PA7100LC ERS

March 30, 1999
Public version 1.00

HEWLETT
PACKARD

Notice

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THE MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with furnishing, performance, or use of this material.

Hewlett-packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-packard.

Contents

Chapter 1

PA7100LC System Overview

1.1 PA7100LC CPU

The PA7100LC is a core PA-RISC processor designed for low cost system applications. The PA7100LC CPU (central processing unit), FP (floating point unit), MIOC (Memory and I/O Controller), and a first-level instruction cache are all fabricated on a single VLSI chip. The design is focused on the primary goals of low cost and leadership price/performance at the system level. The PA7100LC uses standard off-the-shelf SRAMs and DRAMs to form a complete processor/memory subsystem with performance levels comparable to 1991 high-end desktop workstations and servers.

The CPU and FP support PA-RISC1.1 Edition 3 features including non-cached memory pages and also supports little-endian mode for all loads and stores, and for instruction fetches from memory space. PA7100LC has addded implementation dependent instructions to support multimedia applications. These instructions operate on 16-bit operands packed into the 32-bit general registers. PA7100LC is 2-way superscalar, providing the ability to execute integer-integer and integer-FP bundles.

The chip is fabricated in HP's proprietary CMOS26B technology which features 0.75 micron devices and 3 levels of aluminum interconnect, and will be packaged in a 432 pin PGA from Kyocera. The PA7100LC is designed to support system configurations with a range of cache and memory sizes running at several different speeds.

The PA7100LC is designed for single-processor only configurations and interfaces to the GSC bus. The first product platform for PA7100LC is the 712. The 712 will include the LASI chip to support I/O devices (LAN, SCSI, audio, phone, etc) and Crayon (including Artist chip for graphics) to support the user-interface.

1.2 Technology and Specifications

PCX-L CPU

Clock Frequency	60-100 MHz
Performance (estimated)	50-60 Specmarks @ 60 MHz
Power Supplies	Vdd = 5.00V +- 0.25V Vdl = 3.30V +- 0.20V
I/O Levels	TTL compatible
External SRAM access time (standard TTL I/O SRAM)	12ns @ 60 MHz 10ns @ 75 MHz 8ns @ 100 MHz

1.3 General Features

On-Chip Instruction Cache	1 Kbyte 64 bit access, direct-mapped, Prefetch from Off-Chip Icache
Combined Instruction and Data Off-Chip Cache	8 Kbytes - 2 Mbytes, 64 bit access, direct-mapped, hashed address, virtual index. 480-600 Mbyte/s bandwidth.
MMU	64 entry unified I/D TLB, fully associative, NUR replacement, 4K page size TLB, 1-entry ITLB lookaside buffer, 8-entry, 512K - 64M BATC.
Coprocessor	On-Chip, FALU (dp), MUL (dp), DIV/SQRT (srt).
System Interface	GSC
Test/Diagnostics Support	JTAG (IEEE 1149.1-1990) interface. Serial scan path. Operation down to DC. Single-step and N-step.
SuperScalar INT/INT and INT/FP execution.	
Implementation-dependent multimedia (pixel op) support.	
Instruction and Data Cache Bypassing.	
Stall-on-Use Data Cache Miss policy.	
Don't fill on Miss Cache Hint.	
Load and Clear optimizations.	
Read-Tag Write-Data Stores.	
Hardware TLB Miss Handler Support.	
Hardware Static Branch Prediction.	
Instruction Line Prefetch from Memory.	
Parity error detection on I/D caches	
Level 1 PA implementation (48 bit virtual address).	
Up to 4 Gbytes physical memory.	
32 byte cache line size, copyback STORE policy.	

Chapter 2

PA7100LC Features

2.1 New for PA7100LC

2.1.1 Integer SuperScalar

The PA7100LC CPU has added a second integer ALU and extra general register ports so that more instruction combinations are eligible to be executed as a bundle in a single cycle. The following instruction pairs can be executed (without regard to order or doubleword alignment): load-store/integer, integer/integer, load-store/floating-point, integer/floating-point. Branch instructions can be bundled, but they are restricted to be the "newer" of the two bundled instructions: integer/branch, floating-point/branch, and load-store/branch. Certain restrictions do apply, especially due to register source/target conflicts, branch, nullification, and PSW conflicts. In very special cases, PA7100LC can bundle ldw-ldw or stw-stw instructions to create a single "doubleword access" to cache. Please see the chapter on SuperScalar execution for more details.

2.1.2 Uncached Memory Pages

The PA-RISC1.1 Edition 3 Instruction Manual specifies an optional U-bit in the TLB entry for each page to determine whether virtual accesses to a page may be cached. PA7100LC implements the U-bit and software may exploit this feature to optimize memory accesses that are shared between the CPU and I/O devices. For more information, see Appendix.

2.1.3 Little Endian Mode

PA7100LC provides a mode that enables all loads and stores to use little endian byte ordering rather than the PA-RISC default of big endian (as specified in PA-RISC1.1 Edition 3). The PSW-E bit (not part of the system mask) is used to control whether data is accessed as big or little endian. When an interrupt (trap) occurs, the PSW-E bit is set from a diagnose register that specifies the "default" endian mode. When an RFI occurs the PSW-E bit is restored from the IPSW-E bit. Note that software can change the value of the PSW-E bit by changing the IPSW and executing an RFI, but changing the default endian mode requires execution of diagnose instructions. PA7100LC has also implemented 9 implementation dependent load/store instructions that override the PSW-E bit. As of rev2.0, PA7100LC also supports little-endian instruction fetching. For more information, see Appendix.

2.1.4 Multimedia Halfword Arithmetic

PA7100LC provides support for halfword arithmetic in both of its integer ALU's. These are implementation dependent instructions that are encoded in the "majop 02 - arith/logic" instruction space by using currently undefined minor opcode bits. Support is provided for "halfword add", "halfword subtract", "halfword average", and "halfword shift-and-add". Two halfword operands can be encoded into a 32-bit general register, and both ALU's can operate simultaneously. Thus, PA7100LC can calculate 4 halfword results per cycle. For more information, see Appendix.

2.1.5 Floating Point load/store to I/O

The PA-RISC architecture specifies that floating point load and store instructions to I/O space produce undefined results. In order to improve graphics performance, the PA7100LC CPU does perform floating point loads and stores (including both word and doubleword instructions) with defined results. When the FSTWS, FSTWX, FSTDS, FSTDX, FLDWS, FLDWX, FLDDS, FLDDX instructions access I/O space, data is transferred between a floating point register and an I/O space address. The FP store instructions to I/O space incur a one cycle unconditional penalty cycle just like normal I/O space stores. The FP load perform as well as the normal loads from I/O space.

2.1.6 ITLB Hardware Handler and CR28

PA7100LC provides a pointer in control register 28 for both ITLB and DTLB miss traps (traps 6 and 15). PA7100 provided CR28 only for DTLB miss traps. Note: we found a post-silicon bug that makes CR28 unreliable for the ITLB miss handler.

2.2 Leveraged from PA7100/earlier processors

2.2.1 Fast TLB Inserts

PA7100LC supports the "fast" TLB insert instructions invented on PA7100. These instructions insert to the virtual address in the ISR/IOR for DTLB inserts, and IIASQ/IIAOQ for ITLB inserts. In addition, these instructions execute with less penalty cycles than the normal TLB insert instructions. However, certain restrictions on how these instructions are used are required. For more information, see Appendix.

2.2.2 Shadow Registers/RFIR

PA7100LC implements shadow registers on GR's 1,8,9,16,17,24,25. The shadow registers are "backup" copies of the GR's that get set on all traps and can be restored with the RFIR instruction. The RFIR instruction functions exactly like the RFI instruction, except for the restoration of the shadow registers. These features have been used to successfully reduce the software TLB miss penalty.

2.2.3 PA1.1 Floating Point

PA7100LC implements the Floating Point features specified in PA-RISC1.1. Therefore the new opcodes 06, 26, and 0E are all recognized as per that spec.

2.2.4 Graphics Clip Test

PA7100LC implements extra FTEST variants for graphics clip test operations. These are undefined sub-opcodes of the 0C major opcode (and in some cases, of the 0E major opcode as well). See the floating point chapter for more details.

2.2.5 LDCW and Store Hints

PA7100LC implements the optional load-and-clear hints specified in PA-RISC1.1. PA7100LC also implements the store hints for processes executing at privilege level 0 (store hints are ignored at other privilege levels). Therefore, a load-and-clear that "hits dirty" in the cache does not need to issue a memory transaction, and stores do not necessarily cause a memory transaction when the hint is specified and a cache miss occurs. For more information, see the Dcache section of the pipeline chapter.

2.2.6 Pipelined (2 state) Stores

PA7100LC implements stores to the data cache similarly to PA7100. The read-tag and write-data operations necessary for the store are pipelined so that stores effectively busy the cache for 2 states. Therefore, store instructions incur a 1 cycle penalty if the following instruction bundle contains a load or store instruction.

2.2.7 Unprivileged Reads of CR26, CR27

PA7100LC allows nonprivileged users to read control registers 26 and 27 as specified in PA-RISC1.1.

2.2.8 Trap 18 Replaced with 26,27,28

PA7100LC replaces the data memory protection trap (trap 18) with a data memory access rights trap (trap 26), data memory protection ID trap (trap 27), and an unaligned data reference trap (trap 28). This is per PA-RISC1.1.

2.2.9 Smaller Caches/TLB

PA7100LC has a small on-chip instruction cache (1 Kbyte) and a combined off-chip cache (8k-2M). PA7100LC has a 64 entry TLB with 8 block translation entries.

2.2.10 Changes to Floating Point

PA7100LC double precision multiplies incur one state of extra penalty compared to single precision multiplies. The PA7100LC floating point also has reduced pipelining capability for divide and square root operations. Any divide or square root will halt execution of the CPU pipeline until the divide or square root operation is finished.

2.2.11 No MP Support

PA7100LC does not support any MP configurations.

2.2.12 No Graphics Flush Instructions

PA7100LC does not support any of the Venom functionality provided with Viper-based systems. In particular, the graphics flush instructions are not implemented and will cause an illegal instruction trap.

Chapter 3

SuperScalar Execution

3.1 2-Way SuperScalar

PA7100LC is capable of executing two instructions at a time. The instructions proceed together through the execution pipeline and are said to be *bundled.* Superscalar execution is functionally transparent to software, that is the effects of an instruction are the same whether it was executed alone or as part of a superscalar bundle.

There are four kinds of restrictions placed upon bundling: functional unit contention, data dependency restrictions, control flow restrictions, and special instruction type restrictions. These are all described in detail below.

The bundling rules are applied entirely at run time by hardware. Compilers and hand-coders seeking performance will want to order their instructions so that bundling is maximized, but they are not required to do so. The only side-effect of sub-optimal instruction ordering is lower performance.

3.2 Instruction Classes

For the purpose of these bundling rules the instruction set is divided into classes. Below is a list of the classes and which opcodes fall into them. The opcodes are given as hex values for bits 0:5. Conditions in parentheses refer to values of particular bits of the instruction.

class	encoding	description
flop	0C(26==0), 0E, 06, 26	floating-point operations
ldw	12	just LDW
stw	1A	just STW
ldst	09/0B(27:29!=0),	FP loads and stores
	03, 10, 11, 13-19, 1B-1F	other loads and stores
flex	08, 0A, 0D,	integer ALU
	02/24/25/2C/2D(16:19==0)	
mm	34/35(16:18==0),	shifts, extracts, deposits
	27, 2E, 2F, 36, 37, 3C-3F	
nul	02/24/25/2C/2D(16:19!=0),	might nullify successor
	34/35(16:18!=0)	
bv	38, 3A(16==1)	BE, BV
br	20-23, 28-2B, 30-33,	other branches
	3A(16==0), 3B	
fsys	0C(26==1), 09/0B(27:29==0)	FTEST and FP status/exception
sys	00, 01, 04, 05, 07, 0F, 39	system control instructions

3.3 Functional Unit Contention

These kinds of bundles are allowed:

flop	ldw + stw + ldst + flex + mm + nul + bv + br
ldw	flop + flex + mm + nul + br
stw	flop + flex + mm + nul + br
ldst	flop + flex + mm + nul + br
flex	flop + ldw + stw + ldst + flex + mm + nul + br + fsys
mm	flop + ldw + stw + ldst + flex + fsys
nul	flop
ldw	ldw(*)
stw	stw(*)

The ldw/ldw and stw/stw bundles are a special case called double word load/store. The effective addresses of the data references of the two instructions must point to different words of an aligned doubleword. The two instructions must use the same space and base register and the base register must contain an even word address (its bit 29 must be 0).

3.4 Data Dependencies

An instruction which modifies a register will not be bundled with a subsequent instruction which uses that register as an operand, except for the special case of a flop bundled with a floating point store of the flop's result register.

A floating point load to one word of a doubleword floating-point register will not be bundled with a flop which uses the other word.

A flop will not be bundled with a floating-point load if the flop and load have the same target register, or even if their targets are different words of the same doubleword register.

An instruction which might set the carry/borrow bits will not be bundled with an instruction which uses the carry/borrow bits. For this purpose, the instructions which might set carry/borrow are opcodes 24, 25, 2C, and 2D, and opcode 02 with bit 21 equal to 1 and either bit 23 equal to 0 or bits 24 and 25 equal to 0. The instructions which use carry borrow are SUBB, SUBBO, ADDC, ADDCO, DS, DCOR, and IDCOR.

3.5 Control Flow

An instruction which is in the delay slot of a branch is never bundled.

An instruction which is executed as the target of a taken branch and which is at an odd word address is never bundled.

As shown in the bundle table above, an instruction which *might* nullify its successor will not be bundled with that successor, unless that successor is in the *flop* class. Specifically the following types of instructions might nullify their successor: opcode {02,24,25,2C,2D,34,35} with a nonzero test condition (bits 16:19 for {02,24,25,2C,2D}, or bits 16:18 for {34,35}).

A ldw/ldw or stw/stw pair is not bundled if the previous instruction or instruction pair might nullify. For this purpose the instructions which might nullify are those mentioned above plus the ftest instruction. The nullifying instruction would have to be right before the first ldw or stw, or seperated from it only by a flop.

An instruction which is executed as the first target of an RFI or RFIR is never bundled. In addition, the second target is never bundled if it is at an odd word address.

3.6 Special Instruction Types

Instructions in the *sys* class are never bundled.

Chapter 4

Coding for Optimal Performance

System performance is based on the code pathlength (number of instructions) times the instruction CPI (cycles per instruction). Software should attempt to reduce the instruction CPI in addition to the code pathlength.

This section summarizes ways in which code can be optimized for performance for PA7100LC. The pipeline freeze operation section of this chapter gives more details on the actual number of penalty states for the different freeze conditions. The purpose of this section is to summarize the performance penalties and how they can be reduced. Note that many of these issues apply to other PA-RISC implementations.

4.1 SuperScalar Execution

PA7100LC's peak instruction execution rate is two instructions per cycle. A second integer ALU and extra GR ports have been added so that integer/load-store, integer/integer, and integer/branch bundles can be executed in a single CPU cycle. Also, load/load and store/store bundles can be executed in one cycle when certain instruction and addressing criteria are met. This is in addition to the bundles supported on PA7100, including: load-store/flop, int/flop. Various rules affect whether two instructions can be executed concurrently. Please see the SuperScalar chapter for more details.

4.2 Store Instructions

Store instructions to memory space incur a one penalty cycle only when the next instruction bundle accesses the D-Cache (load, store, D-flush, D-Purge, Load and Clear). Note that when instructions are bundled (superscalar execution) it is necessary to increase the distance between a Store and the next D-Cache access in order to not incur a penalty cycle. DCache miss penalties for store instructions can be minimized for privileged software by using "store hints". See the DCache miss section of this chapter.

Store instructions to I/O space also benefit from having a non-D-Cache instruction immediately following it. There is always one unconditional penalty cycle for each I/O store instruction and one more penalty cycle if it is followed immediately by a D-Cache instruction.

4.3 Branch Instructions

All conditional PC relative branches are predicted taken if they are backward and are predicted untaken if they are forward. Code can take advantage of this convention. Unfortunately, only one unconditional branch can be predicted taken. This is the BL instruction. So, when an unconditional branch is needed try to use BL whenever possible.

Also try to avoid placing usually-taken predicted branches in the delay slot of another branch because branch prediction is disabled for these cases.

4.4 Integer Load Instructions

Try not to have the instruction immediately following a Load to a GR be an instruction which uses this GR as an operand. This causes a one cycle interlock penalty and will also disable dual-issuing that instruction pair, possibly resulting in another penalty.

4.5 I-Cache Misses

During an I-Cache Miss, instruction execution proceeds as soon as the missing doubleword instruction returns from memory but before that entire line has been written to cache. The critical doubleword is returned first by the memory system, but the entire line must be copied into the instruction caches. The on-chip Icache allows simultaneous reads and writes on every cycle, but writes to the off-chip cache lock the cache for 2 cycles per write. Therefore, after an Icache miss, try to avoid loads, stores, system control instructions, memory control instructions, and other cache misses up until the end of the Icache line that missed.

The I-Caches are direct mapped caches which can lead to high miss rates for certain applications. The on-chip cache is 1 KBytes and is mapped directly with virtual address bits [22:31].

The off-chip Icache addresses are hashed in virtual mode when enabled with the proper diagnose bit. The hashing function is the following:

```
Tag Address == IADH[12:26]  (Less bits used for a smaller cache)
ivpn == Virtual Offset
isid == Space ID

Cache Address      Virtual Mode           Real Mode
-------------      ---------------------  ---------
IADH[26]           ivpn[26]               ivpn[26]
IADH[25]           ivpn[25]               ivpn[25]
IADH[24]           ivpn[24]               ivpn[24]
IADH[23]           ivpn[23]               ivpn[23]
IADH[22]           ivpn[22]               ivpn[22]
IADH[21]           ivpn[21]               ivpn[21]
IADH[20]           ivpn[20]               ivpn[20]
IADH[19]           ivpn[19] xor isid[29]  ivpn[19]
IADH[18]           ivpn[18] xor isid[30]  ivpn[18]
IADH[17]           ivpn[17] xor isid[31]  ivpn[17]
IADH[16]           ivpn[16] xor isid[28]  ivpn[16]
IADH[15]           ivpn[15] xor isid[27]  ivpn[15]
IADH[14]           ivpn[14] xor isid[26]  ivpn[14]
IADH[13]           ivpn[13] xor isid[25]  ivpn[13]
IADH[12]           ivpn[12] xor isid[24]  ivpn[12]
```

Software should try to achieve as much cache locality as possible. Knowledge of the above hashing function can help accomplish this goal.

4.6 D-Cache Misses

During a D-Cache Miss for both integer and Floating Point Load instructions, instruction execution proceeds until an instruction is encountered which requires the Load target as an operand (or another miss occurs). This policy for D-cache miss handling make it advantageous to separate the Load and its target register use by as many cycles as possible. Data cache access instructions which follow the instruction with the Load Miss will slow down execution but will not necessarily halt instruction execution.

If no copyout is required, there are free cache cycles during the memory latency, and since the memory system returns a doubleword every 3 cycles and cache writes take only 2 cycles, there exist "windows" during the copyin cycles that allow cache accesses instructions to steal cycles during the copyin period. Therefore, separating a load instruction from its use can be advantageous even when it is followed by more D-cache access instructions. However, since only one pending cache miss can exist at a given time try to avoid accessing the missing line (or any other cache line which misses) soon after the instruction with a Load miss.

For Store Misses (unlike for Load Misses) it is a good idea to store some other doublewords of the missing line during the latency period of the Miss because these stores will hit in cache.

For privilege 0 applications which involve block moves, block copies or block zeroing where the block size is 32 Bytes (the cache line size) or greater, the "don't fill" cache hint should be used. This should be helpful for speed critical OS routines, networking software, and graphics routines. Non-privileged users will not see any benefit of the "don't fill" hint.

The D-Cache is a direct mapped cache which can lead to high miss rates for certain applications. D-Cache addresses are hashed in virtual mode when enabled with the proper diagnose bit. The hashing function is the following:

```
Tag Address == DADH[12:26]  (Less bits used for a smaller cache)
dvpn == Virtual Offset
dsid == Space ID

Cache Address       Virtual Mode           Real Mode
-------------       ------------           ---------
DADH[26]            dvpn[26]               dvpn[26]
DADH[25]            dvpn[25]               dvpn[25]
DADH[24]            dvpn[24]               dvpn[24]
DADH[23]            dvpn[23]               dvpn[23]
DADH[22]            dvpn[22]               dvpn[22]
DADH[21]            dvpn[21]               dvpn[21]
DADH[20]            dvpn[20]               dvpn[20]
DADH[19]            dvpn[19] xor dsid[29]  dvpn[19]
DADH[18]            dvpn[18] xor dsid[30]  dvpn[18]
DADH[17]            dvpn[17] xor dsid[31]  dvpn[17]
DADH[16]            dvpn[16] xor dsid[28]  dvpn[16]
DADH[15]            dvpn[15] xor dsid[27]  dvpn[15]
DADH[14]            dvpn[14] xor dsid[26]  dvpn[14]
DADH[13]            dvpn[13] xor dsid[25]  dvpn[13]
DADH[12]            dvpn[12] xor dsid[24]  dvpn[12]
```

Software should try to achieve as much cache locality as possible. Knowledge of the above hashing function can help accomplish this goal.

4.7 TLB Misses

The PA7100LC CPU implements block TLB (also called BATC: Block Address Translation Cache) entries for the unified TLB. 8 block entries can each be programmed to map 128 - 16K pages (512K - 64MB segments). These are intended to be used to map large continuous virtual spaces for both the OS and for graphics applications.

The OS should use the new implementation specific instructions for fast inserts from the interruption parameter registers. In addition, there are many issues related to the effective use of the hardware TLB handler. Contact Joe Martinka for a copy of a study of the TLB performance of PCX-T.

The unified TLB is a 64 entry fully associative TLB. The replacement algorithm for these TLBs is described in the TLB chapter. To reduce the penalty for page crosses (ITLB lookaside buffer update) the BV, BE, or BLE instructions should be used whenever possible when branching to a new page.

The PA7100LC CPU also allows TLB entries to be locked in. Locked in entries will not be replaced until the entry is unlocked. This may be useful when real-time performance of a certain application is critical. Unfortunately, there is no architected PDC call at this time which allows this feature to be used by non privileged code or the OS.

The PA7100LC CPU has implemented backup "Shadow Registers" for GRs 1,8,9,16,17,24,25. for the purpose of reducing the TLB miss penalty. These registers allow the TLB handler to avoid the need to save and restore these registers. An RFIR instruction will automatically restore these GRs to values which existed at the time of the last trap (in addition to doing an RFI).

4.8 Memory Management Instructions

Most memory management instructions cause a substantial performance penalty. Cache flushes have CPU pipeline penalties ranging from 0-5 states. Also, multiple sequential flushes that hit dirty in the cache can quickly overload the memory buffers in the MIOC, stalling the CPU for even more states while the slower DRAM's are being updated. See the pipeline chapter for more description of cache flush penalties. Anything that can be done to reduce the amount of cache flushing is likely to help performance significantly.

PA7100LC does allows the execution of non-D-cache instructions to overlap the copy-back time for cache flush and cache purge instructions (as long as the memory buffers do not overflow).

4.9 Graphics Features

PA7100LC does not support any of the Venom functionality provided on PA7100 and earlier Viper-based systems (block move instructions are not supported). The on-chip Memory and I/O Controller supports efficient movement of data accross GSC with normal load/store instructions. PA7100LC does support floating point word and doubleword load/store operations to I/O space. This should increase the efficiency of large data movements to the frame buffer. The FP store instructions to I/O space incur a one cycle unconditional penalty cycle just like normal I/O space stores. The FP I/O load performs as well as the normal loads from I/O space. The floating point coprocessor does implement support for Graphics Clip test. See the FP chapter for more details on these instructions. Graphics might also benefit from the TLB-U bit, which specifies that pages should not be cached by the processor. See the Appendix for more details. Multimedia support has been added in the form of parallel 16-bit arithmetic instructions. See the Appendix for more details.

Other features mentioned in this section are especially useful for hand coded graphics routines.

4.10 Floating Point Scheduling

The penalties due to floating point interlocks are listed in the chapter on PA7100LC Floating Point.

4.11 Memory Moves via Floating Point Registers

The speed of memory to memory moves on large blocks of data can be improved substantially by using doubleword load/store through the floating point registers. This will utilize the full bandwidth of the 64 bit Data Cache interface.

4.12 Load/Load and Store/Store Bundles

Whenever possible, software should try to utilize bundling of integer loads and stores to memory. For this to occur, pairs of "ldw" or "stw"instructions must be used (none of the other load or store word variants), the same base register must be used for each instruction in the pair, and the effective addresses generated by the pair must be to different words within the same aligned doubleword. If these restrictions can be met, especially for long sequences of loads or stores, the integer memory bandwidth effectively doubles. While the cpu will bundle loads and store to U-bit or I/O space, this should be avoided as it does not result in improved bandwidth and may, in fact, result in worse performance than if the instructions had not been bundled together.

Chapter 5

CPU Pipeline

5.1 Introduction

The PA7100LC CPU pipeline is similar to the PA7100 and earlier CMOS PA-RISC pipelines. PA7100LC can issue two instructions per pipeline stage as explained in the previous sections. The major pipeline change from PA7100 is a shorter instruction fetch pipe stage due to the on-chip ICache.

```
-----------------------------------------
|   F   |   I   |   B   |   A   |   R   |
| 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 |
-------------------------------------------------
        |   F   |   I   |   B   |   A   |   R   |
        | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 |
        -------------------------------------------------
                |   F   |   I   |   B   |   A   |   R   |
                | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 |
                -----------------------------------------

A detailed pipeline diagram is included as an appendix.
```

CK1—F - CK1—I : Instruction Fetch

On-Chip ICache fetches start in CK1 of F and end in CK2 of F. Off-chip ICache prefetches start in CK2 of F-1 and end in CK2 of F. The steering logic operates on CK1 of I and picks appropriate instructions from the on-chip ICache, off-chip ICache, or prefetch buffer to place onto the 3 instruction buses, RIH/LIH/FRIH.

For branches which are predicted to be taken, the branch address calculation starts in CK1/I and completes by the end of CK2/I. This address is issued to I-Cache.

CK2—I - CK1—B : Decode and ALU/SMU Operation

The primary execution stage is between CK2 of I and CK1 of B. In this stage the instruction is decoded, operands are fetched, and the ALU and SMU (Shift Merge Unit) produce their results. The Data Cache address is generated by the ALU by the end of CK1 of B.

For branches which are not predicted or are predicted to be untaken, the branch address calculation starts in CK1/B and completes by the end of CK2/B.

CK2—B - CK2—A : Data Access

D-Cache Reads start in CK2 of B and end in CK2 of A. Load instructions and sub-word store instructions read the Data portion of the D-Cache during this state. For all LOAD and STORE instructions the Tag portion of the D-Cache is read during this state. The Tag portion of the D-Cache is addressed independently from the Data portion of the D-Cache so that tag reads can occur concurrently with a data write for the last store instruction.

Branch condition evaluation is completed by the end of CK2 of B. For branches which were predicted to be taken the sequential address is issued to I-Cache if the condition is true. For branches which were predicted to be untaken the branch address is issued to I-Cache if the condition is true.

D-Cache misses are discovered in this state. At the same time as a load or store miss is discovered the MIOC will begin handling that cache miss.

CK1—R - CK2—R : General Register Set

General Registers are set in CK2 of R. The Store buffer can be written to D-Cache starting on CK2—R and continuing for a total of two cycles. The store buffer is only written on CK2—R when one of the next instructions is a STORE instruction. Whenever the next STORE instruction is encountered, the store buffer will be written out to cache.

The PA7100LC CPU maintains a store buffer which is set on the cycle after CK2—A of each store (often CK2—R).

5.2 Pipeline Details

More detail on each of the pipeline stages will now be given. The reader may wish to skip this section on the first reading.

F-1 Stage (P stage)

CK2 :

- The instruction address is generated from the program counter, branch adder, or PC Queue. The address bits are sent unhashed to the on-chip ICache and are hashed and latched for a possible off-chip ICache access. The off-chip ICache will be accessed whenever the off-chip Cache is not being used for data accesses (load/store/copy-in/copy-out/etc).

F Stage

CK1 :

- The on-chip ICache indexes its RAM arrays.

CK2 :

- The on-chip ICache completes its access, returns a doubleword, and determines hit or miss. If an off-chip ICache prefetch cycle was available, the off-chip cache returns a doubleword of prefetch data.

I Stage

CK1 :

- Instructions are selected from the caches for the separate integer, load/store, and floating point instruction busses.
- Branch address calculation for predicted branches. begins.

CK2 :

- The Instructions are latched by all instruction decode blocks and instruction decode begins.
- General Registers operands (which are potentially bypassed from other registers) and immediate operands are valid by the end of CK2.
- The nullify indications (from the previous instruction), and the CPU Interlock indication are latched by the CPU control.
- Branch address calculation for branches which are predicted taken is completed.

B Stage

CK1 :

- The ALU and SMU generate their results.

- The Data address is calculated by the ALU. This address is hashed in virtual mode and set up in the D-Cache and (on-chip) UTLB address drivers.
- Branch address calculation for branches which are predicted untaken begins.
- Early Trap qualifiers are latched here. 'Early' traps are the traps numbered 2-7, 10-11, 23-25 and sometimes 1. The remaining traps are classified as 'Late' traps. The CPU takes advantage of the fact that Early traps are known before the other traps.

CK2 :

- The Data address is driven to the D-cache and (on-chip) UTLB memory arrays.
- The Data read becomes half-completed at this point.
- The Early traps are OR'ed into one signal.
- The branch and nullify conditions are known here. The branch condition is used to select between the Sequential and Branch addresses on this phase.
- Branch address calculation for branches which are predicted untaken is completed.
- Control Register 11 is set here for MTCTL instructions. This is done to avoid an Interlock for MTCTL x,11 followed by an instruction which uses CR11.
- The system mask is set here for SSM, RSM, and MTSM instructions. This is done to avoid an Interlock after these instructions. The PSW Carry/Borrow bits and the PSW V bit are also set here to avoid an interlock.
- MFCTL data from all CRs except CR17, CR19, and CR20 is valid on this phase early enough to be bypassed (without an interlock) to the next instruction.
- The Recovery counter is decremented for non-nullified instructions if the PSW R bit is set.

A Stage

CK1 :

- The D-RPN (Real Page Number) from the UTLB is driven to the D-Cache hit compare block.
- Qualifiers for all traps except DTLB Protection traps and Assist exception trap are valid here.

CK2 :

- Data from D-Cache is latched by a clock edge which is slightly delayed from CK1.
- The D-Cache Hit indication is latched by CPU control on this phase.
- All traps except the Assist Exception trap are OR'ed into one signal.
- MFCTL data for CR17, CR19, and CR20 is valid here. Data for MFSP, LDSID, LPA, PROBEs, and Move From Diagnose instructions is also valid here.
- A transaction to the MIOC will be issued in the event of a cache miss.

R Stage

CK1 :

- The Assist Exception trap signal is valid in the CPU.
- Store data is placed into the store buffer.

CK2 :

- General Registers are set here after all traps, misses, and nullifies have been resolved.

R+1 Stage

CK2 :

- The CPU may drive its store data to the Data Cache RAMs for store instructions. The completion of the store either overlaps with the execution of subsequent instructions (if there is no Data Cache conflict) or occurs while the pipeline is frozen.

5.3 Freeze Condition Sequencing

Certain conditions cause the pipeline to freeze. While the pipeline is frozen the condition which caused the exception is either serviced or ignored (eg. a trap will cause a miss to be ignored). Many freeze conditions can occur simultaneously and these are usually serviced sequentially in a fixed order. However, in many cases there is some overlap in the servicing of the different freeze conditions. The order that these are serviced is based on the 'priority' of the signal indicating the freeze condition. At the time that a freeze condition can be serviced, the highest priority freeze condition is the next condition to be serviced. In this section, the different types of freeze conditions are explained in order of their priority. Since these freeze conditions represent all components of the total CPI (cycles per instruction), a reasonable approximation of hang states is listed for each.

The pipeline freezes between CK1 and CK2. ie. The pipeline advances by states which start with CK2 and end with CK1. Unfortunately the pipe stage labels (FIBAR) are associated with CK1 — CK2 states instead of CK2 — CK1 states. Although there is no good reason for this, the pipe stage labels have been around too long to change them.

CPU Control guarantees that addresses and operands are valid on the freeze state immediately preceding a pipeline step. The I and D addresses and operands from the last pipeline step is repeated during the freeze states by default unless there is a need to change them. This allows the pipeline to be frozen easily at any time.

5.3.1 Reset

The MIOC looks at the RESETL pad to determine when to drive PRESETDL to reset the main control state machines. This occurs on cycling power and for a broadcast reset on the I/O bus.

Effects of Reset :

- All state machines are reset.
- All PSW bits except the M-Bit are reset. The M-Bit is set.
- The CPU begins fetching instructions from address F0000004. This address is in PDC I/O space. See the 'Start-Up Strategy' section of this ERS for more details.

```
Freeze Type                : Reset
Number of freeze states    : Many
```

5.3.2 Store Interlock

Single and Doubleword store instructions require two states of access time to D-Cache. This two cycle store access time causes the pipeline to freeze whenever the next instruction bundle and sometimes the bundle following that requires use of the Data Cache. The pipeline only freezes to finish a store when it sees an instruction which needs the D-Cache.

Byte and Halfword store instructions require three states of access time to D-Cache (one read cycle and two write cycles). The two write cycles causes the pipeline to freeze whenever the next instruction bundle and the bundle following that requires use of the Data Cache.

The exact number of freeze states is difficult to specify without taking the specific instruction sequence into account

```
Freeze Type                          : Store Interlock
```

```
Number of freeze states             :
Word or Doubleword Store :
    1 minus distance to next bundle with a LOAD or STORE (approx.)
Byte or Halfword Store   :
    2 minus distance to next bundle with a LOAD or STORE (approx.)

examples:
  Store - Load             ==> 1 or 2 cycle penalty
  Store - Flop - Load      ==> 1 or 2 cycle penalty
  Store - Add  - Load      ==> 1 or 2 cycle penalty
  Store - Add/Flop - Load  ==> 0 or 1 cycle penalty
  Store - Flop - Store     ==> 0 or 1 cycle penalty
  Store - Add  - Store     ==> 0 or 1 cycle penalty
  Store - Add/Flop - Store ==> 0 or 1 cycle penalty
```

5.3.3 Coprocessor Interlock

Coprocessor interlocks due to coprocessor instruction scheduling conflicts are summarized in floating point chapter of this ERS.

5.3.4 DTLB miss (hardware handler)

When a DTLB miss occurs the CPU will attempt to handle the TLB Miss in hardware. The CPU will read the PDIR entry (or a software "cache" of entries) and insert the entry into the TLB if it hits. If the hardware handler hits cache and the PDIR then the DTLB miss incurs only a 11 cycle penalty. More details on the hardware TLB handling mechanism are given in the TLB chapter.

```
Freeze Type      : DTLB lookaside buffer miss (Hardware Miss handler)
Number of States : 11 cycles plus possible cycles for one D-Cache Miss
```

5.3.5 Interruptions (traps)

An interruption occurs when any of the interruption conditions is true. The cause of interruptions 2-4, and 6-28 are architected. An interruption freezes the pipeline after CK1/R.

Interruption 1 (HPMC) occurs for all cache parity errors and double bit memory errors in addition to HPMC conditions on the GSC bus. Also, a transfer-of-control (TOC) will cause the CPU to vector to the HPMC address. See the 'Fault Tolerance' section of this ERS for a discussion of when HPMCs are recoverable. HPMC and TOC conditions vector to address F0000000 in PDC I/O space.

Interruption 5 (LPMC) occurs for corrected single bit memory (DRAM) errors and certain GSC errors. LPMC vectors directly to IVA + 5*32.

TOCs are completely decoupled from the PSW-M-Bit. ie. They are not qualified by the M-bit and do not set the M-bit when they occur. However, TOCs are masked by a HVERSION (implementation specific) bit. This bit is SET when 1) a TOC is taken, 2) Broadcast Reset occurs, or 3) when a special Diagnose instruction is executed to set this bit. This bit is RESET when a special Diagnose instruction is executed to reset this bit. PDC must reset this bit after it clears the MIOC TOC bit but before it enters the OS_TOC handler. After broadcast reset, PDC should leave this bit set until the hard or soft boot has completed (at the software IPL interface). It is an architectural requirement that TOCs should not interrupt Hard or Soft boot. PDC may want to set this bit if discovers a situation where TOCs should be masked. One such situation may be in an HPMC handler when it discovers an invalid checksum for the IVA.

Only the Group 3 traps are inhibited if the instruction is nullified.

The Group 4 traps are implemented as "Taken-Before" traps rather than "Taken-After" traps. ie. Transfer traps occur after the privilege change and the Taken Branch trap occurs on the delay slot instruction. However, the trapped instruction does not execute (backout is required) and these traps are the highest priority interruptions (even higher than HPMC). If HPMC was higher priority than the Group 4 traps, software could not recover from an HPMC because it would not know whether a group 4 interruption was lost (ie. signaled but never taken) due to taking the HPMC.

The FIC instruction uses the UTLB for translation and can cause a Non-Access DTLB miss trap.

PA7100LC maintains a set of back-up GRs or "shadow" registers for the purpose of reducing the DTLB and ITLB miss penalty. At the time of a trap (any trap) the values of GRs 1,8,9,16,17,24,25 are copied into their corresponding back-up registers. A special instruction called RFIR can be used to recover these values after trap servicing has been completed. This feature allows a trap handler to use GRs 1,8,9,16,17,24,25 without the overhead of saving and restoring them.

All interruptions except assist exception trap require three pipeline stall cycles. Assist exception trap on FP load, store, or FTEST (not FLOPs) requires five pipeline stall cycles. Assist exception trap is special only because it is valid at the CPU at a later time than the other traps. In the freeze state the trap conditions are re-calculated.

Effects of Interruption :

- Wait for pending instruction and data cache misses to finish.

- Push PSW to IPSW.
- Clear PSW. Set M-bit if HPMC, Reset M-bit if not HPMC.
- Backup 'Early' SAR and Recovery Counter (since these are set early). (The PSW bits which are set in CK2/B never got set into the 'real' PSW. These bits are also cleared here.)
- Prevent the execution of the trapped instruction. This entails backing out of a store, inhibiting register sets etc..
- Flush the pipeline. ie. The next two instructions which have entered the pipeline must not execute.
- Start fetching instructions from (IVA + 32*(trap_class)) if not HPMC, else fetch from F0000000.
- Set GR "shadow registers" from GRs1,8,9,16,17,24,25.
- Ignore the following freeze conditions: CPU Interlock, Inserts, TLB Purges, Diagnose, FIC instructions, Flushes, PDC, Load and Clear instructions, D-Cache miss, I-Cache miss, Branch and RFI.

```
Freeze Type              : Interruption
Number of freeze states  :
   5 cycles for Assist Exception Trap on Floating Point Loads and Stores
   3 cycles for all other traps
Note: There is an additional two state pipeline penalty because
the next two instructions in the pipe are effectively nullified.
```

5.3.6 Data Cache Miss / Cache Contention

D-Cache Miss is signalled for loads/stores that miss the off-chip cache, for flushes/purges that hit the off-chip cache, and for accesses to I/O or U-bit space. D-Cache Miss is only serviced if the instruction did not trap and is not nullified. However, there is a one state freeze penalty when there is a Data-Cache Miss on an instruction which takes an interruption and there is a Load Use Interlock on the next instruction. D-Cache Miss freezes the pipeline after CK1/R.

Cache hints specified by the PA1.1 architecture can be used for stores and load-and-clear instructions to minimize cache miss penalties. Hinted stores executed at privilege level 0 will not cause a cache move-in if they are line-aligned. Hinted load-and-clear instructions will operate in cache if the line is present in cache.

The CPU implements "stall-on-use" for both load and store D-Cache misses and for flushes that hit. The minimum penalty for STORE D-Cache misses is one cycle. The minimum penalty for LOAD D-Cache misses is two cycles. The minimum penalty for FLUSH/PURGE D-Cache hits is one cycle. However, to achieve this effective one cycle miss penalty a number of constraints must be met. These constraints will be described, as well as a description of how the penalty can vary with different parameters. Although it is difficult to meet the requirements for an effective one cycle miss penalty there are many ways to reduce the effective miss penalty. Stall-on-use can be disabled by a diagnose bit, although this should be done only to debug prototype hardware because it will make every data cache miss stall the pipeline until the entire line has been copied from memory into cache.

If a load instruction misses and the target GR is not referenced immediately, the pipeline can unfreeze and continue stepping until the target GR is referenced as a source register, or until there is a conflict on the off-chip cache. Since a data cache miss might cause a copy-out of dirty data and definitely causes a copy-in of 4 doublewords, there are many cycles that the off-chip cache is locked (busy) as a result of the miss. The cache controller is optimized to use all available cache cycles (ie. there is an idle cache cycle between copy-in doublewords), but code can probably reduce freeze states by having as few load/store/flush instructions following a DCache miss as possible.

Since the MIOC returns data in a critical-word-first fashion, if the pipeline is frozen because it is waiting for data, it can un-freeze as soon as the critical word arrives from the MIOC. But, the pipeline may freeze again if cache is busy copying-in the rest of the line and a load/store/flush is encountered in the instruction stream.

It is essential to avoid, if possible, references to the same line as the load miss until the copyin completes. Failure to do so will cause the CPU to freeze until the copyin is finished.

To reduce the effective D-Miss penalty for a Store it is necessary to maximize the distance between the Store instruction and the next load (or BYTE or HALFWORD store) instruction to the same cache line. After the copy-out window is finished, Stores to line which originally missed will now hit. Stores to this line will be executed during the latency period of the miss (hardware knows to throw away each word from main memory which is being stored to in this window). This was termed “scoreboarding” in PA7100. Store misses still busy the cache while copying-in doublewords from MIOC, so the pipeline may freeze if a load/store/flush is encountered during the copy-in.

To reduce the penalties on data cache flushes and purges that hit the D-cache and instruction cache flushes that hit the I-cache, avoid referencing the data cache within five instruction bundles after a flush or within three instruction bundles after a purge. This will help avoid freezing the cpu during the copy-out/invalidate.

Because the CPU can copy-out data faster than the MIOC can write the data to memory, it is possible to overflow the MIOC flush-buffers by having many consecutive flush instructions or U-bit/IO stores. When this happens, the copy-out rate is determined by the time needed to write the DRAM’s.

Typical latencies are as follows:

- If dirty data causes a copy-out, the cache is busy for 4 cycles
- Memory latency to critical doubleword in the CPU is 7 cycles (overlapped with the possible 4 cycle copy-out)
- Remaining doublewords return every 3rd cycle
- The cache is busy for 2 cycles for each doubleword returned from the MIOC

5.3.7 Incorrect Branch Prediction / RFI

Our pipeline requires one extra state for the fetching of an incorrectly predicted branch target. (An additional cycle is required for page crosses and RFI. These penalty cycles are summarized in a later section.)

The following branches cannot be predicted because the address or privilege calculation is dependent upon a general register: GATE, BLR, BV, BE, BLE.

The following branches are always predicted untaken: Forward PC Relative branches. These include COMBT, COMBF, COMIBT, COMIBF, MOVIB, MOVB, BB, BVB, ADDBT, ADDBF, ADDIBT, ADDIBF.

Backward PC Relative branches are usually predicted taken. They are predicted untaken only in the following cases: 1) They are in the delay slot of another branch or I-Cache or Memory Reference Instruction (Opcode = 000x0x). 2) The branch was the target of an RFI.

Note that the nullify indication has no effect on determining whether a branch is predicted or not. eg. A nullified backward PC relative branch is still predicted as taken.

The RFI instruction causes this freeze condition to be true twice. One to fetch the front of the PCO/PCS queues and one to fetch the rear of the PCO/PCS queues. The penalty is two cycles for each RFI target (the extra penalty is required for an ITLB lookaside update).

Incorrect branch prediction freezes the pipeline after CK1/A. RFI freezes the pipeline after CK1/A and after CK1/R.

Effects of Incorrect Branch Prediction / RFI :

- Continue I-fetching (for one state). The address of this instruction is the target address for a taken, unpredicted branch, or PCOffset +8 for an untaken, predicted branch.
- Pop the PSW on the first RFI freeze state (if the RFI does not trap). The instruction

following the RFI (which has entered the pipe) is not executed and never traps.

- Set SR[0] if BLE. (This avoids an Interlock for BLE).
- Update the ITLB lookaside buffer for each RFI target.

```
Freeze Type               : Incorrect Branch Prediction / RFI
Number of states          :
Wrongly Predicted Branch  : 1
RFI                       : 1 (the RFI penalty is taken twice)
Note: There is an additional 1 state penalty for RFI because the
delay slot instruction is effectively 'nullified'.
There is also an additional cycle for RFI associated with an ITLB
lookaside update.
```

5.3.8 TLB Insert/Purge, Diagnose

The following instructions freeze the pipeline after CK1/A: IITLBA, IITLBP, IDTLBA, IDTLBP, PITLB, PITLBE, PDTLB, PDTLBE, and all diagnose instructions except MTDIAG/MFDIAG. These operations are not performed if the instruction traps or is nullified.

This freeze condition is also entered for the implementation specific instructions IITLBPF/IDTLBPF (Fast TLB Insert Protection). See the Appendix. Diagnose instructions need to freeze the pipeline because these are multi-state operations.

```
Freeze Type               : TLB Inserts/Purges and Diagnose
Number of states          :
IxTLBA,IxTLBP,PxTLB with PSW-C=0       : 2
                    with PSW-C=1       : 3
PxTLBE                                 : 2
IITLBPF                                : 3
IDTLBPF                                : 1
Diagnose                               : 3
```

5.3.9 CPU Interlock (load-use, other)

The CPU has two types of interlocks. The first is the Load-Use Interlock. This occurs when a GR operand of an instruction directly after a 'Load' is the same general register the 'Load' is setting. 'Load' in the previous sentence actually includes the following instructions: All loads, MFCTL (CR17, CR19, or CR20), MFSP, LDSID, LPA, PROBE, and Diagnose (MFCPU_T, RDD, RDTLB, RDT, RI2D, RI2T). In addition, the Load Use interlock is actually signalled any time either of the five bit fields 6:10 or 11:15 matches the 'Load' target number even if these fields indicate immediate data.

The second type of CPU interlock is due to MTCTL (except to CR11), MTSP, and Diagnose (MTCPU) instructions. These instructions always cause an interlock regardless of the next instruction. These interlocks simplify the control.

The CPU Interlock occurs a maximum of once per instruction.

Effects of CPU Interlock :

- Latch 'Load' data, set GRs for previous instruction, advance target number and result data pipelines, advance offset and space pipelines.
- Set SRs, CRs, and diagnose registers if MTSP, MTCTL or MTCPU respectfully.

```
Freeze Type               : CPU Interlock
Number of states          :
 for LPA                  : 2
 all other CPU Interlocks : 1
```

5.3.10 Refetch

When a branch instruction is executed, the CPU tries to get the branch address to the TLB so that if a page-cross occurs, the ITLB lookaside buffer can be updated in a single state. If this is successful, the penalties are as listed in the next subsection. If the TLB (or data address bus) is busy and the branch has a page-cross, the CPU must pay an extra 2-cycle penalty to "refetch" the instruction translation from the DTLB into the LAB. This might occur if a page-cross-branch is bundled with a preceeding load/store or if the branch occurred while a pending data cache miss was being handled. This penalty is in additional to the normal LAB update penalty (if any).

```
Freeze Type               : Refetch
Number of states          : 2
```

5.3.11 ITLB Lookaside Buffer Update

The ITLB lookaside buffer needs to update under the following circumstances: 1) Page crosses due to branches in virtual mode, 2) Space changes in virtual mode, 3) Privilege changes in virtual mode, 4) RFI instructions, 5) SSM, RSM, MTSM instructions (in case of a P-Bit change), 6) ITLB Inserts or Purges in virtual mode, 7) MTCTL to CR8, CR9, CR12, CR13 and, 8) Page crosses not due to branches (sequential execution). Some of these updates are overlapped with other pipeline stalls or pipeline steps. All the pipeline stalls due to ITLB lookaside buffer updates which are not overlapped with other pipeline stalls are summarized in the following table.

```
Page Crosses, Space crosses, Privilege Crosses :
  Due to BV, BE, BLE                  : 0 cycle penalty
  Due to all other taken branches     : 1 cycle penalty
    (Regardless of whether it is predicted taken)
RFI (regardless of page cross etc.) : 1 cycle for each target (==>2)
SSM,MTSM,RSM                        : 1
TLB Insert or Purge in virtual mode : 1 cycle
MTCTL to PID                        : 1 cycle
Page Cross not due to taken branch  : 1 cycle
```

5.3.12 ITLB miss (hardware handler)

When an entry does not exist in either the lookaside buffer or the unified TLB, the CPU will attempt to handle the TLB-Miss in hardware. The CPU will read the PDIR entry (or a software "cache" of entries) and insert the entry into the TLB if it hits. If the hardware handler hits cache and the PDIR then the I-TLB miss incurs only a 11 cycle penalty. This 11 cycle penalty includes updating the lookaside buffer. More details on the hardware TLB handling mechanism are given in the TLB chapter.

```
Freeze Type      : ITLB lookaside buffer miss (Hardware Miss handler)
Number of States : 11 cycles plus possible cycles for one D-Cache Miss
```

5.3.13 Instruction Cache Miss

When an instruction fetch does not hit the on-chip instruction buffer, the CPU will freeze in order to check the off-chip ICache. An on-chip-miss, off-chip-hit will normally take 2 freeze cycles, but this freeze will be for only 1-state if the prefetch (or preissue) mechanisms were already in the middle of the fetch when the miss occurred. If the instruction is neither in the on-chip nor off-chip Icaches, then a fetch from memory must occur. An instruction fetch from memory looks very similar to a data cache miss as explained above. The MIOC will return an ICache line starting with the critical word first. The CPU will unfreeze as soon as the critical word arrives, and will stream the instructions directly from the MIOC into the instruction decoders as they are sent to the on-chip and off-chip Icaches. The MIOC also implements memory to off-chip ICache prefetching in order to speed instruction fetches from memory.

```
Freeze Type      : ICache miss
```

```
Number of States :
  Hit to a prefetch or preissue just begun : 1 state
  "Normal" off-chip ICache hit             : 2 states
  Hit to MIOC I-Prefetch Buffer            : 6 states
  "Normal" memory access                   : 9 states
Additional states may be incurred on successive instructions since a
doubleword is returned every 3 cycles.
```

5.4 Reading Reserved and Nonexistent bits

```
SAR[0:26]:       NonExistent Bits; Read always returns zeroes.
CCR[0:23]:       Reserved bits; Read always returns zeroes.
IPSW[0:4,24:26] Reserved bits; Read always returns zeroes.
IVA[22:31]       Read Returns what is written, but is not used for trap vector
PID1-4[0:15]:    Reserved bits; Read always returns zeroes.
SR1-7[0:15] :    NonExistent Bits; Reads return zeroes
ISR[0:15] :      Reads return zeroes
Reserved Registers : Reads return undefined values.
```

Writes to reserved bits or registers cause no harmful side-effects although software should not rely on this.

Chapter 6

TLB

The PA7100LC CPU is equipped with a unified instruction/data tlb. The TLB is organized as 64 fully associative page entries. Each page entry maps 4k bytes of virtual space. In addition to the 64 page entries, the tlb contains 8 Block entries. Each block entry is capable of mapping a contiguous virtual address space ranging in size from 128 pages (smallest) to 16K pages (largest). Some size and alignment restrictions apply to block entries.

In addition to the Unified TLB, PA7100LC contains a one-entry Instruction Lookaside buffer (LAB). The LAB generally contains a translation for the Instruction page that was most recently accessed. If a hit is encountered in the LAB the UTLB is free to perform a Data translation without incurring a penalty.

6.1 TLB organization

The TLB produces real addresses from virtual addresses whenever a memory or IO transaction or instruction fetch occurs in virtual mode. TLB translations are accessed through a 36 bit virtual page number (VPN) computed as follows:

VPN[0:35] = cat(SID[0:15],Page_offset[0:19]);

The translation process produces either a 20 bit real page number (RPN) or any of the TLB traps specified by the architecture.

To facilitate trap generation, the four architected Protection ID (PID) registers (CRs 8,9,12 and 13) are visible to the TLBs.

The TLB contains a 16 bit Diagnose Control register which assists a variety of test and initialization functions.

6.1.1 Page entries

Each page entry stores and compares a 36 bit VPN which identifies a single 4k page. A translation is stored for each entry which contains the 20 bit RPN, the architected PID (15 bits), Access rights (7 bits) and E-flag (valid bit). Translations for data accesses additionally contain the architected T, D, B, and U (uncached page) flags.

Each page entry can be individually locked out (ie. its hit comparator disabled) or locked in (ie. excluded from consideration for replacement).

Insertion of a page entry is accomplished through the use of the architected IITLBA,IDTLBA instructions. The TLB contains hardware which automatically selects an appropriate entry to be the target for these instructions. The protection fields (PID, AR and flags) for the target entry are cleared (zeroed) by these instructions. Insertion of protection is accomplished through the use of the architected IITLBP or IDTLBP instructions. Purging of translations is accomplished through the use of the architected PITLB or PDTLB instructions. Execution of a PITLBE or PDTLBE instruction serves to invalidate ALL page entries in a single instruction. PA7100LC also has non-architected, faster-executing insert address/protection instructions. See the last section of this chapter.

6.1.2 Block entries

Translations via block entries differ from those via the normal page entries in two respects:

1. Each of these entries maps a minimum of 128 pages, thus the low seven bits of the VPN are excluded from hit comparison (ie. not stored). The maximum space mapped by these entries is 16K pages, thus seven additional bits of the VPN are optionally excluded from hit comparison.

2. When a block entry hit is encountered, the 20 bit RPN is assembled by bypassing low order bits of the VPN into the RPN corresponding to the bit positions that were excluded from the VPN compare.

Thus for the smallest space mapped by a block entry (128pages = 512kbytes):

```
if virtual_address[0:28] = VPN[0:28]          {hit on block entry}
then RPN[0:19] = cat(TRANS[0:12],VPN[29:35])  {assemble effective RPN}
```

For the largest space mapped by a block entry (16k pages = 64mbytes):

```
if virtual_address[0:21] = VPN[0:21]          {hit on block entry}
then RPN[0:19] = cat(TRANS[0:5],VPN[22:35])   {assemble effective RPN}
```

Other legal space sizes include 256 pages, 512 pages, 1k pages, 2k pages, 4k pages, 8k pages and 16k pages. Insertion of addresses and protection and the specification of block sizes is accomplished through pdc calls. The mechanisms provided by the CPU for use by this code are discussed under the subtitle Diagnose Functionality. Normally, block entries are unaffected by the architected Insert and Purge instructions.

6.2 TLB Page Replacement

The target entries for IITLBA and IDTLBA instructions are selected by a hardwired algorithm resident in the TLB. When an IITLBA or IDTLBA is performed, one of the 64 page entries, numbered from 0 to 63, is selected as the target for the insertion. The highest priority is given to any entry whose VPN field matches the VPN being inserted, in order to avoid multiple mappings of the same page. If no entry with a matching VPN is found, the lowest numbered entry which is invalid (i.e., its protection field's E-bit is 0) is selected. If both of these attempts fail, then the TLB must select a currently valid entry to replace. A "not-recently-used" selection is made to determine the target in this case (ie. the lowest number entry that has the "used bit" clear is replaced, or if all entries are "used", then entry 0 is replaced and all the "used" bits are cleared).

It is possible to partially shield an entry from being the target of a replacement. Each entry contains, in addition to the architected contents, a one-bit field known as the *lock-in* bit. If this bit in an entry is set, the hardware algorithm will only select that entry for replacement if its VPN matches the VPN being inserted. The entry is then said to be *locked in*. It is also possible, for initialization and test purposes, to PARTIALLY circumvent the hardware algorithm and specify directly which entry is to be replaced during the next insertion, using a six bit pointer to indicate the target entry. (A VPN match in a page entry will ALWAYS cause an insert or purge to operate on that entry in addition to any entry pointed to by the diagnose register). This function is known as *diagnostic insertion*. When this method of selecting the targets for IITLBA and IDTLBA is in use, even locked-in entries may be selected for replacement. Both locking entries in and explicitly designating the replacement target are done through the diagnose functionality present in the TLB.

6.3 Diagnose Functionality

6.3.1 Diagnose Control register

A Diagnose Control register is defined for the TLB. Thus is a 16-bit register which is loaded via Move_to_Diagnose instructions (see Diagnose Chapter for encoding). This diagnose register has partial-read capability as shown below.

TLB: Diagnose register 8 (DR8[16:31])

bit		feature description	feature disabled value
16	r/w	Force VPN Mismatch	0
17	r/w	Inhibit Replacement	0
18	r/w	Page Entry Force-Insert Disable	1
19-24	w	Page Replacement Pointer	don't care
25	r/w	Page Entry LRU-Insert Disable	0
26	w	Accelerated Failure mode	0
27	w	Not Used	don't care
28-30	w	Block Entry Pointer	don't care
31	r/w	Block Entry Force-Insert Enable	0

Explanation:

DR8[16:17]	effect
00	subsequently inserted addresses will be neither locked in nor locked out.
01	Lock-in: Subsequently inserted addresses are shielded from replacement unless they become the targets of a diagnostic insertion or they match the VPN of some future translation to be inserted.
11	Lock-out: Subsequently inserted addresses will be forced to mismatch on every translation attempt.
10	Undefined operation will result.

DR8[18] When clear, subsequent insert instructions (IxTLBx) will insert to the page-entry pointed to by bits 19:24 (AND to any page-entry that has a SID-VPN match). Set to 1 for normal operation.

DR8[19:24] 6-bit unsigned integer (bit 19 is msb). It indexes the Page Entry targeted for inserts when DR8[18]=0.

DR8[25] While set, the normal LRU replacement algorithm that selects page entries for replacement on IxTLBx instructions will be disabled. Set this bit when you want to target a specific page-entry or block-entry for insertion. Set to 0 for normal operation.

DR8[26] Provided as a test feature for wafer screen.

```
                CODE MUST GUARANTEE THIS BIT IS CLEARED (ZERO)
                AT ALL TIMES.

DR8[28:30]      3-bit unsigned integer (bit 28 is msb).  It indexes the
                Block Entry targeted for inserts when DR8[31]=1.

DR8[31]         When set, subsequent insert instructions (IxTLBx) will
                insert to the block-entry pointed to by bits 28:30 (AND to
                any page-entry that has a SID-VPN match). Set
                to zero for normal operation.
```

6.3.2 Entry Lock-in

Each page entry contains a lock-in bit. This bit is loaded from the Inhibit Replacement control bit in the control diagnose register (DR8[17]) whenever an address insertion to that entry occurs. When this bit in an entry is set, the entry will only be selected for replacement if its VPN matches the VPN being inserted or if the entry is the target of a Diagnostic Insertion.

6.3.3 Entry Lock-out

Each page and block entry contains a lock-out bit. This bit is loaded from the Force VPN Mismatch control bit in the control diagnose register (DR8[16]) whenever an insert address to that entry occurs. When set, the effect of the lock-out bit is to force a VPN mismatch on that entry. At system power-on, all 64 page entries and all 8 block entries are locked out. Initialization code is required to clear all of the lock out bits and load the VPN in each entry with a unique address.

It is important to make sure that for all locked out entries the lock-in bit is also set. Otherwise, a locked out entry might be selected as the target of an insertion. As long as both these bits are set in an entry, it will never be a target for replacement and it will never be used during a translation. It is always possible to clear the lock bits for an entry using the Diagnostic Insertion mechanism.

6.3.4 Diagnostic Insertion

Diagnostic Insertion provides a mechanism to explicitly designate, with a six-bit pointer indicating a number between 0 and 63, which page entry will receive the next insertion. To use this mechanism, the Replacement Pointer field (DR8[19:24]) in the control register should be set to the integer index of the entry to be inserted to. In addition, the Page Entry Force-Insert Disable bit (DR8[18]) should be set to '0', the Page Entry LRU-Insert Disable bit (DR8[25]) should be set to '1', and the Block Entry Force Insert Enable bit (DR8[31]) should be set to '0'. The next address insertion will occur to the indexed page entry (AND also to any page entry that has a SID.VPN match to the address being inserted).

6.3.5 Insertion of Block TLB entries

Block TLB insertion is accomplished by pdc using a combination of diagnose write functionality and architected Insert instructions. The block entries do not respond to the architected Insert Address and Protection instructions, nor do they respond to Purge, Purge Entry or Broadcast Purge, unless the Diagnose Control register has been previously loaded with a certain set of values.

The block entry insertion process is explained by the following example:

To insert to block entry N (N=0..7)

- Ensure NO page entry has a SID.VPN match (even for INVALID entries).
- Set the Page Entry Force-Insert Disable bit to 1 (DR8[18])
- Set the Page Entry LRU-Insert Disable bit to 1 (DR8[25])
- Set the Block Entry Force-Insert Enable bit to 1 (DR8[31])
- Set the Block Entry Select field in the Diagnose register (DR8[28:30]) so that it points to the Block entry slot to be inserted (N)
- Assemble a virtual address (= cat(space,offset[0:19])) for any page within the block to be mapped, then
- modify that Virtual address by overwriting the seven least significant bits of the offset (offset[13:19]) with the block size specifier (see table this section, below)
- Assemble the Physical page No. (in GR[r][7:26])
- modify that Physical page No. by overwriting the seven least significant bits of the offset (offset[13:19]) with the block size specifier (see table this section, below)
- Execute an Insert Address instruction
- Execute an Insert Protection instruction as with a page entry
- Reset the Diagnose register to its original (disabled) value.

```
                 vpn bits        rpn bits    block TLB size
  mask[6:12]     compared        returned    (in 4KB pages)
---------------------------------------------------------------
   0000000         0-5             0-5        16,384 pages
   1000000         0-6             0-6         8,192 pages
   1100000         0-7             0-7         4,096 pages
   1110000         0-8             0-8         2,048 pages
   1111000         0-9             0-9         1,024 pages
   1111100        0-10            0-10           512 pages
   1111110        0-11            0-11           256 pages
   1111111        0-12            0-12           128 pages

All other values of mask[6:12] are illegal.

Example:
    Block TLB entry:
          space[16:31] = "0001111100000110"
          vpn[0:12] =    "0111011001011"
          rpn[0:12] =    "0001001000110"
          mask[6:12] =   "1100000"

Translations:

      space[16:31]  offset[0:31] --->  real addr[0:31] (or TLB miss)
     ----------------------------------------------------------------
         0x1f06       0x76584321         0x12584321
         0x1f06       0x76ffffff         0x12ffffff
         0x1f06       0x77777777         TLB miss (bit 7 of vpn)
         0x1f16       0x76000000         TLB miss (bit 27 of space)


     The diagnose register for an insert to block entry #4 should be:
           0x00002049  Lockout = 0
                       Lockin = 0
                       Page Entry Force-Insert Disable = 1
                       Page Entry LRU-Insert Disable = 1
                       Accelerated Failure Mode = 0
                       Block Entry Pointer = 4
                       Block Entry Force Insert Enable = 1

    A PxTLBE instruction WON'T invalidate the block TLB entries; just
    the page entries.
```

A block entry may be purged in a similar manner: When DR8[18] = DR8[25] = DR8[31] = 1, the block entry selected by DR8[28:30] becomes the target for subsequent Purge instructions. Note that insertion of protection in block entries does not function in the same manner as in page entries, which select the IDTLBP or IITLBP target through the VPN referenced in the instruction. Block entries must have both protection and address inserted using their block entry select bit (DR [28:30]). Block entries do not respond to Purge entry instructions. Note also that page entries (even invalid ones) that have a matching SID.VPN will always respond to insert or purge instructions, regardless of the setting of the diagnose register.

Probe and LPA instructions are essentially TLB translation accesses, and thus function the same for block entries as for page entries.

6.4 Instruction Lookaside Buffer

In addition to the 64 page entries and 8 block entries of the UTLB, there is a one-entry Instruction lookaside buffer (LAB). The LAB is architecturally invisible but affects performance due to penalties incurred to update the buffer. These penalties are enumerated in the Performance section of this ERS. The following description of the LAB is included as a qualitative description of the implementation:

The LAB contains:

1. VPN 0:19
2. RPN 0:19
3. gateway privilege 30:31
4. tlb miss trap indication (1 bit)
5. tlb prot trap indication (1 bit)

In real mode operation the LAB is unused. However, hangs will be taken to update the LAB (to save terms or make things simpler).

In normal virtual fetching the RPN is used to check for imisses, the VPN is used to check for branches to new pages, the gateway privilege is used by the GATE instruction, and the trap bits determine if ITLB traps are taken.

The following cases describe when the LAB is updated:

`Case #1: Sequential page crossing`

A page crossing is detected by looking at bits 20:29 of the offset. If the bits are all 1's then a hang will occur (called seqhang). The priority of seqhang is similar to ilock but it will execute in parallel with ilock, refetch and math wait. It is a one state hang that is necessary to update the LAB and check for any ITLB traps. The hang occurs on the Fstate of the instruction that has 20:29 = that crosses the page). The seqhang is qualified by the PSW C-bit (only take the hang if C-bit is set).

`Case #2: BE, BLE, BV`

During the Bstate of these instructions the LAB is updated. At that point the IHANG that would normally be taken for these instructions is incurred.

`Case #3: Branches (predicted right or predicted wrong)`

These incur no penalty unless they are taken and the target is on a different page than current page (determined by checking the LAB's VPN). If taken to a different page and the PSW-C bit is set, there is one or two additional state penalty to update the LAB and check traps. The hang for this appears to be the highest priority hang, but it can execute in parallel with all hangs besides seqhang. A summary of the LAB-update hangs is given in the chapter on the pipeline.

`Case #4: RFI`

An RFI hangs for two states, takes a step, then hangs for two more. The LAB updates and traps are checked for both RFI targets.

`Case #5: P-bit changes`

Whenever the P-bit changes the LAB has to be invalidated and restored from the UTLB. As a result all SSM, RSM, and MTSM instructions will force a seqhang. They will not take default hangs. Seqhang will be set regardless of the PSW C-bit.

`Case #6: Control register changes`

Whenever a MTCTL occurs to one of the PID registers the CPU will take an ILOCK hang. During the ILOCK hang (if it is caused by a MTCTL) a seqhang will be forced (regardless of the PSW C-bit).

`Case #7: TLB instructions (and TLB broadcasts)`

ITLB instructions will force a seqhang on the last state of their hang if the PSW C-bit is set ('1').

Any time the LAB is updated, ITLB traps (miss fault and protection) are checked. If either of these traps occurs then the appropriate bit is set in the LAB and any instruction that accesses the LAB will take the proper trap.

The architected itlb instructions are identical to the corresponding dtlb instructions:

```
IITLBA <--> IDTLBA
IITLBP <--> IDTLBP
PITLB  <--> PDTLB
PITLBE <--> PDTLBE
```

6.5 Initialization and Test

At chip power on, all TLB entries are locked out. Code is responsible for clearing the lock out bit before an entry can be used. This must be done by inserting a unique address, using the diagnostic insertion mechanism, into each location whose lock out bit is to be cleared. Once the entries have had their lock-out bits cleared and their VPN fields set with unique addresses, the subsequent uniqueness of addresses in each entry is guaranteed by the hardwired replacement algorithm. Code must also initialize (clear) the lock in bits to enable replacement. All 64 E-bits should also be cleared (via PxTLBE).

It is possible for initialization code to test the TLB through use of the Probe instructions. Entries can be tested noninteractively by enabling them (clearing the lock-out bit) one entry at a time. The diagnose replacement pointer may be handy for this purpose. Individual failed entries can be mapped out of the TLB by setting both the lock out and the lock in bits. Once initialization is complete, the control diagnose register should be loaded with a value such that all the diagnose features are disabled.

6.6 TLB penalties

The processor must hang the pipeline to handle TLB inserts and purges. The various penalties are summarized below in number of states:

```
IxTLBA,IxTLBP,PxTLB with PSW-C=0   2
                    with PSW-C=1   3
PxTLBE                             2
```

6.7 Hardware TLB Miss Handler

6.7.1 Introduction

The hardware TLB miss handler on PA7100LC is designed to reduce the TLB miss penalty while being low-cost to implement (in complexity and area). The hw TLB handler is invoked on I-side and D-side translations that miss the on-chip TLB. The handler computes the address of a PDIR entry based on the missing space and vpn. It then accesses the PDIR entry. The PDIR entry is checked for three things:

1. Valid tag,
2. Matching tag, and
3. Reference bit = 1.

If the checks pass, the RPN and protection of the PDIR entry are inserted into the on-chip TLB and the original access is re-translated.

If any of the checks fail, the handler will not insert the entry and the instruction will trap to software. For DTLB misses only, a pointer to a PDIR entry is passed to the software TLB handler so that it doesn't have to recompute the PDIR address. The pointer is passed in CR28. CR28 will either have:

1. the address of the current PDIR entry, or
2. the address of the next PDIR entry.

Whether the current or next PDIR entry is passed will depend on the configuration of the diagnose register and which of the checks failed.

The hw TLB handler looks into a table of PDIR entries that will be referred to as the hardware-visible table. For an inverted page table, the hw-visible table contains the first level entries. For a forward mapped page table, the hw-visible table contains a "cache" of entries that are distinct from the actual page tables (this mode is targeted for OSF). Note that these tables must be stored in Little-Endian mode if the default Endian bit in diagnose register #0 is set.

There are bits in diagnose register #25 that disable the hw TLB handlers. If they are not enabled, PA7100LC will take TLB miss traps without first activating the hardware handler.

6.7.2 PDIR Address Generation

The starting address of the hw-visible table is stored in DR24. The table must start on a 4KB page boundary (i.e. bits 0-19 of DR24 are significant). To generate the address of the PDIR entry, the missing space and vpn are hashed together. Upper bits are masked off depending on the size of the hw-visible table to give an offset into the table. This offset is then merged with DR24 to get a 32-bit real address that is used to access the cache (note: cache hashing is turned off on real addresses). Note that the PDIR is accessed in real-mode and the default Endian bit in diagnose register #0 is used to control big/little Endian mode.

The base address is MERGED with the offset (not added to the offset). As a result software must align the hw-visible table to a multiple of its size.

```
Pseudocode for address generation:
---------------------------------
     spc  : Missing space[16:31]
     off  : Missing offset[0:31]
     dr24 : diagnose register #24 -- contains base address of table
     dr25 : diagnose register #25 -- contains mask bits (based on table size)

   extru   off,19,20,off_tmp1          ; right shift vpn
   zdep    off_tmp1,27,20,off_tmp2     ; position vpn at 8:27
   zdep    spc,22,16,spc_tmp1          ; position space at 7:22
   xor     spc_tmp1,off_tmp2,hash_addr ; perform hash
   mfdiag  dr25,mask                   ; get mask value from diag reg
   depi    -1,31,12,mask               ; don't mask bits 20:31
   and     mask,hash_addr,hash_addr    ; mask out bits of hashed addr based
                                       ;   on table size
   mfdiag  dr24,base                   ; get base addr from diag reg
   depi    0,31,12,base                ; only bits 0:19 are significant
   or      base,hash_addr,hash_addr    ; merge in the base addr of the table

   Hash_addr is a 32-bit real addr that is used to access the cache
   and to check for a dmiss.
```

6.7.3 Structure of Hardware-Visible Table

This is what is assumed about the organization of the hw-visible table in order to implement the hw TLB handler. Each 8 word line of the table holds two entries and is organized as shown below. Bits reserved are indicated by S for software fields, 0 for hardware fields.

```
                  +-+--------------------------+--------------------------+
word0 tag1        |V|        offset[0:14]      |        space[16:31]      |
                  +-+--------------------------+--------------------------+
                   0 1                       15 16                      31

                  +--------+-----------+-+-----+-----------------------+---+
word1 prot1       | RSTDB  |   ACR(7)  |U| 000 |     access_id(15)     | S |
                  +--------+-----------+-+-----+-----------------------+---+
                   0      4 5        11 12 13 15 16                   30  31

                  +------------+---------------------------------+--------+
word2 rpn1        |  SSS0000   |             rpn[0:19]           | SS000  |
                  +------------+---------------------------------+--------+
                   0          6 7                              26 27    31

                  +-------------------------------------------------------+
word3 next1       |           real address of next pdir entry             |
                  +-------------------------------------------------------+

                  +-+--------------------------+--------------------------+
word4 tag2        |V|         offset[0:14]     |        space[16:31]      |
                  +-+--------------------------+--------------------------+
                   0 1                       15 16                      31

                  +--------+-----------+-+-----+-----------------------+---+
word5 prot2       | RSTDB  |   ACR(7)  |U| 000 |     access_id(15)     | S |
                  +--------+-----------+-+-----+-----------------------+---+
                   0      4 5        11 12 13 15 16                   30  31

                  +-----------+---------------------------------+---------+
word6 rpn2        |  SSS0000  |             rpn[0:19]           |  SS000  |
                  +-----------+---------------------------------+---------+
                   0         6 7                              26 27     31

                  +-------------------------------------------------------+
word7 next2       |           real address of next pdir entry             |
                  +-------------------------------------------------------+
```

Offset bits 15-19 are implicit in the address of the PDIR entry and are not stored in the tag.

The hashing algorithm will provide an address that points to either word0 or word4 of a line. If the line is not resident in cache the processor will bring it in from main memory (it is required to be in main memory). The tag (either word0 or word4) is read, the valid bit of the tag is checked to ensure that it is valid and the tag is compared to the missing vpn[0:14] and space[16:31]. Also the reference bit (R-bit) of the protection word (either word1 or word5) is checked to ensure that it is a "1". If all of the checks pass then the protection and rpn are inserted into the on-chip TLB.

After the insert to the on-chip TLB is done, the access that had the TLB miss is re-translated and any TLB traps resulting from the re-translation will occur (for example DATA_MEM_PROT_TRAP).

If any of the checks fail then the CPU will trap to software with a DTLB_MISS_FAULT. CR28 is used to pass information to the trap handler about the address of the PDIR. The table below shows how CR28 is updated:

```
 ((R=1) and (V=1))  (Tag Match)  DR25[29]  DR25[30] |  CR28 value
+---------------------------------------------------+--------------+
|     False             DC          DC        DC    | current pdir |
|     True             False        0         DC    | current pdir |
|     True             False        1         0     | next pdir    |
|     True             False        1         1     | word3 of line|
+---------------------------------------------------+--------------+
current pdir  : real addr of current pdir entry (points to word0 or word4)
next pdir     : real addr of next pdir entry (this is word3 if we hashed to
                word0 or word7 if we hashed to word4).
word3 of line : this is word3 regardless of whether we use first or second
                entry in the line.
```

6.7.4 Diagnose Register Organization

```
Diagnose register #24 (write only).
+-----------------------------------+------------+-----------------+
| base address of hw-visible table |   ------   |  dcache conf    |
+-----------------------------------+------------+-----------------+
                0:19                    20:23         24:31

The base address must be aligned to a boundary that is a multiple
of the size of the hardware visible table.  This allows us to merge
in the base address and the table offset instead of doing an addition.
```

The dcache conf bits are explained in the diagnose chapter.

Diagnose register #25 (partial read capability - see DIAGNOSE chapter).

```
+-------------------------------+--------+-----+---+---+---+---+---+
| P | ------ |      mask        | ------ |  FP | - | I | U | N | D |
+-------------------------------+--------+-----+---+---+---+---+---+
  0    1:6          7:19          20:23  24:26  27  28  29  30  31
```

```
P     -- Continuously latches the power-fail signal (read-only).

mask  -- Effectively sets the size of the hw-visible table by
         determining which bits come from the hashed address and
         which bits come from diag reg #24 (base address for table).
         mask[7:19] enables an "or" of base_addr[7:19] and hash_addr[7:19]

         Example:
            A "0" in mask[13] will set real_addr[13] = base_addr[13]
            A "1" in mask[14] will set real_addr[14] = hashed_addr[14]

FP    -- Sets the FP delay as explained in the diagnose chapter.

I     -- Set to "1" to disable the ITLB hw handler, "0" to enable.
         It is not set by power-on or reset but rather by PDC code.

U     -- Set to "1" to enable the updating of CR28 with the next-pointer
         if the tag was valid and didn't match the missing space/offset.
         If set to "0", CR28 will always contain the address of the
         entry in the hw-visible table (current pdir).

N     -- Set to "1" to force the next pointer to always come from
         word3.  If set to "0" the next pointer will come from word3
         if we use the entry in the even quadword or from word7 if
         we use the entry in the odd quadword.

D     -- Set to "1" to disable the DTLB hw handler, "0" to enable.
         It is not set by power-on or reset but rather by PDC code.
```

The minimum size of the hw-visible table is 4Kbytes (256 PDIR entries).
The maximum size of the hw-visible table is 32Mbytes.
The mask field is set based on the table size as shown below:

table size	entries	mask[0:19]
4 Kbytes	256	00000000000000000000
8 Kbytes	512	00000000000000000001
<etc>		
32 Mbytes	2M	00000001111111111111

For every bit set in the mask field the corresponding bit in the base address must be a "0".

6.7.5 Penalties

Type	Penalty
DTLB hit and insert	11 states
DTLB trap to software	11 states
ITLB hit and insert	11 states
ITLB trap to software	11 states

6.7.6 Buddy Pages

PA7100LC will not implement buddy page handling. The decision was made based upon the performance numbers and the estimated complexity that would be required for implementation.

6.7.7 Relied-upon Translations

If the hardware TLB handler inserts any entries to the on-chip TLB it must force a re-translation of B-stage data access (if any) to ensure we don't violate the rules for relied-upon translations outlined in the PA-RISC manual.

6.8 Implementation Specific Inserts

There are two new implementation specific instructions added to reduce the TLB miss penalty for the software miss handler. These instructions use undefined minor opcode bits. The instructions reduce the TLB miss penalty in two different ways:

```
(1) They insert directly from control registers.  For DTLB misses
    ISR/IOR are used.  For ITLB misses the front elements
    of IIASQ/IIAOQ are used.
(2) They execute in fewer cycles than the regular TLB inserts.
```

The architected insert instructions are also implemented in order to be architecturally compliant and also to provide easy-to-use inserts for code which does not need to be handcrafted for performance.

```
Summary of instructions:
    idtlbaf -- fast insert dtlb address     (0 penalty states)
    idtlbpf -- fast insert dtlb protection  (1 penalty state)
    iitlbaf -- fast insert itlb address     (0 penalty states)
    iitlbpf -- fast insert itlb protection  (3 penalty states)
```

See the Appendix for instruction formats and software restrictions.

Chapter 7

Floating Point

7.1 Overview

PA7100LC contains 64 bit floating-point ALU, multiply, and divide/square root circuits and a 32x64 bit floating-point register file.

The floating point unit implements the PA-RISC 1.1 architecture (version 11/90). In addition, it implements the following product-specific features:

- Accelerated graphics clip tests
- Multiply and truncate
- Hardware underflow mode

The latencies and issue rates of floating-point operations are in the table below. The first number is the latency in cycles and the second number is cycles per instruction issue.

	Single	Double
Add/Subtract	2/1	2/1
Multiply	2/1	3/2
Divide	8/8	15/15
Square Root	8/8	15/15

Peak performance at 60 MHz is 120 megaflops single precision and 60 megaflops double precision.

The PA7100LC floating-point model number is x'0D. The revision for the first release is x'01.

7.2 Instruction Decoding Rules

7.2.1 Reserved-Op Exceptions

The reserved-op exception occurs on:

- A x'0C or x'0E instruction with a reserved or undefined sub-op:
  - class 0 subops 1,6,7
  - class 2 subops 2-7
  - class 3 subops 4-7
- A x'0E instruction, with a FMT code of b'11.
- A x'0E instruction other than XMPYU with bit 23 set (integer op).
- A XMPYU with bit 20 set (double precision).

Reserved-op exceptions are always reported through the unimplemented exception/trap rather than an immediate assists exception trap. In other words, the trap handler will see the T bit set and the offending instruction marked unimplemented in exception register 2.

7.2.2 Emulated Instructions

PA7100LC relies on software to emulate the following instructions:

- Any quad precision flop
- Any FRND instruction

These will raise the unimplemented exception/trap. The trap handler will see the T bit set and the offending instruction marked unimplemented in exception register 2.

A subroutine call may be much faster than the OS emulation routines.

7.2.3 Product-Specific Instructions

- FMPYCFXT is encoded as FMPYADD with zeroes in the ra field.
- Graphics clip tests are encoded in bits 27:31 of an FTEST.

For more information see the "Product-Specific Features" section.

7.2.4 FTEST look-alikes

The following are undefined:

- N bit set on a x'0E instruction.
- N bit set on a x'0C instruction other than FTEST (class 2, subop 1).
- N bit clear on an FTEST (x'0C, class 2, subop 1).
- Class 2, subop 1 (would-be FTEST) on a x'0E instruction.

7.2.5 Miscellaneous Undefined Instructions

The following are undefined:

- Any flop which uses register 1, 2, or 3 as a source operand.
- Any flop which uses register 0, 1, 2, or 3 as a result operand.
- FMPYCFXT,SGL when fp register 16L is nonzero.
- A x'0C instruction with a format code of 10.
- FCNVFF,SGL,SGL - this is treated as a FCPY,SGL.
- FCNVFF,DBL,DBL - this is treated as a FABS,DBL.
- COPR,0,0 if the most recent FP instruction was not an FSTD 0.
- COPR,0,0 if the next FP instruction is not an FSTD 0.
- FTEST with clip-test completer, if the most recent FP instruction was a COPR,0,0.

7.3 Unimplemented Exception/Trap

The only kind of exception generated by PA7100LC floating point is the unimplemented exception. It is always signaled with a delayed floating-point exception trap. The unimplemented trap is raised instead of the overflow, underflow, division by zero, invalid and inexact traps.

The I, V, O, U and Z flags in the floating-point status register are never set due to an unimplemented trap.

In the lists below, operands are classified as either norm, denorm, zero, inf, or NaN. Some of the cases are marked with (D) or (!D). This means the exception only occurs when the D bit is set or cleared, respectively.

The following conditions cause the unimplemented trap:

- Various:
  - Reserved-op or Emulated conditions (see above)

- FABS:
  - only the Reserved-op and emulated conditions (see above)
- FCPY:
  - only the Reserved-op and emulated conditions (see above)
- XMPYU:
  - only the Reserved-op and emulated conditions (see above)
- FADD:
  - input nan
  - +inf + -inf with invalid enabled
  - overflow with overflow enabled
  - inexact with inexact enabled
  - input denorm (!D) unless the other operand is inf
  - tiny result (!D)
  - tiny result with inexact enabled (D)
- FSUB:
  - input nan
  - +inf - +inf with invalid enabled
  - -inf - -inf with invalid enabled
  - overflow with overflow enabled
  - inexact with inexact enabled
  - input denorm (!D) unless the other operand is inf
  - tiny result (!D)
  - tiny result with inexact enabled (D)
- FCNVff:
  - input nan
  - overflow with overflow enabled
  - inexact with inexact enabled
  - input denorm (!D)
  - tiny result (!D)
  - tiny result with inexact enabled (D)
- FCNVxf:
  - inexact with inexact enabled
- FCNVfx:
  - input nan
  - input inf
  - input denorm (!D)
  - overflow **
  - inexact with inexact enabled
- FCNVfxt:

  - input nan
  - input inf
  - input denorm (!D)
  - overflow **
  - inexact with inexact enabled
- FCMP:
  - input signalling nan
  - input quiet nan with bit 31 of instruction set
- FMPY:
  - input nan
  - inf * zero and invalid enabled
  - inf * denorm and invalid enabled (D)
  - overflow with overflow enabled
  - inexact with inexact enabled
  - tiny result (!D)
  - tiny result with inexact enabled (D)
  - input denorm (!D) unless other operand is zero or inf
- FDIV:
  - input nan
  - inexact with inexact enabled
  - overflow with overflow enabled
  - denorm / denorm (!D)
  - denorm / norm (!D)
  - norm / denorm (!D)
  - inf / inf with invalid enabled
  - zero / zero with invalid enabled
  - zero / denorm with invalid enabled (D)
  - denorm / zero with invalid enabled (D)
  - denorm / denorm with invalid enabled (D)
  - denorm / zero with divz enabled (!D)
  - norm / zero with divz enabled
  - norm / denorm with divz enabled (D)
  - tiny result (!D)
  - tiny result with inexact enabled (D)
- FSQRT:
  - input nan
  - input negative norm and invalid enabled
  - input negative infinity and invalid enabled
  - input negative denorm and invalid enabled (!D)
  - input positive denorm (!D)
  - inexact with inexact enabled

- FMPYADD:
  - union of multiply and add conditions
- FMPYSUB:
  - union of multiply and subtract conditions
- FMPYCFXT:
  - union of multiply and truncate conditions

** fcnvfx,dbl,sgl and fcnvfxt,dbl,sgl will raise the unimplemented exception when their operand is equal to the largest representable negative integer, even though this is not strictly necessary.

When either operation of a multi-op instruction traps, neither operation writes its result and no exception flags are set in the floating-point status register.

7.3.1 Overflow Exception

The following conditions raise the overflow exception provided that the unimplemented trap does not occur. The O flag in the floating-point status register is set.

- FADD, FSUB, FCNVff, FMPY, FDIV, FMPYADD, FMPYSUB:
  - overflow with overflow disabled
- FMPYCFXT:
  - overflow in multiply with overflow disabled

7.3.2 Division by Zero Exception

The following conditions raise the division by zero exception provided that the unimplemented trap does not occur. The Z flag in the floating-point status register is set.

- FDIV:
  - denorm / zero with divz disabled (!D)
  - norm / zero with divz disabled
  - norm / denorm with divz disabled (D)

7.3.3 Invalid Exception

The following conditions raise the invalid exception provided that the unimplemented trap does not occur. The V flag in the floating-point status register is set. The result register is set to a quiet NaN.

- FADD:
  - +inf + -inf with invalid disabled
- FSUB:
  - +inf - +inf with invalid disabled
  - -inf - -inf with invalid disabled
- FMPY:
  - inf * zero with invalid disabled
  - inf * denorm with invalid disabled (D)
- FSQRT:
  - input negative infinity with invalid trap disabled
  - input negative norm with invalid trap disabled
  - input negative denorm with invalid trap disabled (!D)
- FDIV:
  - inf / inf with invalid disabled
  - zero / zero with invalid disabled
  - zero / denorm with invalid disabled (D)
  - denorm / denorm with invalid disabled (D)

7.3.4 Inexact Exception

The following conditions raise the inexact exception provided that the unimplemented trap does not occur. The I flag in the floating-point status register is set.

- FADD, FSUB, FCNVFF, FCNVFX, FCNVFXT, FCNVXF, FMPY, FDIV, FSQRT, FMPYADD, FMPYSUB, FMPYCFXT:
  - inexact and inexact disabled
- FADD, FSUB, FCNVff, FMPY, FDIV, FSQRT, FMPYADD, FMPYSUB, FMPYCFXT:
  - tiny result and inexact disabled (D)

7.3.5 Underflow Exception

The following conditions raise the underflow exception provided that the unimplemented trap does not occur. The U flag in the floating-point status register is set.

- all operations: never

7.3.6 Exception Registers

The excepting flop will be in exception register 2. PA-RISC 1.1 exception codes are used. Exception registers 1, 3, 4, 5, 6, and 7 may be loaded or stored, but hardware will never place an excepting flop in them.

Exception register 2 is guarranteed to retain its contents as long as the T bit remains set and continuing thereafter until a flop is executed, or until it is explicitly cleared by software with a load.

7.4 Product-Specific Features

PA7100LC has floating-point features which are product-specific (not part of PA-RISC 1.1). Programmers who choose to take advantage of these features are warned that some older machines do not implement them and even future machines may not implement them. Behavior on other machines will be undefined, meaning that even a trap for emulation is not guaranteed. Except for Hardware Underflow Mode, these features cannot be disabled.

7.4.1 Hardware Underflow Mode

PA7100LC implements a quick hardware underflow mode for floating-point operations. This mode is enabled by setting bit 26 of the floating-point status register, called the "D" bit (for Default Underflow Trap Handler).

In hardware underflow mode, operations which would normally signal the underflow exception will just return a zero result with no exception. Input denorms are treated as signed zeroes. The underflow flag is never set. The inexact flag and inexact exception are detected just as in IEEE mode except that denormalized operands are treated as signed zeroes, and when a result is flushed to zero the inexact flag is set (if inexact traps are enabled there will be an exception).

Note that when this mode is enabled, computations are not compliant with the IEEE floating-point standard.

This mode does not affect the speed of floating point operations. It just saves the overhead of a trap handler when there are denormalized operands or when an operation underflows.

Note that FABS and FCPY are not affected by this mode. They do not flush denorms to zero.

Hardware underflow mode is a product-specific feature.

7.4.2 Multiply and Truncate (FMPYCFXT)

```
FMPYCFXT,fmt rm1,rm2,tm ta

-----------------------------------------------------------------
|   x'06    |   rm1   |   rm2   |   ta    |    0    |f |   tm    |
-----------------------------------------------------------------
     6           5         5         5         5     1      5
```

This instruction is like FMPYADD except that instead of doing the add it performs an FCN-VFXT,fmt,sgl on register ta and puts the resulting signed integer back into ta.

The convert always produces a 32 bit result. If the format is double then the result is placed in the MSW (bits 0:31) of ta and the LSW of ta becomes undefined.

The single precision version of this instruction (fmpycfxt,sgl) is undefined if register 16L is nonzero. So before using fmpycfxt,sgl you must load a zero into register 16L.

7.4.3 Graphics Clip Tests

Bits 10:20 of the floating-point status register serve two purposes. First, they are the model and revision fields set by the COPR,0,0 instruction as defined by the PA-RISC 1.1 architecture. Second, when combined with bit 5 (the C bit) they form a 12 entry queue of compare results.

When a compare (FCMP) instruction completes, the queue is advanced as follows:

status[11:20]	:= status[10:19]
status[10]	:= status[5]
status[5]	:= FCMP result

Thus, bit 20 contains the result of the oldest compare, and bit 5 contains the result of the most recent compare (consistent with its PA-RISC 1.1 definition as the C bit).

The queue can be stored to or loaded from memory like other bits of the status register. Obviously the queue will be destroyed by the COPR,0,0 instruction.

The FTEST instruction with no completer tests the C bit as it does in PA-RISC 1.1. Seven new variations of FTEST are implemented.

```
FTEST,cmplt

-----------------------------------------------------------------
|   x'0C   |    0    |    0    |   1   |  0 | 2   |  0  |1 |  cond  |
-----------------------------------------------------------------
     6          5         5        3      2   2     3    1     5
```

The new variations are intended for the "trivial accept" and "trivial reject" cases in 3D and 2D graphics clip testing. ACC6/ACC4/ACC2 may also be useful to check the results of several previous compares with a single FTEST. For example,

```
FCMP; FCMP; FCMP; FCMP; FTEST,acc4; branch;
```

executes in 7 cycles, whereas

```
FCMP; FTEST; branch; FCMP; FTEST; branch;
FCMP; FTEST; branch; FCMP; FTEST; branch;
```

executes in 16 cycles (assuming the branches are usually nullified).

The condition encodings are listed below with their conditions for nullification and their suggested completer mnemonics.

Completer	Encoding (bits 27:31)	Condition for nullifying next instruction
none	x'00	status[5] == b'1
,acc	x'01	status[5,10:20] == b'000000000000
,acc8	x'05	status[5,10:16] == b'00000000
,acc6	x'09	status[5,10:14] == b'000000
,acc4	x'0D	status[5,10:12] == b'0000
,acc2	x'11	status[5,10] == b'00
,rej	x'02	status[5] == 1 && status[15] == 1 || status[10] == 1 && status[16] == 1 || status[11] == 1 && status[17] == 1 || status[12] == 1 && status[18] == 1 || status[13] == 1 && status[19] == 1 || status[14] == 1 && status[20] == 1
,rej8	x'06	status[5] == 1 && status[13] == 1 || status[10] == 1 && status[14] == 1 || status[11] == 1 && status[15] == 1 || status[12] == 1 && status[16] == 1

7.5 Performance Tuning

7.5.1 Notation

Minimum Distance is the number of cycles between two instructions necessary to avoid a pipeline interlock. Consecutive instructions are 1 cycle apart. For example in without super scalar execution this:

```
A
other instruction
B
```

would cause a 1 cycle penalty if there were a minimum distance of 3 between A and B. If there were a minimum distance of 3 between A and B and a minimum distance of 6 between A and C, then

```
A
B
other instruction
C
```

would cause a 3 cycle penalty: 2 cycles on B and 1 cycle on C. This is because penalties associated with two or more constraints can be served in parallel.

7.5.2 Latencies

A Flop is one of the instructions shown in the table below. Loads, stores and FTESTs are not flops. The table shows the Execution Time (latency cycles) for each flop and which functional unit is used to execute it.

Flop	Execution Time	Functional Unit	
FCPY,SGL/DBL	2	ALU	
FABS,SGL/DBL	2	ALU	
FADD,SGL/DBL	2	ALU	
FSUB,SGL/DBL	2	ALU	
FCMP,SGL/DBL	2	ALU	
FCNV*,SGL/DBL	2	ALU	
FMPYADD,SGL	2	ALU and MPY	
FMPYSUB,SGL	2	ALU and MPY	
FMPYCFXT,SGL	2	ALU and MPY	
FMPY,SGL	2	MPY	
FMPYADD,DBL	3	ALU and MPY	
FMPYSUB,DBL	3	ALU and MPY	
FMPYCFXT,DBL	3	ALU and MPY	
FMPY,DBL	3	MPY	
XMPYU	3	MPY	(integer multiply)
FDIV,SGL	8	DIV	
FDIV,DBL	15	DIV	
FSQRT,SGL	8	DIV	
FSQRT,DBL	15	DIV	

In the tables that follow, N1 is the execution time of the first instruction.

These distances are preliminary and are subject to change during design. Of course the performance cases would only be made worse in an extreme situtation.

7.5.3 Data Cache Contention Constraints

1st Instruction	ST*,FSTW*,FSTD*
2nd Instruction	LD*,ST*,FLD*,FST*
Minimum Distance	2 cycles

For the second instruction, LD* does not include LDIL or LDO since these "loads" do not actually access the data cache.

7.5.4 Functional Unit Contention Constraints

1st Instruction	FDIV or FSQRT
2nd Instruction	*
Minimum Distance	N1 cycles

1st Instruction	XMPYU or FMPY,dbl
2nd Instruction	*
Minimum Distance	2 cycles

This rule means that there is always a 1 cycle penalty for any double precision FMPY, a 7 cycle penalty for a single precision FDIV/FSQRT, and a 14 cycle penalty for a double precision FDIV/FSQRT.

7.5.5 Data Dependency Constraints (Performance Cases)

1st Instruction	FLD*
2nd Instruction	Flop
Minimum Distance	2 cycles

The target of the load equals one of the sources of the flop. (Increase the distance by 1 cycle if the load is singleword singleword and the flop's source is double precision.)

1st Instruction	Flop
2nd Instruction	Flop
Minimum Distance	N1 cycles

The target of the first flop equals one of the sources of the second flop. (Increase the distance by 1 cycle if the format is different, i.e. sgl-dbl, dbl-sgl, int-fp, fp-int.)

1st Instruction	FCMP
2nd Instruction	FTEST
Minimum Distance	2 cycles

7.5.6 Dependency Constraints (Non-performance Cases)

1st Instruction	FLD*
2nd Instruction	FST*,FLD*
Minimum Distance	3 cycles

The target of the first load equals the target of the second load or source of the store.

1st Instruction	FLD*
2nd Instruction	Flop
Minimum Distance	3 cycles

The target of load equals a target of the flop. Exception: If the load and flop are bundled together then the minimum distance is only 2 cycles.

1st Instruction	Flop
2nd Instruction	FLD*/FST* 0-3
Minimum Distance	N1+1 cycles

1st Instruction	FLD*/FST* 0-3
2nd Instruction	Flop
Minimum Distance	4 cycles

1st Instruction	FLDD*/FSTD* 0
2nd Instruction	FLD*/FST*/FTEST
Minimum Distance	4 cycles

These are loads and stores of the status and exception registers.

7.5.7 Other Rules

If the first instruction is a load that suffers a d-cache miss then add the latency of the miss to the minimum distance.

The minimum distance may be shortened if one or both of the two instructions is nullified. This varies from case to case and can depend on other factors such as unrelated interlocks.

7.5.8 Example

This example is coded so there are no interlocks.

```
FLD*     mem,r4
other
FADD,sgl r4,r4,r5   # 2 cycles away from load
FADD,sgl r7,r8,r9   # independent of 1st add
FMPY,sgl r5,r4,r6   # 2 cycles away from 1st add
FST*     r6,mem     #
other
FLD*     mem,r10    # 2 cycles away from store
```

With dual issue this could execute in 7 cycles because the FMPY could bundle with the FST. Note that the first FADD will be bundled with the "other" instruction right before it but you don't gain any cycles because the FADD needs to be 2 cycles away from the FLD.

Chapter 8

Diagnose

8.1 Introduction

PA7100LC diagnose is different than previous implementations. One of the largest changes is the removal of the DOUBLE-DIAGNOSE rule. There are also changes to the diagnose registers, the diagnose instructions, and the software restrictions regarding diagnose for PA7100LC. If you plan to use PA7100LC diagnose instructions, please read this entire chapter.

The document is organized into the parts listed below:

DIAGNOSE REGISTERS: Lists all of the diagnose registers in the PCX-L CPU, gives their formats, and a description of the functionality of each field/bit in the diagnose register.

DIAGNOSE INSTRUCTIONS: Contains a description of diagnose instructions and gives their encodings for the PA7100LC processor.

SOFTWARE CONSTRAINTS: Describes software constraints above and beyond those listed previously.

This diagnose chapter combined with the fault tolerance chapter should provide enough information for an architectural review, system initialization, self test, and operating system needs. Hopefully, this will meet the needs of everyone. If not, let us know.

8.2 Diagnose Registers

This section will describe each diagnose register and the subfields contained therein. A brief description of the function of each subfield will be given.

8.2.1 Processor

There are many diagnose registers on the CPU chip:

- CPU Diagnose Register : Reg # 0. This is the main CPU diagnose register with bits for miscellaneous uses.
- CPU Diagnose Register : Reg # 1. This is the architected Itimer Counter (not the Compare register). The MTCPU #1 instruction provides a way to set to the Itimer Counter. This helps reduce self-test time.
- UTLB Diagnose Register : Reg # 8.
- Hardware TLB Handler Address Base and Store Buffer Configuration: Reg #24.
- Hardware TLB Handler Configuration: Reg #25.
- Debug-Mode Configuration: Reg #26.
- Level-1 ICache Even Data Word: Reg #27. This provides a way to read and write the high-word portion of the on-chip instruction cache.
- Level-1 ICache Odd Data Word: Reg #28. This provides a way to read and write the low-word portion of the on-chip instruction cache.
- Level-1 ICache Tag Word: Reg #29. This provides a way to read and write the tag portion of the on-chip instruction cache.

8.2.2 CPU Diagnose Register 0

The format of the CPU Diagnose Register 0 is shown below:

```
 ---------------------------------------------------------------
|                      CPU DIAGNOSE REGISTER 0                  |
|_______________________________________________________________|
|_______________________________________________________________|
|                                0:31                           |
 ---------------------------------------------------------------

 0: 5  Rev#        (r)  : CPU Revision Number
    6  L2IHPMC     (r/c): Level2 (off-chip) I-Cache Error Flag
    7  L2IHPMC_DIS (r/w): Level2 (off-chip) I-Cache HPMC Disable (Mask)
    8  L2DHPMC     (r/c): Level2 (off-chip) D-Cache Error Flag
    9  L2DHPMC_DIS (r/w): Level2 (off-chip) D-Cache HPMC Disable (Mask)
   10  L1IHPMC     (r/c): Level1 ( on-chip) I-Cache Error Flag
   11  L1IHPMC_DIS (r/w): Level1 ( on-chip) I-Cache HPMC Disable (Mask)
12:15  L2PARERR    (r/c): Level2 (off-chip) Cache Parity Error Indicators
   16  STORE[0]    (r/w): Scratch Space
   17  PF_MASK     (r/w): Power-fail trap mask
   18  STORE[1]    (r/w): Scratch Space
   19  FAST_MODE     (r): 0=fast, 1=slow
   20  ISTRM_EN    (r/w): Enable ICache streaming
21:22  DUAL_DIS    (r/w): Disable Dual-Issue (superscalar execution)
   23  ENDIAN      (r/w): Use Little-Endian mode when taking a trap
   24  SOU_EN      (r/w): Stall-on-Use enable for Data Cache Misses
   25  SHINT_EN    (r/w): No-Fill on Miss Store Hints enable
   26  IPREF_EN    (r/w): L2 to L1 Instruction cache prefetch enable
   27  DHASH_EN    (r/w): Level2 (off-chip) D-Cache Hash Enable
   28  IHASH_EN    (r/w): Level2 (off-chip) I-Cache Hash Enable
   29  L1ICACHE_EN (r/w): Level1 ( on-chip) I-Cache Enable
   30  HIT         (r)  : Diagnose Cache Read Hit indication
   31  PARERR      (r)  : Diagnose Cache Read Parity Error indication

(r)   means Read-Only
(r/w) means Read-Write
(r/c) means Read-Clear (Clear by Moving '1' to it)

POWERUP values:
    DR0[0:5] powers up with the revision number
    DR0[6:15] powers up with undefined values
    DR0[16:31] powers up to zero, except [22]=1
```

Descriptions of CPU Diagnose Bits:

RN: This holds the revision number of the CPU chip. For the first release of the CPU the revision number is 0.

L2IHPMC: This bit is set whenever a Level2 (off-chip) ICache Parity Error is detected. It is qualified by the L2IHPMC_DIS bit before causing a trap. This bit remains set until software clears it in the HPMC handler. It is cleared only when a '1' is moved to this bit position. Moving a '0' to this bit position does not change this bit.

L2IHPMC_DIS: This bits disable taking an HPMC due to a Level2 (off-chip) ICache Parity Error. This is provided as a debug feature for early CPU releases. This is more accurately called a mask bit because HPMCs are still collected and are held pending.

L2DHPMC: This bit is set whenever a Level2 (off-chip) DCache Parity Error is detected. It is qualified by the L2DHPMC_DIS bit before causing a trap. This bit remains set until software clears it in the HPMC handler. It is cleared only when a '1' is moved to this bit position. Moving a '0' to this bit position does not change this bit.

L2DHPMC_DIS: This bits disable taking an HPMC due to a Level2 (off-chip) DCache Parity Error. This is provided as a debug feature for early CPU releases. This is more accurately called a mask bit because HPMCs are still collected and are held pending.

L1IHPMC: This bit is set whenever a Level1 (on-chip) ICache Parity Error is detected. It is qualified by the L1IHPMC_DIS bit before causing a trap. This bit remains set until software clears it in the HPMC handler. It is cleared only when a '1' is moved to this bit position. Moving a '0' to this bit position does not change this bit.

L1IHPMC_DIS: This bits disable taking an HPMC due to a Level1 (on-chip) ICache Parity Error. This is provided as a debug feature for early CPU releases. This is more accurately called a mask bit because HPMCs are still collected and are held pending.

L2PARERR: These bits are the outputs of the four parity trees for the off-chip caches. Each bit updates independently. When set, a bit remains set until software clears it by moving a '1' into the appropriate bit position. When a system observes off-chip-cache parity errors, these bits can be used to determine which group of RAMs is having a failure. The order of these bits is as follows: bit[12]=even data parity, bit[13]=odd data parity, bit[14]=tag parity, bit[15]=dirty parity.

STORE: These bits provide for temporary storage. One self-test convention for one of these is to indicate when an HPMC is expected by self-test code rather than 'real'.

PF_MASKH: When this bit is set to 1, power-fail traps are disabled.

FAST_MODE: Read-only bit. 0 indicates running at full-speed.

ISTRM_EN: Enable instruction cache streaming for instruction fetches from memory. Disable only to debug prototype systems.

DUAL_DIS: Disable Dual-Issue (superscalar execution). The following options are supported: 00=all bundles enabled, 01=all bundles except ldw/ldw,stw/stw are enabled, 10=only flop-non_flop bundles are enabled, 11=no bundles are enabled (single issue mode). Disable only to debug prototype hardware.

ENDIAN: Set this bit to enable Little-Endian mode for traps and hardware TLB accesses (ie. on traps, this bit is set into the PSW-E bit, and this bit is used in place of the PSW-E bit for HTLB accesses).

SOU_EN: Enable the "Stall-on-Use" optimization for D-Misses (both loads and stores). Disable only to debug prototype hardware (also must disable ldw/ldw bundles).

SHINT_EN: Enable Store Hints for store instructions at privilege 0. Disable only to debug prototype hardware.

IPREF_EN: Enable L2 to L1 instruction cache prefetching. Disable only to debug prototype hardware.

DHASH_EN: Enable Level2 (off-chip) D-Cache Hashing (VPN and SID xoring) for improved performance.

IHASH_EN: Enable Level2 (off-chip) I-Cache Hashing (VPN and SID xoring) is enabled for improved performance.

L1ICACHE_EN: Enable the Level1 (on-chip) instruction cache. Disable only to debug prototype hardware or to operate in degraded mode after sensing a permanent cache error. Disabling the L1 ICache does not automatically disable HPMC's from L1 ICache. Software should set L1IHPMC_DIS when clearing L1ICACHE_EN. See the Software Constraints for another important restriction.

HIT: This bit latches the Level-2 Cache Hit signal for the Read Diagnose L2 RAMs instructions. There is no similar bit for the L1 cache. This bit is '1' for a hit. This bit aids in the characterization of the Hit Compare path.

PARERR: This bit latches the Level-1 or Level-2 Cache Parity Error signal for the Read Diagnose L1 or L2 RAMs instructions. This bit is '1' for a parity error. This bit aids in the characterization of the parity trees.

8.2.3 CPU Diagnose Register 8

Diagnose Register #8, UTLB Diagnose Register, is described in the TLB chapter of this ERS.

8.2.4 CPU Diagnose Register 24

```
 ----------------------------------------------------------------
|                      CPU DIAGNOSE REGISTER 24                  |
|________________________________________________________________|
|________________________________________________________________|
|    HTLB Handler Base       | ----- |  DCache Size Configuration |
|          0:19              | 20:23 |           24:31            |
|________________________________________________________________|
```

```
HTLB_BASE (w)  : Address in Main Memory where the Hardware visible "PDIR"
table resides.  See the TLB chapter.
DCACHE_SIZE (w): These bits need to be set in accordance with the data cache
size. Each bit which is a "1" in 24:31 will disable the DADH and
TADH D-Cache Address bits 12:19 respectively. ie. For a 4K data cache set all
bits to 1. For a 1M data cache set all bits to zero.
```

8.2.5 CPU Diagnose Register 25

```
 ----------------------------------------------------------------
|                      CPU DIAGNOSE REGISTER 25                  |
|________________________________________________________________|
|________________________________________________________________|
| POWF | ------ | HTLB Mask | ----- | FP Delay | -- | HTLB CNTL  |
|   0  |   1:6  |    7:19   | 20:23 |   24:26  | 27 |   28:31    |
|_______________________________________________________________ |
```

```
POWFAIL      (r):  Set to 0 by hardware when powerfail signal is true.
HTLB_MASK    (w):  Sets the size of the hw-visible table for the hardware TLB
handler.  See the TLB Chapter.
FP_DELAY   (r/w):  Sets the floating-point programmable delay line used
for low frequency chip characterization.  For normal operation, these
bits must be set to 001 by PDC before the CCR is set and flops are executed.
HTLB_CNTL (r/w):  See the TLB Chapter
   28 -- ihtlb_dis
   29 -- upd_cr28_current
   30 -- next_wd3
   31 -- dhtlb_dis
```

8.2.6 CPU Diagnose Register 26

Diagnose Register #26, for hardware debug, is described in the debug chapter of this ERS.

8.2.7 CPU Diagnose Register 27

```
 ----------------------------------------------------------------
|                      CPU DIAGNOSE REGISTER 27                  |
|________________________________________________________________|
|________________________________________________________________|
|                  Even (high-word) data                         |
|                          0:31                                  |
|_______________________________________________________________ |
```

```
Diagnose Register 27, in the Level1 (on-chip) ICache is actually 2
different registers.  Moves to this register set into one of the the ICache
write latches to prepare for a Level1 ICache diagnose write.  Moves
from this register read one of the Level1 ICache read latches to complete
a Level1 ICache diagnose read operation.

EvenData   (r/w): data to/from the even word of the Level1 (on-chip) ICache
```

8.2.8 CPU Diagnose Register 28

```
 ----------------------------------------------------------------
|                      CPU DIAGNOSE REGISTER 28                  |
|________________________________________________________________|
|________________________________________________________________|
|                    Odd (low-word) data                         |
|                          0:31                                  |
|_______________________________________________________________ |
```

```
Diagnose Register 28, in the Level1 (on-chip) ICache is actually 2
different registers.  Moves to this register set into one of the the ICache
write latches to prepare for a Level1 ICache diagnose write.  Moves
from this register read one of the Level1 ICache read latches to complete
a Level1 ICache diagnose read operation.

OddData   (r/w): data to/from the odd word of the Level1 (on-chip) ICache
```

8.2.9 CPU Diagnose Register 29

```
 -------------------------------------------------------------
|                    CPU DIAGNOSE REGISTER 29                  |
|______________________________________________________________|
| FPDATA | FPTAG |  STEER[0:7]  |  RPN[0:19]  |  QPAGE[20:21]  |
|    0   |   1   |      2:9     |    10:29    |     30:31      |
|______________________________________________________________|
```

```
Diagnose Register 29, in the Level1 (on-chip) ICache is actually 2
different registers.  Moves to this register set into one of the the ICache
write latches to prepare for a Level1 ICache diagnose write.  Moves
from this register read one of the Level1 ICache read latches to complete
a Level1 ICache diagnose read operation.

Note: This register has 1 bit that is negative true; even_frih is inverted!

FPDATA/FPTAG (r/w): Parity bit for the doubleword of instructions or tag
(tag := STEER + RPN + QPAGE) in the accessed location of the Level1 ICache.
For reads, this is the bit read from the ICache (note that the Level1 ICache
computes even parity).  For writes, if this bit is '1', the result of the
parity tree will be flipped, causing a parity error to be seeded into the
Level1 ICache.
STEER[0:7] (r/w): Predecoded steering bits for the accessed location of the
Level1 ICache.  For writes, it is up to the code-writer to generate this
bits correctly to correspond with the data in DR27 and DR28.
The bits are as follows:
  [0] even_rih  (positive true) even instruction must go to RIH
  [1] even_frih (negative true) even instruction must go to FRIH
  [2] even_lih  (positive true) even instruction must go to LIH
  [3] even_flex (positive true) even instruction can go to RIH or LIH
  [4] odd_rih   (positive true) odd  instruction must go to RIH
  [5] odd_flex  (positive true) odd  instruction can go to RIH or LIH
  [6] odd_lih   (positive true) odd  instruction must go to LIH
  [7] odd_frih  (positive true) odd  instruction must go to FRIH
RPN[0:19]  (r/w): The real-page-number associated with the accessed location
of the Level1 ICache.  Note that the VALID bit is encoded in this field (ie.
to mark an entry invalid, set the upper 4 bits to 'F' (I/O space)).
QPAGE[20:21] (r/w): Quarter page - the two high-order bits of the page offset
of the address corresponding to the accessed location in the Level1 ICache.
```

Since the architected page size is 4KBytes, and the Level1 ICache is only 1KBytes, it is necessary to keep bits 20:21 of the virtual/real address in the Level1 cache for full address compares.

8.3 Diagnose Instructions

The PCX-L diagnose instructions and their encodings are described below.

8.3.1 Move To Diagnose Register

- Move To CPU Diagnose (MTCPU)

0 5	6 10	11 15	16 18	19 26	27 31
05	*t*	*x*	*0*	*code*	*0*

Figure 8.1: Move To Diagnose Format

```
MTCPU     t=DiagTarget#     x=GRSource#        code= 12
```

The contents of General Register x is moved to CPU Diagnose Register t. There are no Coprocessor or MIOC diagnose registers. Not all diagnose register bits can be affected by a Move To Diagnose Instruction because some diagnose register bits are read-only. These cases are identified in the section describing the Diagnose Registers.

The HPMC bits in CPU Diagnose Register #0 behave differently than other diagnose bits. A Move To Diagnose instruction with a '1' value of GR[x] for these bit positions causes the corresponding bit to be reset while a '0' value leaves the bit unchanged. This allows MTCPU to DR#0 without changing the value of these error bits.

8.3.2 Move From Diagnose Register

There are two instructions of this type:

- Move From CPU Diagnose via internal CH bus (MFCPU_C)
- Move From CPU Diagnose via internal TH bus (MFCPU_T)

0 5	6 10	11 15	16 18	19 26	27 31
05	*r*	*t_ch*	*0*	*code*	*t_th*

Figure 8.2: Move From Diagnose Format

```
MFCPU_C   r=DiagReg#      t_ch=GR#    t_th=0       code= 30
MFCPU_T   r=DiagReg#      t_ch=0      t_th=GR#     code= a0
```

The contents of Diagnose Register r is moved to a General Register. The GR is specified by t_ch or t_th for MFCPU_C and MFCPU_T, respectively. There are two "flavors" of MFCPU in order to make the design easier. Diagnose registers 0,8 are the only ones that uses MFCPU_C. Diagnose registers 25,27,28,29 use MFCPU_T. Diagnose registers 1,24,26 cannot be read. Not all diagnose register bits in each register can be affected by a Move From Diagnose Instruction because some diagnose register bits are write-only. These cases are identified in the section describing the Diagnose Registers. There are no Coprocessor or MIOC diagnose registers.

8.3.3 TOC and GR Shadow Diagnose Control

- TOC Enable (TOC_EN)
- TOC Disable (TOC_DIS)
- Set GR shadow registers from corresponding GRs (GR_SHDW)
- Set GRs from corresponding GR shadow registers (SHDW_GR)

0 5	6 10	11 15	16 18	19 26	27 31
05	*0*	*0*	*0*	*code*	*0*

Figure 8.3: TOC and GR-SHADOW Format

```
TOC_EN              code= 50
TOC_DIS             code= 52
SHDW_GR             code= d0
GR_SHDW             code= d2
```

The TOC_EN (TOC Enable) and TOC_DIS (TOC Disable) instructions enable or disable the taking of a TOC. They do not affect collecting the TOC condition, they merely mask the TOC trap in the same manner that the PSW-M bit masks HPMC trap.

The GR_SHDW instruction causes GR's 1,8,9,16,17,24,25 to be set into their corresponding shadow registers. The SHDW_GR instruction causes these shadow registers to be set into their corresponding GR. These 2 instructions must be preceeded by 2 sync instructions to guarantee no stall-on-use or bypass cases colliding with the gr to/from shadow move.

RESTRICTION: A "sync" instruction should precede these instructions to separate them from any preceding loads, stores, or flushes.

8.3.4 Diagnose Read Off-chip (Level2) Cache RAMs

There are five instructions of this type:

- Diagnose Read Level2 Instruction Data (RI2D)
- Diagnose Read Level2 Instruction Tag (RI2T)
- Diagnose Read Data Data (RDD)
- Diagnose Read Data Tag (RDT)
- Diagnose Read Data Data Virtual and Latch DTLB Diagnose (RDTLB)

0 5	6 10	11 15	16 17	18	19 26	27 31
05	*b*	*0*	*s*	*0*	*code*	*t*

Figure 8.4: Diagnose Read Off-chip RAMs Format

```
RI2D       b=address_base  s=don't care      t=GRTarget#    code=62
RI2T       b=address_base  s=don't care      t=GRTarget#    code=6a
RDD        b=address_base  s=don't care      t=GRTarget#    code=e2
RDT        b=address_base  s=don't care      t=GRTarget#    code=ea
RDTLB      b=address_base  s=address_space   t=GRTarget#    code=e6
```

RI2D, RI2T, RDD, RDT generate a "real-mode" address from GR[b]. RDTLB generates a "virtual-mode" address from GR[b] and the Space ID identified by the "s" field.

```
if s==0 Then SID<--SR[GR[b]{0..1} + 4
        else SID<--SR[s].
ie. Normal Space register encoding
```

RI2D/RDD/RDTLB set the addressed off-chip Instruction or Data Cache data word into GR[t] (even if it misses). The address used by these instructions is a word address. Note that these instructions always return data from the RAM's, never from the store buffer. This implies that a WDD-RDD sequence may not work as expected, but a WDD-SET_TAG-WDT-RDD sequence WILL work as expected.

RI2T/RDT set the addressed off-chip Instruction or Data Cache tag, tag parity, Dirty bit, Dirty Parity, and the two data parity bits into GR[t] according to the format given in Figure 8.5. The address used by this instruction is a double-word address. The "Arb enable" bit will be explained in the section covering diagnose RAM writes.

0 19	20	21	22	23	24	25	26 31
RPN	*Dirty*	*Dirty par*	*Tag par*	*Data par 0*	*Data par 1*	*Arb enable*	*unused*

Figure 8.5: Diagnose Read/Write Off-chip RAMs - Tag Format

RDTLB forces DTLB translation to be enabled while forcing all DTLB traps to be false. This may be useful for testing the UTLB. The RPN used for calculating "cache hit" is the RPN from the UTLB and the address issued to the D-Cache is hashed.

All Diagnose Read RAMs instructions other than RDTLB send a real mode RPN to hit-compare and do not hash the cache address.

CPU Diagnose Register #0 latches the Off-chip Cache Hit and Parity Error signals for the diagnose read access. The CPU will never service a Cache Miss which is signaled during a Diagnose Read Off-chip RAMs instruction.

In normal operation Tag parity is encoded over the twenty RPN bits, Dirty parity is encoded over the one Dirty bit, Data[0] parity is encoded over the most significant word of D-Cache data, and Data[1] parity is encoded over the least significant word D-Cache data.

Cache Parity Errors signalled during Diagnose Read Off-Chip RAM instructions WILL cause HPMCs unless the PSW-M bit is set or the appropriate bit of Diagnose register #0 is set (bit 7 or 9). ie. Such a cache parity error will set bit 6 or 8 of the CPU Diagnose register #0 and this bit will cause an HPMC as soon as the M-bit is clear and DR0 bit 7 or 9 is 0. If an HPMC is not desired then bit 7 or 9 of DR0 should be set (or the PSW-M bit) when executing this instruction and bit 6 or 8 of CPU Diagnose register #0 should be cleared after executing these instructions.

The cache address for Diagnose Read Off-chip RAMs instructions is NOT latched in any processor register.

If the instruction traps, no diagnose register will be set (except possibly the L2 parity and hit bits of CPU diagnose register 0) and no GR will be set.

RESTRICTION: A "sync" instruction should precede a sequence of diagnose read/write off-chip cache instructions to separate them from any preceding loads, stores, or flushes.

8.3.5 Diagnose Write Off-chip (Level2) Cache RAMs

There are five instructions of this type:

- Diagnose Write Level2 Instruction Data (WI2D)

- Diagnose Write Level2 Instruction Tag (WI2T)
- Diagnose Write Data Data (WDD)
- Diagnose Write Data Tag (WDT)
- Diagnose Set Tag (SET_TAG)

0 5	6 10	11 15	16 18	19 26	27 31
05	*b*	*x*	*0*	*code*	*0*

Figure 8.6: Diagnose Write Instruction RAMs

```
WI2D      b=address_base    x=GRsource#     code=72
WI2T      b=address_base                    code=7a
WDD       b=address_base    x=GRsource#     code=f2
WDT       b=address_base                    code=fa
SET_TAG                     x=GRsource#     code=42
```

These instructions are similar to the corresponding Diagnose Read Off-Chip Rams instructions except that the information goes from the cpu general registers into the RAMs. The GR is specified by the x field of these instructions.

WI2D/WDD will write GR[x] into the addressed single word off the Off-Chip instruction or data cache RAMs.

These instructions are best used only in the following sequences: For off-chip instruction writes: WI2D, SET_TAG, WI2T. For off-chip data writes: WDD, SET_TAG, WDT.

WI2D or WDD may be used by themselves only if correct parity is to be encoded and if only data values (ie. not tags) need to be written. Since WI2D and WDD use the CPU's store buffer, the last WI2D or WDD in a sequence will not get written out to RAMs until a subsequent store, SET_TAG/WI2T, or SET_TAG/WDT is executed. We HIGHLY recommend following all sequences of WI2D with SET_TAG/WIT and all sequences of WDD with SET_TAG/WDT as shown in the usage hints of this chapter.

WI2D/WDD will encode correct parity with the data word written to the cache, but this parity can/will be over-written by a subsequent SET_TAG and WI2T/WDT sequence if arbitrary parity is selected.

SET_TAG and WI2T/WDT will write the tag bits, tag parity, dirty bit, dirty parity into cache. They will also write the parity bit for the immediately preceding WI2D or WDD data word if arbitrary parity is selected. The address used by WI2T/WDT is a doubleword address. WI2T/WDT will encode correct parity for the tag and data if the "Arb enable" bit is false (See Figure 8.5). If the "Arb enable" bit is true, then the tag and data parity bits are taken from the corresponding bit positions in the cpu general register (dirty parity comes from the general register regardless of Arb enable). Since arbitrary parity can be written for the preceding WI2D or WDD, there are 2 data parity bits. If the WI2D/WDD address was even, Data Par 0 is written, else Data Par 1 is written. Perhaps it would be easiest to always duplicate the desired arbitrary data parity into both bits. The parity bits are active high (ie. To set a parity bit to '1', the corresponding GR bit should be '1').

In normal operation Tag parity is encoded over the twenty RPN bits, Dirty parity is encoded over the one Dirty bit, Data[0] parity is encoded over the most significant word of D-Cache data, and Data[1] parity is encoded over the least significant word D-Cache data. "Correct" parity means even parity for data and tags of the off-chip cache RAMs, and is odd parity for the dirty bit.

Cache Parity Errors signalled during Diagnose Write Off-Chip RAM instructions WILL also cause HPMCs unless the PSW-M bit is set or bit 9 or 11 of CPU Diagnose register #0 is set (as explained in the Diagnose Read section).

RESTRICTION: SET_TAG must be immediately followed by WI2T or WDT, with no traps occurring between them.

RESTRICTION: For a diagnose tag write (SET_TAG/WI2T or SET_TAG/WDT) to work properly, it should be preceded by a WI2D or WDD to the same doubleword address with no traps, loads, stores, flushes, or other diagnose instructions in between them.

RESTRICTION: A "sync" instruction should precede a sequence of diagnose read/write off-chip cache instructions to separate them from any preceding loads, stores, or flushes.

8.3.6 PA7100LC off-chip cache diagnose usage hints

```
; ----------------------------------------------------------------------------
; Off-Chip Cache Read / Write RAMs
; b - base reg for reads or writes
; t - target reg for reads
; x - source reg for writes
; RDTLB uses a virtual address, all others use a real address
; RDTLB can be used to test the on-chip UTLB
; ----------------------------------------------------------------------------
; To read a tag from Level2 cache, use RDT or RI2T
; To read a 32-bit data value from Level2 cache, issue 1 RDD or RI2D
; To read a 64-bit data value from L2 cache,
; issue 2 RDD or 2 RI2D instructions with appropriate "word addresses"
;
; To write a value to Level2 cache, it is STRONGLY recommended to
```

```
; write both a data value and a tag value.  The correct sequence is
; WDD, SET_TAG, WDT.  The WDD instruction sets a 32-bit data value and
; cache index into the store buffer, The SET_TAG instruction sets the
; supplied tag value into a "store buffer-like" register.  The WDT
; instruction forces the store buffer and "store buffer-like" register
; to be written to the data cache.  Note that if the cache index for
; WDD and WDT do not match, you will get some pretty wild (random)
; results!
;
; When using SET_TAG in the above sequence, the user has the option of
; enabling encoded or arbitrary parity.  If arbitrary parity is enabled, then
; the parity written for the tag and dirty bit AND the immediately
; preceding data WORD is written to cache along with the tag and data.
; If arbitrary parity is not enabled, then correct parity is encoded
; by the hardware for the tag/data words (dirty parity is never encoded
; by the hardware for SET_TAG).
;
; To write a 32-bit data value and the tag, use WDD,  SET_TAG, WDT
;                                          or  WI2D, SET_TAG, WI2T
;
; When testing the cache RAMs, it is possible to have multiple WDD or RDD
; instructions in a row.  Thus, you could write a word, double-word, or
; entire cache line (8 words), before writing the tag with SET_TAG/WDT.
; ie. testing a doubleword.....
;    addi 0,0,r1
;    WDD(r1, r10)
;    addi 4,r1,r1
;    WDD(r1, r11)
;    SET_TAG(r20)
;    WDT(r1)
;    addi 0,0,r1
;    RDD(r1, r10)
;    addi 4,r1,r1
;    RDD(r1, r11)
;    RDT(r1, r20)
;    <check the data values as appropriate>
;
; Note that in this double-word example, arbitrary parity could be written
; only for the odd-data-word and the tag.  The even data word would always
; have good (encoded) parity.
; ---------------------------------------------------------------------------
;
```

8.3.7 Diagnose Read/Write On-chip (Level1) Cache RAMs

There are two instructions of this type:

- Diagnose Read Level1 (on-chip) Instruction Cache (RI1)
- Diagnose Write Level1 (on-chip) Instruction Cache (WI1)

0 5	6 10	11 15	16 17	18 26	27 31
05	*b*	*0*	*0*	*code*	*0*

Figure 8.7: Diagnose Read/Write Level1 ICache RAMs Format

```
RI1     b=0 (address must be in IIAOQR)   code=40
WI1     b=address_base                    code=70
```

These instructions read or write diagnose registers 27,28,29 to or from the Level1 (on-chip) instruction cache. To read or write the Level1 instruction cache, software must first disable it by writing a "0" to the L1ICACHE_EN bit in diagnose register 0 with a MTDIAG instruction. Then to read the Level1 ICache, software must execute a RI1 instruction to get the data into the diagnose registers, then execute MFDIAG_T instructions to load any/all of diagnose registers 27,28,29 into general registers. To write the Level1 cache, software must first execute 3 MTDIAG instructions to load each of the diagnose registers 27,28,29 and then software must execute a WI1 instruction to write all 3 diagnose registers simultaneously into the Level1 ICache. The Level1 ICache computes even-parity for both parity trees (data and tag). See the Diagnose Register section to see the format of registers 27,28,29.

For RI1, the address used to index the Level1 cache is the address found in IIAOQR (control register 18). Since hardware automatically updates CR18 when PSW-Q=1, this implies that PSW-Q must be 0 before executing "MTCTL x,18" and "RI1". For WI1, the address used to index the Level1 cache is contained in GR[b]. Since the architected page size is 4 Kbytes and the Level1 ICache is only 1 KBytes, the generated address can always be considered as a "real" address. No address hashing logic exists on the Level1 ICache.

The on-chip instruction cache stores an RPN, steering bits, etc for every doubleword. Thus, it has a 2-word line size. Level1 ICache misses and Level2 to Level1 prefetching occurs on a 2-word basis. But for FIC/FICE we flush an 8-word line from both on-chip and off-chip ICache. Therefore, the on-chip ICache is always a subset of the off-chip ICache, and PDC can report a 32-byte ICache line size.

CPU Diagnose Register #0 latches the On-chip Cache Hit and Parity Error signals into the HIT and PARERR bits for the diagnose read access. The CPU will never service a Cache Miss which is signaled during an RI1 instruction. DR0 also updates the L1IHPMC bit for RI1/WI1 instructions. Therefore, software should set the L1IHPMC_DIS bit in DR0 to prevent HPMC's while testing the on-chip ICache.

RESTRICTION: A "sync" instruction should precede a diagnose reads and writes of the L1 cache to them from any preceding loads, stores, or flushes.

8.3.8 PA7100LC on-chip cache diagnose usage hints

```
; ---------------------------------------------------------------------------
; On-Chip Instruction Cache Read / Write RAMs
; b - base reg for reads or writes
; For write, the address is specified in register GR[b]
; For reads, PSW-Q must be 0 and the address must first be
;    placed in IIAOQR (control reg 18)
; The data must be read/written to/from 3 separate diagnose registers
;    DR27 - even (high) data word
;    DR28 - odd (low) data word
;    DR29 - tag [RPN, parity, predecode, quarter page index] (see ERS)
; ---------------------------------------------------------------------------
; all Level1 cache diagnose read/writes must occur with the ICACHE_EN bit
; in CPU DR0 cleared so the Level1 cache is disabled from normal reads/writes
; To read  an entry from Level1 cache,
;    use MTCTL, RI1, MFDIAG27, MFDIAG28, MFDIAG29
; To write an entry from Level1 cache,
;    use MTDIAG27, MTDIAG28, MTDIAG29, WI1
;
```

8.3.9 PA7100LC diagnose instruction encodings

Instruction	hex	Opcode Extension binary					
	19:26	19	20:22	23	24	25	26
MTCPU	12	X	001	X	X	1	X
MFCPU_T	a0	1	010	X	X	0	X
MFCPU_C	30	0	011	X	X	0	X
TOC_EN	50	0	101	X	X	0	X
TOC_DIS	52	0	101	X	X	1	X
SHDW_GR	d0	1	101	X	X	0	X
GR_SHDW	d2	1	101	X	X	1	X
RDD	e2	1	110	0	0	1	X
RDTLB	e6	1	110	0	1	1	X
RDT	ea	1	110	1	0	1	X
RI2D	62	0	110	0	0	1	X
RI2T	6a	0	110	1	0	1	X
RI1	40	0	100	0	0	0	X
WDD	f2	1	111	0	0	1	X
WDT	fa	1	111	1	0	1	X
WI2D	72	0	111	0	0	1	X
WI2T	7a	0	111	1	0	1	X
WI1	70	0	111	0	0	0	X
SET_TAG	42	0	100	X	0	1	X

```
bit 19: 0=instr         1=data
bit 20: 0=no_diag_hang  1=diag_hang
bit 22: 0=write         1=read
bit 23: 0=data          1=tag
bit 24: 0=real          1=virtual
bit 25: 0=on_chip_cache 1=off_chip_cache

X indicates reserved bit (assume X==0).
```

```
;Engineers in ESL have set up a macro file for diagnose instructions for PA7100LC.
;Users are STRONGLY encouraged to use this macro file when coding diagnose
;instructions for PA7100LC.  Contact us for an electronic copy of this file.
;We also have macros for cache hint, and the implementation specific
;instructions.
```

8.4 Software Constraints

This section lists software restrictions which were not mentioned in earlier sections or which are repeated in case you missed it earlier. This is not a complete list of restrictions however. PLEASE READ THIS.

- Diagnose Instructions do NOT need to come in PAIRS. Previous PA-RISC processors had this requirement, BUT PA7100LC DOES NOT! Each single diagnose instruction will be executed independently, and they CAN be nullified.
- Diagnose Instructions must not be immediately followed by RFI or RFIR.
- A "sync" instruction should precede a sequence of diagnose instructions (except for MTCPU, MFCPU_T, and MFCPU_C) to separate them from any preceding loads, stores, or flushes. (This is a change from the original spec that needed syncs only for off-chip cache diagnose.)
- A "sync" instruction should precede a MTCPU0 if the SOU_EN bit is being changed from 1 to 0 and there is any chance an outstanding DCache miss is still being serviced.
- The IHE (I-Cache Hash enable) bit cannot be changed while the PSW-C bit (code translation) is set.
- When executing R1I to read the Level1 instruction cache, the Level1 cache must first be disabled by clearing the L1ICACHE_EN bit in CPU diagnose register #0. It must also be guaranteed that the instruction immediately preceding the RI1 instruction is not a branch that branches to a new page.
- As mentioned in the Fault Tolerance section, the CPU does not guarantee that it will take an HPMC trap on the same instruction that caused it to be signaled. eg. A load instruction to a non-existent memory location will not trap on this instruction but will trap on a subsequent instruction. PDC self-test software which wishes to cause an "expected" HPMC, therefore needs to know how long to wait for HPMC traps to occur. The answer depends on many things: 1) I-fetch vs. D-Access, 2) Stall on Use Miss, and 3) whether the MIOC_HPMCH line is signalled during or after the transaction causing the error. Here is one way to check for an expected HPMC: Load instruction to non-existent memory location with the SOU_EN (stall-on-use enable) bit clear, followed by two "sync" instructions will guarantee that the HPMC will be taken on or before the next instruction (following the second sync).
- There are extra restrictions when software runs with code translation enabled and with the L1 cache disabled. After setting the L1 cache disable bit in DR0 and before RFI'ing to virtual mode, software must execute L1 diagnose write and read instructions to put coherent values (where the instructions match the predecoded steering bits) into the read portions of DR27, DR28, and DR29. If the L1 cache is so damaged that it is not possible

to write a coherent pattern and read it back successfully, then software must never return to virtual mode.

```
Example.
        MTCPU27(0)
        MTCPU28(0)
        IMM(0x18800000,gra)
        MTCPU29(gra)
        WI1(grb)
        RI1MTCTL(grb)
        MFCPU27(grx)
        MFCPU28(gry)
        MFCPU29(grz)
        (mask off bits of grz except steering bits, [2:9])
        combf,=,n        0,grx,halt
        combf,=,n        0,gry,halt
        combf,=,n        gra,grz,halt
```

- If the SOU_EN bit is 0 in DR0, then ldw/ldw bundles must also be disabled (ie. DR0[21:22] not equal to 00).
- Bit 20 of DR0 (ISTRM_EN) should be cleared before executing any diagnose instruction (except MTCPU, MFCPU_T, and MFCPU_C) from memory space. This restriction can be relaxed if software can guarantee that no DMA will occur or that no L2 Icache misses will occur within 8 instructions of the diagnose instructions.

Chapter 9

Fault Tolerance

9.1 Introduction

PA7100LC has error detection on all external cache SRAM's and on the on-chip instruction cache. Single bit errors in these caches are detected and cause an HPMC. Thus, it is up to software to determine the recoverability of these errors and to attempt any recovery. The TLB (and other on-chip circuitry) does not have parity circuitry and does not detect errors. The memory system (DRAM's) supports single-bit correct, double-bit detect Hamming codes. Therefore, single bit memory errors are automatically corrected by the hardware and signal LPMC for logging purposes, and double bit memory errors are detected and reported through HPMC.

All TOC's and ICache HPMC's should be recoverable, depending upon the IPSW Q bit being set. All Dcache and Memory HPMC's should be unrecoverable. Group 4 interruptions are never lost due to HPMC. See the rest of this chapter for more details of recoverability.

Note: After any error indication, software must clear the appropriate diagnose register error flags on the CPU (via Move To Diagnose Register) or the appropriate error flags in the MIOC Status Register (via I/O write) before doing an RFI from the HPMC or LPMC trap handler. For details of what information is saved in the diagnose registers, see the diagnose section in this ERS or the MIOC ERS.

9.2 On-chip Instruction Cache

Hardware

Hardware detects an error too late to stop the "errored" instruction from entering the pipeline. The CPU control will stop the errored instruction from affecting architected state and will take an HPMC on the instruction that caused the error or on a prior instruction. Note that HPMC's taken while the PSW Q bit is zero are unrecoverable, because the IIASQF/IIAOQF will not be updated.

Software

The HPMC handler will begin out of I/O space (address F0000000) and should turn off the L1 cache (using DIAGnose) before branching to memory space. The HPMC handler should fully test the L1 cache using DIAGnose instructions. If it finds only soft errors, it can invalidate all L1 cache entries, turn on the L1 cache, and RFI to normal operation. If the HPMC handler finds a hard error in the L1 cache, it has the option of keeping the L1 cache turned off, and returning to normal (but degraded performance) operation. Please see the restriction in the Software Constraints section of the Diagnose chapter that may disallow running in code virtual mode with the L1 cache disabled.

9.3 Off-chip Instruction Cache

Hardware

Hardware detects an error too late to stop the "errored" instruction from entering the pipeline. The CPU control will stop the errored instruction from affecting architected state and will take an HPMC on the instruction that caused the error or on a prior instruction. Note that HPMC's taken while the PSW Q bit is zero are unrecoverable, because the IIASQF/IIAOQF will not be updated.

Software

The HPMC handler will begin out of I/O space (address F0000000) and cannot branch to memory space without taking the risk of getting a nested HPMC. The HPMC handler should fully test the L2 instruction cache using DIAGnose instructions. If it finds only soft errors, it can invalidate all L2 cache entries and RFI to normal operation. If the HPMC handler finds a hard error in the L2 instruction cache, it can either bring the machine down or it can hide the L2 instruction cache error location by loading the appropriate location in the L1 instruction cache using DIAGnose instructions. Note that this second option will result in severly degraded performance since L1 misses that result in L2 access to the error location will result in an HPMC.

9.4 Off-chip Data Cache

Hardware

On systems without error correction codes, data cache errors on dirty data are always unrecoverable. In order to further reduce hardware complexity, PA7100LC treats all data cache errors as unrecoverable. This is the same strategy as PA7100 and earlier processors.

The error signal coming from the L2 cache arrives too late to stop the errored data from corrupting architected state. It also comes too late to guarantee that the CPU will take the HPMC instruction on the instruction that caused the error. The HPMC can occur on the offending instruction or on any of the next two instruction bundless in the pipeline.

Software

Since software has no way to regenerate dirty data that has an error, and since the IIASQF/IIAOQF are not guaranteed to allow retry of instructions with errors, there is no way for software to recover from an HPMC caused by a data cache error.

9.5 Memory Errors

Single bit memory errors are fully corrected by the MIOC and are reported to software via the LPMC handler. Double bit errors are not corrected and are reported via the HPMC handler. Software cannot recover from a double bit memory error. The MIOC Status Register contains information regarding the error, and the error indicator must be cleared before re-enabling HPMC or LPMC. See the MIOC ERS for details.

9.6 I/O Errors

I/O Errors causing an HPMC or LPMC on the GSC bus will be detected by the MIOC and will cause the CPU to vector to the appropriate trap handler. Software can poll the MIOC status register to determine that a GSC error has occurred. See the LASI and MIOC ERS's for more details.

9.7 Software Requirements

This section lists some requirements on PDC software for HPMC, TOC, and LPMC handling. The next section which describes PIM issues is related to this topic.

Requirements for HPMC:

- When an HPMC is signalled by the MIOC during a memory or I/O transaction (due to a miss) the cache line which was copied into may have bad data. Since the cache controller validates the line, this line must be purged (or flushed) before an RFI from the HPMC.
- The diagnose bit indicating the source of the error must be cleared before an RFI from the HPMC. The bits indicating the source of the error are in CPU diagnose register #0 or in an MIOC status register.

Requirements for LPMC:

- The diagnose bit indicating the source of the error must be cleared in the MIOC status register before an RFI from the LPMC (to avoid another LPMC for the same condition). The CPU has no LPMC indicators in DR0.

9.8 PIM Issues

This section will indicate how each of the PIM (Processor Internal Memory) bits are determined from Processor state.

CPU State Word

This includes the following bits

iqv : IIA queue Valid Always set.

iqf : IIA queue Failure Never set. The CPU cannot guarantee to trap on the instruction that caused the HPMC.

ipv : IPRs Valid Never set. The CPU cannot guarantee to trap on the instruction that caused the HPMC.

grv : GRs Valid Always set.

crv : CRs Valid Always set.

srv : SRs Valid Always set.

srv : TRs Valid Always set.

tl : Trap Lost Never set.

sis : Storage Integrity Syncronized. Always set.

cs : Check Severity Set to 0 (CHECK_CRITICAL) for DCache, Memory, and GSC errors. Set to 1,2 or 3 (CHECK_TRANSPARENT or CHECK_ISOLATED) for ICache errors or Transfer-of-Control (TOC) traps.

Cache Check Word

By polling CPU diagnose register #0 software can determine whether a cache HPMC was caused by the on-chip ICache, by the off-chip ICache, or by the off-chip DCache. Also, for off-chip Cache errors, the diagnose register contains 4-bits indicating whether the error occurred in the even-word, odd-word, tag-word, or dirty bit.

TLB Check Word

The TLB is on-chip and does not have parity detection circuitry. Therefore, this word should never be set as a result of HPMC or LPMC.

Bus Check Word

See MIOC ERS for a details concerning the Bus Check Word, Slave Address, and Master address.

9.8.1 Other Check Words

The Assist Check word, and Assist ID word bits are never set because there is never an HPMC or LPMC due to an assist.