もし当該エントリが TLB の置換対称となった場合，置換後のエントリ内容もグローバルと

なってしまう．これは意図しない動作の原因になるだろう．グローバルビットはリセット時と ALL_FLUSH を行った際にのみ自動的に初期化される．グローバルビットは MMU_GLOBAL_BIT_LOW が TLB エントリの 0 番から 31 番，MMU_GLOBAL_BIT_HIGH が TLB エントリの 32 番から 63 番に対応し，LSB 側から TLB エントリの若番に対応する．
（e.g. MMU_GLOBAL_BIT_LOW[0] => TLB_ENTRY[0], MMU_GLOBAL_BIT_HIGH[0] => TLB_ENTRY[32]）

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．

Table 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
エントリミス	要求	0x3	0x8
ページ保護違反	要求	0x5	0xa

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．

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

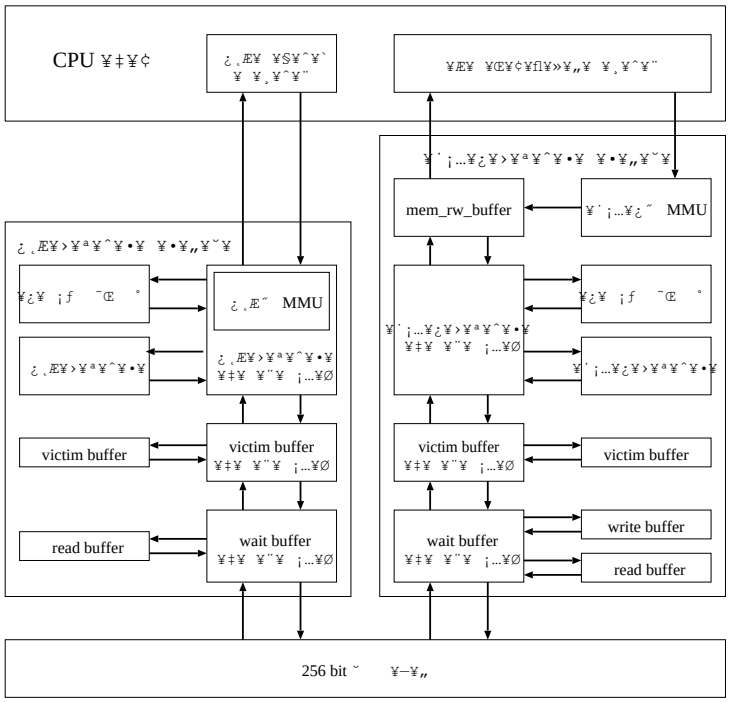


Figure 6.1: キャッシュシステムのモジュール構成

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．

Table 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

Table 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)

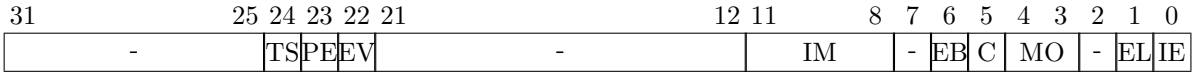
offset	31 24 23 16 15 8 7 0
0x58	Interruption Wait Register (Thread0)
0x59	Interruption Wait Register (Thread1)
0x5a	Interruption Wait Register (Thread2)
0x5b	Interruption Wait Register (Thread3)
0x5c	Interruption Wait Register (Thread4)
0x5d	Interruption Wait Register (Thread5)
0x5e	Interruption Wait 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	Interruption Pending Register
0x69	Interruption Clear Register
0x6a	Exception Base Address Register
0x6b	Interruption Mode Register
0x6c ～ 0x6f	-
0x70 ～ 0x77	Event Link In Register
0x78 ～ 0x7f	Event Link Out Register
0x80 ～ 0x84	Instruction Cache Control Register
0x86 ～ 0x8c	Data Cache Control Register
0x90	Multiplexer Arbitor Mode Bus
0x91	Multiplexer Arbitor Priority 256bit Bus
0x92	Multiplexer Arbitor Priority High 32bit Bus
0x93	Multiplexer Arbitor Priority Low 32bit Bus
0x94	Multiplexer Watchdog Timer 256bit Bus Enable
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
0xb8	IO BAse Address Decoder Control Register
0xb9	Multiplexer Watchdog Timer 32bit Bus Enable
0xba	Multiplexer Error Handler Master32
0xbb	Multiplexer Error Handler Master256
0xbc	Multiplexer Watchdog Timer 32bit Bus Count
0xc9	Reservation Station Aging
0xca	Reservation Station Aging Increment
0xcb	Reservation Station Aging Span
0xcc	PID Parameter Register (Thread0～1)
0xcd	PID Parameter Register (Thread2～3)
0xce	PID Parameter Register (Thread4～5)
0xcf	PID Parameter Register (Thread6～7)
0xd0	Target IPC Register (Thread0)
0xd1	Target IPC Register (Thread1)
0xd2	Target IPC Register (Thread2)
0xd3	Target IPC Register (Thread3)
0xd4	Target IPC Register (Thread4)

offset	31 24 23 16 15 8 7 0
0xd5	Target IPC Register (Thread5)
0xd6	Target IPC Register (Thread6)
0xd7	Target IPC Register (Thread7)
0xd8	Fetch Bound Register (Thread0)
0xd9	Fetch Bound Register (Thread1)
0xda	Fetch Bound Register (Thread2)
0xdb	Fetch Bound Register (Thread3)
0xdc	Fetch Bound Register (Thread4)
0xdd	Fetch Bound Register (Thread5)
0xde	Fetch Bound Register (Thread6)
0xdf	Fetch Bound Register (Thread7)
0xe0	Own Status Register
0xe1	Own Thread Table Register
0xe2	Own Thread ID Register
0xe3	Own Instruction Count Register
0xe4	Own Count Register
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
0xec	Own Target IPC Register
0xed	Own Fetch Bound Register
0xf0	Special Mode Register
0xf1	NMI Mode Register
0xf2	Special Operation Register
0xf3	I Cache ECC ON Register
0xf4	I Cache ECC Mode Register
0xf5	Multiplexer Error Handler I-Cache
0xf6	Multiplexer Error Handler D-Cache
0xf7	D Cache ECC ON Register
0xf8	D Cache ECC Mode Register
0xfd	Commit Mode
0xfe	Proceccor ID
0xff	Chip Version Register
0x100	ROM Address Decoder Control Register
0x101	SRAM Address Decoder Control Register
0x110	EXT0 Address Decoder Control Register
0x111	EXT1 Address Decoder Control Register
0x112	EXT2 Address Decoder Control Register
0x113	EXT3 Address Decoder Control Register
0x114	EXT4 Address Decoder Control Register
0x115	EXT5 Address Decoder Control Register
0x116	EXT6 Address Decoder Control Register
0x117	EXT7 Address Decoder Control Register
0x120	DMAC0 Address Decoder Control Register
0x121	DMAC1 Address Decoder Control Register
0x122	DMAC2 Address Decoder Control Register
0x123	DMAC3 Address Decoder Control Register

offset	31 24 23 16 15 8 7 0
0x124	DMAC4 Address Decoder Control Register
0x125	DMAC5 Address Decoder Control Register
0x126	DMAC6 Address Decoder Control Register
0x127	DMAC7 Address Decoder Control Register
0x128 ～ 0x12b	DMAC DIAG Address Decoder Control Register
0x12c	DMAC256 Address Decoder Control Register
0x130	SDRAM IF0 Address Decoder Control Register
0x131	SDRAM IF1 Address Decoder Control Register
0x132	SDRAM IF2 Address Decoder Control Register
0x133	SDRAM IF3 Address Decoder Control Register
0x138	LINK SDRAM IF Address Decoder Control Register
0x139	LINK SDRAM(ECC) IF Address Decoder Control Register
0x13c	SDRAM IF Mode
0x13d	LINK SDRAM IF Mode
0x13e	LINK SDRAM(ECC) IF Mode
0x140	LINK Address Decoder Control Register
0x141	LINK DPM Address Decoder Control Register
0x142	LINK ECC Control Address Decoder Control Register
0x150	Clock Generator Address Decoder Control Register
0x151	IRC Address Decoder Control Register
0x152	RTC Address Decoder Control Register
0x153	Trace Buffer Address Decoder Control Register
0x154	PIO Address Decoder Control Register
0x155	UART Address Decoder Control Register
0x156	PP Address Decoder Control Register
0x157	SPI Address Decoder Control Register
0x158	I2C Address Decoder Control Register
0x159	PCI Address Decoder Control Register
0x15a	Ethernet Address Decoder Control Register
0x15b	IEEE1394 Address Decoder Control Register
0x15c, 0x15d	USB Address Decoder Control Register
0x170	Extbus1 Status
0x171	Extbus2 Status
0x172	Extbus3 Status
0x173	Extbus4 Status
0x174	Extbus5 Status
0x175	Extbus6 Status
0x176	Extbus7 Status
0x178	ROM Status
0x179	E2M Status
0x17a	Extbus Mem IO
0x180	Multiplexer Error Handler DMAC0
0x181	Multiplexer Error Handler DMAC1
0x182	Multiplexer Error Handler DMAC2
0x183	Multiplexer Error Handler DMAC3
0x184	Multiplexer Error Handler DMAC4
0x185	Multiplexer Error Handler DMAC5
0x188	Multiplexer Error Handler Extbus0
0x189	Multiplexer Error Handler Extbus1
0x18a	Multiplexer Error Handler Extbus2
0x18b	Multiplexer Error Handler Extbus3
0x18c	Multiplexer Error Handler Extbus4
0x18d	Multiplexer Error Handler Extbus5
0x18e	Multiplexer Error Handler Extbus6
0x18f	Multiplexer Error Handler Extbus7
0x1a1	Multiplexer Error Handler MDMAC256

7.1.1 Status Register

Address: 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．



Field Name	Function
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 EBA レジスタで指定した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 (0x60000000) 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

Address: 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．

Field Name	Function
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: Ready 0011: 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.

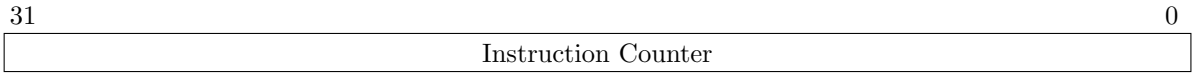
7.1.3 Thread ID Register

Address: 0x10 〜 0x17 (スレッド毎)

Field Name	Function
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

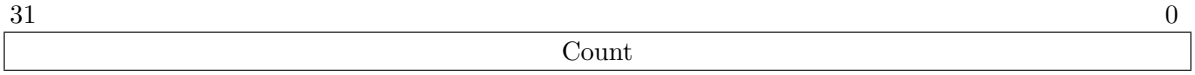
Address: 0x18 〜 0x1f (スレッド毎)



Field Name	Function
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

Address: 0x20 〜 0x27 (スレッド毎)



Field Name	Function
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

Address: 0x28 〜 0x2f (スレッド毎)

Field Name	Function
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

Address: 0x30 〜 0x37 (スレッド毎)

Field Name	Function
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

Address: 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

Field Name	Function
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

Address: 0x39, 0x3a

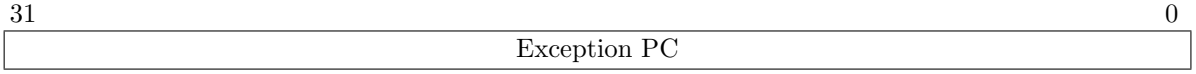
Field Name	Function
CPU Count	64bit カウンタ．リセット時から毎クロック 1 ずつカウントアップしていく．0x39 が下位 32bit，0x3a が上位 32bit を示す．

7.1.10 MMU Register

Address: 0x3b 〜 0x47
MMU 関連の設定レジスタ．

7.1.11 Exception PC Register

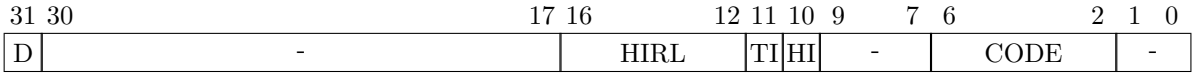
Address: 0x48 〜 0x4f (スレッド毎)



Field Name	Function
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

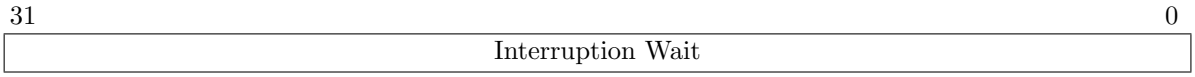
Address: 0x50 〜 0x57 (スレッド毎)
発生した例外の情報を保持するレジスタ．0x00000000 に初期化される．



Field Name	Function
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.
Timer Interruption Pending (TI))	タイマが満了した際に 1 が書き込まれる. タイマ割込みにより起床したタスクは手動でこのビットをクリアする必要がある.
HI	Hardware Interruption 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)

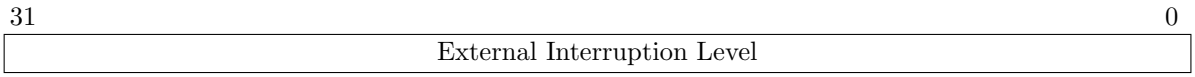
Address: 0x58 〜 0x5f



Field Name	Function
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)

Address: 0x60 〜 0x67



Field Name	Function
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

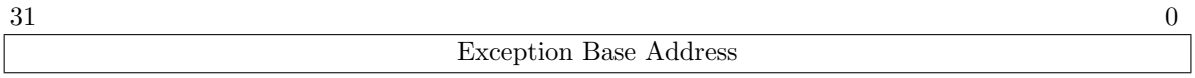
Address: 0x68
現在 (多分) 使用していない.

7.1.16 Interruption Clear Register

Address: 0x69
現在 (多分) 使用していない.

7.1.17 Exception Base Address Register

Address: 0x6a



Field Name	Function
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．Status Register の EV ビットを 1 にすると，例外発生時に例外の種類に依らず EBA の番地に制御が移る．両方を 1 にした場合 EBAE レジスタが優先される．

7.1.18 Event Link In Register

Address: 0x70 〜 0x73

7.1.19 Event Link Out Register

Address: 0x74 〜 0x77

7.1.20 Instruction Cache Control Register

Address: 0x80〜0x84
6 章 (CACHE) のキャッシュのコントロールレジスタを参照.

7.1.21 Data Cache Control Register

Address: 0x86〜0x8c
6 章 (CACHE) のキャッシュのコントロールレジスタを参照.

7.1.22 Multiplexer Arbitor Mode Bus

Address: 0x90

31 … 5	4	3	2	1	0
Reserved	pac	PM1	PM0	M1	M0

Field Name	Function
M0(256bit arbiter mode)	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．固定優先度モードではより高い値をセットされたバスマスタにより高い優先度が与えられる．0 に初期化され優先度が等しい場合には下位ビットのバスマスタが優先される．
M1(32bit arbiter mode)	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．固定優先度モードではより高い値をセットされたバスマスタにより高い優先度が与えられる．0 に初期化され優先度が等しい場合には下位ビットのバスマスタが優先される．

7.1.23 Multiplexer Arbitor Priorty 256bit Bus

Address: 0x91

31 … 8	7	6	5	4	3	2	1	0
Reserved	IC		DC		DMAC256		32bit Bus	

256bit Bus のバスマスタの優先度を設定する．デフォルトは 0 (最低優先度).

7.1.24 Multiplexer Arbitor Priorty High 32bit Bus

Address: 0x92

31 … 28	27 … 24	23 … 20	19 … 16	15 … 12	11 … 8	7 … 4	3 … 0
32bit Bus	OCE	DMA5	DMA4	DMA3	DMA2	DMA1	DMA0

32bit Bus のバスマスタの優先度を設定する．デフォルトは 0 (最低優先度).

7.1.25 Multiplexer Arbitor Priorty Low 32bit Bus

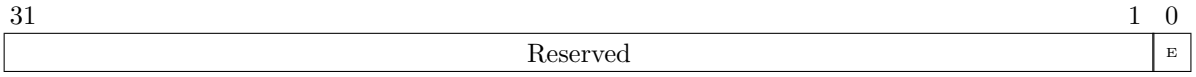
Address: 0x93

31 … 28	27 … 24	23 … 20	19 … 16	15 … 12	11 … 8	7 … 4	3 … 0
Extbus7	Extbus6	Extbus5	Extbus4	Extbus3	Extbus2	Extbus1	Extbus0

32bit Bus のバスマスタの優先度を設定する．デフォルトは 0 (最低優先度).

7.1.26 Multiplexer Watchdog Timer 256bit Bus Enable

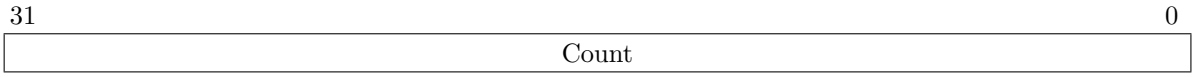
Address: 0x94



Field Name	Function
E	Default: 0 0: Disable 1: Enable このレジスタと Watchdog Timer 256bit Count の両方を設定しないと有効にならない.

7.1.27 Multiplexer Watchdog Timer 256bit Bus Count

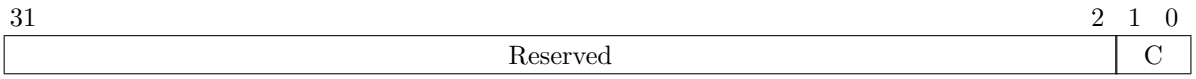
Address: 0x95



Field Name	Function
Count	Default: 0 256bit バスに Address Strobe が降りてから，Ready が返ってくるまでの時間が書き込まれた値のクロックサイクル数を超えた場合，Watchdog Timer Error 割込みを発生させる. Watchdog Timer Enable が設定されていない場合，無効.

7.1.28 Multiplexer Error Handler State 256bit Bus

Address: 0x96
エラーを起こしている箇所を示すレジスタ.
エラーの詳細は対応するエラーハンドルレジスタを参照する.



Field Name	Function
C	Default: 0 00: I Cache Error 01: D Cache Error 10: DMAC256 Error 11: 32bit bus Error

7.1.29 Multiplexer Error Handler State 32bit Bus

Address: 0x97

7.1.30 Multiplexer Error Handler Instruction Cache

Address: 0x98
Instruction Cache がバスマスタの場合のエラーハンドラ.

31 … 8	7	6	5 … 0
Reserved	SE		ES

Field Name	Function
SE	エラーの種類を示す. Default: 0 00: No Error 01: Reserved 10: Address Error (無効なアドレス) 11: Watchdog Timer Error
ES	エラーを起こしたスレーブを示す. Default: 0 0x00: SDRAM IF 0 0x01: SDRAM IF 1 0x02: SDRAM IF 2 0x03: SDRAM IF 3 0x04: SDRAM IF Mode 0x05: ROM 0x06: SRAM 0x07: DMAC 256 0x08: DMAC 0 0x09: DMAC 1 0x0a: DMAC 2 0x0b: DMAC 3 0x0c: DMAC 4 0x10: Extbus 0 0x11: Extbus 1 0x12: Extbus 2 0x13: Extbus 3 0x14: PCI 0x15: Ethernet 0x16: IEEE1394 0x17: UART 0x18: PP 0x19: IRC 0x1a: Clock Generator 0x1b: SPI 0x1c: PIO 0x1d: RTC 0x1e: I2C 0x1f: Trace Buffer 0x20: Responsive Link 0x21: Responsive Link DPM 0x22: Link SDRAM IF 0x23: Link SDRAM IF Mode 0x24: Link SDRAM ECC IF 0x25: Link SDRAM ECC IF Mode 0x26: Link SDRAM ECC Controler

7.1.31 Multiplexer Error Handler Data Cache

Address: 0x99

Dada Cache がバスマスタの場合のエラーハンドラ．7.1.30 項の Multiplexer Error Handler Instruction Cache と同様．

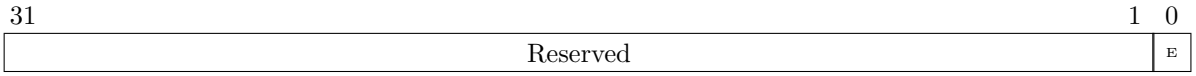
7.1.32 Multiplexer Error Handler MDMAC256

Address: 0x1a1

DMAC256 がバスマスタの場合のエラーハンドラ．7.1.30 項の Multiplexer Error Handler Instruction Cache と同様．

7.1.33 Multiplexer Watchdog Timer 32bit Bus Enable

Address: 0xb9



Field Name	Function
E	Default: 0 0: Disable 1: Enable このレジスタと Watchdog Timer 32bit Count の両方を設定しないと有効にならない．

7.1.34 Multiplexer Error Handler Master32

Address: 0xba

現在使用されていない．

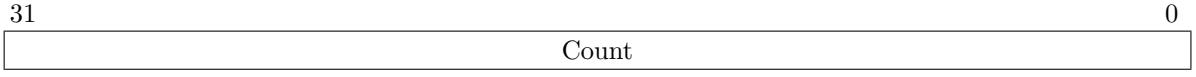
7.1.35 Multiplexer Error Handler Master256

Address: 0xbb

現在使用されていない．

7.1.36 Multiplexer Watchdog Timer 32bit Bus Count

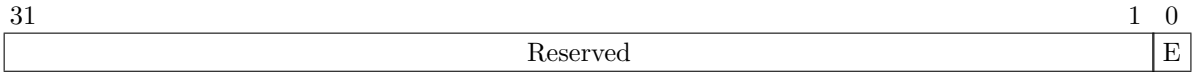
Address: 0xbc



Field Name	Function
Count	Default: 0 32bit バスに Address Strobe が降りてから，Ready が返ってくるまでの時間 (単位: Clock cycle) が書き込まれた値のクロックサイクル数を超えた場合，Watchdog Timer Error 割込みを発生させる． Watchdog Timer Enable が設定されていない場合，無効．

7.1.37 Reservation Station Aging

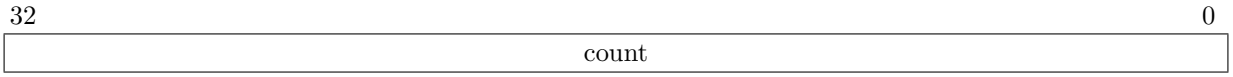
Address: 0xc9



Reservation Station のエントリのエイジングを行う

7.1.38 Reservation Station Aging Increment

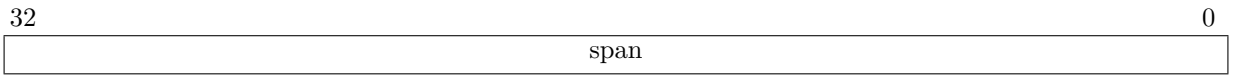
Address: 0xca



Reservation Station のエントリのエイジング量

7.1.39 Reservation Station Aging Span

Address: 0xcb

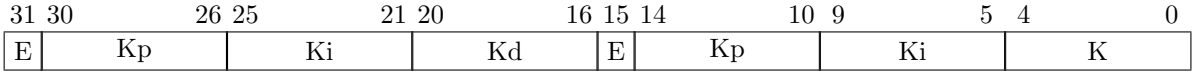


Reservation Station のエントリのエイジングスパン

7.1.40 PID Parameter Register

Address: 0xcc 〜 0xcf (2 スレッド毎)

PID 制御の比例ゲインと積分ゲイン，微分ゲインの変更を行う．Target IPC Register で PID 制御を用いない場合には無意味．31〜16 ビットが奇数スレッド用で，15〜0 ビットが偶数スレッド用．



Field Name	Function
E	PID Parameter Enable 0: ハードウェアで設定されたゲインを用いる． 1: ソフトウェアから任意のゲインを設定する．
Kp，Ki，Kd	10000: 5 ビット右シフトする．($\frac{1}{32}$) 01000: 4 ビット右シフトする．($\frac{1}{16}$) 00100: 3 ビット右シフトする．($\frac{1}{8}$) 00010: 2 ビット右シフトする．($\frac{1}{4}$) 00001: 1 ビット右シフトする．($\frac{1}{2}$) 複数のビットを指定した場合には，シフト後に合計する． 例: 10011 と指定した場合には，ゲインは $\frac{1}{32}+\frac{1}{4}+\frac{1}{2}=\frac{25}{32}$ となる．

7.1.41 Target IPC Register

Address: 0xd0 〜 0xd7 (各スレッド毎)
IPC 制御を有効にする．Compare Register を設定しないと動作しない．

31	30	29	28	…	0
E	MO	Target IPC			

Field Name	Function
E	IPC Enable 0: IPC 制御を行わない． 1: IPC 制御を行う．
MO	IPC 制御に用いる制御手法の選択 00: PID 制御 01: フィードフォワード制御 それ以外: PID 制御
Target IPC	目標の IPC(実際には，Compare Register で設定したクロックサイクル数内に実行する命令数) を設定 例: Compare Register を 10000 に設定していて目標 IPC を 0.7 にしたい場合には，7000 を指定．

7.1.42 Fetch Bound Register

Address: 0xd8 〜 0xdf (2 スレッド毎)

31	…	0
Fetch Bound		

Field Name	Function
Fetch Bound	フェッチ数の上限値．

7.1.43 Own Status Register

Address: 0xe0
自身のコンテキスト番号の Status Register を参照する

7.1.44 Own Thread Table Register

Address: 0xe1
自身のコンテキスト番号の Thread Table Register を参照

7.1.45 Own Thread ID Register

Address: 0xe2
自身のコンテキスト番号の Thread ID Register を参照

7.1.46 Own Instruction Count Register

Address: 0xe3
自身のコンテキスト番号の Instruction Count Register を参照

7.1.47 Own Count Register

Address: 0xe4
自身のコンテキスト番号の Count Register を参照

7.1.48 Own Compare Register

Address: 0xe5
自身のコンテキスト番号の Compare Register を参照

7.1.49 Own Floating-Point Control Register

Address: 0xe6
自身のコンテキスト番号の Floating-Point Control Register を参照

7.1.50 Own Bad Virtual Address Register

Address: 0xe7
自身のコンテキスト番号の Bad Virtual Address Register を参照

7.1.51 Own Exception PC Register

Address: 0xe8
自身のコンテキスト番号の Exception PC Register を参照

7.1.52 Own Exception Cause Register

Address: 0xe9
自身のコンテキスト番号の Exception Cause Register を参照

7.1.53 Own Interruption Wait Register

Address: 0xea
自身のコンテキスト番号の Interruption Wait Register を参照

7.1.54 Own External Interruption Level Register

Address: 0xeb
自身のコンテキスト番号の External Interruption Level Register を参照

7.1.55 Own Target IPC Register

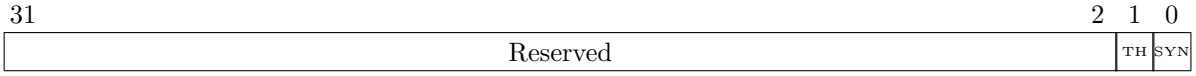
Address: 0xec
自身のコンテキスト番号の Target IPC Register を参照

7.1.56 Own Fetch Bound Register

Address: 0xed
自身のコンテキスト番号の Fetch Bound Register を参照

7.1.57 Special Mode Register

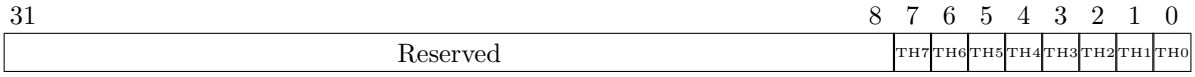
Address: 0xf0



Field Name	Function
Thread (TH)	default: 0 マルチスレッディングを利用する場合に 1 にセットする． 0: rstrth 命令等によって復帰したスレッドは自動的に実行を開始する．(Active Thread RUN 状態) 1: 復帰したスレッドは自動的に実行開始せず（Active Thread STOP 状態），runth 命令により明示的に実行を開始させる必要がある．
Sync (SYN)	default: 0 0: Sync MODE 無効 1: 全ての命令に強制的に sync を挿入

7.1.58 NMI Mode Register

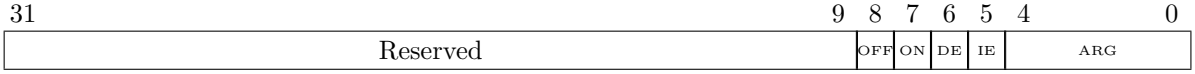
Address: 0xf1



Field Name	Function
Thread (TH)	default: TH0 = 1 外部からの NMI の入力を受け取るスレッドを設定する． 該当するスレッドは NMI の入力があった場合，指定されたアドレスへジャンプする． Status Register の EBAE ビットを 1 に設定し，Exception Base Address Register に例外ベクタのベースアドレスを設定する必要がある． EBAE ビットが 0 の場合，アドレスは 0x80000000 に設定される．

7.1.59 Special Operation Register

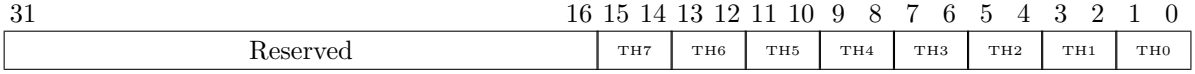
Address: 0xf2 命令を実行したコンテキストの system register に対して書き込みを行う．(write only)



Field Name	Function
ARG	0: 全ての割込み源 other: 指定した bit 番目の割込み源
IE	0: 割込みを無効化 (status レジスタの IE と同様)．1: 割込みを有効化 (status レジスタの IE と同様)．
DE	1: 割込みを無効化 (status レジスタの IE と同様)．0: 割込みを有効化 (status レジスタの IE と同様)．DE を 1 にしていた場合 IE の値に関係なく割込みが無効化される．
ON	ARG で指定した場所の interruption wait を有効化．
OFF	ARG で指定した場所の interruption wait を無効化．OFF を 1 にしていた場合 ON の値に関係なく割込みが無効化される．

7.1.60 I Cache ECC ON Register

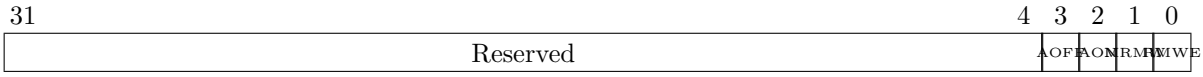
Address: 0xf3



Field Name	Function
TH	I Cache の ECC ON，OFF を設定する．デフォルトでは全て off. このレジスタの設定には共通設定形式 (図 5.1) を用いる． 1 つでも ECC ON のスレッドが存在する場合，Write と Invalidation はすべて ECC ON で行われる．Read はそのスレッドの ECC ON，OFF に従う．

7.1.61　I Cache ECC Mode Register

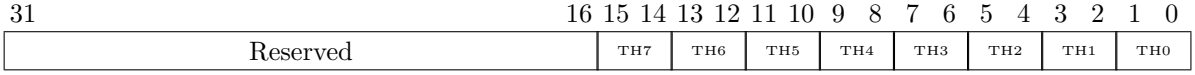
Address: 0xf4



Field Name	Function
AOFF	All ECC OFF 0: 何もしない． 1: 全スレッド及び Invalidation の ECC を強制的に OFF にする． ECC ON レジスタや，ALL ECC ON より優先される．
AON	All ECC ON 0: 何もしない． 1: 全スレッド及び Invalidation の ECC を強制的に ON にする． ECC ON レジスタや，ECC OFF より優先される．
MRMW	Manual Read Modify Write 0: 何もしない． 1: Read modefy Write が始まる．完了すると 0 に戻る．
RMWE	Read Modify Write Enable 0: ECC OFF から ECC ON となっても Read Modify Write は行わない． 1: ECC OFF から ECC ON となった際に Read modefy Write を行う．

7.1.62　D Cache ECC ON Register

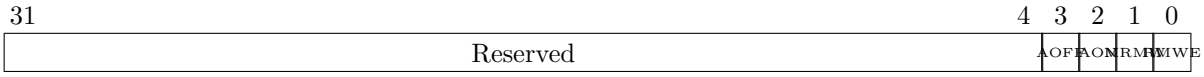
Address: 0xf7



Field Name	Function
TH	D Cache の ECC ON，OFF を設定する．デフォルトでは全て off. このレジスタの設定には共通設定形式 (図 5.1) を用いる． 1 つでも ECC ON のスレッドが存在する場合，Write と Invalidation はすべて ECC ON で行われる．Read はそのスレッドの ECC ON，OFF に従う．

7.1.63 D Cache ECC Mode Register

Address: 0xf8



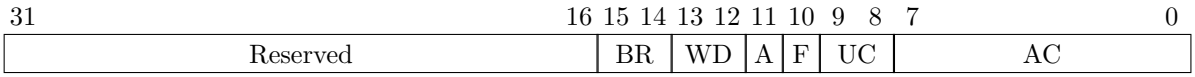
Field Name	Function
AOFF	All ECC OFF 0: 何もしない. 1: 全スレッド及び Invalidation の ECC を強制的に OFF にする. ECC ON レジスタや，ALL ECC ON より優先される.
AON	All ECC ON 0: 何もしない. 1: 全スレッド及び Invalidation の ECC を強制的に ON にする. ECC ON レジスタや，ECC OFF より優先される.
MRMW	Manual Read Modify Write 0: 何もしない. 1: Read modefy Write が始まる．完了すると 0 に戻る.
RMWE	Read Modify Write Enable 0: ECC OFF から ECC ON となっても Read Modify Write は行わない. 1: ECC OFF から ECC ON となった際に Read modefy Write を行う.

7.1.64 Address Decoder Control Register

Address: 0x100 〜 0x139, 0x140 〜 0x158
4 章 (アドレスデコーダ) のアドレスマップを参照.

7.1.65 Extbus0 Status

Address: 0x170 Default: 0x3000



Field Name	Function
BR	バースト長を設定する． 00: default 10: IO 11: MEMORY
WD	バスのアクセス幅を設定する． 11: 32bit 01: 16bit 10: 8bit
A	Auto Ready の設定をする． 0: Disable 1: Enable
F	Flash IF を使用するか設定する．この設定は Extbus1 のみで有効になる． 0: Disable 1: Enable
UC	Un cache を設定する (現在は使用していない)．
AC	Auto Ready を返すまでのカウント値を設定する．

7.1.66 Extbus1 Status

Address: 0x170
7.1.65 項の Extbus0 Status と同様．

7.1.67 Extbus2 Status

Address: 0x171
7.1.65 項の Extbus0 Status と同様．

7.1.68 Extbus3 Status

Address: 0x172
7.1.65 項の Extbus0 Status と同様．

7.1.69 Extbus4 Status

Address: 0x173
7.1.65 項の Extbus0 Status と同様．

7.1.70 Extbus5 Status

Address: 0x174
7.1.65 項の Extbus0 Status と同様．

7.1.71 Extbus6 Status

Address: 0x175
7.1.65 項の Extbus0 Status と同様.

7.1.72 Extbus7 Status

Address: 0x176
7.1.65 項の Extbus0 Status と同様.

7.1.73 ROM Status

Address: 0x178
現在は使用していない

7.1.74 E2M Status

Address: 0x179
現在は使用していない

7.1.75 Extbus Mem IO

Address: 0x17a
現在は使用していない

7.1.76 Multiplexer Error Handler DMAC0-5

Address: 0x180 〜 0x185
7.1.30 項の Multiplexer Error Handler Instruction Cache と同様.

7.1.77 Multiplexer Error Handler Extbus0-7

Address: 0x188 〜 0x18f
7.1.30 項の Multiplexer Error Handler Instruction Cache と同様.

8

例外処理

8.1 割り込みコントローラ (IRC)

割り込みコントローラ (IRC) は，同じ機能の 3 つの IRC がカスケードされています．Sub IRC の割り込み出力 (IRL) は OR されて Main IRC の IRQ26, 27 に接続されています．Main IRC の IRL が RMT PU のコアに接続されています．

Initial Address: Main IRC: 0xffff9000, Sub IRC1: 0xffff9400, Sub IRC2: 0xffff9800

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 26ch 25ch 24ch	23ch 22ch 21ch 20ch 19ch 18ch 17ch		16ch
0x04	15ch 14ch 13ch 12ch 11ch 10ch 9ch 8ch	7ch 6ch 5ch 4ch 3ch 2ch 1ch		0x00
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

Offset: 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		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		7ch		6ch		5ch		4ch		3ch		2ch		1ch		0x00	

Field Name	Function
Trigger	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．

bit	Trigger Mode
00	High Level
01	Low Level
10	Rise Edge
11	Fall Edge

8.1.3 Request Sense Register

31 … 1	0
Request Sense Register	0

Field Name	Function
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

31 … 1	0
Request Clear Register	0

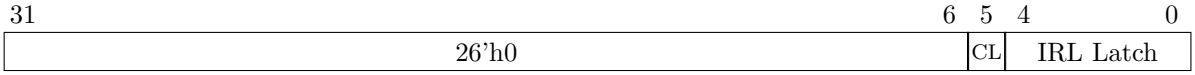
Field Name	Function
Request Clear	Request Clear Register の bit31-1 の中で 1 がセットされるとそれに対応する保持されていた割り込み要求が 0 になる．write のみ．

8.1.5 Mask Register

31 … 1	0
Mask Register	MI

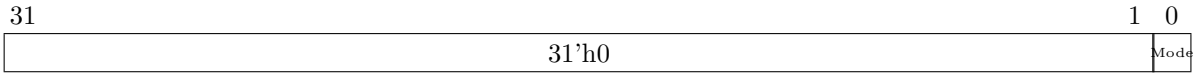
Field Name	Function
Mask	31-1 ビットが割り込みチャネルの 31-1 に対応し，1 をセットすることで割り込みをマスクできる．ただし，Mask が 1 の場合でも Request Sense Register はセットされる．
MI	0 ならば，IRLOUT に割り込みレベルラッチの内容を出力．1 ならばマスクし，IRLOUT には “L” を出力

8.1.6 IRL Latch/Clear



Field Name	Function
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



Field Name	Function
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] に出力されますが，この機能は使用しないでください．

表に Main IRC の割り込みマップを，表に Sub IRC の割り込みマップを示します．Sub IRC の割り込み出力 (IRL) は OR されて Main IRC の IRQ16 にカスケード接続されています．

Table 8.1: Main IRC 割り込みマップ

IRQ31	Bus Error
IRQ30	Address Error
IRQ29	Watch Dog Timer Error
IRQ28	Responsive Link
IRQ27	Sub IRC2
IRQ26	Sub IRC1
IRQ25	Reserved
IRQ24	SRAM ECC Fatal Error
IRQ23	SRAM ECC Correct Error
IRQ22	SDRAM ECC Fatal Error
IRQ21	SDRAM ECC Correct Error
IRQ20	I-Cache ECC Fatal Error
IRQ19	I-Cache ECC Correct Error
IRQ18	D-Cache ECC Fatal Error
IRQ17	D-Cache ECC Correct Error
IRQ16	Reserved
IRQ15	64-bit Timer 1
IRQ14	64-bit Timer 0
IRQ13	32-bit Timer 1
IRQ12	32-bit Timer 0
IRQ11	RTC
IRQ10	External IO 0
IRQ9	DMAC2
IRQ8	DMAC1
IRQ7	DMAC0
IRQ6	DMAC256
IRQ5	SPI 0
IRQ4	I^2C
IRQ3	GPIO
IRQ2	UART 1
IRQ1	UART 0

Table 8.2: Sub IRC1 割り込みマップ

IRQ31	Reserved
IRQ30	Reserved
IRQ29	Pulse Counter 5
IRQ28	Pulse Counter 4
IRQ27	Pulse Counter 3
IRQ26	Pulse Counter 2
IRQ25	Pulse Counter 1
IRQ24	Pulse Counter 0
IRQ23	Reserved
IRQ22	Reserved
IRQ21	PWM Input 5
IRQ20	PWM Input 4
IRQ19	PWM Input 3
IRQ18	PWM Input 2
IRQ17	PWM Input 1
IRQ16	PWM Input 0
IRQ15	Reserved
IRQ14	Reserved
IRQ13	Reserved
IRQ12	PWM11
IRQ11	PWM10
IRQ10	PWM9
IRQ9	PWM8
IRQ8	PWM7
IRQ7	PWM6
IRQ6	PWM5
IRQ5	PWM4
IRQ4	PWM3
IRQ3	PWM2
IRQ2	PWM1
IRQ1	PWM0

Table 8.3: Sub IRC2 割り込みマップ

IRQ31	Reserved
IRQ30	Reserved
IRQ29	64-bit Timer 3
IRQ28	64-bit Timer 2
IRQ27	Reserved
IRQ26	Reserved
IRQ25	32-bit Timer 3
IRQ24	32-bit Timer 2
IRQ23	Reserved
IRQ22	Reserved
IRQ21	Reserved
IRQ20	Reserved
IRQ19	Reserved
IRQ18	DMAC5
IRQ17	DMAC4
IRQ16	DMAC3
IRQ15	Reserved
IRQ14	Reserved
IRQ13	Reserved
IRQ12	Reserved
IRQ11	Reserved
IRQ10	External IO 3
IRQ9	External IO 2
IRQ8	External IO 1
IRQ7	Reserved
IRQ6	Reserved
IRQ5	Reserved
IRQ4	SPI 1
IRQ3	Reserved
IRQ2	UART 3
IRQ1	UART 2

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 かつ，対象のスレッドの Status Register の Interruption Mode (IE) のビットが 0 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 8.4) 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

Table 8.4: 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

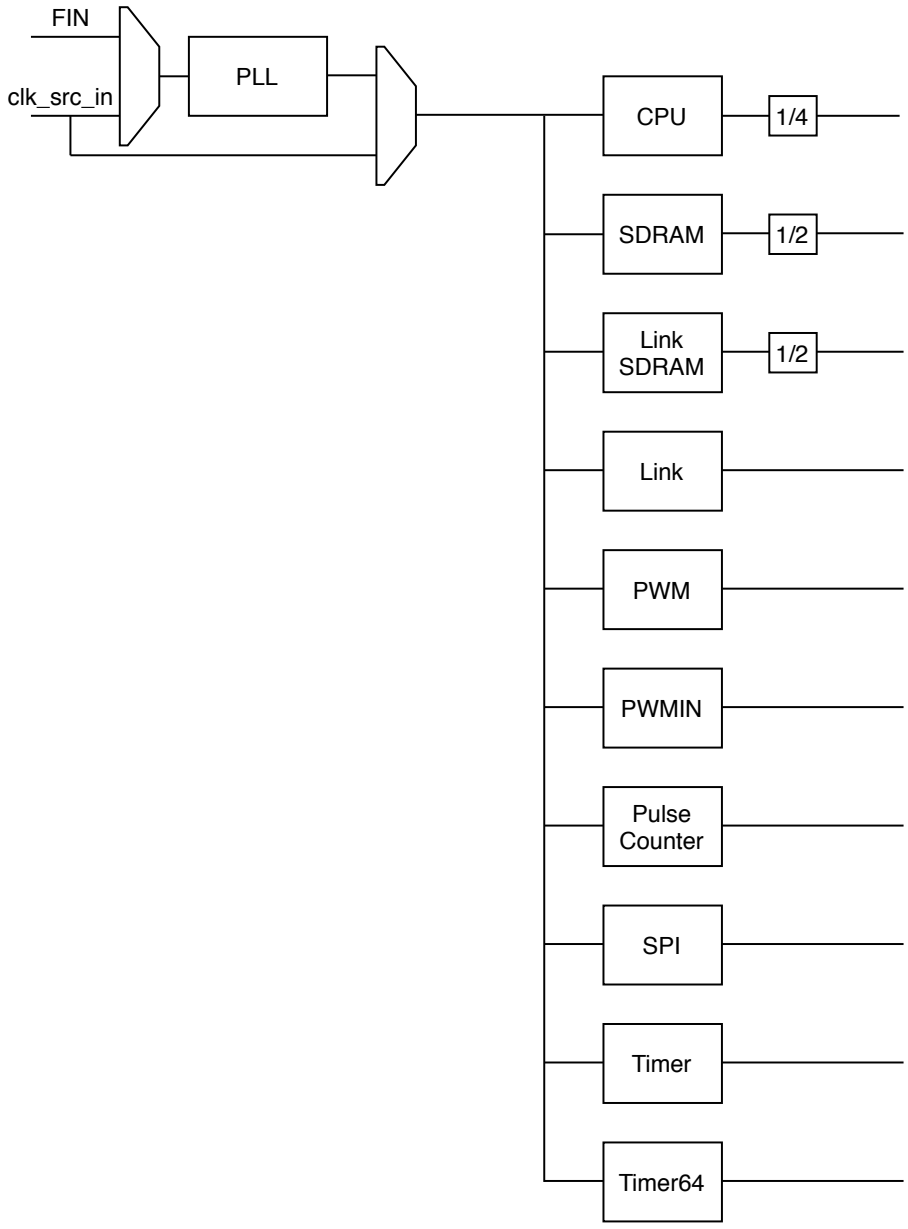


Figure 9.1: クロック生成部

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

Initial Address: 0xffffa000

9.2.1 Clock Enable

Offset: 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.

31 28	27 0
Reserve	Enable

Field Name	Function
Enable	各クロックを有効/無効にする．

9.2.2 Soft Reset

Offset: 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).

31 28	27 0
Reserve	Reset

Field Name	Function
Reset	各モジュールにリセットをかける．

9.2.3 Clock Enable / Soft Reset Bit Map

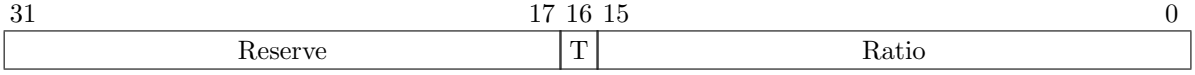
Clock Enable, Soft Reset 共にビットマップは以下の様になっている．なお，Outer とは外部ピンに対して出力するクロックである．Responsive Link の通信速度に関わるクロックが Link である．Half link は，Responsive Link のコントロールを行うクロックである．仕様上 Link の半分の値に設定するため，便宜上 Half link と定義している．

bit	module	bit	module
00	CPU	14	Vector Floating-Point
01	DMAC0	15	SDRAM
02	DMAC1	16	Pulse Counter
03	DMAC2	17	PWM
04	DMAC3	18	Outer
05	DMAC4	19	Link
06	DMAC DIAG	20	PWM Input
07	Context Cache	21	Link SDRAM
08	Complex INT	22	Reserved
09	Floating-Point Unit	23	PIO
10	SIMD	24	SPI
11	Floating-Point Reservation Station	25	Half Link
12	Synchronize	26	Timer
13	Vector Integer	27	Timer64

9.2.4 Divider Ratio

Offset: 0x0008〜0x001c, 0x0028, 0x002c, 0x0030, 0x0034, 0x0038
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
Pulse 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
SPI	1/8	0x0030
Timer	1/2	0x0034
Timer64	1/2	0x0038



Field Name	Function
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．

9.2.5 Clock Synchronization

Offset: 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．

Field Name	Function
Sync	クロックの立ち上りエッジを CPU のクロックにそろえる．

ビットマップは以下の様になっている．

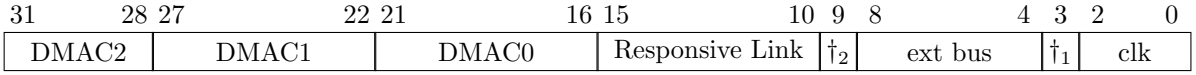
bit	module	bit	module
1	SDRAM	7	Link SDRAM
2	Pulse Counter	8	SPI
3	PWM	9	HALF Link
4	Outer	10	Timer
5	Link	11	Timer64
6	PWM Input		

9.2.6 All Reset

Offset: 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．

9.2.7 Micro Reset

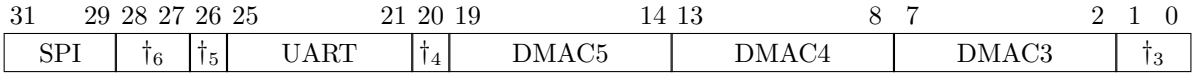
対応する bit に 0 を書き込むと, リセットがかかる．
Offset: 0x0208



†1: IRC, †2: OCE

bit	概要	bit	概要	bit	概要	bit	概要
0	全体	8	auto_ready1	16	dmac0 の ch0	24	dmac1 の ch2
1	clk	9	OCE	17	dmac0 の ch1	25	dmac1 の ch3
2	hiz	10	Responsive Link の bus I/F	18	dmac0 の ch2	26	dmac1 の arbiter
3	IRC	11	コントローラー	19	dmac0 の ch3	27	dmac1 の I/F
4	ext_bus の I/F	12	シリパラ変換器	20	dmac0 の arbiter	28	dmac2 の ch0
5	rom の I/F	13	sdram I/F	21	dmac0 の bus I/F	29	dmac2 の ch1
6	flash の I/F	14	dmacdiag の bus I/F	22	dmac1 の ch0	30	dmac2 の ch2
7	auto_ready0	15	dmacdiag	23	dmac1 の ch1	31	dmac2 の ch3

Offset: 0x020c



†3: DMAC CH2, †4: DMAC256, †5: PIO, †6: SDRAM

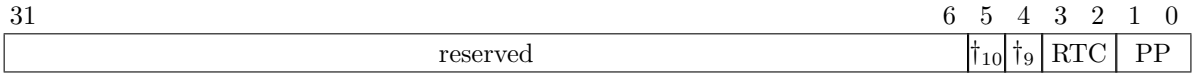
bit	概要	bit	概要	bit	概要	bit	概要
0	dmac2 の arbiter	8	dmac4 の ch0	16	reserved	24	uart の ch3
1	dmac2 の bus I/F	9	dmac4 の ch1	17	reserved	25	uart の bus I/F
2	dmac3 の ch0	10	dmac4 の ch2	18	reserved	26	pio
3	dmac3 の ch1	11	dmac4 の ch3	19	dmac diag の bus I/F	27	sdram I/F (bus)
4	dmac3 の ch2	12	dmac4 の arbiter	20	dmac256	28	sdram I/F (sdram)
5	dmac3 の ch3	13	dmac4 の bus I/F	21	uart の ch0	29	spi0
6	dmac3 の arbiter	14	dmac diag の ch0	22	uart の ch1	30	spi1
7	dmac3 の bus I/F	15	reserved	23	uart の ch2	31	spi の bus I/F

Offset: 0x0210

†7: SPI, †8: I2C

bit	概要	bit	概要	bit	概要	bit	概要
0	spi と BUS I/F の sync	8	pwm out0	16	pwm out8	24	pwm in4
1	i2c	9	pwm out1	17	pwm out9	25	pwm in5
2	pls counter0	10	pwm out2	18	pwm out10	26	ext_timer0
3	pls counter1	11	pwm out3	19	pwm out11	27	ext_timer1
4	pls counter2	12	pwm out4	20	pwm in0	28	ext_timer2
5	pls counter3	13	pwm out5	21	pwm in1	29	ext_timer3
6	pls counter4	14	pwm out6	22	pwm in2	30	timer64 の ch0
7	pls counter5	15	pwm out7	23	pwm in3	31	timer64 の ch1

Offset: 0x0214



$\dagger_9$: Trace Buffer, $\dagger_{10}$: SRAM

bit	概要
0	timer64 の ch2
1	timer64 の ch3
2	rtc の bus I/F
3	rtc
4	trace buffer
5	sram

9.2.8 HiZ Control

以下のオフセットに 0x0 を書き込むと, 対応するモジュールが HiZ となる.
オフセットは以下の通りである.

モジュール	アドレスオフセット
Responsive Link	0x0400
Link SDRAM	0x0404
SDRAM	0x0408
CLK	0x040c
EBIU	0x0410
SPI	0x0414
PWM Out	0x0418
UART	0x041c
PIO	0x0420
I2C	0x0424
On Chip Emulator	0x0428

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 スレッドの概要

RMT Processor は 8 つのハードウェアコンテキストを所有しており，それぞれのハードウェアコンテキストは専用の物理レジスタセット (GPR と FPR) を所有している．ブート時には 1 つのハードウェアスレッドが 1 つのハードウェアコンテキストで動作する．ソフトウェアはスレッド制御命令を発行することで動的にハードウェアスレッドを生成/削除することが可能であり，それらを未使用のハードウェアコンテキストで動作させることができる．これにより，*RMT Processor* 上のソフトウェアは現在動作しているハードウェアスレッドを認識し，タスクを割り当てることが可能になる．

RMT Processor ではグローバルスケジューリングにおけるコンテキストスイッチを取り除くために 8 つのハードウェアコンテキストを実装している．

10.2 スレッド起床メカニズム

RMT Processor は IRC ユニットを用いた特殊な割込み機構を所有している．割込みが発生すると，対応するスリープスレッドがハードウェアにより次のクロックサイクルで起床する．これによりイベントへの応答時間の短縮が可能であり，I/O イベントに対して適用した場合はイベントドリブンプログラミングが可能になる．リアルタイム実行では応答時間の短縮は重要である．また，この割込み機構は内部/外部タイマーにも適用でき，タイムドリブンの正確なスレッド制御が可能になる．このスレッド起床メカニズムで生成された特殊なタスクは Responsive Task と呼ばれる．Responsive Task はコンテキストスイッチのオーバーヘッドを削減し，低ジッタなリアルタイム実行を実現する．

10.3 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．

10.4　スレッド制御命令

10.4.1　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．Stack Pointer 等は設定されないため，後で作成されたスレッド自身によって設定される必要がある．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．

10.4.2　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．

10.4.3 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 を指定し，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る．復帰時の状態はシステムレジスタ Special Mode Register(0xf0) の Thread bit(1bit 目) の値によって異なる．1 の場合は停止状態で復帰するため，明示的に runth 命令により実行状態に移す必要がある．0 の場合は実行状態で復帰する．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る．復帰したスレッドの状態はシステムレジスタ Special Mode Register(0xf0) の Thread bit(1bit 目) の値によって異なる．1 の場合，復帰するスレッドは退避するアクティブスレッドの状態を踏襲する．0 の場合，復帰するスレッドは退避するアクティブスレッドの状態に関わらず実行状態となる．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．

10.5　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.

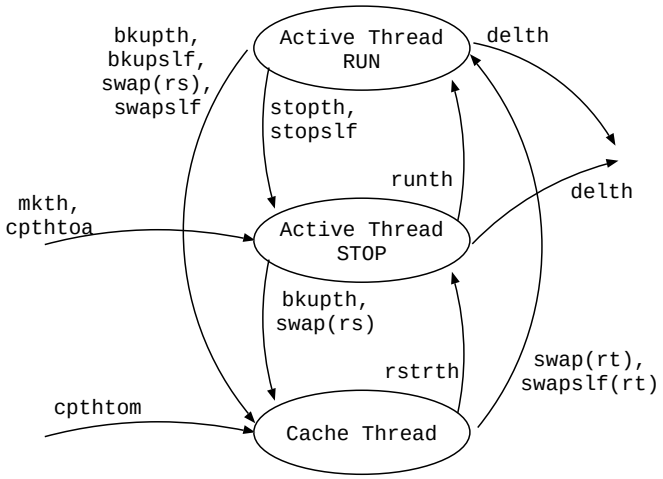


Figure 10.1: スレッドの状態遷移

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 概要

RMTP には 31 個の共有レジスタ (64bit) が存在し, これを使用することでスレッド間での排他制御, バリア同期が可能になる. また, コンテキストキャッシュに存在するスレッドとの排他制御, バリア同期も可能である. RMTP には共有レジスタとは別に, バイナリセマフォを実現するための 1024 個のバイナリセマフォレジスタが存在する. 共有レジスタが tid を保持するのに対しバイナリセマフォは tid を保持せず, ハードウェアコストを抑えてバイナリセマフォを実現することができる.

11.2 共有レジスタバイナリ

31 個の共有レジスタ (0, 1, ... , 30) と 1 つの設定レジスタ ($31) からなる. それぞれの共有レジスタには

- Full/Empty (F/E) bit (1bit)
- Exclusive / Producer-Consumer (E/P) bit (1bit)
- Barrier bit (1bit)
- Thread ID (32bit)

が割り当てられ, その共有レジスタの使用権利を持つスレッドと権利の状態が保存される.

Full/Empty bit はそのレジスタにロックがかかっているかどうかを示し, 他 2bit はかかっているロックの状態を示す.

11.3 バイナリセマフォレジスタ

1024 個のバイナリセマフォレジスタ (0, 1, ..., 1023) からなる.

11.4 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ある.

- RGPSH, RFPSH rt, ss
  Read Shared 命令. rs に GPR (FPR) を, ss に対象となる共有レジスタを指定する.
  対象の共有レジスタの F/E bit が Empty のときに成功する. 対象となる共有レジスタから GPR (FPR) に書き込む.

- WGPSH, WFPSH rt, sd
  Write Shared 命令. rs に GPR (FPR) を, sd に対象となる共有レジスタを指定する.
  対象の共有レジスタの F/E bit が Empty のときに成功する. 対象となる共有レジスタに GPR (FPR) から書き込む.

- RGPEX, RFPEX rt, ss
  Read Exclusive 命令. rt に GPR (FPR) を, ss に対象となる共有レジスタを指定する.
  対象の共有レジスタの F/E bit が Empty のときに成功する. 対象とする共有レジスタから GPR (FPR) に書き込む. 成功すると対象の共有レジスタの F/E bit を Full にし, 自分のスレッド ID を書き込む.

- WGPEX, WFPEX rt, sd
  Write Exclusive 命令. rt に GPR (FPR) を, sd に対象となる共有レジスタを指定する.
  対象の共有レジスタの F/E bit が Full かつ自分のスレッド ID が書き込まれているときに成功する. 対象となる共有レジスタに GPR (FPR) から書き込む. 成功すると対象の共有レジスタの F/E bit を Empty にする. 失敗したときには NOP として実行される.

- GPPR, FPPR rt, sd, tid
  Write Producer 命令. rt に GPR (FPR) を, sd に対象となる共有レジスタを, tid に権利を与えるスレッド ID を指定する.
  対象の共有レジスタの F/E bit が Empty のときに成功する. 対象となる共有レジスタに GPR (FPR) から書き込む. 成功すると対象の共有レジスタの F/E bit を Full にし, tid 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む.

- GPCO, FPCO rt, ss, tid
  Read Consumer 命令. rt に GPR (FPR) を, ss に対象となる共有レジスタを, tid に GPPR (FPPR) を実行したスレッド ID を指定する.
  対象の共有レジスタの F/E bit が Full かつ自分のスレッド ID が書き込まれているときに成功する. 対象とする共有レジスタから GPR (FPR) に書き込む. 成功すると対象の共有レジスタの F/E bit を Empty にする. 両方の条件を満たしていないときには NOP として実行される.

- BAR rt, sd
  Barrier 命令. rt に到着を待つスレッド数を, sd に使用する共有レジスタを指定する.
  初めて実行された場合, 対象の共有レジスタの値を Full にし, F/E bit が Full になる. 2 回目以降は対象の共有レジスタの値をインクリメントすることで指定したスレッド数と等しくなるまで, スレッドをストールする. 指定したスレッド数と等しくなると, すべてのスレッドのストールを解除し、 F/E bit を Empty にする.

- PBAR sd
  Pre Barrier 命令. sd に排他制御やバリア同期に使用する共有レジスタを指定する.
  共有レジスタ 31 に 0 以外の値を書き込み, PBAR を実行したスレッドはコンテキストキャッシュに退避されていても, BAR 命令が指定した共有レジスタを用いて行われた際に BAR を行った命令と swap されることでコンテキストキャッシュも含めた同期が取れるようになる.

共有レジスタアクセス命令の実行条件を表 11.1 に示す.

Table 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	最初に到着した命令
	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.

バイナリセマフォレジスタには次の命令を用いてアクセス可能である.

- SEMLOCK, rt, ss
  Semaphore Lock 命令. rt に GPR を, ss に対象となるバイナリセマフォレジスタを指定する. ロックの確保に成功すると 1 を返し, 失敗した場合対象のバイナリセマフォレジスタのロックが開放されるまでスレッドの実行を停止する.
- SEMREL, rt, ss
  Semaphore Release 命令. rt に GPR を, ss に対象となるバイナリセマフォレジスタを指定する. ロックの開放に成功すると 1 を返し, 失敗した場合 0 を返す.
- SEMTRY, rt, ss
  Semaphore Try 命令. rt に GPR を, ss に対象となるバイナリセマフォレジスタを指定する. ss で指定

したバイナリセマフォレジスタのロックを獲得する. ロックの獲得が成功した場合 1 を返し, 失敗した場合は 0 を返す. 失敗した場合でもスレッドは停止せず実行し続ける.

12

IPC Control Mechanism

12.1 Abstract

Stabilize the number of executed instructions of each thread per unit time (control period) by using IPC (Instructions Per Clockcycle) control. Feed Forward control or PID control can be selected as IPC control scheme. By configuring parameters properly, PID control can stabilize throughput more than Feed Forward control.

12.2 Configurable Parameters

Configurable parameters are shown below.

- Proportional gain, integral gain, derivative gain of PID control (see also section7.1.40).
- Enable/Disable IPC control (see also section7.1.41).
- The number of instructions which will be executed during control period (see also section7.1.41).

Also, to use IPC control, you should configure the register below.

- Clock number of control period (see also section7.1.6).

12.3 Usage

In this IPC control mechanism, the number of instructions configured by Target IPC Register are executed in each control period configured by Compare Register. If the value of Target IPC Register is higher than the number of actual executed instructions, IPC control will not work. If the number of actual executed instructions exceed the configured number, instruction fetch of the thread is stalled until next IPC control period begins. The limit number of instructions calculated in control period is stored in Fetch Bound Register (see also section7.1.42).

The procedure of using IPC control is shown below.

1. Start a thread.

2. Configure a cache, MMU if necessary.
3. If you use PID control, set PID gain (PID Parameter Register).
4. Enabling IPC control by setting Target IPC Register.
5. Configure a control period (unit: clocks) by setting Compare Register.
6. Start timer by setting Status Register.

In this IPC control mechanism, longer the control period, higher the stability of thread speed. Also, if the number of instructions are too many, IPC control can not work. Please take care.

12.4 Program Example

This is a program example of the procedure described in section12.3, step 3 to 5. mtc0 instructions is used to write to control registers.

```
/// Preprocessor macro examples ///
/*****
 * Status Register
 * Status Register Address: 0x00-0x07
 *****/
#define RMT_TH_STATUS(x)                (0x00+(x))
#define RMT_TH_STATUS_PE                0x00800000 // Timer Start
#define RMT_TH_STATUS_TM                0x01000000 // Period

/*****
 * Compare Register
 * Compare Register Address: 0x28-0x2f
 *****/
#define RMT_TH_PERIOD(x)                (0x28+(x))

/*****
 * Target IPC Register
 * IPC Control Register Address: 0xd0-0xd7
 *****/
#define RMT_TH_TARGET_IPC(x)    (0xd0+(x)) // xth thread's target IPC
#define RMT_IPC_PID_MODE        0x80000000 // Enable IPC control by PID
#define RMT_IPC_FF_MODE         0xA0000000 // Enable IPC control by Feed Forward

/*****
 * PID Parameter Register
 * Configurations of 2 threads are on single register.
 *****/
#define RMT_PID_PARAM01         0xcc       // PID Param Register (Thread 0,1)
#define RMT_PID_PARAM23         0xcd       // PID Param Register (Thread 2,3)
#define RMT_PID_PARAM45         0xce       // PID Param Register (Thread 4,5)
```

```
#define RMT_PID_PARAM67         0xcf       // PID Param Register (Thread 6,7)

#define RMT_PID_ENABLE_ODD      0x80000000 // Setting PID gain of odd thread
#define RMT_PID_ENABLE_EVEN     0x00008000 // Setting PID gain of even thread
#define RMT_PID_K_DERI_EVEN(x)  (x<<0)     // Kd of even thread
#define RMT_PID_K_INTE_EVEN(x)  (x<<5)     // Ki of even thread
#define RMT_PID_K_PROP_EVEN(x)  (x<<10)    // Kp of even thread
#define RMT_PID_K_DERI_ODD(x)   (x<<16)    // Kd of odd thread
#define RMT_PID_K_INTE_ODD(x)   (x<<21)    // Ki of odd thread
#define RMT_PID_K_PROP_ODD(x)   (x<<26)    // Kp of odd thread

/*****
 * Configurations of IPC control period, instruction numbers.
 *****/
#define PERIOD_TH1 10000 // Set control period of thread 1 to 10,000
#define PERIOD_TH2 20000 // Set control period of thread 2 to 20,000

#define TARGET_IPC_TH1 4000 // Set target IPC of thread 1 to 0.4 (4,000/10,000)
#define TARGET_IPC_TH2 6000 // Set target IPC of thread 2 to 0.3 (6,000/20,000)

/******
 * Example: IPC control on thread 1 using PID
 * Target IPC: 0.4, IPC control period: 10,000 clockcycles
 * Kp: 0.5, Ki: 0.125, Kd: 0.25
 ******/
// Set PID parameters of thread 1
mtc0(( RMT_PID_ENABLE_ODD | RMT_PID_K_PROP_ODD(0x01)|
       RMT_PID_K_INTE_ODD(0x04)|RMT_PID_K_DERI_ODD(0x02) ),
       RMT_PID_PARAM01 );
// Enable IPC control
mtc0(( RMT_IPC_PID_MODE + TARGET_IPC_TH1 ), RMT_TH_TARGET_IPC(1) );
// Set control period of thread 1
mtc0( PERIOD_TH1, RMT_TH_PERIOD(1) );
// Start a timer
mtc0( ( RMT_TH_STATUS_PE | RMT_TH_STATUS_TM ), RMT_TH_STATUS(1) );

/******
 * Example: IPC control on thread 2 using FF
 * Target IPC: 0.3, IPC control period: 20,000 clockcycles
 ******/
// Enable IPC control
mtc0(( RMT_FF_PID_MODE + TARGET_IPC_TH2 ), RMT_TH_TARGET_IPC(2) );
// Set control period of thread 2
mtc0( PERIOD_TH2, RMT_TH_PERIOD(2) );
// Start a timer
mtc0( ( RMT_TH_STATUS_PE | RMT_TH_STATUS_TM ), RMT_TH_STATUS(2) );
```

13

Vector Unit

13.1 概要

RMT Processor の Vector Unit のブロック図を図 13.1 に示す．Vector Integer Unit，Vector Floating Point Unit 共に大きく 3 つの部分から成る．

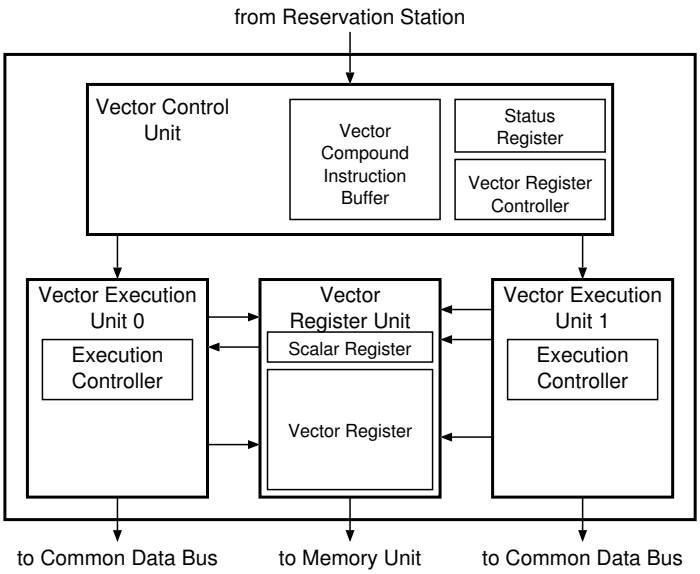


Figure 13.1: Vector Unit のブロック図

- 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．

13.1.1 Vector Execution Unit

Vector Execution Unit のブロック図を図 13.2 に示す．

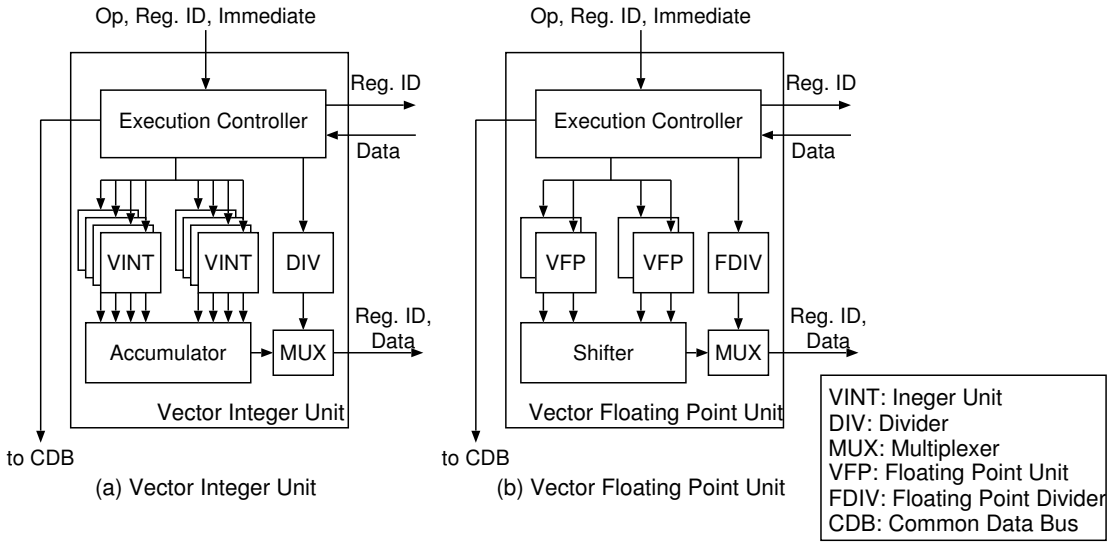


Figure 13.2: Vector Execution Unit のブロック図

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．

13.1.2 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 13.3 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．

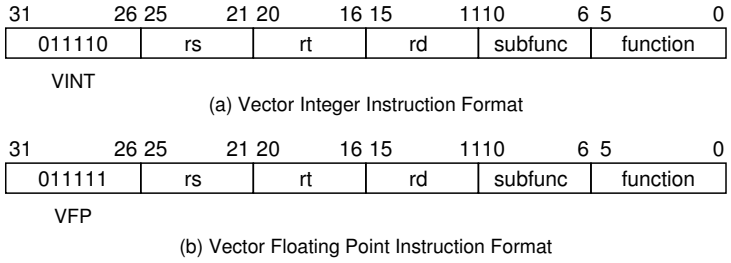


Figure 13.3: ベクトル演算命令フォーマット

13.2 ベクトルレジスタ

RMTP では 512 エントリのベクトルレジスタを複数スレッドで共有している．各スレッドは Reserve 命令を用いることで必要に応じて 128 エントリ，256 エントリ，512 エントリ，のいずれかの量を確保して使用する．確保する際には図 13.4 のように，ベクトルレジスタを 4 つの領域に分け，確保したエントリに応じた領域が割り当てられる．スカラレジスタも 32 エントリを共有して使用し，確保したベクトルレジスタに応じたスカラレジスタが割り当てられる．使用した後は Release 命令を用いてレジスタを解放することで，他のスレッドでも使用できるようになる．

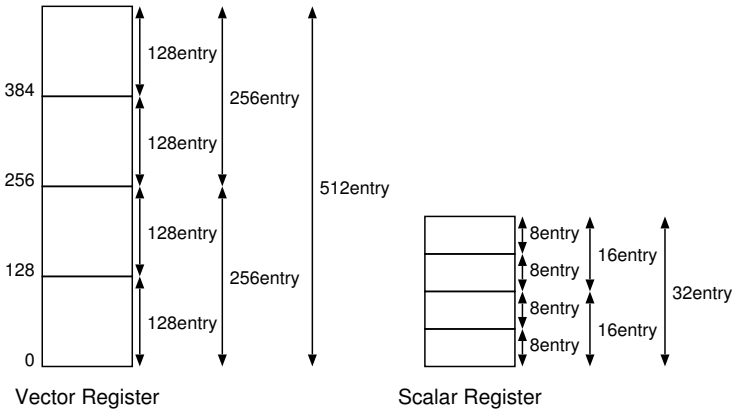


Figure 13.4: ベクトルレジスタのサイズ

また，確保したベクトルレジスタはあらかじめ設定されたベクトル長に応じて分割される．ベクトル長は 8, 16, 32, 64 の中から選択する．このベクトル長は確保したエントリ数によって選択出来る値が変わる．RMTP で選択出来るレジスタの構成は図 13.5 のとおりである．

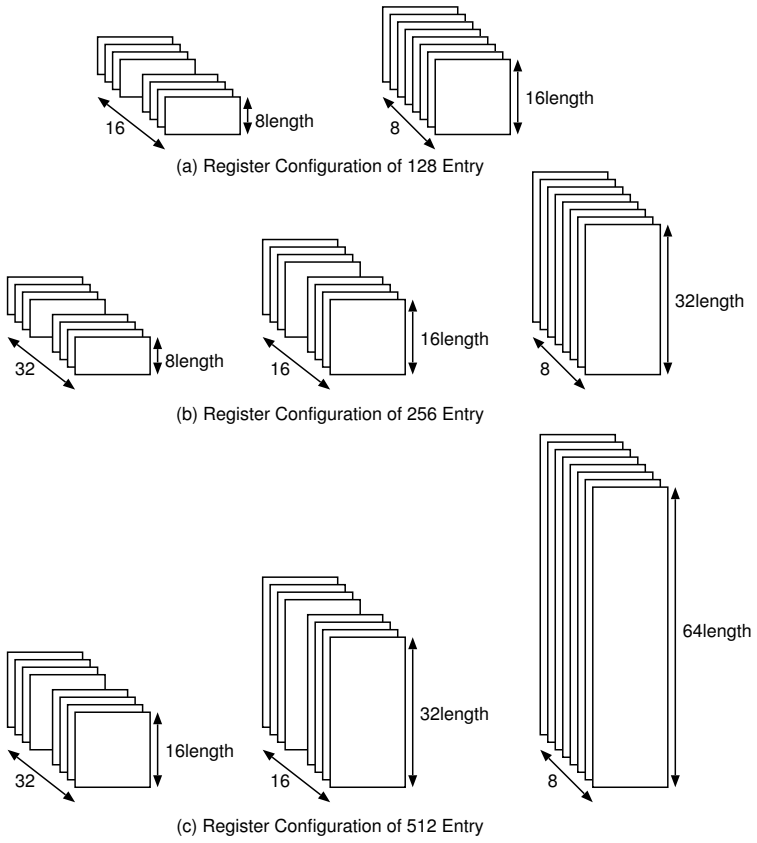


Figure 13.5: ベクトルレジスタの構成

- (a)128 エントリ
  - ベクトル長 8 × レジスタ 32 個
  - ベクトル長 16× レジスタ 16 個
- (b)256 エントリ
  - ベクトル長 8 × レジスタ 32 個
  - ベクトル長 16× レジスタ 16 個
  - ベクトル長 32× レジスタ 8 個
- (c)512 エントリ
  - ベクトル長 16× レジスタ 32 個
  - ベクトル長 32× レジスタ 16 個
  - ベクトル長 64× レジスタ 8 個

この構成はベクトルレジスタを確保する際にモードを指定することで設定する．モードは表 13.1 の中から選択する．

Table 13.1: Vector Register Mode の指定

(128 Entry)	
8 length × 16	0x4
16 length × 8	0x5
(256 Entry)	
8 length × 32	0x8
16 length × 16	0x9
32 length × 8	0xA
(512 Entry)	
16 length × 32	0xD
32 length × 16	0xE
64 length × 8	0xF

13.3 ステータスレジスタ

RMTP では，各スレッドごとに表 13.2 のようなステータスレジスタを持つ．Status Register はベクトル演算を行うために必要な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．

Table 13.2: Vector Status Register

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 +∞ , 3: Round to -∞)
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．

13.4 使用例

図 13.6 に以下の条件での実際の演算のプログラム例を示す.

- ベクトル長 32×8 個のモードを使用
- $8，$9，にデータの先頭アドレスを格納済み
- $10 に計算結果を格納する先頭のアドレスを格納済み
- vector float を使用
- stride は使用しない

```
addiu  $12, $0,  0x000A  # 256 Entry ( 32 length x 8 ) Mode
lui    $11, $0,  0x0001
addiu  $11, $11, 0x1111  # length (32 - 1)
virsv  $13, $12          # Reserve Instruction
mtc1   $11, $f0          # store length to FPR0
vfmtc  $2,  $f0          # set length
mtc1   $0,  $f0          # store stride(0) to FPR0
vfmtc  $3,  $f0          # set stride
vfld   $0,  $8           # load from $8
vfld   $1,  $9           # load from $9
add.v  $2,  $0,  $1      # add vreg0, vreg1

  == Vector Execution ==

vfsd      $2,  $10       # store to $10
virls     $10            # Release Instruction
```

Figure 13.6: ベクトル演算のプログラム例

13.5 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 13.7 に Compound Instruction Buffer のフォーマットを示す.

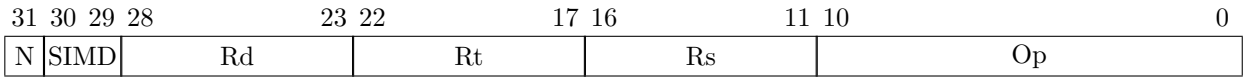


Figure 13.7: Compound Instruction Buffer のフォーマット

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．

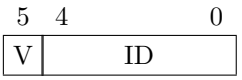


Figure 13.8: レジスタの指定

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 13.5 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．

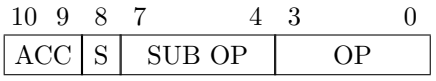


Figure 13.9: Operation のフォーマット (整数演算)

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: $\neq$，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．

Figure 13.10: 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 13.11 に示す．

複合演算定義命令では rs に図 13.7 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 13.12 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．

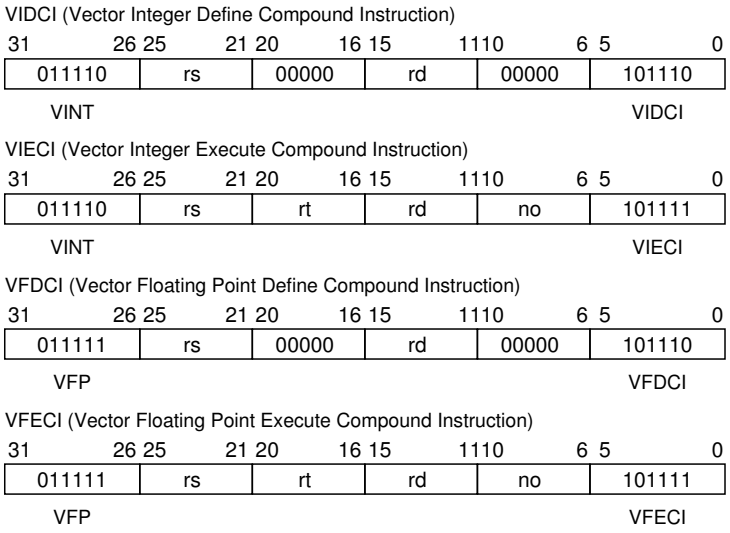


Figure 13.11: 複合演算命令のフォーマット

	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

Figure 13.12: 複合演算の定義例

14

Responsive Link

14.1 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．

14.2 *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 14.1 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．

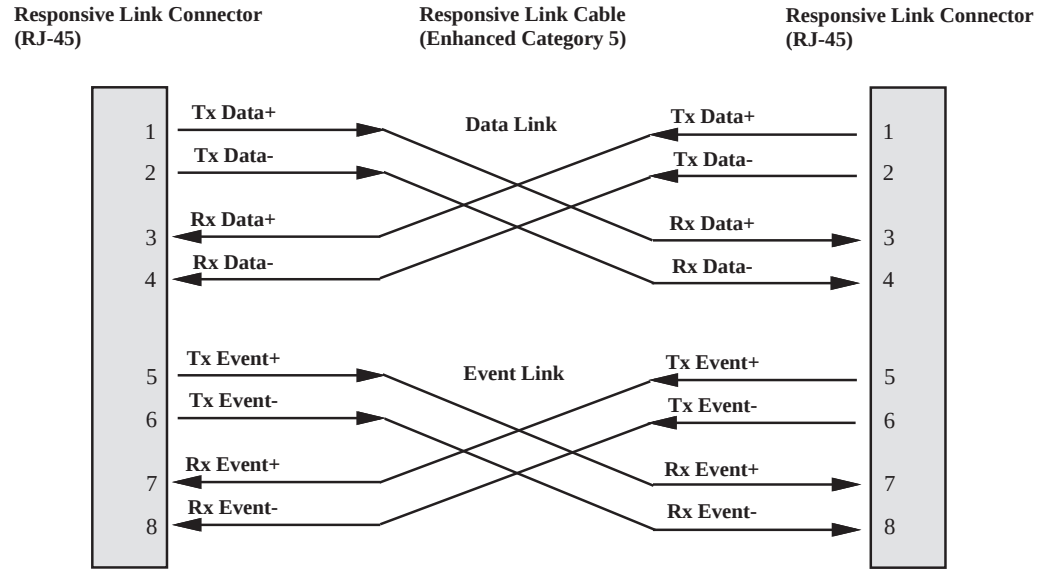


Figure 14.1: *Responsive Link* インタフェース

14.3 パケットフォーマット

図 14.2 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．

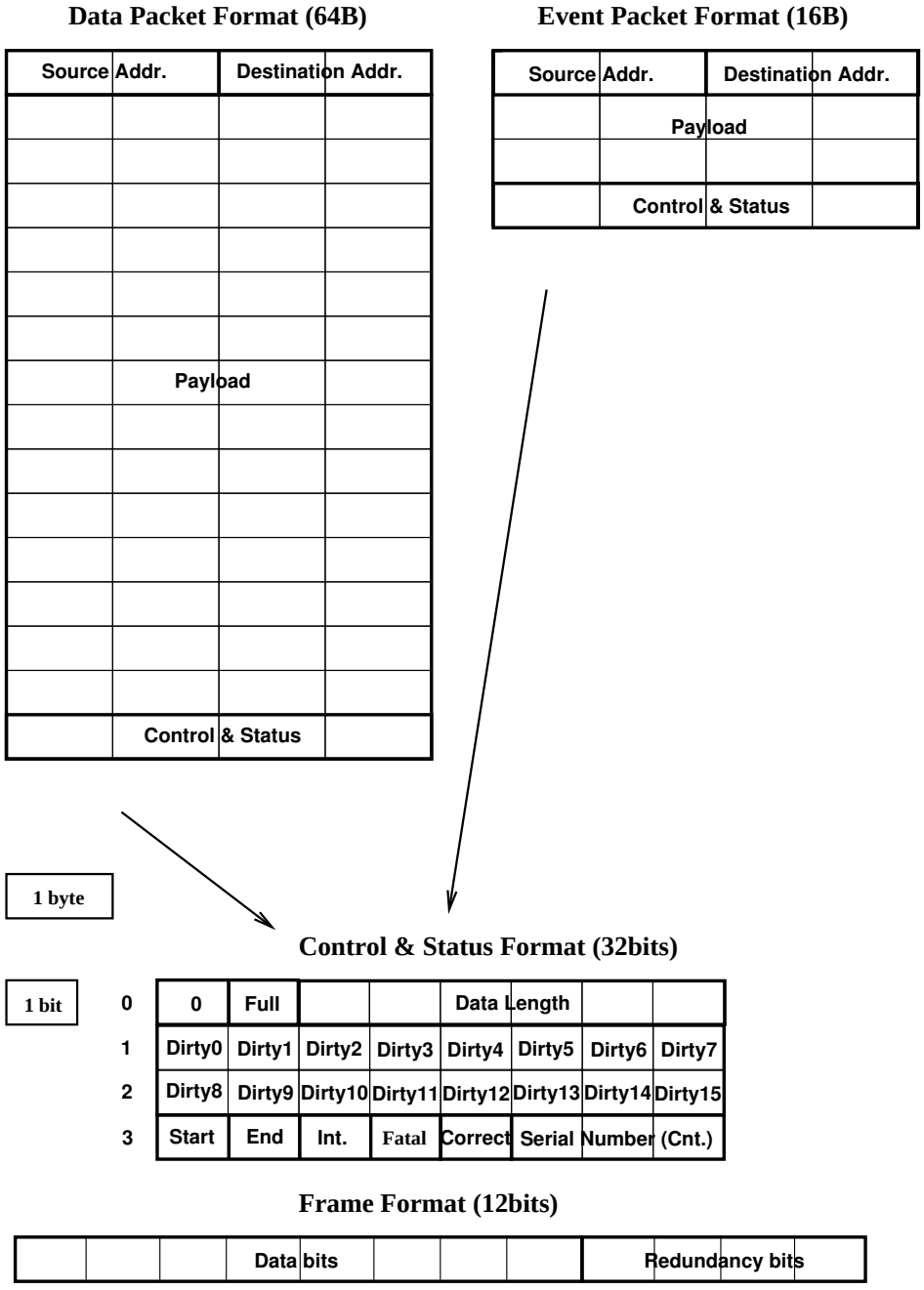


Figure 14.2: *Responsive Link* のパケットフォーマット

図 14.2 の通信パケットのヘッダ部に対して，図 14.3 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．

Figure 14.3: *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 14.3 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．

14.3.1 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．

14.3.2 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．

14.3.3 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 14.4 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 14.4 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 14.4 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 14.5 は図 14.4 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 14.5 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 14.6 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 14.5 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．

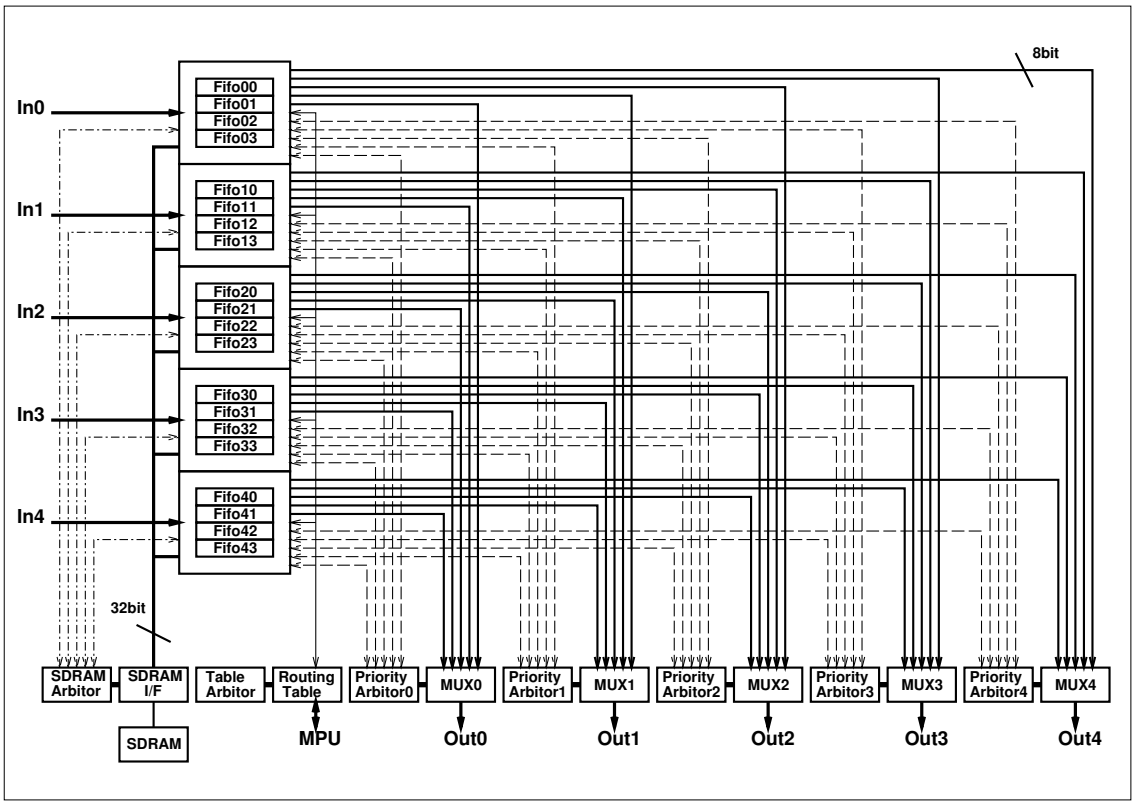


Figure 14.4: *Responsive Link* のネットワークスイッチ

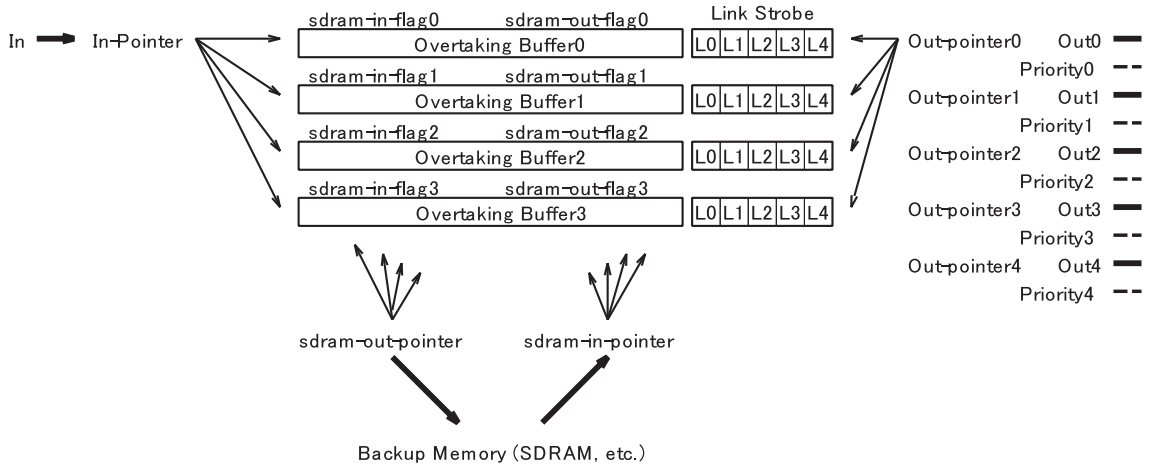


Figure 14.5: *Responsive Link* の追い越し用バッファ

図 14.5 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 14.4 の Priority ArbitorN) に対して優先度と共に出力要求を行なう．図 14.5 の PriorityN は図 14.4 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 14.5 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．

14.4 フレームフォーマット

1byte は，図 14.2 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

14.5 ルーティング・テーブル

Responsive Link の経路制御は，図 14.6 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 14.6 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．

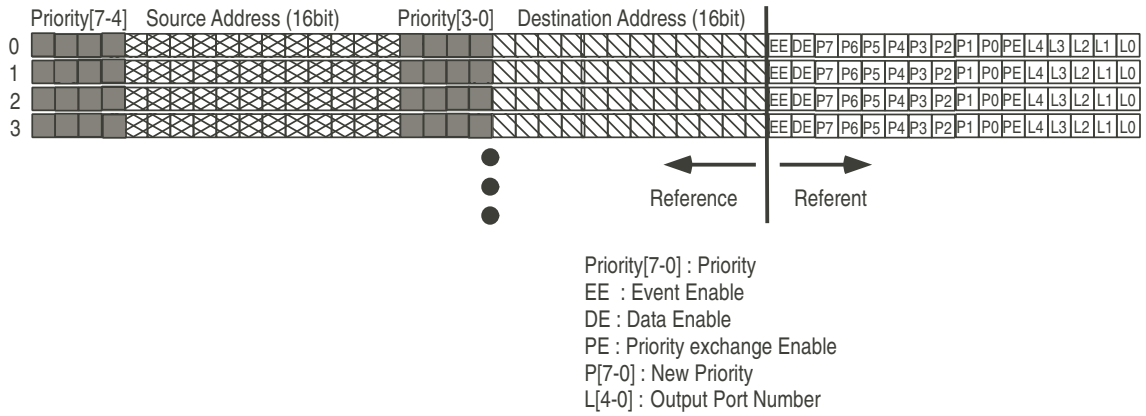


Figure 14.6: *Responsive Link* のルーティングテーブル

14.6 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 14.6 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 14.6 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 14.6 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．

14.7 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 14.7 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．

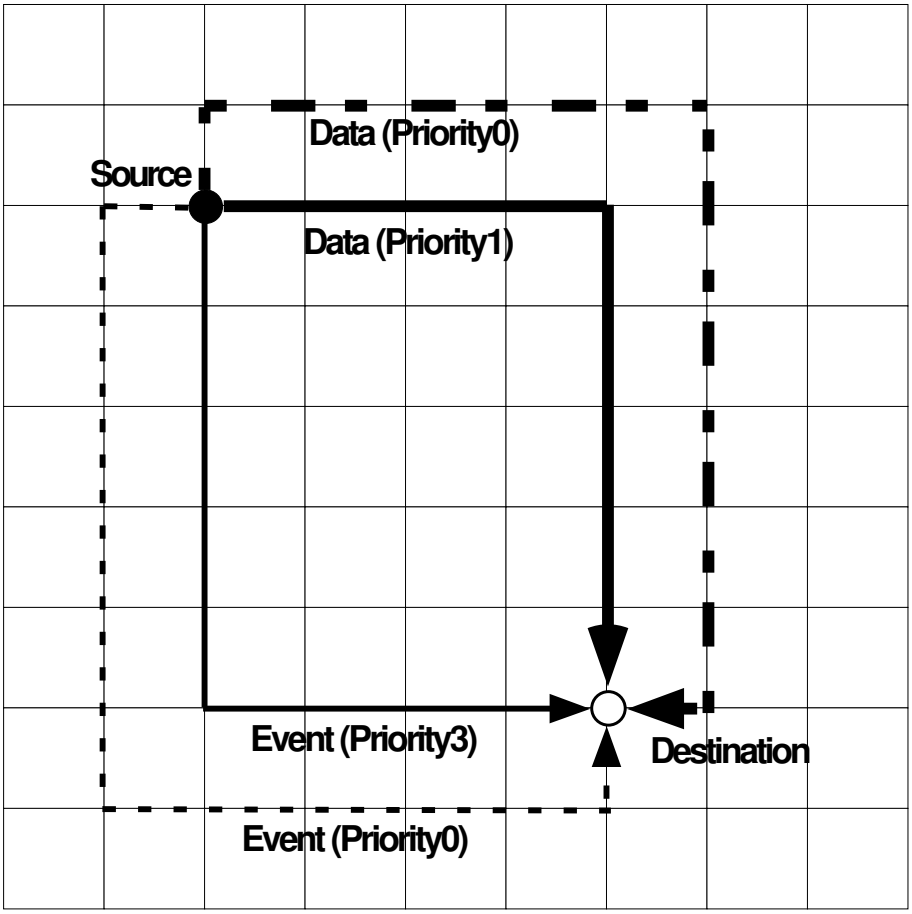


Figure 14.7: *Responsive Link* の優先度付経路

制御用の分散システムでは図 14.8 のような木構造を採る場合が多い．図 14.8 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．

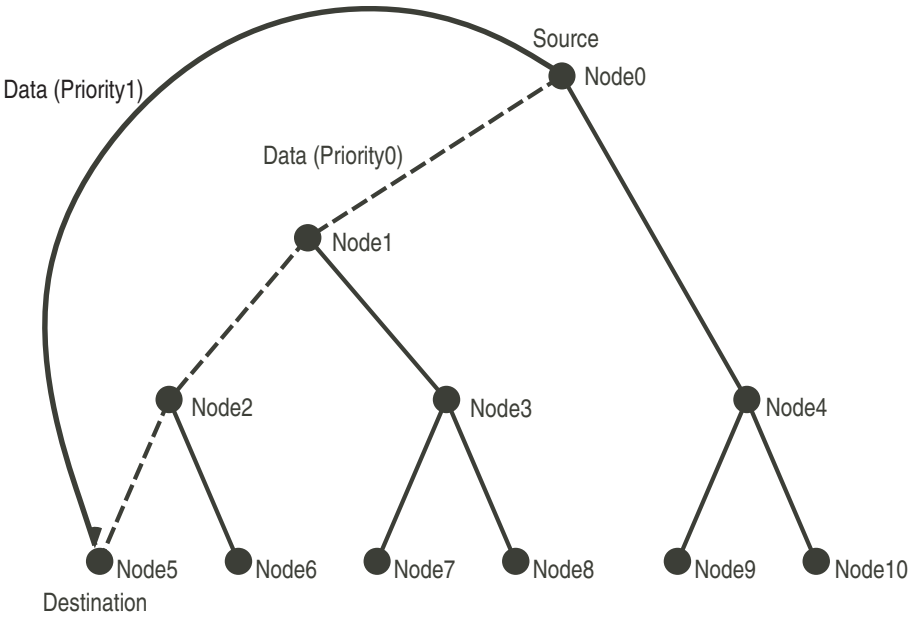


Figure 14.8: *Responsive Link* の優先度付木構造経路

14.8 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 14.2 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．

14.8.1 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．

14.8.2 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 14.1 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．

Table 14.1: 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

14.8.3 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．

14.8.4 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．

14.8.5 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．

Table 14.3: 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

14.8.6 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 14.2 に DPLL のモードを示す．なお，Mode 1 は EXT_RL_CLK が有効な場合のみ設定可能である．

モード名	p_mode2	p_mode1	p_mode0	d_clk 周期/1bit 転送
Mode1	1	1	0	1
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

Table 14.2: DPLL モードの設定

14.8.7 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 14.2 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．

14.8.8 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 14.3 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．

14.9 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．

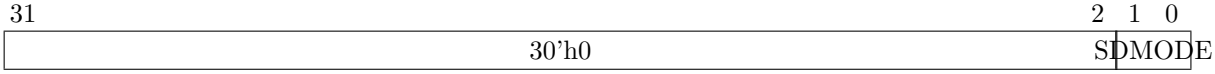
デコードアドレス	接続される I/O
0xfffe_0xxx	レスポンシブリンク内部レジスタ
0xfffe_1xxx	レスポンシブリンク用 IRC (r/w)
0xfffe_2xxx	ルーティングテーブルアドレス部 (r/w)
0xfffe_3xxx	ルーティングテーブルリンク部 (r/w)
0xC0xx_xxxx	イベント入力用 DPM (r)
0xC4xx_xxxx	イベント出力用 DPM (r/w)
0xC8xx_xxxx	データ入力用 DPM (r)
0xCCxx_xxxx	データ出力用 DPM (r/w)

Initial Address: 0xfffe0000

14.10 レジスタマップ

14.10.1 SDRAM モードレジスタ

Offset: 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．

Field Name	Function
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

14.10.2 レスポンシブリンク速度設定レジスタ

Offset: 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.
110 : Mode1
111 : Mode2
000 : Mode4
001 : Mode8
010 : Mode16
011 : Mode32

Field Name	Function
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

14.10.3 レスポンシブリンク初期化レジスタ

Offset: 0xfffe_0008
属性 リード／ライト

31 29	28 25	24	23 21	20 17	16	15 13	12 9	8	7 5	4 1	0
-	EDINIT	EMI	-	EEINIT	E_s	-	DDINIT	DMI	-	DEINIT	D_s

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，デュアルポートメモリの初期化を行なう.
0: 通常動作
1: 初期化

Field Name	Function
EDINIT	Event Link のデコーダの初期化 EDINIT[4]: RLINIT[28]: Event link4 の初期化 EDINIT[3]: RLINIT[27]: Event link3 の初期化 EDINIT[2]: RLINIT[26]: Event link2 の初期化 EDINIT[1]: RLINIT[25]: Event link1 の初期化
EMI	Event Link 用のデュアルポートメモリコントローラの初期化（メモリの内容は保持される）
ELINIT	Event link の各エンコーダの初期化 EEINIT[4]: RLINIT[20]: Event link4 の初期化 EEINIT[3]: RLINIT[19]: Event link3 の初期化 EEINIT[2]: RLINIT[18]: Event link2 の初期化 EEINIT[1]: RLINIT[17]: Event link1 の初期化
E_s	Event link switch の初期化
DDINIT	Data link の各デコーダの初期化 DDINIT[4]: RLINIT[12]: Data link4 の初期化 DDINIT[3]: RLINIT[11]: Data link3 の初期化 DDINIT[2]: RLINIT[10]: Data link2 の初期化 DDINIT[1]: RLINIT[9]: Data link1 の初期化
DMI	Data Link 用のデュアルポートメモリコントローラの初期化（メモリの内容は保持される）
DEINIT	Data link の各エンコーダの初期化 DEINIT[4]: RLINIT[4]: Data link4 の初期化 DEINIT[3]: RLINIT[3]: Data link3 の初期化 DEINIT[2]: RLINIT[2]: Data link2 の初期化 DEINIT[1]: RLINIT[1]: Data link1 の初期化
D_s	Data link switch の初期化

14.10.4 レスポンシブリンク割り込みクリアレジスタ

Offset: 0xfffe_000c 属性 ライト

本レジスタのオフセットはデコーダリセット割り込みクリアレジスタのオフセットと同じである．*Responsive Link* の IRQ1-6 のいずれかがイネーブルになったときのみ，本レジスタに書き込み可能となる．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

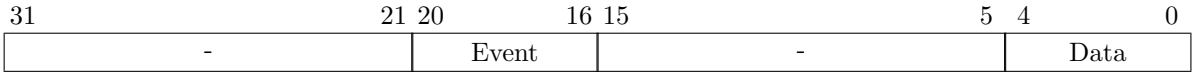
0: 通常動作

1: クリア

Field Name	Function
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

14.10.5 デコーダリセット割り込みクリアレジスタ

Offset: 0xfffe_000c 属性 リード／ライト



本レジスタのオフセットはレスポンシブリンク割り込みクリアレジスタのオフセットと同じである．*Responsive Link* の IRQ1-6 のすべてがディスエーブルになったときのみ，本レジスタに読み書き可能となる．デコーダリセット割り込み要求のクリアを行なう．

Default 0
0: クリア
1: 割り込み発生（デバッグ用）

Field Name	Function
Event	Event Link のデコーダリセット割り込み Event[4]: Event link4 のデコーダリセット割り込み Event[3]: Event link3 のデコーダリセット割り込み Event[2]: Event link2 のデコーダリセット割り込み Event[1]: Event link1 のデコーダリセット割り込み Event[0]: Event link0 のデコーダリセット割り込み
Data	Data Link のデコーダリセット割り込み Data[4]: Data link4 のデコーダリセット割り込み Data[3]: Data link3 のデコーダリセット割り込み Data[2]: Data link2 のデコーダリセット割り込み Data[1]: Data link1 のデコーダリセット割り込み Data[0]: Data link0 のデコーダリセット割り込み

14.10.6 レスポンシブリンク送信停止割り込みクリアレジスタ

Offset: 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

Field Name	Function
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)

14.10.7 レスポンシブリンク継続割り込みクリアレジスタ

Offset: 0xfffe_0014 属性 リード／ライト

31 — 21	20 — 16	15 — 5	4 — 0
-	DCIC	-	ECIC

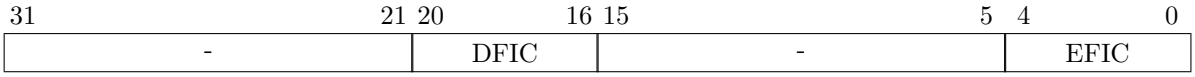
Responsive Link は，SDRAM に退避されたパケットがスイッチに書き戻された（再度送信された）際にレスポンシブリンク継続割り込み CI(Coutinuous Irq) を発生する．CIC(Continuous Irq Clear) レジスタはその割り込み要求 CI のクリアを行なう．

Default 0
0: 通常動作
1: クリア

Field Name	Function
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)

14.10.8 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

Offset: 0xfffe_0018 属性 リード／ライト



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

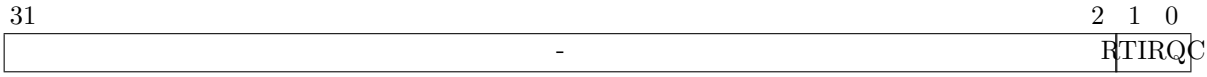
0: 通常動作

1: クリア

Field Name	Function
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)

14.10.9 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

Offset: 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)

Field Name	Function
RTIC[0]	Event Routing Table IRQ Clear
RTIC[1]	Data Routing Table IRQ Clear

14.10.10 レスポンシブリンク SDRAM バスリクエストレジスタ

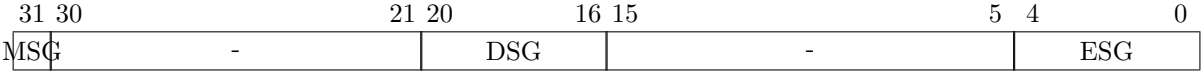
Offset: 0xfffe_0020 属性 リード／ライト

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．

Field Name	Function
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

14.10.11 レスポンシブリンク SDRAM バスグラントレジスタ

Offset: 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．

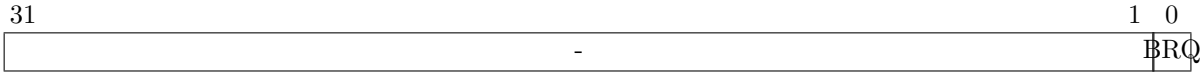
0: バス権獲得

1: バス権開放

Field Name	Function
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)

14.10.12 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

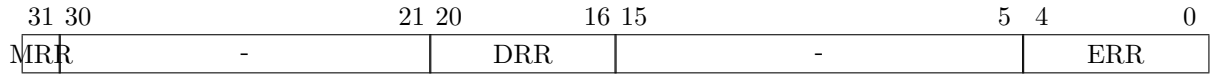
Offset: 0xfffe_0028 属性 ライト



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．

Field Name	Function
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

Offset: 0xfffe_0028 属性 リード

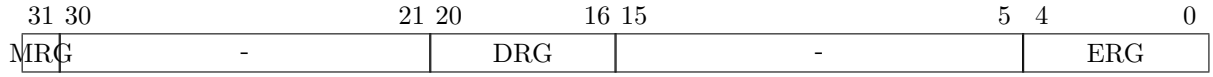


本ビットはプロセッサバス側からレスポンシブリンクのルーティングテーブルバスへのバスリクエストを示す.
0: バスリクエスト有
1: バスリクエスト無

Field Name	Function
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)

14.10.13 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

Offset: 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

Field Name	Function
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)

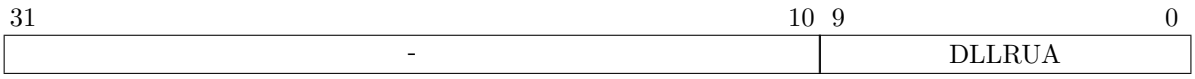
14.10.14 イベントリンク LRU アドレスレジスタ

Offset: 0xfffe_0030 属性 リード

Field Name	Function
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.

14.10.15 データリンク LRU アドレスレジスタ

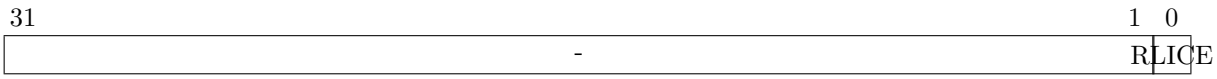
Offset: 0xfffe_0034 属性 リード



Field Name	Function
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.

14.10.16 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

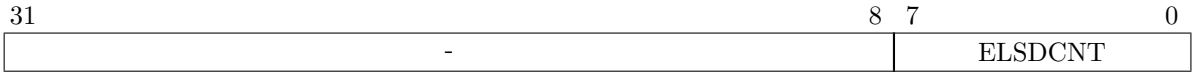
Offset: 0xfffe_0038 属性 リード



Field Name	Function
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．

14.10.17 イベントリンク用 SDRAM ループカウントレジスタ

Offset: 0xfffe_0040 属性 リード／ライト

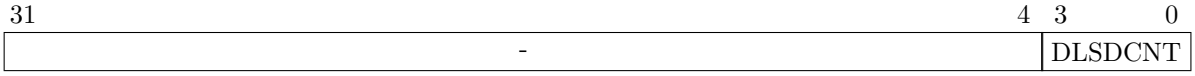


追い越し用 SDRAM に退避されたイベントパケットを *Responsive Link* イベントスイッチに再度送信してよいかどうかを調べる間隔を指定する．短すぎると消費電力が大きくなり，長すぎるとリアルタイム性が損なわれる．

Field Name	Function
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

14.10.18 データリンク用 SDRAM ループカウントレジスタ

Offset: 0xfffe_0044 属性 リード／ライト

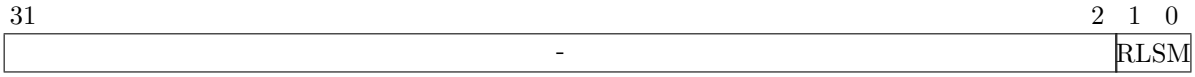


追い越し用 SDRAM に退避されたデータパケットを *Responsive Link* データスイッチに再度送信してよいかどうかを調べる間隔を指定する．短すぎると消費電力が大きくなり，長すぎるとリアルタイム性が損なわれる．

Field Name	Function
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

14.10.19 レスポンシブリンクスイッチモードレジスタ

Offset: 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

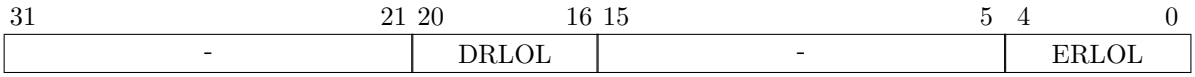
1: Store and Forward Mode レイテンシ的に不利であるがパケットの追越しをしやすい

Default: 0

Field Name	Function
RLSM[0]	Event Link Switch の設定
RSLM[1]	Data Link Switch の設定

14.10.20 レスポンシブリンク用オフラインレジスタ

Offset: 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

Field Name	Function
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)

Offset: 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

Field Name	Function
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

14.10.21 パラレルモードレジスタ

Offset: 0xfffe_0050 属性 リード／ライト

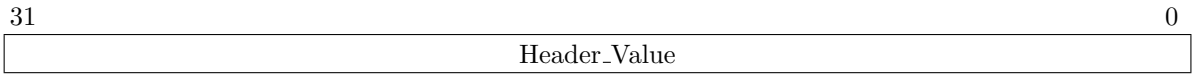
31	5	4	1	0
-		Port		-

Responsive Link の各ポートがパラレル／シリアルモードのどちらで動作するかを設定する．
1: パラレル
0: シリアル

Field Name	Function
Port	*Responsive Link* の各ポート Port[3]: ポート 4 Port[2]: ポート 3 Port[1]: ポート 2 Port[0]: ポート 1

14.10.22 エラーパケットヘッダレジスタ

Offset: Event Link : 0xfffe_0054 属性 リード
Offset: Data Link : 0xfffe_0058 属性 リード



Routing Table に該当しないパケットのヘッダの値が入る.

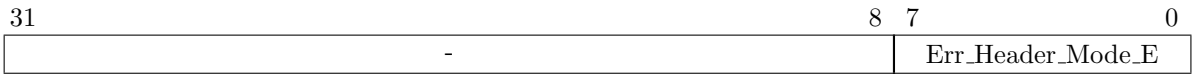
14.10.23 エラーヘッダポインタレジスタ

Offset: Event Link : 0xfffe_005c 属性 リード

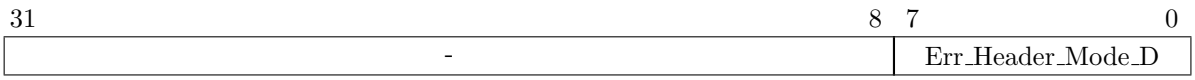
エラーパケットの数を格納する．ただし，直前のエラーパケット同一のヘッダである場合はインクリメントされない（デバッグ用）.
Default 0

14.10.24 エラーパケットモードレジスタ

Offset: Event Link : 0xfffe_0060 属性 リード／ライト



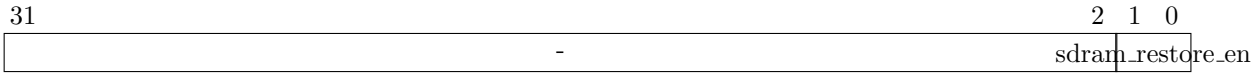
Offset: Data Link : 0xfffe_0064 属性 リード／ライト



本レジスタはエラーヘッダポインタレジスタのために存在する（デバッグ用）.

14.10.25 SDRAM 回復イネーブルレジスタ

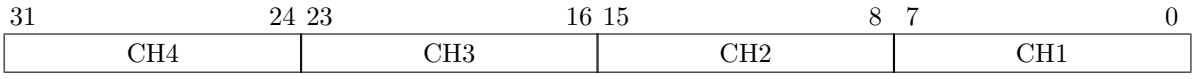
Offset: Event Link : 0xfffe_0068 属性 リード／ライト



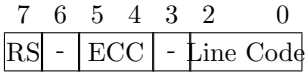
（デバッグ用）

14.10.26 通信コーデック設定レジスタ

Offset: Event Link : 0xfffe_006c 属性 リード／ライト
Offset: Data Link : 0xfffe_0070 属性 リード／ライト



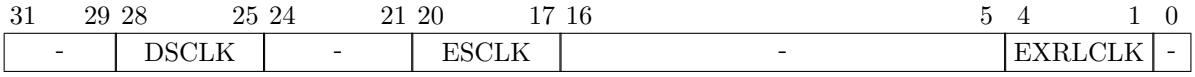
Responsive Link 各チャネルのコーデックを設定する．チャネル毎の設定項目のビットマップを以下に示す．



Field Name	Function
CH*[7]	本ビットをセットすると Reed Solomon 符号によるエラー訂正が有効になる．
CH*[5:4]	バイト毎の ECC を設定する． 00: ECC なし 01: Hamming 符号 10: BCH 符号
CH*[2:0]	伝送路符号を設定する． 001: NRZI+BitStuffing 010: 8B10B 100: 4B10B

14.10.27 レスポンシブリンク用 EXT_RL_CLK イネーブルレジスタ

Offset: 0xfffe_0074 属性 リード／ライト



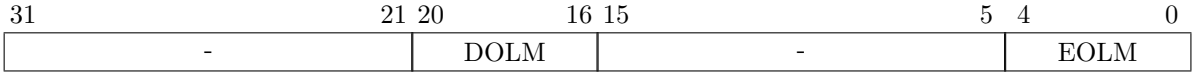
Responsive Link の EXT_RL_CLK の設定を行う．本レジスタは EXT_RL_CLK を使用可能な設定の時にマッピングされる．
さらに，DSCLK, ESCLK は SHARED_D_CLK が有効な時に設定可能となり，各通信リンクに d_clk を流し，もう一方の通信リンクではその d_clk を用いてデコードを行うことがチャネルごとに可能となる．

例えば ESCLK[4] を有効にすると Event link4 に d_clk が流れ，チャネル 4 の接続先ノードで Data link のデコードを行う際に Event link から流れてきた d_clk を用いてデコードを行う．
SHARED_D_CLK を行う際には同時に EXRLCLK の対応するビットも有効にする必要がある．
0: EXT_RL_CLK を無効にする
1: EXT_RL_CLK を有効にする

Field Name	Function
DSCLK	Data Link を SHARED_D_CLK として利用する DSCLK[4]: Data link4 DSCLK[3]: Data link3 DSCLK[2]: Data link2 DSCLK[1]: Data link1
ESCLK	Event Link を SHARED_D_CLK として利用する ESCLK[4]: Event link4 ESCLK[3]: Event link3 ESCLK[2]: Event link2 ESCLK[1]: Event link1
EXRLCLK	EXT_RL_CLK を有効にする EXRLCLK[4]: ポート 4 の EXT_RL_CLK の設定 EXRLCLK[3]: ポート 3 の EXT_RL_CLK の設定 EXRLCLK[2]: ポート 2 の EXT_RL_CLK の設定 EXRLCLK[1]: ポート 1 の EXT_RL_CLK の設定

14.10.28 レスポンシブリンク用オフライン割り込みマスクレジスタ

Offset: 0xfffe_0080 属性 リード／ライト



Responsive Link のオンライン割り込み及びオフライン割り込みのマスクを通信リンク，チャネルごとに行う．
0: 割り込みマスクを無効にする
1: 割り込みマスクを有効にする

Field Name	Function
DOLM	Data Link のオンライン/オフライン割り込みマスク設定 DOLM[4]: Data link4 DOLM[3]: Data link3 DOLM[2]: Data link2 DOLM[1]: Data link1 DOLM[0]: Data link0(CPU)
EOLM	Event Link のオンライン/オフライン割り込みマスク設定 EOLM[4]: Event link4 EOLM[3]: Event link3 EOLM[2]: Event link2 EOLM[1]: Event link1 EOLM[0]: Event link0(CPU)

14.10.29 送信用 DPLL モード設定レジスタ

Offset: 0xfffe_0084 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
-	DTXM4	DTXM3	DTXM2	DTXM1	-	ETXM4	ETXM3	ETXM2	ETXM1

Responsive Link の送信用の通信速度設定をチャネル毎，通信リンク毎に行う．
本レジスタの設定は，送信用通信速度・コーデックイネーブルレジスタで有効にしたチャネル・通信リンクにのみ適用され，受信用の DPLL モードは従来のレジスタで設定したものが用いられる．

110 : Mode1
111 : Mode2
000 : Mode4
001 : Mode8
010 : Mode16
011 : Mode32

Field Name	Function
DTXM[1-4]	Data Link の送信用 DPLL の設定 DTXM4: Data link4 用 RSL DTXM3: Data link3 用 RSL DTXM2: Data link2 用 RSL DTXM1: Data link1 用 RSL
ETXM[1-4]	Event Link の送信用 DPLL の設定 ETXM4: Event link4 用 RSL ETXM3: Event link3 用 RSL ETXM2: Event link2 用 RSL ETXM1: Event link1 用 RSL

14.10.30 送信用通信コーデック設定レジスタ

Offset: Event Link : 0xfffe_0088 属性 リード／ライト
Offset: Data Link : 0xfffe_008c 属性 リード／ライト

31 24	23 16	15 8	7 0
CH4	CH3	CH2	CH1

Responsive Link 各チャネル対応する通信リンクの送信用通信コーデックを設定する．本レジスタの設定は，送信用通信速度・コーデックイネーブルレジスタで有効にしたチャネル・通信リンクにのみ適用され，受信用のコーデックは従来のレジスタで設定したものが用いられる．
チャネル毎の設定項目のビットマップを以下に示す．

7	6	5 4	3	2 0
RS	-	ECC	-	Line Code

Field Name	Function
CH*[7]	本ビットをセットすると Reed Solomon 符号によるエラー訂正が有効になる．
CH*[5:4]	バイト毎の ECC を設定する． 00: ECC なし 01: Hamming 符号 10: BCH 符号
CH*[2:0]	伝送路符号を設定する． 001: NRZI+BitStuffing 010: 8B10B 100: 4B10B

14.10.31 送信用通信速度・コーデックイネーブルレジスタ

Offset: 0xfffe_0090 属性 リード／ライト

31 29	28 25	24 21	20 17	16 13	12 9	8 5	4 1	0
-	DTCE	-	DTME	-	ETCE	-	ETME	-

Responsive Link の送信用の通信速度・コーデックの設定を有効にする．
本レジスタで送信用の通信速度を有効にすると，送信用 DPLL モード設定レジスタの対応するチャネルの通信リンクの送信用 DPLL モードが適用される．
本レジスタで送信用の通信コーデックを有効にすると，送信用通信コーデック設定レジスタの対応するチャネルの通信リンクの送信用 DPLL モードが適用される．
本レジスタで送信用通信速度・コーデックの設定を有効にしていないチャネル・通信リンクについては，レスポンシブリンク速度設定レジスタおよび通信コーデック設定レジスタにおける設定が送受信共に適用される．
0: 送信用 mode/codec を無効にする
1: 送信用 mode/codec を有効にする

Field Name	Function
DTCE	Data Link の送信用 codec 設定を有効にする DTCE4: Data link4 DTCE3: Data link3 DTCE2: Data link2 DTCE1: Data link1
DTME	Data Link の送信用 DPLL 設定を有効にする DTME4: Data link4 DTME3: Data link3 DTME2: Data link2 DTME1: Data link1
ETCE	Event Link の送信用 codec 設定を有効にする ETCE4: Event link4 ETCE3: Event link3 ETCE2: Event link2 ETCE1: Event link1
ETME	Event Link の送信用 DPLL 設定を有効にする ETME4: Event link4 ETME3: Event link3 ETME2: Event link2 ETME1: Event link1

14.10.32 モード 1 用サンプリングエッジ設定レジスタ

Offset: 0xfffe_0094 属性 リード／ライト

31 21	20 17	16 5	4 1	0
-	DD1M	-	ED1M	-

Responsive Link のモード 1 設定時に，クロックの立ち上がりエッジと立ち下がりエッジどちらでサンプリングを行うか設定する．

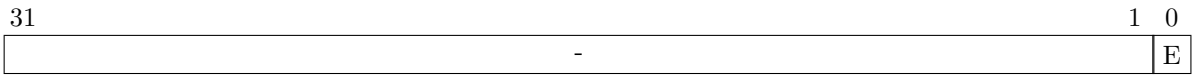
0: クロックの立ち上がりエッジを使用する．

1: クロックの立ち下がりエッジを使用する．

Field Name	Function
DD1M	Data Link のサンプリングエッジの設定 DD1M[4]: Data link4 DD1M[3]: Data link3 DD1M[2]: Data link2 DD1M[1]: Data link1
ED1M	Event Link のサンプリングエッジの設定 ED1M[4]: Event link4 ED1M[3]: Event link3 ED1M[2]: Event link2 ED1M[1]: Event link1

14.10.33 Routing Table ECC 設定レジスタ

Offset: 0xfffe_0098 属性 リード／ライト



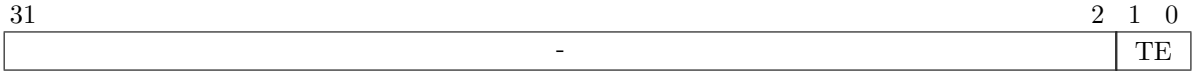
Responsive Link の Routing Table に付属している ECC のオンオフを設定する．

0: Disable.
1: Enable.

Field Name	Function
E	Routing Table の ECC のオンオフを設定する．

14.10.34 タイムアウト設定レジスタ

Offset: 0xfffe_009c 属性 リード／ライト



イベントリンク出力用 DPM，データリンク出力用 DPM に有効なデータが存在し，後述のタイムアウトカウント設定レジスタで設定したカウント数だけアクセスがなかった場合，RL_EVENT_TIMEOUT 割込み，RL_DATA_TIMEOUT 割込みをそれぞれ発生させるかを設定する．
0: Disable.
1: Enable.

Field Name	Function
TE	タイムアウト割込みを発生させるかを設定する． 0: EVENT LINK. 1: DATA LINK.

14.10.35 イベントリンクタイムアウトカウント設定レジスタ

Offset: 0xfffe_00a0 属性 リード／ライト

イベントリンク出力用 DPM に有効なデータが書き込まれてから，RL_EVENT_TIMEOUT 割込みが発生するまでの時間を設定する．
DPM に書き込み後，指定した時間が経過するより早く DPM にアクセスがあった場合，割込みは発生しない．

Field Name	Function
Timeout Count	タイムアウト割込みを発生させるまでの時間を設定する． 単位は clock cycle.

14.10.36 データリンクタイムアウトカウント設定レジスタ

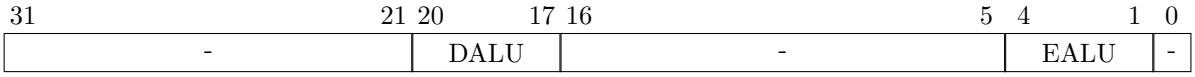
Offset: 0xfffe_00a4 属性 リード／ライト

データリンク出力用 DPM に有効なデータが書き込まれてから，RL_DATA_TIMEOUT 割込みが発生するまでの時間を設定する．
DPM に書き込み後，指定した時間が経過するより早く DPM にアクセスがあった場合，割込みは発生しない．

Field Name	Function
Timeout Count	タイムアウト割込みを発生させるまでの時間を設定する． 単位は clock cycle.

14.10.37 オートリンクアップ設定レジスタ

Offset: 0xfffe_00a8 属性 リード／ライト



リンクの初期化による通信の確立を自動で行う，オートリンクアップ機能の設定を行う．
オートリンクアップ機能を有効にすると，リンクの初期化を明示的に行わなくとも，イニシャルパケットを自動で送信し，リンクアップを行う．

0: オートリンクアップを無効にする．
1: オートリンクアップを有効にする．

Field Name	Function
DALU	Data Link のオートリンクアップの設定 DALU[4]: Data link4 DALU[3]: Data link3 DALU[2]: Data link2 DALU[1]: Data link1
EALU	Event Link のオートリンクアップの設定 EALU[4]: Event link4 EALU[3]: Event link3 EALU[2]: Event link2 EALU[1]: Event link1

14.11 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．

14.11.1 Event Output

図 14.9 にイベントリンク出力用 DPM の構成を示す．Event out control register（図 14.10 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 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．）

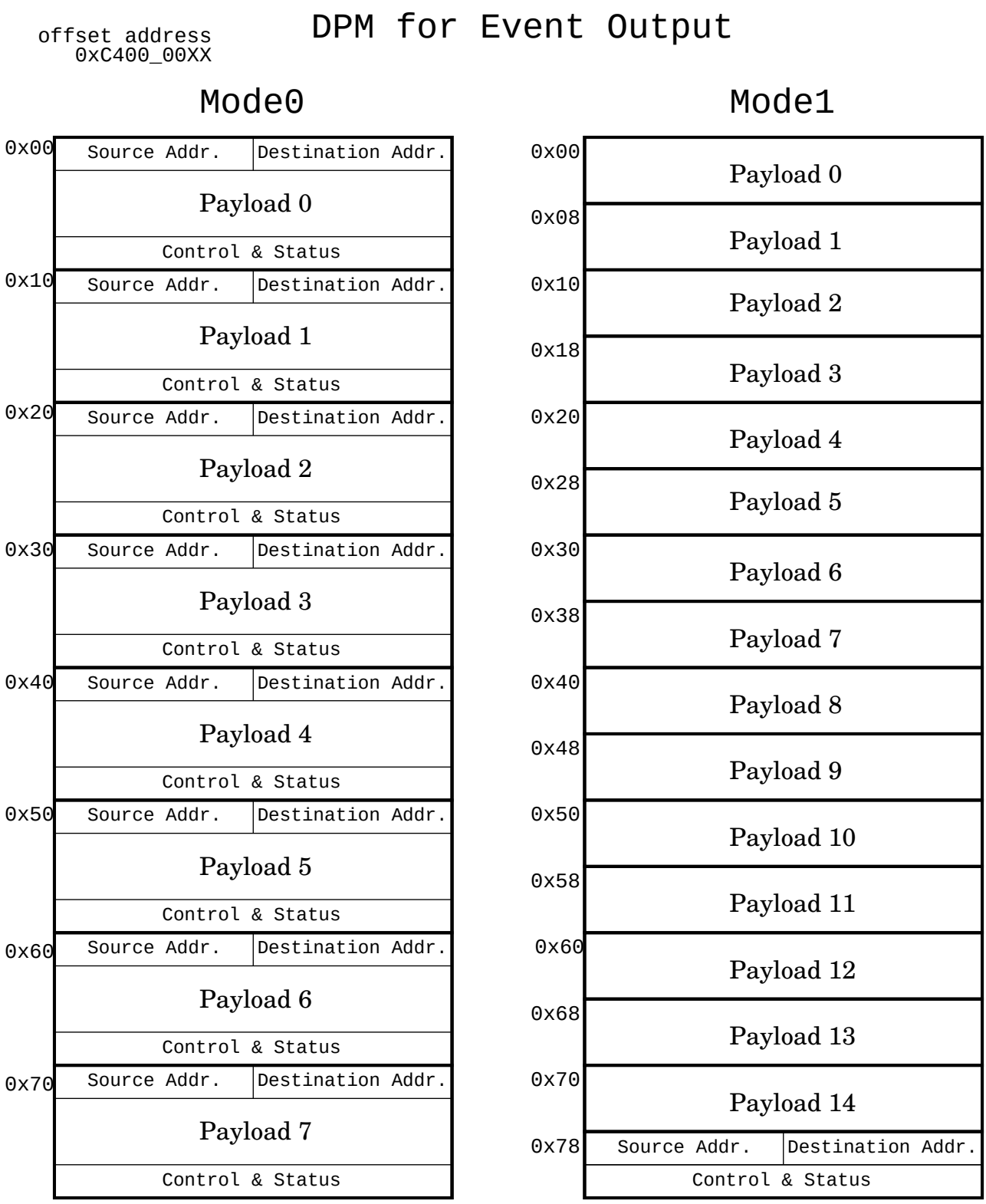


Figure 14.9: DPM for Event Output

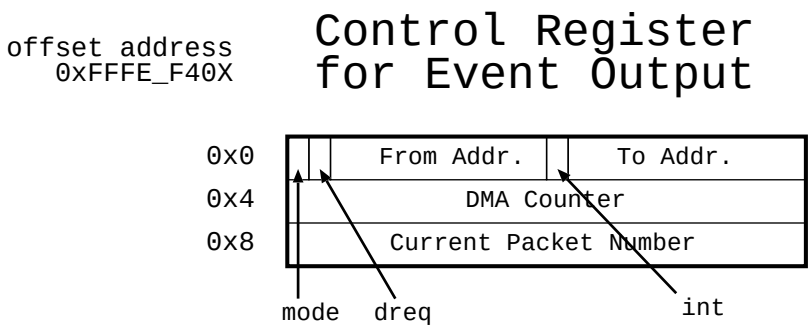


Figure 14.10: Event Out Control Register

DPM 制御レジスタ

DPM の制御レジスタ（図 14.10 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 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 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 14.9 の payload 番号に相当）を示す

14.11.2 Event Input

図 14.11 にイベントリンク入力用 DPM の構成を示す．Event in control register（図 14.12 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 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.）

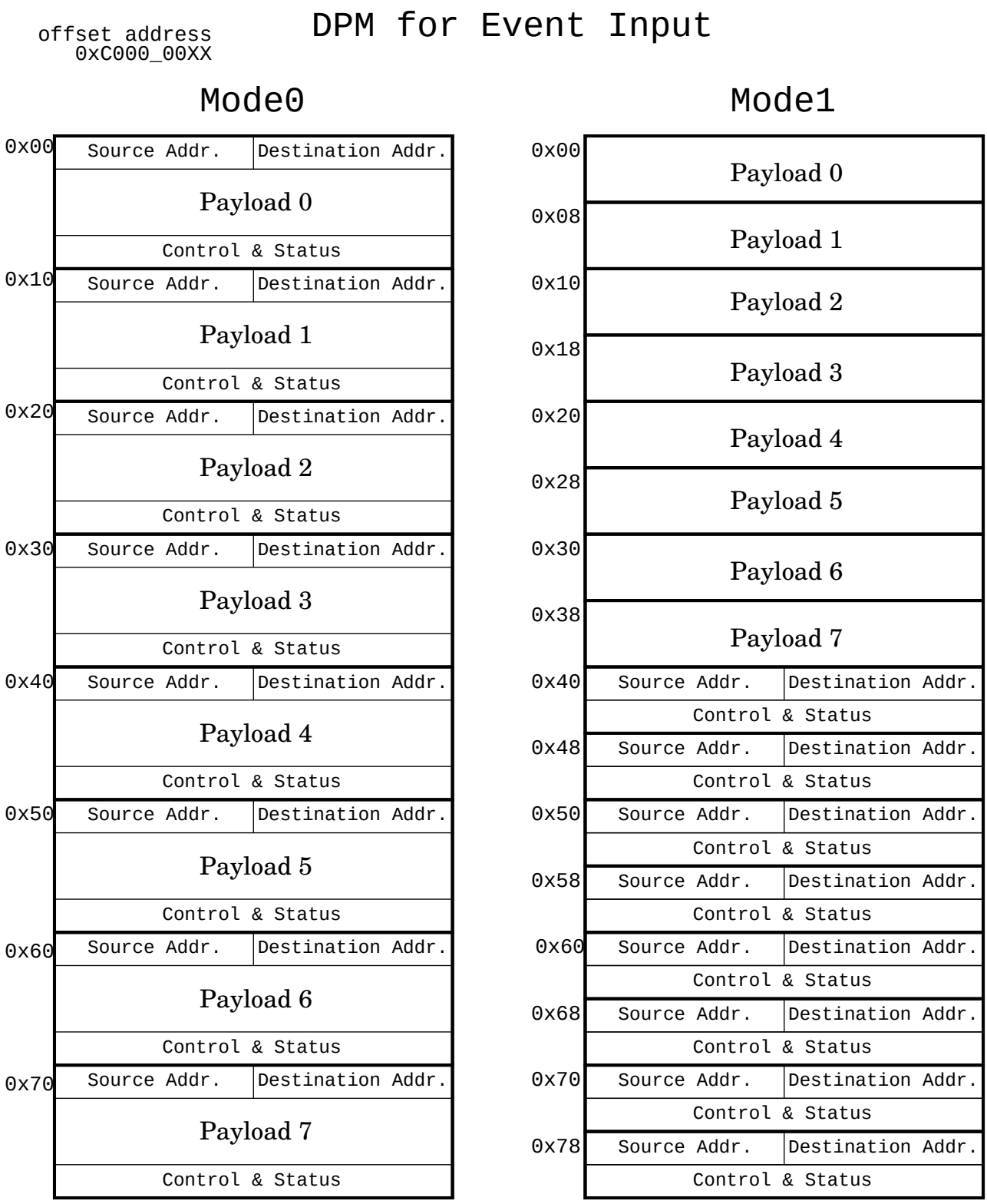


Figure 14.11: DPM for Event Input

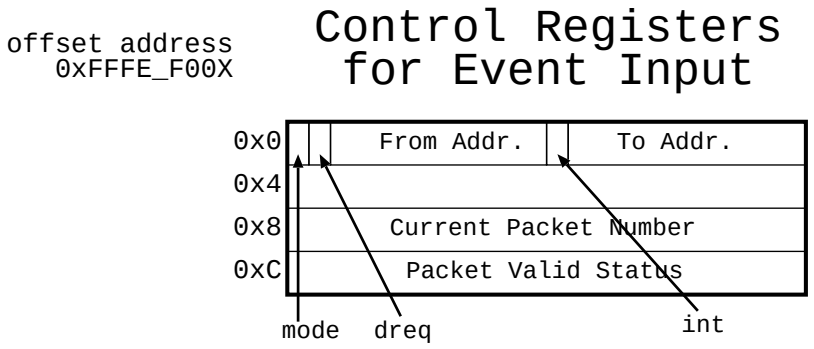


Figure 14.12: Event in control register

DPM 制御レジスタ

DPM の制御レジスタ（図 14.12 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 14.11 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 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 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 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) 現在受信しているパケット番号（図 14.11 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

14.11.3 Data Output

図 14.13 にデータリンク出力用 DPM の構成を示す．Data out control register（図 14.14 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 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.）

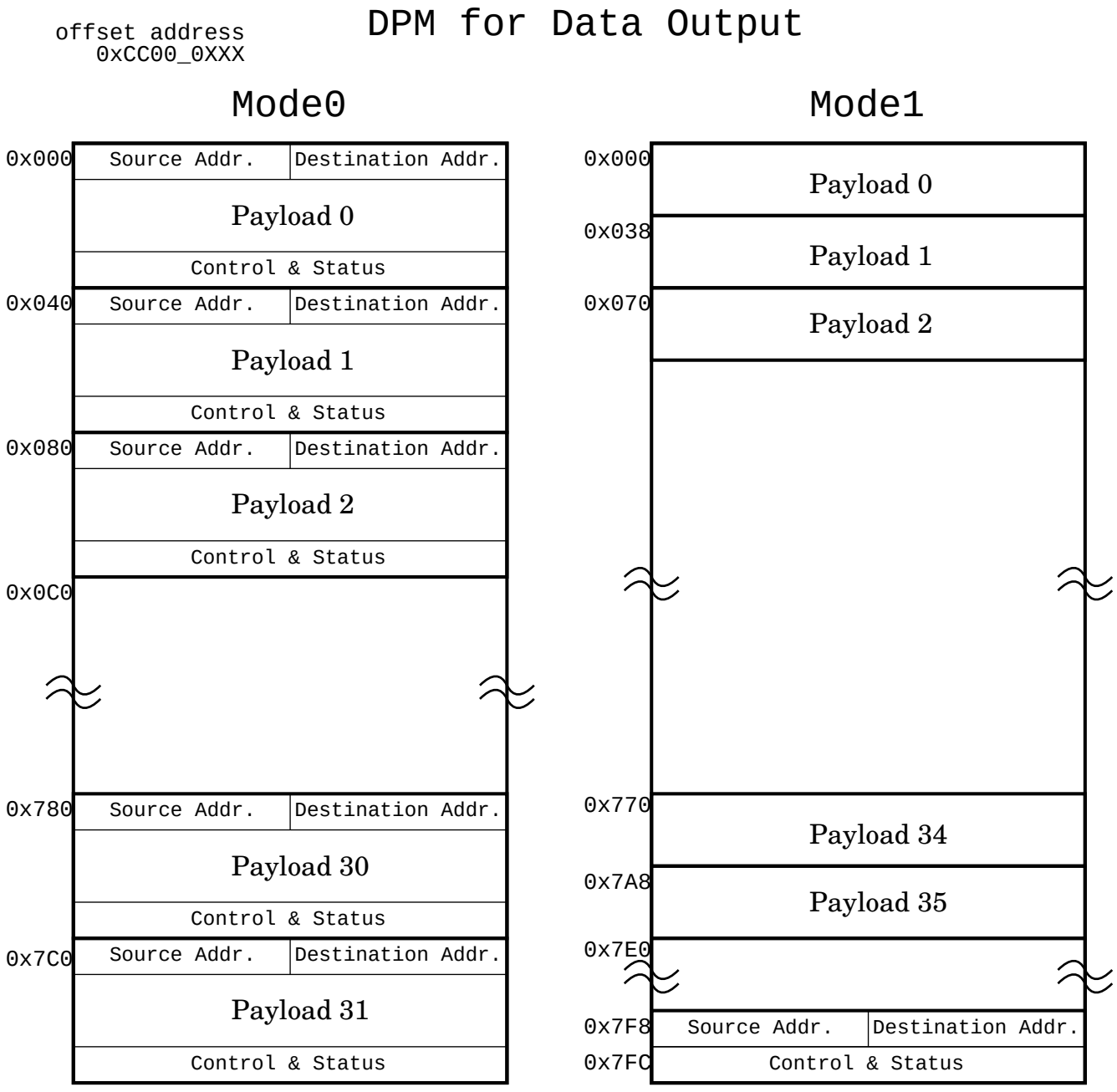


Figure 14.13: DPM for Data Output

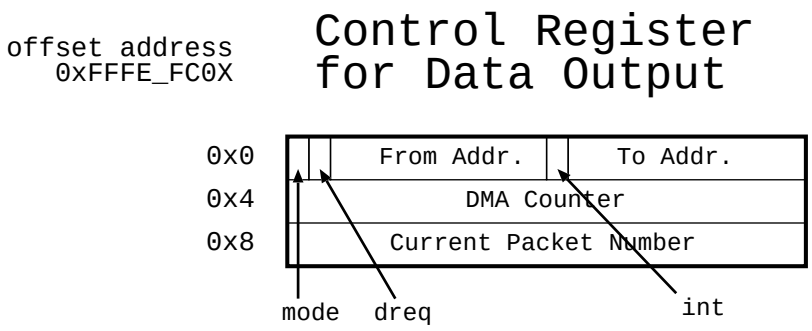


Figure 14.14: Data Out Control Register

DPM 制御レジスタ

DPM の制御レジスタ（図 14.14 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 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 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 14.13 の payload 番号に相当）を示す

14.11.4 Data Input

図 14.15 にデータリンク入力用 DPM の構成を示す．Data in control register（図 14.16 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 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.）

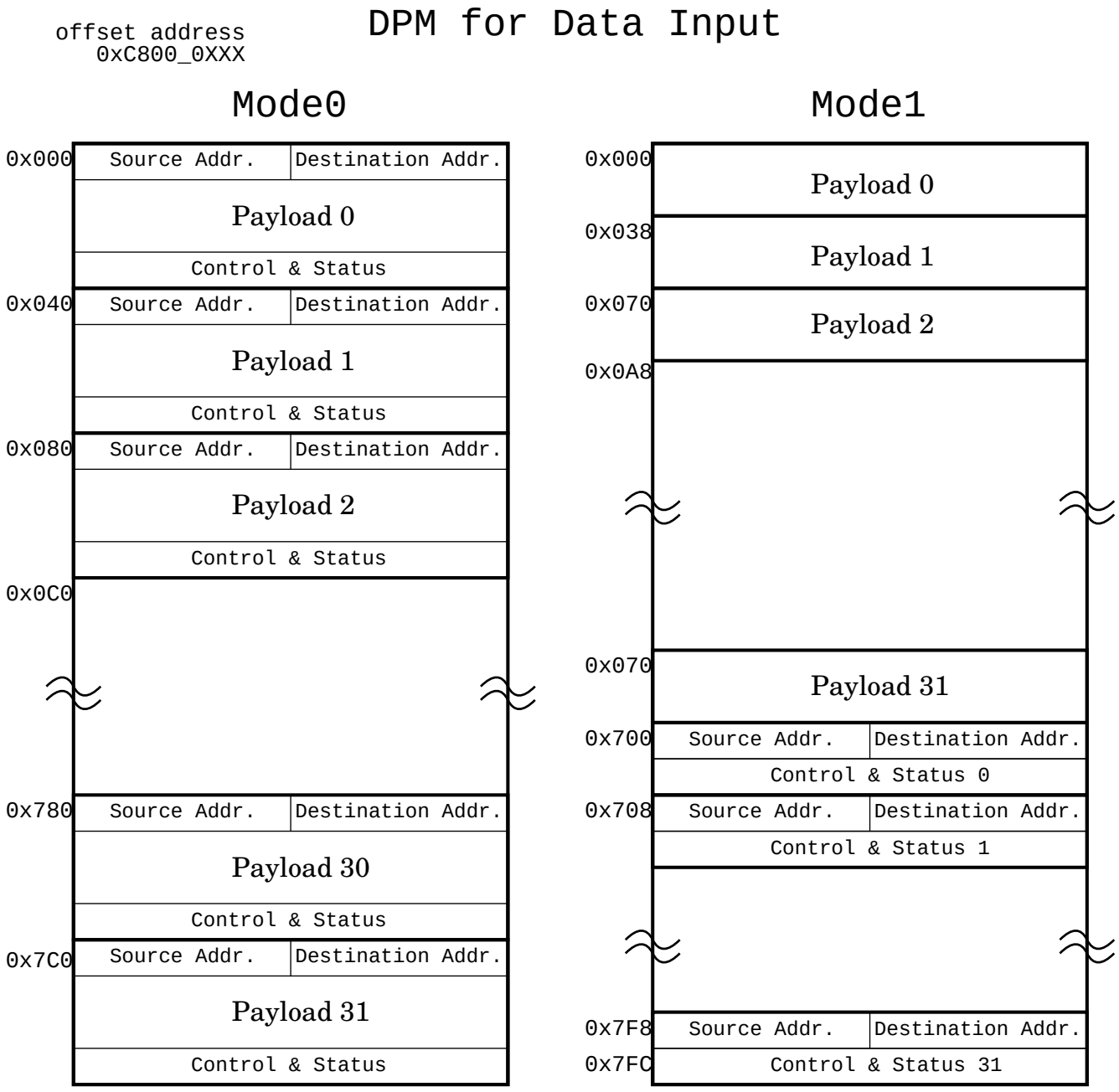


Figure 14.15: DPM for Data Input

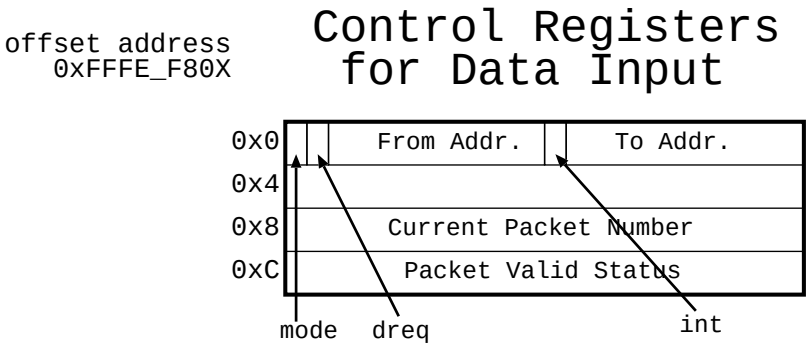


Figure 14.16: Data In Control Register

DPM 制御レジスタ

DPM の制御レジスタ（図 14.16 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 14.15 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 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) 現在受信しているパケット番号（図 14.15 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

14.12 通信方法

14.12.1 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 f<36，イベントリンクでは f<15 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)

14.12.2 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.)

14.13 *Responsive Link* の割り込みコントローラ

IRQ1∼4 は DPM のコントロールレジスタの from addr と to addr までパケットが到達した時の割込みである．また，IRQ1∼6 はレスポンシブリンク割込みクリアレジスタ (0xfffe_000c) に対応している．

Initial Address: *Responsive Link* IRC: 0xfffe1000

14.13.1 レジスタマップ

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
0x00	31ch	30ch	29ch	28ch	27ch	26ch	25ch	24ch	23ch	22ch	21ch	20ch	19ch	18ch	17ch	16ch	15ch	14ch	13ch	12ch	11ch	10ch	9ch	8ch	7ch	6ch	5ch	4ch	3ch	2ch	1ch	0x00
0x04	Request Sense Register																															0
0x08	Request Clear Register																															0
0x0c	Mask Register																															MI
0x10	26'h0																													CL	IRL Latch	
0x14	31'h0																															Mode
0x18																																

Table 14.4: *Responsive Link* レジスタマップのオフセット

Offset	Name
0x00	RL_IRC_TMR0_OFFSET
0x04	RL_IRC_TMR1_OFFSET
0x08	RL_IRC_RSR_OFFSET
0x0c	RL_IRC_RCR_OFFSET
0x10	RL_IRC_MR_OFFSET
0x14	RL_IRC_ICR_OFFSET
0x18	RL_IRC_MOD_OFFSET

Table 14.5: *Responsive Link* IRC 割り込みマップ

IRQ	Name
IRQ31∼IRQ14	Reserved
IRQ17	RL_EVENT_TIMEOUT_IRC
IRQ16	RL_DATA_TIMEOUT_IRC
IRQ15	RL_ROUTING_FATAL_IRC
IRQ13	RL_DEC_RESET_IRC
IRQ12	RL_IRQ_DOWN
IRQ11	RL_IRQ_WAKEUP
IRQ10	RL_IRQ_FATAL
IRQ9	RL_IRQ_TABLE
IRQ8	RL_IRQ_WAIT
IRQ7	RL_IRQ_CONT
IRQ6	RL_IRQ_EVP_IN
IRQ5	RL_IRQ_DAP_IN
IRQ4	RL_IRQ_EV_INEOP
IRQ3	RL_IRQ_DA_INEOP
IRQ2	RL_IRQ_EV_OUTEOP
IRQ1	RL_IRQ_DA_OUTEOP

IRQ	Name	Description
RL_EVENT_TIMEOUT_IRC	Event Timeout IRQ	Event Input DPM に有効なデータが入っている状態で設定した時間だけアクセスがないと発生
RL_DATA_TIMEOUT_IRC	Data Timeout IRQ	Data Input DPM に有効なデータが入っている状態で設定した時間だけアクセスがないと発生
RL_ROUTING_FATAL_IRC	Routing Table ECC 致命的エラー割込み (Fatal IRQ)	Routing Table に回復不可能なエラーが存在する場合に発生
RL_DEC_RESET_IRC	-	デコーダーリセットしたときの割込み
RL_IRQ_DOWN	-	リンクダウンしたら発生
RL_IRQ_WAKEUP	-	リンクアップしたら発生
RL_IRQ_FATAL	致命的エラー割込み FI (Fatal IRQ)	受信パケットに回復不可能なエラーが存在する場合に発生
RL_IRQ_TABLE	ルーティングテーブル割込み RTIRQ (Routing IRQ)	ルーティングテーブルにマッチするエントリが存在しない場合に発生
RL_IRQ_WAIT	送信停止割込み WIRQ (Wait IRQ)	パケット追い越し用 SDRAM を使用している際に，追い越し用 SDRAM が溢れそうになると送信停止割込みを自動生成する
RL_IRQ_CONT	レスポンシブリンク継続割込み CI (Continuous IRQ)	SDRAM に退避されたパケットがスイッチに書き戻された（再送信された）際に発生
RL_IRQ_EVP_IN	Event Packet-In IRQ	割込みフラグ付きの event パケットを受信した際に発生
RL_IRQ_DAP_IN	Data Packet-In IRQ	割込みフラグ付きの data パケットを受信した際に発生
RL_IRQ_EV_INEOP	Event-In End of Packet	Event-In DPM で設定した場所までパケットを受信した際に発生
RL_IRQ_DA_INEOP	Data-In End of Packet	Data-In DPM で設定した場所までパケットを受信した際に発生
RL_IRQ_EV_OUTEOP	Event-Out End of Packet	Event-Out で DPM からパケットを発射した際に発生
RL_IRQ_DA_OUTEOP	Data-Out End of Packet	Data-Out で DPM からパケットを発射した際に発生

15

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)

15.1 レジスタマップ

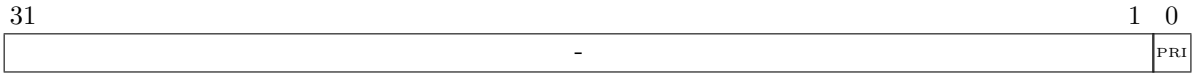
DMAC	初期アドレス
DMAC0	FFFF0000
DMAC1	FFFF1000
DMAC2	FFFF2000
DMAC3	FFFF3000
DMAC4	FFFF4000
DMAC DIAG	FFFF5000

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)+0x10	- DAS SAU BM RL PCM TTM MR 32P 16P 8P S16 S8 IER IED ST
0x40*(x)+0x14	- ER ED
0x40*(x)+0x18	LN<31:0>

15.1.1 DMA 制御レジスタ

ライト／リード

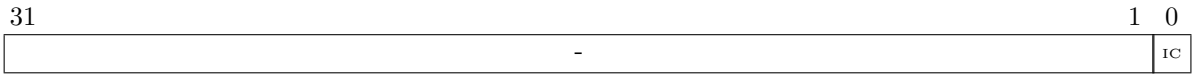
Offset: 0x800



Field Name	Function
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

15.1.2 DMA 割り込みクリアレジスタ

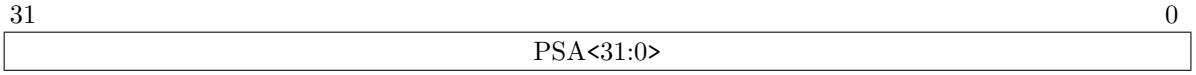
Offset: 0x804



Field Name	Function
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

15.1.3 ポート／ソースアドレスレジスタ

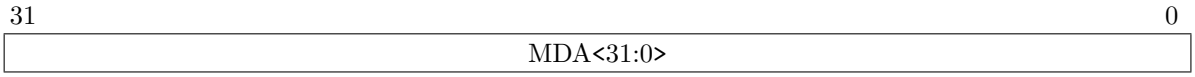
Offset: 0x40*(x) +0x04



Field Name	Function
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．

15.1.4 メモリ／デスティネーションアドレスレジスタ

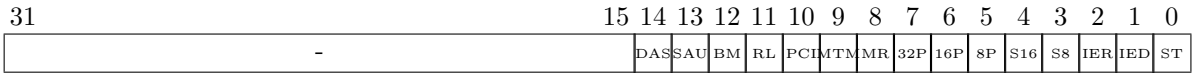
Offset: 0x40*(x) +0x08



Field Name	Function
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.

15.1.5 転送モード制御レジスタ

Offset: 0x40*(x) +0x10



ライト／リード

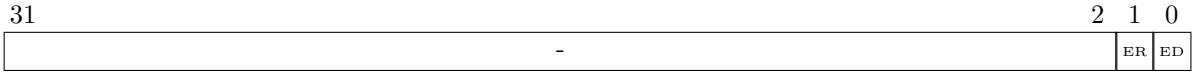
Field Name	Function
DAS	Destination Address Update :Default X 0:メモリアドレスレジスタで設定したアドレスが次の転送にも使用される． 1:メモリアドレスレジスタの値は，最後に転送を行ったアドレスより 1 ワード先を示す.
SAU	Source Address Update :Default X 0:ポートアドレスレジスタで設定したアドレスが次の転送にも使用される． 1:ポートアドレスレジスタの値は，最後に転送を行ったアドレスより 1 ワード先を示す.
BM	Burst Mode :Default X 0:バースト転送長を Auto Negotiation で決定する．1:強制的にバースト転送する．基本的に On にはしないこと
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)	○	×	×	○

15.1.6 ステータスレジスタ

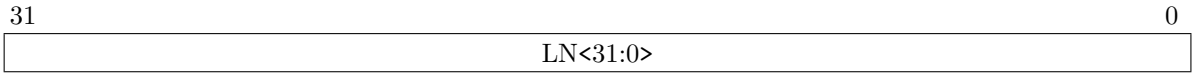
Offset: 0x40*(x)+0x14 ライト／リード



Field Name	Function
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．

15.1.7 転送レングスレジスタ

Offset: 0x40*(x) +0x18



Field Name	Function
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．

15.2 I/O DMA リクエスト

DMAC の各チャネルには I/O からの DMA リクエストを受けて転送を自動で開始する機能が存在する．

DMAC0	ch0	reserved
	ch1	reserved
	ch2	reserved
	ch3	reserved
DMAC1	ch0	Extbus0
	ch1	ExtBus1
	ch2	reserved
	ch3	reserved
DMAC2	ch0	RL event in
	ch1	RL data in
	ch2	RL event out
	ch3	RL data out
DMAC3	ch0	SPI0
	ch1	SPI1
	ch2	ExtBus2
	ch3	ExtBus3
DMAC4	ch0	UART0 RX
	ch1	UART0 TX
	ch2	UART1 RX
	ch3	UART1 TX

16

DMACDIAG

- 32/16/8 bit I/F
- 入力チャネル：1
- 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)
- DMAC モード/Write Only モード/Read with Compare モード/Write and Read with Compare モード

DMAC DIAG は DMA Controller にメモリチェック機能を追加したモジュールである．通常の DMA Contoroller の機能に加えて 3 種類のメモリチェックモードを持つ．表 16.1 に各転送モードの概要を示す．

Table 16.1: DMAC DIAG の転送モード

転送モード	概要
DMAC モード	通常の DMA Controller と同様の転送を行う
Write Only モード	データバッファの値をサイクリックに書き込む
Read with Compare モード	読み込んだ値とデータバッファの値を比較する
Write and Read with Compare モード	データバッファの値をサイクリックに書き込んだ後，同じアドレスに対して読み出しを行い，データバッファの値と比較する．

16.1 レジスタマップ

	ベースアドレス
DMAC DIAG	0xFFFF5000

offset	31 24 23 16 15 8 7 0
0x800	- PRI
0x804	- IC
0x808	AE - cmp err
0x80C	Current Error Address
0x810, 0x818	Error Address
0x814, 0x81C	Error Data
0xC00 - 0xC1C	Data
0x04	PSA< 31 : 0 >
0x08	MDA< 31 : 0 >
0x10	- DM BL OTI OTM DAS SAU BM RL PCM TM MR 32P 16P 8P S16 S8 IER IED ST
0x14	- ER ED
0x18	LN< 31 : 0 >
0x400	- Reset
0x404	- AN
0x408	- Compare

16.1.1 DMA 制御レジスタ

ライト／リード

Offset: 0x800

31 1 0
- PRI

Field Name	Function
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 $ch0 > ch1 > ch2 > ch3$ DMAC DIAG のチャンネルは 1 つのみであるが、本ビットは DMAC との互換性を保つため残されている

16.1.2 DMA 割り込みクリアレジスタ

Offset: 0x804

31 1 0
- IC

Field Name	Function
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

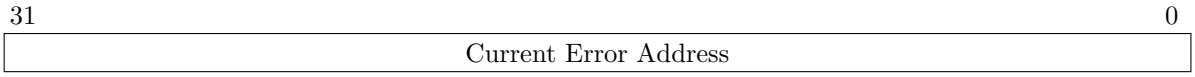
16.1.3 コンペアリザルトレジスタ

Offset: 0x808

Field Name	Function
AE	All Compare Error: 本ビットはメモリチェックで発生したエラー状況を示す. 0: 全てのメモリチェックが正常 1: 1 箇所以上メモリチェックエラーが発生
cmp err	Compare Error データバッファとの比較結果を示す. 0: 正常 1: ビット位置に対応する Data Buffer との比較時にエラーが発生

16.1.4 カレントエラーアドレスレジスタ

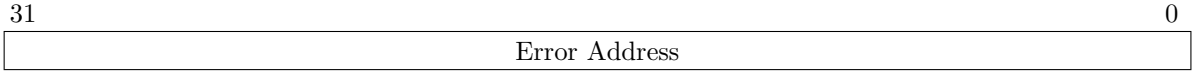
Offset: 0x80C



Field Name	Function
Current Error Address	本レジスタはメモリチェックでエラーが発生した最新のアドレスを示す.

16.1.5 エラーアドレスレジスタ

Offset: 0x810, 0x818



Field Name	Function
Error Address	本レジスタはメモリチェックでエラーが発生したアドレスを示す.

16.1.6 エラーデータレジスタ

Offset: 0x814, 0x81C

Field Name	Function
Error Address	本レジスタはメモリチェックでエラーが発生した時の読み込んだ値を示す.

16.1.7 データバッファレジスタ

Offset: 0xC00 - 0xC1C

31	0
Data	

Field Name	Function
Data	DMA 転送用のバッファ．本バッファを用いて書き込みや比較を行う.

16.1.8 ポート／ソースアドレスレジスタ

Offset: 0x04

31	0
PSA< 31 : 0 >	

Field Name	Function
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.

16.1.9 メモリ／デスティネーションアドレスレジスタ

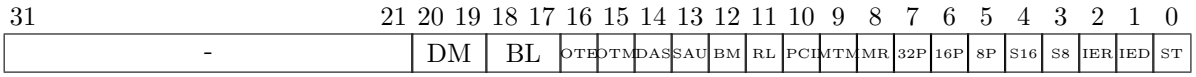
Offset: 0x08

31	0
MDA< 31 : 0 >	

Field Name	Function
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.

16.1.10 転送モード制御レジスタ

Offset: 0x10



ライト／リード

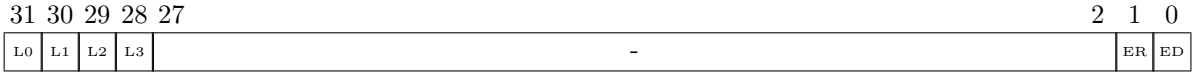
Field Name	Function
DM	Diag Mode : Default 0x0 DMAC DIAG の動作モードを設定する． 0x0: DMAC Mode（通常の DMAC の動作と等しい）0x1: Write Only Mode 0x2: Read with Compare Mode 0x3: Write and Read with Compare Mode
BL	Burst Length : Default 0x0 最大バースト長を設定する． 0x0: 8 バースト 0x1: 4 バースト 0x3: シングル転送
OTE	One Time End : Default 0x0 本ビットが 1 の時，One Time Mode での転送が終了したことを示す．
OTM	One Time Mode : Default 0x0 DMAC DIAG の転送回数を設定する． 0: DMAC DIAG を通常通り使用する 1: DMAC DIAG を 1 度のみ使用する
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

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)	○	×	×	○

16.1.11 ステータスレジスタ

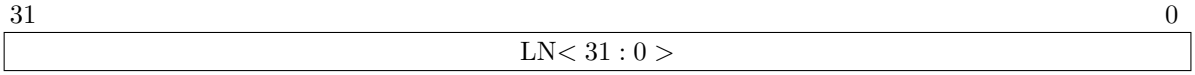
Offset: 0x14 ライト／リード



Field Name	Function
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．

16.1.12 転送レングスレジスタ

Offset: 0x18



Field Name	Function
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．

17

バスサイジング機能付きDMA

17.1 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．転送時のアドレス及び転送長は 32byte アラインでなければならない．端数バイトは切り捨てられる．

offset	31 24	23 16	15 8	7 0
0x00	-			IC
0x04	PSA<31:0>			
0x08	MDA<31:0>			
0x10	-	DAU SAU BM	- MTM MR	- IER IED ST
0x14	-			ER ED
0x18	LN<31:0>			

17.2 制御レジスタ詳細

17.2.1 DMA割り込みクリアレジスタ

Offset: 0x00

31 1	0
-	IC

Field Name	Function
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

17.2.2　ポート／ソースアドレスレジスタ

Offset: 0x04

31	0
PSA<31:0>	

Field Name	Function
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.

17.2.3　メモリ／デスティネーションアドレスレジスタ

Offset: 0x08

31	0
MDA<31:0>	

Field Name	Function
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.

17.2.4　転送モード制御レジスタ

Offset: 0x10

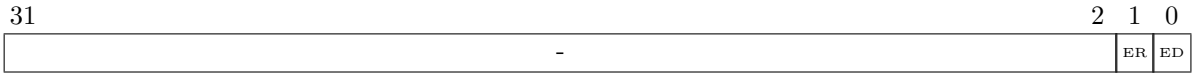
31	15	14	13	12	11	10	9	8	7	3	2	1	0
-		DAU	SAU	BM	-		MTM	MR	-		IER	IED	ST

ライト／リード

Field Name	Function
DAU	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．32bit-DMAC とは異なり、これがないと bmreq を出さないため、基本的には 1 にする
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.
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.

17.2.5 ステータスレジスタ

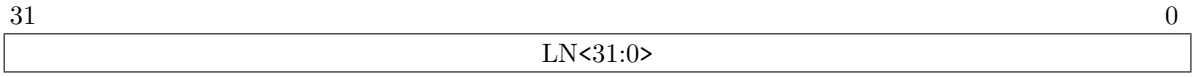
Offset: 0x14 ライト／リード



Field Name	Function
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.

17.2.6 転送レングスレジスタ

Offset: 0x18



Field Name	Function
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．

18

パルスカウンタ

18.1 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）
- Z 相によるリセット／割り込み機能（ソフトウェアで選択可能）
- 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
- 外部入力
- チャネル数：4

18.2 レジスタインタフェース

18.2.1 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	PLSCTRL[0]
0xFFFF7020	PLSCTRL[1]
0xFFFF7040	PLSCTRL[2]
0xFFFF7060	PLSCTRL[3]

リード／ライト時

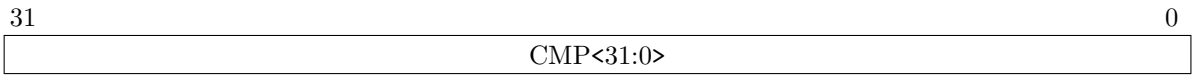
31	30	12	11	10	9	8	7	6	5	4	3	2	1	0
INT	-		CPCE	SIZE	ZF	RFZ	ST	TI	SEL	MD		IE	CLR	CE

Field Name	Function
INT	Interrupt :Default 0 ro 0:割込みの発生なし．1:割込みが発生している．割り込みはパルスカウンタ割り込み，タイマ割り込み，Z 相割り込みのいずれかの要因で発生する．本レジスタをリードすると，パルスカウンタ割り込みとタイマ割り込みがクリアされる．
IPCE	Int Pulse Counter Enable :Default 0 0 : パルスカウンタによる割り込みを発生させない．1 : パルスカウンタによる割り込みを発生させる．カウンタ値がコンペアデータレジスタの値と等しくなると割り込みを発生する．
IZE	Int Z Enable :Default 0 0 : Z 相入力があった際に，割り込みを発生させない．1 : Z 相入力があった際に，割り込みを発生させる．
ZF	Z Flag :Default 0 0 : 現状態は Z 相ではない．1 : Z 相入力があった際に 1 に設定される．クリアする際には 0 を書く．Z 相割り込み (IZE) を有効にしている場合，0 を書くと Z 相割り込みをクリアする．
RFZ	Reset Flag by phaze Z :Default 0 0 : Z 相入力によるカウンタのリセットを行わない．1 : Z 相入力によるカウンタのリセットを行う．
ST	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．
TI	Timer Interrupt :Default 0 0 : 内部タイマによる周期割り込みを発生させない．1 : 内部タイマによる周期割り込みを発生させる．
SEL	Select :Default 0 カウンタのラッチの動作モードの選択を行う．0 : カウンタ値のラッチを行わない．1 : 内部タイマにより設定された値によって，周期的にカウンタ値をラッチする．
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．
IE	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
CE	Count Enable :Default 0 0:カウンタを停止する 1:カウンタを起動する

18.2.2　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	CMP[0]
0xFFFF7024	CMP[1]
0xFFFF7044	CMP[2]
0xFFFF7064	CMP[3]

リード／ライト時



Field Name	Function
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．

18.2.3 カウンタレジスタ

アドレス	カウンタレジスタ
0xFFFF7008	CNT[0]
0xFFFF7028	CNT[1]
0xFFFF7048	CNT[2]
0xFFFF7068	CNT[3]

リード時

31	0
CNT<31:0>	

Field Name	Function
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.

18.2.4 タイマレジスタ

アドレス	タイマレジスタ
0xFFFF700C	TIMER[0]
0xFFFF702C	TIMER[1]
0xFFFF704C	TIMER[2]
0xFFFF706C	TIMER[3]

リード／ライト時

31	0
TIMER<31:0>	

Field Name	Function
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.

19

PWM発生器

19.1 PWM 発生器概要

- 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
- Bit 幅：32bit
- ノコギリ波を用いて PWM を発生するノコギリ波モードと三角波を用いた三角波モード
- デッドタイムの設定
- 正論理，負論理の設定
- PWM チャネルをグルーピングしてグループ毎に同期可能
- PWM の周期毎の割り込み可能
- 汎用出力としても利用可能
- チャネル数：12

図 19.1 にノコギリ波モード，図 19.2 に三角波モードの PWM 波形を示す．

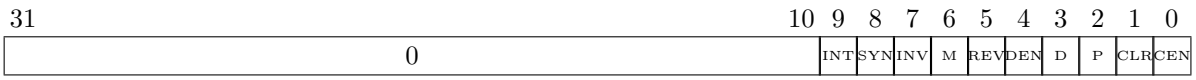
デッドタイム付反転出力は，隣の PWM 発生器から出力することができるようにサイクリックにカスケード接続されている．具体的には，PWM 発生器 N のデッドタイム付反転出力は，REV bit を立てることにより，PWM 発生器 N+1 で利用することができる．

また，カウンタのスタートが同期した PWM のグループを作ることができる．

19.2 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]
0xFFFF7320	PWMCTRL[9]
0xFFFF7340	PWMCTRL[10]
0xFFFF7360	PWMCTRL[11]

リード／ライト



Field Name	Function
INT	Invert: Default 0 0: 割り込みを発生させない 1: 割り込みを発生させる 割り込みは，周期の開始時に発生される．
SYN	Invert: Default 0 0: カウントの開始を同期しない（CEN のみを使用する） 1: カウントの開始を同期する（本 PWM 発生器より一つ若い番号の PWM 発生器で生成されたスタート信号を使用する） 本 PWM のカウンタのスタート信号は，CEN が 1 になった時，または本 SYN が 1 かつ本 PWM 発生器より一つ若い番号のスタート信号が入力されたときにアクティブとなる．したがって，ある PWM 発生器の CEN を，引き続く複数の PWM 発生器で使用することができ，カウンタのスタートが同期した PWM のグループを作ることができる．
INV	Invert: Default 0 0: PWM 波をそのまま出力する 1: PWM 波を出力する最終段において PWM 波を反転して出力する 最も優先度が高い
M	Mode: Default 0 0: ノコギリ波で PWM を生成するノコギリ波モード 1: 三角波で PWM を生成する三角波モード
REV	Reverse mode enable: Default 0 0: 本 PWM 発生器で生成された PWM 波を出力する通常モード 1: 本 PWM 発生器より一つ若い番号の PWM 発生器で生成された PWM 波のデッドタイム付反転出力を出力するモード（本 PWM 発生器内のカウンタは使用しない） DEN より優先度が高い
DEN	Data Enable: Default 0 0: 生成した PWM 波を出力する 1: D bit に設定された値（一定値）を出力する REV より優先度が低い
D	Data: Default 0 DEN が 1 の時，本 D bit に設定された値（一定値）を出力する
P	Positive: Default 0 PWM 波の論理を決定する（図 19.1, 19.2 参照）． 0: 負論理 1: 正論理
CLR	Counter clear: Default 0 0: 通常動作 1: カウンタをクリアする
CEN	Count Enable: Default 0 0: カウンタを停止する 1: カウンタを起動する

19.3 PWM 周期制御レジスタ

アドレス	PWM 正転制御レジスタ
0xFFFF7204	FWCNT[0]
0xFFFF7224	FWCNT[1]
0xFFFF7244	FWCNT[2]
0xFFFF7264	FWCNT[3]
0xFFFF7284	FWCNT[4]
0xFFFF72A4	FWCNT[5]
0xFFFF72C4	FWCNT[6]
0xFFFF72E4	FWCNT[7]
0xFFFF7304	FWCNT[8]
0xFFFF7324	FWCNT[9]
0xFFFF7344	FWCNT[10]
0xFFFF7364	FWCNT[11]

リード／ライト

31	0
FWCNT<31:0>	

Field Name	Function
FWCNT	Forward Counter: Default 0 PWM の周期を決定するカウンタレジスタである． Mode が 0(ノコギリ波モード) の時には，PWM の周期を決定する．PWM 用カウンタが 0 から FWCNT までカウントアップすると，次のクロックで 0 に戻るようなノコギリ波を生成する（図 19.1 参照）． Mode が 1(三角波モード) の時には，PWM の半周期を決定する．PWM 用カウンタが 0 から FWCNT までカウントアップすると，次のクロックから 0 にカウントダウンするような三角波を生成する（図 19.2 参照）．

19.4 PWM 反転制御レジスタ

アドレス	PWM 反転制御レジスタ
0xFFFF7208	REVCNT[0]
0xFFFF7228	REVCNT[1]
0xFFFF7248	REVCNT[2]
0xFFFF7268	REVCNT[3]
0xFFFF7288	REVCNT[4]
0xFFFF72A8	REVCNT[5]
0xFFFF72C8	REVCNT[6]
0xFFFF72E8	REVCNT[7]
0xFFFF7308	REVCNT[8]
0xFFFF7328	REVCNT[9]
0xFFFF7348	REVCNT[10]
0xFFFF7368	REVCNT[11]

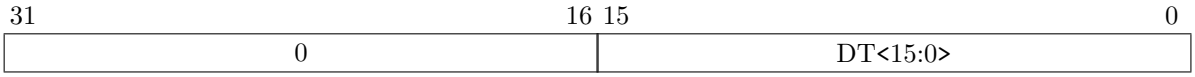
リード／ライト

Field Name	Function
REVCNT	Reverse Counter: Default 0 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（図 19.1, 19.2 参照）.

19.5 デッドタイムレジスタ

アドレス	デッドタイムレジスタ
0xFFFF720C	DT[0]
0xFFFF722C	DT[1]
0xFFFF724C	DT[2]
0xFFFF726C	DT[3]
0xFFFF728C	DT[4]
0xFFFF72AC	DT[5]
0xFFFF72CC	DT[6]
0xFFFF72EC	DT[7]
0xFFFF730C	DT[8]
0xFFFF732C	DT[9]
0xFFFF734C	DT[10]
0xFFFF736C	DT[11]

リード／ライト



Field Name	Function
DT<15:0>	Reverse Counter :Default 0 デッドタイムを指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（図 19.1, 19.2 参照）.

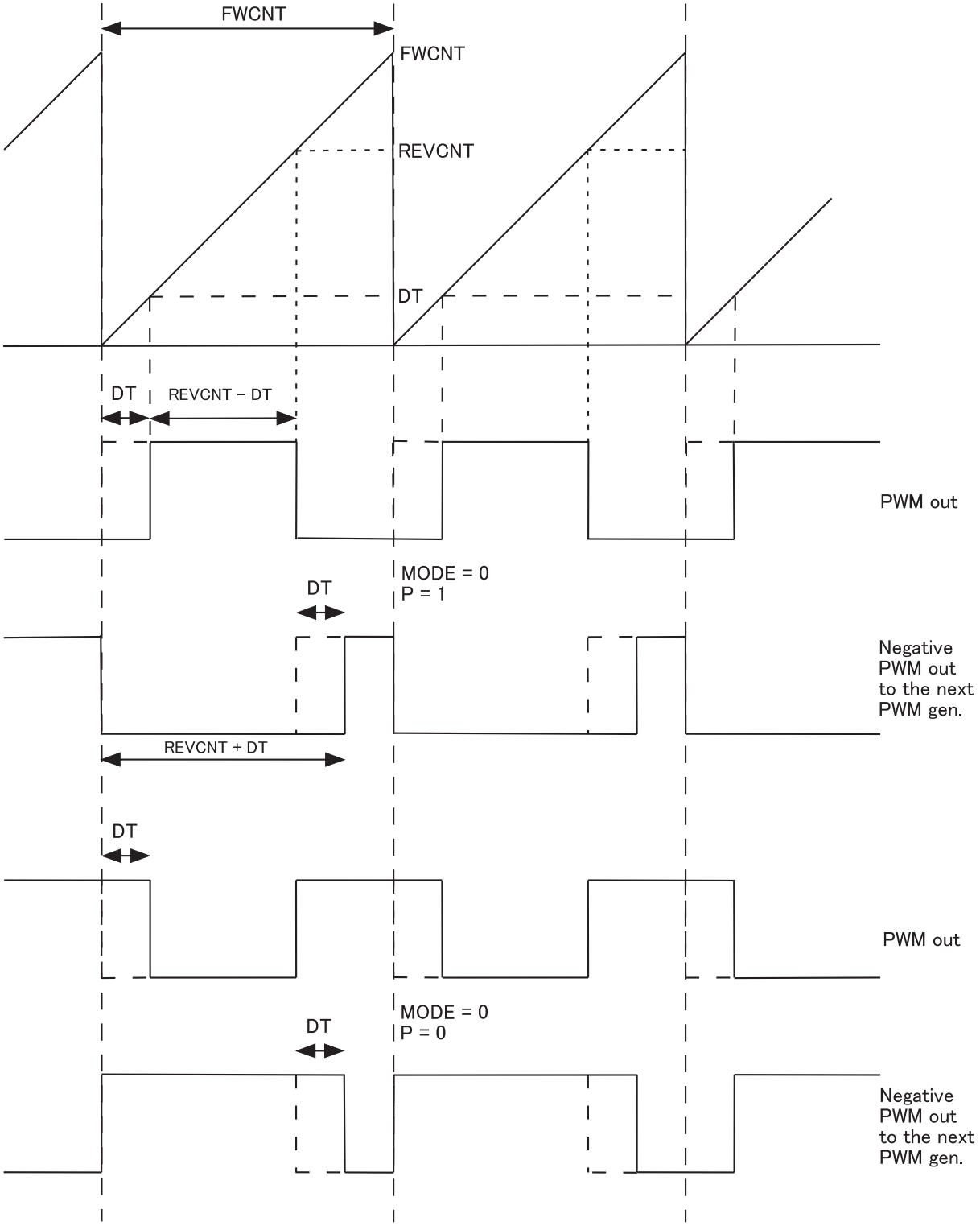


Figure 19.1: ノコギリ波モード

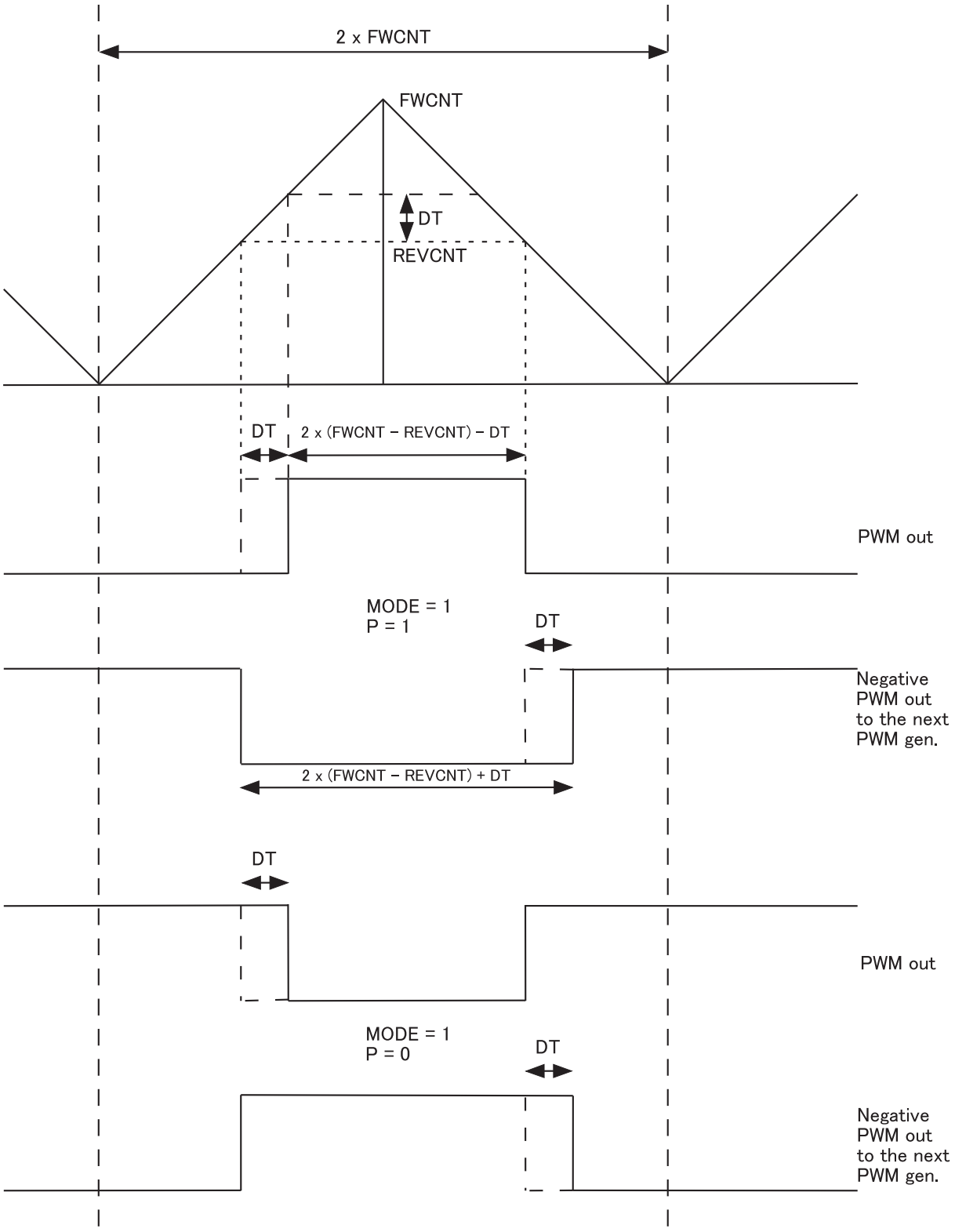


Figure 19.2: 三角波モード

20

PWM入力器

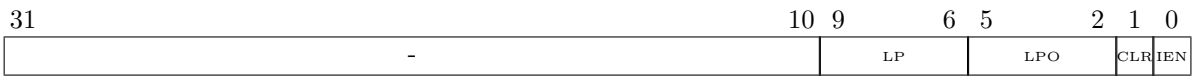
20.1 PWM 入力器概要

- PWM 入力のデューティ比を High カウンタと Low カウンタの比に数値化
- Bit 幅：32bit
- チャネル数：3
- クロックジェネレータ（9 章参照）で生成した基準クロックによってカウント
- 複数周期のデューティ比を求めて平均化する機構
- 割り込み発生機能

20.2 PWMIN コントロールレジスタ

アドレス	PWMIN コントロールレジスタ
0xFFFF7400	PWMINCTRL[0]
0xFFFF7420	PWMINCTRL[1]

リード／ライト



Field Name	Function
IEN	Interrupt Enable :Default 0 r/w 0: 割り込みを発生しない．1: 設定した周期分のデューティ比をカウント後に毎回割り込みを発生させる．
CLR	Interrupt Clear :Default 0 r/w 0:割り込みをクリアしない．1:割り込みをクリアする．割り込みクリア完了後に 0 にリセットされる．
LPO	Loop Original :Default 1 r/w 何周期分のデューティ比を平均化するか，その周期を設定する（1 から 15 まで，0 は禁止）．
LP	Loop :Default 1 ro 現在実行している周期を示す．

20.3 PWMIN HIGH レジスタ

アドレス	PWMIN HIGH レジスタ
0xFFFF7404	HIGH[0]
0xFFFF7424	HIGH[1]

リード

31 0

HIGH<31:0>

Field Name	Function
HIGH<31:0>	High :Default X 指定した PWM 周期分合計の High の期間．単位は，クロックジェネレータでプログラマブルに設定した PWMIN 用クロックのサイクル数．

20.4 PWMIN LOW レジスタ

アドレス	PWMIN LOW レジスタ
0xFFFF7408	LOW[0]
0xFFFF7428	LOW[1]

リード／ライト

31 0

LOW<31:0>

Field Name	Function
LOW<31:0>	Low :Default X 指定した PWM 周期分合計の Low の期間．単位は，クロックジェネレータでプログラマブルに設定した PWMIN 用クロックのサイクル数．

21

Ext Timer

21.1 概要

タイマは一定時間毎に割り込みを発生させるユニットである.
ワード長以外でのアクセスはサポートしない.

21.2 レジスタマップ

21.2.1 アドレスマップ

TIMER	初期アドレス
TIMER0	FFFF7800
TIMER1	FFFF7820
TIMER2	FFFF7840
TIMER3	FFFF7860

Table 21.1: 制御レジスタのアドレスマップ

名称	オフセット	アクセス	詳細
Control	0x0	R/W	コントロール
Interrupt	0x4	R/W	割り込み
Expiration	0x8	R/W	満了値
Counter	0xC	R/W	カウンタ

offset	31 24 23 16 15 8 7 0		
0x00	-	P	S
0x04	-		I
0x08	EXPR		
0x0c	COUNT		

21.2.2 ビットマップ

Control : コントロール

Offset: 0x00

31 2	1	0
Reserved	P	S

Field Name	Function
P	この bit が 0 の場合タイマはワンショットタイマとして動作し，この bit が 1 の場合タイマはピリオディックタイマとして動作する.
S	この bit に 1 がセットされた場合タイマが動作する.

Interrupt : 割り込み

Offset: 0x04

31 1	0
Reserved	I

Field Name	Function
Interrupt (I)	タイマが満了すると自動的にセットされる．この bit が 1 の場合割り込みが発生する.

Expiration : 満了値

Offset: 0x08

31 0
EXPR

Field Name	Function
Expiration (EXPR)	カウンタがこの値に達した場合に割り込みを発生させる.

Counter : カウンタ

Offset: 0x0c

Field Name	Function
Counter (COUNTER)	カウンタが満了値に達した場合に割り込みを発生させる.

22

64-bit Ext Timer

22.1 概要

タイマは一定時間毎に割り込みを発生させるユニットである.
ワード長以外でのアクセスはサポートしない.

22.2 レジスタマップ

22.2.1 アドレスマップ

TIMER	初期アドレス
TIMER0	FFFF7A00
TIMER1	FFFF7A20
TIMER2	FFFF7A40
TIMER3	FFFF7A60

Table 22.1: 制御レジスタのアドレスマップ

名称	オフセット	アクセス	詳細
Control	0x0	R/W	コントロール
Interrupt	0x4	R/W	割り込み
Expiration	0x8	R/W	満了値
Counter	0xC	R/W	カウンタ

offset	31 24	23 16	15 8	7 0
0x00	-			P S
0x04	-			I
0x08	EXPR (HIGH)			
0x0c	EXPR (LOW)			
0x10	COUNT (HIGH)			
0x14	COUNT (LOW)			

22.2.2 ビットマップ

Control : コントロール

Offset: 0x00

31 2	1	0
Reserved	P	S

Field Name	Function
P	この bit が 0 の場合タイマはワンショットタイマとして動作し，この bit が 1 の場合タイマはピリオディックタイマとして動作する.
S	この bit に 1 がセットされた場合タイマが動作する.

Interrupt : 割り込み

Offset: 0x04

31 1	0
Reserved	I

Field Name	Function
Interrupt (I)	タイマが満了すると自動的にセットされる．この bit が 1 の場合割り込みが発生する.

Expiration : 満了値

Offset: 0x08, 0x0c

31 0
EXPR

Field Name	Function
Expiration (EXPR)	カウンタがこの値に達した場合に割り込みを発生させる.

Counter : カウンタ

Offset: 0x10, 0x14

31	0
COUNTER	

Field Name	Function
Counter (COUNTER)	カウンタが満了値に達した場合に割り込みを発生させる.

23

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

23.1 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

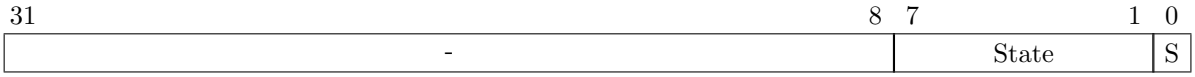
23.1.1 主記憶 I/F 幅設定レジスタ

Offset: 0x20

Field Name	Function
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

23.1.2 I/F 起動レジスタ

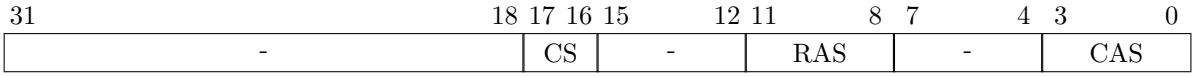
Offset: 0x0



Field Name	Function
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

23.1.3 メモリモジュール設定レジスタ

Offset: 0x4



Field Name	Function
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.

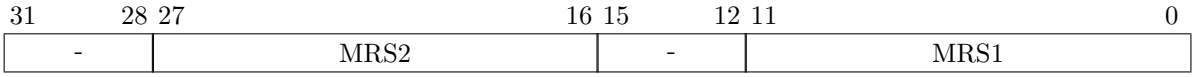
23.1.4 EMRS 設定レジスタ

Offset: 0x8

Field Name	Function
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．

23.1.5 MRS 設定レジスタ

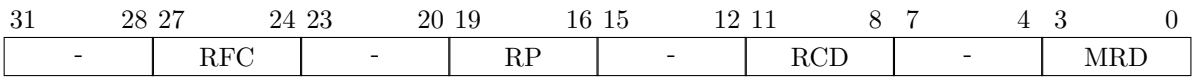
Offset: 0xC



Field Name	Function
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．

23.1.6 DDR 設定レジスタ 1

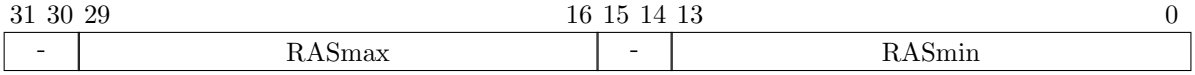
Offset: 0x14



Field Name	Function
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．

23.1.7 DDR 設定レジスタ 2

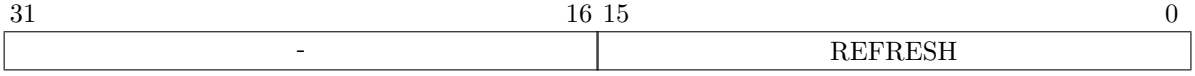
Offset: 0x18



Field Name	Function
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．

23.1.8 リフレッシュインターバル設定レジスタ

Offset: 0x18



Field Name	Function
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．

23.2 ECC 制御レジスタマップ

DDR SDRAM I/F の ECC 機能を制御する．SRMTP では 128bit I/F に置いて, 128bit のデータに 16bit の ReedSolomon 符号を付与し, 1Biyte のブロックエラー訂正及び 2byte のブロックエラーの検知が可能となっている.

ベースアドレス	FFFFF400

offset	31 24	23 16	15 8	7 0
0x0	-			E
0x4	-			C F
0x8	-			Next Error Ptr
0xc	Current Error Address			-

23.2.1 ECC 設定レジスタ

Offset: 0x00

31 1	0
-	E

Field Name	Function
E	ECC Enable : Default 0 本ビットでは ECC の on/off を設定する． 0: off 1: on

23.2.2 Fatal/Correct レジスタ

Offset: 0x04

31 2	1	0
-	C	F

Field Name	Function
C	Correct : 本ビットが 1 の場合，エラーが発生したがエラー訂正に成功したことを示す.
F	Fatal : 本ビットが 1 の場合，訂正できないエラーが発生したことを示す.

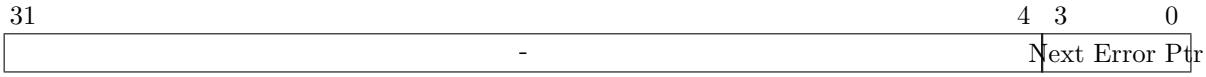
23.2.3 カレントエラーアドレスレジスタ

Offset: 0x08

Field Name	Function
Current Error Address	エラーが発生した最新のアドレスを格納する．アドレスは 8word アラインで保持される．

23.2.4 ネクストエラーアドレスポインタレジスタ

Offset: 0x0C

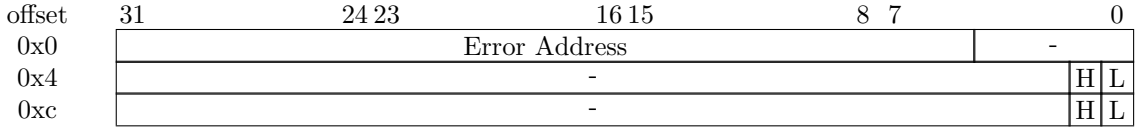


Field Name	Function
Next Error Ptr	次にエラーが発生した場合にエラーアドレスを格納するエラーアドレスバッファの番号を示す． 本レジスタの値 - 1 が最新のエラーアドレスを格納するエラーアドレスバッファである．

23.3 エラーアドレスバッファ

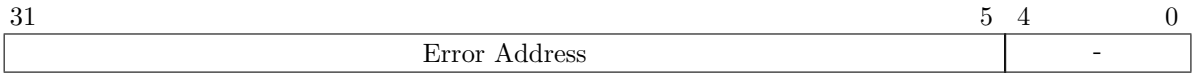
エラーの発生したアドレスを保持するリングバッファ．最大 8 つのエラー情報を保持する．

ベースアドレス	FFFFF500 - FFFFF570



23.3.1 エラーアドレスバッファレジスタ

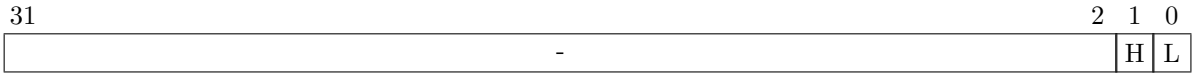
Offset: 0x0



Field Name	Function
Error Address	エラーが発生したアドレスを格納する．アドレスは 8word アラインで保持される.

23.3.2 訂正可能エラーバッファレジスタ

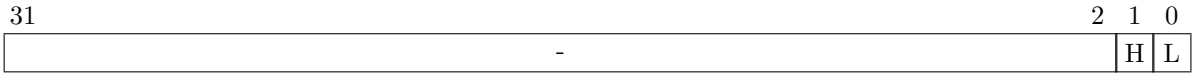
Offset: 0x0



Field Name	Function
H	High: 本ビットが 1 の場合, 上位 128bit でエラー訂正に成功したことを示す.
L	Low: 本ビットが 1 の場合, 下位 128bit でエラー訂正に成功したことを示す.

23.3.3 訂正不可能エラーバッファレジスタ

Offset: 0x0



Field Name	Function
H	High: 本ビットが 1 の場合, 上位 128bit で訂正できないエラーが発生したことを示す.
L	Low: 本ビットが 1 の場合, 下位 128bit で訂正できないエラーが発生したことを示す.

24

SRAMコントローラ

24.1 概要

SRAM の機能を制御する．ECC 機能の on/off やエラー発生状況などを確認することが可能である．

24.2 SRAM コントローラレジスタマップ

ベースアドレス	ffff7c00

offset	31 24	23 16	15 8	7 0
0x0	-			E
0x4	-			C F
0x8	-			Next Error Ptr
0xc	-	Current Error Address		-

24.2.1 ECC 設定レジスタ

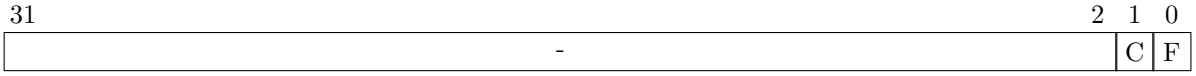
Offset: 0x00

31 1	0
-	E

Field Name	Function
E	ECC Enable : Default 0 本ビットでは SRAM の ECC の on/off を設定する. 0: off 1: on

24.2.2 Fatal/Correct レジスタ

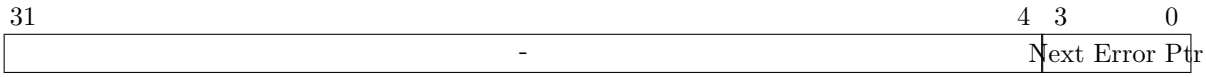
Offset: 0x04



Field Name	Function
C	Correct : 本ビットが 1 の場合，エラーが発生したがエラー訂正に成功したことを示す.
F	Fatal : 本ビットが 1 の場合，訂正できないエラーが発生したことを示す.

24.2.3 ネクストエラーアドレスポインタレジスタ

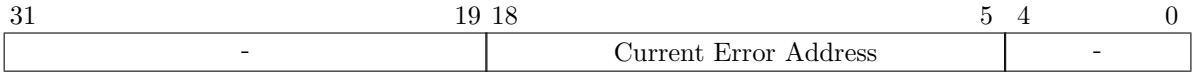
Offset: 0x08



Field Name	Function
Next Error Ptr	次にエラーが発生した場合にエラーアドレスを格納するエラーアドレスバッファの番号を示す. 本レジスタの値 - 1 が最新のエラーアドレスを格納するエラーアドレスバッファである.

24.2.4 カレントエラーアドレスレジスタ

Offset: 0x0c



Field Name	Function
Current Error Address	エラーが発生した最新のアドレスを格納する．SRAM 内のオフセットを示すため実際のアドレスはこれに SRAM のベースアドレス (0x98000000) を足したものとなる。

24.3 エラーアドレスバッファ

エラーの発生したアドレスを保持するリングバッファ. 最大 8 つのエラー情報を保持する.

ベースアドレス	FFFFFD00 - FFFFFD70

offset	31 24	23 16	15 8	7 0
0x0	Error Address			-
0x4	-			Fatal error map
0xc	-			Correct error map

24.3.1 エラーアドレスバッファレジスタ

Offset: 0x0

31 5	4 0
Error Address	-

Field Name	Function
Error Address	エラーが発生したアドレスを格納する. アドレスは 8word アラインで保持される.

24.3.2 訂正不可エラーバッファレジスタ

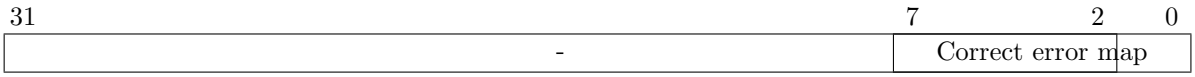
Offset: 0x4

31 8	7 0
-	Fatal error map

Field Name	Function
Fatal error map	上記のエラーアドレスで示された 8word 中, 訂正不可エラーの発生した箇所を示す.

24.3.3 訂正可能エラーバッファレジスタ

Offset: 0x8



Field Name	Function
Correct error map	
上記のエラーアドレスで示された 8word 中, 訂正可能エラーの発生した箇所を示す.	

25

Flash I/F

25.1 概要

Flash I/F. フラッシュからのブートとフラッシュへのアクセスについて.

25.2 アドレス空間

Flash と ROM は out_cs_toggle_信号によりアドレス空間が変動する.

out_cs_toggle_信号が High のときは ROM のアドレス空間は EXT_0(0x00000000 〜 0x00ffffff)，Flash のアドレス空間は EXT_1(0x40000000 〜 0x4fffffff) となる．out_cs_toggle_信号が Low のときは ROM のアドレス空間が EXT_1 に，Flash のアドレス空間が EXT_0 になる.

Flash からブートさせる場合は out_cs_toggle_信号を Low にする.

Flash を EXT_1 に接続する場合，システムレジスタ 0x8f の 13bit 目を 1 にセットする必要がある (Default 設定のままでよい)．それ以外の I/O を接続する場合は 0 に設定しなければならない.

25.3 アクセス

フラッシュに対してソフトウェアでコマンド発行することにより，読み書きを行う．フラッシュは M29W128G が 32 ビットバスに 2 つ接続されている．32 ビット単位，16 ビット単位，8 ビット単位でのアクセスが可能．デフォルトでは 16bit モードで接続されており，後述の設定レジスタにより 8bit モードへの変更が可能．コマンドの詳細は M29W128G のマニュアルを参照.

通常書き込み (ワードアクセス，バイトアクセス) に関してのみ，後述の設定レジスタ (Auto Write Enable) を利用することでハードウェアにより書き込みコマンドを発行する．その場合はデータ書き込み前に必要となるコマンドの発行は不要となり，直接アドレスを指定して書き込むことができる.

25.4 Flash I/F の設定レジスタ

Flash I/F の設定レジスタのアドレスマップは，EXT_8(0x27000000 〜 0x27ffffff) に割り当てられている.

EXT_8 は I/O インターフェースの設定レジスタ専用のアドレス空間であり，外部への制御信号は存在しない.

Auto Write Enable offset: 0x00

31	1	0
Reserved		EN

Field Name	Range	Description
EN	0	Default : 0 bit が 1 の場合は，通常ソフトウェアから行われる Write コマンドの発行をハードウェアにより行う．Flash を含む DMA 転送を行う場合は，この bit を必ず 1 にセットしなければならない．

Byte Mode offset: 0x04

31	1	0
Reserved		Byte

Field Name	Range	Description
Byte	0	Default : 1 バイト単位での書き込みを行う場合は 0 にセットしなければならない． それ以外 (16, 32bit 書き込み) の場合は 1 にセットしなければならない．

26

Universal Asynchronous Receiver/Transmitter

Initial Address: Channel0:0xffff6000 + 0x80 * CH (0 - 3)

26.1 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

26.1.1 Receiver Buffer (RB) / Transmitter Holding Register (THR)

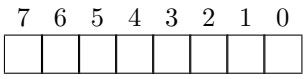
Offset: 0x0000

7 0

Field Name	Function
7-0	送信 FIFO の入力および受信 FIFO の出力.

26.1.2 Interrupt Enable Register (IER)

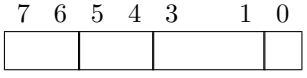
Offset: 0x0004



Field Name	Function
0	Received Data availble interrupt. (Buffer trigger) ‘ 0 ’ - Disabled. ‘ 1 ’ - Enabled.
1	Received Data availble interrupt. (Time out) ‘ 0 ’ - Disabled. ‘ 1 ’ - Enabled.
2	Transmitter Holding Register empty interrupt. ‘ 0 ’ - Disabled. ‘ 1 ’ - Enabled.
3	Receiver Line Status Interrupt. ‘ 0 ’ - Disabled. ‘ 1 ’ - Enabled.
4	Modem Status Interrupt. ‘ 0 ’ - Disabled. ‘ 1 ’ - Enabled.
5	Create DMA-request when recieved data available. (UART 0,1 only) ‘ 0 ’ - Disabled. ‘ 1 ’ - Enabled.
6	Create DMA-request when transmitter holding register empty. (UART 0,1 only) ‘ 0 ’ - Disabled. ‘ 1 ’ - Enabled.
7	Reserved. Should be logic ‘ 0 ’.

26.1.3 Interrupt Identification Register (IIR)

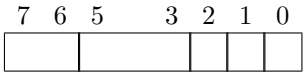
Offset: 0x0008



<table>
<tr><td>Field Name</td><td>Function</td></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>

26.1.4 FIFO Control Register (FCR)

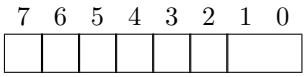
Offset: 0x0008



Field Name	Function
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

26.1.5 Line Control Register (LCR)

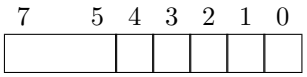
Offset: 0x000c



Field Name	Function
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. ‘1’ - 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.

26.1.6 Modem Control Register (MCR)

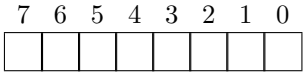
Offset: 0x0010



Field Name	Function
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.

26.1.7 Line Status Register (LSR)

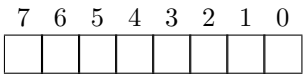
Offset: 0x0014



Field Name	Function
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.

26.1.8 Modem Status Register (MSR)

Offset: 0x0018

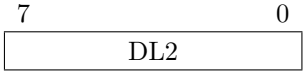


Field Name	Function
0	Delta Clear To Send (DCTS) indicator. ‘1’ - The CTS line has changed its state.
1	Delta Data Set Ready (DDSR) indicator. ‘1’ - The DSR line has changed its state.
2	Trailing Edge of Ring Indicator (TERI) detector. The RI line has changed its state from low to high state.
3	Delta 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.

26.1.9 Divisor Latches (DL)

Offset: 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.



Field Name	Function
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.

26.2 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.

26.2.1 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.

27

Serial Peripheral Interface Unit

27.1 Outline

Serial Peripheral Interface Unit はクロック同期式のシリアルインターフェースであり，SPI の規格に準拠した周辺機器を制御する．本ユニットは 4 つのスレーブに対応するが，M-RMTP ではチップセレクト (cs_) が 3 本しか外に出ていないため，スレーブ 3 の設定は意味を持たない．

27.2 Interface

27.2.1 Address Format

Serial Peripheral Interface Unit の初期ベースアドレスは 0xffffb000 である．SPI Unit は 2 チャンネル実装されており，それぞれのベースアドレスは以下のようになっている．

0xffffb000 : チャネル 0
0xffffb040 : チャネル 1

SPI Unit の制御レジスタのアドレスは次のようになる．

Field Name	Range	Description
Offset	5:0	設定する項目を指定する．

27.2.2 Control Register

Serial Peripheral Interface Unit の制御を行う場合，以下に示す Offset をアドレスの Offset に指定することにより，該当設定レジスタにアクセスする．

Slave Control offset: 0x00

Slave の設定を行う．

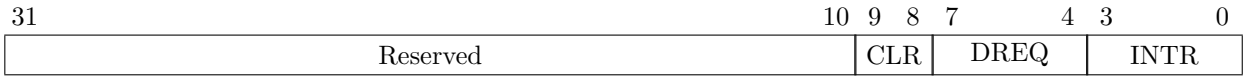
31 … 5	4 … 1	0
Reserved	SS_	A

Field Name	Range	Description
Auto (A)	0	1 に設定すると FIFO に最初に書き込まれたデータを繰り返し転送する．0 を設定すると，マニュアルで SPI の転送を行う．
Slave Select_ (SS_)	4:1	Auto bit が 0 の場合は，アクセスする Slave を指定する．複数の bit が 0 の場合，下位の bit が優先される．Auto bit が 1 の場合は，自動でアクセス Slave を指定する．複数の bit が 0 の場合，下位の bit から順番にアクセスを行う．

FIFO Control

offset: 0x04

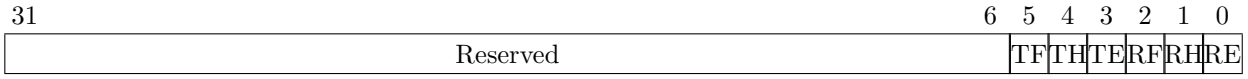
FIFO の設定を行う．



Field Name	Range	Description
Interrupt (INTR)	3:0	1 を設定した場合，要因により割り込みを発生させる． 0: 受信 FIFO にデータが半分たまると割り込みを発生させる． 1: 受信 FIFO がいっぱいになると割り込みを発生させる 2: 送信 FIFO のデータが半分未満になると割り込みを発生させる． 3: 送信 FIFO が空になると割り込みを発生させる．
DMA Request (DREQ)	7:4	1 を設定した場合，要因により DMA Request を発生させる． 4: 受信 FIFO にデータが半分たまると DMA Request を発生させる． 5: 受信 FIFO がいっぱいになると DMA Request を発生させる． 6: 送信 FIFO のデータが半分より少くなると DMA Request を発生させる． 7: 送信 FIFO がからになると DMA Request を発生させる．
Clear (CLR)	9:8	1 にすると FIFO をクリアする．この bit は自動的に 0 になる． 8: 受信 FIFO をクリアする． 9: 送信 FIFO をクリアする．

FIFO Status

offset: 0x08 (Read Only)



Field Name	Range	Description
Rx Empty (RE)	0	受信 FIFO が空の場合 1 になる．
Rx Half (RH)	1	受信 FIFO に半分以上データがたまっている場合 1 になる．
Rx Full (RF)	2	受信 FIFO がいっぱいの場合 1 になる．
Tx Empty (RE)	3	送信 FIFO が空の場合 1 になる．
Tx Half (RH)	4	送信 FIFO に半分以上データがたまっている場合 1 になる．
Tx Full (RF)	5	送信 FIFO がいっぱいの場合 1 になる．

FIFO

offset: 0x0c

31	0
FIFO	

Field Name	Range	Description
FIFO	31:0	書き込みの場合は送信 FIFO に値が書き込まれる．Slave Control レジスタの Auto bit が 0 の場合，値を書き込むことにより，データの転送を開始する．読み込みの場合は受信 FIFO から値が読み出される．

Interrupt

offset: 0x10

割り込みの要因を示す．1 を書き込むことにより，その bit をクリアする．

31	4	3	2	1	0
Reserved		TE	TH	RF	RH

Field Name	Range	Description
Rx Half (RH)	0	受信 FIFO に半分以上データがたまっている．
Rx Full (RF)	1	受信 FIFO がいっぱいである．
Tx Half (RH)	2	送信 FIFO のデータが半分未満になった．
Tx Empty (RE)	3	送信 FIFO が空である．

Interval

offset: 0x14

31	0
Interval	

Field Name	Range	Description
Interval	31:0	Slave Control レジスタの Auto bit が 1 の場合，Slave に対する一連のアクセスが終わった後の待ち時間を指定する．

Mode0 offset: 0x20

Slave Select0 用の設定を行なう．

31	30	29	28 ... 24	23	22	21 ... 0
W_	R_	L	Size	OL	HA	Clock Ratio

Field Name	Range	Description
Clock Ratio	21:0	同期クロックで出力するクロックの分周率を指定する．実際には指定した数値 × 2 で SPI の内部クロックを分周し，出力する．0 を指定した場合は $2^{22} \times 2$ 分周される．デフォルトは 0.
HA, OL	23:22	SPI の動作モードを指定する． 0x0　同期クロックは正極性．立ち上りでデータを受け取る． 0x1　同期クロックは正極性．立ち下りでデータを受け取る． 0x2　同期クロックは負極性．立ち下りでデータを受け取る． 0x3　同期クロックは負極性．立ち上りでデータを受け取る．
Size	28:24	データの転送サイズ．指定した値 + 1 bit を転送する．
LSB (L)	29	1 を指定すると LSB から転送を開始する．0 の場合は MSB から転送する．
Read Enable (R_)	30	0 を指定すると外部からの入力を読み込み，受信 FIFO に値を格納する．1 の場合は外部からのデータを読み込まない．
Write Enable (W_)	31	0 を指定すると送信 FIFO のデータを外部に転送する．1 の場合はデータを送信しない．

Mode1 offset: 0x24

Slave Select1 用の設定を行なう．

31	30	29	28 ... 24	23	22	21 ... 0
W_	R_	L	Size	OL	HA	Clock Ratio

Field Name	Range	Description
Clock Ratio	21:0	同期クロックで出力するクロックの分周率を指定する．実際には指定した数値 × 2 で SPI の内部クロックを分周し，出力する．0 を指定した場合は $2^{22} \times 2$ 分周される．デフォルトは 0.
HA, OL	23:22	SPI の動作モードを指定する． 0x0 同期クロックは正極性．立ち上りでデータを受け取る． 0x1 同期クロックは正極性．立ち下りでデータを受け取る． 0x2 同期クロックは負極性．立ち下りでデータを受け取る． 0x3 同期クロックは負極性．立ち上りでデータを受け取る．
Size	28:24	データの転送サイズ．指定した値 + 1 bit を転送する．
LSB (L)	29	1 を指定すると LSB から転送を開始する．0 の場合は MSB から転送する．
Read Enable (R_)	30	0 を指定すると外部からの入力を読み込み，受信 FIFO に値を格納する．1 の場合は外部からのデータを読み込まない．
Write Enable (W_)	31	0 を指定すると送信 FIFO のデータを外部に転送する．1 の場合はデータを送信しない．

Mode2 offset: 0x28

Slave Select2 用の設定を行なう．

31	30	29	28	24	23	22	21	0
W_	R_	L	Size		OL	HA	Clock Ratio	

Field Name	Range	Description
Clock Ratio	21:0	同期クロックで出力するクロックの分周率を指定する．実際には指定した数値 × 2 で SPI の内部クロックを分周し，出力する．0 を指定した場合は $2^{22} \times 2$ 分周される．デフォルトは 0.
HA, OL	23:22	SPI の動作モードを指定する． 0x0　同期クロックは正極性．立ち上りでデータを受け取る． 0x1　同期クロックは正極性．立ち下りでデータを受け取る． 0x2　同期クロックは負極性．立ち下りでデータを受け取る． 0x3　同期クロックは負極性．立ち上りでデータを受け取る．
Size	28:24	データの転送サイズ．指定した値 + 1 bit を転送する．
LSB (L)	29	1 を指定すると LSB から転送を開始する．0 の場合は MSB から転送する．
Read Enable (R_)	30	0 を指定すると外部からの入力を読み込み，受信 FIFO に値を格納する．1 の場合は外部からのデータを読み込まない．
Write Enable (W_)	31	0 を指定すると送信 FIFO のデータを外部に転送する．1 の場合はデータを送信しない．

Mode3 offset: 0x2c

Slave Select3 用の設定を行なう.

31	30	29	28 ... 24	23	22	21 ... 0
W_	R_	L	Size	OL	HA	Clock Ratio

Field Name	Range	Description
Clock Ratio	21:0	同期クロックで出力するクロックの分周率を指定する．実際には指定した数値 × 2 で SPI の内部クロックを分周し，出力する．0 を指定した場合は $2^{22} \times 2$ 分周される．デフォルトは 0.
HA, OL	23:22	SPI の動作モードを指定する. 0x0 同期クロックは正極性．立ち上りでデータを受け取る. 0x1 同期クロックは正極性．立ち下りでデータを受け取る. 0x2 同期クロックは負極性．立ち下りでデータを受け取る. 0x3 同期クロックは負極性．立ち上りでデータを受け取る.
Size	28:24	データの転送サイズ．指定した値 + 1 bit を転送する.
LSB (L)	29	1 を指定すると LSB から転送を開始する．0 の場合は MSB から転送する.
Read Enable (R_)	30	0 を指定すると外部からの入力を読み込み，受信 FIFO に値を格納する．1 の場合は外部からのデータを読み込まない.
Write Enable (W_)	31	0 を指定すると送信 FIFO のデータを外部に転送する．1 の場合はデータを送信しない.

Configuration offset: 0x30 (Read Only)

31 ... 2	1	0
Reserved		FS

Field Name	Range	Description
FIFO Size (FS)	1:0	FIFO のサイズを示す. 0x1 : 8 Entry 0x2 : 16 Entry 0x3 : 32 Entry

27.3 Operation

本 SPI Unit は 4 本の Slave Select を持ち，各 Slave に対して個別の設定を行うことができる．設定は各 Slave の Mode レジスタで行う．

本 SPI Unit は Slave へのアクセスの方法として，自動で継続的に Slave から値を読み込むモードと，1 つ 1 つ送受信を行うモードがある．

27.3.1 Manual Mode

1 つ 1 つ送受信を行う場合，Slave Control レジスタの Auto bit を 0 にし，アクセスする Slave を Slave Control レジスタの Slave Select_ bit で指定する．

値を送信したい場合は，各 Slave の Mode レジスタの W_ bit を 0 にする．送信したい値を FIFO に書き込むことにより，インターフェースから値が送信される．

値を受信したい場合は，各 Slave の Mode レジスタの R_ bit を 0 にする．FIFO に値を書き込むことによりインターフェスが動作し，Slave から値を受信する．受信した値は受信 FIFO に書き込まれる．

SPI は送受信を同時に行うことができる．各 Slave の Mode レジスタの W_ bit と R_ bit を両方 0 にすることにより，送受信を同時に行う．

27.3.2 Auto Mode

Auto Mode は Slave から自動で継続的に値を読み出す場合に使用する．Slave Control レジスタの Auto bit を 1 にすることにより Auto Mode として動作する．値を読み出す Slave は Slave Control レジスタの Slave Select_ bit で指定する．

各 Slave に対するアクセスは各 Slave の Mode レジスタの設定による．そのため，Mode レジスタの R_ bit を 0 にし，W_ bit を 1 にする必要がある．

Auto Mode では，SPI Unit は Slave Control レジスタの Slave Select_ bit が 0 になっている Slave に対して下位側 (0 番) から順番にアクセスを行う．指定された全ての Slave に対してアクセスを行った後，Interval レジスタで指定されているサイクル数だけ待った後，指定された Slave に対してアクセスを開始する．

28

Parallel I/O Unit

28.1 Outline

Parallel I/O Unit は 8 bit の入出力を提供する．

28.2 Interface

28.2.1 Address Format

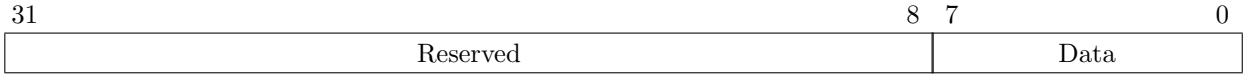
Parallel I/O Unit の初期ベースアドレスは 0xffffc000 である．Parallel I/O Unit の制御レジスタのアドレスは次のようになる．

Field Name	Range	Description
Offset	4:0	設定する項目を指定する．

28.2.2 Control Register

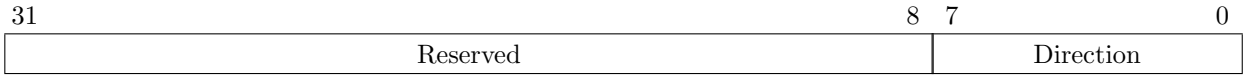
Parallel I/O Unit の制御を行う場合，以下に示す Offset をアドレスの Offset に指定することにより，該当設定レジスタにアクセスする．

Data offset: 0x00



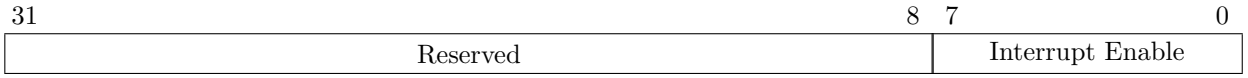
Field Name	Range	Description
Data	7:0	bit が入力の場合は，読むことにより外部からの入力を得る．bit が出力の場合は，書き込むことにより外部に出力を与える．

Direction offset: 0x04



Field Name	Range	Description
Direction	7:0	0 の場合は入力．1 の場合は出力となる．

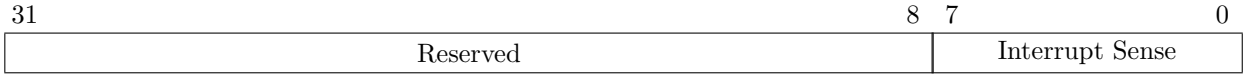
Interrupt Enable offset: 0x08



Field Name	Range	Description
Interrupt Enable	7:0	1 の場合，Data レジスタの対応する値が変化した時に割り込みを発生させる．割り込み発生条件は Interrupt Upedge レジスタ，Interrupt Downedge レジスタで設定する．

Interrupt Sense

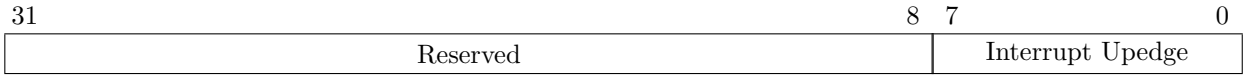
offset: 0x0c



Field Name	Range	Description
Interrupt Sense	7:0	割り込みの発生要因になった bit に 1 がセットされる．このレジスタに 1 を書き込むと，対応する bit がクリアされる．

Interrupt Upedge

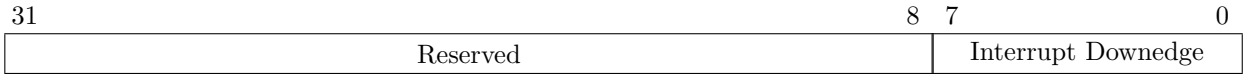
offset: 0x10



Field Name	Range	Description
Interrupt Upedge	7:0	1 の場合，データが 0 から 1 に変化した時に割り込みを発生させる．

Interrupt Downedge

offset: 0x14



Field Name	Range	Description
Interrupt Downedge	7:0	1 の場合，データが 1 から 0 に変化した時に割り込みを発生させる．

Configuration offset: 0x18 (Read Only)

31	2	1	0
Reserved			BW

Field Name	Range	Description
Bit Width (BW)	1:0	Parallel I/O の Bit Width を示す. 0x1 : 8 Bit 0x2 : 16 Bit 0x3 : 32 Bit

28.3 Operation

Parallel I/O は 8 bit の幅を持ち，bit ごとに入出力の方向を設定することができる．設定は Direction レジスタで行う．

入力の場合，Parallel I/O に入力されるクロックにより，I/O ピンのデータが Data レジスタにラッチされる．出力の場合，Data レジスタの値が I/O ピンに出力される．

各 bit は指定した条件により割り込みを発生させることができる．割り込みを発生させるためには，Interrupt Enable レジスタの対応する bit ををを 1 にセットする．また，割り込み発生条件により，Interrupt Upedge レジスタ，Interrupt Downedge レジスタの対応する bit を 1 にセットする．両方 1 にセットした場合，値が変化するたびに割り込みが発生する．

29

I2C Master Controller

29.1 Outline

I2C is a two-wire, bi-directional serial bus that provides a simple and efficient method of data exchange between devices. It is most suitable for applications requiring occasional communication over a short distance between many devices. The I2C standard is a true multi-master bus including collision detection and arbitration that prevents data corruption if two or more masters attempt to control the bus simultaneously.

29.2 Interface

29.2.1 Address Format

I2C Master Controller のベースアドレスは 0xffffc800 である．I2C Master Controller の制御レジスタのアドレスは次のようになる．

Field Name	Range	Description
Offset	4:0	設定する項目を指定する．

29.2.2 Control Register

I2C Master Controller の制御を行う場合，以下に示す Offset をアドレスの Offset に指定することにより，該当設定レジスタにアクセスする．

Clock Prescale (lo-byte) offset: 0x00

31	8	7	0
Reserved		scale	

Clock Prescale (hi-byte) offset: 0x04

31	8	7	0
Reserved		scale	

Field Name	Range	Description
scale	7:0	This register is used to prescale the SCL clock line. Due to the structure of the I2C interface, the core uses a 5*SCL clock internally. The prescale register must be programmed to this 5*SCL frequency (minus 1). Change the value of the prescale register only when the 'EN' bit is cleared.

Example: wb_clk_i = 32MHz, desired SCL = 100KHz

$$prescale = \frac{32MHz}{5 * 100KHz} - 1 = 63(dec) = 3F(hex)$$

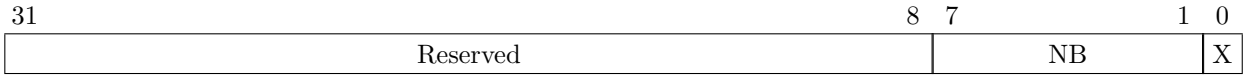
Control offset: 0x08

31	8	7	6	5	0
Reserved		EN	IEN	Reserved	

Field Name	Range	Description
I2C core enable bit (EN)	7	When set to '1', the core is enabled. When set to '0', the core is disable.
I2C core interrput enable bit (IEN)	6	When set to '1', interrupt is enabled. When set to '0', interrupt is disable.

Transmit

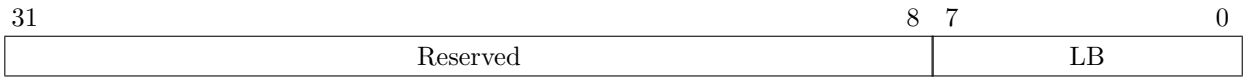
offset: 0x0c (Write Only)



Field Name	Range	Description
Next byte (NB)	7:0	Next byte to transmit via I2C.
X	0	In case of a data tranfer this bit represents the data's LSB. In case of a slave address transfer this bit represents the RW bit. '1' = reading from slave. '0' = writing to slave.

Receive

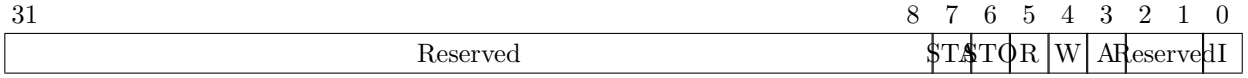
offset: 0x0c (Read Only)



Field Name	Range	Description
Last byte (LB)	7:0	Last byte received via I2C.

Command

offset: 0x10 (Write Only)



Field Name	Range	Description
Start (STA)	7	Generate (repeated) start condition.
Stop (STO)	6	Generate stop condition.
Read (R)	5	Read from slave.
Write (W)	4	Write to slave.
ACK (A)	3	When a receiver, sent ACK (ACK = '0') or NACK (ACK = '1').
Interrput ACK (I)	1	Interrupt acknowledge. When set, clears a pending interrupt.

The STA, STO, R, W, and I bits are cleared automatically. These bits are always read as zeros.

Status

offset: 0x10 (Read Only)

31 ... 8	7	6	5	4 ... 2	1	0
Reserved	R	B	A	Reserved	T	I

Field Name	Range	Description
Received acknowledge from slave (R)	7	This flag represents acknowledge from the addressed slave. '1' = No acknowledge received. '0' = Acknowledge received.
I2C bus busy (B)	6	'1' after START signal detect. '0' after STOP signal detect.
Arbitration lost (A)	5	This bit is set when the core lost arbitration. Arbitration is lost when: • a STOP signal is detected, but non requested. • The master drives SDA high, but SDA is low. See bus-arbitration section for more information.
Transfer in progress (T)	1	'1' when transferring data. '0' when transfer complete.
Interrupt Flag (I)	0	This bit is set when an interrupt is pending, which will cause a processor interrupt request if the IEN bit is set. The Interrupt Flag is set when: • one byte transfer has been completed. • arbitration is lost.

Transmit Register

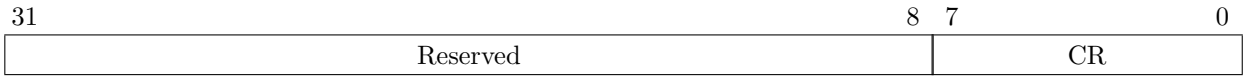
offset: 0x14 (Read Only)

アクセス極性が異なり、Transmit register(offset: 0x0c) と同じ値となる。

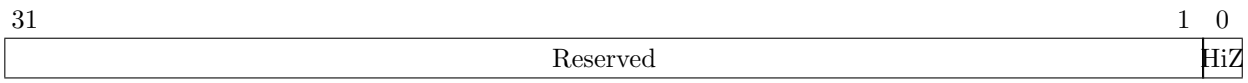
31 ... 8	7 ... 0
Reserved	TXR

Command Register offset: 0x18 (Read Only)

アクセス極性が異なり、Command register(offset: 0x10) と同じ値となる。



HiZ Status Register offset: 0x1C (Read Only)



29.3 Operation

29.3.1 System Configuration

I2C system uses a serial data line (SDA) and a serial clock line (SCL) for data transfers. All devices connected to these two signals must have open drain or open collector outputs. The logic AND function is exercised on both lines with external pull-up resistors.

Data is transferred between a Master and a Slave synchronously to SCL on the SDA line on a byte-by-byte basis. Each data byte is 8 bits long. There is one SCL clock pulse for each data bit with the MSB being transmitted first. An acknowledge bit follows each transferred byte. Each bit is sampled during the high period of SCL; therefore, the SDA line may be changed only during the low period of SCL and must be held stable during the high period of SCL. A transition on the SDA line while SCL is high is interpreted as a command (see START and STOP signals).

29.3.2 I2C Protocol

Normally, a standard communication consists of four parts:

1. START signal generation
2. Slave address transfer
3. Data transfer

4. STOP signal generation

START signal

When the bus is free/idle, meaning no master device is engaging the bus (both SCL and SDA lines are high), a master can initiate a transfer by sending a START signal. A START signal, usually referred to as the S-bit, is defined as a high-to-low transition of SDA while SCL is high. The START signal denotes the beginning of a new data transfer. A Repeated START is a START signal without first generating a STOP signal. The master uses this method to communicate with another slave or the same slave in a different transfer direction (e.g. from writing to a device to reading from a device) without releasing the bus.

The core generates a START signal when the STA-bit in the Command Register is set and the RD or WR bits are set. Depending on the current status of the SCL line, a START or Repeated START is generated.

Slave Address Transfer

The first byte of data transferred by the master immediately after the START signal is the slave address. This is a seven-bits calling address followed by a RW bit. The RW bit signals the slave the data transfer direction. No two slaves in the system can have the same address. Only the slave with an address that matches the one transmitted by the master will respond by returning an acknowledge bit by pulling the SDA low at the 9th SCL clock cycle.

Note: The core supports 10bit slave addresses by generating two address transfers. See the Philips I2C specifications for more details.

The core treats a Slave Address Transfer as any other write action. Store the slave device's address in the Transmit Register and set the WR bit. The core will then transfer the slave address on the bus.

Data Transfer

Once successful slave addressing has been achieved, the data transfer can proceed on a byte-by-byte basis in the direction specified by the RW bit sent by the master. Each transferred byte is followed by an acknowledge bit on the 9th SCL clock cycle. If the slave signals a No Acknowledge, the master can generate a STOP signal to abort the data transfer or generate a Repeated START signal and start a new transfer cycle.

If the master, as the receiving device, does not acknowledge the slave, the slave releases the SDA line for the master to generate a STOP or Repeated START signal.

To write data to a slave, store the data to be transmitted in the Transmit Register and set the WR bit. To read data from a slave, set the RD bit. During a transfer the core set the TIP flag, indicating that a Transfer is In Progress. When the transfer is done the TIP flag is reset, the IF flag set and, when enabled, an interrupt generated. The Receive Register contains valid data after the IF flag has been set. The user may issue a new write or read command when the TIP flag is reset.

STOP signal

The master can terminate the communication by generating a STOP signal. A STOP signal, usually referred to as the P-bit, is defined as a low-to-high transition of SDA while SCL is at logical '1'.

29.3.3 Arbitration Procudure

The I2C bus is a true multimaster bus that allows more than one master to be connected on it. If two or more masters simultaneously try to control the bus, a clock synchronization procedure determines the bus clock. Because of the wired-AND connection of the I2C signals a high to low transition affects all devices connected to the bus. Therefore a high to low transition on the SCL line causes all concerned devices to count off their low period. Once a device clock has gone low it will hold the SCL line in that state until the clock high state is reached. Due to the wired-AND connection the SCL line will therefore be held low by the device with the longest low period, and held high by the device with the shortest high period.

29.3.4 Clock Stretching

Slave devices can use the clock synchronization mechanism to slow down the transfer bit rate. After the master has driven SCL low, the slave can drive SCL low for the required period and then release it. If the slave's SCL low period is greater than the master's SCL low period, the resulting SCL bus signal low period is stretched, thus inserting wait-states.

30

外部バス

30.1 外部バス仕様

内部バスアクセスと同様．ビッグエンディアン．例として，0x00000000 番地は BE_[3], 0x00000001 番地は BE_[2], 0x00000002 番地は BE_[1], 0x00000003 番地は BE_[0] に対応．BMREQ_はバーストリクエスト信号．バーストのリクエストはバースト長で指定します．指定する値とバースト長の対応は以下の通りです．BMREQ == 11: 1Word, BMREQ == 10: 2Word, BMREQ == 01: 4Word, BMREQ == 00: 8Word．BMACK_ではバースト可能な長さをスレーブ側が出力する．

タイミングについて以下に示す．

Figure 30.1: Read

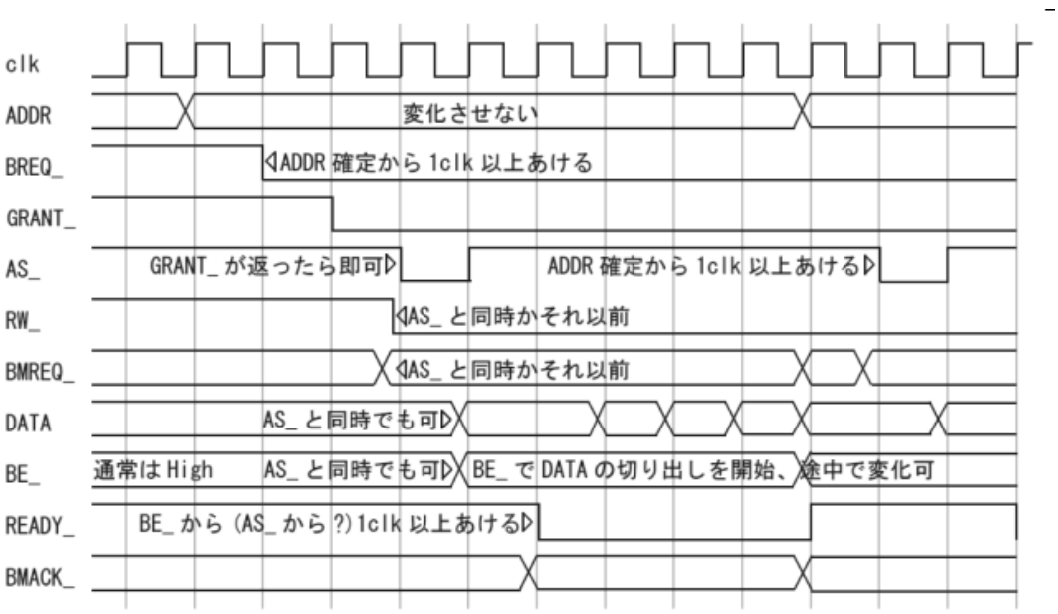


Figure 30.2: Write

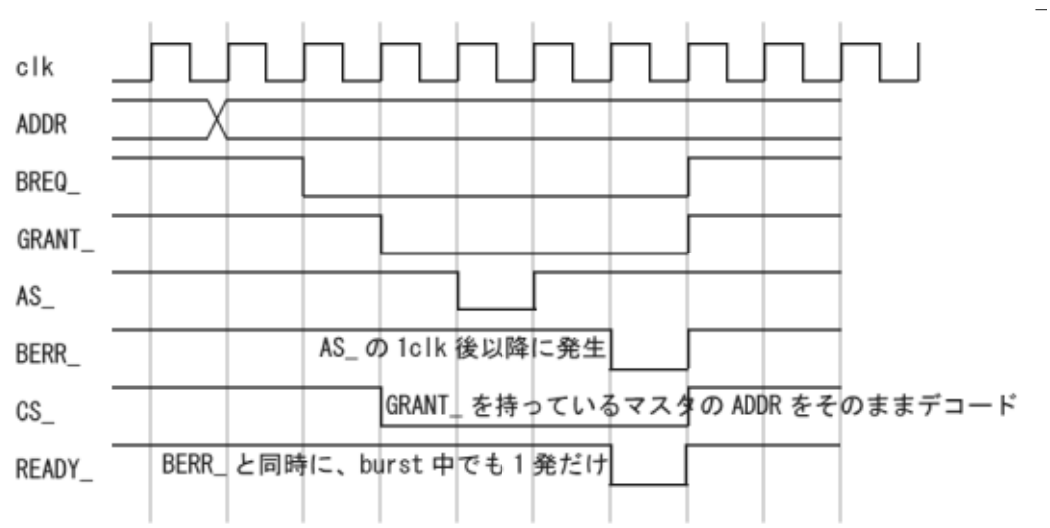


Figure 30.3: Error

30.2 EXT_0(ROM)

EXT_0(ROM) に 8bit, 16bit モードでアクセスする場合，be[0], be[1] が下位アドレスとして使用される．be[0] が LSB.

30.3 デフォルトメモリマップ（予定）

cs_0 0x00000000(ROM 専用)

cs_1 0x40000000(FlashIF)

cs_2 0x21000000(SiP FPGA)

cs_3 0x22000000(Board FPGA)

cs_4 0x23000000(REGFILE, LED)

cs_5 0x24000000

cs_6 0x25000000

cs_7 0x26000000

但し，cs_toggle を入れると，cs_0 と cs_1 が入れ替わる．また，auto ready が機能するのは cs_0 のみ.

31

Real Time Clock Unit

31.1 Outline

Real Time Clock Unit は外部クロックを用いてカレンダー機能を提供するユニットである．外部から供給されたクロックから 1 秒を計測し，各カウンタを制御する．カレンダー機能では年 (下位 2 ケタ)，月，日，曜日，時，分，秒を計測する．指定した日にち，時刻に割り込みを発生させるアラーム機能を持つ．また，うるう年に対応している．

さらに 1 秒単位で設定可能なタイマ機能を持つ．

クロックは外部ピン rtc_clk から入力する．デフォルトでは 32.768kHz を入力することにより 1 秒を計測するが，他の周波数であっても Clock Compare レジスタの値を変更することにより対応可能である．

外部ピン rtc_hold_をアサートすることにより，プロセッサ内部バスからの信号を受け付けなくなる．これにより，プロセッサの他の部分の電力をカットし，バスの信号が不定になった場合でも Real Time Clock Unit は外部からの信号に影響されずに，正しく時間を計測し続けることが可能となる．

31.2 Interface

31.2.1 Address Map

Second offset: 0x00

31	8	7	6	0
Reserved		U	Second	

Field Name	Range	Description
Update (U)	7	前回値をリードしてから値が更新された場合に 1 がセットされる．リードすることにより 0 にクリアされる．
Second	6:0	秒がセットされる．値は BCD コードで表す．

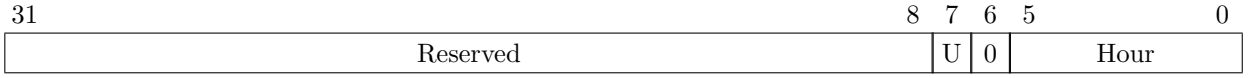
Minute offset: 0x04

31	8	7	6	0
Reserved		U	Minute	

Field Name	Range	Description
Update (U)	7	前回値をリードしてから値が更新された場合に 1 がセットされる．リードすることにより 0 にクリアされる．
Minute	6:0	分がセットされる．値は BCD コードで表す．

Hour

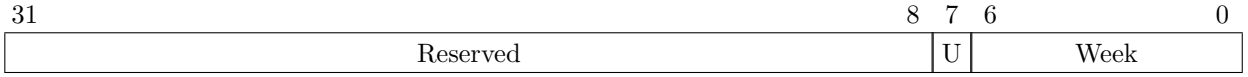
offset: 0x08



Field Name	Range	Description
Update (U)	7	前回値をリードしてから値が更新された場合に 1 がセットされる．リードすることにより 0 にクリアされる．
Hour	5:0	時がセットされる．値は BCD コードで表す．

Week

offset: 0x0c



Field Name	Range	Description
Update (U)	7	前回値をリードしてから値が更新された場合に 1 がセットされる．リードすることにより 0 にクリアされる．
Week	6:0	曜日がセットされる．値は 0 ビット目から日曜日 6 ビット目が土曜日を示す．

Day

offset: 0x10

31	8	7	6	5	0
Reserved		U	0	Day	

Field Name	Range	Description
Update (U)	7	前回値をリードしてから値が更新された場合に 1 がセットされる．リードすることにより 0 にクリアされる．
Day	5:0	日がセットされる．値は BCD コードで表す．

Month

offset: 0x14

31	8	7	6	5	4	0
Reserved		U	00		Month	

Field Name	Range	Description
Update (U)	7	前回値をリードしてから値が更新された場合に 1 がセットされる．リードすることにより 0 にクリアされる．
Month	4:0	月がセットされる．値は BCD コードで表す．

Year

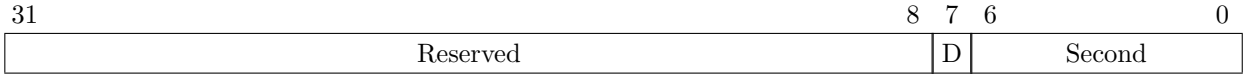
offset: 0x18

31	8	7	0
Reserved		Year	

Field Name	Range	Description
Year	7:0	年の下 2 けたがセットされる．値は BCD コードで表す．

Second Alarm

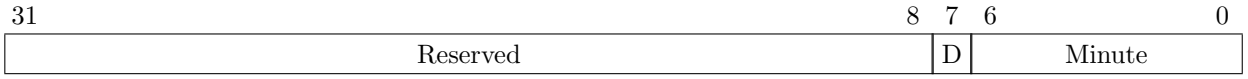
offset: 0x20



Field Name	Range	Description
Don't Care (D)	7	1 をセットすると，秒に関する条件を無視する．0 の場合，Second に指定した条件にマッチした場合にアラームの割り込みが発生する．
Second	6:0	アラームの秒を指定する．値は BCD コードで表す．

Minute Alarm

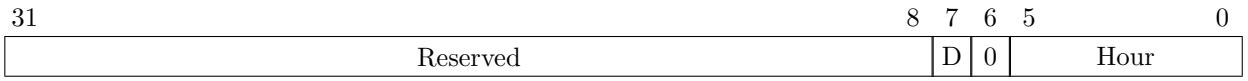
offset: 0x24



Field Name	Range	Description
Don't Care (D)	7	1 をセットすると，分に関する条件を無視する．0 の場合，Minute に指定した条件にマッチした場合にアラームの割り込みが発生する．
Minute	6:0	アラームの分を指定する．値は BCD コードで表す．

Hour Alarm

offset: 0x28



Field Name	Range	Description
Don't Care (D)	7	1 をセットすると，時に関する条件を無視する．0 の場合，Hour に指定した条件にマッチした場合にアラームの割り込みが発生する．
Hour	5:0	アラームの時を指定する．値は BCD コードで表す．

Week Alarm offset: 0x2c

31 … 8	7	6 … 0
Reserved	D	Week

Field Name	Range	Description
Don't Care (D)	7	1 をセットすると，曜日に関する条件を無視する．0 の場合，Week に指定した条件にマッチした場合にアラームの割り込みが発生する．
Week	6:0	アラームの曜日を指定する．1 を設定した曜日にアラーム割り込みを発生させる．

Day Alarm offset: 0x30

31 … 8	7	6	5 … 0
Reserved	D	0	Day

Field Name	Range	Description
Don't Care (D)	7	1 をセットすると，日に関する条件を無視する．0 の場合，Day に指定した条件にマッチした場合にアラームの割り込みが発生する．
Day	5:0	アラームの日を指定する．値は BCD コードで表す．

Month Alarm

offset: 0x34

31	8	7	6	5	4	0
Reserved		D	00		Month	

Field Name	Range	Description
Don't Care (D)	7	1 をセットすると，月に関する条件を無視する．0 の場合，Month に指定した条件にマッチした場合にアラームの割り込みが発生する．
Month	4:0	アラームの月を指定する．値は BCD コードで表す．

Time

offset: 0x38 (Read Only)

31	24	23	16	15	8	7	0
Reserved		Hour		Minute		Second	

Field Name	Range	Description
Hour	23:16	Hour レジスタの内容．
Minute	15:8	Minute レジスタの内容．
Second	7:0	Second レジスタの内容．

Date

offset: 0x3c (Read Only)

31 24	23 16	15 8	7 0
Week	Year	Month	Day

Field Name	Range	Description
Week	31:24	Week レジスタの内容.
Year	23:16	Year レジスタの内容.
Month	15:8	Month レジスタの内容.
Day	7:0	Day レジスタの内容.

Mode

offset: 0x40

31 5	4	3	2	1	0
Reserved	TM	PT	TE	AE	EN

Field Name	Range	Description
Test Mode (TM)	4	テストモード. 0 に設定すること.
Periodic Timer (PT)	3	1 を設定すると Periodic Timer，0 を設定すると One Shot Timer になる.
Timer Enable (TE)	2	1 をセットするとタイマが作動する. タイマが Expire した場合，One Shot Timer ならば自動的に 0 にクリアされる.
Alarm Enable (AE)	1	1 をセットするとアラームが動作する. アラームに指定した時間がくると割り込みが発生する.
Enable (EN)	0	1 をセットすると Real Time Clock が動作する.

Sense offset: 0x44

31	2	1	0
Reserved		TI	AI

Field Name	Range	Description
Timer Interrupt (TI)	1	タイマ割り込みが発生した場合に 1 がセットされる．1 を書き込むことによりクリアされる．
Alarm Interrupt (AI)	0	アラーム割り込みが発生した場合に 1 がセットされる．1 を書き込むことによりクリアされる．

Timer Compare offset: 0x48

31	0
Timer Compare	

Field Name	Range	Description
Timer Compare	31:0	タイマ割り込みをかける秒数を指定する．

Timer Count offset: 0x4c (Read Only)

31	0
Timer Count	

Field Name	Range	Description
Timer Count	31:0	現在のタイマのカウント数．この値が Timer Setup になるとタイマ割り込みが発生する．

Clock Compare

offset: 0x50

31	0
Clock Compare	

Field Name	Range	Description
Clock Compare	31:0	本モジュールに入力されているクロックの周波数を指定する．Clock Count がこのレジスタの値と等しくなった場合に 1 秒経過したと判定される．

Clock Count

offset: 0x54 (Read Only)

31	0
Clock Count	

Field Name	Range	Description
Clock Count	31:0	クロック毎にカウントアップされ，1 秒を計測する．このレジスタの値が Clock Compare と等しくなった場合，1 秒経過したと判定される．

32

Trace Buffer

32.1 概要

Trace Buffer は，プロセッサが実行した命令，レジスタ，発生した例外を記録する．

32.2 アドレスマップ

Trace Buffer の初期ベースアドレスは 0x10000000 である．

0x1000_0000∼0x1000_00FF	:	制御レジスタ領域
0x1004_0000∼0x1004_7FFF	:	Trace PC Buffer 領域 0
0x1004_8000∼0x1004_FFFF	:	Trace PC Buffer 領域 1
0x100C_0000∼0x100C_7FFF	:	Trace Exception Buffer 領域 0
0x100C_8000∼0x100C_FFFF	:	Trace Exception Buffer 領域 1

32.3 制御レジスタ領域

Trace PC Control　offset: 0x20

命令トレースの制御を行う．

31 … 1	0
Reserved	E

Field Name	Range	Description
Enable (E)	0	本ビットを 1 にすると，プログラムカウンタのトレースが有効になる．

Trace Exception Control offset: 0x60, 0x64

例外トレースの制御を行う.

31	4	3 1	0
Reserved		TH	E

Field Name	Range	Description
Thread (TH)	3:1	本フィールドに設定したスレッド番号の例外をトレースする.
Enable (E)	0	本ビットを 1 にすると，レジスタファイルのトレースが有効になる.

Trace Buffer Initialize offset: 0x80

トレースバッファの初期化..

31	0
Initialize	

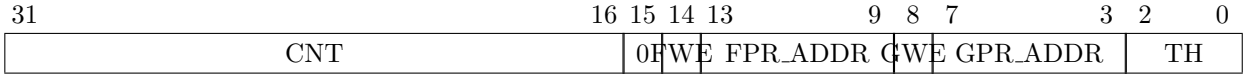
Field Name	Range	Description
Initialize	0	本レジスタに書き込みを行うと，トレースバッファの初期化を行う.

32.4 Trace PC Buffer 領域

SRTMP では 2 スレッドまで同時に命令トレースを行うことが可能となっている．格納されるトレース情報のフォーマットは以下の通り.

Trace PC Buffer 0

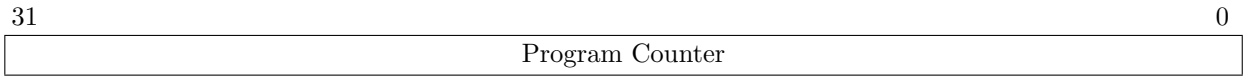
offset: 0x0



Field Name	Range	Description
Counter (CNT)	31:16	通し番号.
FPR Write Enable(FWE)	14	浮動小数点レジスタ書き込み有効．実行した命令で浮動小数点レジスタに書き込みを行った場合，このビットがセットされる.
FPR Address(FPR_ADDR)	13:9	浮動小数点レジスタ番号．実行した命令で浮動小数点レジスタに書き込みを行った場合，書き込んだレジスタ番号がセットされる.
GPR Write Enable(GWE)	8	汎用レジスタ書き込み有効．実行した命令で汎用レジスタに書き込みを行った場合，このビットがセットされる.
GPR Address(GPR_ADDR)	7:3	汎用レジスタ番号．実行した命令で汎用レジスタに書き込みを行った場合，書き込んだレジスタ番号がセットされる.
Thread(TH)	2:0	スレッド番号．命令を実行したスレッドの番号が格納される.

Trace PC Buffer 1

offset: 0x4



Field Name	Range	Description
Program Counter	31:0	実行した命令のプログラムカウンタが格納される.

Trace PC Buffer 2 offset: 0x8

31	0
Register Write Data 0	

Field Name	Range	Description
Register Write Data 0	31:0	汎用レジスタに書き込みが行われた場合，そのデータが格納される．浮動小数点レジスタに書き込みが行われた場合，そのデータの上位 32 ビットが格納される．いずれのレジスタへの書き込みが無い場合，このフィールドは省略される．

Trace PC Buffer 3 offset: 0xC

31	0
Register Write Data 1	

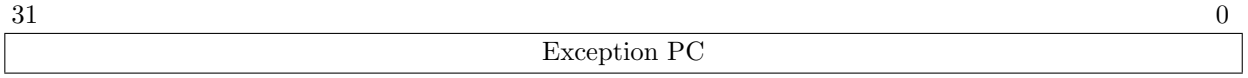
Field Name	Range	Description
Register Write Data 1	31:0	浮動小数点レジスタに書き込みが行われた場合，そのデータの下位 32 ビットが格納される．浮動小数点レジスタへの書き込みが無い場合，このフィールドは省略される．

32.5 Trace Exception Buffer 領域

SRTMP では 2 スレッドまで同時に例外トレースを行うことが可能となっている．格納されるトレース情報のフォーマットは以下の通り．

Trace Exception Buffer

offset: 0x0



Field Name	Range	Description
Exception PC	31:0	例外の発生したプログラムカウンタが格納される.

33

On-Chip Emulator

33.1 Outline

On-Chip Emulator は SPI 通信でのシングルワードのリード・ライトのエミュレーター機能を提供する．

33.2 Operation

On-Chip Emulator は適当な SPI マスターと接続して使用する．使用するデータ長は 32bit である．SPI の動作モードは，同期クロックは正極性で立ち上がりでデータを受け取り，MSB から転送を開始するように設定する．On-Chip Emulator は RELOAD ピンを High にすることでマスターにバスアクセスの完了を通知する．このピンは一度 High になると次の有効なコマンドが発行されるまで Low にならない．SPI 通信でコマンドを受信することによって，初期状態からライト，リードそれぞれのモードに遷移する．状態遷移は On-Chip Emulator に接続された Slave Select の Enable によって行われる．

33.2.1 Single Write

シングルライト転送は以下のように行う．Master は SPI のマスター，Slave は SPI のスレーブ (On-Chip Emulator が動作する側) である．

1. Master: SPI で On-Chip Emulator に 32bit のコマンドデータ 0xAB を入力する．
2. Master: SPI で On-Chip Emulator にアドレスデータを 32bit 入力する．
3. Master: SPI で On-Chip Emulator にライトデータを 32bit 入力する．
4. Slave: On-Chip Emulator がシングルライト転送を開始する．
5. Slave: シングルライトが完了．
6. Slave: On-Chip Emulator の RELOAD ピンを High にし，初期状態に遷移する．

33.2.2 Single Read

シングルリード転送は以下のように行う．Master は SPI のマスター，Slave は SPI のスレーブ (On-Chip Emulator が動作する側) である．

1. Master: SPI で On-Chip Emulator に 32bit のコマンドデータ 0xAA を入力する．
2. Master: SPI で On-Chip Emulator にアドレスデータを 32bit 入力する．
3. Slave: On-Chip Emulator がシングルリード転送を開始する．
4. Slave: リードが完了すると RELOAD ピンを High にする．
5. Master: Slave Select を Enable にする．
6. Slave: On-Chip Emulator から SPI マスターへ結果を送信し，RELOAD ピンが Low になる．
7. Master: 再度 Slave Select を Enable にする． この際に受信されるデータは意味を持たないため，ダミーとして処理する必要がある点に注意する．
8. Slave: 初期状態に遷移する．

34

Update History

Revision	Date	Description
1	2016/10/01	Publication the 1st version.
2	2019/07/15	Fixed the content according to the specifications.
3	2019/09/25	Update abstract.
4	2019/12/26	Fixed figure and pin assignments. Added calculation unit latency to abstract.
5	2020/1/18	Update abstract.
6	2020/9/28	Fixed specification about Flash I/F.