Space *Responsive Multithreaded Processor* Version 1 Specification

The 5th Edition.

January 8, 2021

Yamasaki Lab.,

Department of Information and Computer Science,
Keio University

— NY0030 —

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

Contents

1

Abstract

1.1 Overview

*RMT Processor*is a system LSI, designed for simultaneous operation of real-time communication, processing and control to implement Distributed real-time system. System construction by Combining common platform with basic functions for real-time Communication make it easier and more effective to implement of Distributed real-time system. *RMT Processor*includes following functions on a chip (System-on-a-chip), which can be embedded in various systems as a platform. Graph 1.1 shows a general block diagram.

- Real-time processing function (*RMT Processing Unit*)
- Real-time Communication function (*Responsive Link*)
- Peripheral functions for Computer (UART232C, DDR SDRAM I/Fs, DMAC, etc.)
- Peripheral control function (PWM Generators, Pulse Counters, etc.)

System designers can implement necessary function, only by connecting necessary I/Os (sensors, actuator, digital camera, and so on) to this chip. Distributed real-time system can be implemented, by connecting multiple *RMT Processor*with unique I/Os functions using *Responsive Link* .

1.2 Design Policy

*RMT Processor*is designed to achieve real-time processing, communication using unique feature of our processor. Real-time scheduler includes EDF (Earliest Deadline First) as dynamic scheduling, and RM (Rate Monotonic) as static scheduling. Almost all of the real-time scheduling require exection and communication with preemption for tasks with higher priority. Preemption of exection or processing is same as context switch, and that of communication is equivalent to packet overtaking.

Therefore, overhead for preferential execution or context switch of threads with priority by hardware will decrease. And, mechanism for packet overtaking based on priority, which was impossible in conventional communication, can be realized.

With these functions implemented on the hardware, *RMT Processor*enables priority based control of processing and communication by real-time scheduler (software) based on real-time scheduling algorithms.

1.3 Overall Structure

Picture 1.1 shows block diagram of *RMT Processor*. *RMT PU*connects DDR SDRAM I/F via 256bit bus. Buses with wide bandwidth as interface between processors and main memories improve throughput of memory accesses in instruction fetches and vector operations.

*Responsive Link*and I/Os are connected to 32bit bus. There is a gateway between 32bit bus and 256bit one. Data in each bus is resized by the gateway and passed to the other bus. Event link of *Responsive Link* is directly connected with memory access unit of *RMT PU*. Therefore, processor can read data as a part of control register without accessing buses. It realizes quick access of event link.

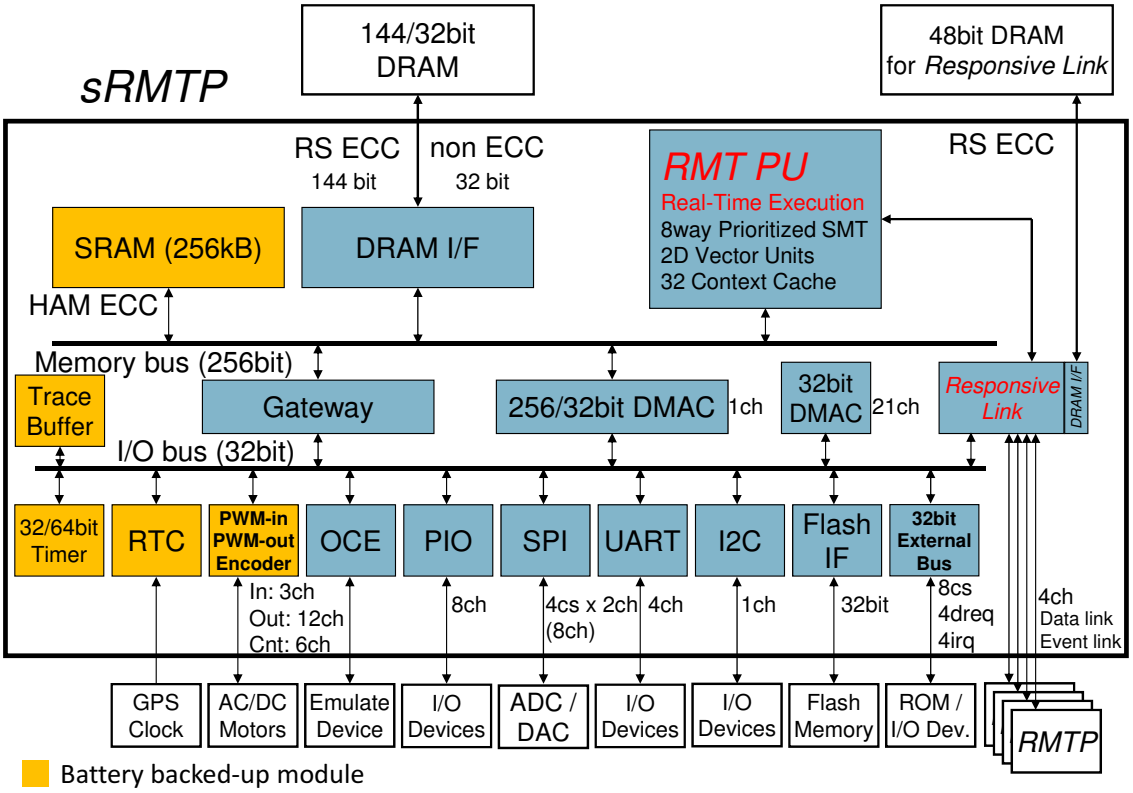


Figure 1.1: Block Diagram of *RMT Processor*

*RMT Processor*has following I/Os.

- *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*supports various hardware level real-time processing, by controling detailed 8-way multithreading using priority. Due to multithread architecture, multiple threads are executed in parallel, conflict of computational resources such as operation units or cache systems may occur among threads. When it occurs, *RMT PU*refers to priority set to each thread, and allocates resource to instructions with higher priority. For this reason, those with higher priority is prioritized, in parallel threads.

Figure 1.2 is block diagram of *RMT PU*. It is generally divided into three parts: issue unit, execution unit and cache unit. Issue unit controls execution of each thread and sends instructions to the execution unit according to priority. Execution unit operates instructions sent from issue unit. Cache unit process instruction fetch requests from issue unit and data access requests from execution unit.

1.4.1 Instruction Issue Unit

Instruction issue unit controls execution of each thread, and issue instruction to instruction operation unit. Table 1.1 shows abstract of instruction issue unit.

Active thread means thread saved in the processor. It can be exected with low latency. Cache thread are thread saved in the context cache, which is described later. Priority is set in 256levels (8bits), and threads with larger value have higher priority.

Thread control unit controls each thread. Active threads are managed by a thread table int the unit. Format of thread tables are shown in the figure 1.3 . ENABLE field shows whether active threads are valid or not. STATE field shows status of active threads, such as executing, blocked and evacuated in the context cache, which is described later. KEEP field shows whether or not to keep active threads in the processor. PRIOPRITY field shows threads' priority, and it is used all over the *RMT PU*.

Instructions for thread creating, erasing, executing, stoping and setting priority are newly added in *RMT PU*. When these instructions are issued, thread control unit rewrites thread table and controls active threads.

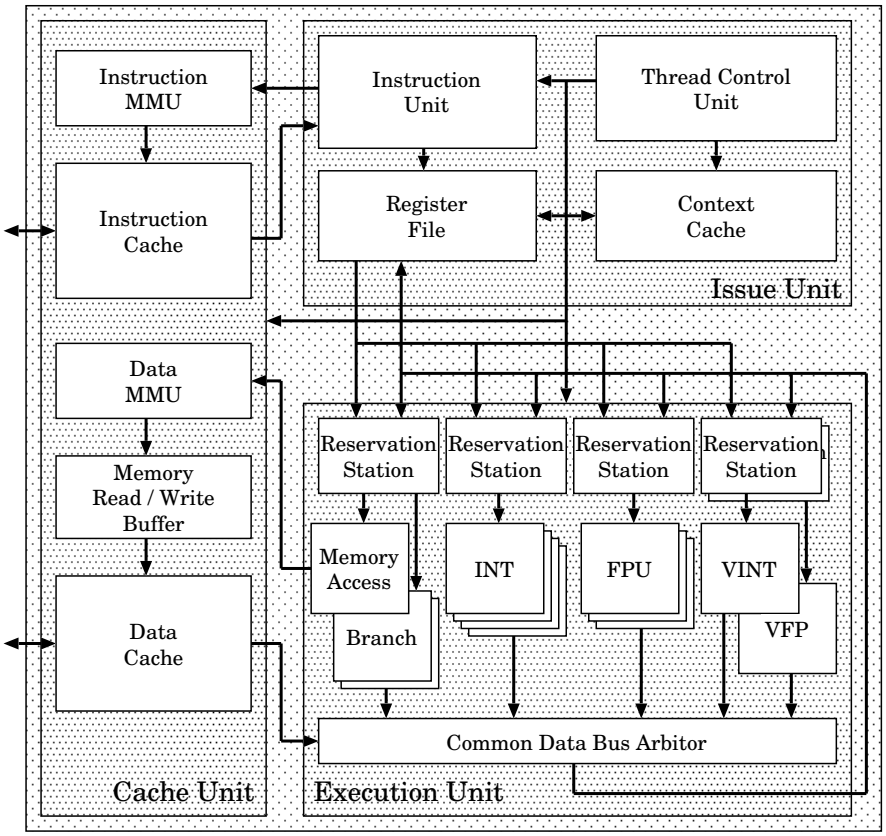


Figure 1.2: Block Diagram of *RMT PU*

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

Figure 1.3: Thread Table Format

*RMT PU*can save 8 contexts in processor. However, if processor try to execute more than that, context switch occurs. Context switch evacuates context of current executing thread into memory. Then newly executed thread must be restored from memory. So overhead becomes larger.

*RMT PU*has an on-chip cache exclusively for storing context. It is connected with register file through wide bus (GPR:256bit, FPR:128bit). The context cache can save up to 32 contexts. This hardware reduces overhead of context switch to 4 clock cycles.

Thread replacement unit performs evacuation of active threads to context cache, restoration of cache threads in the processor, exchange of active threads and cache threads, using newly added instructions. When the thread control unit receives a thread evacuation, restoration or exchange instruction, it searches the cache thread table it retains and generates addresses to access the context cache. Then it accesses the context cache.

The issue unit arbitrate the instruction cache accesses and instruction issues to the execution unit using priority.

Table 1.1: Abstract of Instruction Issue Unit

Number of Active Threads	8
Number of Cache Threads	32
Priority setting	256 levels
Number of Instruction Fetches	8
Simultaneous Instruction Issues	4
Simultaneous Instruction completes	4
Integer Register	32bit × 32entry × 8set
Integer Rename Register	32bit × 64entry
Floating Point Register	64bit × 8entry × 8set
Floating Point Rename Register	64bit × 64entry

Table 1.2: Abstract of Instruction Operation Unit

Integer Unit	4 + 1 (Divider)
Floting Point Unit	2 + 1 (Divider)
64bit Integer Unit	1
Integer Vector Unit	1 (8IU × 2unit)
Floating Point vector Unit	1 (4FPU × 2unit)
Branch Unit	2
Memory Access Unit	1
Synchronization Unit	1

1.4.2 Instruction Operation Unit

Instruction operation unit operates instructions issued by instruction issue unit. Table 1.2 shows abstract of instruction operation Unit.

*RMT PU*performs out-of-order execution, using reservation stations and reorder buffers. Since *RMT PU*executes multiple threads concurrently, confliction of arithmetic units between threads may occur. The execution unit controls reservation stations using priority. Reservation stations retain instructions until all the necessary operands for executions are ready. Then those stations are ready, instructions are issued to each arithmetic unit. In *RMT PU*, when multiple instructions are ready to execute, reservation stations compare priority, and issue one with higher priority. Therefore, allocation of arithmetic units to threads with higher priority are prioritized.

On the other hand, soft real-time processing, such as multi-media processing, performs repetitive operation for a lot of data. Therefore high operation performance are required. These processing utilize data parallelism to achieve higer performance.

*RMT PU*uses vector operation units. Vector operations utilize few instruction slots effectively, and realize high operation performace necessary for soft real-time processing. The number of vector registers required differ dependent on the number of threads and programs which execute vector operations. Therefore, *RMT PU*has 512 sets of vector registers for both integer and floating point, whose structure can be changed into different vector length and number of registers. And multiple threads share them. This mechanism realize flexible vector operations by multiple threads.

Table 1.3: 命令演算ユニットのレイテンシ

Integer Unit	1 cycle
Integer Unit (Divide)	3 cycle
Floting Point Unit	2 cycle
Floting Point Unit (Divide)	13 cycle
64bit Integer Unit	2 cycle
Integer Vector Unit	⌈ vlen/8 ⌉
Integer Vector Unit (Divide)	6 + vlen
Floating Point vector Unit	⌈ vlen/4 ⌉
Floating Point vector Unit (Divide)	3 + vlen × 13
Memory Access Unit (Load, Cache hit)	8 cycle
Memory Access Unit (Load, SRAM)	22 cycle
Memory Access Unit (Load, SDRAM)	35 cycle
Memory Access Unit (Store)	-

Table 1.4: Abstract of Cache Unit

TLB entry (instruction, data)	64 entry
cache size (instruction, data)	32K byte
victim cache (instruction, data)	512 byte

Vector operation, both for integer and floating points, are executed with two parallel operation piplelines, which enables concurrent vector operation by multiple threads possible. Integer pipelines have 8 operation units for each, and foating point pipelines have 4 for each, so that they can accept multiple vector operations concurrently. Also, by executing comprex operations defined by programmers as one instruction, utilization of vector units and performance of the operation improve.

1.4.3 Cache Unit

The cache unit processes instruction fetch request from the issue unit and data access request from the execution unit. Table 1.4 shows abstract of the cache unit.

The cache unit have MMU (Memory Magement Unit), which supports address translation using hardware. So, each thread is able to run a program with virtual address.

Dependent on where MMU is placed, whether to access cache using physical or virtual address are decided. In figure 1.2 , since MMU is placed before the cache, address is translated before accessing. Therefore the cache is accessed using physical address. Although address translation before cache accesses increase latency, there is no need to flush the cache even when executing thread changes by context switches. Also, when multiple threads share a memory area, using virtual address to access same physical memory can arise a proble that the same data is cached in multiple area(synonym). By using physical address to access the cache, *RMT PU*avoids the problem.

The TLB entries in MMU points shared information used by multiple thread, context group number as well as virtual and physical page number. Since *RMT PU*operates up to 8 threads, TLB entry miss is thought to be high. Using shared information, multiple threads share TLB entry to reduce TLB entry.

After setting shared information, context group number is used to add new thread in shared information. In order to set the TLB, by pointing context group number, add thread information into an entry's shared information, whose context group number match the thread's number. Thus,it make TLB valid without increasing TLB entries.

Both instruction cache and data cache are 8way set-associative cache, block size is 32byte, and cache access is pipelined. When cache miss occurs, there are two ways to select a block to be evacuated, LRU and priority. Thread with lower priority is evacuated from cache when using priority, so that higher priority thread remains in the cache.

The victim cache retains cache block evacuated from cache in full associative way. When cache miss occurs, if there is data left in the victim cache, the block is returned to the cache. It reduces requests to internal bus in cache miss, and improve memory access latency.

When accessing memory of lower layer because of cache miss or some data requests, accesses are controled based on priority. Since memory accesses are slower than cache, queues might arise. In this case, threads with higher priority obtain bus first to access memories.

1.5 *Responsive Link*

Responsive Link offers various functions to support flexible real-time communication. For instance, Separation of soft real-time communication(data-link) and hard real-time communication (event-link), overtaking packets with lower priority at each node, assignment of different path for different packet priority, changing priority of packet at each node to enable control packet speed in distributed control mode, error correction by hardware, dynamic control of communication speed, toplogy free and Hot-Plug&Play are available.

Responsive Link is standardized as IPSJ-TS 2003:0006 in Japan, and global standardization is in progress as ISO/IEC JTC1 SC25 WG4.

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

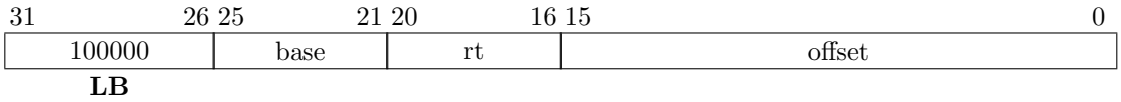
3.1 Instructions compatible with MIPS ISA

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 :

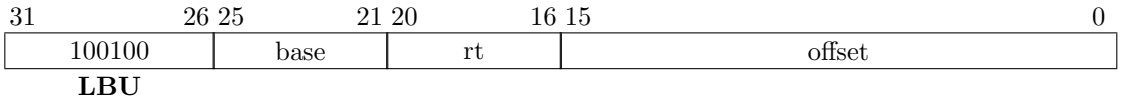
D-TLB No Entry Matched
D-TLB Protection Error

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 :

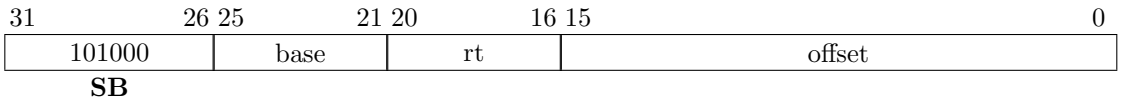
D-TLB No Entry Matched
D-TLB Protection Error

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 :

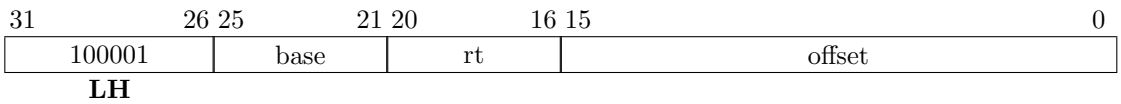
D-TLB No Entry Matched
D-TLB Protection Error

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 :

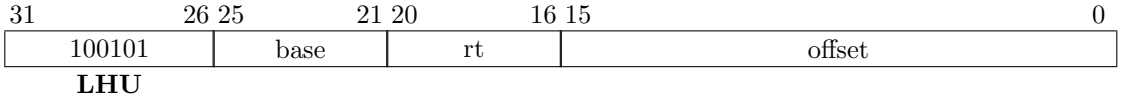
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Load)

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 :

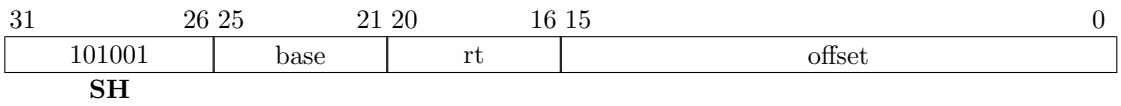
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Load)

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 :

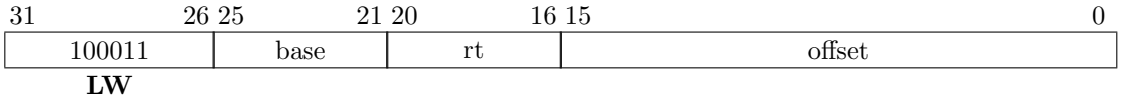
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Store)

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 :

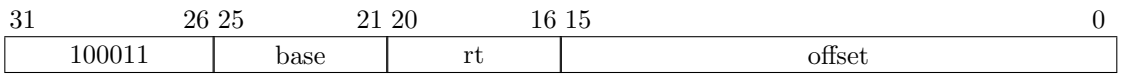
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Load)

Overview :

Load a word from memory.

SW — Store Word

32bit Load — MIPS I



LW

Mnemonic:

SW rt, offset(base)

Function :

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

Exception :

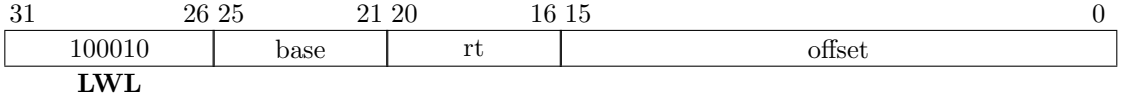
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Store)

Overview :

Store 1 word to memory.

LWL — Load Word Left

32bit unaligned load left — MIPS I



LWL

Mnemonic:

LWL rt, offset(base)

Function :

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

Exception :

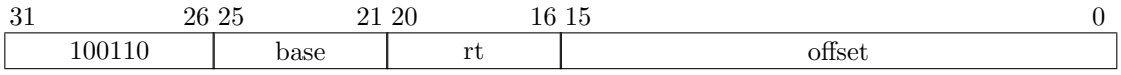
D-TLB No Entry Matched
D-TLB Protection Error

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 :

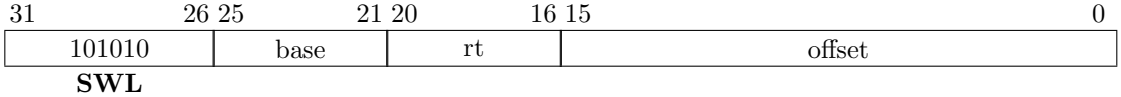
D-TLB No Entry Matched
D-TLB Protection Error

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 :

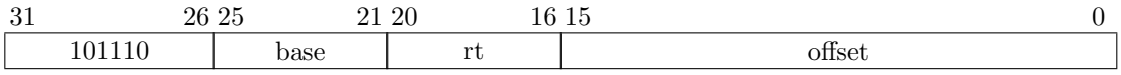
D-TLB No Entry Matched
D-TLB Protection Error

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 :

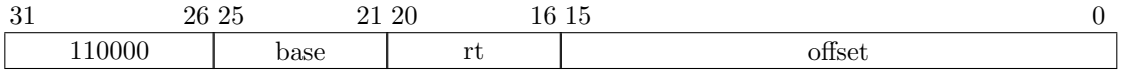
D-TLB No Entry Matched
D-TLB Protection Error

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 :

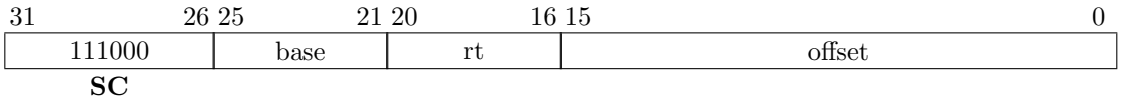
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Load)

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 :

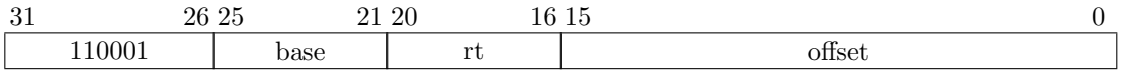
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Load)

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 :

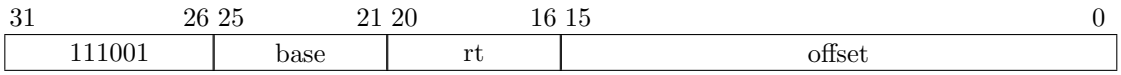
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Load)

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 :

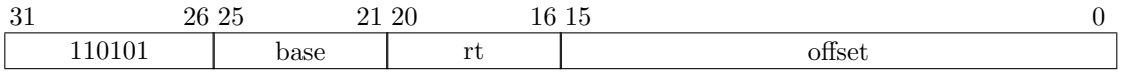
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Store)

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 :

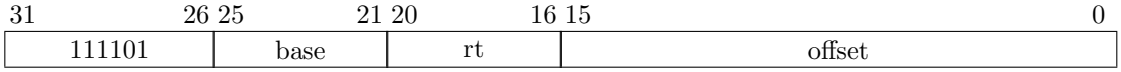
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Load)

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)

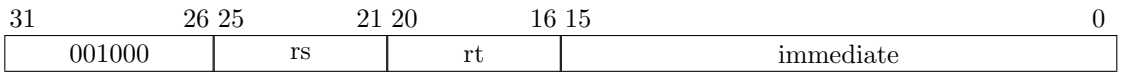
Overview :

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 :

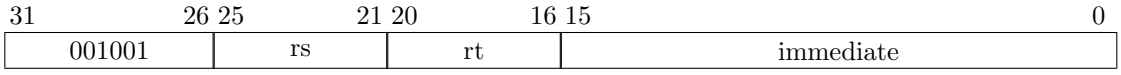
Overflow

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 :

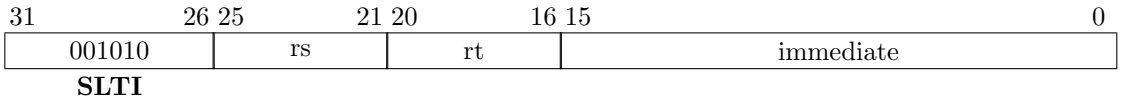
None

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 :

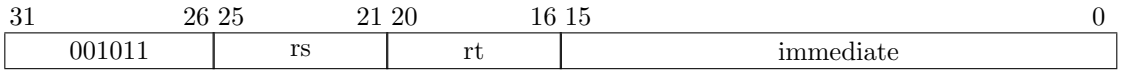
None

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 :

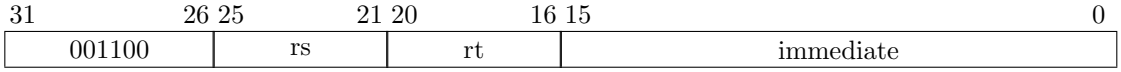
None

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 :

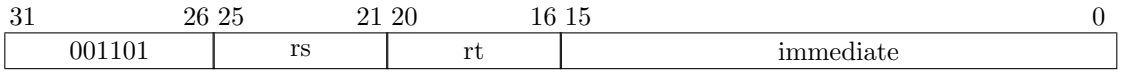
None

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 :

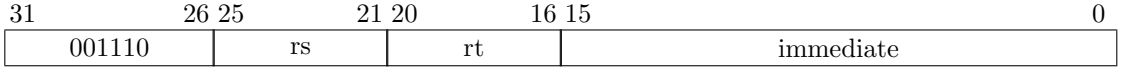
None

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 :

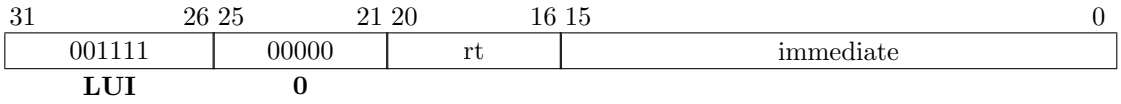
None

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 :

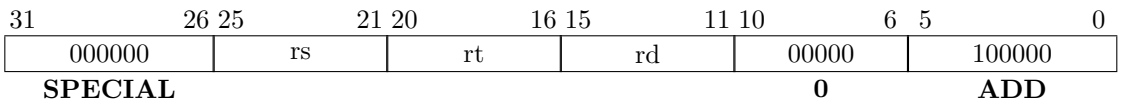
None

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 :

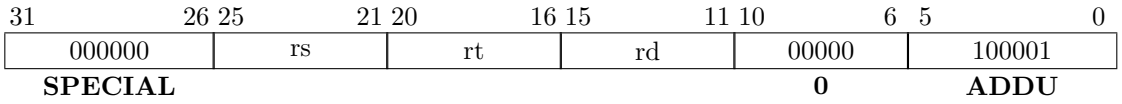
Overflow

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 :

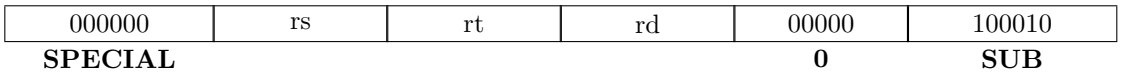
None

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 :

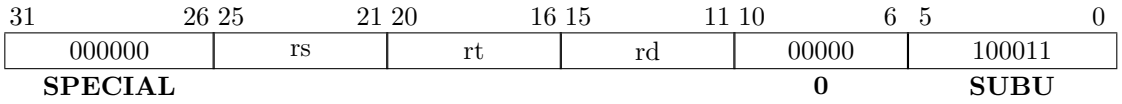
Overflow

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 :

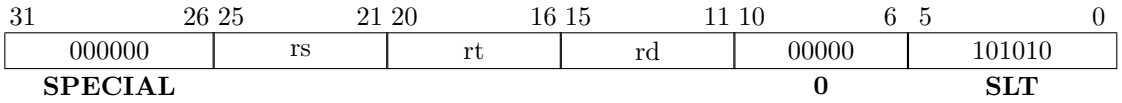
None

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 :

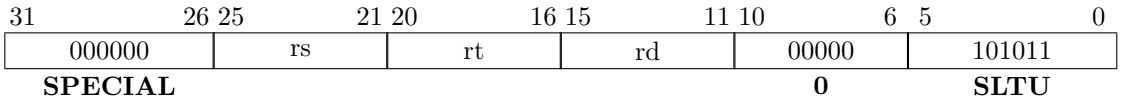
None

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 :

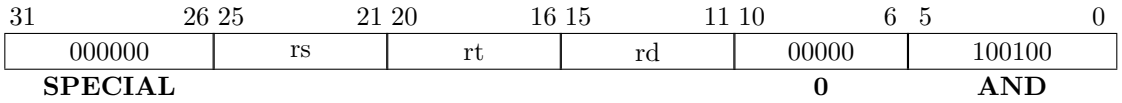
None

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 :

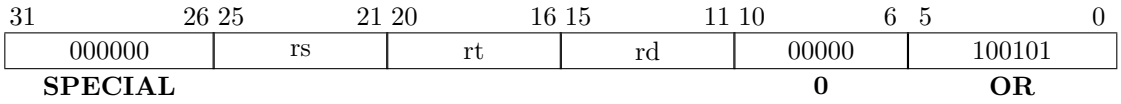
None

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 :

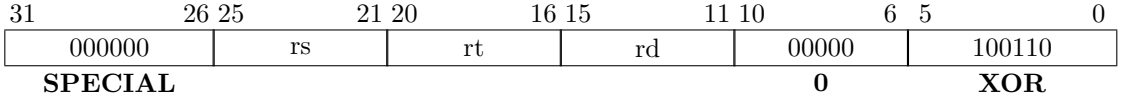
None

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 :

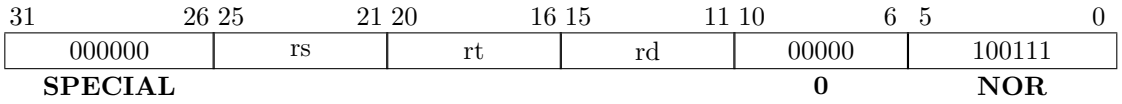
None

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 :

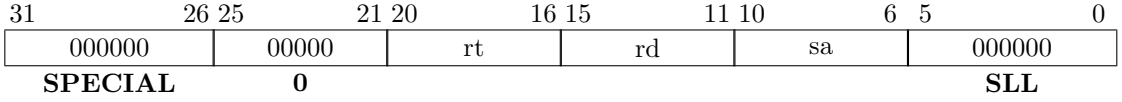
None

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 :

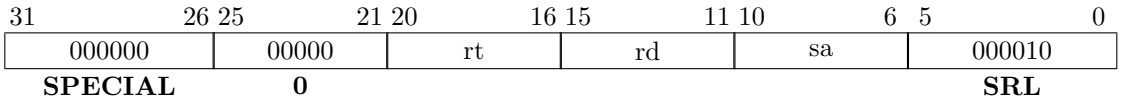
None

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 :

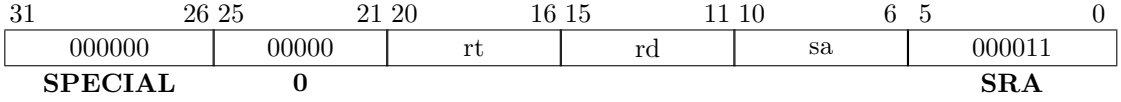
None

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 :

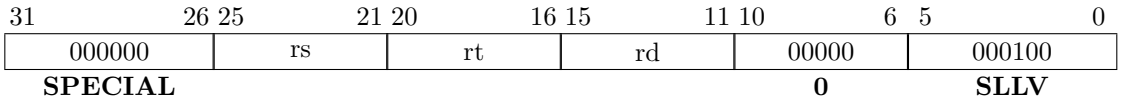
None

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 :

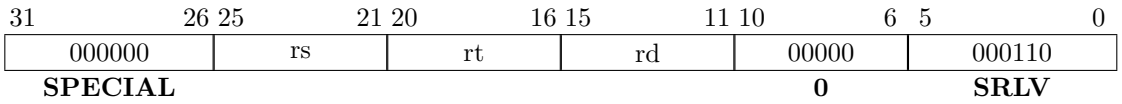
None

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 :

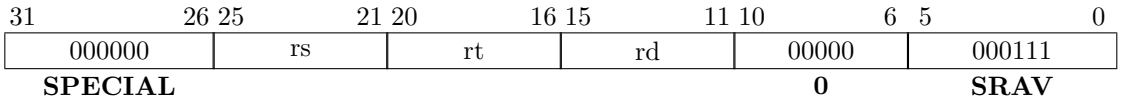
None

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

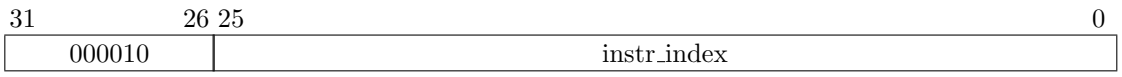
Overview :

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 :

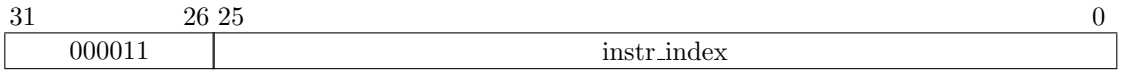
None

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 :

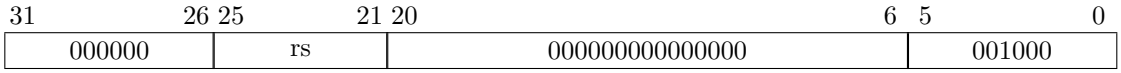
None

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 :

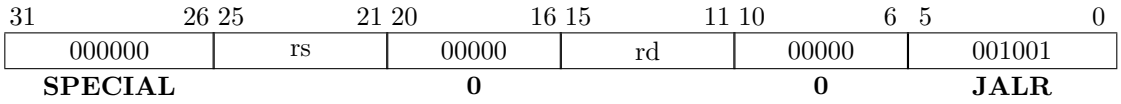
None

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 :

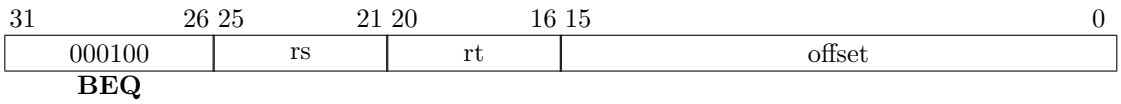
None

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 :

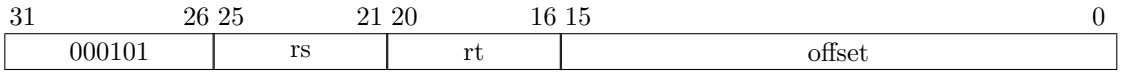
None

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 :

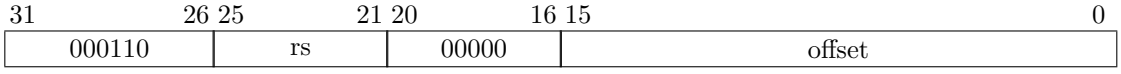
None

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 :

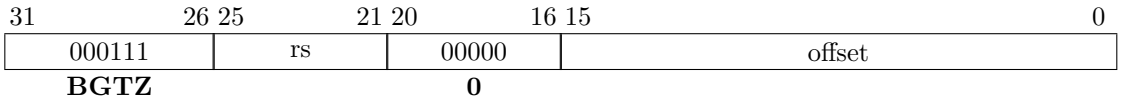
None

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 :

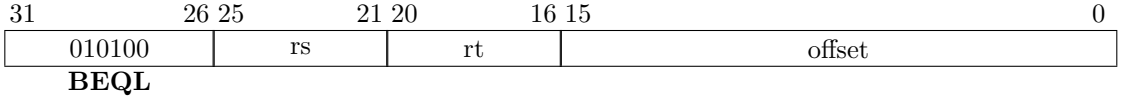
None

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 :

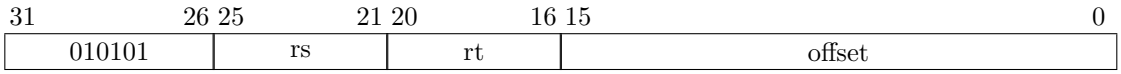
None

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



Mnemonic:

BNEL rs, rt, offset

Function :

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

Exception :

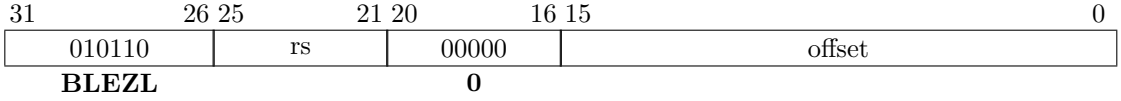
None

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



Mnemonic:

BLEZL rs, offset

Function :

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

Exception :

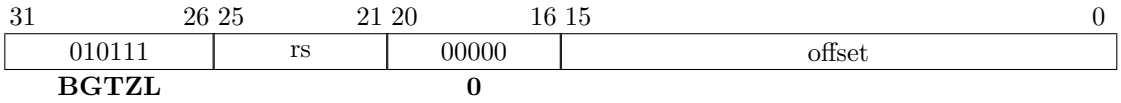
None

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 :

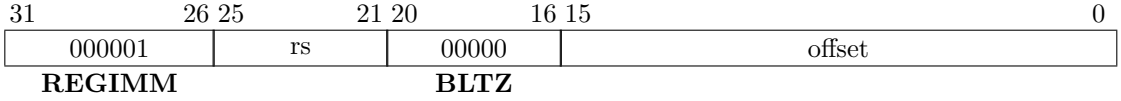
None

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 :

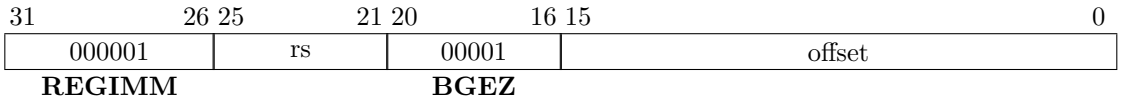
None

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 :

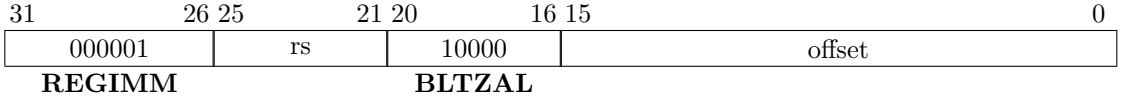
None

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 :

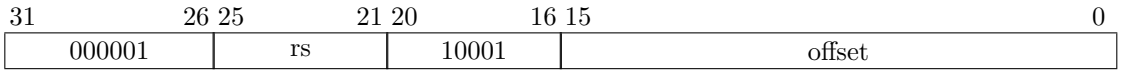
None

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 :

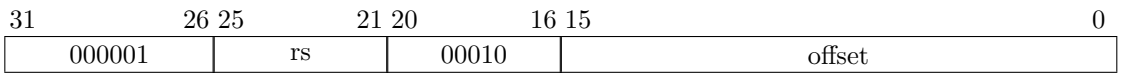
None

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 :

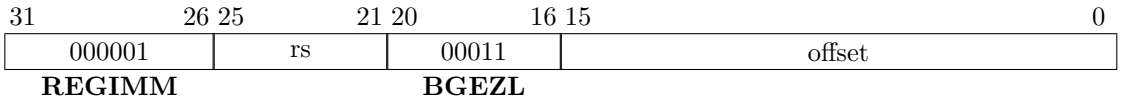
None

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 :

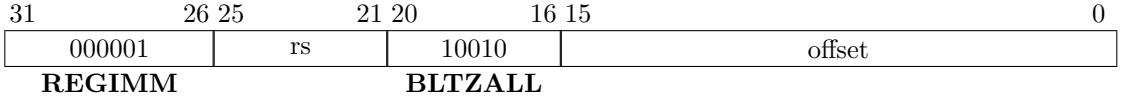
None

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 :

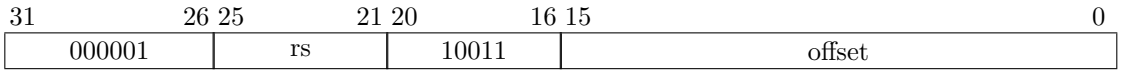
None

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

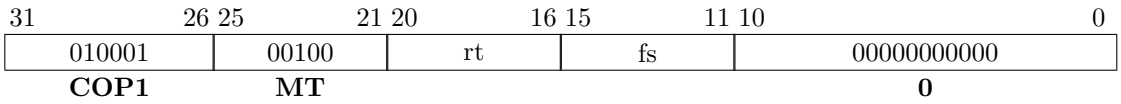
Overview :

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 :

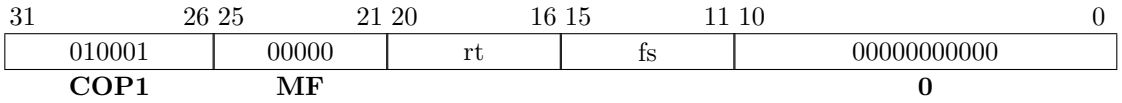
None

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 :

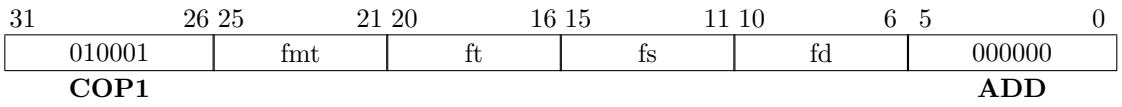
None

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 :

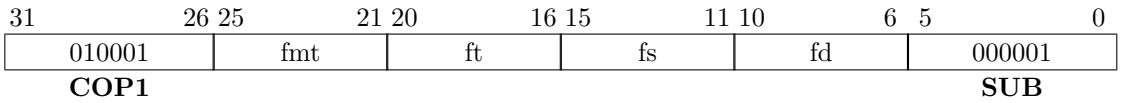
Floating Point Invalid Operation
Floating Point Inexact
Floating Point Overflow
Floating Point Underflow

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 :

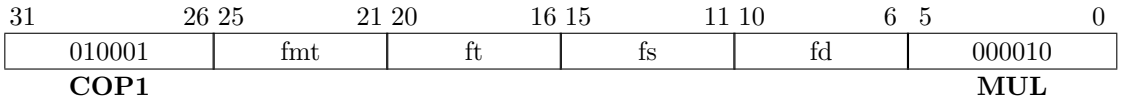
Floating Point Invalid Operation
Floating Point Inexact
Floating Point Overflow
Floating Point Underflow

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 :

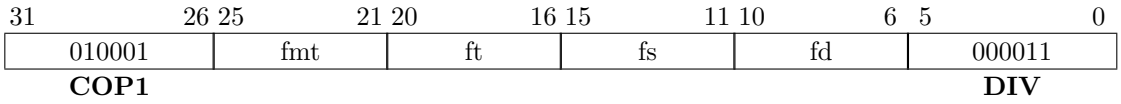
Floating Point Invalid Operation
Floating Point Inexact
Floating Point Overflow
Floating Point Underflow

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 :

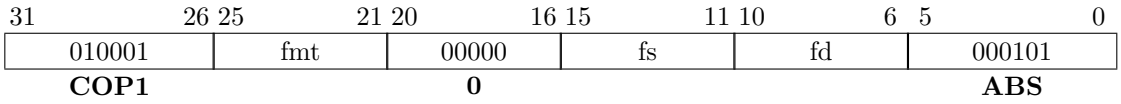
Floating Point Invalid Operation
Floating Point Inexact
Floating Point Overflow
Floating Point Underflow
Floating Point Divide By 0

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 :

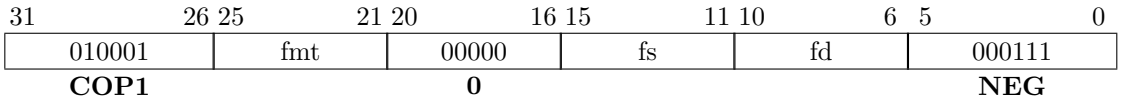
Floating Point Invalid Operation

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 :

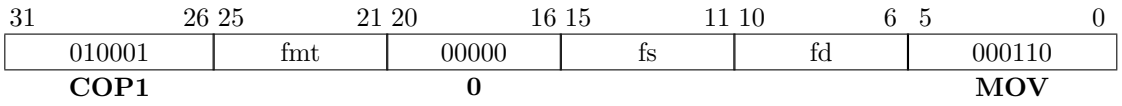
Floating Point Invalid Operation

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 :

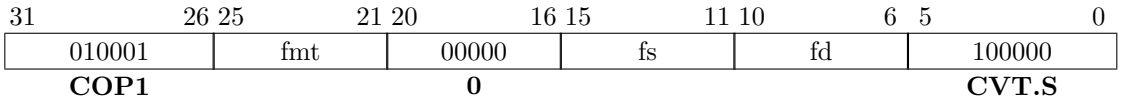
None

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 :

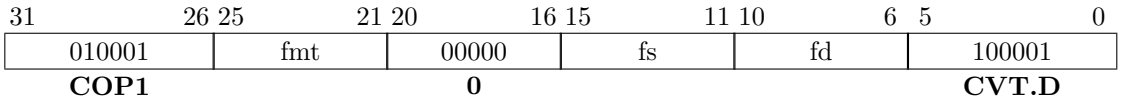
Floating Point Invalid Operation
Floating Point Inexact
Floating Point Overflow
Floating Point Underflow

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 :

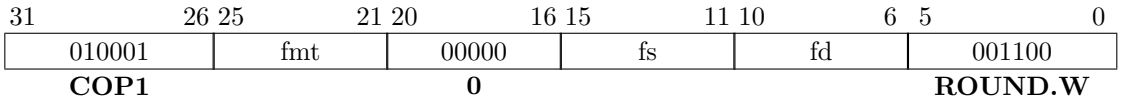
Floating Point Invalid Operation
Floating Point Inexact

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 :

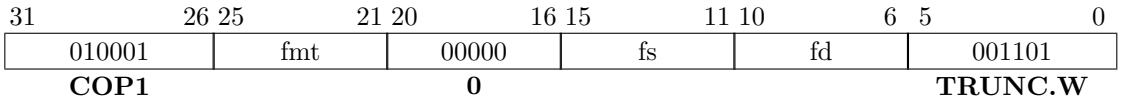
Floating Point Invalid Operation
Floating Point Inexact
Floating Point Overflow

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 :

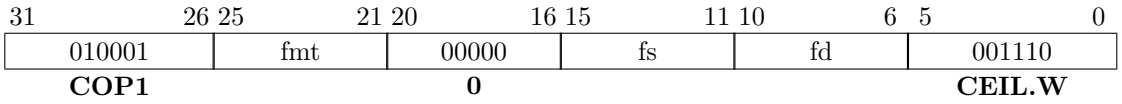
Floating Point Invalid Operation
Floating Point Inexact
Floating Point Overflow

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 :

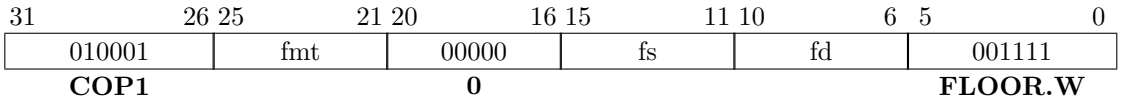
Floating Point Invalid Operation
Floating Point Inexact
Floating Point Overflow

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

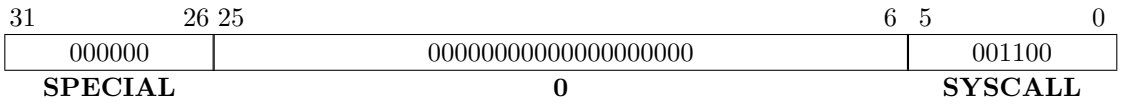
Overview :

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 :

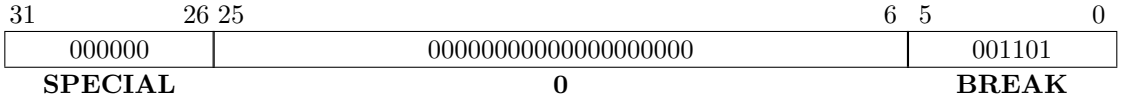
System Call

Overview :

Cause a System Call exception.

BREAK — Breakpoint

Breakpoint MIPS I



Mnemonic:

BREAK

Function :

exception(breakpoint)

Exception :

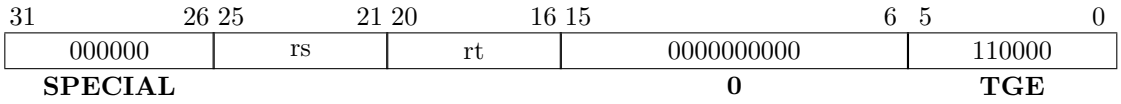
Break Point

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 :

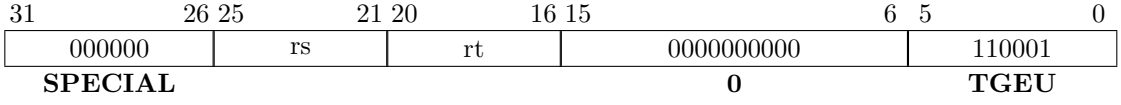
Trap

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 :

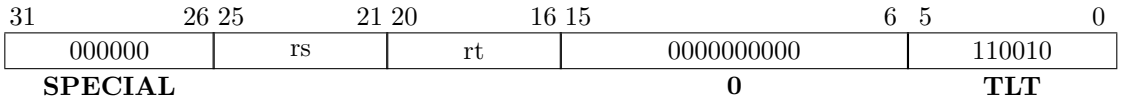
Trap

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 :

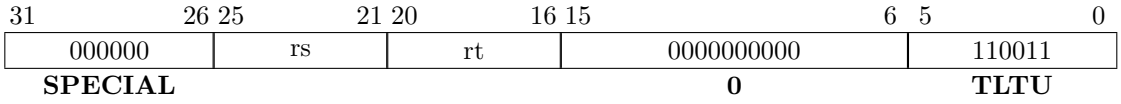
Trap

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 :

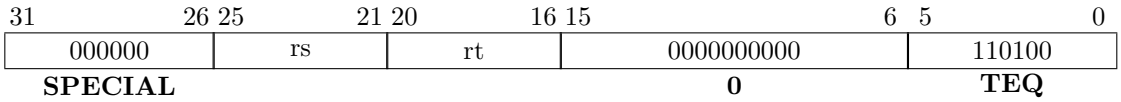
Trap

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 :

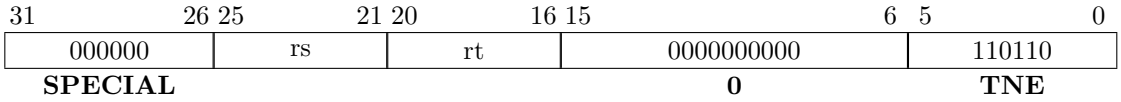
Trap

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 :

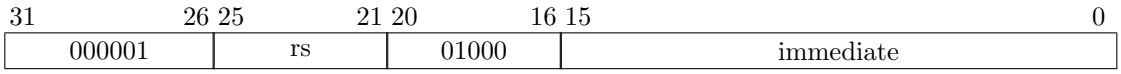
Trap

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 :

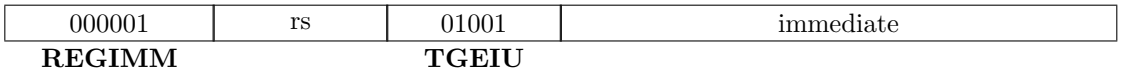
Trap

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 :

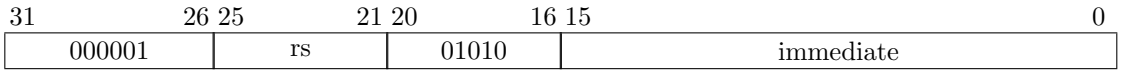
Trap

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 :

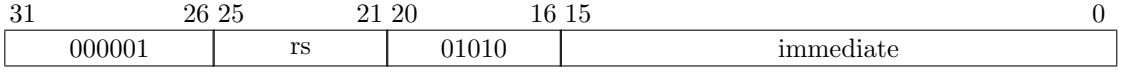
Trap

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 :

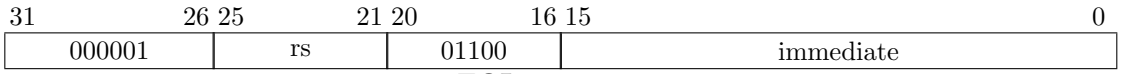
Trap

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 :

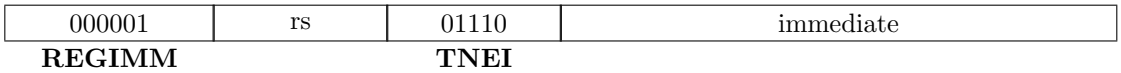
Trap

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.

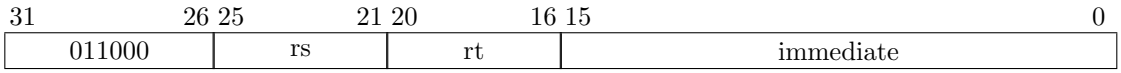
3.2 Instructions which are not compatible with MIPS ISA

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 :

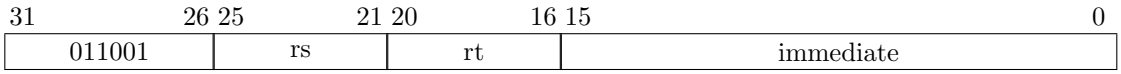
Overflow

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 :

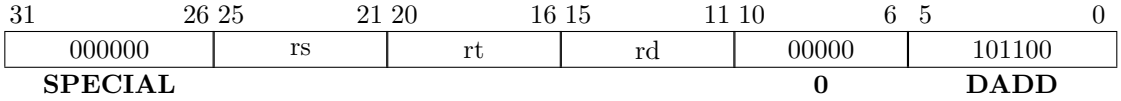
None

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 :

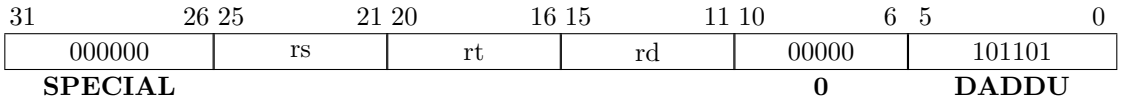
Overflow

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 :

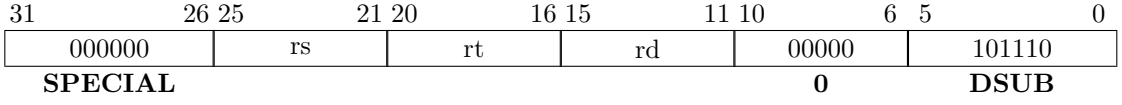
None

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 :

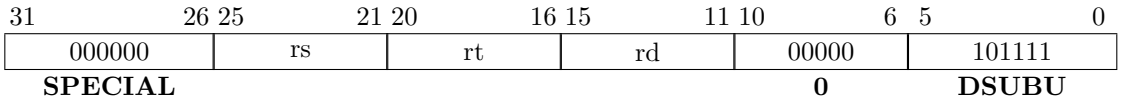
Overflow

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 :

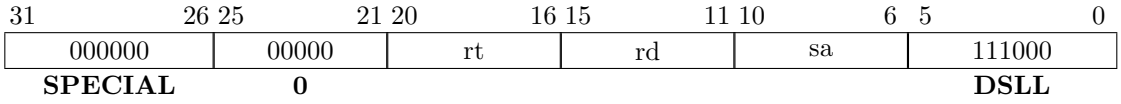
None

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 :

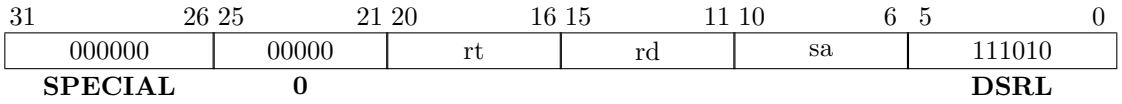
None

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 :

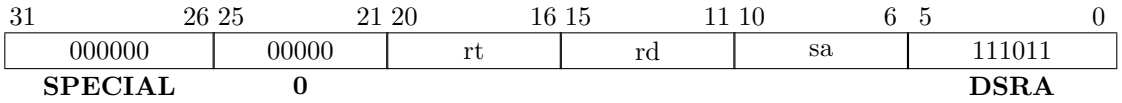
None

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 :

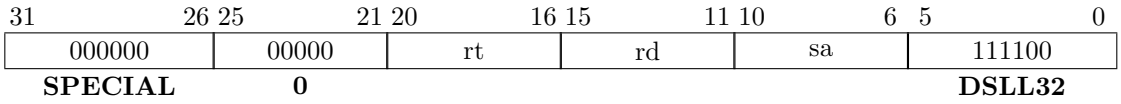
None

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 :

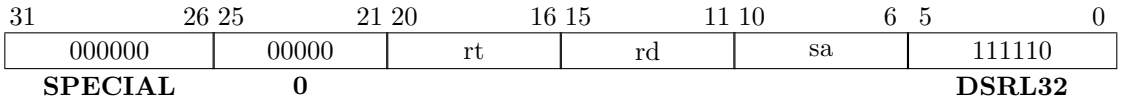
None

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 :

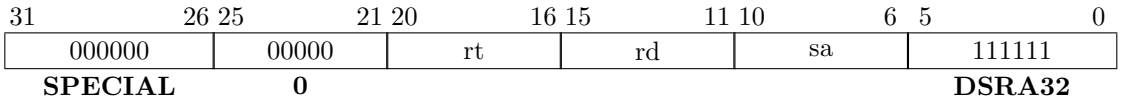
None

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 :

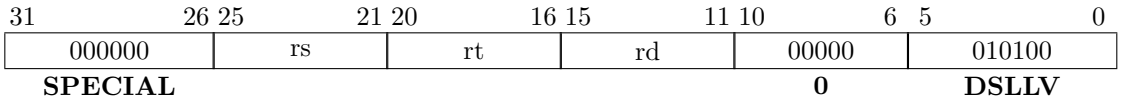
None

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 :

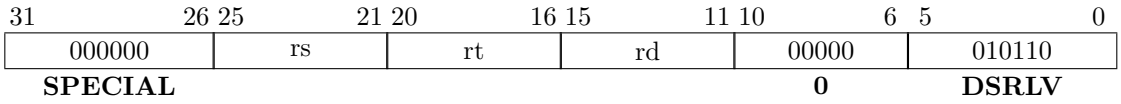
None

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 :

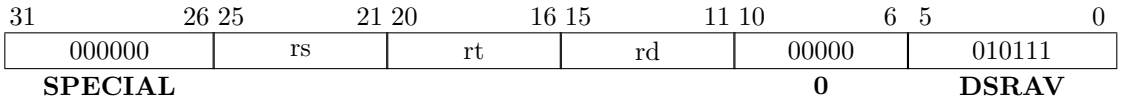
None

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 :

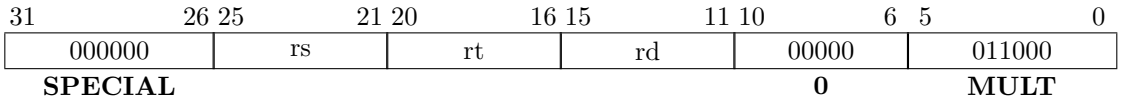
None

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 :

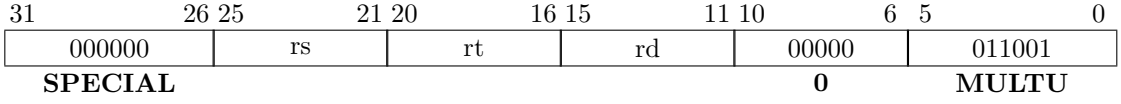
None

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 :

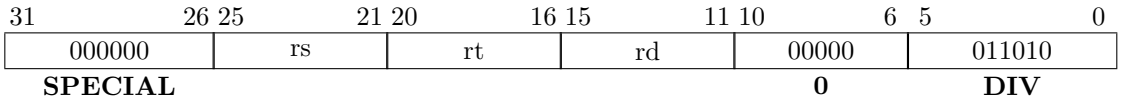
None

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 :

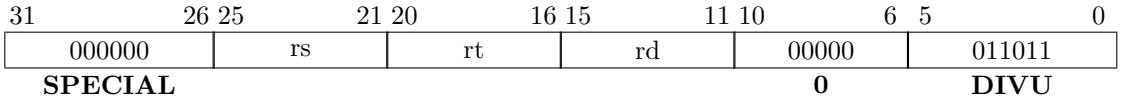
Divide by Zero

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 :

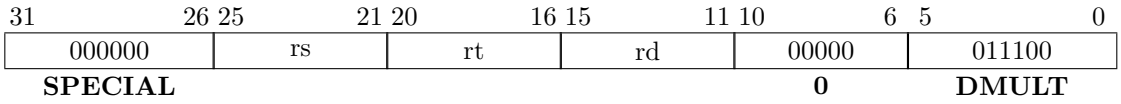
Divide by Zero

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 :

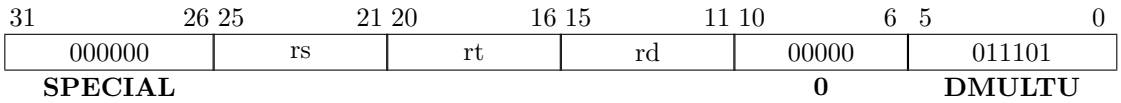
None

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

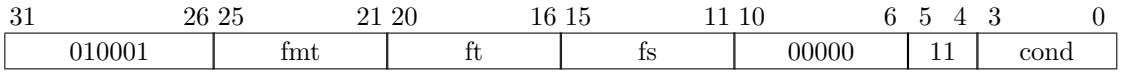
Overview :

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 :

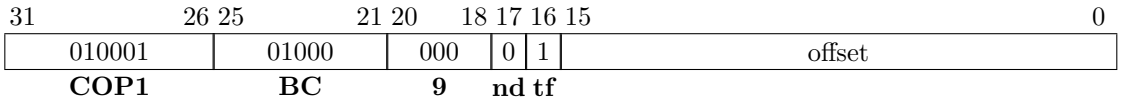
Floating Point Invalid

Overview :

Compare FPR values. In *Responsive Multithreaded Processor*, its result is not placed into status register, but FPR[rd].

BC1T — Branch on FP True

Branch on FP True — MIPS I modified



Mnemonic:

BC1T offset

Function :

if FPR[7] = 1 then
branch

Exception :

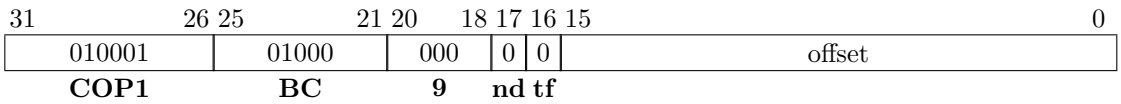
None

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 :

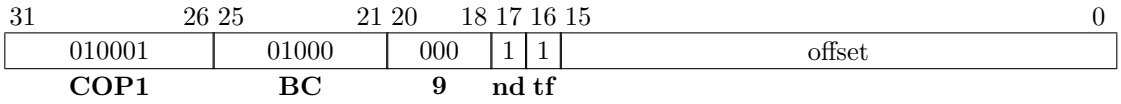
None

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 :

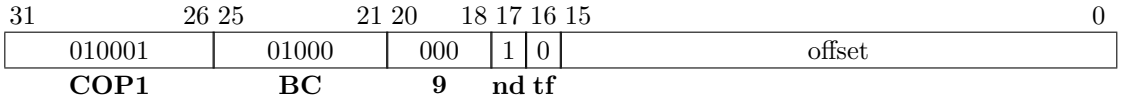
None

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

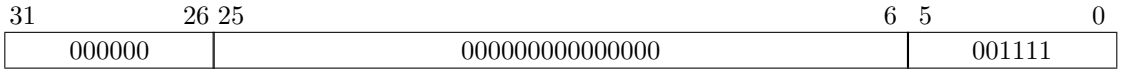
Overview :

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

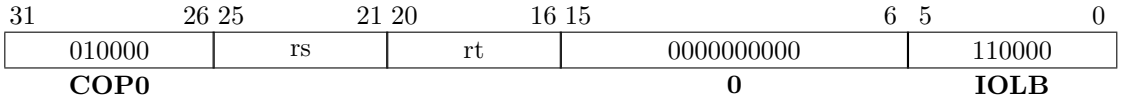
3.3 Responsive Multithreaded Processor Specific Instructions

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 :

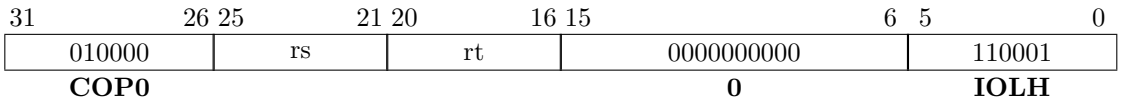
D-TLB No Entry Matched
D-TLB Protection Error

Overview :

Load a Byte from specified address. Because this instruction is not speculative execution, it is used for I/Os whoes status may be changed after load.

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 :

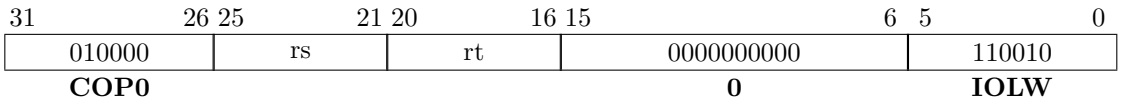
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Load)

Overview :

Load a Half Word from specified address. Because this instruction is not speculative execution, it is used for I/Os whoes status may be changed after load.

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)

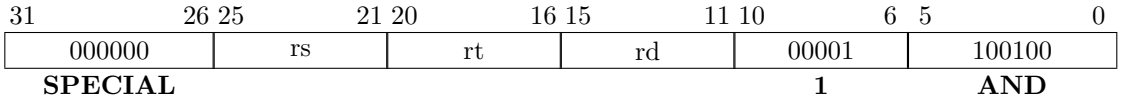
Overview :

Load a Word from specified address. Because this instruction is not speculative execution, it is used for I/Os whoes status may be changed after load.

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 :

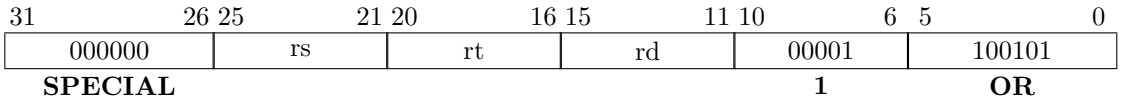
None

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 :

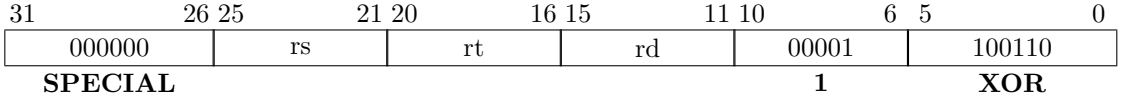
None

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 :

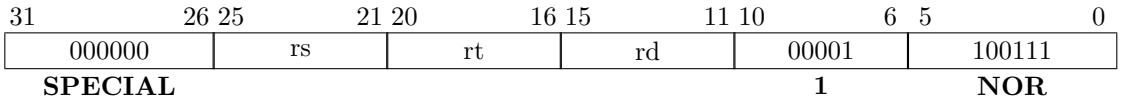
None

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 :

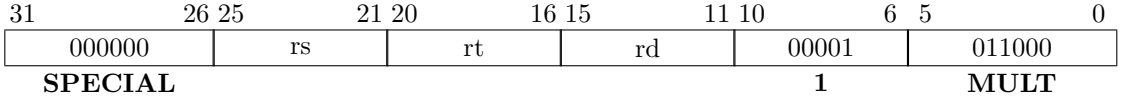
None

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 :

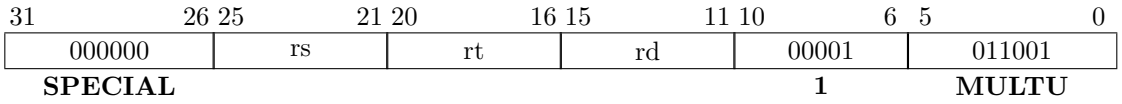
None

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 :

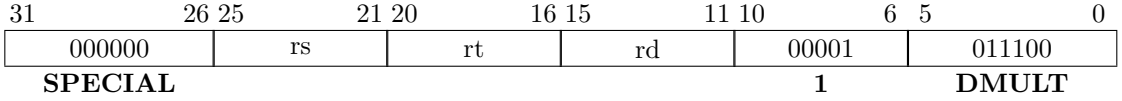
None

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 :

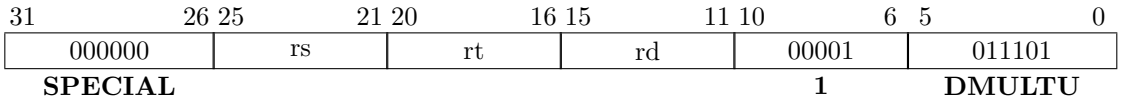
None

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 :

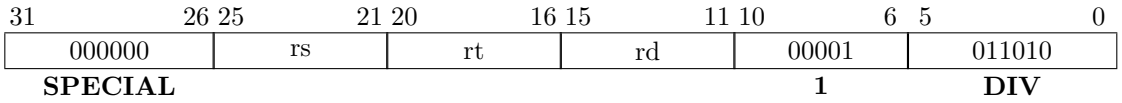
None

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 :

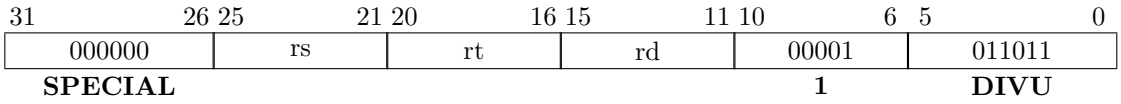
Divide by Zero

Overview :

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

REMU — Reminder Word Unsigned

Unsigned Reminder RESPII



Mnemonic:

REMU rd, rs, rt

Function :

GPR[rd] ← GPR[rs] ÷ GPR[rt]

Exception :

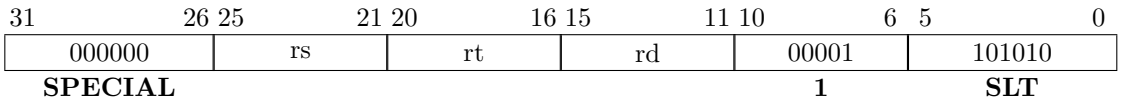
Divide by Zero

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



Mnemonic:

DSLT rd, rs, rt

Function :

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

Exception :

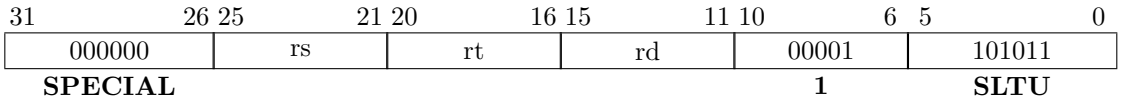
None

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 :

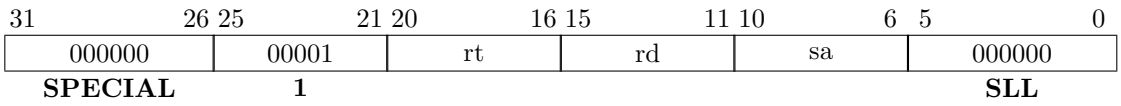
None

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 :

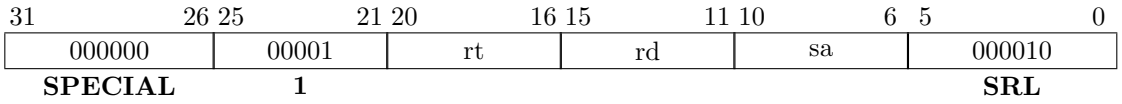
None

Overview :

Perform a left rotate operation.

RTR — Rotate Right

Rotate Right RESPII



Mnemonic:

RTR rd, rt, sa

Function :

GPR[rd] ← GPR[rt] >>> sa

Exception :

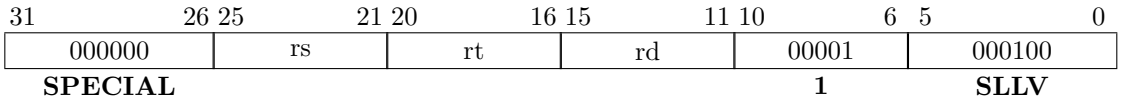
None

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 :

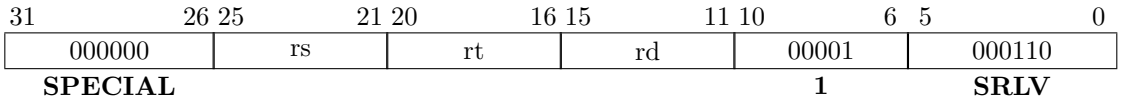
None

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 :

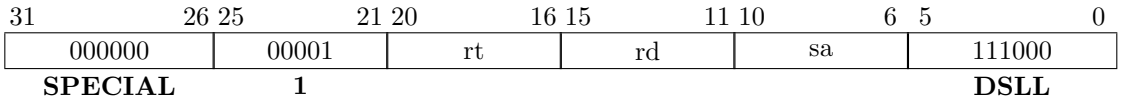
None

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 :

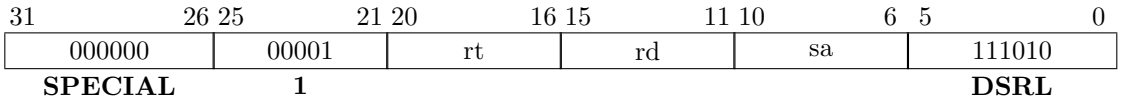
None

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 :

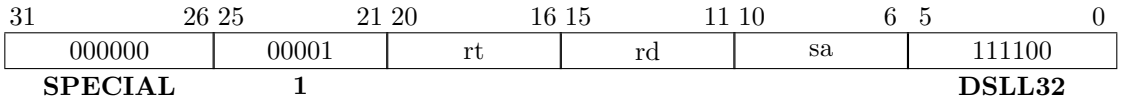
None

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 :

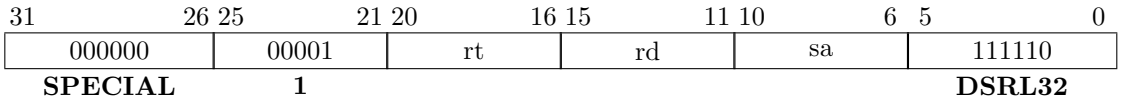
None

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 :

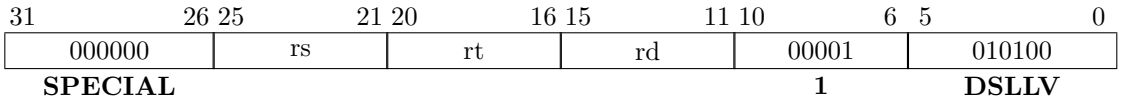
None

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 :

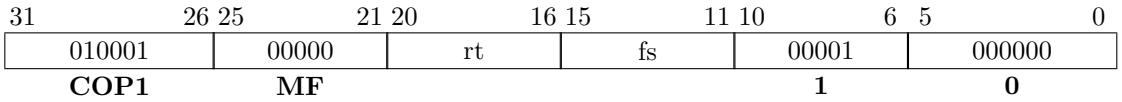
None

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



Mnemonic:

MFC1H rt, fs

Function :

GPR[rt] ← high_32bit(FPR[fs])

Exception :

None

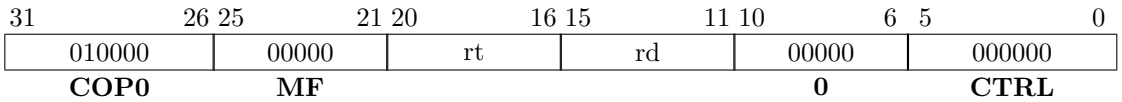
Overview :

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)



Mnemonic:

MFC0 rt, rd

Function :

GPR[rt] ← SYSTEM[GPR[rd]]

Exception :

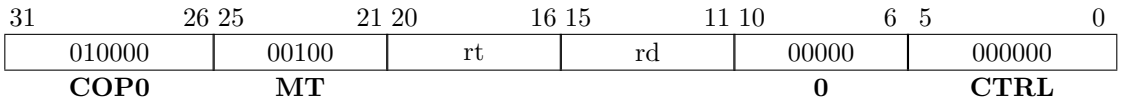
Coprocessor Unusable

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 :

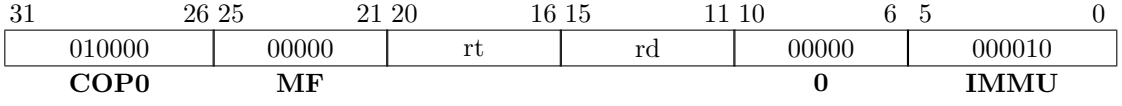
Coprocessor Unusable

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 :

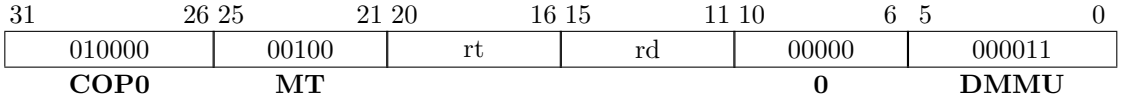
Coprocessor Unusable

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 :

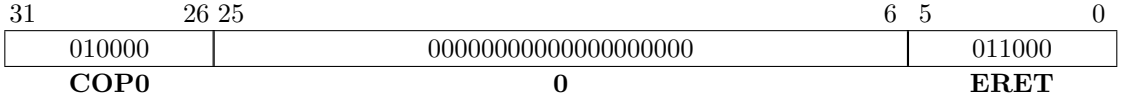
Coprocessor Unusable

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

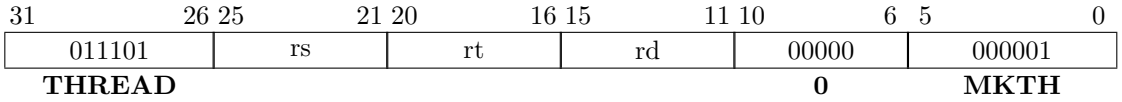
Overview :

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

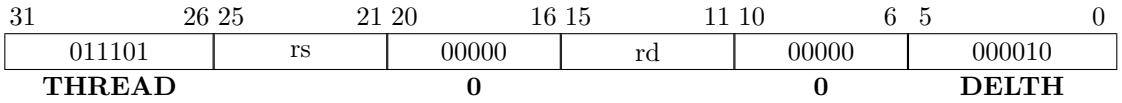
Exception :

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
```

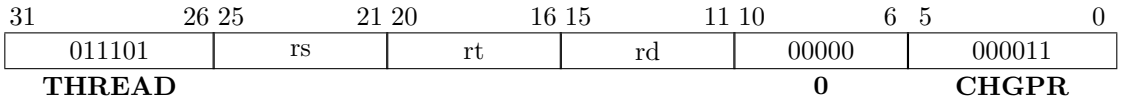
Exception :

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

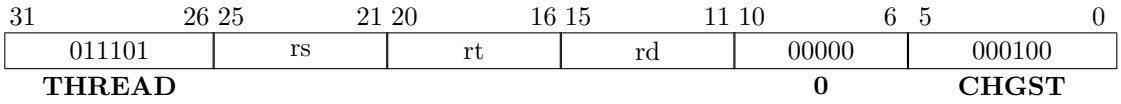
Exception :

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

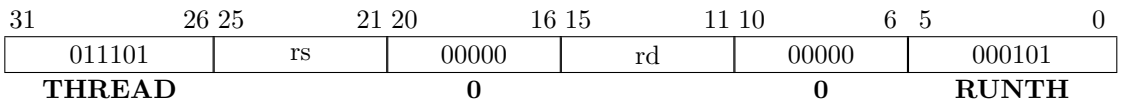
Exception :

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

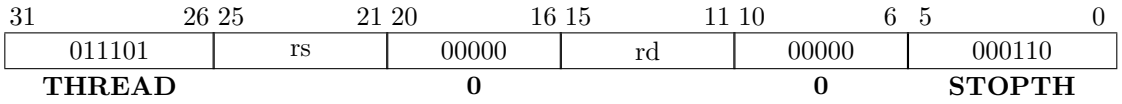
Exception :

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

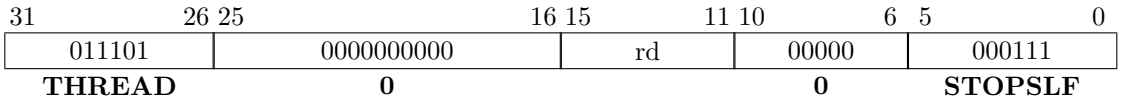
Exception :

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

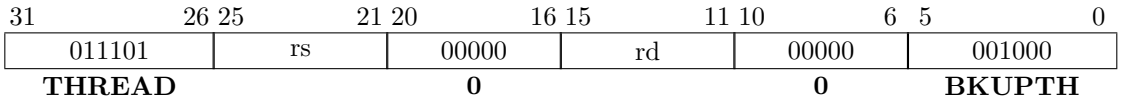
Exception :

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

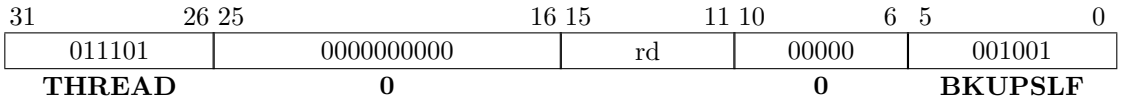
Exception :

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
```

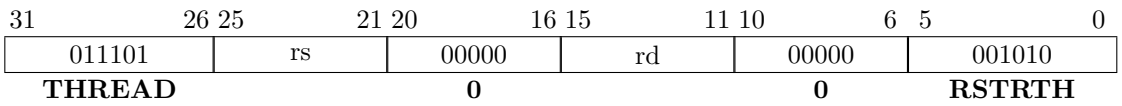
Exception :

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
```

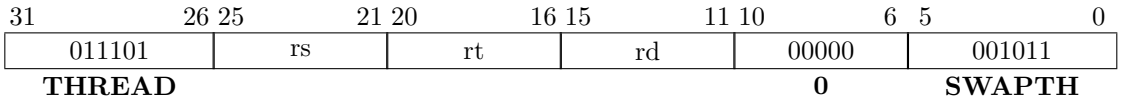
Exception :

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

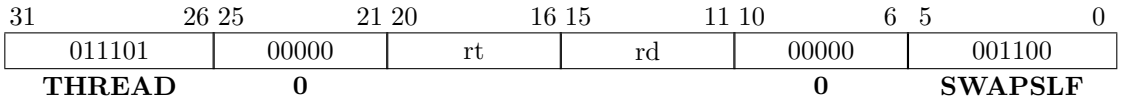
Exception :

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
```

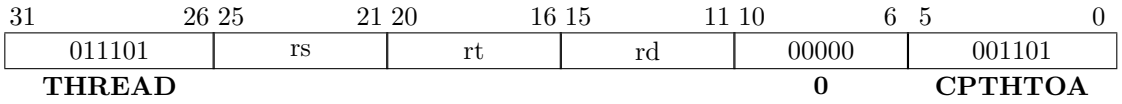
Exception :

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

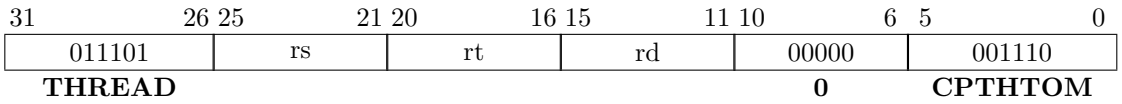
Exception :

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

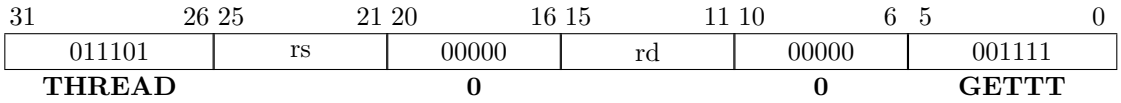
Exception :

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]

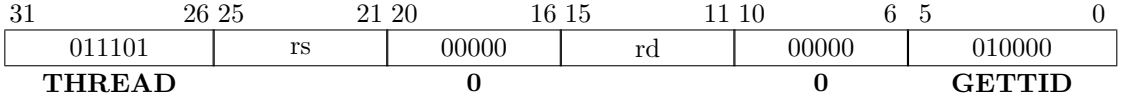
Exception :

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]

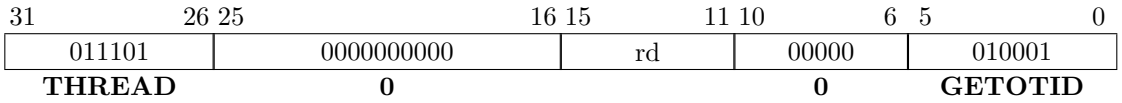
Exception :

Overview :

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

GETOTID — Get Own Thread ID

Get Own Thread ID THREAD



Mnemonic:

GETOTID rd

Function :

GPR[rd] ← Own ThreadID

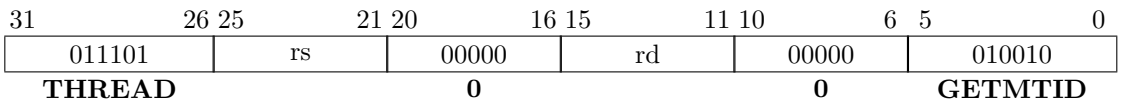
Exception :

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]

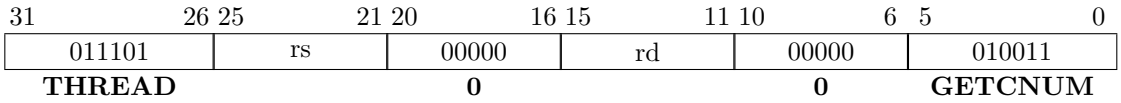
Exception :

Overview :

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

GETCNUM — Get Context ID Number

Get Context ID THREAD



Mnemonic:

GETCNUM rd, rs

Function :

GPR[rd] ← ContextID of GPR[rs]

Exception :

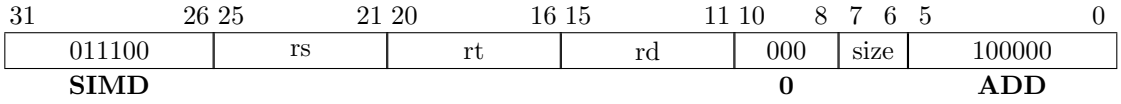
Overview :

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 :

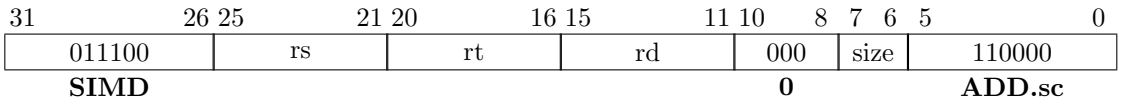
None

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 :

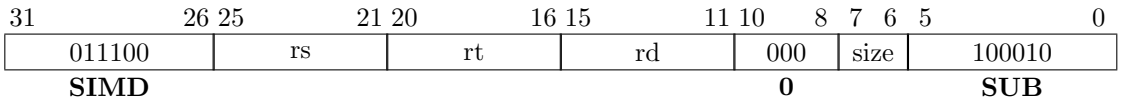
None

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 :

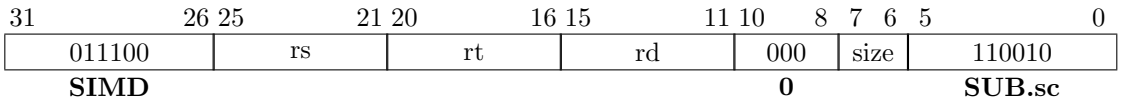
None

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 :

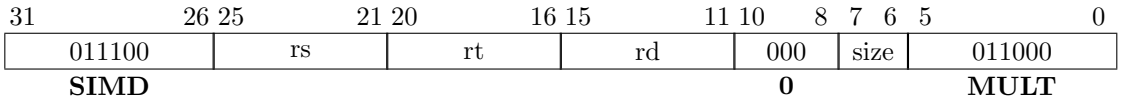
None

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 :

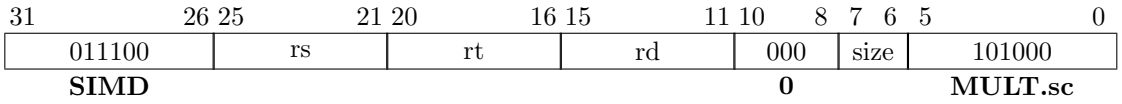
None

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 :

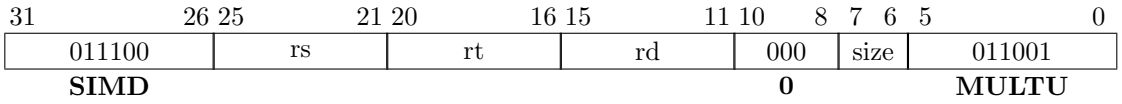
None

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



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

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 :

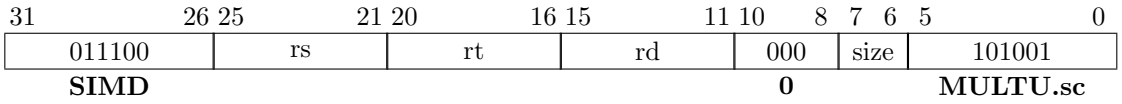
None

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 :

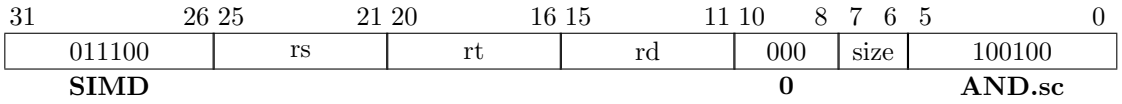
None

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 :

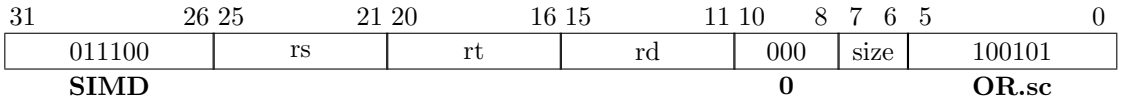
None

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 :

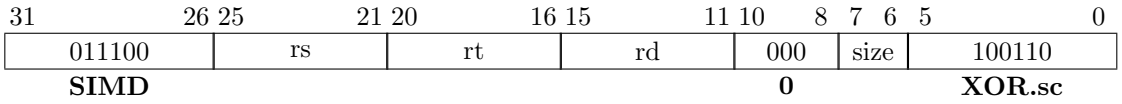
None

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 :

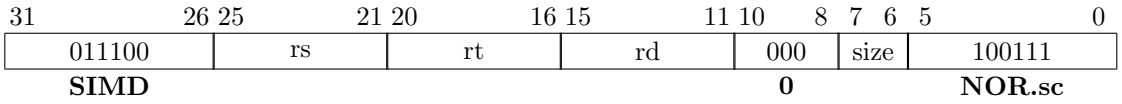
None

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 :

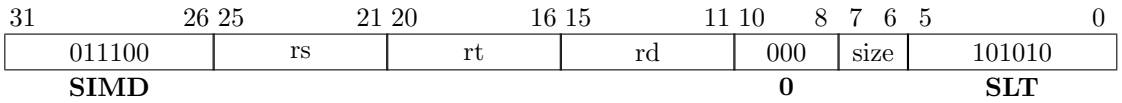
None

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 :

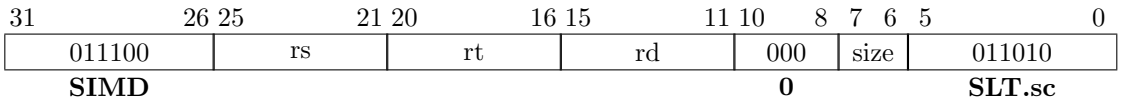
None

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 :

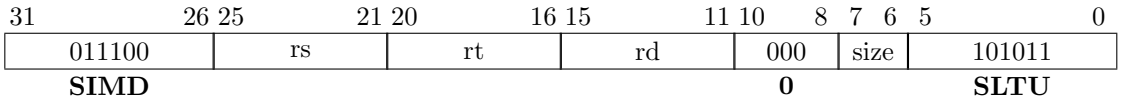
None

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 :

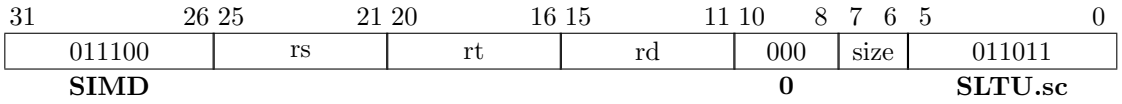
None

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 :

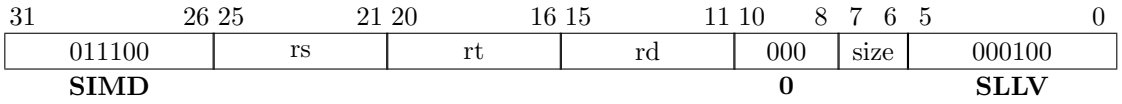
None

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 :

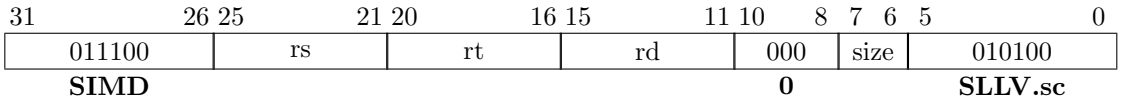
None

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 :

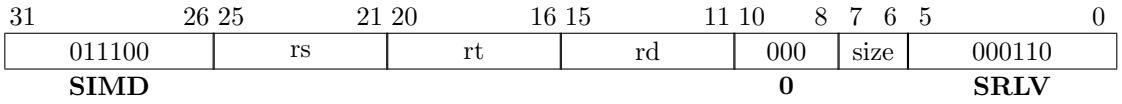
None

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 :

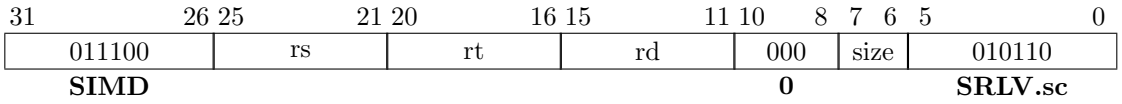
None

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 :

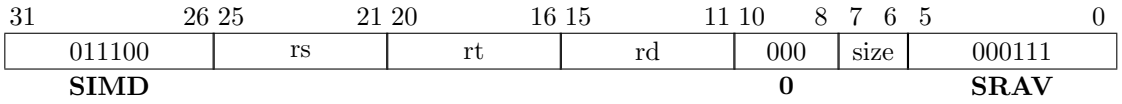
None

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 :

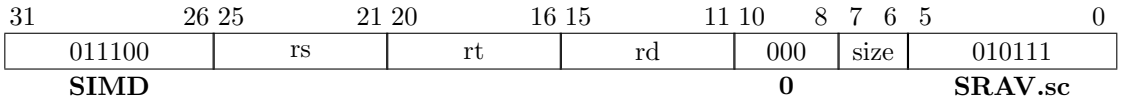
None

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 :

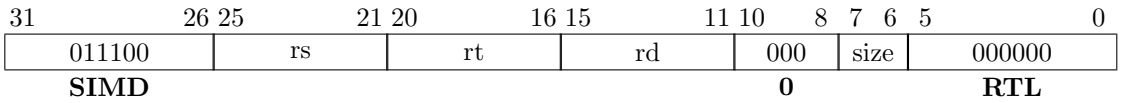
None

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 :

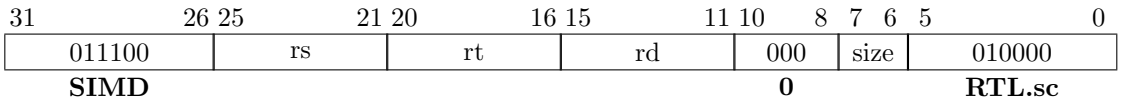
None

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 :

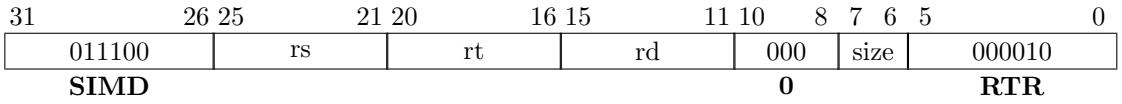
None

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 :

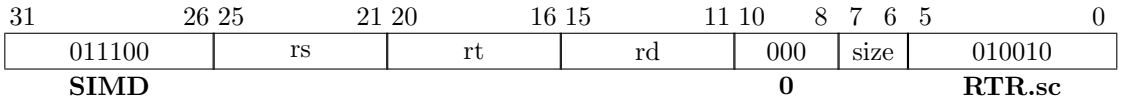
None

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 :

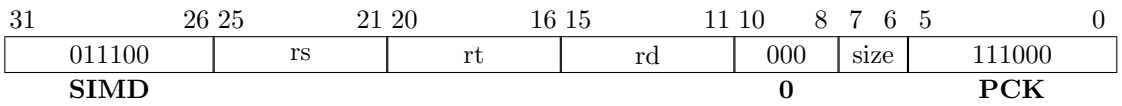
None

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 :

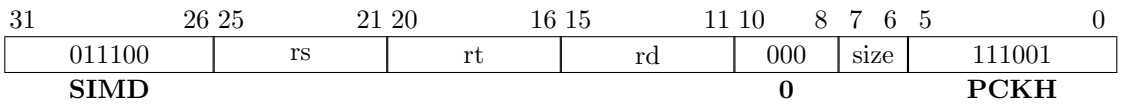
None

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 :

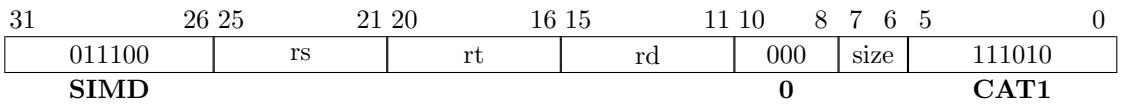
None

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 :

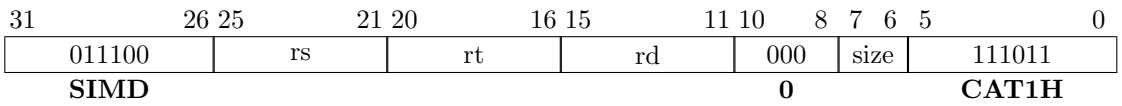
None

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 :

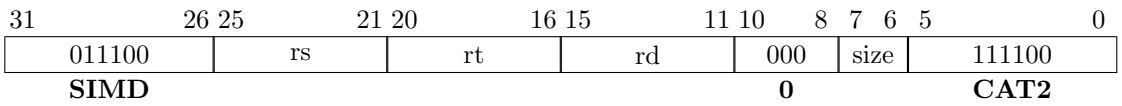
None

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 :

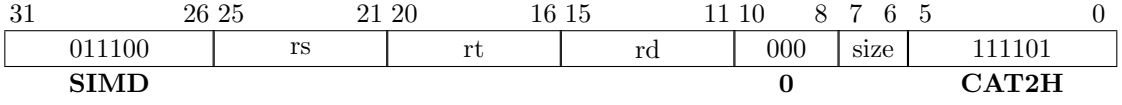
None

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 :

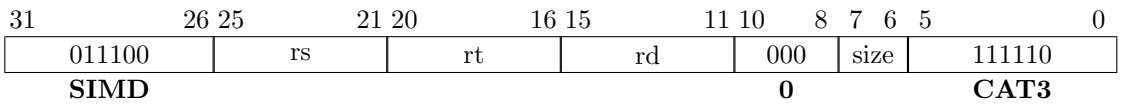
None

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 :

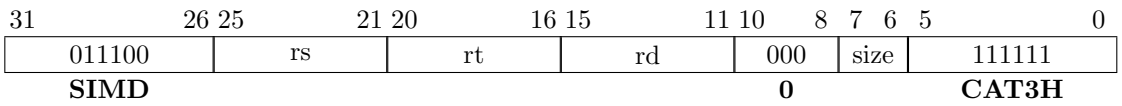
None

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 Synchronization Instruction

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 :

Move a word from shared register to GPR.

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 :

Move a word to shared register from GPR.

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 :

Move a word from shared register to FPR.

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 :

Move a word to shared register from FPR.

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 :

Move a word from shared register to GPR.

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 :

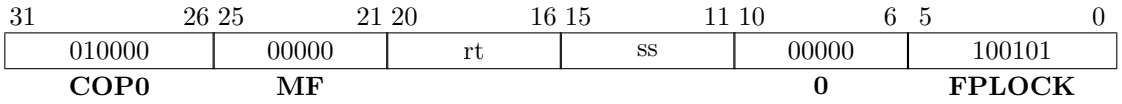
None

Overview :

Move a word to shared register from GPR.

RFPEX — Read Exclusive(Floating-Point Register)

Synchronization Instruction SYNC



Mnemonic:

RFPEX rt, ss

Function :

FPR[rt] ← SHARE[ss]

Exception :

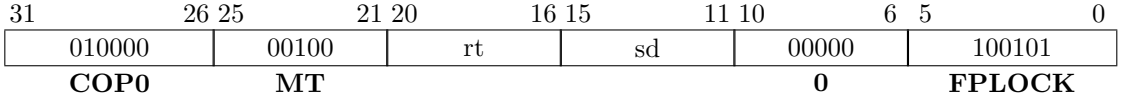
None

Overview :

Move a word from shared register to FPR.

WFPEX — Write Exclusive(Floating-Point Register)

Synchronization Instruction SYNC



Mnemonic:

WFPEX rt, sd

Function :

SHARE[sd] ← FPR[rt]

Exception :

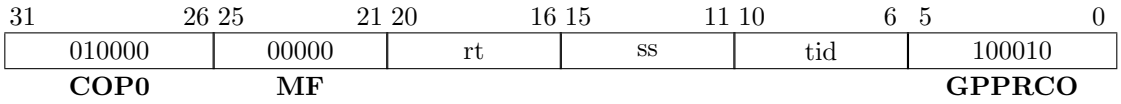
None

Overview :

Move a word to shared register from FPR.

GPCO — Read Consumer(General Purpose Register)

Synchronization Instruction — SYNC



Mnemonic:

GPCO rt, ss, tid

Function :

GPR[rt] ← SHARE[ss]

Exception :

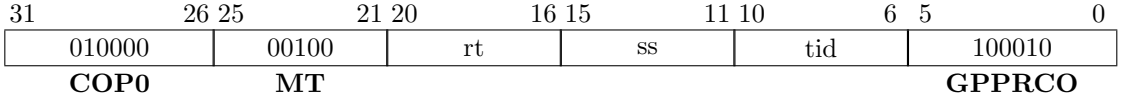
None

Overview :

Move a word from shared register to GPR. The Thread ID is contained to tid.

GPPR — Write Producer(General Purpose Register)

Synchronization Instruction — SYNC



Mnemonic:

GPPR rt, sd, tid

Function :

SHARE[sd] ← GPR[rt]

Exception :

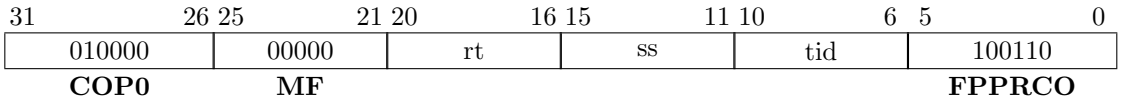
None

Overview :

Move a word to shared register from GPR. The Thread ID is contained to tid.

FPCO — Read Consumer(Floating-Point Register)

Synchronization Instruction SYNC



Mnemonic:

FPCO rt, ss, tid

Function :

FPR[rt] ← SHARE[ss]

Exception :

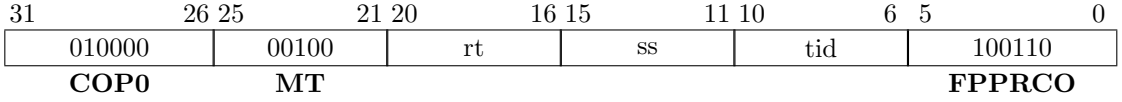
None

Overview :

Move a word from shared register to FPR. The Thread ID is contained to tid.

FPPR — Write Producer(Floating-Point Register)

Synchronization Instruction SYNC



Mnemonic:

FPPR rt, sd, tid

Function :

SHARE[sd] ← FPR[rt]

Exception :

None

Overview :

Move a word to shared register from FPR. The Thread ID is contained to tid.

BAR — Barrier

Synchronization Instruction — SYNC

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

Mnemonic:

BAR rt, sd

Function :

SHARE[sd] ← GPR[rt] + 1

Exception :

None

Overview :

—

PBAR — Pre Barrier

Synchronization Instruction — SYNC

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

Mnemonic:

PBAR sd

Function :

—

Exception :

None

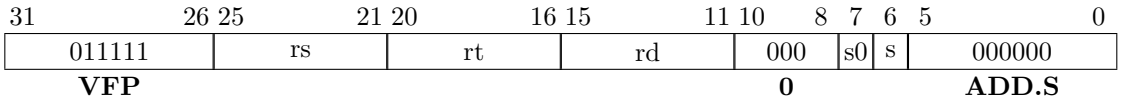
Overview :

—

3.3.8 Floating Point Vector Instruction

VADD.S — Vector Add Single

Vector Add — 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 :

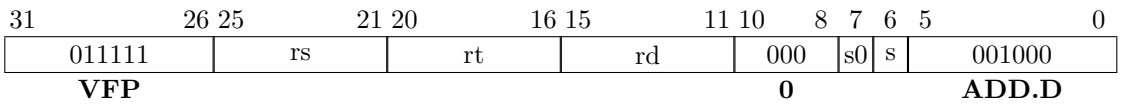
Vector Floating Point Exception

Overview :

Single-Precision Vector Add. When s0 is 1, execute operations using Scalar Register (SFPR[rt]) instead of VFPR[rt]. When sync is 1, suppress the speculative execution.

VADD.D — Vector Add Double

Vector Add 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 :

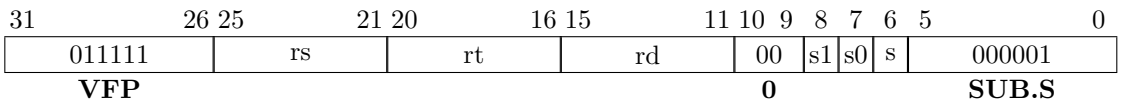
Vector Floating Point Exception

Overview :

Double-Precision Vector Add. When s0 is 1, execute operations using Scalar Register (SFPR[rt]) instead of VFPR[rt]. When sync is 1, suppress the speculative execution.

VSUB.S — Vector Subtract Single

Vector Substract — 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 :

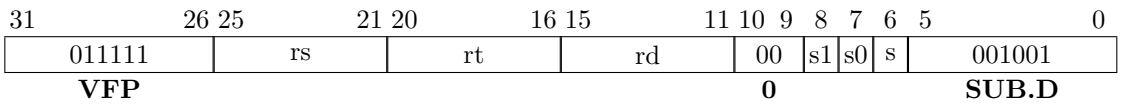
Vector Floating Point Exception

Overview :

Single-Precision Vector Substract. When s0 is 1, execute operations using Scalar Register (SFPR[rt]) instead of VFPR[rt]. When s1 is 1, execute operations using Scalar Register (SFPR[rs]) instead of VFPR[rs]. When sync is 1, suppress the speculative execution.

VSUB.D — Vector Subtract Double

Vector Substract 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] $\leftarrow$ VFPR[rs] $-$ VFPR[rt]

Exception :

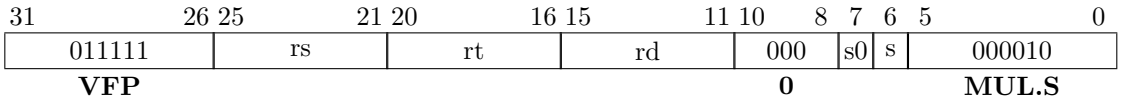
Vector Floating Point Exception

Overview :

Double-Precision Vector Substract. When s0 is 1, execute operations using Scalar Register (SFPR[rt]) instead of VFPR[rt]. When s1 is 1, execute operations using Scalar Register (SFPR[rs]) instead of VFPR[rs]. When sync is 1, suppress the speculative execution.

VMUL.S — Vector Multiply Single

Vector Multiply 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 :

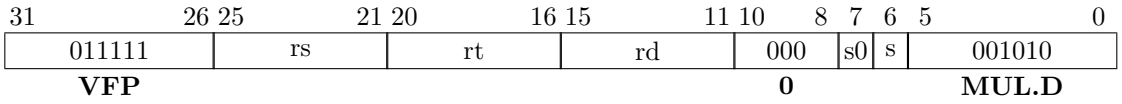
Vector Floating Point Exception

Overview :

Single-Precision Vector Multiply. When s0 is 1, execute operations using Scalar Register (SFPR[rt]) instead of VFPR[rt]. When sync is 1, suppress the speculative execution.

VMUL.D — Vector Multiply Double

Vector Multiply 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 :

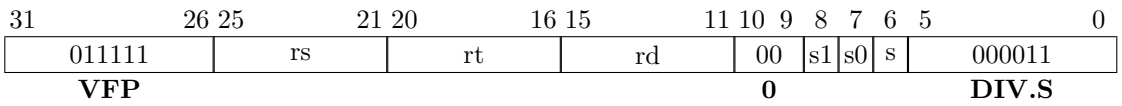
Vector Floating Point Exception

Overview :

Double-Precision Vector Multiply. When s0 is 1, execute operations using Scalar Register (SFPR[rt]) instead of VFPR[rt]. When sync is 1, suppress the speculative execution.

VDIV.S — Vector Divide Single

Vector Divide — 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 :

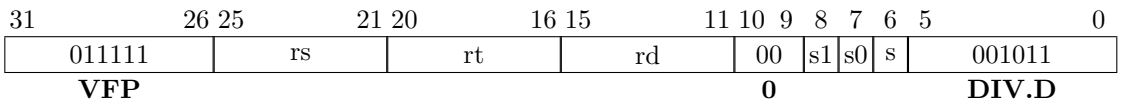
Vector Floating Point Exception

Overview :

Single-Precision Vector Divide. When s0 is 1, execute operations using Scalar Register (SFPR[rt]) instead of VFPR[rt]. When s1 is 1, execute operations using Scalar Register (SFPR[rs]) instead of VFPR[rs]. When sync is 1, suppress the speculative execution.

VDIV.D — Vector Divide Double

Vector Divide 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] $\leftarrow$ VFPR[rs] $\div$ VFPR[rt]

Exception :

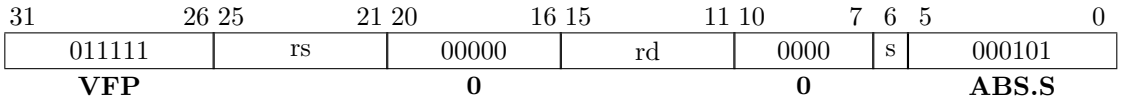
Vector Floating Point Exception

Overview :

Double-Precision Vector Divide. When s0 is 1, execute operations using Scalar Register (SFPR[rt]) instead of VFPR[rt]. When s1 is 1, execute operations using Scalar Register (SFPR[rs]) instead of VFPR[rs]. When sync is 1, suppress the speculative execution.

VABS.S Vector Absolute Single

Vector Absolute Operation VECTOR



Mnemonic:

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

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

Function :

VFPR[rd] ← | VFPR[rs] |

Exception :

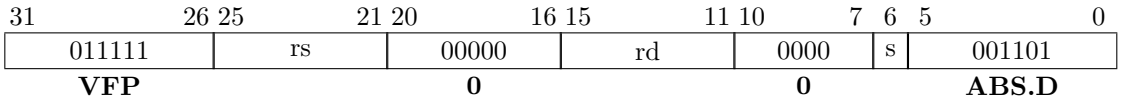
Vector Floating Point Exception

Overview :

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

VABS.D Vector Absolute Double

Vector Absolute Operation VECTOR



Mnemonic:

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

Function :

VFPR[rd] ← | VFPR[rs] |

Exception :

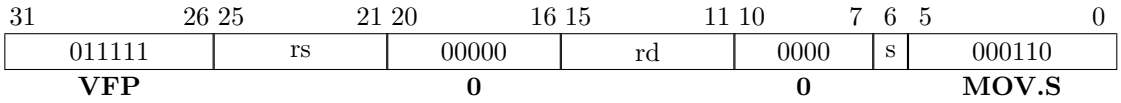
Vector Floating Point Exception

Overview :

Double-Precision Vector Absolute Operation. When sync is 1, suppress the speculative execution.

VMOV.S — Vector Move Single

Vector Move VECTOR



Mnemonic:

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

Function :

VFPR[rd] ← VFPR[rs]

Exception :

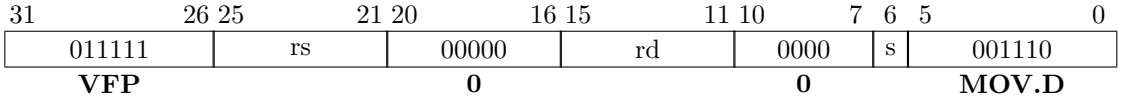
Vector Floating Point Exception

Overview :

Single-Precision Vector Move Instruction. When sync is 1, suppress the speculative execution.

VMOV.D — Vector Move Double

Vector Move VECTOR



Mnemonic:

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

Function :

VFPR[rd] ← VFPR[rs]

Exception :

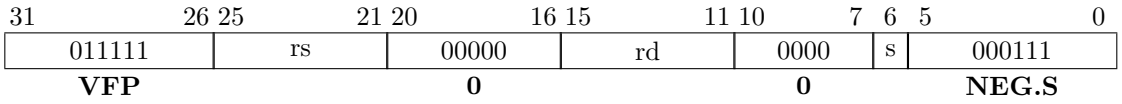
Vector Floating Point Exception

Overview :

Double-Precision Vector Move Instruction. When sync is 1, suppress the speculative execution.

VNEG.S — Vector Negate Single

Vector Negate VECTOR



Mnemonic:

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

Function :

VFPR[rd] ← −1 × VFPR[rs]

Exception :

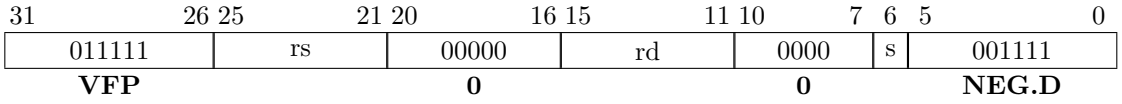
Vector Floating Point Exception

Overview :

Single-Precision Vector Negate Operation. When sync is 1, suppress the speculative execution.

VNEG.D — Vector Negate Double

Vector Negate VECTOR



Mnemonic:

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

Function :

VFPR[rd] ← −1 × VFPR[rs]

Exception :

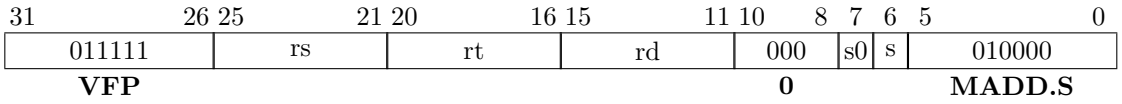
Vector Floating Point Exception

Overview :

Double-Precision Vector Negate Operation. When sync is 1, suppress the speculative execution.

VMADD.S Vector Multiply and Add Single

Vector Multiply and Add Operation 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 :

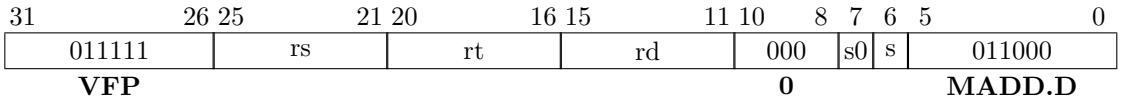
Vector Floating Point Exception

Overview :

Single-Precision Vector Multiply and Add Operation. When s0 is 1, execute operations using Scalar Register (SFPR[rt]) instead of VFPR[rt]. When sync is 1, suppress the speculative execution.

VMADD.D — Vector Multiply and Add Double

Vector Multiply and Add Operation — 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 :

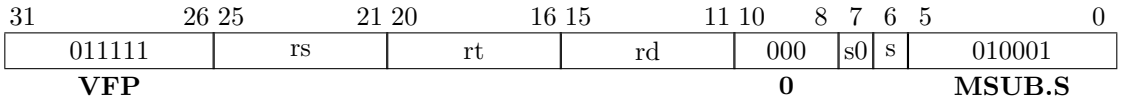
Vector Floating Point Exception

Overview :

Double-Precision Vector Multiply and Add Operation. When s0 is 1, execute operations using Scalar Register (SFPR[rt]) instead of VFPR[rt]. When sync is 1, suppress the speculative execution.

VMSUB.S Vector Multiply and Subtract Single

Vector Multiply and Substract Operation 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] $\leftarrow$ VFPR[rs] $\times$ VFPR[rt] $-$ VFPR[rd]

Exception :

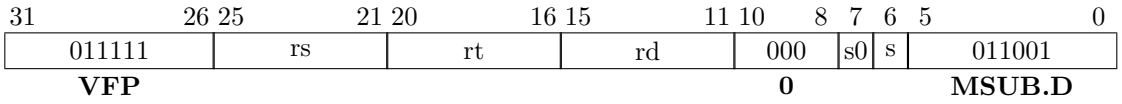
Vector Floating Point Exception

Overview :

Single-Precision Vector Multiply and Substract Operation. When s0 is 1, execute operations using Scalar Register (SFPR[rt]) instead of VFPR[rt]. When sync is 1, suppress the speculative execution.

VMSUB.D — Vector Multiply and Subtract Double

Vector Multiply and Substract Operation — 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] $\leftarrow$ VFPR[rs] $\times$ VFPR[rt] $-$ VFPR[rd]

Exception :

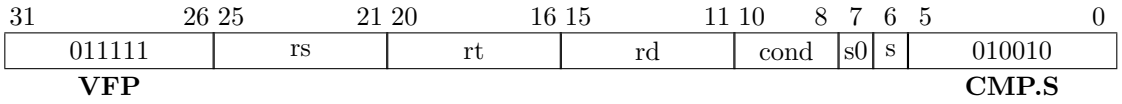
Vector Floating Point Exception

Overview :

Double-Precision Vector Multiply and Substract Operation. When s0 is 1, execute operations using Scalar Register (SFPR[rt]) instead of VFPR[rt]. When sync is 1, suppress the speculative execution.

VCMP.S — Vector Compare Single

Vector Compare 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 :

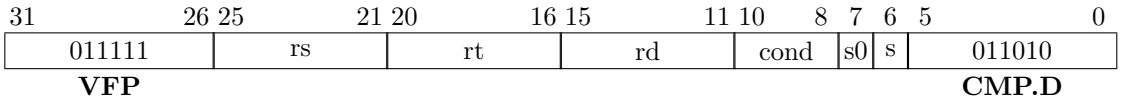
Vector Floating Point Exception

Overview :

Single-Precision Vector Compare Instruction. Determinate value of VFPR[rd] by condition. When s0 is 1, execute operations using 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($\geq$).

VCMP.D Vector Compare Double

Vector Compare 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 :

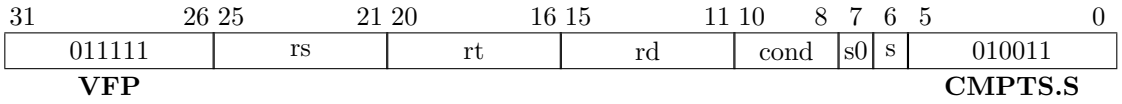
Vector Floating Point Exception

Overview :

Double-Precision Vector Compare Instruction. Determinate value of VFPR[rd] by condition. When s0 is 1, execute operations using 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.S Vector Compare Single to Scalar Register

Vector Compare 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 :

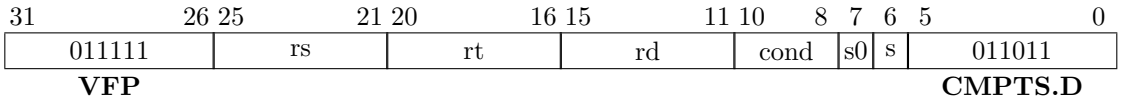
Vector Floating Point Exception

Overview :

Single-Precision Vector Compare Instruction. Operation result is assigned 1bit for each elements, and stored scalar register. When s0 is 1, execute operations using 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.D Vector Compare Double to Scalar Register

Vector Compare 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 :

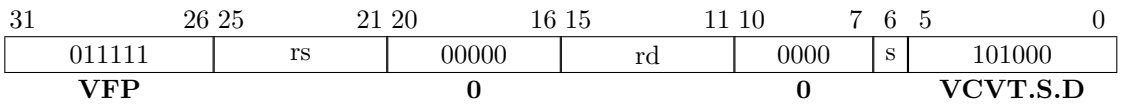
Vector Floating Point Exception

Overview :

Double-Precision Vector Compare Instruction. Operation result is assigned 1bit for each elements, and stored scalar register. When s0 is 1, execute operations using 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($\geq$).

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

Vector Format Converison 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 :

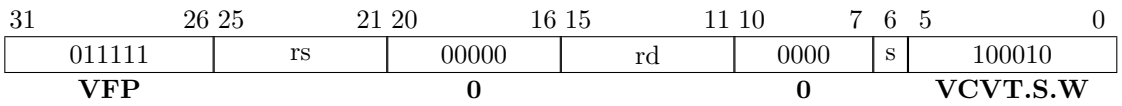
Vector Floating Point Exception

Overview :

Convert from double-precision format to single-precision format. When sync is 1, suppress the speculative execution.

VCVT.S.W — Vector Convert to Single from Word

Vector Format Conversion — 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 :

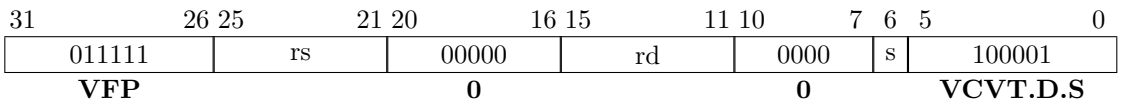
Vector Floating Point Exception

Overview :

Convert from integer format to single-precision format. When sync is 1, suppress the speculative execution.

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

Vector Format Conversion 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 :

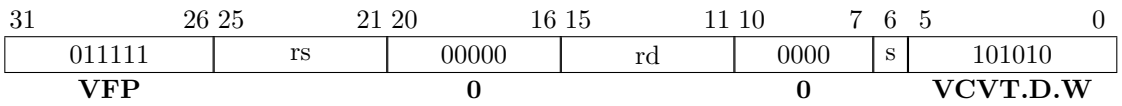
Vector Floating Point Exception

Overview :

Convert from single-precision format to double-precision format. When sync is 1, suppress the speculative execution.

VCVT.D.W — Vector Convert to Double from Word

Vector Format Conversion — 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 :

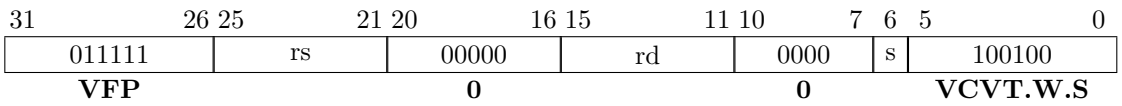
Vector Floating Point Exception

Overview :

Convert from integer format to double-precision format. When sync is 1, suppress the speculative execution.

VCVT.W.S — Vector Convert to Word from Single

Vector Format Conversion VECTOR



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 :

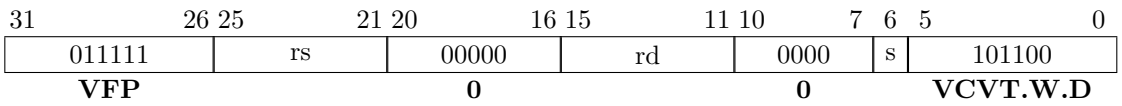
Vector Floating Point Exception

Overview :

Convert from single-precision format to integer format. When sync is 1, suppress the speculative execution.

VCVT.W.D — Vector Convert to Word from Double

Vector Format Conversion VECTOR



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 :

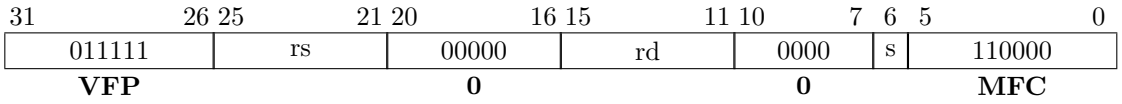
Vector Floating Point Exception

Overview :

Convert from double-precision format to integer format. When sync is 1, suppress the speculative execution.

VFMFC Move from Vector Floating Point Control Register

Control Register Read VECTOR



Mnemonic:

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

Function :

FPR[rd] ← VFCTRL[rs]

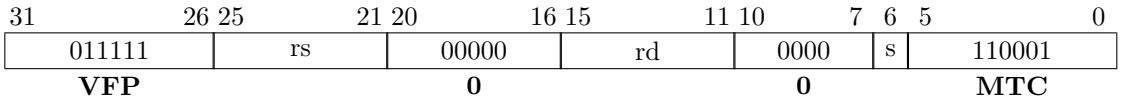
Exception :

Overview :

Floating Point Vector Control Register Read Instruction. This instruction store value of control register which rs specify into floating point register. When sync is 1, suppress the speculative execution.

VFMTC — Move to Vector Floating Point Control Register

Control Register Write — VECTOR



Mnemonic:

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

Function :

VFCTRL[rd] ← FPR[rs]

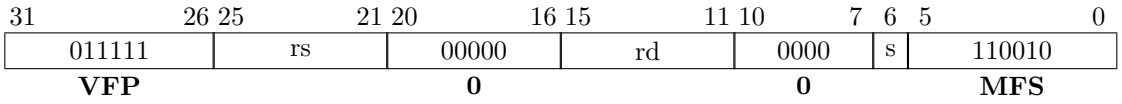
Exception :

Overview :

Floating Point Vector Control Register Write Instruction. This instruction store value of floating point register into control register which rd specify. When sync is 1, suppress the speculative execution.

VFMFS Move from Vector Floating Point Scalar Register

Floating Point Scalar Register Read VECTOR



Mnemonic:

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

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

Function :

FPR[rd] ← SFPR[rs]

Exception :

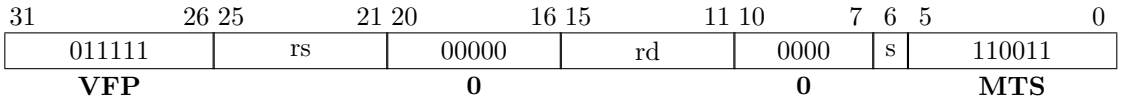
Vector Floating Point Exception

Overview :

Floating Point Scalar Register Read Instruction. This instruction store value of floating point scalar register which rs specify into floating point register. When sync is 1, suppress the speculative execution.

VFMTS Move to Vector Floating Point Scalar Register

Floating Point Scalar Register Write VECTOR



Mnemonic:

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

Function :

SFPR[rd] ← FPR[rs]

Exception :

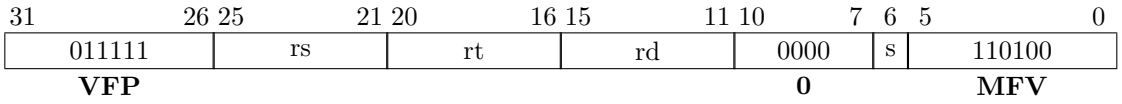
Vector Floating Point Exception

Overview :

Floating Point Scalar Register Write Instruction. This instruction store value of floating point register into floating point scalar register which rd specify. When sync is 1, suppress the speculative execution.

VFMFV Move from Vector Floating Point Vector Register

Floating Point Vector Register Read VECTOR



Mnemonic:

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

Function :

SFPR[rd] ← VFPR[rs][rt]

Exception :

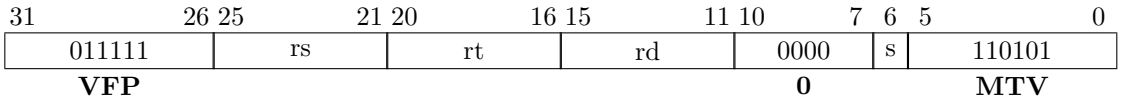
Vector Floating Point Exception

Overview :

Floating Point Vector Register Read Instruction. This instruction store value of rt-th element of floating point vector register which rs specify into floating point register. When sync is 1, suppress the speculative execution.

VFMTV Move to Vector Floating Point Vector Register

Floating Point Vector Register Write VECTOR



Mnemonic:

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

Function :

SFPR[rd] ← VFPR[rs][rt]

Exception :

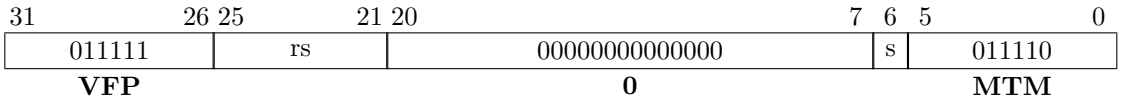
Vector Floating Point Exception

Overview :

Floating Point Vector Register Write Instruction. This instruction write value of floating point register into rt-th element of floating point vector register which rd specify. When sync is 1, suppress the speculative execution.

VFMTM Move to Vector Floating Point Mask Register

Floating Point Vector Mask Register Write VECTOR



Mnemonic:

VFMTM rs	(sync(s) = 0)
VFMTM.sy rs	(sync(s) = 1)

Function :

SFPR[rd] ← VFPR[rs][rt]

Exception :

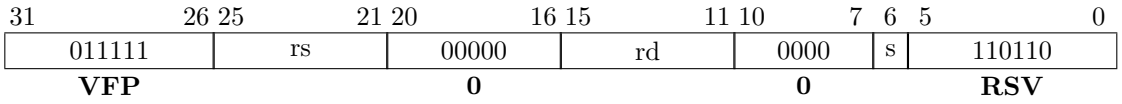
Vector Floating Point Exception

Overview :

Floating Point Vector Mask Register Write Instruction. This instruction store value of floating point scalar register which rd specify into mask register. When sync is 1, suppress the speculative execution.

VFRSV Vector Floating Point Register Reserve

Floating Point Vector Register Reservation 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 :

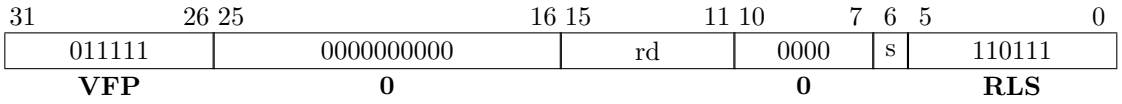
None

Overview :

Floating Point Vector Register Reservation Instruction. Specify configuration of register which reserve at GPR[rs]. If reservaion is succeeded, store 1 into GPR[rd]. If reservation is failed, store 0 into GPR[rd]. When sync is 1, suppress the speculative execution.

VFRLS Vector Floating Point Register Release

Floating Point Vector Register Release 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 :

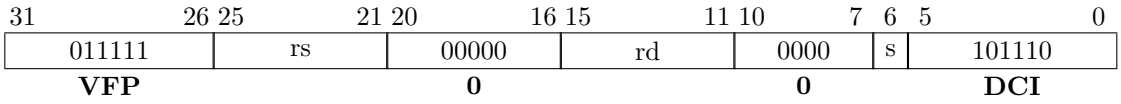
None

Overview :

Floating Point Vector Register Release Instruction. If release is succeeded, store 1 into GPR[rd]. If release is failed, store 0 into GPR[rd]. When sync is 1, suppress the speculative execution.

VFDCI Vector Floating Point Define Compound Instruction

Floating Point Vector Compound Instruction Definition VECTOR



Mnemonic:

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

Function :

VFCPD[rd] ← GPR[rs]

Exception :

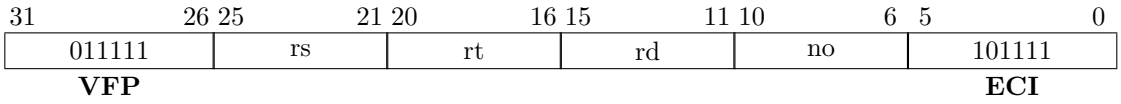
Vector Floating Point Exception

Overview :

Define Floating Point Vector Compound Instruction. Store instuction defined GPR[rs] into address of compound instruction buffer specified rd. When sync is 1, suppress the speculative execution.

VFECI Vector Floating Point Execute Compound Instruction

Floating Point Vector Compound Instruction Execution VECTOR



Mnemonic:

VFDCI rd, rs, rt, no

Function :

VFPR[rd] ← VFPR[rs] op VFPR[rt]

Exception :

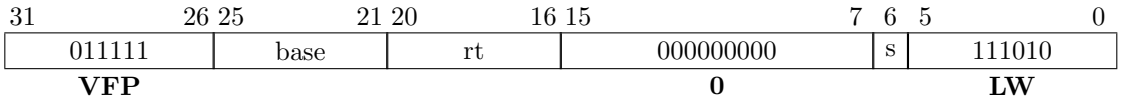
Vector Floating Point Exception

Overview :

Execute Floating Point Vector Compound Instruction. Execute instruction from address of compound instruction buffer specified no.

VFLW — Vector Floating Point Load Word

Floating Point Vector Load — VECTOR



Mnemonic:

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

Function :

VFPR[rt] ← MEM.WORD[GPR[base]]

Exception :

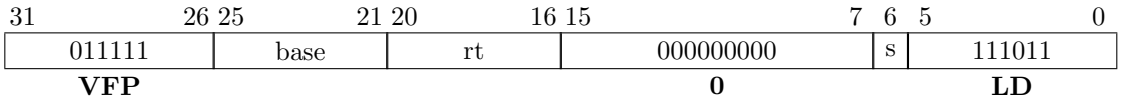
Vector Floating Point Exception
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Load)

Overview :

Load from memory to floating point vector register by the word. When sync is 1, suppress the speculative execution.

VFLD — Vector Floating Point Load Double

Floating Point Vector Load — VECTOR



Mnemonic:

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

Function :

VFPR[rt] ← MEM.DWORD[GPR[base]]

Exception :

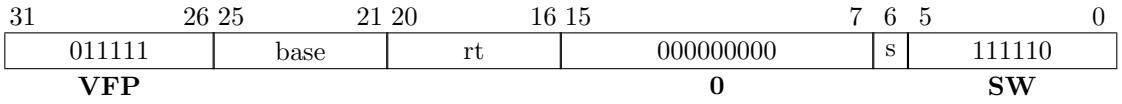
Vector Floating Point Exception
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Load)

Overview :

Load from memory to floating point vector register by the double word. When sync is 1, suppress the speculative execution.

VFSW — Vector Floating Point Store Word

Floating Point Vector Store — VECTOR



Mnemonic:

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

Function :

MEM.WORD[GPR[base]] ← VFPR[rt]

Exception :

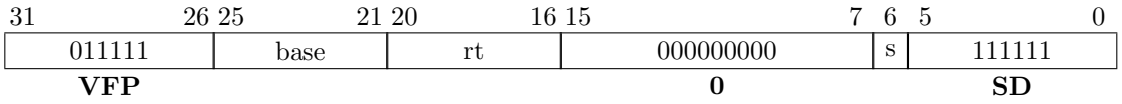
Vector Floating Point Exception
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Store)

Overview :

Store from floating point vector register to memory by the word. When sync is 1, suppress the speculative execution.

VFSD — Vector Floating Point Store Double

Floating Point Vector Store — VECTOR



Mnemonic:

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

Function :

MEM.DWORD[GPR[base]] ← VFPR[rt]

Exception :

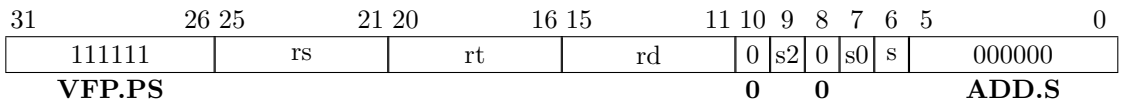
Vector Floating Point Exception
D-TLB No Entry Matched
D-TLB Protection Error
Data Address Miss Align (Store)

Overview :

Store from floating point vector register to memory by the double word. When sync is 1, suppress the speculative execution.

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 :

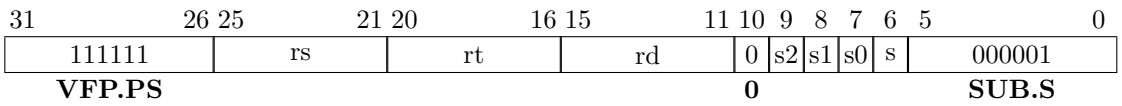
Vector Integer Exception

Overview :

32bitx2, Single-Precision Vector Add. When s0 is 1, execute operation using scalar register (SFPR[rt]) instead of VFPR[rt]. When s2 is 1, execute operation using 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 :

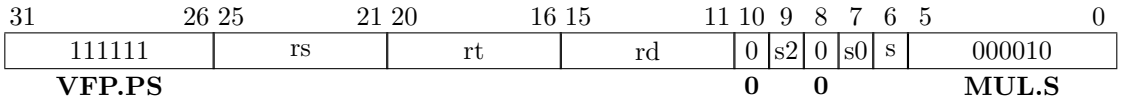
Vector Integer Exception

Overview :

32bitx2, Single-Precision Vector Substract. When s0 is 1, execute operation using scalar register (SFPR[rt]) instead of VFPR[rt]. When s1 is 1, execute operation using scalar register (SFPR[rs]) instead of VFPR[rs]. When s2 is 1, execute operation using 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 :

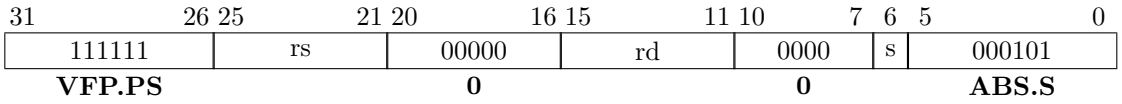
Vector Integer Exception

Overview :

32bitx2, Single-Precision Vector Multiply. When s0 is 1, execute operation using scalar register (SFPR[rt]) instead of VFPR[rt]. When s2 is 1, execute operation using 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 :

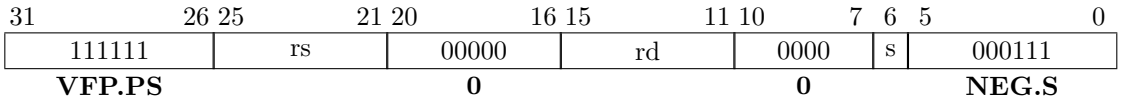
Vector Integer Exception

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 :

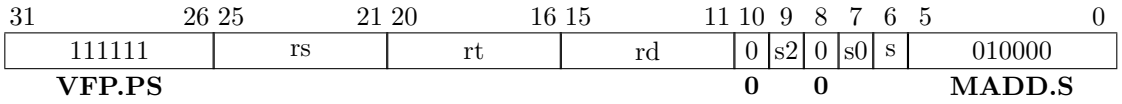
Vector Integer Exception

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 :

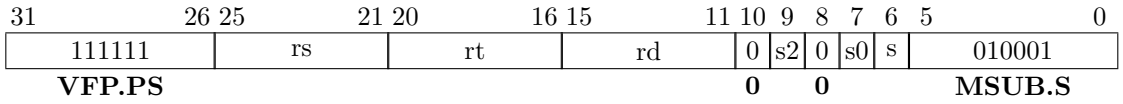
Vector Integer Exception

Overview :

32bitx2, Single-Precision Vector Multiply and Add Operation. When s0 is 1, execute operation using scalar register (SFPR[rt]) instead of VFPR[rt]. When s2 is 1, execute operation using 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 :

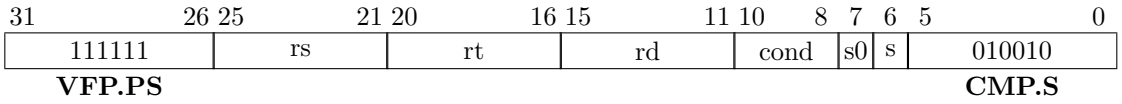
Vector Integer Exception

Overview :

32bitx2, Single-Precision Vector Multiply and Substract Operation. When s0 is 1, execute operation using scalar register (SFPR[rt]) instead of VFPR[rt]. When s2 is 1, execute operation using 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 :

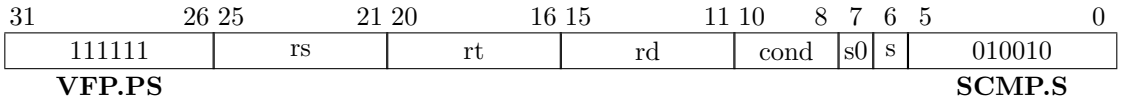
Vector Integer Exception

Overview :

32bitx2, Single-Precision Vector Compare Instruction. Determinate value of VFPR[rd] using condition. When s0 is 1, execute operation using scalar register (SFPR[rt]) instead of VFPR[rt]. When sync is 1, suppress the speculative execution. Choose one for cond: 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 :

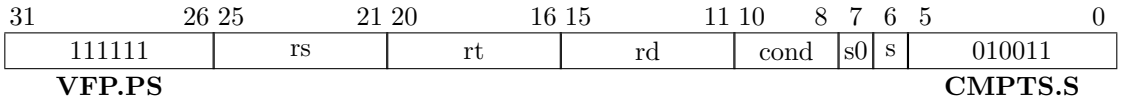
Vector Integer Exception

Overview :

32bitx2, Single-Precision Vector Compare Instruction. Determinate value of VFPR[rd] using condition. Execute operation using lower 32bit of VFPR[rt]. When s0 is 1, execute operation using scalar register (SFPR[rt]) instead of VFPR[rt]. When sync is 1, suppress the speculative execution. Choose one for cond: 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 :

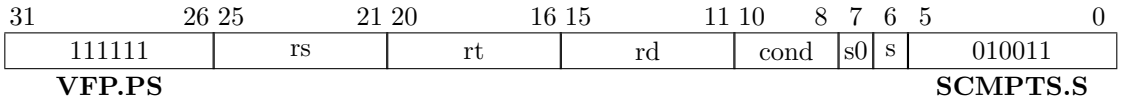
Vector Integer Exception

Overview :

32bitx2, Single-Precision Vector Compare Instruction. Operation result is assigned 1bit for each elements, and saved into scalar register. When s0 is 1, execute operation using scalar register (SFPR[rt]) instead of VFPR[rt]. When sync is 1, suppress the speculative execution. Choose one for cond: 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 saved into scalar register. Execute operation using lower 32bit of VFPR[rt]. When s0 is 1, execute operation using scalar register (SFPR[rt]) instead of VFPR[rt]. When sync is 1, suppress the speculative execution. Choose one for cond: eq(=), gt(>), lt(<), ne($\neq$), le($\leq$), ge($\geq$).

4

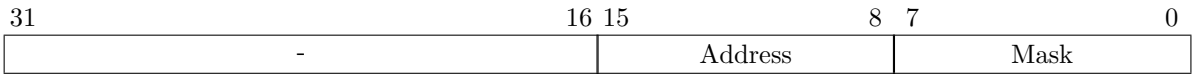
Address Decoder

4.1 Register Interface

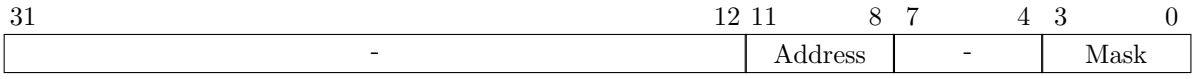
Each of the address decoder configuration registers are defined as a system register. Therefore, to configure, mtc0 instruction is used.

Values are different by a module.

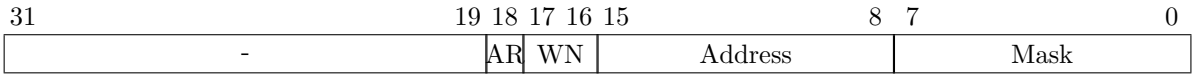
- Standard (TYPE A)



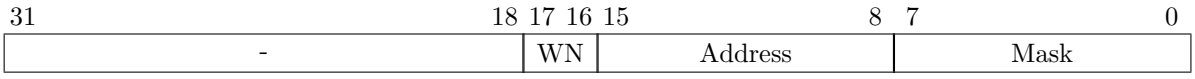
- I/O (TYPE B)



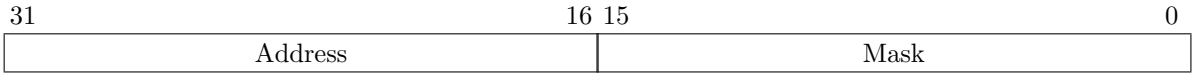
- External Bus (TYPE C)



- 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

System Registers

System Registers are accessed by using MFC0 and MTC0 instructions. Specify the register number you want to access as rd.

5.1 Register Map

offset	31 24 23 16 15 8 7 0
0x00	Status Register (Thread0)
0x01	Status Register (Thread1)
0x02	Status Register (Thread2)
0x03	Status Register (Thread3)
0x04	Status Register (Thread4)
0x05	Status Register (Thread5)
0x06	Status Register (Thread6)
0x07	Status Register (Thread7)
0x08	Thread Table Register (Thread0)
0x09	Thread Table Register (Thread1)
0x0a	Thread Table Register (Thread2)
0x0b	Thread Table Register (Thread3)
0x0c	Thread Table Register (Thread4)
0x0d	Thread Table Register (Thread5)
0x0e	Thread Table Register (Thread6)
0x0f	Thread Table Register (Thread7)
0x10	Thread ID Register (Thread0)
0x11	Thread ID Register (Thread1)
0x12	Thread ID Register (Thread2)
0x13	Thread ID Register (Thread3)
0x14	Thread ID Register (Thread4)
0x15	Thread ID Register (Thread5)
0x16	Thread ID Register (Thread6)
0x17	Thread ID Register (Thread7)

offset	31 24 23 16 15 8 7 0
0x18	Instruction Count Register (Thread0)
0x19	Instruction Count Register (Thread1)
0x1a	Instruction Count Register (Thread2)
0x1b	Instruction Count Register (Thread3)
0x1c	Instruction Count Register (Thread4)
0x1d	Instruction Count Register (Thread5)
0x1e	Instruction Count Register (Thread6)
0x1f	Instruction Count Register (Thread7)
0x20	Count Register (Thread0)
0x21	Count Register (Thread1)
0x22	Count Register (Thread2)
0x23	Count Register (Thread3)
0x24	Count Register (Thread4)
0x25	Count Register (Thread5)
0x26	Count Register (Thread6)
0x27	Count Register (Thread7)
0x28	Compare Register (Thread0)
0x29	Compare Register (Thread1)
0x2a	Compare Register (Thread2)
0x2b	Compare Register (Thread3)
0x2c	Compare Register (Thread4)
0x2d	Compare Register (Thread5)
0x2e	Compare Register (Thread6)
0x2f	Compare Register (Thread7)
0x30	Floating-Point Control Register (Thread0)
0x31	Floating-Point Control Register (Thread1)
0x32	Floating-Point Control Register (Thread2)
0x33	Floating-Point Control Register (Thread3)
0x34	Floating-Point Control Register (Thread4)
0x35	Floating-Point Control Register (Thread5)
0x36	Floating-Point Control Register (Thread6)
0x37	Floating-Point Control Register (Thread7)
0x38	Issue Mode Register
0x39	CPU Count Register (Low)
0x3a	CPU Count Register (High)
0x3b ～ 0x47	MMU Register
0x48	Exception PC Register (Thread0)
0x49	Exception PC Register (Thread1)
0x4a	Exception PC Register (Thread2)
0x4b	Exception PC Register (Thread3)
0x4c	Exception PC Register (Thread4)
0x4d	Exception PC Register (Thread5)
0x4e	Exception PC Register (Thread6)
0x4f	Exception PC Register (Thread7)
0x50	Exception Cause Register (Thread0)
0x51	Exception Cause Register (Thread1)
0x52	Exception Cause Register (Thread2)
0x53	Exception Cause Register (Thread3)
0x54	Exception Cause Register (Thread4)
0x55	Exception Cause Register (Thread5)
0x56	Exception Cause Register (Thread6)
0x57	Exception Cause Register (Thread7)

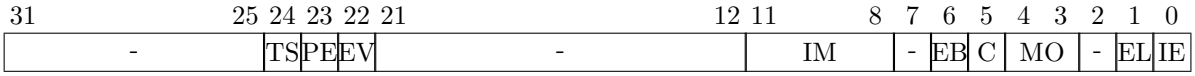
offset	31 24 23 16 15 8 7 0
0x58	Interruption Wait Register (Thread0)
0x59	Interruption Wait Register (Thread1)
0x5a	Interruption Wait Register (Thread2)
0x5b	Interruption Wait Register (Thread3)
0x5c	Interruption Wait Register (Thread4)
0x5d	Interruption Wait Register (Thread5)
0x5e	Interruption Wait Register (Thread6)
0x5f	Interruption Wait Register (Thread7)
0x60	External Interruption Level Register (Thread0)
0x61	External Interruption Level Register (Thread1)
0x62	External Interruption Level Register (Thread2)
0x63	External Interruption Level Register (Thread3)
0x64	External Interruption Level Register (Thread4)
0x65	External Interruption Level Register (Thread5)
0x66	External Interruption Level Register (Thread6)
0x67	External Interruption Level Register (Thread7)
0x68 ～ 0x69	-
0x6a	Exception Base Address Register
0x6b ～ 0x6f	-
0x70 ～ 0x73	Event Link In Register
0x74 ～ 0x77	Event Link Out Register
0x80～0x84	Instruction Cache Control Register
0x86～0x8c	Data Cache Control Register
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
0xfa	Multiplexer Watchdog Timer 32bit Bus Mode
0xfb	Multiplexer Watchdog Timer 32bit Bus Reset
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
0x188	Multiplexer Error Handler Extbus0
0x189	Multiplexer Error Handler Extbus1
0x18a	Multiplexer Error Handler Extbus2
0x18b	Multiplexer Error Handler Extbus3

5.1.1 Status Register

Address: 0x00 ∼ 0x07 (For Each Thread)
Show the thread status for each thread. the initial value is 0x00000000.

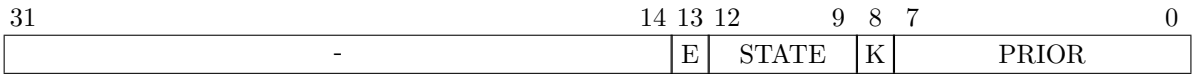


Field Name	Function
TS	Timer Start 1: Start the timer 0: Stop the timer
PE	Period 1: Generate Timer Interrupt periodically. 0: One Shot
EV	Exception Vector Location. 1: Bootstrap ··· When an exception occurs, control is transferred to the booting exception vector. 0: Normal ··· When an exception occurs, control is transferred to the normal exception vector.
IM	Interruption Mask. Mask corresponding Interruptions by setting 1. 11: Tiemr Interruption 10: Hardware Interruption 9: Software Interruption1 8: Software Interruption0
EBAE (EB)	Exception Base Address Enable -3: Variable ··· Use the exception vector where is besed on TBA(Table Base Address). 0: Fixed ··· Use the exception vector where address is fixed.
CRAM (C)	Control Register Access Mode 0: Precise ··· Force to wait for issuing access instructions to the control register until commitng the last instruction.
MO	Mode Bit 00: Kernel Mode 01: Superviser Mode 10: User Mode
EL	Exception Level. Set to 1 when an exception occurs. set to 0 by ERET instruction.
IE	Interruption Mode 0: Interruption Disable 1: Interruption Enable

5.1.2 Thread Table Register

Address: 0x08 ~ 0x0f (For Each Thread)

Showing the thread status. Basically change by using a thread control instruction. It can read and write, but there is no guarantee when writing forcefully. The default is 0x00000000 except 0x08.



Field Name	Function
E	Thread Enable 0: The context is not assigned with an active theread. 1: The context is assigned with an active thread.
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: normal 1: Make insturctions which evacuate the thread to a context cache disable.
PRIOR	Thread Priority. 256 Level.

5.1.3 Thread ID Register

Address: 0x10 ~ 0x17 (For Each Thread)

Field Name	Function
Thread ID	Use to specify the thread to access.

5.1.4 Instruction Counter Register

Address: 0x18 ~ 0x1f (For Each Thread)

31	0
Instruction Counter	

Field Name	Function
Instruction Counter	Count the total number of commited instruction from generating the thread.

5.1.5 Count Register

Address: 0x20 ～ 0x27 (For Each Thread)

Field Name	Function
Count	A counter which counts up every clock. Clear to 0 when the value is equal to Compare Register.

5.1.6 Compare Register

Address: 0x28 ～ 0x2f (For Each Thread)

31	0
Compare	

Field Name	Function
Compare	Generate a timer interruption when setting to non-zero and the value of Count Register is equal to this register. Timer interruptions are set to enable or disable by IM and IE field.

5.1.7 Floating-Point Control Register

Address: 0x30 ～ 0x37 (For Each Thread)

31	6	5	4	3	0
-		RND		EM	

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 Mask corresponding interruptions by setting to 1 3: Inexact Exception 2: Underflow Exception 1: Overflow Exception 0: Invalid Exception

5.1.8 Issue Mode Register

Address: 0x38

Register to set mode which selects issued instructions. the default is 0x00000000.

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)	Issuing method assigning thread to issue slot(TH_ASSIGN). Issue instructions from the specified thread by this field when SUB_POLICY field is set to SUB_FIX and the mainthread for each slot cannot issue an instruction,
Main Assign0, 1, 2, 3 (MA0, 1, 2, 3)	Issuing method assigning thread to issue slot(TH_ASSIGN). Specify the main thread for each slot.
Sub Policy (SP)	Set the sub policy of the instruction issuing selection policy. • 1INST_1TH 00: NORMAL 01: PRED_STOP ··· When the next intruction has a possibility to be canceled because of a branch prediction, make the thread priority decrease. 10: MINST_STOP ··· When the re-order buffer of a thread is filled over than half, make the thread priority decrease. • TH_ASSIGN 00: SUB_PRIOR ··· When there is no issuable instruction in the main thread of the slot, issue an instruction from the highest priority thread which is not assigned to the slot. 01: SUB_FIX ··· When there is no issuable instruction in the main thread of the slot, issue an instruction from the sub thread. When there is no issuable instruction in the sub thread neigher, be an empty slot.
Policy (PO)	Set the issue selection policy. 00: 1INST_1TH ··· In every clock cycle, the only one instruction is issuable for each thread. It means there is empty slots if the number of executing thread is less than 4. 01: HIGHEST_FAST ··· In every clock cycle, issue instructions as many as possible from the highest priority thread, and remaind slots are assigned to lower priority threads. 10: TH_ASSIGN ··· each slot is assigned to the fixed thread. Issue an instruction of other thread only if the assigned thread cannot issue an instruction.

5.1.9 CPU Count Register

Address: 0x39, 0x3a

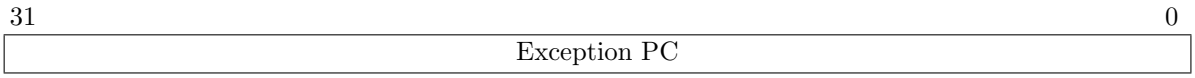
Field Name	Function
CPU Count	64 bit counter Counting up every clock after reset. 0x39 indicates lower 32 bit and 0x3a is upper 32 bit.

5.1.10 MMU Register

Address: 0x3b ~ 0x47
Control register about MMU

5.1.11 Exception PC Register

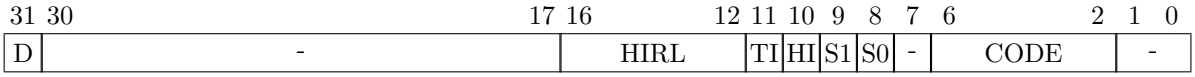
Address: 0x48 ~ 0x4f (For Each Thrad)



Field Name	Function
Exception PC	Keep PC which is the last commit when an exception occurs. It is refered as a return address from an exeception handling by ERET instruction

5.1.12 Exception Cause Register

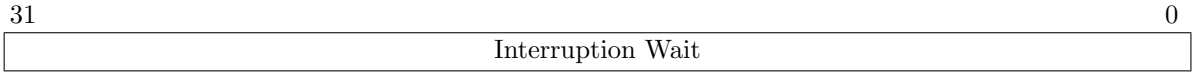
Address: 0x50 ~ 0x57 (For Each Thread)
Keep information about the occured exception. The default is 0x00000000.



Field Name	Function
Delay Bit (D)	Set to 1 if the instruction causing an exeption is in Delay Slot.
Hardware Interruption Level (HIRL)	Level of external interruption
Timer Interruption Pending (TI))	Written 1 when expiration of the timer. The task which was generated by a timer interruption must clear this bit manually.
HI	Hardware Interruption Pending
S1	Software Interruption 1 Pending
S0	Software Interruption 0 Pending
Exception Code (CODE)	Keep the exception code that happened the last.

5.1.13 Interruption Wait Register (For Each Thread)

Address: 0x58 ～ 0x5f



Field Name	Function
Interruption Wait	Each bit corresponds with IRL level. When the bit is 1, a thread accepts the corresponding external interruption.

5.1.14 External Interruption Level Register (For Each Thread)

Address: 0x60 ～ 0x67

Field Name	Function
External Interruption Level	Keep IRL of the external interrupton which was inputed the last from IRC.

5.1.15 Interruption Pending Register

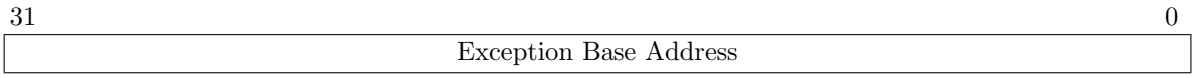
Address: 0x68
Currently Unused (probably).

5.1.16 Interruption Clear Register

Address: 0x69
Currently Unused (probably).

5.1.17 Exception Base Address Register

Address: 0x6a



Field Name	Function
Exception Base Address	Keep the base address of the exeption vector. By setting EBAE bit of Status Register to 1, control is transferred to the address obtained by adding the offset according to the contents of the exception to the EBA.

5.1.18 Event Link In Register

Address: 0x70 ~ 0x73

5.1.19 Event Link Out Register

Address: 0x74 ~ 0x77

5.1.20 Instruction Cache Control Register

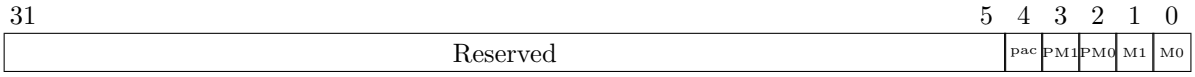
Address: 0x80~0x84
??Refer Control Register at the section(CACHE).

5.1.21 Data Cache Control Register

Address: 0x86~0x8c
??Refer Control Register at the section(CACHE).

5.1.22 Multiplexer Arbitor Mode Bus

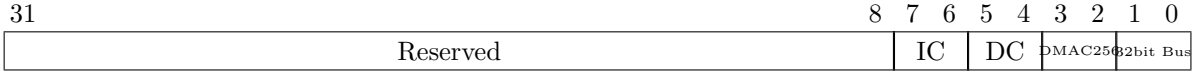
Address: 0x90



Field Name	Function
M0(256bit arbiter mode)	Set Bus Arbitration Mode. 0: Fixed Priority 1: Round Robin Grants bus master control of bus.
M1(32bit arbiter mode)	Set Bus Arbitration Mode 0: Fixed Priority 1: Round Robin Grants bus master control of bus.

5.1.23 Multiplexer Arbitor Priorty 256bit Bus

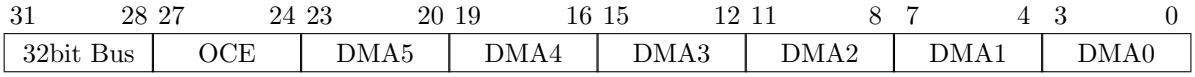
Address: 0x91



Sets 256bit bus's bus master priority. default is set to 0 (lowest priority)

5.1.24 Multiplexer Arbitor Priorty High 32bit Bus

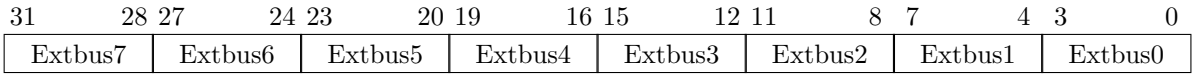
Address: 0x92



Sets 32bit bus's bus master priority. default is set to 0 (lowest priority)

5.1.25 Multiplexer Arbitor Priorty Low 32bit Bus

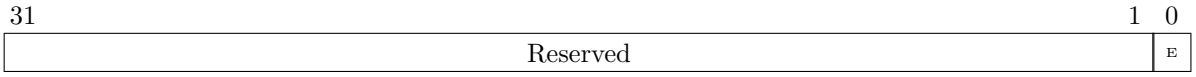
Address: 0x93



Sets 32bit bus's bus master priority. default is set to 0 (lowest priority)

5.1.26 Multiplexer Watchdog Timer 256bit Bus Enable

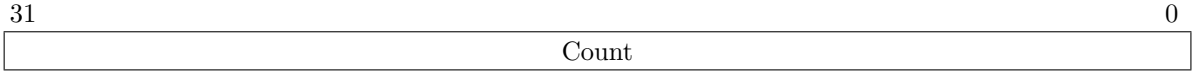
Address: 0x94



Field Name	Function
E	Default: 0 0: Disable 1: Enable Set both registers of Watchdog Timer 256bit Count and this register to enable settings.

5.1.27 Multiplexer Watchdog Timer 256bit Bus Count

Address: 0x95



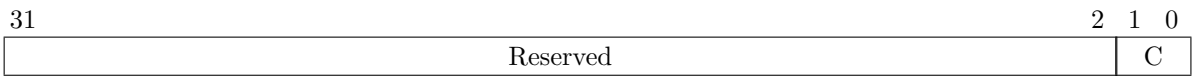
Field Name	Function
Count	Default: 0 If the time it takes for a ready signal to return after an address strobe signal is emitted exceeds the set clock cycles, a Watchdog Timer Error interruption occurs. If Watchdog Timer Enable is not set it is disabled.

5.1.28 Multiplexer Error Handler State 256bit Bus

Address: 0x96

Indicates error location.

For error details refer error handle registers.



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

5.1.29 Multiplexer Error Handler State 32bit Bus

Address: 0x97

5.1.30 Multiplexer Error Handler Instruction Cache

Address: 0x98

Field Name	Function
SE	Inddicates Error Type. Default: 0 00: No Error 01: Reserved 10: Address Error (Invalid Address) 11: Watchdog Timer Error
ES	Indicates Error Slave. 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

5.1.31 Multiplexer Error Handler Data Cache

Address: 0x99

Same as Section5.1.30 Multiplexer Error Handler Instruction Cache.

5.1.32 Multiplexer Error Handler Bus Interface Unit

Address: 0x9e

Same as Section5.1.30 Multiplexer Error Handler Instruction Cache.

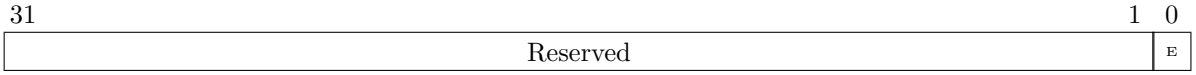
5.1.33 Multiplexer Error Handler MDMAC256

Address: 0x9f

Same as Section5.1.30 Multiplexer Error Handler Instruction Cache.

5.1.34 Multiplexer Watchdog Timer 32bit Bus Enable

Address: 0xb9



Field Name	Function
E	Default: 0 0: Disable 1: Enable Set both this register and Watchdog Timer 32bit Count to validate.

5.1.35 Multiplexer Error Handler Master32

Address: 0xba

Same as Section5.1.30 Multiplexer Error Handler Instruction Cache.

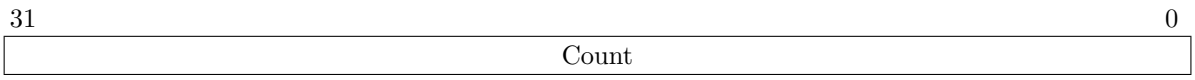
5.1.36 Multiplexer Error Handler Master256

Address: 0xbb

Same as Section5.1.30 Multiplexer Error Handler Instruction Cache.

5.1.37 Multiplexer Watchdog Timer 32bit Bus Count

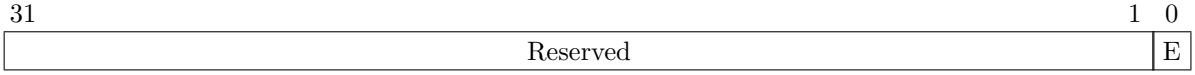
Address: 0xbc



Field Name	Function
Count	Default: 0 If the time(clock cycles) between Address Strobe enable and Ready returning from slave on 32bit bus, transcends clock cycle written in this register, a Watchdog Timer Error Interrupt occurs. Only Enable when watchdog timer enable is set.

5.1.38 Reservation Station Aging

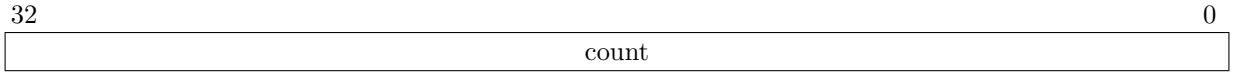
Address: 0xc9



Aging of Reservation Station entry.

5.1.39 Reservation Station Aging Increment

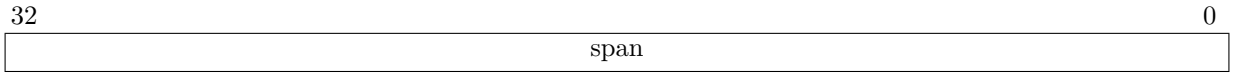
Address: 0xca



Aging Increment amount of Reservation Station entry.

5.1.40 Reservation Station Aging Span

Address: 0xcb

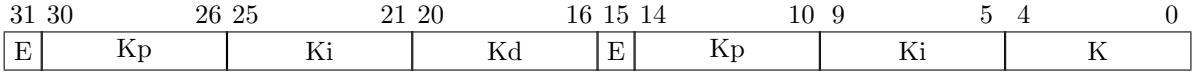


Aging Span of Reservation Station Span.

5.1.41 PID Parameter Register

Address: 0xcc ∼ 0xcf (For Each 2 Thread)

Change proportional gain, integral gain and differential gain of PID control. There is no point if not using PID control in Target IPC register. from 31st to 16th bit are for odd threads, and from 15th to 0th bit are for even.

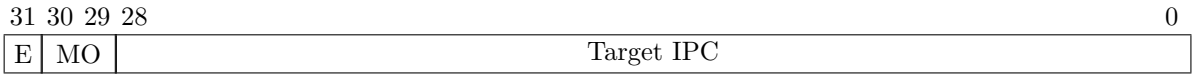


Field Name	Function
E	PID Parameter Enable 0: Use gain set by hardware. 1: set any gain by software.
Kp, Ki, Kd	10000: Shift logical right (5 bit)($\frac{1}{32}$) 01000: Shift logical right (4 bit)($\frac{1}{16}$) 00100: Shift logical right (3 bit)($\frac{1}{8}$) 00010: Shift logical right (2 bit)($\frac{1}{4}$) 00001: Shift logical right (1 bit)($\frac{1}{2}$) When set several bits, make sum after shifting. Example: If setting to 10011, then the gain is $\frac{1}{32} + \frac{1}{4} + \frac{1}{2} = \frac{25}{32}$.

5.1.42 Target IPC Register

Address: 0xd0 ∼ 0xd7 (For Each Thread)

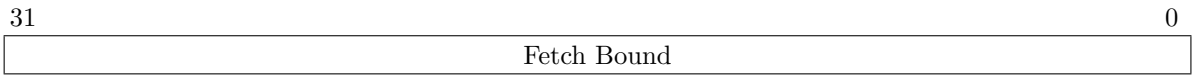
Make IPC control enable. To use IPC control, set Compare Register.



Field Name	Function
E	IPC Enable 0: not use IPC control. 1: use IPC control
MO	Control method used for IPC control 00: PID control 01: Feedforward control otherwise: PDI control
Target IPC	Specify the target IPC (actually set the number of instruction for clockcycles specified in Compare Register). Example: When Compare Register is set to 10000, and the target IPC is 0.7, set to 7000.

5.1.43 Fetch Bound Register

Address: 0xd8 ∼ 0xdf (For Each 2 Thread)



Field Name	Function
Fetch Bound	The upper bound of the number of fetch.

5.1.44 Own Status Register

Address: 0xe0

Refer Status Register of own context number.

5.1.45 Own Thread Table Register

Address: 0xe1

Refer Thread Table Register of own context number.

5.1.46 Own Thread ID Register

Address: 0xe2

Refer Thread ID Register of own context number.

5.1.47 Own Instruction Count Register

Address: 0xe3

Refer Instuction Count Register of own context number.

5.1.48 Own Count Register

Address: 0xe4

Refer Count Register of own context number.

5.1.49 Own Compare Register

Address: 0xe5

Refer Compare Register of own context number.

5.1.50 Own Floating-Point Control Register

Address: 0xe6

Refer Floating-Point Control Register of own context number.

5.1.51 Own Bad Virtual Address Register

Address: 0xe7
Refer Bad Virtual Address Register of own context number.

5.1.52 Own Exception PC Register

Address: 0xe8
Refer Exception PC Register of own context number.

5.1.53 Own Exception Cause Register

Address: 0xe9
Refer Exception Cause Register of own context number.

5.1.54 Own Interruption Wait Register

Address: 0xea
Refer Interruption Wait Register of own context number.

5.1.55 Own External Interruption Level Register

Address: 0xeb
Refer External Interruption Register of own context number.

5.1.56 Own Target IPC Register

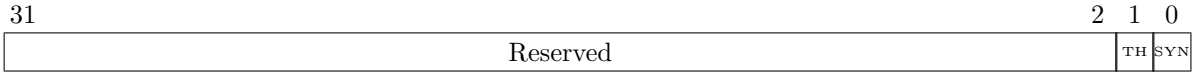
Address: 0xec
Refer Target IPC Register of own context number.

5.1.57 Own Fetch Bound Register

Address: 0xed
Refer Fetch Bound Register of own context number.

5.1.58 Special Mode Register

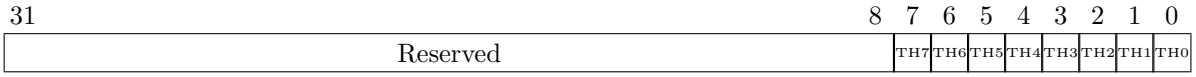
Address: 0xf0



Field Name	Function
Thread (TH)	default: 0 Set to 1 when using multithreading. 0: A thread resumed by insructions such as rstrth starts to run automatically. (Active Thread Run) 1: A resumed thread does not start automatically (Active Thread STOP). Use explicitly the runth instruction to start.
Sync (SYN)	default: 0 0: Sync MODE disable 1: Insert sync into all instruction forcefully

5.1.59 NMI Mode Register

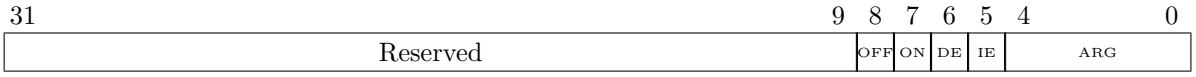
Address: 0xf1



Field Name	Function
Thread (TH)	default: TH0 = 1 Set the thread which accepts NMI input from outside. The thread jumps to the specified address when NMI inputing Set EBAE of Status Register to 1 and write the base address of the exception vector to Exception Base Address Register. When EBAE is set to 0, the address is set to 0x80000000.

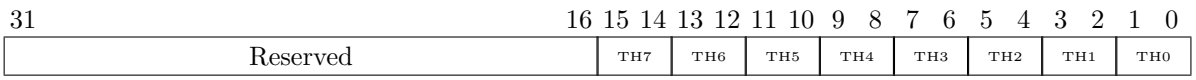
5.1.60 Special Operation Register

Address: 0xf2



5.1.61 I Cache ECC ON Register

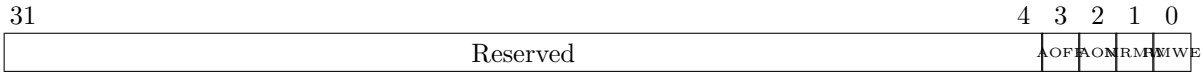
Address: 0xf3



Field Name	Function
TH	Set I Cache ECC ON or OFF. In default it is set to all off. Use shared setting format (Fig.**??**). If at least 1 ECC ON thread exists, write and invalidation is done with ECC ON. Read follows its thread's ECC ON, OFF.

5.1.62 I Cache ECC Mode Register

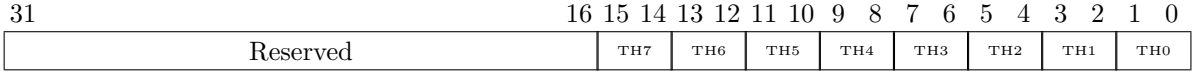
Address: 0xf4



Field Name	Function
AOFF	All ECC OFF 0: does nothing 1: Turn ECC OFF for all threads and Invalidation Prioritized over ECC ON or ALL ECC ON register
AON	All ECC ON 0: does nothing 1: Turn ECC ON for all threads and Invalidation Prioritized over ECC ON or ECC OFF register
MRMW	Manual Read Modify Write 0: does nothing 1: Starts Read modify Write. Once done goes back to 0.
RMWE	Read Modify Write Enable 0: Does not Read Modify Write when ECC is turned from OFF to ON. 1: Read Modify Write when ECC is turned from OFF to ON.

5.1.63 D Cache ECC ON Register

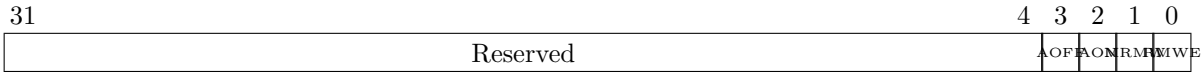
Address: 0xf7



Field Name	Function
TH	Set D Cache ECC ON or OFF. In default it is set to all off. Use shared setting format (Fig.**??**). If at least 1 ECC ON thread exists, write and invalidation is done with ECC ON. Read follows its thread's ECC ON, OFF.

5.1.64 D Cache ECC Mode Register

Address: 0xf8



Field Name	Function
AOFF	All ECC OFF 0: does nothing 1: Turn ECC OFF for all threads and Invalidation Prioritized over ECC ON or ALL ECC ON register
AON	All ECC ON 0: does nothing 1: Turn ECC ON for all threads and Invalidation Prioritized over ECC ON or ECC OFF register
MRMW	Manual Read Modify Write 0: does nothing 1: Starts Read modify Write. Once done goes back to 0.
RMWE	Read Modify Write Enable 0: Does not Read Modify Write when ECC is turned from OFF to ON. 1: Read Modify Write when ECC is turned from OFF to ON.

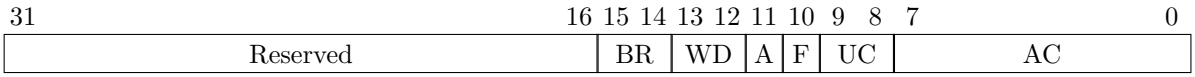
5.1.65 Address Decoder Control Register

Address: 0x100 ∼ 0x139, 0x140 ∼ 0x158

Refer Chapter 4(Address Decoder) address map.

5.1.66 Extbus0 Status

Address: 0x170 Default: 0x3000



Field Name	Function
BR	Set Burst Length 00: default 10: IO 11: MEMORY
WD	Set Bus Access Bandwidth 11: 32bit 01: 16bit 10: 8bit
A	Set Auto Ready 0: Disable 1: Enable
F	Set use of Flash IF This setting is only valid for external bus. 0: Disable 1: Enable
UC	Set Uncache (currently unavailable)
AC	Set Counts until Auto Ready is returned

5.1.67 Extbus1 Status

Address: 0x170
Same as Section5.1.66 Extbus0 Status.

5.1.68 Extbus2 Status

Address: 0x171
Same as Section5.1.66 Extbus0 Status.

5.1.69 Extbus3 Status

Address: 0x172
Same as Section5.1.66 Extbus0 Status.

5.1.70 Extbus4 Status

Address: 0x173
Same as Section5.1.66 Extbus0 Status.

5.1.71 Extbus5 Status

Address: 0x174
Same as Section5.1.66 Extbus0 Status.

5.1.72 Extbus6 Status

Address: 0x175
Same as Section5.1.66 Extbus0 Status.

5.1.73 Extbus7 Status

Address: 0x176
Same as Section5.1.66 Extbus0 Status.

5.1.74 ROM Status

Address: 0x178
Same as Section5.1.66 Extbus0 Status.

5.1.75 E2M Status

Address: 0x179
Currently Unused

5.1.76 Extbus Mem IO

Address: 0x17a
Currently Unused

5.1.77 Multiplexer Error Handler DMAC0

Address: 0x180
Same as Section5.1.30 Multiplexer Error Handler Instruction Cache.

5.1.78 Multiplexer Error Handler DMAC1

Address: 0x181
Same as Section5.1.30 Multiplexer Error Handler Instruction Cache.

5.1.79 Multiplexer Error Handler DMAC2

Address: 0x182
Same as Section5.1.30 Multiplexer Error Handler Instruction Cache.

5.1.80 Multiplexer Error Handler DMAC3

Address: 0x183
Same as Section5.1.30 Multiplexer Error Handler Instruction Cache.

5.1.81 Multiplexer Error Handler Extbus0

Address: 0x188

Same as Section5.1.30 Multiplexer Error Handler Instruction Cache.

5.1.82 Multiplexer Error Handler Extbus1

Address: 0x189

Same as Section5.1.30 Multiplexer Error Handler Instruction Cache.

5.1.83 Multiplexer Error Handler Extbus2

Address: 0x18a

Same as Section5.1.30 Multiplexer Error Handler Instruction Cache.

5.1.84 Multiplexer Error Handler Extbus3

Address: 0x18b

Same as Section5.1.30 Multiplexer Error Handler Instruction Cache.

6

Exception

6.1 Interruption Request Controller

Interruption request controller(IRC) is cascaded with two IRCs which have the same function. Outpus of Sub IRC(IRL; Interruption request level) are ORed and connected with IRQ16 in Main IRC. And RMT PU connects with Main IRC.

Initial Address: Main IRC: 0xffff9000, Sub IRC: 0xffff9400

6.1.1 Register Map

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

6.1.2 Trigger Mode Register

Offset: 0x00,0x04

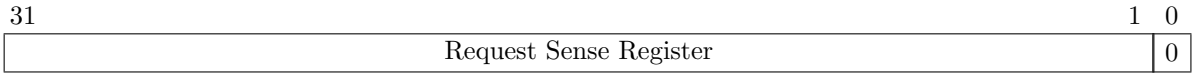
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
31ch		30ch		29ch		28ch		27ch		26ch		25ch		24ch		23ch		22ch		21ch		20ch		19ch		18ch		17ch		16ch	

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	

<table>
<tr><th>Field Name</th><th>Function</th></tr>
<tr><td>Trigger</td><td>Options of trigger mode of channel; channel 31-17 when offset is 0x00. channel 16-1 when offset is 0x04

<table><tr><th>bit</th><th>Trigger Mode</th></tr><tr><td>00</td><td>High Level</td></tr><tr><td>01</td><td>Low Level</td></tr><tr><td>10</td><td>Rise Edge</td></tr><tr><td>11</td><td>Fall Edge</td></tr></table></td></tr>
</table>

6.1.3 Request Sense Register



Field Name	Function
Request Sense	When a trigger, which is set in Trigger Mode Register, asserts on pin IRLIN or IRQIN, make the bit related with the interruption channel set to 1. Bit31 has relationship with IRQ31, and Bit1 has with IRQ1.(Read Only)

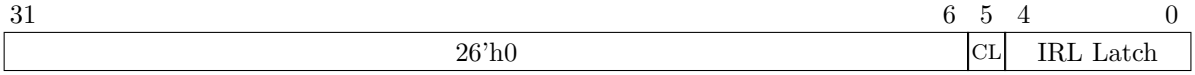
6.1.4 Request Clear Register

Field Name	Function
Request Clear	Make a bit in the intrruption register set to 0 when a bit in Reqest Clear Register(31-1) is set to 1. (Write Only)

6.1.5 Mask Register

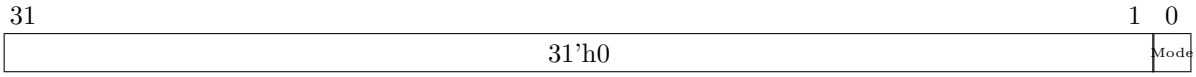
Field Name	Function
Mask	mask bits from 31 to 1 are used for the interruption channnel(31-1). it can use to mask interruptions to set to 1. However Request Sense Register will set to 1 even if a mask bit is 1.
MI	if 0, it will output value of the interruption level latch to IRLOUT. if 1 then it will mask and output “L”.

6.1.6 IRL Latch/Clear



Field Name	Function
IRL Latch	Outout value of the interruption level latch.
CL	Clear value of the interruption level latch and latch the next interruption level to write 1.

6.1.7 IRC Mode Register



Field Name	Function
Mode	if 1 then use IRLIN as IRQIN[31:27]. if 0 then output IRLIN to IRLOUT.

6.2 Function and Usage

6.2.1 IRC

When IRC mode register is set to 1, the interruption signals inputting to IRLIN[4:0](IRQIN[31:27]) and IRQIN[26:1] are stored following to the trigger mode in Trigger Mode Register. Interruption level latch(IRL Latch) stores the interruption which has the highest priority and is not masked by the mask register in these interruptions. Value of IRL Latch outputs to IRLOUT[4:0] if MI bit in the mask register is 0.

Do not use this following function; value of IRLIN[4:0] will output to IRLOUT[4:0] to set IRC Mode Register to 0.

Showing the Main IRC interruption map at Table and the Sub IRC interruption map at Table . Sub IRC interruption outouts(IRL) are ORed and cascaded with IRQ16 in Main IRC.

Table 6.1: Main IRC Interruption Map

IRQ31	Bus Error
IRQ30	Address Error
IRQ29	Watch Dog Timer Error
IRQ28	Responsive Link
IRQ27	DMAC3
IRQ26	DMAC2
IRQ25	DMAC1
IRQ24	DMAC0
IRQ23	PCI
IRQ22	DMAC256
IRQ21	IEEE1394
IRQ20	Ether MAC
IRQ19	SPI 1
IRQ18	SPI 0
IRQ17	PWM Input 2
IRQ16	Sub IRC
IRQ15	Pulse Counter 2
IRQ14	Pulse Counter 1
IRQ13	Pulse Counter 0
IRQ12	UART 1
IRQ11	UART 0
IRQ10	External IO 1
IRQ9	External IO 0
IRQ8	PWM Input 1
IRQ7	PWM Input 0
IRQ6	GPIO
IRQ5	RTC
IRQ4	DMAC4
IRQ3	PWM0
IRQ2	I^2C
IRQ1	Reserved

Table 6.2: Sub IRC Interrption Map

IRQ31	Reserved
IRQ30	Reserved
IRQ29	Reserved
IRQ28	Reserved
IRQ27	Reserved
IRQ26	External IO 3
IRQ25	External IO 2
IRQ24	Reserved
IRQ23	Reserved
IRQ22	Pulse Conter 3
IRQ21	Reserved
IRQ20	Reserved
IRQ19	PWM11
IRQ18	PWM10
IRQ17	PWM9
IRQ16	PWM8
IRQ15	PWM7
IRQ14	PWM6
IRQ13	PWM5
IRQ12	PWM4
IRQ11	PWM3
IRQ10	PWM2
IRQ9	PWM1
IRQ8	Reserved
IRQ7	Reserved
IRQ6	UART3
IRQ5	UART2
IRQ4	Reserved
IRQ3	Reserved
IRQ2	Reserved
IRQ1	Reserved

6.2.2 Original Functions of RMT

- IRL Unit

  It is quite inefficient that all running threads activate an intteruption handler because it makes response time increase. Thus RMT has IRL Unit to assign IRL to each thread.

  IRL Unit has Interruption Wait Register for each thread. Each bit of Interruption Wait Register is related with IRL. If the related bit of Interruption Wait Register recieved from IRL by IRC is 1, RMT accepts the interruption and notice it and its IRL to Exception Unit.

  Interruption Wait Register is backuped and restored when each context switch, therefore it does not reset as context switch.

- Create Thread since Interruption
  In order to shrten response time for thread operating for exteral envents, RMT runs a sleeping thread(STATE FIELD in Status Register is 0010 or 0011) when an external interruption happens on this thread.

  However, a bit of Interruption Wait Register relateing with IRL must 1 to use this function.

6.2.3 Exception Handle Process

Pending Register will be set to 1 when timer interruptions, hardware interruptions(external interruptions), or software interruptions happen. In addition, if external interruptions happen, be set to IRL noticed from HIRL of Exception Cause Register. Interruption Mask(bit11-8) and Status Register Interruption Mode(bit0) must set to 0 in order that interruptions happens normally.

When an exeption is happened by interruptions or RMTPU, it handles normally if Status Register Exception Level(bit1). Exception Unit handles exceptions referencing exception signals of CPU core and Pending Register of Exception Cause Register. The priorities of exception handlings; the highest is exceptions, timer interruptions, hardware interruptions, and the lowest is software interruptions.

When an exception handling runs, do actions below;

- Set Exception Level of Status Register to 1
- Store the mode at that time and shift to kernel mode
- Store PC which is the last commit before an exception happen to PC Register
- Write the identification code(Table6.3) to CODE field in Exception Code Register
- if Base Address Enable in Status Registser is 1 then shift control to the address adding value of Exceotion Base Address Register with offset following the exception
  if Base Address Enable is 0 then shift control to 0xbfc00200(if Exception Vector Location is 1) or 0x80000000(if EVL is 0)

If targetd thread sleeps when an external interruption happens, RMT will only run the thread in stead of normal exception handling.

In the exception handling routine for external interruption, it needs to clear interrupt requests stored in IRC and clear IRL Latch. As a result, RMT can store the next interruption in IRL Latch and update Pending Field related with Exception Cause Register. It requires to clear expressly Pending Field related with Exception Cause Register if timer interruptions or software interruptions.

At the end of the exception handling, do actions below by execute ERET instruction

- Set Exception Level of Status Register to 0
- Restore mode
- Shift control to the address stored in Exception PC Register

Table 6.3: Exception Code, Offset

Type	Code	Offset
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

7

Clock Genarator

7.1 Connection Graph

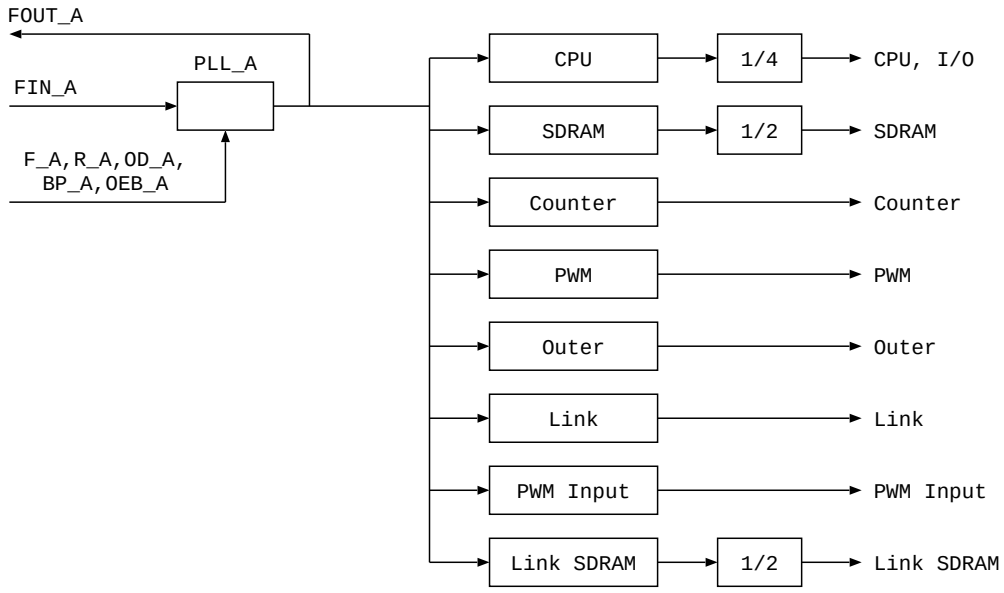


Figure 7.1: Generate Block

Pin Name	Abstract	Default Value
FIN_A	clock input	-
FOUT_A	PLL output	-
F_A	Control PLL Multiplier	16
R_A	Control PLL Multiplier	3
OD_A	Cotrol PLL Multiplier	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

As a default, input clock (75MHz) from FIN_A, and output clock (800MHz) from PLL_A. These are frequency-divided by a frequency divider and supplied to each module.

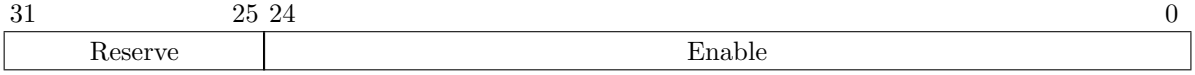
7.2 Control Register

Initial Address: 0xffffa000

7.2.1 Clock Enable

Offset: 0x0000

Set each clock to enable/disable. To be disable, set the corresponding to 1, to be enable set to 0. The default is all 0.

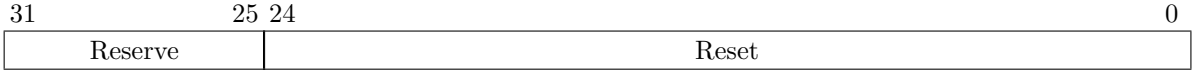


Field Name	Function
Enable	Set each clock to enable/disable.

7.2.2 Soft Reset

Offset: 0x0004

Reset each module. set the corresponding bit to 0 to reset. Reset continues until setting the bit to 1. (except cpu)



Field Name	Function
Reset	Reset each module.

7.2.3 Clock Enable / Soft Reset Bit Map

Bit map is the same Clock Enable and Soft Reset.

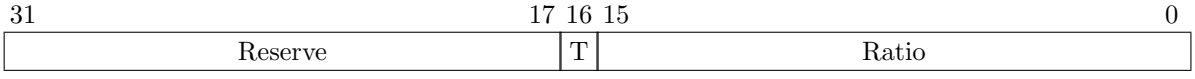
bit	module	bit	module
00	CPU	13	Vector Integer
01	DMAC0	14	Vector Floating-Point
02	DMAC1	15	SDRAM
03	DMAC2	16	Counter
04	IEEE1394	17	PWM
05	-	18	Outer
06	PCI	19	Link
07	Context Cache	20	PWM Input
08	Complex INT	21	Link Sdram
09	Floating-Point Unit	22	Ethernet MAC
10	SIMD	23	PIO
11	Floating-Point Reservation Station	24	SPI
12	Synchronize		

7.2.4 Divider Ratio

Offset: 0x0008～0x001c, 0x0028, 0x002c, 0x0030

Set the division ratio of each clock. The address and initial value of the corresponding frequency divider are as follows.

Divider	Default Value	Address offset
CPU	1/2	0x0008
SDRAM	1/4	0x000c
Counter	1/8	0x0010
PWM	1/512	0x0014
Outer	1/8	0x0018
Link	1/1	0x001c
PWM Input	1/512	0x0028
Link SDRAM	1/4	0x002c
SPI	1/8	0x0030

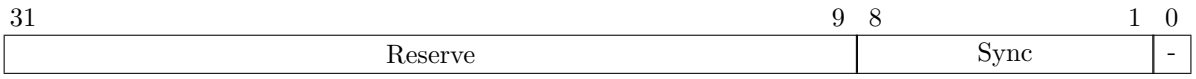


Field Name	Function
T	Through the clock without dividing (When 1/1)
Ratio	Specify the clock division ratio. The clock falls at half of the specified value, and the clock rises with the specified value. Operation when 1 is specified is not guaranteed. In the case of 1/1, set the T bit to 1.

7.2.5 Clock Synchronization

Offset: 0x0020

Align the rising edge of the clock specified 1 to the clock of the CPU. In the next clock value is reset automatically.



Field Name	Function
Sync	Align the rising edge of the clock to the clock of the CPU.

Bit map is below.

bit	module	bit	module
1	SDRAM	5	Link
2	Counter	6	PWM Input
3	PWM	7	Link SDRAM
4	Outer	8	SPI

7.2.6 All Reset

Offset: 0x0024

All reset to write to this address

8

Thread Control

This chapter describes thread controlling on *Responsive Multithreaded Processor*.

8.1 Thread Types

There are two types of threads on *RMT Processor*.

- Active Thread
- Cached Thread

Active Thread is threads whose hardware resources are reserved in processor, and can run immediately. Cached Thread is threads which is stored in the Context Cache. *RMT Processor* executes instructions of the thread who is active and executing. If reset signal is asserted, *RMT Processor* starts execution from an instruction at address 0 of thread 0 on priority 0.

8.2 Thread Control Instructions

8.2.1 Make, Delete

mkth instruction is used to create a new thread. It also can be used to create a new thread by copying an active thread.

- mkth: Make a new active thread. Set thread ID by rs, and set start address by rt. Other thread informations like stack pointer are not set by this instruction, it must be set by thread itself. Created active thread's state is stopped, and runth instruction is used to run stopped thread. rd is set to 1 if mkth is executed successfully, otherwise set to 0.
- delth: Delete an active thread which is selected by rs. If specified thread is deleted successfully, rd is set to 1, otherwise set to 0. If this instruction executed successfully, thread is removed from the processor.

- cpthtoa: Copy an active thread to another active thread. Original thread's ID is specified by rs, new thread's ID is specified by rt. New thread is on stopped state, and runth instructions is used to run. If this instruction is executed successfully, rd is set to 1, otherwise set to 0.
- cpthtom: Copy an active thread to a cached thread. Original thread's ID is specified by rs, new thread's ID is specified by rt. New copied thread is placed in Context Cache, and it should be activated using rstrth instruction. If this instruction is executed successfully, rd is set to 1, otherwise set to 0.

8.2.2 Thread State Control

Active thread is in a running state or a stopped state. Thread state is controlled by using the instructions below.

- runth: Activate a thread in stopped state. A thread ID to activate is specified by rs. If this instruction is executed successfully, rd is set to 1, otherwise set to 0.
- stopth: Stop a thread in active state. A thread ID to stop is specified by rs. To activate again, runth instruction is used. If this instruction is executed successfully, rd is set to 1, otherwise set to 0.
- stopslf:

  Stop a thread that executes this instruction. If this instruction is executed successfully, rd is set to 1, otherwise set to 0.
- chgpr:

  Change priority of a thread. A thread ID is specified by rs, new priority is specified by rt. If a priority is changed successfully, rd is set to 1, otherwise set to 0. The new priority is used from next clock cycle.

8.2.3 Transfer

RMT Processor has a context cache to reduce a context switch overhead. Instruction used to transfer data from/to context cache is shown below.

- bkupth:

  Save an active thread to context cache. Rs can select an active thread id to save. The selected active thread stops and save to context cache. When succeed to save, return 1 to rs, when fail return 0.
- bkupslf:

  Save itself to context cache. When succeed to save, return 1 to rd, when fail return 0.
- rstrth:

  Resume a cached thread as an active thread. Rs can select a cached thread id to resume, and the thread is loaded from context cache. Thread status after resuming is depened on the value of thread bit (the 1st bit) of Special Mode Register(0xfo). If this bit is 1 then the thread is resumed as a stopping thread, thus must use the runth inst in order to be an executing thread. If 0, the thread is resumed as excuting thread. When success to resume, return 1 to rd, when fail return 0.

- swapth:

  Swap an active thread and a cache thread. Rs selects an active thread id to save and rt selects cached thread id to resume. The selected active thread stops and saves to context cache. In the other hand, the selected cache thread is load from context cache. Thread status after resuming is depened on the value of thread bit (the 1st bit) of Special Mode Register(0xfo). If this bit is 1 then the resuming thread inherits status from saving thread. If 0, the resuming threads becomes an excuting thread. When succeed to swap, return 1 to rd, when fail return 0.

- swapslf:

  Swap itself and a cache thread. Rt selects a cached thread to resume. Stop itself and save to context cache. In the other hand, the selected cache thread is load from context cache and be an executing thread. When succeed to swap, return 1 to rd, when fail return 0.

8.3 State Transition

Fig. 8.1 shows state transition of

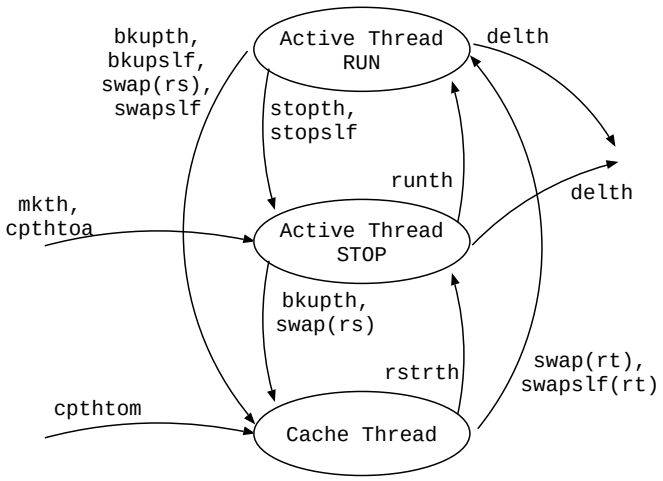


Figure 8.1: Thread Status Transition

Processers execute only threads under Active Thread RUN following priorities.

9

Synchronization

9.1 Shared Register

SRMTP has 31 64-bit shared registers, and they are used to synchronize by locking.

Shared registers are assigned to a thread id and 3bit which shows which threads have the right of this register.

- Full/Empty bit
- Exclusive / Producer-Consumer bit
- Barrier bit

Full/Empty bit indicates that this register is locked or not, and other 2 bit indicate types of lock.

9.2 Synchronization Instruction

To access shared registers, use instructions below.

- RGPEX, WGPEX, RFPEX, WFPEX
  Read opearation is executed when F/E bit of the target register is 0, and write the value of the target register to the destination register. When read operation is succeeded, set F/E bit of the target register to 1, and write own thread id to thread id field of the target register.

  Write operation is executed when F/E bit of the target register is 1 and the thread has the right of this shared registesr. When operation is succeeded, write the value of the source register to tha target register and free lock by setting F/E bit to 0. When target is unlocked, operation is nop.

- GPCO, GPPR, FPCO, FPPR
  Write operation is executed when F/E bit of the target shared register is 0, this operation writes the vaule of source register to the shared register. And write the thread id specified by the instruction to thread id field, this id has the right of the shared register.

  Read operation is executed when F/E bit of the target shared register is 1 and the thread has the right of this shared register. this operation reads the value of the shared register into the destination register.

- RGPSH, WGPSH, RFPSH, WFPSH
  Register access operation without locking. It can execute when hen F/E bit of the target resgister is 0.

- BAR
  F/E bit を 1 にします. Barrier operation. The shared register is used to count arrival thread. Source register indicates the target number of threads. At the first executing, set F/E bit and the value to 1. Otherwise, increment the value of the register and stall. When the value of shared register is equal to the vaule of source register, all stalls are released.

Table. 9.1 shows the execution condition of accessing shared register instruction.

Table 9.1: Execution Condition of Accessing Shared Register Instruction

Inst Type	Condition			After executing			Annotation
	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	match TH ID, otherwise nop
CON_READ	F	P	0	E	x	x	match TH ID
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	The first instrution which arrives to barrier
	F	x	1	F	x	1	Barrier wating
				E	x	0	Release barrier

For avoiding deadlocking, release all instructions in pipeline of the thread and stop to fetch when failing synchronazation. When the lock of the target register is released, stall is also released and re-start to fetch.

When the synchronous instruction fails, check the next thread ID and if the thread is stored in the context cache, replace this thread with the failed thread.

- When the target shared register is locked
  The thread which has the lock

- When Read Consumer falis since any value is not written
  The thread specified by Read Consumer operation.

When Waiting for other arriving thread with barrier instruction, check group threads which execute the same barrier instruction. PBAR is an instruction to set the thread group. PBAR specifies a shared register to use for barrier. Threads waiting for the barrier search context cache for a thread that PBAR operation specified the same index of shared register. If existing, swap the thread for itself.

This thread-change function becomes available by write the 31th shared register to 1. it is unavailable as a default.

10

IPC Control Mechanism

10.1 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.

10.2 Configurable Parameters

Configurable parameters are shown below.

- Proportional gain, integral gain, derivative gain of PID control (see also section5.1.41).
- Enable/Disable IPC control (see also section5.1.42).
- The number of instructions which will be executed during control period (see also section5.1.42).

Also, to use IPC control, you should configure the register below.

- Clock number of control period (see also section5.1.6).

10.3 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 section5.1.43).

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.

10.4 Program Example

This is a program example of the procedure described in section10.3, 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_IPC_FF_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) );
```

11

Vector Unit

11.1 Abstract

Fig. 11.1 is the block diagram of Vector Unit of RMT Processor. Vector Integer Unit mainly consists from three parts, and Vector Floating Point Unit is too.

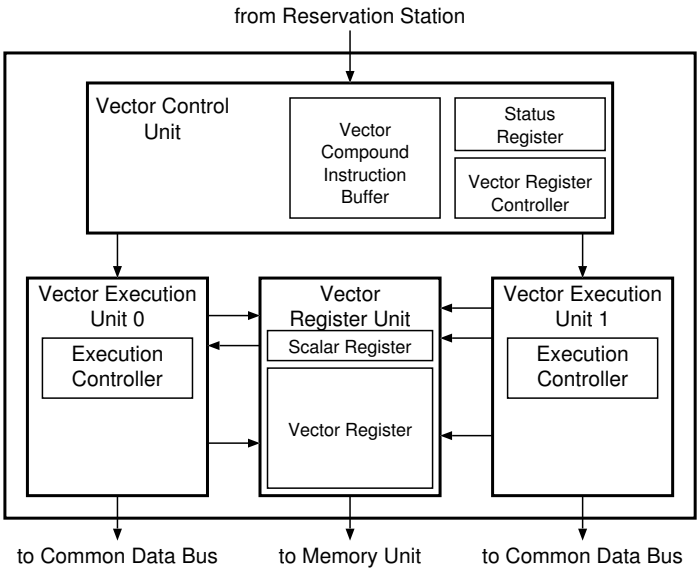


Figure 11.1: Diagram of Vector Unit

- Vector Control Unit

  It controls the execution unit, issues instructions, assigns and releases Vector Register which is explained later. It also manage control infomation for Vector operations such as Vector Length, Mask Bit and so on.

- Vector Register Unit

It has registers to execute Vector operations. It has a connection port to Vector Execution Unit and to Memory Unit. RMT Processor has 512 registers.

- Vector Execution Unit

  It pick vector elements up from Vector Register Unit to execute a vector operation, and store the result into Vector Register.

11.1.1 Vector Execution Unit

Fig. 11.2 is the block diagram of Vector Execution Unit.

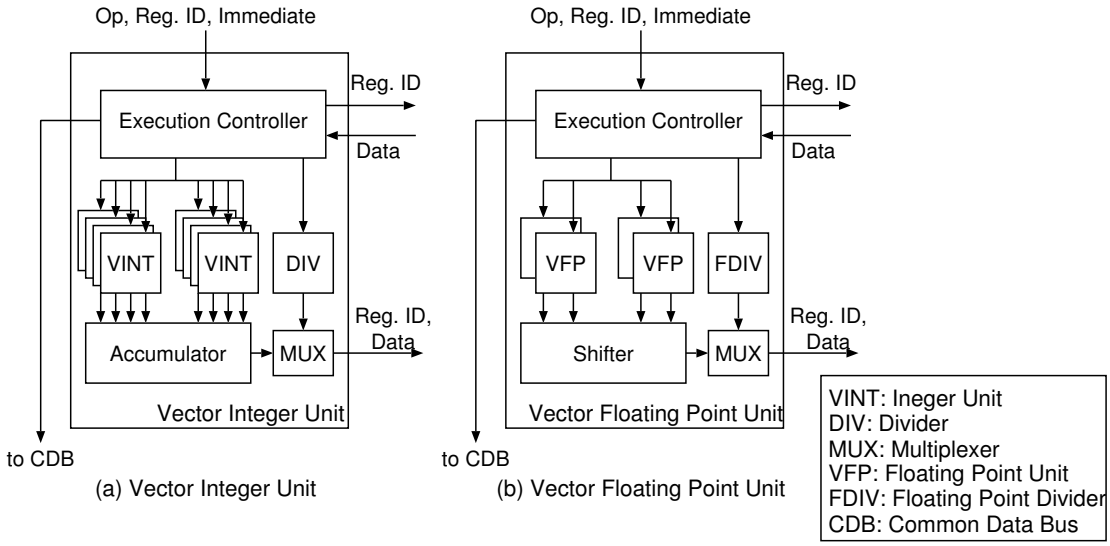


Figure 11.2: Diagram of Vector Execution Unit

Execution Controller picks vector data up from Vector Register Unit, and transports to Vector Integer (or Floating Point) Unit.

In RMT Processor, Vector Integer Unit has eight ALU and Vecotor Floating Point Unit have four in order to improve the performance of Vector operations. Each ALU is pipelied and controls one operations per clock.

Because division is used rarely, only one of two Vector Execution Units has the division circuit.

11.1.2 Instruction Format

The format of Vector operations is based on extended R-type operations. Opecode field is used for SIMD operations; 0x011110 (Word), 0x110110 (Paired HalfWord) and 0x111110 (Quad Byte) are for Vector Integer instructions, and 0x011111 (Double / Single) and 0x111111 (Paired Single) are for Vector Floating Point instructions. The format of instructions of Vector operations is shown at Fig. 11.3 .

Rs and rt field specify Source Register of Vector operations, and rd field specifies Destination Register. Fucnction field specifies the type of operations. The use of Subfunc field depends on the type of operations: scalar bit for Vector-Scalar operations, cond bit for compare operations, and sync bit for ordering instructions.

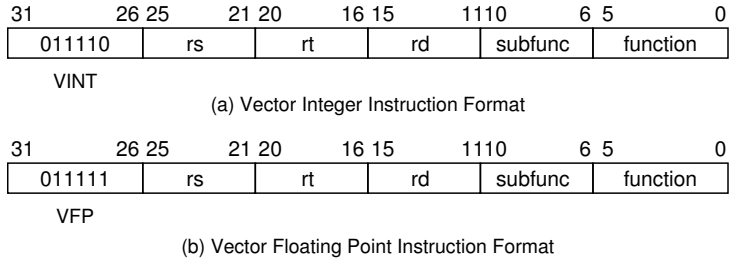


Figure 11.3: Format of Vector Instructions

11.2 Reserve/Release 命令

A big Vector Register is required to execute Vector operations. If each thread has this vector register, it causes increase of gate size and waste of registers. Thus there is one Vector Register in RMT Processor and several threads share it. Each thread reserves Vector Register as much as needed.

Sharing Vector Register is achived by dividing Vector Register into 4 area as Fig. 11.4 . Reservation size is fixed: 128 entries, 256, and 512. in the same way, Scalar Register is divided into 4 area, and the size of the scalar register that can be used depends on the size of the reserved vector register.

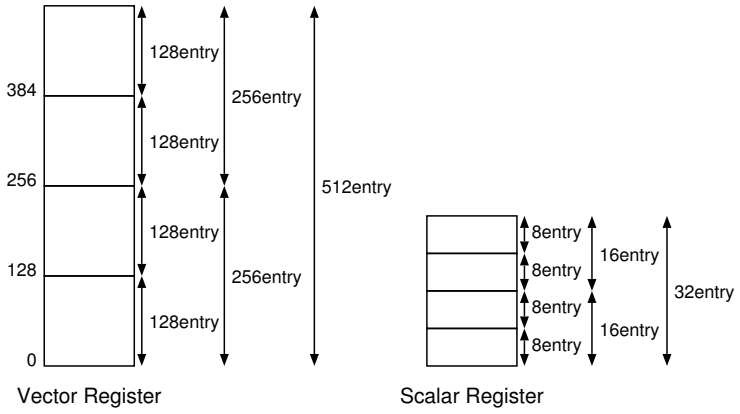


Figure 11.4: Size of Vector Register

Reserved Vector Register is diveided by vector length. Vector length is 8length, 16length, 32length, or 64length. Fig. 11.5 shows selectable constitutions of Vector Register in RMT Processor.

(a) is the contitution when reserving 128 Vector Registers. In this case, 16 8length-registers and 8 16length-registers are available. (b) is the contitution when reserving 256 Vector Registers. In this case, 32 8length-registers, 16 16length-registers, and 8 32length-registers are available. (b) is the contitution when reserving 512 Vector Registers. In this case, 64 8length-registers, 32 16length-registers, and 16 32length-registers are available. Choose one.

When executing an operation in Vector Unit, firstly reserve Vector Register as much as needed before executing vector operations. Vector Reserve instruction makes reservation of Vector Register. Vector Release instruction releases the reservated area, and after that other thread can use that area. The example using Reserve and Release is shown on Fig. 11.6 ,

The operand rs of the Vector-Reserve instruction chooses constitution mode of Vector Register in Fig. 11.5 . Choose one mode from Table 11.1 . The upper 2 bits mean register size reserving, and lower 2 bits

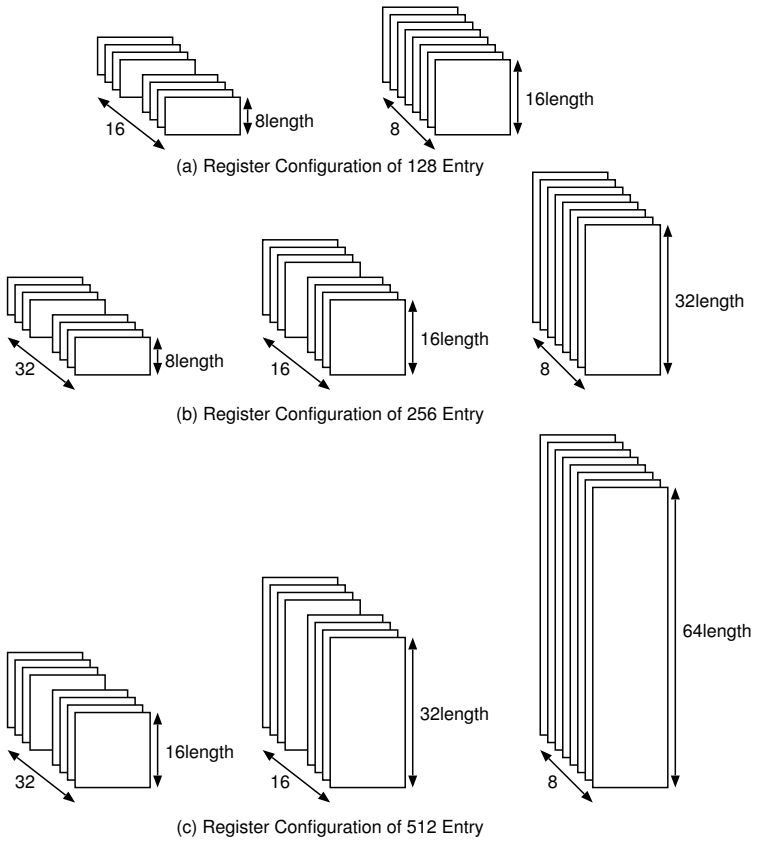


Figure 11.5: Constitution of Vector Register

are vector length. Vector Reserve instruction returns 1 to rd when succeeding reservation. when failing, returns 0.

When Vector Register is reserved by Vector Reserve instruction, the information of reserved Vector Register is writen into Register Status Table in Vector Register Controller in Vector Control Unit. Fig. 11.7 is the format of Vector Status Table.

Busy Bit indicates that a thread is reserving Vector Register or not. Start Address indicates which part of Vector Register divided into four as shown at Fig. 11.4 is reserved. Mode stores the institution of Vector Register specified by rs of Vector Reserve instruction.

When executing Vector Release instruction, it releases reserving Vector Registser and return 1 to rd. When executing this instruction while not reserving any Vector Register, return 0 to rd.

```
addu      $11, $0, 0x000A       # 256 Entry ( 32Depth x 8 ) Mode
virsv     $10, $11              # Reserve Instruction

== Vector Execution ==

virls     $10                   # Release Instruction
```

Figure 11.6: Vector Operation Example

Table 11.1: Vector Register Mode

(128 Entry)	
8 Depth × 16	0x4
16 Depth × 8	0x5
(256 Entry)	
8 Depth × 32	0x8
16 Depth × 16	0x9
32 Depth × 8	0xA
(512 Entry)	
16 Depth × 32	0xD
32 Depth × 16	0xE
64 Depth × 8	0xF

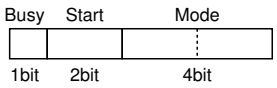


Figure 11.7: Vector Status Table

11.3 Status Register

Status Register holds infomation shown below to execute Vector operations for each thread.
In order to access Status Register, use vimfc or vfmfc (for read), vimtc or vfmtc (for write).

11.4 Compound Instruction

The utilization of Vector Unit is improved by operating compound instructions defined by the users. Compound instructions is defined in Compound Instruction Buffer, in Conpound Instruction Controller, in Vector Control Unit. Likely Vector Register and Scalar Register, Compound Instruction Buffer is divided into four and used the same area to reserved Vector Register. Compound Instruction Buffer has totally 32 entries, thus when reserving 128 Vector Registers, 8 entries are avaliable. And when reserving 256, 16 entries are available, and when 512, 32 entries are.

Fig. 11.8 shows the format of Comound Instruction Buffer.

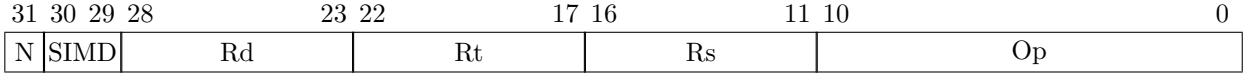


Figure 11.8: Format of Compound Instruction Buffer

Define one instruction in one entry and Compund them in order to define compound instructions. Next (N) bit indicates the compund instruction continues to the next entry. Next bit of the last instrcution of

Address	Name	Description
0x00	Mask (Low)	Corresponds to the element (lower half) of the vector register and masks the operation by setting 1. LSB corresponds the first element and MSB corresponds the 32th element.
0x01 (Int)	Mask (High)	Corresponds to the element (upper half) of the vector register and masks the operation by setting 1. MSB corresponds the 64th element and LSB corresponds the 33th element.
0x01 (FP)	Routing Node	Specifies floating point routing mode. (0: Round to Nearest, 1: Round to 0, 2: ROund to $+\infty$, 3: Round to $-\infty$)
0x02	Length	Specifies Vector Length for operations (acctually Vector Lenght - 1).
0x03	Stride	Specifies Address stride of Load / Store. Actually specifies word numbers which is an interval between elements. When setting to 0, Loads Vector elements from a continuous address. When setting to 1, load elements every other word.

the compund instruction must be 0. To define several compund instrucitons, cut a compound instruction off by setting Next bit to 0.

SIMD field Specifies bit length for SIMD operations. When integer operations, execute 32bit operations (not SIMD operation) by setting this field to 0x0. 0x1 means 16bit $\times$ 2 operations, and 0x2 is 8bit $\times$ 4 operations. When floating point operations, 0x0 means normal operations (not SIMD), and 0x1 is 32bit $\times$ 2 operations.

Rs and Rt is source registers, and Rd is a distination regsiter. The format is shown below.

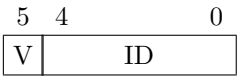


Figure 11.9: Register Setting

In order to use Vector Register, set V bit to 1. To use Scalar Register, set it to 0. ID field specifies register ID to use. The available bit length of ID bit is decided (show Fig. 11.5). For examples, In the case of 8length $\times$ 16, the four lower bits of ID are available. In addition, in the case of 16length $\times$ 32, 5 bits are.

Rs, rt, and rd are actually used as offset of rs, rt, and rd which are specified by VIECI or VFECI instruction. For example, when rs of VIECI instruction is 1 and rs of ID of Compound Instruction Buffer is 3, the Register ID is 1 + 3, thus 4.

Specify the instruction to operate in the operation field. the format of integer operation is shown below.

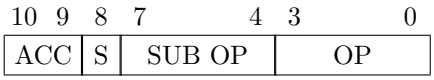


Figure 11.10: Format of Operation (Interger Operation)

specify one for OP

- NOP (0x0)

  Nothing

- AND (0x1)

  Logical conjuction.

- OR (0x2)

  Logical disjuction. When setting the 6th bit (SUB OP) to 1, NOP.

- XOR (0x3)

  Exclusive logical disjuction.

- ADD (0x4)

  Addition. When setting the 6th bit (SUB OP) to 1, substruction.

- MULT (0x5)

  Multiplication. When setting the 4th bit (SUB OP) to 1, unsigned operation. When setting the 6th bit (SUB OP) to 1, return upper 32 bits of 64bit operation result.

- SHIFT (0x6)

  Shift logical. When settign the 4th bit (SUB OP) to 0, shift left logical, and when 1, shift right logical. When setting the 6th bit (SUB OP) to 1, arithmetic shift logical. When the 5th bit (SUB OP) is 1, not shift but rotation.

- COMPARE (0x7)

  Compare operation. When setting the 4th bit (SUB OP) to 1, assume operands as unsigned values. the 5-7th bits (SUB OP) specify the condition: 0x0: constant false, 0x1: =, 0x2: >=, 0x3: >, 0x4: constant true, 0x5: $\neq$, 0x6: <, 0x7: <=.

- THROUGH (0x8) return rs value.

- MADD (0x9)

  Multiplication and addition. When setting the 6th bit (SUB OP) to 1, multiplication and substruction.

- DIV (0xf)

  Division. When setting the 4th bit (SUB OP) to 1, unsigned division. When setting the 6th bit (SUB OP) to 1, modulo.

When setting S bit to 1, Scalar operations in SIMD operations. For example, in the case of 8bit $\times$ 4, the lower 8bits of rt are used at all field.

Add operation results of each vector element by setting ACC field to 0x2. In this case, must set Rd to a scalar register.

The format of floating point operations is shown below.

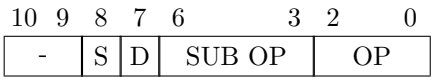


Figure 11.11: Operation Format (Floating Point)

specify one for OP

- NOP (0x0)

  Nothing.

- THROUGH (0x1)

  Return rs value. When setting the 3rd bit (SUB OP) to 1, reverse sign. When setting the 4th bit (SUB OP) to 1, return absolute value.

- ADD (0x2)

  Add. When setting the 4th bit (SUB OP) to 1, substrcut.

- MULT (0x3)

  Multiply.

- CONVERT (0x4)

  Convert the format. the 4th-3rd bits specify the format: 0x0: single float, 0x1: double float, 0x2: integer. When setting the 6th bit (SUB OP) to 1, assume the source operand as integer.

- COMPARE (0x5)

  Compare operation. the 5th-3rd bits (SUB OP) specify the format: 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)

  Multiplication and addition. When setting the 6th bit (SUB OP) to 1, multiplication and substruction.

- DIV (0x7) Division.

When D bit is 1, assume the operand as a double float. when 0, as a single float.

When setting S bit to 1, Scalar operations in SIMD operations. For example, in the case of 32bit $\times$ 2, the lower 32bits of rt are used at all field.

Define a compound instruction in Compound Instruction Buffer with a VIDCI or VFDCI instruction. And begin to execute the compound instruction with a VIECI or VFECI instruction. The format of each instrcution is shown at Fig. 11.12 .

To define a compound instruction, specify a register having value following Fig. 11.8 , and ID of Compound Instruction Buffer to store into rd. The execute a compound instruction, specify Source Register in the rt and rs field, Destination Register in the rd field, and the location of Compound Instruction Buffer in the no field.

When issuing a execute instruction, Compound Instruction Controller reads the instruction in the entry specified by no field from Compound Instruction Buffer, and transmits the read instruction to Vector

VIDCI (Vector Integer Define Compound Instruction)

31–26	25–21	20–16	15–11	10–6	5–0
011110	rs	00000	rd	00000	101110
VINT					VIDCI

VIECI (Vector Integer Execute Compound Instruction)

31–26	25–21	20–16	15–11	10–6	5–0
011110	rs	rt	rd	no	101111
VINT					VIECI

VFDCI (Vector Floating Point Define Compound Instruction)

31–26	25–21	20–16	15–11	10–6	5–0
011111	rs	00000	rd	00000	101110
VFP					VFDCI

VFECI (Vector Floating Point Execute Compound Instruction)

31–26	25–21	20–16	15–11	10–6	5–0
011111	rs	rt	rd	no	101111
VFP					VFECI

Figure 11.12: Format of Compound Instruction

Execution Unit after converting Register ID. If Next Bit of the read instruction is 1, read an instruction in the next entry from Compound Instruction Buffer and continue. Also if Next Bit is 0, finish the compound operation and accept the next instruction.

	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 11.13: Example of definition of Compound Instruction

Fig. 11.13 is examples of compound instructions. in the examples, entries 0 and 1 make a compound instruction; compare with the scalar register value after adding vector. And entries from 2 to 9 make an another compound instruction; convert vector. When setting no field of execute instruction to 0, execute adding and comparing, and when 2, execute vector conversion.

12

DMAC

- 32/16/8 bit I/F
- Num of Input Channel: 4
- Priority: Fixed and Round-Robin
- Memory to memory Transport
- Bus sizing (for 8, 16bit I/O)
- Bus swapping (for 8, 16bit I/O)

12.1 Register Map

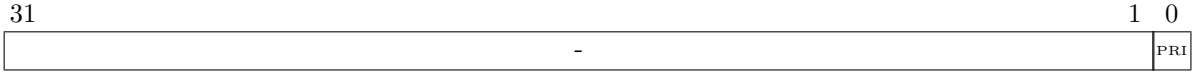
DMAC	Initial Address
DMAC0	FFFF0000
DMAC1	FFFF1000
DMAC2	FFFF2000

offset	31	24 23	16 15	8 7	0
0x800	-				PRI
0x804	-				IC
0x40*(x)+0x04	PSA<31:0>				
0x40*(x)+0x08	MDA<31:0>				
0x40*(x)+0x18	LN<31:0>				
0x40*(x)+0x0c	ID<31:0>				
0x40*(x)+0x10	-		DAS	SAU	BM	RL	PCM	TM	MR	32P	16P	8P	S16	S8	IER	IED	ST
0x40*(x)+0x14	L0	L1	L2	L3	-											ER	ED

12.1.1 DMA Control Register

Write/Read

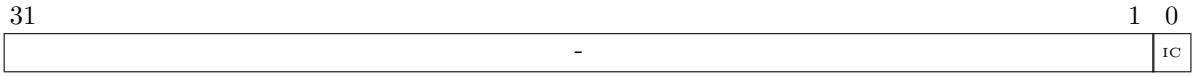
Offset: 0x800



Field Name	Function
PRI	PRIority :Default 0 This bit shows the priority of the DMA channel. 0: Round-Robin 1: ch0>ch1>ch2>ch3

12.1.2 DMA Interrupt Clear Register

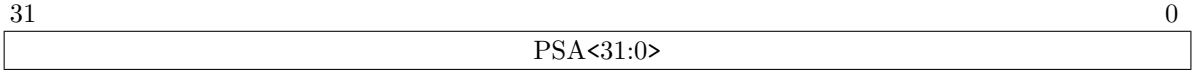
Offset: 0x804



Field Name	Function
IC	Interrupt Clear This bit clears DMA interruptions, 0: Clear interruptions

12.1.3 Port/Source Address Register

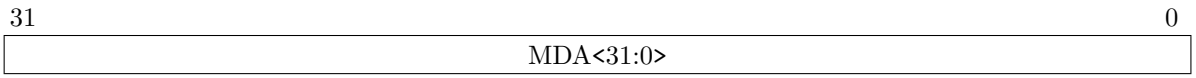
Offset: 0x40*(x) +0x04



Field Name	Function
PSA<31:0>	Port/Source Address :Default X This field shows the port/source address . When the MTM bit of MODE register is 0 (from memory to I/O), this bit means port address. When the MTM bit of MODE register is 1 (from memory to memory), this bit means source address.

12.1.4 Memory / Destination Address Register

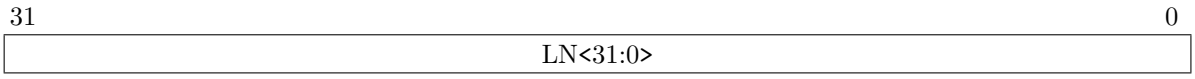
Offset: 0x40*(x) +0x08



Field Name	Function
MDA<31:0>	Memory/Destination Address :Default X This field shows the memory/destination address for channel x. When the MTM bit of MODE register is 0 (from memory to I/O), this bit means memory address. When the MTM bit of MODE register is 1 (from memory to memory), this bit means destination address.

12.1.5 Transport Length Register

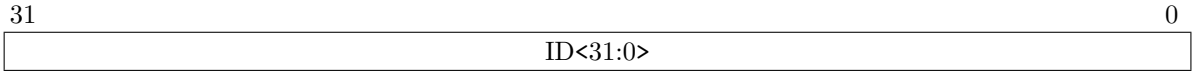
Offset: 0x40*(x) +0x18



Field Name	Function
LN<31:0>	transfer LeNgth :Default X This field shows the transport length (byte) for channel x.

12.1.6 Data Buffer Register

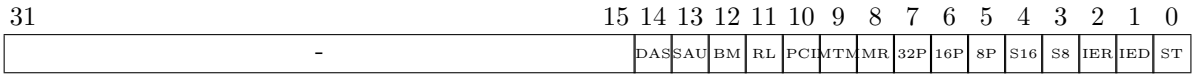
Offset: 0x40*(x) +0x0c



Field Name	Function
ID<31:0>	Internal Data :Default X This field stores the data before transporting to the DMA of channel x. Status Register has where are valid data in Data Buffer Register.

12.1.7 Transfer Mode Control Register

Offset: 0x40*(x) +0x10



Write/Read

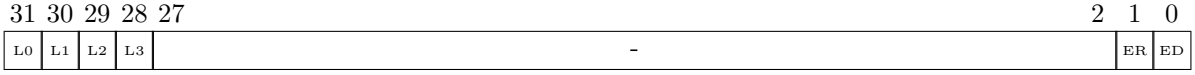
Field Name	Function
DAS	Destination Address Update :Default X 0:The Address which is set in Memory Address Register is used for the next transpotation. 1:The Address points 1 word ahead at the last trasportation.
SAU	Source Address Update :Default X 0:The Address which is set in Port Address Register is used for the next transpotation. 1:The Address points 1 word ahead at the last trasportation.
BM	Burst Mode :Default X 0:Normal Transportation 1:Burst Transportation
RL	Responsive Link :Default X 1:DMA-transport to DPM for Responsive Link.
PCI	PCI :Default X 1:DMA-transport to PCI
MTM	Memory To Memory transfer :Default X 0:DMA-transfer between the memory and the I/O port set by Port Address Register. MR bit specifies the transfer direction. 1:DMA-transfer from the source address to the destination address, the data length is set by Length Register. Address counter only counts up, not down. And in the case of Memory to Memory, DMA-transfer only supports 4 byte boundary about address and length.
MR	MR Memory Read :Default X 0:transfer from I/O to Memory 1:transger from Memory to I/O
32P	32bit I/O Port :Default X 0:don ' t care
16P	16P 16bit I/O Port :Default X 0:don ' t care 1:transfer from/to 16bit I/O when MTM bit is 0. Bit 0 of the port address is ignored. and bit 1 shows where data bus is connected with. (D31-16 or D15-0)
8P	8P 8bit I/O Port :Default X 0:don ' t care 1:transfer from/to 16bit I/O when MTM bit is 0. Bit 0 and 1 show where data bus is connected with. (D31-24 or D23-16 or D15-8 or D7-0)
S16	Swap at 16bit :Default X 0:don ' t care 1:data swap with 16 bit unit 31 | A | B | C | D | 0 → 31 | C | D | A | B | 0
S8	Swap at 8bit :Default X 0:don ' t care 1:data swap with 16 bit unit 31 | A | B | C | D | 0 → 31 | B | A | D | C | 0 If both S16 and S8 are set to 1, the swap result is shown below. 31 | A | B | C | D | 0 → 31 | D | C | B | A | 0
IER	Interrupt enable of ER-bit :Default 0 0:Don't allow interruptions. 1:Allow interruptions.
IED	Interrupt enable of ED-bit :Default 0 0:Don't allow interruptions. 1:Allow interruptions.
ST	Start :Default 0 0:Stop DMA-tranfer. DMAC is initialized after write 0 to ST. 1:Run DMA-transfer.

All swap modes are shown in the next page.

transfer mode		no swap (S16=0,S8=0)	swap (S16=0,S8=1)	swap (S16=1,S8=0)	little endian (S16=1,S8=1)
Memory (32bit)	Memory (32bit)	O	X	X	O
Memory (32bit)	I/O 32bit(D31-0)	O	O	O	O
	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)	○	×	×	○
Memory(D31-16)	I/O 8bit(D31-24)	○	×	×	○

12.1.8 Status Register

Offset: 0x40*(x)+0x14 Write/Read



Field Name	Function
L0	Location 0 :Default 0 0: The internal registers D31-24 are invalid data. 1: The internal registers D31-24 are valid data.
L1	Location 1 :Default 0 0: The internal registers D23-16 are invalid data. 1: The internal registers D23-16 are valid data.
L2	Location 2 :Default 0 0: The internal registers D15-8 are invalid data. 1: The internal registers D15-8 are valid data.
L3	Location 3 :Default 0 0: The internal registers D7-0 are invalid data. 1: The internal registers D7-0 are valid data.
ER	Error :Default 0 0:don't care 1:DMA-transfer is errored and stopped. Write 0 to clear.
ED	END :Default 0 0:don ' t care 1:DMA-transfer ends. Write 0 to clear.

13

DMA with Bus Sizing

13.1 Abstract of This DMA

transport bus-sizeing from 256 bit to 32 bit.

13.2 Control Register

Table 13.1: List of Control Register

Address	Register Name	Use
0xFFFFD000	PSA	Specify the transfer source address(32 bit)
0xFFFFD004	MDA	Specify the transfer desitination address(32 bit)
0xFFFFD008	LENGTH	Specify the transfer data size(byte)
0xFFFFD00C	MODE	Specify the transfer mode and start transfer

13.3 Details of Control Regsiter

13.3.1 PSA Register

Address: 0xFFFFD000

Specify the address of DMA transfer destination with 32 bit. Read/Write.

13.3.2 MDA Register

Address: 0xFFFFD004

Specify the address of DMA transfer source with 32 bit. Read/Write.

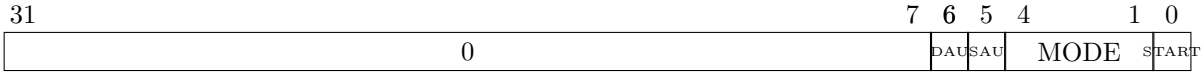
13.3.3 LENGTH Register

Address: 0xFFFFD008

Specidy the trasfer data size (byte). Read/Write

13.3.4 MODE Register

Address: 0xFFFFD00C



<table>
<tr><td>Field Name</td><td>Function</td></tr>
<tr><td>DAU</td><td>The Address as the transfer desitination at the start of transfer. Only Write

<table><tr><td>Value</td><td>Behavior</td></tr><tr><td>0</td><td>Address set to PSA</td></tr><tr><td>1</td><td>Address of the Last transfer destination</td></tr></table></td></tr>
<tr><td>SAU</td><td>The Address as the transfer source at the start of transfer. Only Write

<table><tr><td>Value</td><td>Behavior</td></tr><tr><td>0</td><td>Address set to MDA</td></tr><tr><td>1</td><td>Address of the Last transfer destination</td></tr></table></td></tr>
<tr><td>MODE</td><td>Specify the source unit and the destination unit. Read/Write

<table><tr><td>設定値</td><td>転送先</td><td>転送元</td></tr><tr><td>0000</td><td>I/O</td><td>I/O</td></tr><tr><td>0100</td><td>Memory</td><td>I/O</td></tr><tr><td>0001</td><td>I/O</td><td>Memory</td></tr><tr><td>0101</td><td>Memory</td><td>Memory</td></tr><tr><td>1100</td><td>SDRAM</td><td>I/O</td></tr><tr><td>1101</td><td>SDRAM</td><td>Memory</td></tr><tr><td>0011</td><td>I/O</td><td>SDRAM</td></tr><tr><td>0111</td><td>Memory</td><td>SDRAM</td></tr><tr><td>1111</td><td>SDRAM</td><td>SDRAM</td></tr></table>• SDRAM : DDR SDRAM I/F (256 bit bus)
• Memory : The Memory connected with 32 bit Bus
• I/O : The I/O connected with 32 bit Bus</td></tr>
<tr><td>START</td><td>DMA の転送の開始を指定する．Read/Write 可能．Start DMA transfer. Read/Write

<table><tr><td>Value</td><td>Behavior</td></tr><tr><td>0</td><td>DMA transfer is finished</td></tr><tr><td>1</td><td>DMA transfer is work in process</td></tr></table></td></tr>
</table>

13.4 Caution

- The address specified for data transfer must be aligned to 8 words (32 bit).

14

パルスカウンタ

14.1 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

14.2 レジスタインタフェース

14.2.1 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

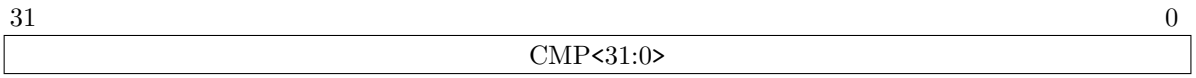
31	30	12	11	10	9	8	7	6	5	4	3	2	1	0
INT	-		IPCE	IZE	ZF	RFZ	ST	TI	SEL	MD		IE	CLR	CE

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

14.2.2 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．

14.2.3 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.

14.2.4 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.

15

PWM Generator

15.1 Abstract of PWM Generator

- Counter and Timer
- Up-Down Counter
- PWM Output: duty aspect follows options of internal registers
- Bit length: 32bit
- Generate PWM by using sawtooth wave or triangle wave
- Set Deadtime
- Set Active High/Low
- Group PWM channel and synchronize
- Interrupt at each PWM clock cycle
- Use as global output
- number of channels: 12

Figure15.1 shows PWM waves under mode sawtooth and triangle.

Reversed outputs with deadtime have a cyclic cascade connection in order to output from a neighbor PWM generator. It means that the PWM generator N can use reversed outputs with deadtime from generator N-1 by asserting the REV bit.

In addition, it is able to make a PWM group to synchronize their starts of counters.

15.2 PWM Controll Register

Address	CTRL Register
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]

Read/Write

Field Name	Function
INT	Invert: Default 0 0: Don't allow interruptions 1: Allow interruptions Interruptions happen at beginning of period.
SYN	Invert: Default 0 0: Don't synchronize beginnings of count (use only CEN). 1: Synchronize beginnings of count (use the start signal generated by the younger neighbor generator). Start signal of a PWM counter becomes active when CEN becomes 1, or SYN is 1 and it recieves the start signal from the younger neighbor generator. It means that several PWM generators can use one CEN signal. This is a group synchronizing beginnings of counters.
INV	Invert: Default 0 0: Outout PWM waves 1: Inverse and output PWM waves at the last step of the PWM output. It has the highest priority in options.
M	Mode: Default 0 0: Generate PWM waves with sawtooth waves 1: Genarate PWM waves with triangle waves
REV	Reverse mode enable: Default 0 0: Output PWM waves generated by this generator. 1: Output reversed PWM waves with deadtime generated by the younger neighbor generator (the counter in this generator is not used). It has higher priority than DEN.
DEN	Data Enable: Default 0 0: Output PWM waves. 1: Outout the constant value set in D bit. It has lower priority than REV.
D	Data: Default 0 If DEN is 1 then output the constant value set in this D bit.
P	Positive: Default 0 Determine active high/low (ref. Figure15.1, 15.2). 0: Active Low 1: Active High
CLR	Counter clear: Default 0 0: Nothing. 1: Clear the counter.
CEN	Count Enable: Default 0 0: Stop the counter. 1: Start the counter.

15.3 PWM Period Control Register

Address	PWM Forward Control Register
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]

Read/Write

Field Name	Function
FWCNT	Forward Counter: Default 0 It is a counter register to determine period of PWM. If Mode is 0 (sawtooth mode), it determines period of PWM. PWM signal value synchronizes with this counter value (ref. Figure15.1). If Mode is 1 (triangle mode), it determine a half of period of PWM. PWM signal value increases while PWM counter increases from 0 to FWCNT at a clock, and the next clock PWM signal value decreases. (ref. Figure15.2).

15.4 PWM Inverse Control Register

Address	PWM Inverse Control Register
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]

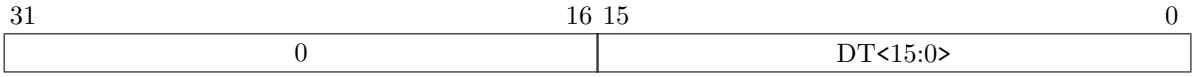
Read/Write

Field Name	Function
REVCNT	Reverse Counter: Default 0 It is a register to determine time to inverse PWM output. PWM output will be inversed when value of PWM counter is equal to value of this register (ref. Figure15.1, 15.2).

15.5 Deadtime Register

Address	Deadtime Register
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]

Read/Write



Field Name	Function
DT<15:0>	Reverse Counter :Default 0 It is a register to set deadtime. PWM output will be inversed when value of PWM counter is equal to value of this register (ref. Figure15.1, 15.2).

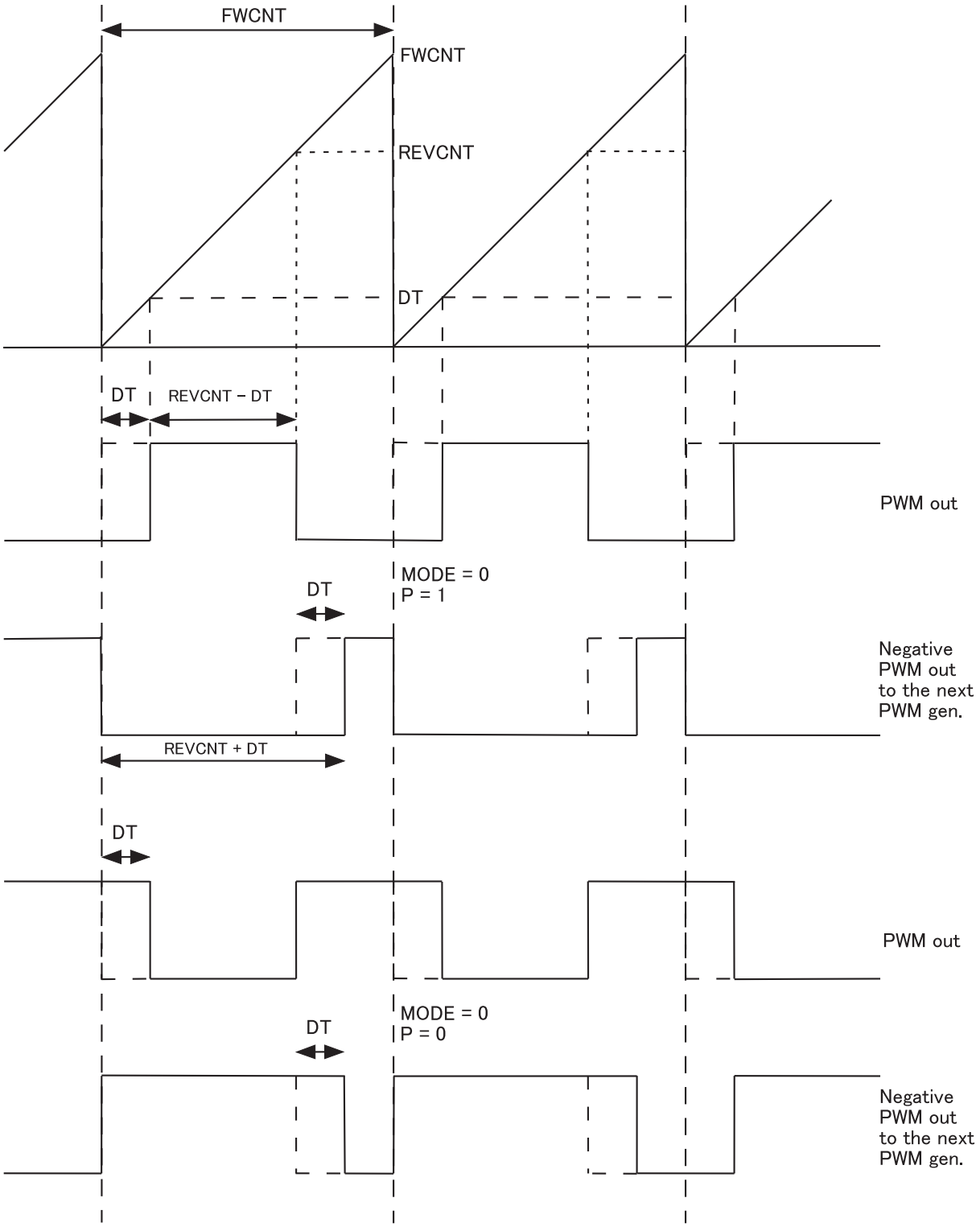


Figure 15.1: Sawtooth Wave Mode

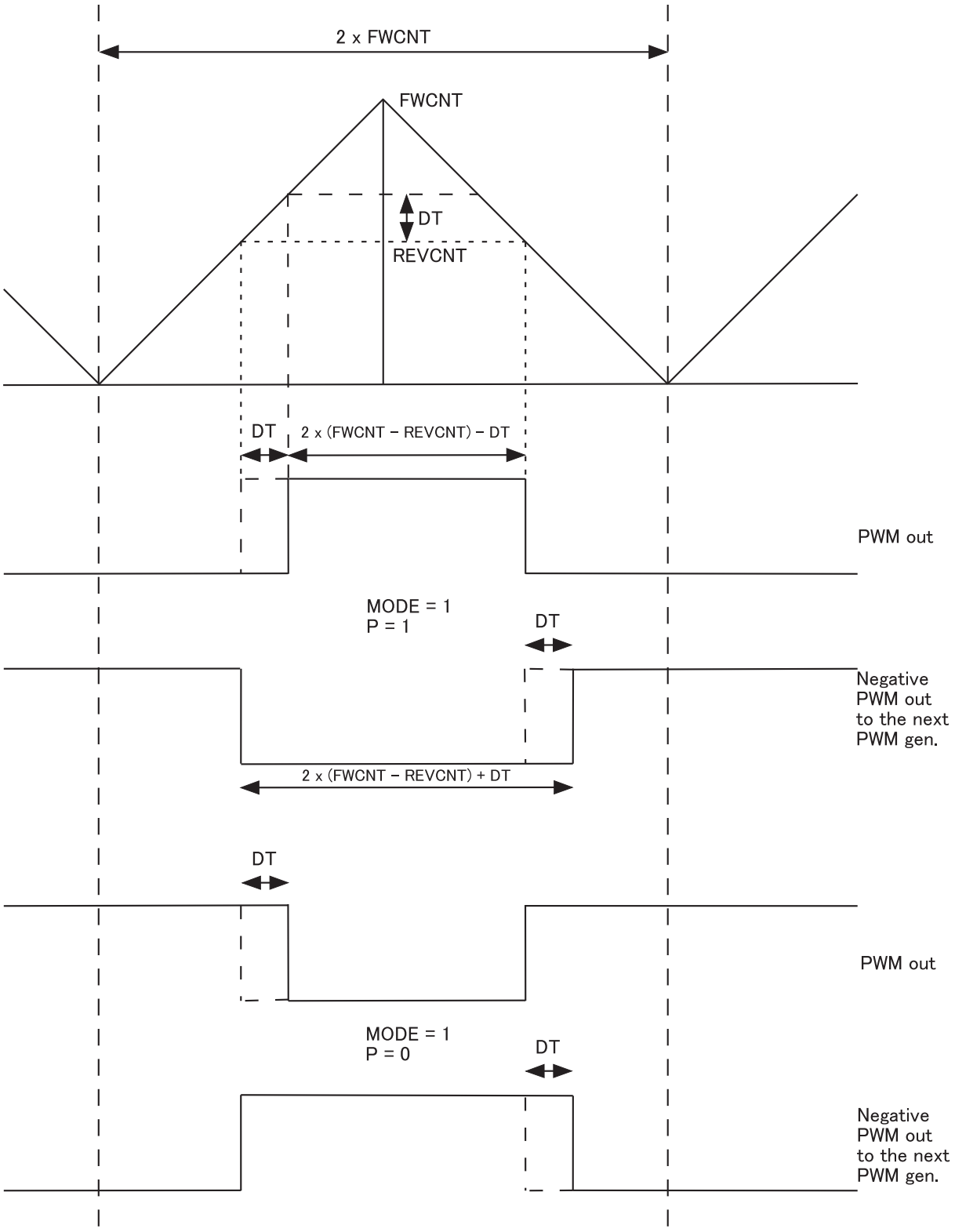


Figure 15.2: Triangle Wave Mode

16

PWM Input

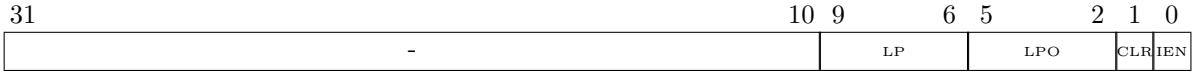
16.1 Abstract

- Digitize the duty ratio of PWM input to the ratio of High counter and Low counter
- Bit Width : 32bit
- Number of channels : 3
- Count using base clock generated by Clock Generator (refered in Section 7)
- Mechanism that finds and averages duty ratios of multiple cycles
- Interrupt generation

16.2 PWMIN Controll Register

Address	PWMIN Control Register
0xFFFF7400	PWMINCTRL[0]
0xFFFF7420	PWMINCTRL[1]

Read / Write

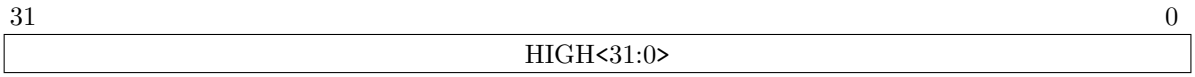


Field Name	Function
IEN	Interrupt Enable :Default 0 r/w 0: Disable interrupt. 1: Interrupts are generated each time after counting the duty ratio for the set period.
CLR	Interrupt Clear :Default 0 r/w 0:Unclear interrupt. 1:Clear interrupt. Reset to 0 after complete to clear interrupt.
LPO	Loop Original :Default 1 r/w Set how many cycles of the duty rato is averaged. (from 1 to 15. 0 is prohibited)
LP	Loop :Default 1 ro Dedicate cycles now executing.

16.3 PWMIN HIGH Register

Address	PWMIN HIGH Register
0xFFFF7404	HIGH[0]
0xFFFF7424	HIGH[1]

Read

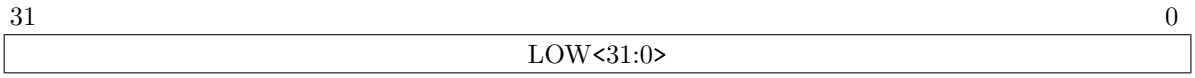


Field Name	Function
HIGH<31:0>	High :Default X High period of the total for the specified PWM period. The unit is the number of PWMIN clock cycles programmably set by the clock generator.

16.4 PWMIN LOW Register

Address	PWMIN LOW Register
0xFFFF7408	LOW[0]
0xFFFF7428	LOW[1]

Read / Write



Field Name	Function
LOW<31:0>	Low :Default X Low period of the total for the specified PWM period. The unit is the number of PWMIN clock cycles programmably set by the clock generator.

17

Flash I/F

17.1 Abstract

About boot from the I/F. flash and access to the flash.

17.2 Address Space

Address spaces of the Flash and the ROM is changed by out_cs_toggle_signals.

When out_cs_toggle_signal are High, the ROM address space is EXT_0(0x00000000　0x00ffffff), the Flash address space is EXT_1(0x20000000　0x20ffffff). When out_cs_toggle_signal are Low, the ROM address space is EXT_1, the flash address space is EXT_0,

If boot from the flash, deassert out_cs_toggle_signals.

In order to connect the flash with EXT_1, set the 13th bit of the system register (0x8f) to 1, which is default. In order to connect the flash with other I/O, set it to 0.

17.3 Access

Read and write by issuing command from a software to the flash. Two M29W128G of the flash are connected with the 32bit bus. Access with 32bit/16bit/8bit unit. The default is 16bit unit, to set to 8bit mode, use the option register. Details of commands is in the M29W128G manual.

About only normal writing (word access, byte access), by using the option register (Auto Write Enable), the write command is issued by the hardware. In this case, can write with direct addresses. Command issuing before data writing does not need.

17.4 Flash I/F Option Register

The address map of the option register of Flash I/F is assigned to EXT_4 (0x23000000　0x23ffffff).

EXT_4 is the exclusive address space for the option register of I/O interfaces. An external control signal does not exist.

Auto Write Enable

offset: 0x00

31	1	0
Reserved		EN

Field Name	Range	Description
EN	0	Default : 0 When this bit is 1, the hardware issues write commands instead of software. Have to set this bit to 1 in order to DMA-transport including the flash.

Byte Mode

offset: 0x04

31	1	0
Reserved		Byte

Field Name	Range	Description
Byte	0	Default : 1 0 is for a byte unit writeing. 1 is for 16byte/32byte unit writing.

18

Universal Asynchronous Receiver/Transmitter

Initial Address: Channel0:0xffff6000 Channel1:0xffff6080

18.1 Address Map

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

18.1.1 Receiver Buffer (RB) / Transmitter Holding Register (THR)

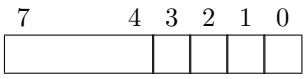
Offset: 0x0000

7	0

Field Name	Function
7-0	Transmit FIFO Input and Recieve FIFO Output.

18.1.2 Interrupt Enable Register (IER)

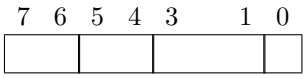
Offset: 0x0004



Field Name	Function
0	Received Data availble interrupt. ‘0’ - Disabled. ‘1’ - Enabled.
1	Transmitter Holding Register empty interrupt. ‘0’ - Disabled. ‘1’ - Enabled.
2	Receiver Line Status Interrupt. ‘0’ - Disabled. ‘1’ - Enabled.
3	Modem Status Interrupt. ‘0’ - Disabled. ‘1’ - Enabled.
7-4	Reserved. Should be logic ‘0’.

18.1.3 Interrupt Identification Register (IIR)

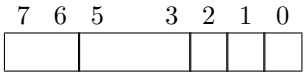
Offset: 0x0008



<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>

18.1.4 FIFO Control Register (FCR)

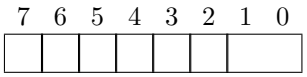
Offset: 0x0008



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

18.1.5 Line Control Register (LCR)

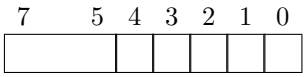
Offset: 0x000c



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.

18.1.6 Modem Control Register (MCR)

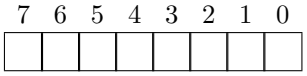
Offset: 0x0010



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.

18.1.7 Line Status Register (LSR)

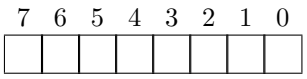
Offset: 0x0014



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.

18.1.8 Modem Status Register (MSR)

Offset: 0x0018

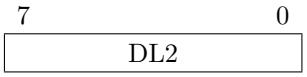


Field Name	Function
0	Delta Clear To Send (DCTS) indicator. ‘1’ - The CTS line has changed its state.
1	Delta Data Set Ready (DDSR) indicator. ‘1’ - The DSR line has changed its state.
2	Trailing Edge of Ring Indicator (TERI) detector. The RI line has changed its state from low to high state.
3	Delta 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.

18.1.9 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.

18.2 Function / Usage

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.

18.2.1 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.

19

Parallel I/O Unit

19.1 Outline

Parallel I/O Unit provides 8 bit inputs and outputs.

19.2 Interface

19.2.1 Address Format

The initial base address of Parallel I/O Unit is 0xffffc000. The addresses of control regiters of Parallel I/O Unit is below.

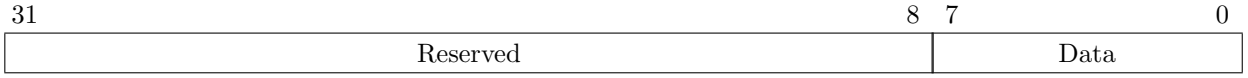
Field Name	Range	Description
Offset	4:0	Specify an item to set up.

19.2.2 Control Register

When Controling Parallel I/O Unit, access the register by specifying offset shown below.

Data

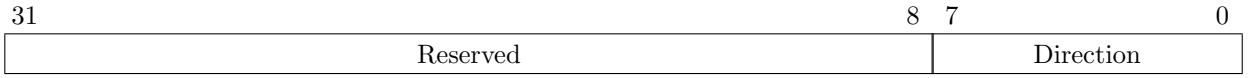
offset: 0x00



Field Name	Range	Description
Data	7:0	When bit is input, get input from outside by reading. When bit is output, give output to outside by writing.

Direction

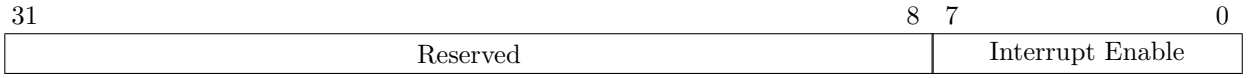
offset: 0x04



Field Name	Range	Description
Direction	7:0	When 0, input. When 1, output.

Interrupt Enable

offset: 0x08



Field Name	Range	Description
Interrupt Enable	7:0	When 1, generate an interruption when changing value corresponding with Data Register, Conditions of interruptions is set by Interrupt Up-edge Register and Interrupt Downedge Register.

Interrupt Sense

offset: 0x0c

31	8	7	0
Reserved		Interrupt Sense	

Field Name	Range	Description
Interrupt Sense	7:0	Set the bit which caused the interrupt to 1. When writing 1 into this register, clear corresponding bit.

Interrupt Upedge

offset: 0x10

31	8	7	0
Reserved		Interrupt Upedge	

Field Name	Range	Description
Interrupt Upedge	7:0	When 1, generate an interruption when data changes from 0 to 1.

Interrupt Downedge

offset: 0x14

31	8	7	0
Reserved		Interrupt Downedge	

Field Name	Range	Description
Interrupt Downedge	7:0	When 1, generate an interruption when data changes from 1 to 0.

Configuration offset: 0x18 (Read Only)

31	2	1	0
Reserved			BW

Field Name	Range	Description
Bit Width (BW)	1:0	Show Bit Width of Parallel I/O. 0x1 : 8 Bit 0x2 : 16 Bit 0x3 : 32 Bit

19.3 Operation

Parallel I/O have 8 bit length, and can set I/O direction for each bit at Direction Register.

When input, Data of I/O pin is latched in Data Register by the clock which is given to Parallel I/O . When output, output value of Data Register to I/O pin.

Each bit can generate an interruption based on specified conditions. to generate an interruption, set the corresponding bit of Interrupt Enable Register to 1. And set the corresponding bit of Interrupt Upedge Register or Interrupt Downedge Register to 1. When set both to 1, generating an interruption for every change of value.

20

External Bus

20.1 Specification

Same to internal bus access extrenal bus is accessed by big endian. For example, 0x00000000 corresponds with BE_[3], 0x00000001 with BE_[2], 0x00000002 with BE_[1], and 0x00000003 with BE_[0]. BMREQ_ is a burst request signal which needs to specify the burst length. Below are the supported burst lengths.

BMREQ == 11: 1Word, BMREQ == 10: 2Word BMREQ == 01: 4Word, BMREQ == 00: 8Word

With BMACK_ the slave outputs burstable length.

The figures below shows exmaple of timings of the external bus.

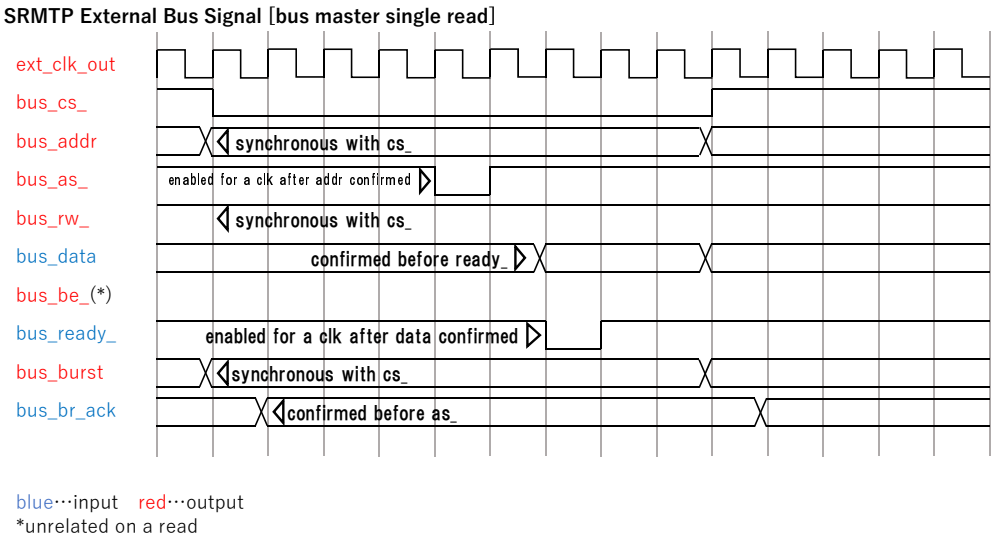


Figure 20.1: Read

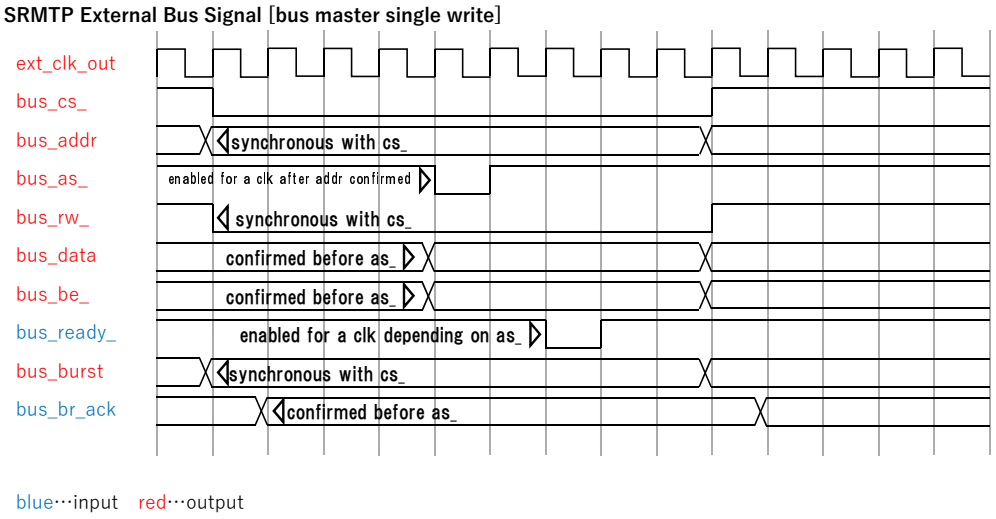


Figure 20.2: Write

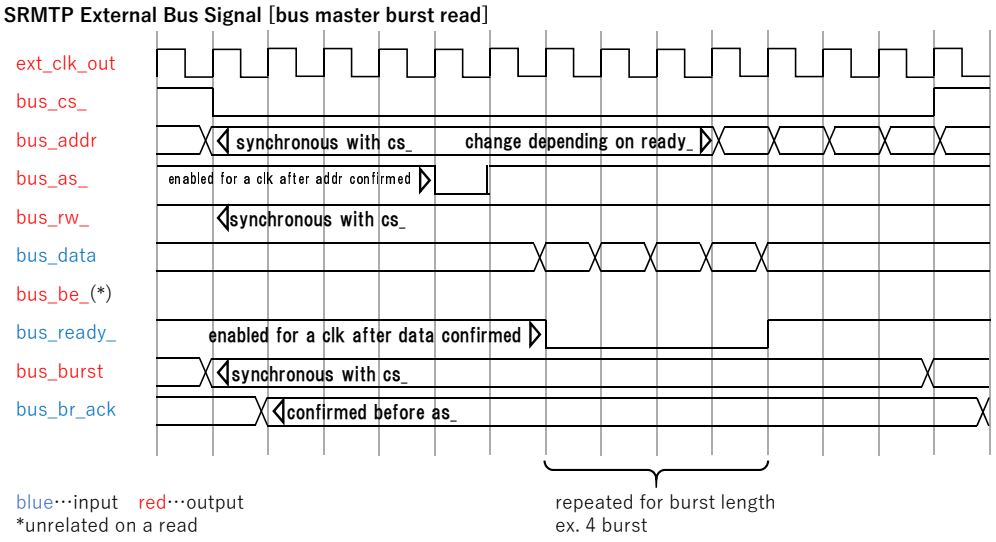


Figure 20.3: Burst Read

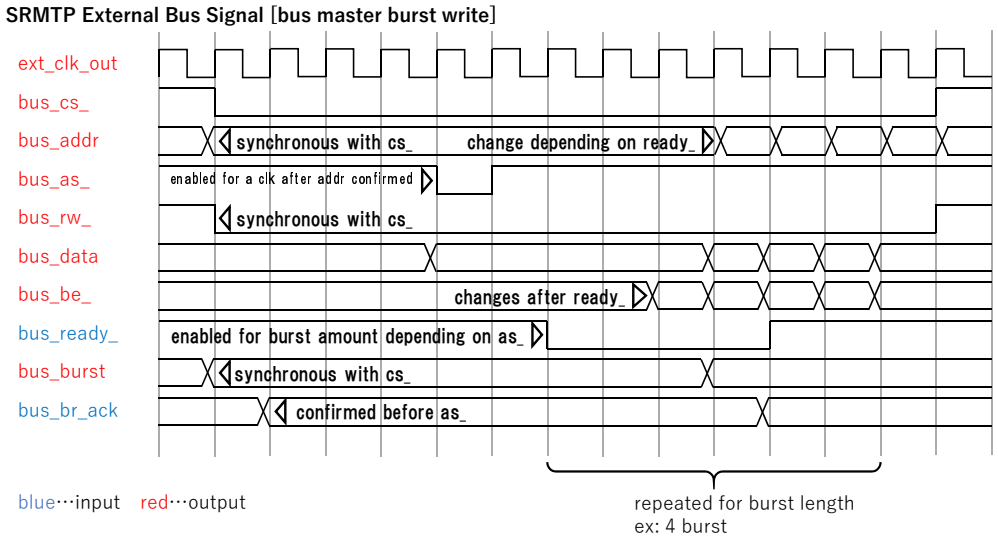


Figure 20.4: Burst Write

20.2 EXT_0(ROM)

When accessing EXT_0 (ROM) using 8 or 16 bit mode, be[0] and be[1] are used as the lower address. The LSB is be[0].

20.3 Default Memory Map

cs_0 0x00000000(for ROM only)

cs_1 0x20000000(Flash I/F)

cs_2 0x21000000(SiP FPGA)

cs_3 0x22000000(Board FPGA)

cs_4 0x23000000(REGFILE, LED)

cs_5 0x24000000

cs_6 0x25000000

cs_7 0x26000000

Caution: When asserting cs_toggle, cs_0 and cs_1 are swapped. Auto ready is only available in cs_0.

21

Real Time Clock Unit

21.1 Outline

RTC module give calender function that work from external clock. This module measure 1 second from external clock and control counters. Calender function give information of year$Under2digits$, month, date, weekday, hour, minute and second. Furthermore, this module have a alarm function that cause an interruption in user defined timing. This module have a timer function that can set in second order. Calender function adapt leap years.

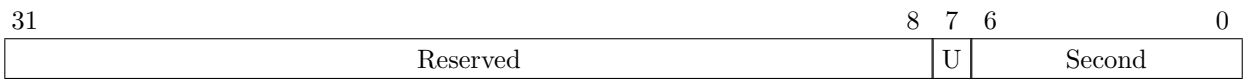
External pin rtc_clk is clock input. Default frequency is 32.768kHz. However, other frequency can be handled by changing value of Clock Compare register.

If external pin rtc_hold_ is asserted, this module ignore signal from processor internal bus. As a result, Real Time Clock Unit can measure accurate time in case of power trouble.

21.2 Interface

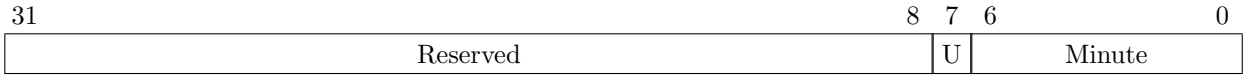
21.2.1 Address Map

Second offset: 0x00



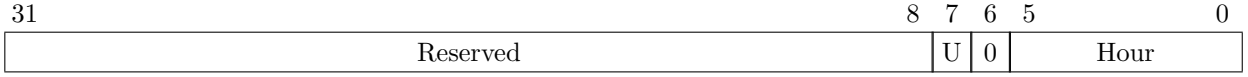
Field Name	Range	Description
Update (U)	7	This bit set in case the value is update from last read. This bit set to 0 by reading.
Second	6:0	This bits represents BCD code of second.

Minute offset: 0x04



Field Name	Range	Description
Update (U)	7	This bit set in case the value is update from last read. This bit set to 0 by reading.
Minute	6:0	This bits represents BCD code of minute.

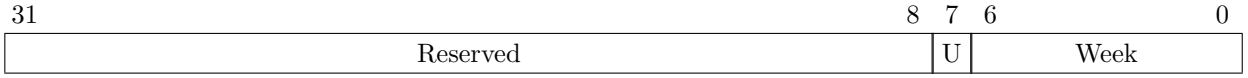
Hour offset: 0x08



Field Name	Range	Description
Update (U)	7	This bit set in case the value is update from last read. This bit set to 0 by reading.
Hour	5:0	This bits represents BCD code of hour.

Week

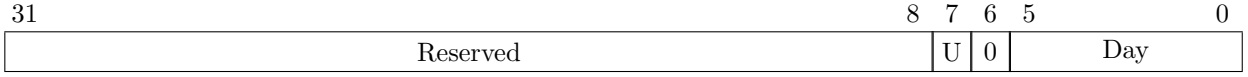
offset: 0x0c



Field Name	Range	Description
Update (U)	7	This bit set in case the value is update from last read. This bit set to 0 by reading.
Week	6:0	This bits represents weekday. MSB is Saturday, LSB is Sunday

Day

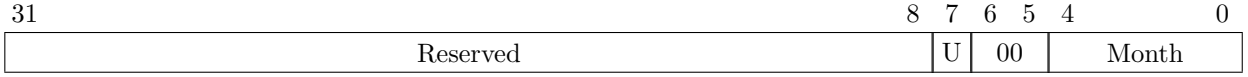
offset: 0x10



Field Name	Range	Description
Update (U)	7	This bit set in case the value is update from last read. This bit set to 0 by reading.
Day	5:0	This bits represents BCD code of date.

Month

offset: 0x14



Field Name	Range	Description
Update (U)	7	This bit set in case the value is update from last read. This bit set to 0 by reading.
Month	4:0	This bits represents BCD code of month.

Year offset: 0x18

31	8	7	0
Reserved		Year	

Field Name	Range	Description
Year	7:0	This bits represents BCD code of year.

Second Alarm offset: 0x20

31	8	7	6	0
Reserved		D	Second	

Field Name	Range	Description
Don't Care (D)	7	If set to 1, ignore second status. If 0, an alarm interrupt occurs when the condition specified in Second is matched.
Second	6:0	Specify the second of the alarm. The value is expressed by BCD code.

Minute Alarm offset: 0x24

31	8	7	6	0
Reserved		D	Minute	

Field Name	Range	Description
Don't Care (D)	7	If set to 1, ignore minute status. If 0, an alarm interrupt occurs when the condition specified in Minute is matched.
Minute	6:0	Specify the minute of the alarm. The value is expressed by BCD code.

Hour Alarm

offset: 0x28

31 … 8	7	6	5 … 0
Reserved	D	0	Hour

Field Name	Range	Description
Don't Care (D)	7	If set to 1, ignore hour status. If 0, an alarm interrupt occurs when the condition specified in Hour is matched.
Hour	5:0	Specify the hour of the alarm. The value is expressed by BCD code.

Week Alarm

offset: 0x2c

31 … 8	7	6 … 0
Reserved	D	Week

Field Name	Range	Description
Don't Care (D)	7	If set to 1, ignore weekday status. If 0, an alarm interrupt occurs when the condition specified in Week is matched.
Week	6:0	Specify the weekday of the alarm.

Day Alarm

offset: 0x30

31	8	7	6	5	0
Reserved		D	0	Day	

Field Name	Range	Description
Don't Care (D)	7	If set to 1, ignore date status. If 0, an alarm interrupt occurs when the condition specified in Day is matched.
Day	5:0	Specify the date of the alarm. The value is expressed by BCD code.

Month Alarm

offset: 0x34

31	8	7	6	5	4	0
Reserved		D	00		Month	

Field Name	Range	Description
Don't Care (D)	7	If set to 1, ignore month status. If 0, an alarm interrupt occurs when the condition specified in Month is matched.
Month	4:0	Specify the month of the alarm. The value is expressed by BCD code.

Time

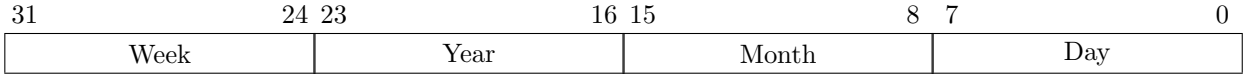
offset: 0x38 (Read Only)

31	24	23	16	15	8	7	0
Reserved		Hour		Minute		Second	

Field Name	Range	Description
Hour	23:16	Value of Hour register.
Minute	15:8	Value of Minute register.
Second	7:0	Value of Second register.

Date

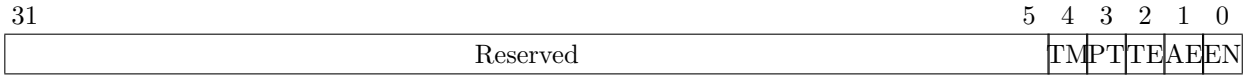
offset: 0x3c (Read Only)



Field Name	Range	Description
Week	31:24	Value of Week register.
Year	23:16	Value of Year register.
Month	15:8	Value of Month register.
Day	7:0	Value of Day register.

Mode

offset: 0x40



Field Name	Range	Description
Test Mode (TM)	4	Test mode bit. Set to 0.
Periodic Timer (PT)	3	If set to 0, timer works periodically. If set to 1, timer works one shot mode.
Timer Enable (TE)	2	If set to 1, timer is enabled. If timer works on one shot mode, this bit set to 0 when timer ie expired.
Alarm Enable (AE)	1	If set to 1, alarm is enabled. An interrupt occurs when the time specified for the alarm comes.
Enable (EN)	0	If set to 1, real time clock module is enabled.

Sense offset: 0x44

31	2	1	0
Reserved		TI	AI

Field Name	Range	Description
Timer Interrupt (TI)	1	This bit is set to 1 when timer interrupt occurs. Cleared by writing 1.
Alarm Interrupt (AI)	0	This bit is set to 1 when alarm interrupt occurs. Cleared by writing 1.

Timer Compare offset: 0x48

31	0
Timer Compare	

Field Name	Range	Description
Timer Compare	31:0	Specify the number of seconds of timer interrupt.

Timer Count offset: 0x4c (Read Only)

31	0
Timer Count	

Field Name	Range	Description
Timer Count	31:0	Number of timer count. Timer interruption is occur when this value is equaled to Timer Setup.

Clock Compare

offset: 0x50

31	0
Clock Compare	

Field Name	Range	Description
Clock Compare	31:0	This register specify input clock frequency. When the Clock Count equal to the value of thie register, this module count 1 second.

Clock Count

offset: 0x54 (Read Only)

31	0
Clock Count	

Field Name	Range	Description
Clock Count	31:0	It count up every clock and measures 1 second. When the Clock Compare equal to the value of thie register, this module count 1 second.

22

Trace Buffer

22.1 Abstract

Trace Buffer memorizes instructions, registers, and exceptions the processer excuted.

22.2 Address Map

The initial base address of Trace Buffer is 0x10000000.

0x1000_0000∼0x1000_00FF	:	Field of Control Register
0x1004_0000∼0x1004_FFFF	:	Field of Trace Buffer

22.3 Field of Control Register

Trace PC Control

offset: 0x20

Control intruction tracings.

31	1	0
Reserved		E

Field Name	Range	Description
Enable (E)	0	PC tracing is enabled when this bit is set to 1.

Trace Register Control

offset: 0x40, 0x44

Control register tracing.

31 … 4	3 … 1	0
Reserved	TH	E

Field Name	Range	Description
Thread (TH)	3:1	Registers of the thread set in this field are traced.
Enable (E)	0	Register file tracing is enabled when this bit is set to 1.

Trace Exception Control

offset: 0x60, 0x64

Control exception tracing.

31 … 4	3 … 1	0
Reserved	TH	E

Field Name	Range	Description
Thread (TH)	3:1	Exceptions of the thread set in this field are traced.
Enable (E)	0	Register file tracing is enabled when this bit is set to 1.

Trace Buffer Initialize

offset: 0x80

Initialize the trace bufffer.

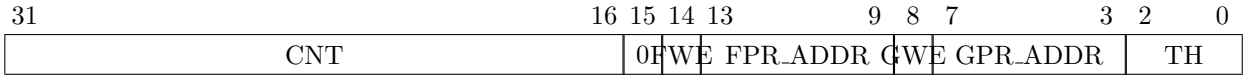
31 … 0
Initialize

Field Name	Range	Description
Initialize	0	The trace buffer is initialized by writing to this register.

22.4 Trace PC Buffer 領域

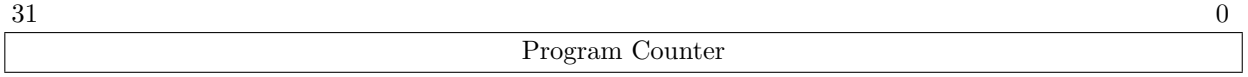
The format of trace information is shown below.

Trace PC Buffer 0 offset: 0x0



Field Name	Range	Description
Counter (CNT)	31:16	Serial number.
FPR Write Enable(FWE)	14	This bit is assert when an instruction writes to FPR.
FPR Address(FPR_ADDR)	13:9	This field is set to the register number when an instruction writes to FPR.
GPR Write Enable(GWE)	8	This bit is assert when an instruction writes to GPR.
GPR Address(GPR_ADDR)	7:3	This field is set to the register number when an instruction writes to GPR.
Thread(TH)	2:0	This field is set to the thread number executing.

Trace PC Buffer 1 offset: 0x4



Field Name	Range	Description
Program Counter	31:0	This field is set to value of PC of the executed instruction.

Trace PC Buffer 2 offset: 0x8

31	0
Register Write Data 0	

Field Name	Range	Description
Register Write Data 0	31:0	When writing to GPR happens, this field stores that GPR data. When writing to FPR, this field stores upper half of that data (32bit). When any register is not writen, this field is skipped.

Trace PC Buffer 3 offset: 0xC

31	0
Register Write Data 1	

Field Name	Range	Description
Register Write Data 1	31:0	When writing to FPR, this field stores lower half of that data (32bit). When any register is not writen, this field is skipped.

23

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	2020/04/17	Fixed figure and abstract and some specification