SB-1 USER'S MANUAL

SB-1 USER'S MANUAL

CONFIDENTIAL

SEPTEMBER 2000

PRELIMINARY

REVISION 0.91

Table of Contents

List of Figures

List of Tables

CHAPTER 1 Introduction

1.1 Introduction

This document supplements MIPS64 Specification by MIPS Technologies Incorporated (MTI). The user should refer to the MIPS64 document for any specifics regarding the MIPS64 instruction set architecture. The aforementioned document provides a history of the MIPS64 architecture together with details on the CPU, FPU and coprocessor architectures, and the MIPS64 Privileged Resource Architecture.

This specification describes the actual implementation of SB-1 processor. The purpose of this document is to clarify any and all implementation-specific features listed in the MIPS64 Specification document.

The intended audience for this document is hardware designers and software engineers comprising compiler designers, operating system kernel writers, and application writers wishing to optimize software performance on the SB-1 processor.

1.2 Document Organization

This document is organized as follows:

Chapter 2 provides a general block diagram for the overall CPU and covers the functionality of the basic blocks.

Chapter 3 presents a general description of the ALU block together with an overview of the integer and load/store instructions and their latencies.

Chapter 4 delves into floating point architecture specifics and covers FP instructions, latencies, and restrictions for CP1 category of instructions. This chapter also covers MIPS-3D Application Specific Extension (ASE) category of instructions.

Chapter 5 deals with MIPS MDMX ASE and related issues.

Chapter 6 provides a basic description of SB-1 supported memory hierarchy, Caches, TLBs, Cache Operations, and Cache Coherency Attributes.

Chapter 7 covers the MIPS64 Address Space as implemented by SB-1 core.

Chapter 8 specifies a complete listing of the CP0 registers supported in SB-1 core.

Chapter 9 details the debug architecture for SB-1. SiByte-specific enhancements to the standard MIPS64 Debug architecture are described here.

Chapter 10 describes the Error handling capabilities of SB-1.

Chapter 11 addresses the performance monitoring architecture supported by SB-1. A complete listing of these features and their usage through CP0 architecture space registers is described in this chapter.

Chapter 12 provides a high level overview of the multiprocessing features supported by SB-1. Example code segments featuring the MP capabilities of the processor are provided here.

Chapter 13 provides a list of all implementation-specific features in the MIPS64 Specification and provides the SB-1 resolutions of these features.

1.3 Additional Documentation

The following documents are required as supplement to this specification.

TABLE 1-1 Supplemental Documents to SB-1 Users Manual

Document	**Description**	**Source**
MIPS64 Specification, Revision 1.0	Consolidated MIPS I, II, III, IV, and V ISA Specifications with a new Privileged Resource Architecture based on MIPS R4000 Processor	MIPS Technologies Incorporated
MIPS-3D ASE Specification, Revision 1.0	Describes 3D enhancements to the basic MIPS64 Architecture	MIPS Technologies Incorporated
MDMX Version 2.0 Specification, Revision 0.3.2	Describes MIPS Multimedia Extensions to the basis MIPS64 Architecture	MIPS Technologies Incorporated
MIPS RISC Architecture Volume I	Describes MIPS basic instructions in detail	MIPS Technologies Incorporated
MIPS RISC Architecture Volume II	Describes MIPS basic instructions in detail	MIPS Technologies Incorporated
MIPS Extended JTAG (EJTAG), Version 2.5	Describes MIPS EJTAG Specification	MIPS Technologies Incorporated

1.4 What is Missing or Incomplete in this Version of the Document?

The following Chapters need additional work. A future revision of this document will provide further details.

- Chapter 4: *The FPU (CP1) and MIPS-3D ASE Instructions.* The exception specific implementation details for the FPU unit need to be additionally elaborated upon.
- Chapter 5: *The MDMX ASE Instructions.* The implementation details of this unit are not finalized and are subject to change from their current description in Chapter 5.
- Chapter 11: *The Performance Monitor Architecture.* The implementation details of this unit are not finalized and are subject to change from their current description in Chapter 11.

CHAPTER 2 SB-1 Overview

2.1 Introduction

This chapter elaborates on high level features supported by SB-1 core. Further detail on specific SB-1 functionality is provided in subsequent chapters.

2.2 High Level Features

Table 2-1 shows the high level features supported by SB-1.

TABLE 2-1 **SB-1 High Level Specification**

FEATURE	SPECIFICATION
Instruction Set Architecture (ISA)	MIPS64 with SIMD Floating Point Functionality MIPS-3D ASE MDMX ASE
Frequency	600-1000 MHz
Issue Type	Quad Inorder
Pipe Architecture	Dual Enhanced-Skew Execute Dual Memory Support for 0-cycle load-to-use instruction sequences

TABLE 2-1 **SB-1 High Level Specification**

FEATURE	SPECIFICATION
Caches	Split I and D (Harvard Architecture)
Instruction	32K, 4-way, 32-Byte Lines, LRU Replacement Policy
Data	32K, 4-way, Non-blocking, 32-Byte Lines, LRU Replacement Policy
Branch Predictors	3x Structures
Direction Predictor	2-Level GShare, 4K-entry x 2-bit
Jump Register Cache	64-entry
Return Stack	8-entry
Peak OPS at 1000 MHz[a]	
Integer	4 per cycle, 4 BOPS
Floating Point	8 per cycle, 8 GFLOPS
Power Consumption at 1000 MHz	< 4 Watts
Dhrystone 2.1 MIPS at 1000 MHz	> 2000

a. MADD instruction is counted as 2 distinct operations: one multiply and one add

Figure 2-1 shows a simplified block diagram for SB-1 core. Subsequent sections provide additional detail with regard to the internals of SB-1.

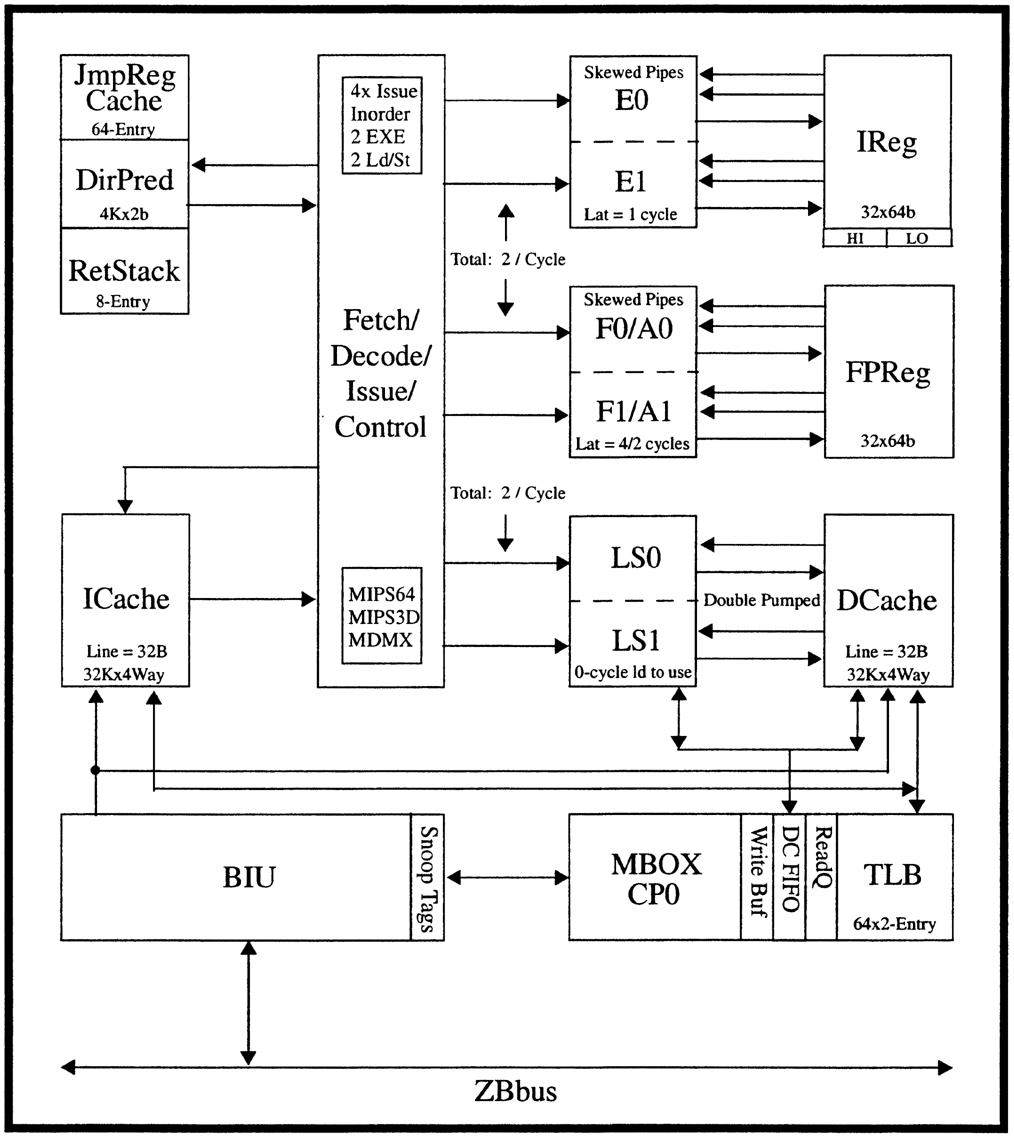

FIGURE 2-1 **Simplified Block Diagram of SB-1**

2.3 SB-1 Units

Internally, the SB-1 core comprises the PC Unit, the Issue Unit, the Execute Unit, the Load/Store Unit, the Floating Point Unit, the MDMX Unit, the Memory Unit, the Bus Interface Unit, and the Level One Instruction and Data Caches. The following sections briefly describe the functionality of each unit.

2.3.1 The PC Unit

The PC Unit performs the sequencing of the instruction fetch together with completing instructions and detecting exceptions. These functions are implemented via two subunits: the Fetch Unit and the Graduation Unit. The responsibility of the Fetch Unit is to predict the program flow and qualify fetched instructions. The Graduation Unit ensures that the instructions modify architected state in program order in light of branch mispredicts and instruction exceptions.

2.3.1.1 The Branch Unit

As shown in Figure 2-1, SB-1 supports three unique structures to aid program control flow in three distinct areas.

2.3.1.1.1 Two Level GShare, Branch Direction Predictor

This structure, with 4K x 2bit entries works in conjunction with a 9-bit Branch History Register (BHR). The contents of BHR are Exclusive-ORed with 11 PC bits to provide an index into the direction predictor table which uses the 2-bit counter scheme to predict branch outcome (taken vs. not-taken). There can be up to 2 predictions per cycle.

2.3.1.1.2 Return Stack (RS)

The eight entry processor-based Return Stack provides a mechanism to predict return addresses for subroutine calls. SB-1 supports an 8-entry Return Stack.

2.3.1.1.3 Jump Register Cache (JRC)

The Jump Register Cache is used to accelerate the execution of indirect branches through registers. SB-1 supports a 64-entry JRC. It provides a prediction mechanism for the target of indirect jumps through registers.

2.3.2 The Issue Unit (IBox)

The Issue Unit is responsible for issuing instructions to various functional units, and for tracking their progress until they can be handed off to the graduation part of the PC Unit. This unit examines and keeps track of all structural hazards as well as data dependencies in order to issue up to 4 instructions per cycle.

SB-1 supports decoupled front and back ends; the machine can continue processing instructions on instruction cache misses. An instruction queue (IQ) buffers instructions as they are fetched from memory.

2.3.3 The Execute Unit (E0, E1)

The Execute Unit is responsible for execution of ALU, Shift, and Branch instructions in the MIPS64 ISA. This unit supports thirty-two 64-bit Integer registers and 64-bit HI/LO registers for multiplies. The execute unit supports dual 8-stage, fully pipelined, 1-cycle execution latency pipes with enhanced skewing to allow zero-cycle load-to-use sequences. Multiply and divide instructions take additional cycles to complete.

2.3.4 The Load/Store Unit (LS0, LS1)

The Load/Store Unit executes memory load and store operations supported by the MIPS64 ISA. The load/store unit supports dual 8-stage load/store pipelines with the ability to execute simple ALU instructions in one pipe. This reduces ALU to address generation penalty for load/store address computations.

2.3.5 The Floating Point Unit (F0, F1)

The Floating Point Unit executes MIPS64 floating point and MIPS-3D ASE categories of instructions. It is IEEE-754 compatible and has support for Single, Double, and Paired-Single data formats.

There are thirty-two 64-bit Floating Point Registers in the FP Unit. The unit supports dual 11-stage, fully pipelined, 4-cycle execution latency pipes with enhanced skewing to allow zero-cycle load-to-use sequences.

2.3.6 The MDMX Unit (A0, A1)

The MDMX unit implements MIPS MDMX instructions using the same registers as the Floating Point Unit. It supports thirty-two 64-bit Floating Point Registers. The unit supports dual 9-stage, fully pipelined, 2-cycle execution latency pipes with enhanced skewing to allow zero-cycle load-to-use sequences.

The MDMX unit has extended accumulator support, with 24 and 48-bit modes for 8 and 16-bit SIMD computations, respectively.

2.3.7 The Memory Unit (MBox)

The Memory Unit implements the memory management functionality, as outlined in the MIPS64 Privileged Resource Architecture. In particular, it supports Coprocessor 0 (CP0) functionality of TLB, CACHE, and SYNC category of instructions.

2.3.8 The Bus Interface Unit (BIU)

This unit provides the interface between the core and the external bus.

2.3.9 Level One Instruction and Data Caches

SB-1 supports a 32KB, 4-way set associative, virtually-indexed and virtually-tagged instruction cache and a 32KB, 4-way set associative, physically-indexed and physically-tagged data cache. This provides the processor with a sizable portion of fast, on-chip memory.

SB-1 has a non-blocking data cache with support for up to 8 outstanding cachelines.

2.4 SB-1 Specifics

The remaining chapters in this document provide further details on the specifics of the major units in SB-1 core.

CHAPTER 3 The CPU Instructions

3.1 Introduction

This chapter provides a general overview of MIPS64 CPU instructions, as supported in SB-1 implementation. The information provided here should be regarded as a supplement to MIPS64 Specification document.

3.2 List of Instructions

Table 3-1 through Table 3-9 provide the list of CPU category instructions supported in SB-1.

3.2.1 CPU Load, Store and Memory Control Instructions

TABLE 3-1 **CPU Load, Store, and Memory Control Instructions**

Mnemonic	**Instruction**
LB	Load Byte
LBU	Load Byte Unsigned
LH	Load Halfword
LHU	Load Halfword Unsigned

TABLE 3-1 CPU Load, Store, and Memory Control Instructions

Mnemonic	Instruction
LW	Load Word
LWL	Load Word Left
LWR	Load Word Right
SB	Store Byte
SH	Store Halfword
SW	Store Word
SWL	Store Word Left
SWR	Store Word Right
LL	Load Linked Word
SC	Store Conditional Word
SYNC	Synchronize Memory Operations
LD	Load Doubleword
LDL	Load Doubleword Left
LDR	Load Doubleword Right
LLD	Load Linked Doubleword
LWU	Load Word Unsigned
SCD	Store Conditional Doubleword
SD	Store Doubleword
SDL	Store Doubleword Left
SDR	Store Doubleword Right
PREF	Prefetch Memory Data
PREFX	Prefetch Memory Data Indexed

3.2.2 CPU Arithmetic Instructions

TABLE 3-2 CPU Arithmetic Instructions

Mnemonic	Instruction
ADD	Add Word
ADDI	Add Immediate Word
ADDIU	Add Immediate Unsigned Word
ADDU	Add Unsigned Word

TABLE 3-2 **CPU Arithmetic Instructions**

Mnemonic	Instruction
DIV	Divide Word
DIVU	Divide Unsigned Word
MULT	Multiply Word
MULTU	Multiply Unsigned Word
SLT	Set on Less Than
SLTI	Set on Less Than Immediate
SLTIU	Set on Less Than Immediate Unsigned
SLTU	Set on Less Than Unsigned
SUB	Subtract Word
SUBU	Subtract Unsigned Word
DADD	Add Doubleword
DADDI	Add Immediate Doubleword
DADDIU	Add Immediate Unsigned Doubleword
DADDU	Add Unsigned Doubleword
DDIV	Divide Doubleword
DDIVU	Divide Unsigned Doubleword
DMULT	Multiply Doubleword
DMULTU	Multiply Unsigned Doubleword
DSUB	Subtract Doubleword
DSUBU	Subtract Unsigned Doubleword

3.2.3 CPU Logical Instructions

TABLE 3-3 **CPU Logical Instructions**

Mnemonic	Instruction
AND	Logical AND
ANDI	Logical AND Immediate
LUI	Load Upper Immediate
NOR	Logical NOR
OR	Logical OR
ORI	Logical OR Immediate

TABLE 3-3 CPU Logical Instructions

Mnemonic	Instruction
XOR	Logical XOR
XORI	Logical XOR Immediate

3.2.4 CPU Move Instructions

TABLE 3-4 CPU Move Instructions

Mnemonic	Instruction
MFHI	Move from HI
MFLO	Move from LO
MTHI	Move to HI
MTLO	Move to LO
MOVF	Move Conditional on Floating Point False
MOVN	Move Conditional on Not Zero
MOVT	Move Conditional on Floating Point True
MOVZ	Move Conditional on Zero

3.2.5 CPU Shift Instruction

TABLE 3-5 CPU Shift Instructions

Mnemonic	Instructions
SLL	Shift Word Left Logical
SLLV	Shift Word Left Logical Variable
SRA	Shift Word Right Arithmetic
SRAV	Shift Word Right Arithmetic Variable
SRL	Shift Word Right Logical
SRLV	Shift Word Right Logical Variable
DSLL	Shift Doubleword Left Logical
DSLL32	Shift Doubleword Left Logical +32
DSLLV	Shift Doubleword Right Logical Variable
DSRA	Shift Doubleword Right Arithmetic

TABLE 3-5 CPU Shift Instructions

Mnemonic	Instructions
DSRA32	Shift Doubleword Right Arithmetic +32
DSRAV	Shift Doubleword Right Arithmetic Variable
DSRL	Shift Doubleword Right Logical
DSRL32	Shift Doubleword Right Logical +32
DSRLV	Shift Doubleword Right Logical Variable

3.2.6 CPU Branch and Jump Instructions

TABLE 3-6 CPU Branch and Jump Instructions

Mnemonic	Instructions
BEQ	Branch on Equal
BGEZ	Branch on Greater Than or Equal Zero
BGEZAL	Branch on Greater Than or Equal Zero and Link
BGTZ	Branch on Greater Than Zero
BLEZ	Branch on Less Than or Equal Zero
BLTZ	Branch on Less Than Zero
BLTZAL	Branch on Less Than Zero and Link
BNE	Branch on Not Equal
J	Jump
JAL	Jump and Link
JALR	Jump and Link Register
JR	Jump Register

3.2.7 CPU Trap Instructions

TABLE 3-7 CPU Trap Instructions

Mnemonic	Instruction
BREAK	Breakpoint
SYSCALL	System Call
TEQ	Trap if Equal

TABLE 3-7 CPU Trap Instructions

Mnemonic	Instruction
TEQI	Trap if Equal Immediate
TGE	Trap if Greater Than or Equal
TGEI	Trap if Greater Than or Equal Immediate
TGEIU	Trap if Greater Than or Equal Immediate Unsigned
TGEU	Trap if Greater Than or Equal Unsigned
TLT	Trap if Less Than
TLTI	Trap if Less Than Immediate
TLTIU	Trap if Less Than Immediate Unsigned
TLTU	Trap if Less Than Unsigned
TNE	Trap if Not Equal
TNEI	Trap if Not Equal Immediate

3.2.8 Obsolete Branch Instructions

Software is strongly encouraged to avoid use of the Branch Likely instructions, as they will be removed from a future revision of the MIPS64 Architecture.

TABLE 3-8 Obsolete Branch Instructions

Mnemonic	Instruction
BEQL	Branch on Equal Likely
BGEZALL	Branch on Greater Than or Equal Zero and Link Likely
BGEZL	Branch on Greater Than or Equal Zero Likely
BGTZL	Branch on Greater Than Zero Likely
BLEZL	Branch on Less Than or Equal Zero Likely
BLTZALL	Branch on Less Than Zero and Link Likely
BLTZL	Branch on Less Than Zero Likely
BNEL	Branch on Not Equal Likely

3.2.9 Embedded Application Instructions

TABLE 3-9 **Embedded Application Instructions**

Mnemonic	**Instruction**
CLO	Count Leading Ones in Word
CLZ	Count Leading Zeros in Word
DCLO	Count Leading Ones in Doubleword
DCLZ	Count Leading Zeros in Doubleword
MADD	Multiply and Add Word
MADDU	Multiply and Add Unsigned Word
MSUB	Multiply and Subtract Word
MSUBU	Multiply and Subtract Unsigned Word
MUL	Multiply Word to Register
SSNOP	Superscalar Inhibit NOP

3.3 Block and Pipeline Diagrams

The CPU block of SB-1 consists of a an Execute unit (EXE) and a Load/Store Unit (LS). Figure 3-1 shows the pipes in each unit.

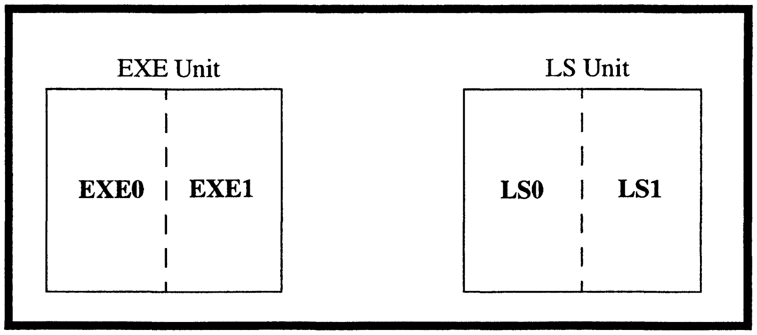

FIGURE 3-1 **EXE and LS Pipes in SB-1**

3.3.1 The EXE0 Unit

Figure 3-2 shows a block diagram of instruction execution flow in the EXE0 pipe.

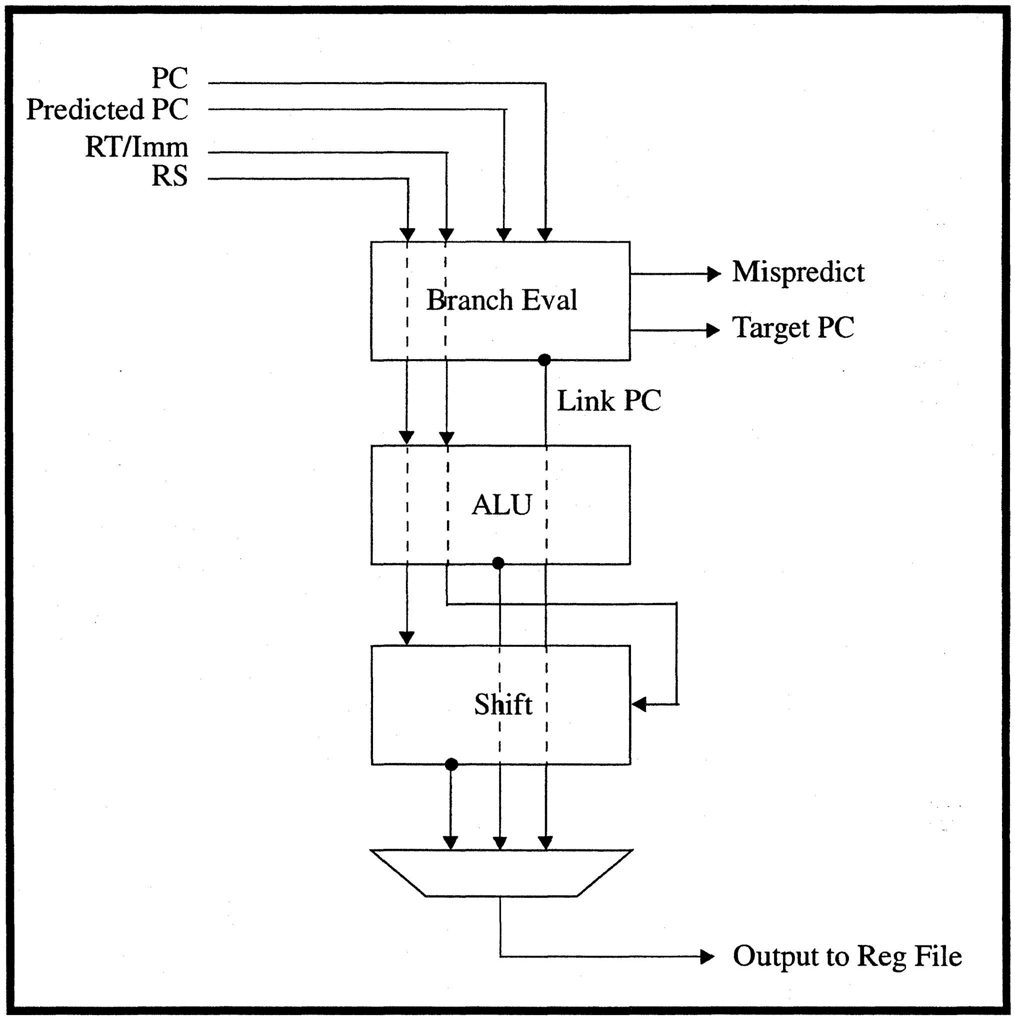

FIGURE 3-2 **EXE0 Block Diagram**

The List of instructions supported by EXE0 Unit is shown in Table 3-10.

TABLE 3-10 **Instructions Supported by the EXE0 Unit**

List of Instructions supported by EXE0 Unit
ADDs, SUBs, Logical Ops
Shifts
LUI
Branches/Jumps
CP1 Branches
Sets
Traps
CLZ/CLO
Conditional Moves
MOVT, MOVF
MOVZ, MOVN

The pipeline diagram for EXE0 unit is shown in Table 3-11.

TABLE 3-11 **EXE0 Pipe Stages in SB-1**

Stage	**F**	**D**	**I**	**1**	**2**	**3**[a]	**4**	**5**
ALU and Shift	Fetch	Decode	Issue	Skew1	Skew2	Read RF	Execute	Write RF
Branch Ops	Fetch	Decode	Issue	Skew1	Skew2	Read RF	Check Prediction -- signal redirect if mispredicted; Compute target and link addresses	Write Link Address to RF
CP1 Branch Ops	Fetch	Decode	Issue	Skew1	Skew2	Read RF Read FP CCs	Check Prediction -- signal redirect if mispredicted; Compute target and link addresses	Write Link Address to RF
MOVF, MOVT	Fetch	Decode	Issue	Skew1	Skew2	Read FP CCs	Evaluate Condition	Write RF
MOVZ, MOVN	Fetch	Decode	Issue	Skew1	Skew2	Read RF	Compare rt to zero, Signal result to FP Unit	

a. FP Condition Codes are not bypassed

The skewed slots in pipe stages 1 and 2 allow the coissuing of load/store and dependent EXE instructions in the same cycle.

3.3.2 The EXE1 Unit

The block diagram for EXE1 Unit is shown in Figure 3-3.

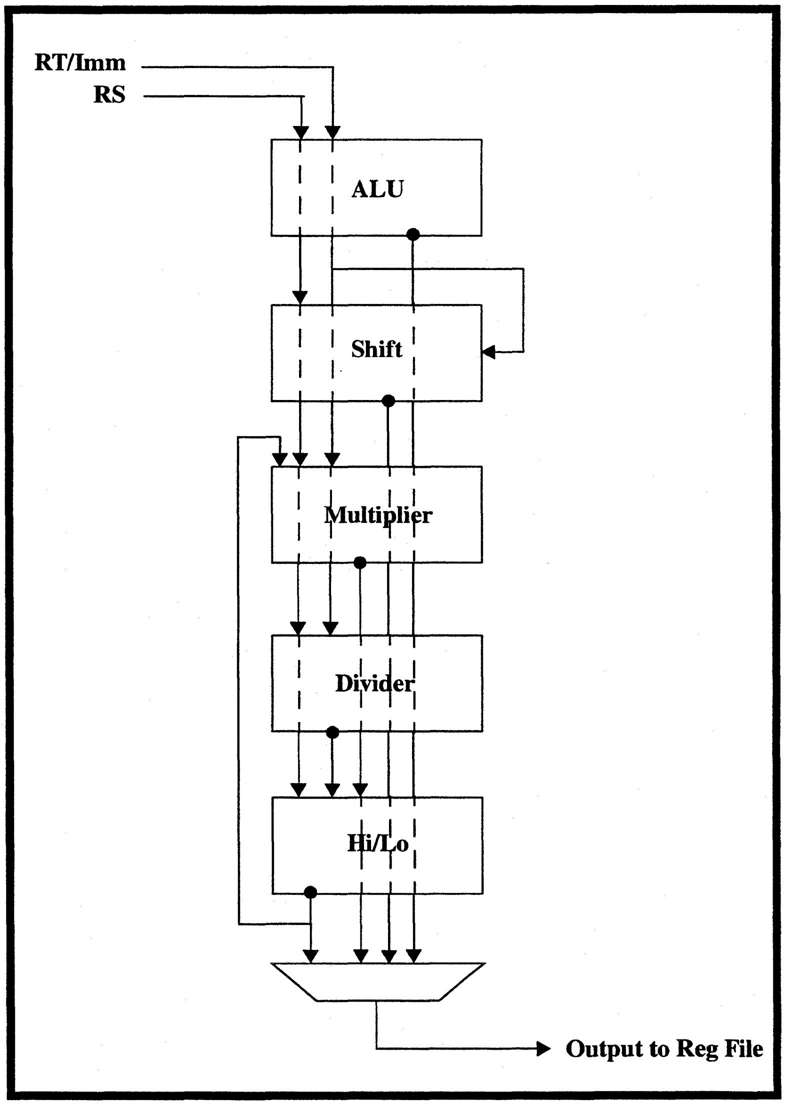

FIGURE 3-3 **EXE1 Block Diagram**

The List of instructions supported by EXE0 Unit is shown in Table 3-12.

TABLE 3-12 **Instructions Supported by the EXE1 Unit**

List of Instructions supported by EXE1 Unit
ADDs, SUBs, Logical Ops
Shifts
LUI
Conditional Moves
Multiplies
Divides
MT/MF HI/LO
MOVT, MOVF
MOVZ, MOVN

The pipeline diagram for EXE1 unit is shown in Table 3-13.

TABLE 3-13 **EXE1 Pipe Stages in SB-1 (All Except Divide)**

Stage	F	D	I	1	2	3[a]	4	5	6	7	8
Simple ALU Ops and Shifts	Fetch	Decode	Issue	Skew1	Skew2	Read RF	Execute	Write RF			
MULT	Fetch	Decode	Issue	Skew1	Skew2	Read RF	Execute1	Execute2	Execute3	Write HI/LO	
DMULT	Fetch	Decode	Issue	Skew1	Skew2	Read RF	Execute1	Execute2	Execute3	Execute4; Write LO	Write HI
MUL	Fetch	Decode	Issue	Skew1	Skew2	Read RF	Execute1	Execute2	Execute3	Write RF	
MADD/MSUB	Fetch	Decode	Issue	Skew1	Skew2	Read RF	Execute1	Execute2	Read HI/LO; Execute3	Write HI/LO	
MTHI/MTLO	Fetch	Decode	Issue	Skew1	Skew2	Read RF				Write HI/LO	
MFHI/MFLO	Fetch	Decode	Issue	Skew1	Skew2		Read HI/LO	Write RF			
MOVF, MOVT	Fetch	Decode	Issue	Skew1	Skew2	Read FP CCs	Evaluate Condition	Write RF			
MOVZ, MOVN	Fetch	Decode	Issue	Skew1	Skew2	Read RF	Compare rt to zero, Signal result to FP Unit				

a. FP Condition Codes are not bypassed

As in EXE0, the skewed slots in pipe stages 1 and 2 allow the coissuing of load/store and dependent EXE instructions in the same cycle.

Table 3-14 shows the pipeline stages for integer divide operations supported in EXE1 unit.

TABLE 3-14 **EXE1 Pipe Stages in SB-1 for Divide Instructions**

Stage	F	D	I	1	2	3	4	...	36/68	...	40/72
DIV/DIVU	Fetch	Decode	Issue	Skew1	Skew2	Read RF	Assert div_busy; Execute1	...	Execute33; Deassert div_busy	...	Write HI/LO
DDIV/DDIVU	Fetch	Decode	Issue	Skew1	Skew2	Read RF	Assert div_busy; Execute1	...	Execute65; Deassert div_busy	...	Write HI/LO

3.3.3 The LS0 Unit

The block diagram for LS0 Unit is shown in Figure 3-4.

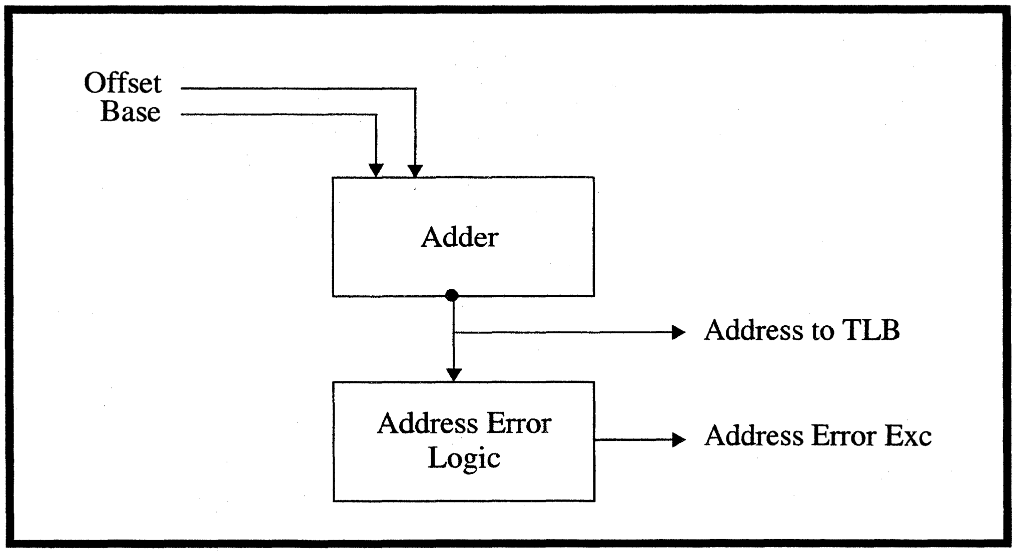

FIGURE 3-4 **LS0 Unit Block Diagram**

The List of instructions supported by LS0 Unit is shown in Table 3-15.

TABLE 3-15 **Instructions Supported by the LS0 Unit**

List of Instructions supported by LS0 Unit
Integer and Floating Point Loads and Stores

The pipeline diagram for LS0 unit is shown in Table 3-16. The diagram applies to both integer and floating point loads and stores.

TABLE 3-16 **LS0 Pipe Stages in SB-1**

Stage	**F**	**D**	**I**	**1**	**2**	**3**	**4**	**5**	**...**	**9**
Loads	Fetch	Decode	Read RF	Compute Address; Access TLB	Cache Tag Lookup, Cache Data Read and Way Select	-	Write RF			
Stores	Fetch	Decode	Read RF	Compute Address; Access TLB	PA Pushed into DCFIFO	-	-	Data Pushed into DCFIFO	...	Cache accessed after graduation of store instruction and availability of free slot in DCache

3.3.4 The LS1 Unit

The block diagram for LS1 Unit is shown in Figure 3-5.

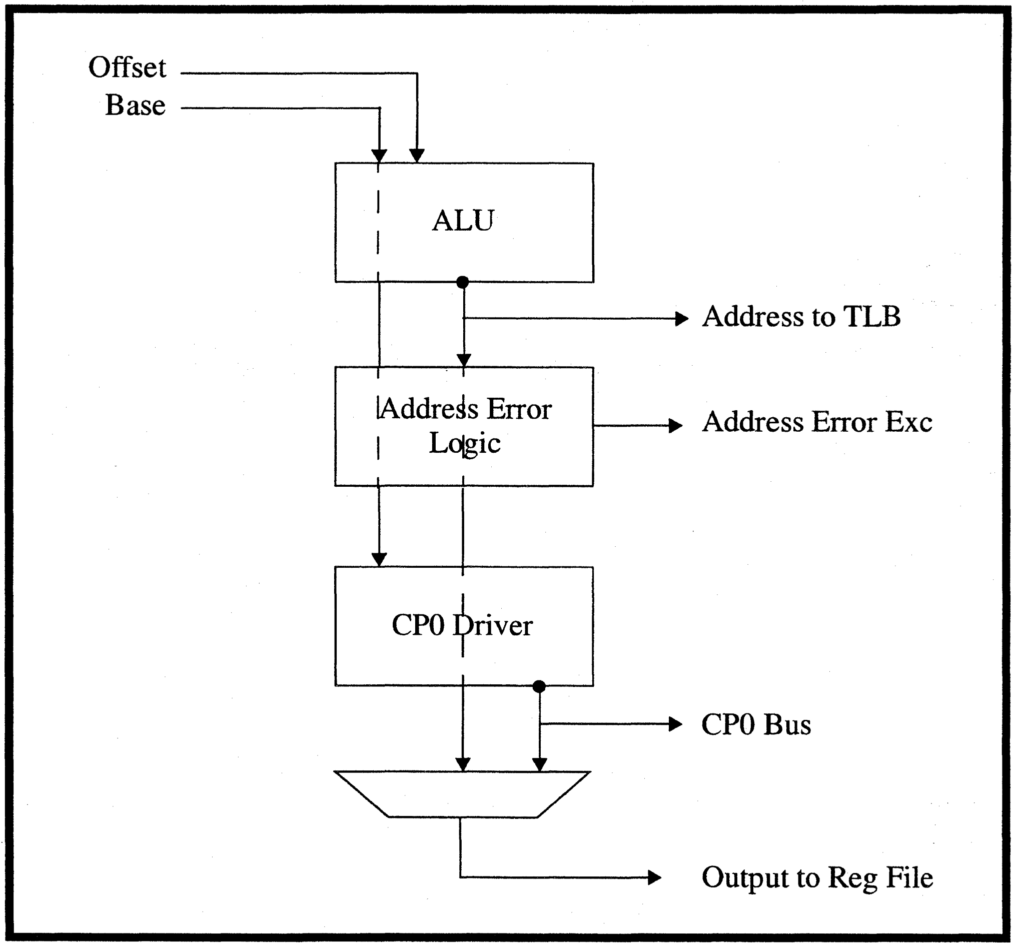

FIGURE 3-5 LS1 Block Diagram

The list of instructions supported by LS1 Unit is shown in Table 3-17.

TABLE 3-17 **Instructions Supported by the LS1 Unit**

List of Instructions supported by LS1 Unit
ADDs, SUBs, Logical Ops
LUI
Loads and Stores
Indexed Loads/Stores
TLB OPs
MT/MF CP0
Cache Ops

The pipeline diagram for LS1 unit is shown in Figure 3-18.

TABLE 3-18 **LS1 Pipe Stages in SB-1**

Stage	F	D	I	1	2	3	4	5	6	7	...	9
Loads	Fetch	Decode	Read RF	Compute Address	Access TLB, Cache Tag Lookup	Cache Data Read and Way Select	Write RF					
Stores	Fetch	Decode	Read RF	Compute Address	Access TLB; PA Pushed into DCFIFO			Data Pushed into DCFIFO			...	Cache accessed after graduation of store instruction and availability of free slot in DCache
ALU Ops	Fetch	Decode	Read RF	Execute	Pipe	Pipe	Write RF					
MF C0	Fetch	Decode		Drive Control	Drive CP0 Bus	Pipe	Write RF					
MT C0	Fetch	Decode	Read RF	Pipe	Pipe	Pipe	Pipe	Drive Control	Drive CP0 Bus	Write RF		
Cache Ops	Fetch	Decode	Read RF	Compute Address								

3.4 Instruction Latency and Throughput by Category of Instructions

Table 3-19 shows the latency and throughput by category for all instructions supported in the EXE and LS Units.

TABLE 3-19 Instruction Throughput and Latency for EXE and LS Units by Inst Category

Instruction Category	Latency	Throughput (1 Instruction/x cycles per supported pipe)	Co-issue w/ Dependent Op?
ALU ops to EXE Pipes	1	1	No
ALU ops to LS1 Pipe	1	1	Yes -- to EXE Pipes Only
Shifts	1	1	No
Branches	1	1	NA
32-bit Multiplies	3	1	No
64-bit Multiplies	3 to LO, 4 to HI	2	No
MTLO/MTHI	-	1	No
MFLO/MFHI	-	1	No
32-bit Divides	36	35	No
64-bit Divides	68	67	No
Load	-	1	Yes -- to EXE Pipes Only
Stores	-	1	No
MFC0	-	1	Yes -- to EXE Pipes Only
MTC0	-	1	No

3.5 Available Bypasses

Table 3-20 shows the available bypasses among EX0, EX1, LS0 and LS1 units.

TABLE 3-20 List of Available Bypasses in SB-1 Core for EX0, EX1, LS0, and LS1 Units

From/To	EX0	EX1	LS0	LS1
EX0	Yes	Yes		
EX1	Yes	Yes		
EX1 (MUL Inst Only)	Yes	Yes		
LS0 Load	Yes	Yes	Yes	Yes
LS1 Data	Yes	Yes	Yes	Yes
LS1 Load	Yes	Yes	Yes	Yes

3.6 Instruction Types Issued to each Pipe

Table 3-21 summarizes the types of instructions that can be issued to each one of EX0, EX1, LS0, and LS1 pipes.

TABLE 3-21 Instruction Types Issued to each Pipe

Instruction Type	EX0 Pipe	EX1 Pipe	LS0 Pipe	LS1 Pipe
ADDs, SUBs, Logical Ops	Yes	Yes		Yes
Shifts	Yes	Yes		
LUI	Yes	Yes		Yes
Branches/Jumps	Yes			
Sets	Yes			
Traps	Yes			
CLZ/CLO	Yes			
Integer MOVZ, MOVN	Yes	Yes		
FP MOVZ, MOVN	Yes	Yes		
MOVF, MOVT	Yes	Yes		
Multiplies		Yes		
Divides		Yes		
MT/MF HI/LO		Yes		
Loads/Stores			Yes	Yes
Indexed Loads/Stores				Yes
Cache OPs				Yes
TLB OPs				Yes
MT/MF CP0				Yes

3.7 Issue Rules and Restrictions

Table 3-22 identifies issue rules and restrictions for EXE and LS pipes. These restrictions are enforced by hardware interlocks.

TABLE 3-22 Instruction Issue Rules and Restrictions for CPU instructions

Instruction A	Instruction B	Restrictions
MUL	Any dependent op to EX0 or EX1	Dependent op can issue 3 cycles after MUL
MUL	Any dependent op to LS0 or LS1	Dependent op can issue 8 cycles after the MUL

TABLE 3-22 Instruction Issue Rules and Restrictions for CPU instructions

Instruction A	Instruction B	Restrictions
MULT, MADD, MSUB	MFLO, MFHI	MF* instruction can issue two cycles after the multiply
DMULT	MFLO	MFLO can issue two cycles after DMULT
DMULT	MFHI	MFHI can issue three cycles after DMULT
DMULT	Any Multiply	No multiply instructions may be issued in the cycle immediately following a DMULT
DIV, DDIV	DIV, DDIV	Another divide cannot be issued while there is a divide in the pipe
DIV, DDIV	MFLO, MFHI	HI/LO reads cannot be issued while there is a divide in the pipe
DIV, DDIV	Any multiply except MUL	Multiplies that write HI/LO cannot be issued while there is a divide in the pipe
Shift or ALU op to EXE pipes	Any dependent LS op	Dependent op can issue 4 cycles after the shift
ALU op to LS1 pipe	Dependent LS op	Dependent op can issue the next cycle after the ALU op; cannot co-issue
Load, Store, MFC0	Dependent LS op	Dependent op can issue 4 cycles after the LD/ST

3.8 Differences between 32 and 64-bit Modes of Operation

EXE Ops: In 32-bit mode, 64-bit instructions cause reserved instruction exceptions

LS/ST Ops: Address errors are generated based on the mode as specified in the MIPS64 Specification.

CHAPTER 4 The FPU (CP1) and MIPS-3D ASE Instructions

4.1 Introduction

This chapter provides a general overview of MIPS64 CP1 and MIPS-3D ASE (Application Specific Extension) instructions, as supported in SB-1. The information provided here should be regarded as a supplement to MIPS64 Specification and MIPS-3D ASE documents.

4.2 High Level Description of FP Block

Table 4-1 provides high level characteristics of the FP Units in SB-1.

TABLE 4-1 **FP Block Description**

Feature	Description
Number of FP Pipes	2: FP0 Pipe and FP1 Pipe
Symmetrical Pipes?	No (See detailed pipe descriptions)
Number of Stages per FP Pipe	11
FP Latency	4 for a majority of the instructions (See detailed pipe descriptions)
Fully Pipelined?	Yes for a majority of the instructions (See detailed pipe descriptions)

TABLE 4-1 **FP Block Description**

Feature	Description
Maximum Number of FP Operations per Cycle per Pipe	4 Single Precision FP Operations (2 Multiply Adds on Paired Single Operands per One Instruction)
Maximum Number of FP Operations per Cycle in SB-1	8 Single Precision FP Operations

4.3 Block and Pipeline Diagrams

Figure 4-1 shows a high level block diagram of the Floating Point Unit in SB-1.

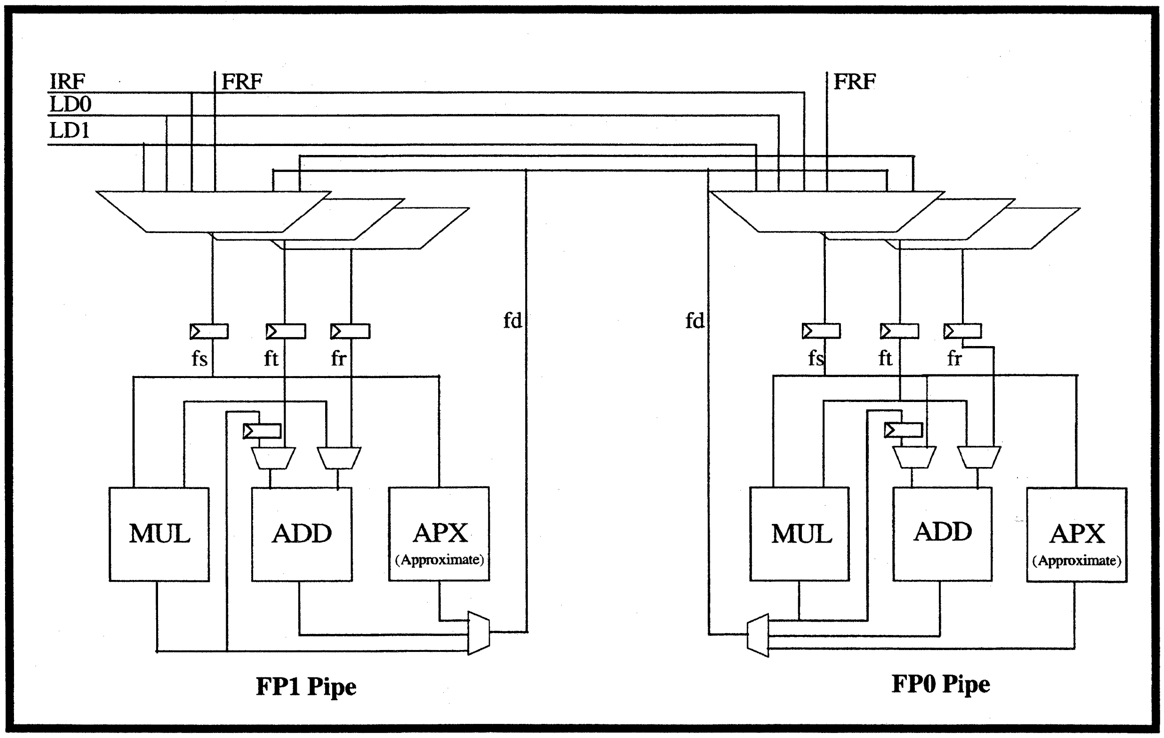

FIGURE 4-1 **Block Diagram of the Floating Point Unit**

Table 4-2 shows Floating Point pipe diagram for FP0 and FP1 pipes.

TABLE 4-2 **FP0 and FP1 Pipe Operation**

Stage	F	D	I	1	2	3	4	5	6	7	8
FP Ops	Fetch	Decode	Issue	S1	S2	Read RF	Execute1	Execute2	Execute3	Execute4	Write RF

4.4 Instruction Latency and Throughput by Category of Instructions

The following tables present the list of supported instructions in SB-1 Floating Point Unit and their associated latencies.

TABLE 4-3 **FPU Load/Store Instructions Supported in CPU Unit (Chapter 3)**

Instruction	Description
LWC1	Refer to Chapter 3
SWC1	Refer to Chapter 3
LDC1	Refer to Chapter 3
SDC1	Refer to Chapter 3
LDXC1	Refer to Chapter 3
LWXC1	Refer to Chapter 3
SDXC1	Refer to Chapter 3
SWXC1	Refer to Chapter 3
LUXC1	Refer to Chapter 3
SUXC1	Refer to Chapter 3

All instructions in Table 4-3 can be issued to LS1 pipe. The first four instructions can be additionally issued to LS0 pipe.

TABLE 4-4 **FPU Arithmetic Instructions**

Instruction	Supported Data Formats	Latency	Throughput (1 Instruction/x cycles per supported pipe)
ABS	Single, Double, Paired Single	4/4/4	1
ADD	Single, Double, Paired Single	4/4/4	1
C.cond	Single, Double, Paired Single	4/4/4	1
DIV	Single, Double, Paired Single	24/32/24	4 Insts/24 cycles (S), 4 Insts/32 cycles (D), 4 Insts/24 cycles (PS)
MUL	Single, Double, Paired Single	4/4/4	1

TABLE 4-4 FPU Arithmetic Instructions

Instruction	Supported Data Formats	Latency	Throughput (1 Instruction/x cycles per supported pipe)
NEG	Single, Double, Paired Single	4/4/4	1
SUB	Single, Double, Paired Single	4/4/4	1
SQRT	Single, Double, Paired Single	28/40/28	4 Insts/28 cycles (S), 4 Insts/40 cycles (D), 4 Insts/28 cycles (PS)
MADD	Single, Double, Paired Single	8/8/8	1
MSUB	Single, Double, Paired Single	8/8/8	1
NMADD	Single, Double, Paired Single	8/8/8	1
NMSUB	Single, Double, Paired Single	8/8/8	1
RECIP	Single, Double, Paired Single	12/20/12	4 Insts/12 cycles (S), 4 Insts/20 cycles (D), 4 Insts/12 cycles (PS)
RSQRT	Single, Double, Paired Single	16/28/16	4 Insts/16 cycles (S), 4 Insts/28 cycles (D), 4 Insts/16 cycles (PS)

TABLE 4-5 FPU Move Instructions

Instruction	Supported Data Formats	Latency	Throughput (1 Instruction/x cycles per supported pipe)
CFC1	-	1	1
CTC1	-	4	1
MFC1	-	1	1
MTC1	-	4	1
DMFC1	-	1	1
DMTC1	-	4	1
MOV	Single, Double, Paired Single	4	1
MOVF	Single, Double, Paired Single	4	1
MOVN	Single, Double	4	1
MOVT	Single, Double, Paired Single	4	1
MOVZ	Single, Double	4	1

TABLE 4-6 FPU Convert Instructions

Instruction	Supported Data Formats	Latency	Throughput (1 Instruction/x cycles per supported pipe)
CVT.D	Single, Word, Long	4	1
CVT.L	Single, Double	4	1
CVT.PS	Single, (Paired Word)	4	1
CVT.S	Word, Double, Long, Paired Lower, Paired Upper	4	1
CVT.W	Single, Double	4	1
CEIL.W	Single, Double	4	1
CEIL.L	Single, Double	4	1
FLOOR.W	Single, Double	4	1
FLOOR.L	Single, Double	4	1
ROUND.W	Single, Double	4	1
ROUND.L	Single, Double	4	1
TRUNC.W	Single, Double	4	1
TRUNC.L	Single, Double	4	1
ALNV	Paired Single	4	1
PLL	Paired Single	4	1
PLU	Paired Single	4	1
PUL	Paired Single	4	1
PUU	Paired Single	4	1

TABLE 4-7 FPU Branch Instructions

Instruction	Latency
BC1F	Refer to Chapter 3
BC1T	Refer to Chapter 3

TABLE 4-8 **Obsolete[a] FPU Branch Instructions**

Instruction	Supported Data Formats
BC1FL	Refer to Chapter 3
BC1TL	Refer to Chapter 3

a. Software is strongly encouraged to avoid use of the Branch Likely instructions, as they will be removed from a future revision of the MIPS64 architecture.

4.5 MIPS-3D ASE Instructions

Table 4-9 lists the MIPS-3D ASE instructions supported in SB-1. The execution of these instructions is supported through the Floating Point Unit.

TABLE 4-9 **MIPS-3D Instructions in the SB-1 Core**

Instruction	Supported Data Formats	Latency	Throughput (1 Instruction/x cycles per supported pipe)
ADDR	Paired Single	4	1
MULR	Paired Single	4	1
RECIP1	Single, Double, Paired Single	4	1
RECIP2[a]	Single, Double, Paired Single	8	1
RSQRT1	Single, Double, Paired Single	4	1
RSQRT2	Single, Double, Paired Single	8	1
CVT.PS	(Single), Paired Word	4	1
CVT.PW	Paired Single	4	1
CABS	Single, Double, Paired Single	4	1
BC1ANY2F	-	-	-
BC1ANY2T	-	-	-
BC1ANY4F	-	-	-
BC1ANY4T	-	-	-

a. RECIP2 is implemented as nmsub fd, 1, fs, ft

4.6 Available Bypasses

Table 4-10 shows the available bypasses among Floating Point, Load/Store and Integer Register File units.

TABLE 4-10 **List of Available Bypasses in SB-1 Core for EX0, EX1, LS0, and LS1 Units**

From/To	FP0	FP1
FP0	Yes	Yes
FP1	Yes	Yes
LS0 Load	Yes	Yes
LS1 Load	Yes	Yes
Integer Register File	Yes	Yes

4.7 Differences between the Pipes

Table 4-11 summarizes the types of instructions that can be issued to each one of FP0 and FP1 floating point pipes. The two pipes are symmetrical for most regular floating point operations, but the majority of MIPS-3D instructions can be issued to FP1 pipe only.

TABLE 4-11 **Instruction Types Issued to each Pipe**

Instruction Type	FP0 Pipe	FP1 Pipe
All Except Below	Yes	Yes
C.cond	Yes	No
DIV	Yes	Yes
RECIP	Yes	Yes
RSQRT	Yes	Yes
SQRT	Yes	Yes
RECIP1	Yes	Yes
RECIP2	Yes	Yes
RSQRT1	Yes	Yes
RSQRT2	Yes	Yes
CABS	Yes	No

4.8 Issue Rules and Restrictions

Table 4-12 identifies the issue rules and restrictions for floating point instructions.

TABLE 4-12 Issue Rules and Restrictions for Floating Point Instructions

Instruction A	Instruction B	Restrictions
All Except below	All Except below	Dependent op can issue 4 cycles after instruction
RECIP	Any dependent op	Dependent op can issue 9 cycles after for Single Precision and Paired Singles and 15 cycles after for Double Precision
RSQRT	Any dependent op	Dependent op can issue 12 cycles after for Single Precision or Paired Singles and 21 cycles after for Double Precision
DIV	Any dependent op	Dependent op can issue 18 cycles after for Single Precision and Paired Singles and 24 cycles after for Double Precision
SQRT	Any dependent op	Dependent op can issue 28 cycles after for Single Precision and Paired Singles and 40 cycles after for Double Precision
MADD, MSUB, NMADD, NMSUB, RECIP2, RSQRT2	Any dependent op	Dependent op can issue 8 cycles after for Single Precision, Double Precision, and Paired Singles (unless accumulator of MADD, See Section 4.9.1).

4.9 Implementation Details on Special Instructions

The next sections comment on SB-1 specific implementation details with regard to a few FP instructions.

4.9.1 MADD, MSUB, NMADD, NMSUB

```
OPERATION  fd, fr, fs, ft;    fd <-fs * ft +/- fr
```

This group of instructions is implemented as an IEEE rounded multiply followed by an IEEE rounded add, all with an 8-cycle latency. Operand fr is read 4 cycles after operands fs and ft. It can also be sourced from a bypass. These instructions behave like a separately issued MUL followed by an ADD. Exception flags of both MUL and ADD are ORed and stored in the FCSR.

An operation that accumulates the result of several multiplies is executed with 3 bubbles between subsequent ops. To avoid the bubbles, it is recommended to process up to 4 different multiply-accumulate type operations in parallel. An example follows:

```
MADD f0, f0, f1, f2        (f0 = f0 + f1 * f2)
3 bubbles (nops)
MADD f0, f0, f4, f5        (f0 = f0 + f4 * f5)
3 bubbles (nops)
```

```
MADD f0, f0, f6, f7        (f0 = f0 + f6 * f7)
```

The above sequence can be optimized by interleaving four independent streams as such:

```
MADD f0,  f0,  f1,  f2     Stream 1
MADD f8,  f8,  f9,  f10    Stream 2
MADD f16, f16, f17, f18    Stream 3
MADD f24, f24, f25, f26    Stream 4

MADD f0,  f0,  f4,  f5     Stream 1
MADD f8,  f8,  f11, f12    Stream 2
MADD f16, f16, f19, f20    Stream 3
MADD f24, f24, f27, f28    Stream 4

MADD f0,  f0,  f6,  f7     Stream 1
MADD f8,  f8,  f13, f14    Stream 2
MADD f16, f16, f21, f22    Stream 3
MADD f24, f24, f29, f30    Stream 4
```

These instructions are fully pipelined, i.e., each pipe can absorb a multiply-add type operation every cycle.

4.9.2 DIV Operation

```
DIV  fd, fs, ft;    fd ←fs / ft
```

In SB-1, this operation is implemented using RECIP.fmt, MUL.fmt, and a rounding step to obtain the correctly rounded IEEE result. DIV operations with exponent ft = 254, 253 for single precision and exponent of ft = 2047, 2046 for double precision computes are not implemented and will cause an unimplemented exception.

If rounding precision is not required, this instruction can be implemented using RECIP and MUL instructions. This saves the rounding step which take 8 cycles to execute.

Hence the sequence

```
RECIP.fmt  f1, f2
MUL.fmt    f1, f1, f3
```

has 8 fewer cycles than

```
DIV.fmt    f1, f3, f2
```

4.9.3 SQRT Operation

```
        SQRT  fd, fs, ft;    fd ←sqrt(fs)
```

In SB-1, this operation is implemented using RSQRT.fmt, MUL.fmt, and a rounding step to obtain the correctly rounded IEEE result.

If rounding precision is not required, this instruction can be implemented using RECIP and MUL instructions. This saves the rounding step which take 8 cycles to execute.

Hence the sequence

```
        RSQRT.fmt  f1, f2
        MUL.fmt    f1, f1, f2
```

has 8 fewer cycles than

```
        SQRT.fmt    f1, f2
```

4.9.4 RECIP1 and RSQRT1 Operations

RECIP1 computes an approximation of 1/x and RSQRT1 computes an approximation of 1/sqrt(x), both with at least 14 bits of precision for Single, Double and Paired Single operands.

For further detail on these operations, refer to MIPS-3D Specifications.

4.9.5 RECIP2

RECIP2 computes -(a * b - 1) for any number in Single, Double, and Paired Single format and is implemented using NMSUB fd, 1, fs, ft operation.

4.9.6 RSQRT2

RSQRT2 computes -(a * b - 1) / 2 for any number in Single, Double, and Paired Single format and is implemented using NMSUB fd, 1, fs, ft operation with divide by 2 factored in at the end.

4.10 Supplemental FP Instruction in SB-1

This section describes the supplemental floating point instructions supported in SB-1.

Floating Point Divide **DIV.fmt**

31 26	25 21	20 16	15 11	10 6	5 0
COP1 010001	fmt	ft	fs	fd	DIV 000011
6	5	5	5	5	6

FIGURE 4-2 **DIV Format**

Format:

```
DIV.S   fd, fs, ft
DIV.D   fd, fs, ft
DIV.PS  fd, fs, ft              SB-1  Addition
```

Purpose: To divide FP values

Description: `fd ← fs / ft`

The value in FPR fs is divided by the value in FPR ft. The result is calculated to infinite precision, rounded according to the current rounding mode in FCSR, and placed into FPR fd. The operands and result are values in format fmt.

Restrictions:

The fields fs, ft, and fd must specify FPRs valid for operands of type fmt; if they are not valid, the result is undefined.

The operands must be values in format fmt; if they are not, the result is undefined and the value of the operand FPRs becomes undefined.

Unimplemented Exceptions for exponent of ft = 254, 253 for DIV.S and DIV.PS and for exponent of ft = 2047, 2046 for DIV.D

Operation:

```
StoreFPR (fd, fmt, ValueFPR(fs, fmt) / ValueFPR(ft, fmt))
```

Exceptions: Coprocessor Unusable, Reserved Instruction

Floating Point Exceptions: Inexact, Invalid Operation, Unimplemented Operation, Division-by-zero, Overflow, Underflow

Reciprocal Approximation **RECIP.fmt**

31 26	25 21	20 16	15 11	10 6	5 0
COP1 010001	fmt	0	fs	fd	RECIP 010101
6	5	5	5	5	6

FIGURE 4-3 **RECIP Format**

Format:

```
RECIP.S    fd, fs
RECIP.D    fd, fs
RECIP.PS   fd, fs          SB-1 Addition
```

Purpose: To approximate the reciprocal of an FP value (quickly)

Description: `fd ←1.0 / fs`

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

The numeric accuracy of this operation does not meet the accuracy specified by the IEEE 754 Floating Point standard. The computed result differs from both the exact result and the IEEE-mandated representation of the exact result by no more than one unit in the least-significant place (ULP).

The result is not affected by the current rounding mode in FCSR.

Restrictions:

The fields fs and fd must specify FPRs valid for operands of type fmt; if they are not valid, the result is undefined.

The operand must be a value in format fmt; if it is not, the result is undefined and the value of the operand FPR becomes undefined.

Operation:

```
StoreFPR(fd, fmt, 1.0 / ValueFPR(fs, fmt))
```

Exceptions: Coprocessor Unusable, Reserved Instruction

Floating Point Exceptions: Inexact, Invalid Operation, Unimplemented Operation, Division-by-zero, Overflow, Underflow

Reciprocal Square Root Approximation RSQRT.fmt

31 26	25 21	20 16	15 11	10 6	5 0
COP1 010001	fmt	0	fs	fd	RSQRT 010110
6	5	5	5	5	6

FIGURE 4-4 **RSQRT Format**

Format:

```
RSQRT.S   fd, fs
RSQRT.D  fd, fs
```
`RSQRT.PS  fd, fs          SB-1 Addition`

Purpose: To approximate the reciprocal square root of an FP value (quickly)

Description: `fd ←1.0 / SQRT(fs)`

The reciprocal of the positive square root of the value in FPR fs is approximated and placed into FPR fd. The operand and result are values in format fmt.

The numeric accuracy of this operation does not meet the accuracy specified by the IEEE 754 Floating Point standard. The computed result differs from both the exact result and the IEEE-mandated representation of the exact result by no more than two units in the least-significant place (ULP).

The result is not affected by the current rounding mode in FCSR.

Restrictions:

The fields fs and fd must specify FPRs valid for operands of type fmt; if they are not valid, the result is undefined.

The operand must be a value in format fmt; if it is not, the result is undefined and the value of the operand FPR becomes undefined.

Operation:

```
StoreFPR(fd, fmt, 1.0 / SquareRoot(ValueFPR(fs, fmt)))
```

Exceptions: Coprocessor Unusable, Reserved Instruction

Floating Point Exceptions: Inexact, Invalid Operation, Unimplemented Operation, Division-by-zero, Overflow, Underflow

Floating Point Square Root **SQRT.fmt**

31 26	25 21	20 16	15 11	10 6	5 0
COP1 010001	fmt	0	fs	fd	SQRT 000100
6	5	5	5	5	6

FIGURE 4-5 **SQRT Format**

Format:

```
SQRT.S   fd, fs
SQRT.D   fd, fs
SQRT.PS  fd, fs          SB-1 Addition
```

Purpose: To compute the square root of an FP value

Description: `fd ← SQRT(fs)`

The square root of the value in FPR fs is calculated to infinite precision, rounded according to the current rounding mode in FCSR, and placed into FPR fd. The operand and result are values in format fmt.

If the value in FPR fs corresponds to – 0, the result is – 0.

Restrictions:

If the value in FPR fs is less than 0, an Invalid Operation condition is raised.

The fields fs and fd must specify FPRs valid for operands of type fmt; if they are not valid, the result is undefined.

The operand must be a value in format fmt; if it is not, the result is undefined and the value of the operand FPR becomes undefined.

Operation:

```
StoreFPR(fd, fmt, SquareRoot(valueFPR(fs, fmt)))
```

Exceptions: Coprocessor Unusable, Reserved Instruction

Floating Point Exceptions: Inexact, Invalid Operation, Unimplemented Operation

4.11 FIR Register Implementation in SB-1

The Floating Point Implementation Register (FIR) is a 32-bit read-only register that contains information identifying the capabilities of the floating point unit, the floating point processor identification, and the revision level of the floating point unit. Figure 4-6 shows the format of the *FIR* register and Table 4-13 describes the *FIR* register fields.

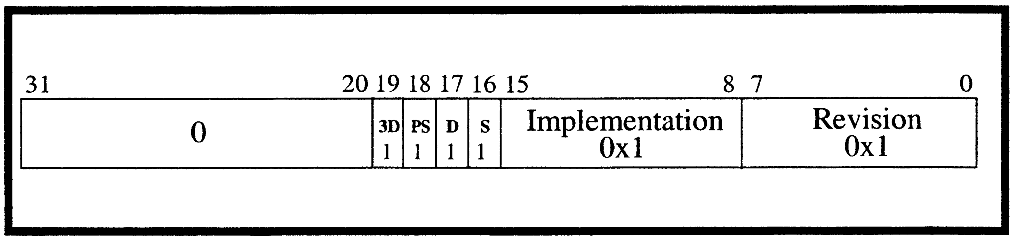

FIGURE 4-6 FIR Register Format in SB-1

TABLE 4-13 FIR Register Field Descriptions

Name	**Bits**	**Description**	**Read/Write**	**Reset State**
0	31:20	Reserved for future use; reads as zero	0	0
3D	19	Indicates that the MIPS-3D ASE is implemented	R	1
PS	18	Indicates that the paired single (PS) floating point data type and instructions are implemented	R	1
D	17	Indicates that the double-precision (D) floating point data type and instructions are implemented	R	1
S	16	Indicates that the single-precision (S) floating point data type and instructions are implemented	R	1
ProcessorID	15:8	Identifies the floating point processor. This value matches the corresponding field of the PRId CP0 register	R	Preset
Revision	7:0	Specifies the revision number of the floating point unit. This allows software to distinguish between one revision and another of the same floating point processor type.	R	Preset

4.12 Exception Processing

This section is currently being worked on and will be fully included in the next revision of this document.

4.12.1 RESET

After Reset, all exceptions are disabled, flush to zero is enabled, and rounding mode is set to RN (Round to nearest-even).

4.12.2 FP Instruction Issue Policy with Exception Off Mode

If no exception is enabled and flush to zero is enabled, the issue box optimally schedules FP operations into the FP unit.

4.12.3 FP Instruction Issue Policy with Exception On Mode

If any exception is enabled or flush to zero is disabled, then no further operations will be issued for one cycle. If the current operation is a long-latency operation (DIV, SQRT, RECIP, RSQRT), then no operation will be issued until the long-latency operation is within 3 cycles of completion.

Medium-latency operations (MADD, MSUB, NMADD, NMSUB, RECIP2, RSQRT2) will hold off the issue of any short or long latency operation until the medium latency operation is within 3 cycles of completion.

4.12.4 Denormals

The SB-1 will flush all denormals to zero if flush to zero is enabled. It will also flush all underflow results to zero.

If flush to zero is disabled, the SB-1 will cause an unimplemented operation exception for denormal inputs and underflowing results for arithmetic operations.

4.12.5 Exception Flags

The following table shows the exception flag settings for various categories of floating point operations.

TABLE 4-14 **SB-1 Exception Behavior**

Operation	Input	Result	FS	E	V	Z	O	U	I
Arith	x	Underflow	1	0	0	0	0	1	1
Arith	x	Underflow	0	1	0	0	0	0	0
Arith	x	Overflow	x	0	0	0	1	0	1
SQRT	Negative Number	x	x	0	1	0	0	0	0
DIV	x/0 = inf, (x != 0)	x	x	0	0	1	0	0	0
DIV	0/0	x	x	0	1	0	0	0	0
DIV	0/inf = 0	x	x	0	0	0	0	0	0
DIV	inf/inf	x	x	0	1	0	0	0	0

CHAPTER 5 The MDMX ASE Instructions

5.1 Introduction

This chapter provides a general overview of MDMX ASE instructions, as supported in SB-1. The information provided here should be regarded as a supplement to MDMX v2.0 Specification document.

5.2 List of Supplemental Instructions

In addition to the standard MIPS MDMX instructions, the SB-1 core supports 3 additional instructions. These instructions are listed in Table 5-1.

TABLE 5-1 SiByte Supported Additional MDMX Instructions

Instruction	Description
PAVG	Averages the elements of a pair of 8 x 8bit vectors
PABSDIFF	Provides the absolute value of the difference of the elements of a pair of 8 x 8bit vectors
PABSDIFFC	Provides the same functionality as PABSDIFF on the input vector pairs and accumulates the results

A description of these instructions follows:

PAVG.OB Perform Bytewise Averaging

31 26	25 21	20 16	15 11	10 6	5 0
MDMX 011110	fmtsel	vt	vs	vd	PAVG 001000
6	5	5	5	5	6

FIGURE 5-1 **PAVG.OB Format**

Format: PAVG.OB vd, vs, vt

Purpose: Perform Bytewise Averaging

Description: `vd[i] ←(vs[i] + select(fmtsel, vt)[i]) / 2`

This instruction only supports OB format.

The sel field selects the values of vt[] used for each i.

Restrictions:

No data-dependent exceptions are possible.

The operands must be values in the specified format. If they are not, the results are undefined and the values of the operand vectors become undefined.

The result of this instruction is undefined if the processor is executing in 16 FP register mode.

Operation:

```
PAVG.OB
        ts ←CPR[vs]
        tt ←select(fmtsel, vt)
        OPR[vd] ←PAVGOB (ts$_{63..56}$ , tt$_{63..56}$ )
                || PAVGOB (ts$_{55..48}$ , tt$_{55..48}$ )
                || PAVGOB (ts$_{47..40}$ , tt$_{47..40}$ )
                || PAVGOB (ts$_{39..32}$ , tt$_{39..32}$ )
                || PAVGOB (ts$_{31..24}$ , tt$_{31..24}$ )
                || PAVGOB (ts$_{23..16}$ , tt$_{23..16}$ )
                || PAVGOB (ts$_{15..08}$ , tt$_{15..08}$ )
                || PAVGOB (ts$_{07..00}$ , tt$_{07..00}$ )
                function PAVGOB(ts, tt)
                        PAVGOB ←[(0 || ts) + (0 || tt) + 1] >> 1
                end PAVGOB
```

Exceptions: Co-processor Unusable, Reserved Instruction, MDMX Unusable

PABSDIFF.OB **Perform Bytewise Absolute Value**

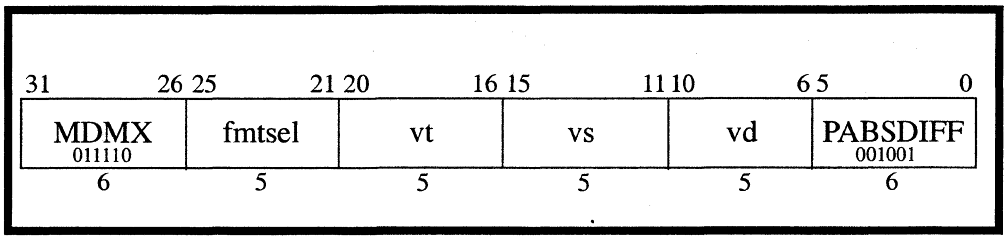

FIGURE 5-2 **PAVG.OB Format**

Format: PABSDIFF.OB vd, vs, vt

Purpose: Perform Bytewise Absolute Value of Differences

Description: `vd[i] ←(vs[i] > select(fmtsel, vt)[i]) ?`

```
(vs[i] - select(fmtsel, vt)[i]) : (select(fmtsel, vt)[i] - vs[i])
```

This instruction only supports OB format.

The sel field selects the values of vt[] used for each i.

Restrictions:

No data-dependent exceptions are possible.

The operands must be values in the specified format. If they are not, the results are undefined and the values of the operand vectors become undefined.

The result of this instruction is undefined if the processor is executing in 16 FP register mode.

Operation:

```
PABSDIFF.OB
       ts ←CPR[vs]
       tt ←select(fmtsel, vt)
       OPR[vd] ← PABSDIFFOB ($ts_{63..56}$ , $tt_{63..56}$ )
              || PABSDIFFOB ($ts_{55..48}$ , $tt_{55..48}$ )
              || PABSDIFFOB ($ts_{47..40}$ , $tt_{47..40}$ )
              || PABSDIFFOB ($ts_{39..32}$ , $tt_{39..32}$ )
              || PABSDIFFOB ($ts_{31..24}$ , $tt_{31..24}$ )
              || PABSDIFFOB ($ts_{23..16}$ , $tt_{23..16}$ )
              || PABSDIFFOB ($ts_{15..08}$ , $tt_{15..08}$ )
              || PABSDIFFOB ($ts_{07..00}$ , $tt_{07..00}$ )
              function PABSDIFFOB(ts, tt)
                     if ts >= tt
                            PABSDIFFOB ←ts - tt
                     else
                            PABSDIFFOB ←tt - ts
              end PABSDIFFOB
```

Exceptions: Co-processor Unusable, Reserved Instruction, MDMX Unusable

PABSDIFFC.OB Perform Bytewise Absolute Value

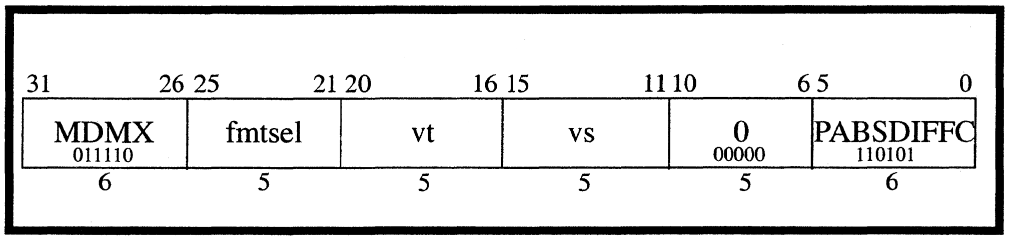

FIGURE 5-3 **PAVG.OB Format**

Format: PABSDIFFC.OB vs, vt

Purpose: Accumulate Absolute Values of Differences of Byte Vectors

Description:

```
acc[i] ← acc[i] + (vs[i] > select(fmtsel, vt)[i]) ?
        (vs[i] - select(fmtsel, vt)[i]) : (select(fmtsel, vt)[i] - vs[i])
```

This instruction only supports OB format.

The sel field selects the values of vt[] used for each i

Restrictions:

No data-dependent exceptions are possible.

The operands must be a value in the specified format. If they are not, the results are undefined and the values of the operand vectors become undefined.

The result of this instruction is undefined if the processor is executing in 16 FP register mode.

Operation:

```
PABSDIFFC.OB
        ts ←CPR[vs]
        tt ←select(fmtsel, vt)
        ACC ←PABSDIFFCOB (acc$_{191..168}$, ts$_{63..56}$ , tt$_{63..56}$ )
                || PABSDIFFCOB (acc$_{167..144}$, ts$_{55..48}$ , tt$_{55..48}$ )
                || PABSDIFFCOB (acc$_{143..120}$, ts$_{47..40}$ , tt$_{47..40}$ )
                || PABSDIFFCOB (acc$_{119..096}$, ts$_{39..32}$ , tt$_{39..32}$ )
                || PABSDIFFCOB (acc$_{095..072}$, ts$_{31..24}$ , tt$_{31..24}$ )
                || PABSDIFFCOB (acc$_{071..048}$, ts$_{23..16}$ , tt$_{23..16}$ )
                || PABSDIFFCOB (acc$_{047..024}$, ts$_{15..08}$ , tt$_{15..08}$ )
                || PABSDIFFCOB (acc$_{023..000}$, ts$_{07..00}$ , tt$_{07..00}$ )
                function PABSDIFFCOB(a, ts, tt)
                        PABSDIFFCOB ←a + PABSDIFFOB(ts, tt)
                end PABSDIFFCOB
```

Exceptions: Co-processor Unusable, Reserved Instruction, MDMX Unusable

5.3 MDMX ASE Instruction Categories in SB-1

MDMX ASE instructions fall into one of three categories as implemented in SB-1. These categories are shown in Table 5-2.

TABLE 5-2 **MDMX Instruction Categories in SB-1**

Category	List of Instructions
TYPE I Non-Accumulator Instructions	**TYPE I-0**: No Condition Code (CC) involvement ADD, SUB, MUL AND, OR, NOR, XOR, SLL, SRL, SRA, MSGN, ALNI, ALNV, MIN, MAX, SHFL, PAVG (SB-1 specific), PABSDIFF (SB-1 specific) **TYPE I-1**: Read CC PICKF, PICKT **TYPE I-2**: Write CC C.EQ, C.LT, C.LE
TYPE II Accumulator Based Instructions	MULS, MULSL, MULL, MULA, SUBA, SUBL, ADDA, ADDL, WACH, WACL, PABSDIFFC (SB-1 specific)
TYPE III Accumulator Read Instructions	RZU, RNAU, RNEU, RZS, RNAS, RNES, RACH, RACL, RACM

5.4 MDMX Unit Block Diagram

The MDMX unit supports 2 execution pipes: A0 and A1. Figure 5-4 shows a block diagram of the A0 pipe and Figure 5-4 show the block diagram for A1 pipe. The following sections specify the types of instructions that can be issued to either pipe.

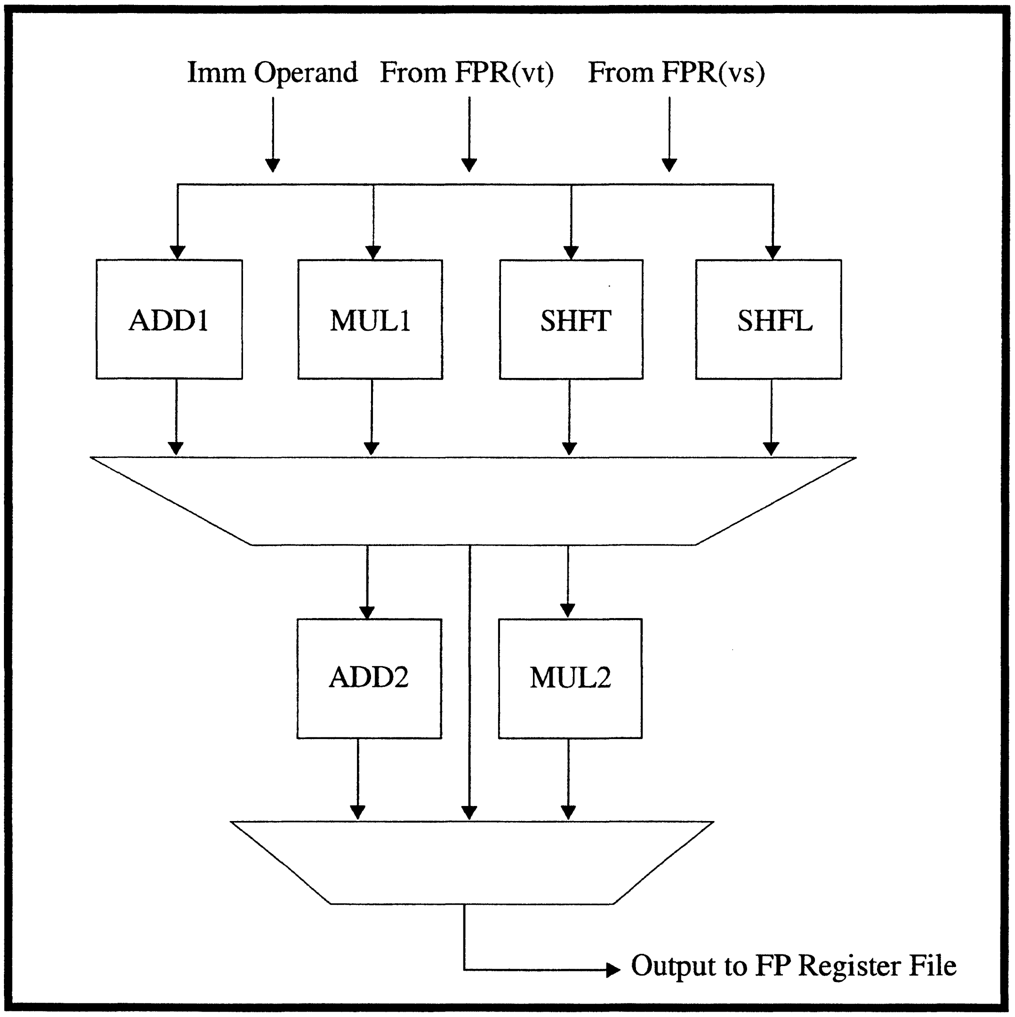

FIGURE 5-4 **A0 Pipe Block Diagram**

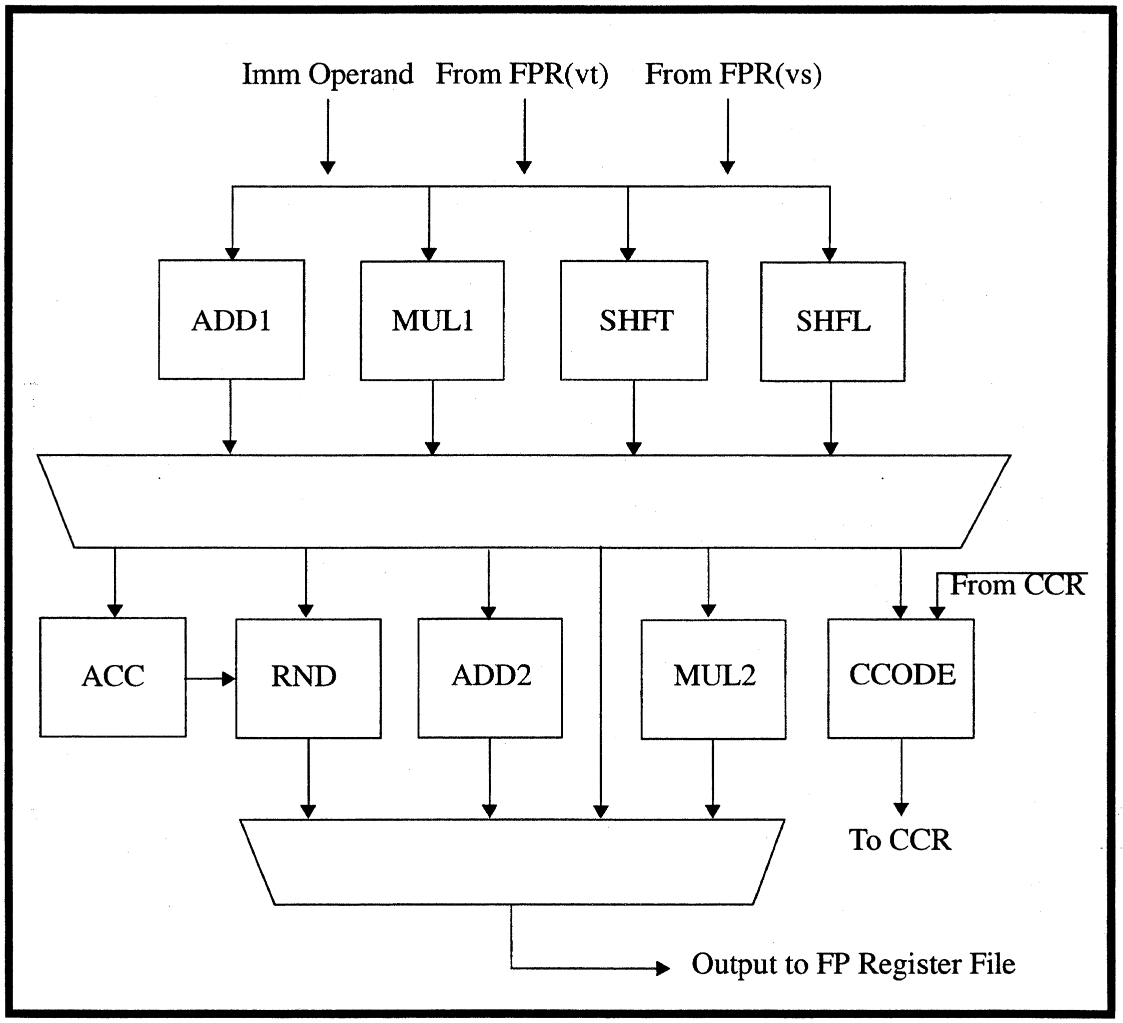

FIGURE 5-5 A1 Pipe Block Diagram

5.5 Pipeline Flow by Category of Instructions

The following sections outline the pipeline flow for the three types of instructions supported by the MDMX unit.

5.5.1 TYPE I Pipe

Table 5-3 shows MDMX pipe diagram for TYPE-I instructions.

TABLE 5-3 MDMX TYPE-I Pipe Operation

Stage	F	D	I	1	2	3	4	5	6
TYPE I-0	Fetch	Decode	Issue	S1	S2	Read RF	Execute1	Execute2	Write RF
TYPE I-1	Fetch	Decode	Issue	S1	S2	Read RF	Execute1 Read CC	Execute 2	Write RF
TYPE I-2	Fetch	Decode	Issue	S1	S2	Read RF	Execute1	Execute 2	Write CC

TYPE-I0 and I2 instructions can be issued to either A0 or A1 pipe of the MDMX Unit, and TYPE-I1 can be issued to A1 pipe only.

5.5.2 TYPE II Pipe

Table 5-4 shows MDMX pipe diagram for TYPE-II instructions.

TABLE 5-4 MDMX TYPE-II Pipe Operation

Stage	F	D	I	1	2	3	4	5	6
TYPE II	Fetch	Decode	Issue	S1	S2	Read RF	Execute1	Execute2	Write Accumulator

TYPE-II instructions can be issued only to A1 pipe of the MDMX Unit.

5.5.3 TYPE III Pipe

Table 5-5 shows MDMX pipe diagram for TYPE-III instructions.

TABLE 5-5 MDMX TYPE-III Pipe Operation

Stage	F	D	I	1	2	3	4	5	6
TYPE III	Fetch	Decode	Issue	S1	S2	Read RF	Execute1 Read Accumulator	Execute2	Write RF

TYPE-III instructions can be issued only to A1 pipe of the MDMX Unit.

5.6 Instruction Latency and Throughput by Category of Instructions

Table 5-6 presents a list of supported instructions in SB-1 Floating Point Unit and the associated latency.

TABLE 5-6 MDMX Instruction Latency and Throughput

Instruction	Latency	Throughput (1 Instruction/x cycles per supported pipe)	Co-issue Dependent Op?
TYPE I	2	1	No
TYPE II	2	1	No
TYPE III	2	1	No

5.7 Available Bypasses

Table 5-7 shows the available bypasses for the MDMX unit.

TABLE 5-7 List of Available Bypasses in SB-1 Core for EX0, EX1, LS0, and LS1 Units

From/To	A0	A1
A0	Yes	Yes
A1	Yes	Yes
LS0	Yes	Yes
LS1	Yes	Yes

5.8 Differences between the Pipes

Table 5-8 summarizes the types of instructions that can be issued to each one of A0 and A1 MDMX pipes. Except for TYPE-I Category instructions, all other instructions can be issued to A1 pipe only.

TABLE 5-8 Instruction Types Issued to each Pipe

Instruction Type	A0 Pipe	A1 Pipe
TYPE I-0, Type I-2	Yes	Yes
TYPE I-1	No	Yes
TYPE II	No	Yes
TYPE III	No	Yes

5.9 Issue Rules and Restrictions

Table 5-9 identifies issue rules and restrictions for MDMX ASE instructions.

TABLE 5-9 Issue Rules and Restrictions for MDMX Instructions

Instruction A	Instruction B	Restrictions
Any TYPE I, II, or III	Any Dependent TYPE I, II, or III	1 cycle bubble

CHAPTER 6 Memory Hierarchy and the Primary Instruction and Data Caches

6.1 Introduction

This chapter elaborates on the supported memory hierarchy and the specifics of level one instruction and data caches in SB-1. For information on level two cache and system interface issues, refer to the appropriate system level user's manual.

6.2 Supported Cache and Memory Hierarchy

Figure 6-1 shows the organization of instruction and data caches, the memory controller, and the bus interface unit around the SB-1 core. Bus widths and their speeds relative to the core are additionally shown on the diagram.

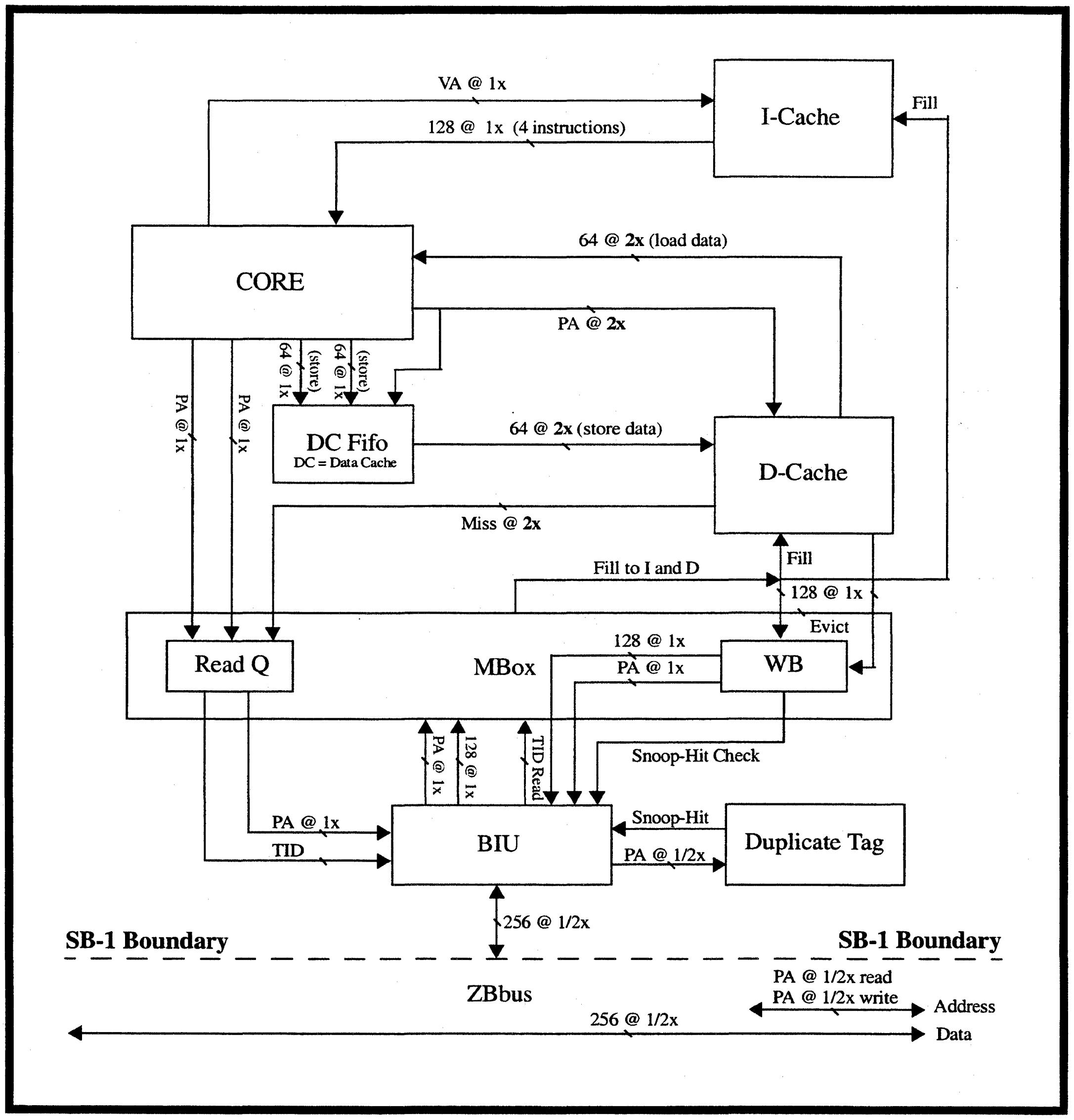

FIGURE 6-1 Memory Structures and Bus Organization around the SB-1 Core

6.2.1 Level One (Primary) Caches

The SB-1 core implements built-in level one instruction and data caches with flexible streaming features. The next two sections elaborate on the specifics of the primary caches.

6.2.1.1 Instruction Cache (I-Cache)

Table 6-1 outlines the main features of the primary instruction cache in SB-1.

TABLE 6-1 **SB-1 Primary Instruction Cache Characteristics**

Cache Feature	**Specifics**
Size	32 KBytes
Associativity	4 way
Replacement Algorithm	LRU (Least Recently Used)
Line Size	32 Bytes
Indexing/Tagging	Virtually Indexed (44-bit address), Virtually Tagged[a]
Read Policy	Blocking
Read Order	Critical Quad Word First (Half Line Resolution)
Write Policy	Not Applicable
Write Order	Not Applicable
Data Parity	1 bit/Byte
Tag Parity	2 bits/Tag
Lock Support	No
Streaming Support	No
ECC Support	No

a. Includes ASID/G bit to avoid flushing for every context switch

6.2.1.1.1 Accessing the Instruction Cache

Figure 6-2 shows the manner in which the 44-bit virtual address is used to access a line. As shown in the figure, the index consists of 8 bits, resulting in 256 individually accessible sets of 32-byte lines by 4 ways. The address portion of the tag consists of 31 bits.

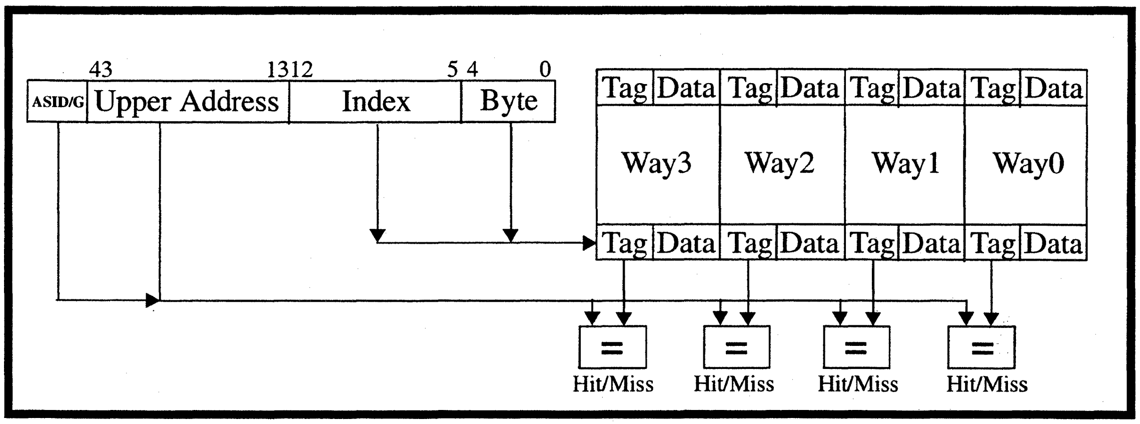

FIGURE 6-2 Primary Instruction Cache Indexing in SB-1

6.2.1.1.2 Address Fields Decoding

Figure 6-3 shows bank organization in the primary instruction cache for SB-1.

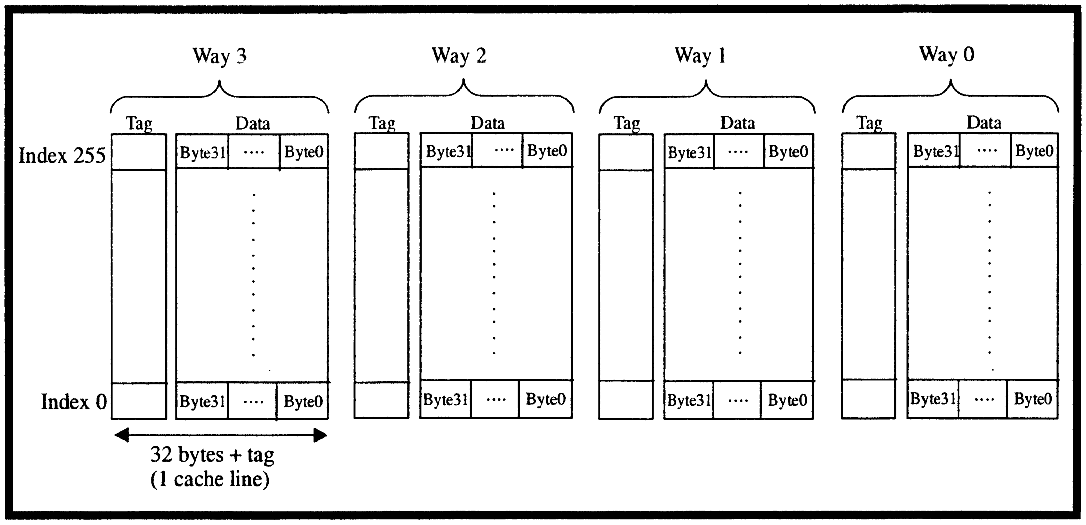

FIGURE 6-3 **Instruction Cache Organization in SB-1**

6.2.1.1.3 Parity/ECC Support

The primary instruction cache in SB-1 supports data and tag parities (shown in Table 6-1) but does not have ECC.

6.2.1.1.4 Notes on the Virtual Nature of the Instruction Cache

The following should be considered when dealing with the instruction cache:

1. Virtual aliases may cause multiple copies of the same cache data to appear in the instruction cache.
2. If a mapped address is changed from a cached attribute to an uncached attribute, the cache lines must be flushed from the instruction cache to eliminate the stale instructions. For correct operation, mapped addresses to an uncached space must never be present in the instruction cache.

   Specifically, cachable and uncachable references to the same space do not preserve coherence. Note that the L1 I-cache does not participate in the coherence algorithms.
3. Because of (2) above, the I-Cache must be flushed before seeing a write into the code stream (e.g., planting a breakpoint).

4. Mapped addresses to uncached space will cause an instruction cache lookup and subsequent error detection to be performed. As a result, it is possible to detect an instruction cache error even though the page mapping for that address is uncached, causing what may be referred to phantom CacheError exceptions.
5. The D-Cache will not supply data to satisfy an I-Cache miss for the same CPU.
6. The ASID field in the EntryHi register should only be modified by a DMTC0 or TLBR instruction in unmapped or in mapped global space, i.e. the G bit is set in the TLB entry. If the ASID is changed in mapped space that is not global, i.e. the G bit is cleared, the behavior of the processor is UNDEFINED, and TLB exceptions, including TLB Shutdown, may result.

6.2.1.2 Data Cache (D-Cache)

Table 6-2 outlines the main features of the Primary Data cache in SB-1.

TABLE 6-2 SB-1 Primary Data Cache Characteristics

Cache Feature	Specifics
Size	32 KByte
Associativity	4 way
Replacement Algorithm	LRU (Least Recently Used)
Line Size	32 Bytes
Indexing/Tagging	Physically Indexed (40-bit address), Physically Tagged
Read Policy	Non-Blocking -- Up to 8 outstanding cache lines
Read Order	Critical Double Word First
Write Policy	Write Back only (no Write-Through support)
Data Parity	See ECC support
Tag Parity	2 bits/tag
Lock Support	No
Streaming Support	Yes: Up to a maximum of 8KByte (1 way)
ECC Support	Yes: 8 bits per 64-bit doubleword; single-bit error correction, double-bit error detection

6.2.1.2.1 Accessing the Data Cache

Figure 6-4 shows the manner in which the 44-bit virtual address is used to access a line. As shown in the figure, the index consists of 8 bits, resulting in 256 individually accessible 32-byte lines by 4 ways. The address portion of the tag consists of 28 bits.[1] The next section elaborates on the full composition of bits in the tag field.

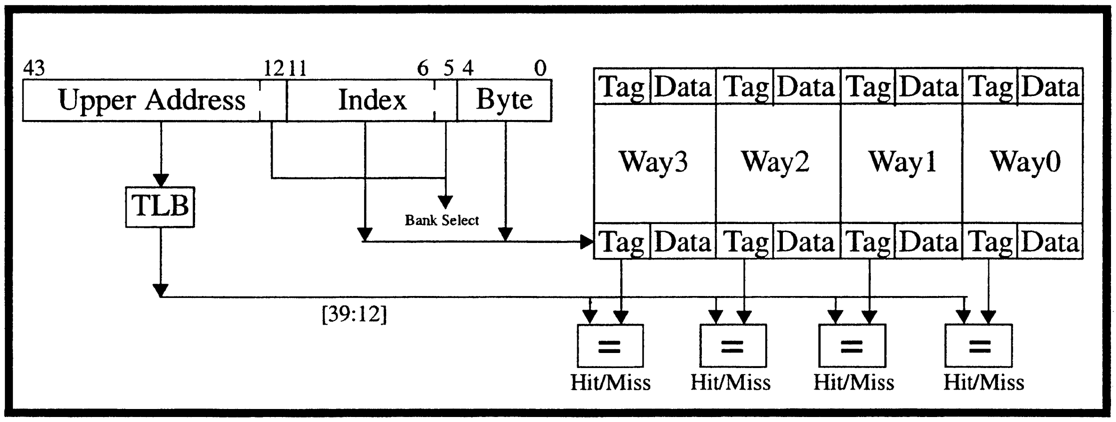

FIGURE 6-4 Primary Data Cache Indexing in SB-1

1. Bit 12 (shown in Figure 6-4) is part of the tag and part of the index. A program making explicit reference to tags (via TagLo register) must be aware of this and maintain consistency between index and tags at that index.

6.2.1.2.2 Address Fields Decoding

Figure 6-5 shows bank organization in the primary data cache for SB-1.

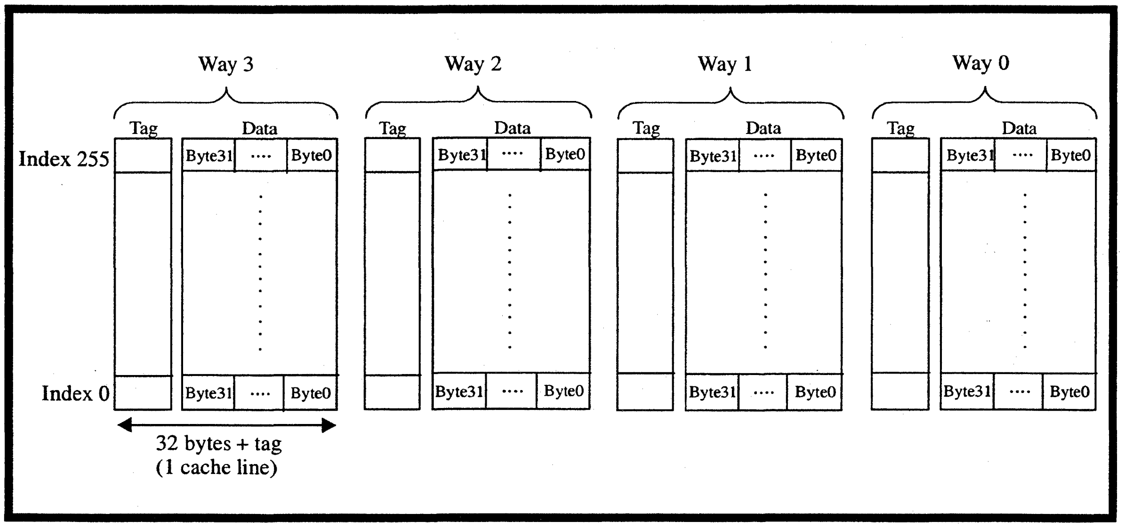

FIGURE 6-5 **Data Cache Organization in SB-1**

6.2.1.2.3 Parity/ECC Support

The Primary Data Cache in SB-1 supports tag parity (shown in Table 6-1) and has 64-bit ECC for the data portion. Single-bit errors are corrected and double-bit errors are detected. Refer to Chapter 10 for further description of these error cases.

6.2.2 Rules for Uncached Data Accesses

The following list tabulates the rules and restrictions for uncached accesses:

1. Uncached accesses are issued in order.
2. Uncached accesses are never issued speculatively.
3. Uncached writes are blocked if there are any outstanding uncached reads.
4. Uncached accesses may be issued in any number, subject to normal ReadQ/Write buffer depths.
5. External system must maintain ordering:

   - Reads to a device must not pass reads or writes to same device.

- Writes to a device must not pass writes to same device.

6.2.3 Operation of the Write Buffer

The write buffer is a 10-deep storage structure that holds data on its way to memory. Each entry consists of a PA and 32 bytes of data. There are three logical sources for the data being put into the write buffer, outlined below:

1. Lines evicted from the Data Cache (as a result of a fill or a CACHE instruction). The data is always a full cache line (32 bytes).
2. Uncacheable stores. These data come from the DCFIFO where they are held until store instructions graduate. The data in this case is between 1 and 8 bytes wide.
3. Uncacheable accelerated stores. These are the same as uncacheable stores but can be merged in the write buffer, provided they obey the merging rules outlined next.
4. Uncacheable loads (in order to maintain order with uncacheable stores).

6.2.3.1 Merging Rules

The write buffer contains one 32-byte merge buffer.

The merge buffer begins merging when an uncached accelerated (UAC) double or single word block-aligned store is executed. Merging continues if the next uncacheable write buffer request is a UAC double or single word store to an address within the same block. There are two merging modes. If the next request is to an identical address, then the merging mode is auto-increment, otherwise, the merging mode is sequential. The merging mode is established by the second UAC store to the block.

Merging stops when one of the following conditions is met:

- An uncached or UAC load is executed.
- An uncached store is executed.
- A UAC partial-word store is executed.
- A change in the current merging mode is observed.
- A complete block is gathered[1].
- The time-out counter indicates that 512 cycles have passed since the last UAC store was observed and no other write buffer request has happened.

1. In sequential mode, the block is considered complete when the 2 highest-addressed words of the block are written.

Cached accesses to the write buffer do not disturb merging. When merging terminates, the data is placed into a write buffer entry and is ready to be issued to the system interface bus.

When gathering in auto-increment mode, UAC double or singleword stores may be freely mixed. The data will be appended to the end of the already merged data in the merge buffer. However, if the merge buffer already contains seven valid words and the next request is a UAC double store, the doubleword will not fit into the same 32-byte block. In this case, the seven words in the merge buffer are placed into a write buffer entry and the new double store starts a new merging block.

When gathering in sequential mode, UAC singleword stores must occur in pairs to prevent address error exceptions.

6.2.4 Prefetch Support for Primary Data Cache (User Level Prefetching and Streaming)

The primary data cache in SB-1 supports a number of the Prefetch Hints as specified in MIPS64 Specification. Among the supported hints are regular data prefetching and streaming data through the data cache.

Table 6-3 presents the high level features provided by PREF (PREFX) instruction. The subsequent two sections describe regular prefetching and streaming through the PREF (PREFX) instruction.

TABLE 6-3 Cache Prefetch Support for Primary Data Cache

Feature	Description
Instruction Type	PREF (or PREFX) Instruction, a Non-Privileged Instruction (refer to MIPS64 Specifications).
Description of Instruction	PREF adds the 16-bit signed *offset* to the contents of GPR *base* to form an effective byte address. PREFX adds the contents of GPR index to the contents of GPR base to form an effective byte address. The *hint* field supplies information about how the addressed data is to be manipulated. PREF (PREFX) enables the processor to take some action as specified by the *hint* field, to improve program performance. The action taken for a specific PREF (PREFX) instruction is both system and context dependent. Any action, including doing nothing, is permitted as long as it does not change architecturally visible state or alter the meaning of a program. A PREF (PREFX) instruction either does nothing or takes an action that increases the performance of the program. PREF (PREFX) does not cause addressing-related exceptions. If it does happen to raise an exception condition, the exception condition is ignored. If an addressing-related exception condition is raised and ignored, no data movement occurs. PREF (PREFX) never generates a memory operation for a location with an uncached memory access type. For a cached location, the expected and useful action for the processor is to move a block of data between cache and the memory hierarchy. The size of the block transferred in SB-1 is one line of data (32 Bytes).

TABLE 6-3 **Cache Prefetch Support for Primary Data Cache**

Feature	Description
Granularity	1 Line (32 Bytes)
Programming Notes	Prefetch cannot access a mapped location unless the translation for that location is present in the TLB. Locations in memory pages that have not been accessed recently may not have translations in the TLB, so prefetch may not be effective for such locations. Prefetch does not cause addressing exceptions. It does not cause an exception to prefetch using a pointer before the validity of the pointer is determined. Hint field encodings whose function is described as "streamed" convey usage intent from software to hardware. Software should not assume that hardware will always prefetch data in an optimal way.

6.2.4.1 Regular Data Prefetching

Table 6-4 describes the regular data prefetch support provided by SB-1 core.

TABLE 6-4 **Regular Data Prefetch Support Provided by SB-1**

Feature	Description
Instruction Type	PREF or PREFX Instruction, Non-Privileged (refer to MIPS64 Specifications).
Load Hint (Hint Value = 0)	Use: Prefetched data is expected to be read (not modified) Action: Fetch data as if for a load
Store Hint (Hint Value = 1)	Use: Prefetched data is expected to be stored or modified Action: Fetch data as if for a store
Data Placement	Always put in current LRU, upgrading way to MRU

6.2.4.2 Streaming Prefetch Support in SB-1

Table 6-5 outlines the streaming prefetch support provided by the primary data cache.

TABLE 6-5 **Streaming Prefetch Support in SB-1**

Feature	Description
Instruction Type	PREF or PREFX Instruction, Non-Privileged (refer to MIPS64 Specifications)
Load_Streamed Hint (Hint Value = 4)	Use: Prefetched data is expected to be read (not modified) but not reused extensively; it "streams" through the cache.
Store_Streamed Hint (Hint Value = 5)	Use: Prefetched data is expected to be stored or modified but not reused extensively; it "streams" through the cache.
Data Placement	If the block is already in the cache, treat as regular prefetch (can upgrade to MRU). Otherwise, replace the LRU way without upgrading that way, and mark the way so that subsequent hits do not upgrade. The next fill to that index resets the way settings.

6.2.5 The PREF and PREFX Instructions in SB-1[1]

This section presents a general description of PREF and PREFX instructions in SB-1.

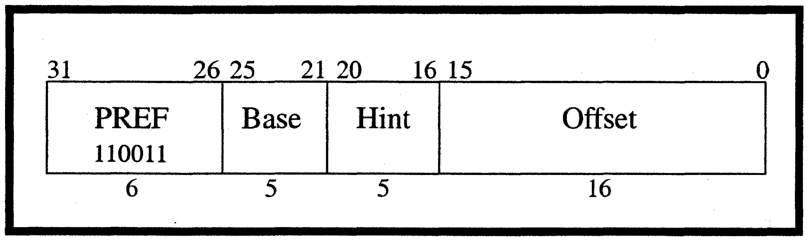

FIGURE 6-6 **Format for PREF Instruction**

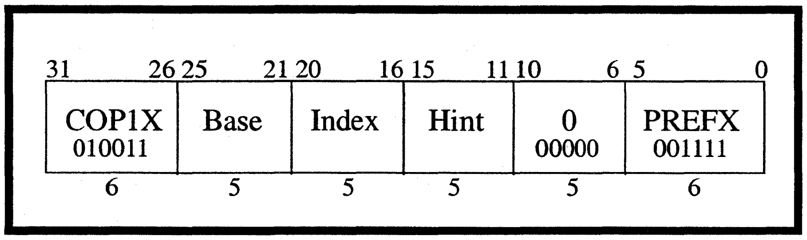

FIGURE 6-7 **Format for PREFX Instruction**

Format:

PREF hint, offset(base)

PREFX hint, index(base)

Purpose:
To move data between memory and cache.

1. The PREFX instruction is identical to PREF but supports base + index addressing instead. Except for address computation, all descriptions for PREF apply equally to PREFX.

Description:

PREF adds the 16-bit signed *offset* to the contents of GPR *base* to form an effective byte address.

PREFX adds the contents of GPR *index* to the contents of GPR *base* to form an effective byte address.

The *hint* field supplies information about how the addressed data is to be manipulated.

PREF enables the processor to take some action as specified by the *hint* field, to improve program performance. The action taken for a specific PREF instruction is both system and context dependent. Any action, including doing nothing, is permitted as long as it does not change architecturally visible state or alter the meaning of a program (refer to Table 6-6 for more details).

PREF does not cause addressing-related exceptions. If it does happen to raise an exception condition, the exception condition is ignored. If an addressing-related exception condition is raised and ignored, no data movement occurs.

PREF never generates a memory operation for a location with an uncached memory access type. For a cached location, the expected and useful action for the processor is to move a block of data between cache and the memory hierarchy. The size of the block transferred in SB-1 is one line (32 bytes).

Table 6-6 defines the hint field values.

TABLE 6-6 PREF Hint Field Encodings

Value	Name	Data Use and Desired PREF Action	SB-1 Reference
0	load	Use: Prefetched data is expected to be read (not modified) Action: Fetch data as if for a load.	Supported in SB-1 Section 6.2.4.1: "Regular Data Prefetching"
1	store	Use: Prefetched data is expected to be stored or modified Action: Fetch data as if for a store.	Supported in SB-1 Section 6.2.4.1: "Regular Data Prefetching"
2-3	Reserved	Reserved for future use - not available to implementations.	-
4	load_streamed	Use: Prefetched data is expected to be read (not modified) but not reused extensively; it "streams" through the cache Action: Fetch data as if for a load and place it in the cache so that it does not displace data prefetched as "retained"	Supported in SB-1 Section 6.2.4.2: "Streaming Prefetch Support in SB-1"
5	store_streamed	Use: Prefetched data is expected to be stored or modified but not reused extensively; it "streams" through the cache Action: Fetch data as if for a store and place it in the cache so that it does not displace data prefetched as "retained"	Supported in SB-1 Section 6.2.4.2: "Streaming Prefetch Support in SB-1"
6	load_retained	Not Applicable	Not Supported in SB-1
7	store_retained	Not Applicable	Not Supported in SB-1
8-24	Reserved	Reserved for future use - not available to implementations	-

TABLE 6-6 **PREF Hint Field Encodings**

Value	Name	Data Use and Desired PREF Action	SB-1 Reference
25	writeback_invalidate (also known as nudge)	Use: Data is no longer to be expected to be used Action: schedule a writeback of any dirty data. At the completion of the writeback, mark as invalid the state of any cache line written back.	Supported in SB-1 Explanation to the left
26-31	Implementation Dependent	Unassigned by the Architecture	-

Restrictions:
None

Operation:

```
vAddr ← GPR[base] + sign_extend(offset)[1]
(pAddr, CCA) ← AddressTranslation(vAddr, DATA, LOAD)[2]
Prefetch(CCA, pAddr, vAddr, DATA, hint)
```

Exceptions:
Prefetch does not take any TLB-related or address-related exceptions under any circumstances.

Programming Notes:
Prefetch cannot access a mapped location unless the translation for that location is present in the TLB. Locations in memory pages that have not been accessed recently may not have translations in the TLB, so prefetch may not be effective for such locations.

Prefetch does not cause addressing exceptions. It does not cause an exception to prefetch using a pointer before the validity of the pointer is determined.

Hint field encodings whose function is described as "streamed" convey usage intent from software to hardware. Software should not assume that hardware will always prefetch data in an optimal way.

Implementation Notes:
The SB-1 does not trigger a data watch by a prefetch instruction whose address matches the Watch register address match conditions.

1. For PREFX, the address computation is: vAddr ← GPR[base] + GPR[index]
2. AddressTranslation, as used here, cannot raise any exceptions.

6.3 CACHE Instructions

This section covers the microarchitectural implementation of the MIPS CACHE instructions. For reference, the table of CACHE instructions is repeated here with a brief description of each operation. In addition, the registers used with the CACHE instructions, i.e., TagHi, TagLo, DataHi, and DataLo are defined.

6.3.1 CACHE Variants

The following tables, divided by cache type, list the types of cache operations defined by MIPS and implemented by SB-1. Special debug cache operations are described as well.

TABLE 6-7 Instruction Cache

Operation	Bits [20:16] of Cache Inst[a]	Description
Index Inval	000I	Invalidate the cache line at the specified index
Index Load Tag	001I	Read the cache line tag at the specified index into the TagHi/TagLo registers
Index Store Tag	010I	Write the cache line tag at the specified index from the TagHi/TagLo registers
Hit Inval	100I	Invalidate the cache line at the specified address if it is present in the cache
Index Load Data	001T	Read the contents of the data array into DataHi/DataLo registers (debug only)
Index Str Data	010T	Write the contents of the DataHi/DataLo registers into the data array (debug only)

a. I = 00, D = 01, T = 10, S = 11

TABLE 6-8 Data Cache

Operation	Bits [20:16] of Cache Inst[a]	Description
Index WB Inval	000D	Writeback dirty data and invalidate the cache line at the specified index
Index Load Tag	001D	Read the cache line tag at the specified index into the TagHi/TagLo registers
Index Store Tag	010D	Write the cache line tag at the specified index from the TagHi/TagLo registers
Hit Inval	100D	Invalidate the cache line at the specified address if it is present in the cache
Hit WB Inval	101D	Writeback dirty data and invalidate the cache line at the specified address if it is present in the cache
Hit WB	110D	Writeback dirty data and set the cache line state to clean at the specified address if the cache line is present in the cache
Index Load Data	001S	Read the contents of the data array into DataHi/DataLo registers (debug only)
Index Str Data	010S	Write the contents of the DataHi/DataLo registers into the data array (debug only)

a. I = 00, D = 01, T = 10, S = 11

The effective address for a CACHE instruction is calculated by adding the instruction offset field to a base register. The resulting address is translated by the TLB, and depending on the target cache, either the effective address or the translated address is used to access the cache. The process of translation may cause a TLB Refill or TLB Invalid exception but not a TLB Modified exception. In addition, the effective address for a CACHE instruction never generates a Watch exception, although address errors will be asserted for addresses that are not legal in the current operating mode.

For the instruction cache, the effective address is used to access the cache. The address is translated regardless of the operation, but all TLB errors are suppressed, even though address errors may still result. Index operations use bits [12:5] to specify a set index and bits [14:13] to specify a way for the 32K cache on the SB-1 core. For hit operations, the ASID in the EntryHi register is coupled with the effective address to generate a virtual address, which is compared against the cache tags to detect a hit or miss.

For the physically addressed data cache, the effective address is translated through the TLB for hit variants. TLB exceptions, due to the translation process, as well as address error exceptions may occur for these CACHE instructions. The resulting physical address is used to determine a hit or miss in the cache. Index operations, however, bypass the TLB, so no TLB exceptions will occur. Address errors may arise if the effective address is not valid for the current operating mode. Like the instruction cache, bits [12:5] indicate a set index, while bits [14:13] designate a way for index operations.

In the SB-1 implementation, CACHE instructions ignore byte alignment. As such, address errors due to misalignment will never occur.

The Index Load/Store Data operations, defined for debug purposes, require additional bits to specify the double word location in a cache line. Bits [4:3] of the address are used for this purpose, so the operation effectively behaves like a double word load/store.

The following sections describe the cache operations supported by the SB-1 core.

6.3.2 Index Invalidate (I)

The Index Invalidate variant sets the state of an instruction cache line at the specified index to invalid by clearing the valid and parity bits (VP = 00). The index is taken from the effective address bits [12:5] and the way is selected by bits [14:13]. The LRU remains unchanged and no parity check is performed. Address errors may occur for invalid addresses, but no TLB exceptions are raised.

6.3.3 Index Load Tag (I)

The Index Load Tag operation loads the instruction cache TagHi and TagLo registers with the information stored in the I-Cache tag array. The tag index and way are taken from address bits [12:5] and [14:13], respectively. See the TagHi and TagLo definitions below for the format and data transferred by these registers. The LRU remains unchanged and no parity check is performed. Address errors may occur for invalid addresses, but no TLB exceptions are raised. The LU bit is set to one when an Index Load Tag is performed.

6.3.4 Index Store Tag (I)

The Index Store Tag operation reads the instruction cache TagHi and TagLo registers and stores the information into the cache tag array. The tag index and way are taken from address bits [12:5] and [14:13], respectively. See the TagHi and TagLo definitions in section 6.6 for the format and data transferred by these registers. Address errors may occur for invalid computed addresses, but no TLB exceptions are raised. If the LU bit is set in the TagHi register, then the LRU is set to the state indicated by the LRU field; otherwise, it is set to a default state. Invalid LRU values will also be reset to a default state. See the LRU implementation notes below. A parity calculation is not performed by this operation, so the parity bits for the tag are taken directly from the P1, P0, and P bits in the Tag registers.

6.3.5 Hit Invalidate (I)

The Hit Invalidate operation clears the state of an instruction cache line if the effective address and the ASID match a tag in the cache. A hit sets the valid and parity bits to zero (VP = 00). The LRU remains unchanged, and any detected parity errors are ignored. Address errors may occur for invalid addresses, but no TLB exceptions are raised.

6.3.6 Index Load Data (I)

The Index Load Data operation is implemented purely for debug purposes and loads the instructions from the data array into the DataHi and DataLo registers. Two instructions are loaded into the DataLo register, while the parity and predecode for those instructions is written into the DataHi register.

This operation is endian-neutral, so software must interpret the instructions correctly. For big-endian code, InstA contains instruction word 0 and InstB contains instruction word 1. The opposite is true for little-endian code, so InstA contains word 1 and InstB contains word 0. This format allows software to move the double word data directly to a register and perform double word stores to code space without swapping. The LRU remains unchanged for Index Load Data CACHE instructions, and a parity check is not performed. Address errors may occur for invalid addresses, but no TLB exceptions are raised.

6.3.7 Index Store Data (I)

The Index Store Data operation is implemented purely for debug purposes and stores the instructions contained in the DataHi and DataLo registers into the instruction cache data array. The DataLo register contains two instructions, while the DataHi register includes the parity and predecode for those instructions. The predecode logic is bypassed for this operation, so the predecode bits for each instruction are taken from the register.

This operation is endian-neutral, so software must write instructions into the InstA and InstB fields in the DataLo register for the appropriate endianness. For big-endian code, InstA contains instruction word 0 and InstB contains instruction word 1. The opposite is true for little-endian code, so InstA contains word 1 and InstB contains word 0. This format allows software to perform doubleword loads to code space and to move the doubleword data directly to DataLo without swapping.

The LRU remains unchanged for Index Store Data CACHE instructions. A parity calculation is not performed by this operation, so the parity bits for the data are taken directly from the IPA and IPB fields. The predecode bits and predecode parity must also be calculated by software and placed in the DataHi register. Address errors may occur for invalid addresses, but no TLB exceptions are raised.

6.3.8 Index Invalidate (D)

The Index Invalidate variant sets the state of a data cache line at the specified index to invalid by clearing the state, coherent, and check bits to all zeros. The index is taken from the effective address bits [12:5] and the way is selected by bits [14:13]. The LRU remains unchanged and no parity check is performed. Address errors may occur for invalid addresses, but no TLB exceptions are raised.

6.3.9 Index Load Tag (D)

The Index Load Tag operation loads the data cache TagHi and TagLo registers with the information stored in the cache tag array. The tag index and way are taken from address bits [12:5] and [14:13], respectively. See the TagHi and TagLo definitions below for the format and data transferred by these registers. The LRU remains unchanged and no parity check is performed. Address errors may occur for invalid addresses, but no TLB exceptions are raised. The LU bit is set to one when an Index Load Tag is performed.

6.3.10 Index Store Tag (D)

The Index Store Tag operation reads the data cache TagHi and TagLo registers and stores the information into the cache tag array. The tag index and way are taken from address bits [12:5] and [14:13], respectively. See the TagHi and TagLo definitions below for the format and data transferred by these registers. Address errors may occur for invalid addresses, but no TLB exceptions are raised.

If the LU bit is set in the TagHi register, then the LRU is set to the state indicated by the LRU field; otherwise, it is set to a default state. Invalid LRU values will also be reset to a default state. See the LRU implementation notes below.

A parity calculation is not performed by this operation, so the parity bits for the tag are taken directly from the P1 and P0 bits in the TagLo register. In addition, the state check bits are sourced directly from the TagHi register.

6.3.11 Hit Invalidate (D)

The Hit Invalidate operation clears the state of a data cache line if the translated physical address matches a tag in the cache. A hit sets the state, coherent, and check bits to zero. The LRU remains unchanged. Address errors may occur for invalid addresses, and TLB exceptions may be raised as a result of the address translation.

Parity errors detected by this operation leave the state of the data cache unchanged. In addition, any cache error exceptions will be taken imprecisely.

6.3.12 Hit Writeback Invalidate (D)

The Hit Writeback Invalidate operation causes a cache line to be written back to memory if the translated physical address matches a tag in the cache and the data are dirty. Additionally, the state of the line is cleared, and the state, coherent, and check bits are set to zero. The LRU remains unchanged. Address errors may occur for invalid addresses, and TLB exceptions may be raised as a result of the address translation.

Tag parity errors detected by this operation leave the state of the data cache unchanged, and no writebacks to the bus occur. In addition, any cache error exceptions will be taken imprecisely.

A single-bit ECC error detected by this operation is corrected, and the data are written into memory with a corrected data code. A double-bit ECC error detected by this operation is not corrected, and the data are written to memory with an uncorrected data code. In either case, any cache error exceptions will be taken imprecisely.

6.3.13 Hit Writeback (D)

The Hit Writeback operation causes a cache line to be written back to memory if the translated physical address matches a tag in the cache and the data are dirty. Additionally, the state of the line is modified to clean, and the the data are retained in the cache. For coherent lines, the state becomes 0b10 and the check bits change to 0b11. For non-coherent lines, the state becomes 0b10 and the check bits change to 0b10. The LRU remains unchanged. Address errors may occur for invalid addresses, and TLB exceptions may be raised as a result of the address translation.

Tag parity errors detected by this operation leave the state of the data cache unchanged, and no writebacks to the bus occur. In addition, any cache error exceptions will be taken imprecisely.

A single-bit ECC error detected by this operation is corrected, and the data are written into memory with a corrected data code. A double-bit ECC error detected by this operation is not corrected, and the data are written to memory with an uncorrected data code. In either case, any cache error exceptions will be taken imprecisely.

6.3.14 Index Load Data (D)

The Index Load Data operation is implemented purely for debug purposes and loads doubleword data and ECC from the data array into the DataHi and DataLo registers. The data are loaded into the DataLo register, while the ECC bits for the data are written into the DataHi register. The index and way for the operation come from bits [12:5] and [14:13], respectively.

The LRU remains unchanged for Index Load Data CACHE instructions, and an ECC check is not performed. Address errors may occur for invalid addresses, but no TLB exceptions are raised.

6.3.15 Index Store Data (D)

The Index Store Data operation is implemented purely for debug purposes and stores the doubleword data and ECC contained in the DataHi and DataLo registers into the data cache data array. The DataLo register contains the doubleword data, while the DataHi register includes the ECC for the data. The index and way for the operation come from bits [12:5] and [14:13], respectively.

The LRU remains unchanged for Index Store Data CACHE instructions. An ECC calculation is not performed by this operation, and the ECC bits for the data are taken directly from the DataHi register. Address errors may occur for invalid addresses, but no TLB exceptions are raised.

6.4 Cache Operation Effects on Duplicate Tags

This section is TBD.

6.5 CACHE Instruction Issue Rules

CACHE instructions are serially issued, i.e. all previous instructions must graduate so that potential mispredicts and exceptions are cleared before the operation executes. In addition, the CACHE operation performs an implicit memory synchronization since outstanding loads and stores (and even other CACHE instructions) may update the cache state. An implicit memory synchronization follows the CACHE operation as well so subsequent loads and stores can observe the effect of the CACHE instruction.

Note that the synchronization does not apply to instruction accesses, so the result of a CACHE operation on the instruction cache is unpredictable if the effective address generated by the CACHE operation is near a potential cached instruction fetch path.

6.6 Register Definitions

This following sections cover the tag and data registers supported by SB-1 core.

6.6.1 Tag Registers (MIPS Compliant)

TABLE 6-9 TagLo Register: Register 28, Select 0 (Instruction Cache)

Bits	Size	Field	Description
[63:62]	2b	R	Virtual Address Region bits
[61:44]	18b	0	Read as zeros; ignored on write
[43:25]	19b	VTag1	Virtual Address bits [43:25]
[24:13]	12b	VTag0	Virtual Address bits [24:13]
[12]	1b	0	Read as zeros; ignored on write
[11]	1b	P1	Parity Bit 1; even parity for R and VTag1
[10]	1b	P0	Parity Bit 0; even parity for VTag0, G, and ASID
[9]	1b	0	Read as zeros; ignored on write
[8]	1b	G	Global Bit
[7:0]	8b	ASID	Address Space Identifier

TABLE 6-10 TagHi Register: Register 29, Select 0 (Instruction Cache)

Bits	Size	Field	Description
[31:30]	2b	0	Read as zeros; ignored on write
[29]	1b	V	Cache Tag Valid Bit
[28]	1b	0	Read as zeros; ignored on write
[27]	1b	P	Parity Bit; even parity for V
[26:23]	3b	0	Read as zeros; ignored on write
[22]	1b	LU	LRU Update Bit
[21:14]	8b	LRU	Least Recently Used Pointer
[13:0]	14b	0	Read as zeros; ignored on write

TABLE 6-11 TagLo Register: Register 28, Select 2 (Data Cache)

Bits	Size	Field	Description
[63:40]	24b	0	Read as zeros; ignored on write
[39:26]	14b	PTag1	Physical Address bits [39:26]
[25:13]	13b	PTag0	Physical Address bits [25:13]

TABLE 6-11 TagLo Register: Register 28, Select 2 (Data Cache)

Bits	Size	Field	Description
[12]	1b	0	Read as zeros; ignored on write
[11]	1b	P1	Parity Bit 1; even parity for PTag1
[10]	1b	P0	Parity Bit 0; even parity for PTag0
[9:0]	10b	0	Read as zeros; ignored on write

TABLE 6-12 TagHi Register: Register 29, Select 2 (Data Cache)

Bits	Size	Field	Description
[31:30]	2b	0	Read as zeros; ignored on write
[29:28]	2b	State	Data Cache State
[27]	1b	Coh	Coherent Data Bit
[26:25]	2b	Check	Check Bits
[24]	1b	ExtNC	Not cached in external Caches (e.g. L2)
[23]	1b	Stream	Stream Bit
[22]	1b	LU	LRU Update Bit
[21:14]	8b	LRU	Least Recently Used Pointer
[13:0]	14b	0	Read as zeros; ignored on write

TABLE 6-13 State/Coherent/Check Field Encodings[a]

State	Coherent	Check	Description
00	0	00	Invalid
01	1	11	Coherent-Shared
10	0	11	Non-Coherent-Exclusive-Clean
11	0	01	Non-Coherent-Exclusive-Dirty
10	1	10	Coherent-Exclusive-Clean
11	1	00	Coherent-Exclusive-Dirty

a. All other combinations are error combinations

LRU Implementation Notes:

The LRU pointer contains four 2-bit entries corresponding to the MRU to LRU ways, i.e. the two most-significant bits indicate the MRU way while the two least-significant bits indicate the LRU (LRU acts as a FIFO.) For the LRU to be valid, each entry must contain a unique 2-bit way number, which results in a total of 24 valid combinations. The LRU pointer is corrected by the cache when one of the invalid combinations is detected; the default value for the error case forces the entries, from MRU to LRU, to the following: way 3, way 2, way 1, way 0. The default value is also written during an Index Store Tag when the LU bit in the TagHi register is clear. In addition, the LU bit is read as a one when an Index Load Tag operation is performed.

6.6.2 Data Registers (SiByte Debug Defined)

The following tables define the Data Register portion of CACHE operations supported by SB-1.

TABLE 6-14 **DataLo Register: Register 28, Select 1 (Instruction Cache)**

Bits	Size	Field	Description
[63:32]	32b	InstA	Instruction A
[31:0]	32b	InstB	Instruction B

TABLE 6-15 **DataHi Register: Register 29, Select 1 (Instruction Cache)**

Bits	Size	Field	Description
[63:17]	47b	0	Read as zeros; ignored on write
[16]	1b	PDP	Predecode Parity Bit; even parity for PDA and PDB
[15:12]	4b	PDA	Instruction Predecode bits for InstA
[11:8]	4b	PDB	Instruction Predecode bits for InstB
[7:4]	4b	IPA	Even Byte Parity for InstA
[3:0]	4b	IPB	Even Byte Parity for InstB

TABLE 6-16 **DataLo Register: Register 28, Select 3 (Data Cache)**

Bits	Size	Field	Description
[63:0]	64b	Data	Cache Data

TABLE 6-17 **DataHi Register: Register 29, Select 3 (Data Cache)**

Bits	Size	Field	Description
[63:8]	56b	0	Read as zeros; ignored on write
[7:0]	8b	ECC	Cache Data ECC

6.6.3 Cache Coherency Attributes

Table 6-18 shows the Cache Coherency Attributes supported in SB-1. The "C Field" shown below is part of EntryLo0 and EntryLo1 registers in CP0 (Registers 2 and 3, Select 0). Refer to MIPS64 Specification for additional detail regarding CP0 Registers.

TABLE 6-18 SB-1 Cache Coherency Attributes

C(5:3)	Cache Coherency Attributes With Historical Usage	SB-1 Assignment
0	Available for implementation dependent use Historical usage: - Reserved (R4000®, VR5400, R10000®) - Unused, defaults to cached (R4300™) - Cacheable, noncoherent, write through, no write allocate (RC32364, RM5200)	Cacheable Coherent: Exclusive in L1, Uncacheable in L2. Similar to C = 4, but do not allocate in L2.
1	Available for implementation dependent use Historical usage: - Reserved (R4000) - Unused, defaults to cached (R4300) - Cacheable, noncoherent, write through, write allocate(RC32364, RM5200) - Cacheable write-through, write allocate (VR5400)	Cacheable Coherent: Shared in L1, Uncacheable in L2. Similar to C = 5, but do not allocate in L2.
2	Uncached Historical usage: - Uncached (all processors)	Uncached
3	Cacheable Historical usage: - Cacheable noncoherent (noncoherent) (R4000, R10000) - Cached (R4300) - Cacheable, noncoherent (writeback) (RC32364, RM5200) - Cacheable, writeback (VR5400)	Cacheable Noncoherent
4	Available for implementation dependent use Historical usage: - Cacheable coherent exclusive (exclusive) (R4000, R10000) - Unused, defaults to cached (R4300) - Reserved (RC32364, RM5200, VR5400)	Cacheable Coherent Exclusive Line is always fetched exclusive.
5	Available for implementation dependent use Historical usage: - Cacheable coherent exclusive on write (sharable) (R4000, R10000) - Unused, defaults to cached (R4300) - Reserved (RC32364, RM5200, VR5400)	Cacheable Coherent Sharable Line is fetched shared on a load miss, exclusive on a store miss. Line is upgraded to exclusive if it is fetched shared but no other processor has it.

TABLE 6-18 SB-1 Cache Coherency Attributes

C(5:3)	Cache Coherency Attributes With Historical Usage	SB-1 Assignment
6	Available for implementation dependent use Historical usage: - Cacheable coherent update on write (update) (R4000) - Unused, defaults to cached (R4300) - Reserved (RC32364, RM5200, R10000)	Not Used
7	Available for implementation dependent use Historical usage: - Reserved (R4000) - Unused, defaults to cached (R4300) - Reserved (RC32364, RM5200) - Uncached accelerated (VR5400, R10000)	Uncached Accelerated: Merge in Write Buffer

CHAPTER 7 Virtual Memory Address Space and the TLB Format

7.1 Introduction

This chapter elaborates on virtual memory address space and the TLB format in SB-1. The material presented in this chapter should be used in conjunction with The MIPS64 Privileged Resource Architecture in MIPS64 Specification.

7.2 Supported Memory Address Space in SB-1

Table 7-1 shows the MIPS64 virtual memory address space, as supported in SB-1. The supported physical address space in SB-1 is 40 bits wide and the virtual address space is 44 bits wide[1].

TABLE 7-1 Virtual Memory Address Space

$VA_{63..62}$	Segment Name(s)	Maximum Address Range	64-bit Address Enable	Associated with Mode	Reference Legal from Mode(s)	Actual Segment Size	Segment Type
11	kseg3	0xFFFF FFFF FFFF FFFF through 0xFFFF FFFF E000 0000	Always	Kernel	Kernel	2^{29} Bytes	32-bit Compatibility
	sseg ksseg	0xFFFF FFFF DFFF FFFF through 0xFFFF FFFF C000 0000	Always	Supervisor	Supervisor Kernel	2^{29} Bytes	32-bit Compatibility
	kseg1	0xFFFF FFFF BFFF FFFF through 0xFFFF FFFF A000 0000	Always	Kernel	Kernel	2^{29} Bytes	32-bit Compatibility
	kseg0	0xFFFF FFFF 9FFF FFFF through 0xFFFF FFFF 8000 0000	Always	Kernel	Kernel	2^{29} Bytes	32-bit Compatibility
	Address Error	0xFFFF FFFF 7FFF FFFF through 0xC000 0FFF 8000 0000	--	--	--	--	--
	xkseg	0xC000 0FFF 7FFF FFFF through 0xC000 0000 0000 0000	KX	Kernel	Kernel	$(2^{44} - 2^{31})$ Bytes	64-bit
10	xkphys	0xBFFF FFFF FFFF FFFF through 0x8000 0000 0000 0000	KX	Kernel	Kernel	8 x 2^{40} Byte regions within the 2^{62} Byte Segment	64-bit
01	***Address Error***	0x7FFF FFFF FFFF FFFF through 0x4000 1000 0000 0000	--	--	--	--	--
	xsseg xksseg	0x4000 0FFF FFFF FFFF through 0x4000 0000 0000 0000	SX	Supervisor	Supervisor Kernel	2^{44} Bytes	64-bit

TABLE 7-1 Virtual Memory Address Space

$VA_{63..62}$	Segment Name(s)	Maximum Address Range	64-bit Address Enable	Associated with Mode	Reference Legal from Mode(s)	Actual Segment Size	Segment Type
00	***Address Error***	0x3FFF FFFF FFFF FFFF through 0x0000 1000 FFFF FFFF	--	--	--	--	--
	xuseg xsuseg xkuseg	0x0000 0FFF FFFF FFFF through 0x0000 0000 8000 0000	UX	User	User Supervisor Kernel	$(2^{44} - 2^{31})$ Bytes	64-bit
	useg suseg kuseg	0x0000 0000 7FFF FFFF through 0x0000 0000 0000 0000	Always	User	User Supervisor Kernel	2^{31} Bytes	32-bit Compatibility

1. PABITS and SEGBITS, respectively, as referenced in the MIPS64 Specification

Figure 7-1 shows the virtual address space supported by SB-1.

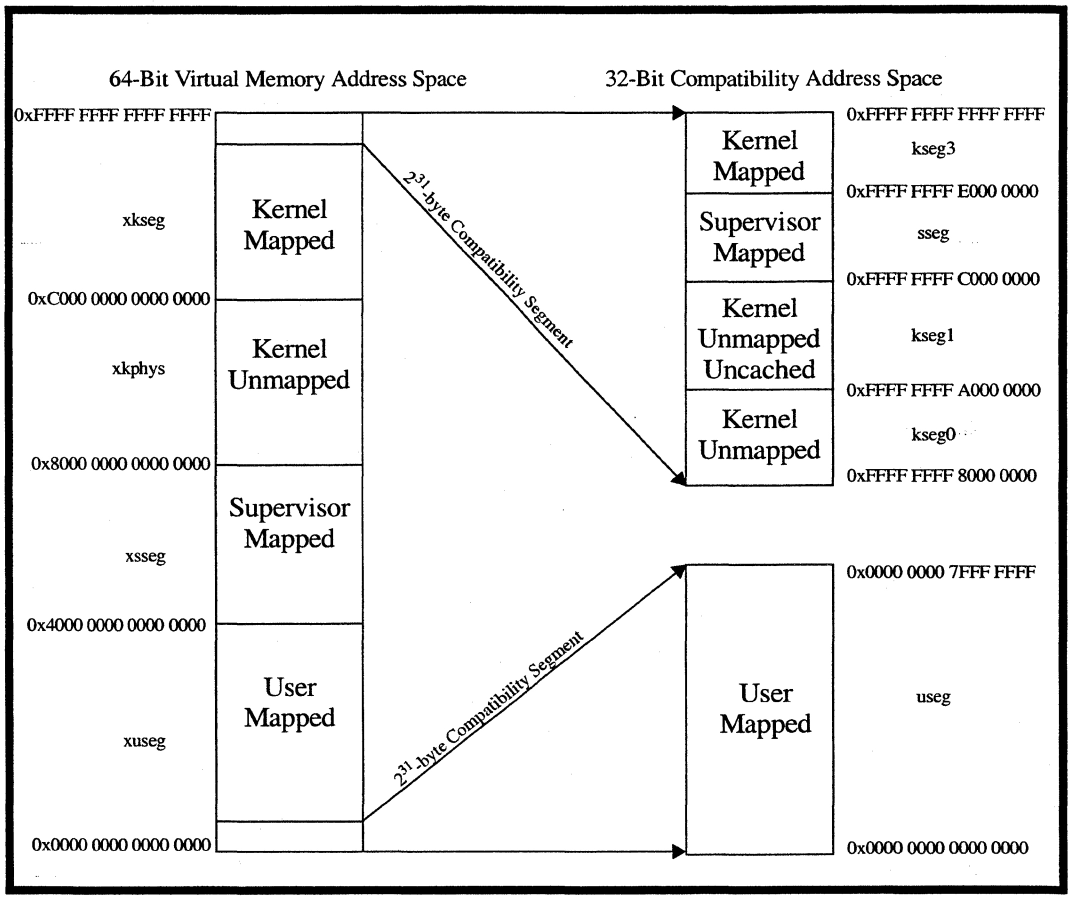

FIGURE 7-1 SB-1 Virtual Address Space

For reference purposes, Table 6-18 from Chapter 6 is repeated below to clarify the cache coherency attribute encoding (CCA field <61:59> of virtual address) used in constructing the full virtual address.

TABLE 7-2 **SB-1 Cache Coherency Attributes**

C(5:3)	Cache Coherency Attributes With Historical Usage	SB-1 Assignment
0	Available for implementation dependent use Historical usage: - Reserved (R4000®, VR5400, R10000®) - Unused, defaults to cached (R4300™) - Cacheable, noncoherent, write through, no write allocate (RC32364, RM5200)	Cacheable Coherent: Exclusive in L1, Uncacheable in L2. Similar to C = 4, but do not allocate in L2.
1	Available for implementation dependent use Historical usage: - Reserved (R4000) - Unused, defaults to cached (R4300) - Cacheable, noncoherent, write through, write allocate(RC32364, RM5200) - Cacheable write-through, write allocate (VR5400)	Cacheable Coherent: Shared in L1, Uncacheable in L2. Similar to C = 5, but do not allocate in L2.
2	Uncached Historical usage: - Uncached (all processors)	Uncached
3	Cacheable Historical usage: - Cacheable noncoherent (noncoherent) (R4000, R10000) - Cached (R4300) - Cacheable, noncoherent (writeback) (RC32364, RM5200) - Cacheable, writeback (VR5400)	Cacheable Noncoherent
4	Available for implementation dependent use Historical usage: - Cacheable coherent exclusive (exclusive) (R4000, R10000) - Unused, defaults to cached (R4300) - Reserved (RC32364, RM5200, VR5400)	Cacheable Coherent Exclusive Line is always fetched exclusive.
5	Available for implementation dependent use Historical usage: - Cacheable coherent exclusive on write (sharable) (R4000, R10000) - Unused, defaults to cached (R4300) - Reserved (RC32364, RM5200, VR5400)	Cacheable Coherent Sharable Line is fetched shared on a load miss, exclusive on a store miss. Line is upgraded to exclusive if it is fetched shared but there is no sharing.

TABLE 7-2 **SB-1 Cache Coherency Attributes**

C(5:3)	Cache Coherency Attributes With Historical Usage	SB-1 Assignment
6	Available for implementation dependent use Historical usage: - Cacheable coherent update on write (update) (R4000) - Unused, defaults to cached (R4300) - Reserved (RC32364, RM5200, R10000)	Not Used
7	Available for implementation dependent use Historical usage: - Reserved (R4000) - Unused, defaults to cached (R4300) - Reserved (RC32364, RM5200) - Uncached accelerated (VR5400, R10000)	Uncached Accelerated: Merge in Write Buffer

7.3 The TLB

Table 7-3 shows the organization of the Translation Lookaside Buffer in SB-1.

TABLE 7-3 **TLB Organization in SB-1**

Cache Feature	Specifics
Size	128 Entries, Arranged in 64 Rows of Even/Odd Pairs
Associativity	Full
Replacement Algorithm	Software Managed, with assist from Random
Parity	None
ECC Support	None

7.3.1 TLB Entry Format

Figure 7-2 shows the TLB Format supported in SB-1.

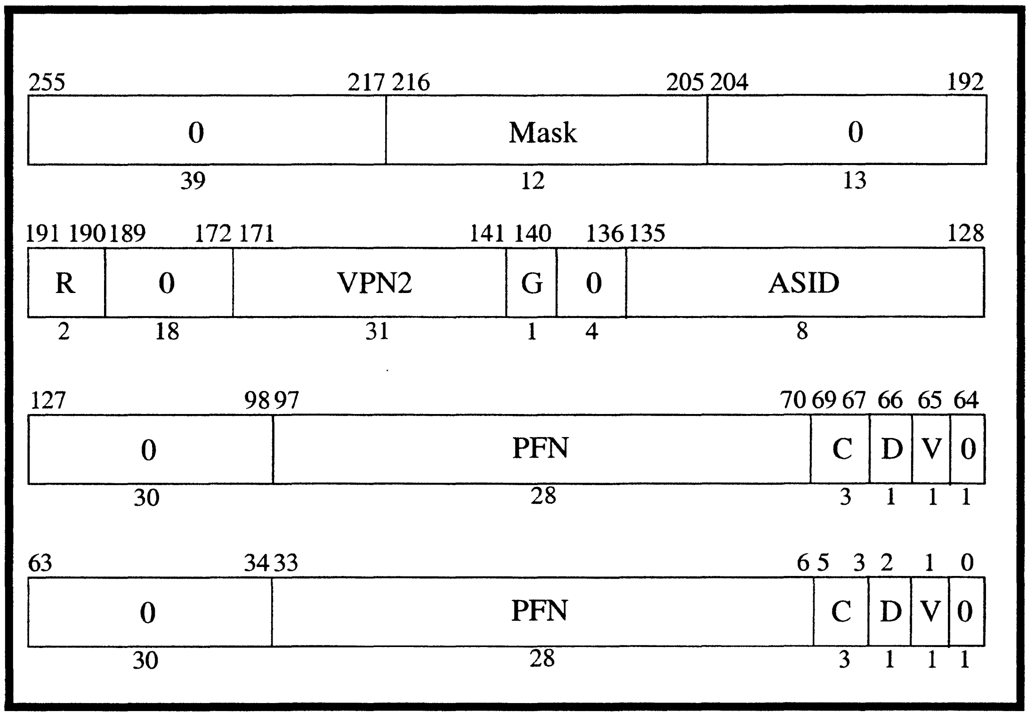

FIGURE 7-2 TLB Entry Format in SB-1

CHAPTER 8 The CP0 Architecture

8.1 Introduction

This chapter provides the list of CP0 registers supported in SB-1. The MIP64 Specification provides two sets of CP0 registers, Required and Optional. For details on Required registers, refer to the MIPS64 Specification. This chapter elaborates mainly on the list of Optional registers and optional fields within Required registers as supported in SB-1 core.

8.2 Overview of CP0 Registers

Table 8-1 provides a list of all CP0 Registers provided in SB-1.

TABLE 8-1 List of CP0 Registers in SB-1

Register Number	Sel	Register Name	Function	Compliance Level	Reference: MIPS64 Specification
0	0	Index	Index into the TLB entry	Required	Section 4.9.1, pg. 105
1	0	Random	Randomly generated index into the TLB array	Required	Section 4.9.2, pg. 106
2	0	EntryLo0	Low-order portion of the TLB entry for even-numbered virtual pages	Required	Section 4.9.3, pg. 107
3	0	EntryLo1	Low-order portion of the TLB entry for odd-numbered virtual pages	Required	Section 4.9.3, pg. 107

TABLE 8-1 List of CP0 Registers in SB-1

Register Number	Sel	Register Name	Function	Compliance Level	Reference: MIPS64 Specification
4	0	Context	Pointer to page table entry in memory	Required	Section 4.9.4, pg. 110
5	0	PageMask	Control for variable page size in TLB entries	Required	Section 4.9.5, pg. 111
6	0	Wired	Controls the number of fixed ("wired") TLB entries	Required	Section 4.9.6, pg. 112
7	all		Reserved for future extension	Reserved	
8	0	BadVAddr	Reports the address for the most recent address-related exception	Required	Section 4.9.7, pg. 113
9	0	Count	Processor cycle count	Required	Section 4.9.8, pg. 114
10	0	EntryHi	High-order portion of the TLB entry	Required	Section 4.9.9, pg. 114
11	0	Compare	Timer interrupt control	Required	Section 4.9.10, pg. 116
12	0	Status	Processor status and control	Required	Section 4.9.11, pg. 116
13	0	Cause	Cause of last general exception	Required	Section 4.9.12, pg. 123
14	0	EPC	Program counter at last exception	Required	Section 4.9.13, pg. 126
15	0	PRId	Processor identification and revision	Required	Section 4.9.14, pg. 127
16	0	Config	Configuration register	Required	Section 4.9.15, pg. 128
16	1	Config1	Configuration register 1	Required	Section 4.9.16, pg. 130
17	0	LLAddr	Load linked address	Implemented in SB-1	Section 4.9.17, pg. 132
18	0	WatchLo	Instruction Watchpoint address	Implemented in SB-1	Section 4.9.18, pg. 132 Also, refer to the Debug Architecture Chapter in this manual.
18	1	WatchLo	Data Watchpoint address	Implemented in SB-1	Section 4.9.18, pg. 132 Also, refer to the Debug Architecture Chapter in this manual.
19	0	WatchHi	Instruction Watchpoint control	Implemented in SB-1	Section 4.9.19, pg. 134 Also, refer to the Debug Architecture Chapter in this manual.
19	1	WatchHi	Data Watchpoint control	Implemented in SB-1	Section 4.9.19, pg. 134 Also, refer to the Debug Architecture Chapter in this manual.
20	0	XContext	Extended Addressing Page Table Context	Required	Section 4.9.20, pg. 135
21	all		Reserved for future extensions	Reserved	
22	all		Performance Event register	Implemented in SB-1	Section 4.9.21, pg. 136

TABLE 8-1 **List of CP0 Registers in SB-1**

Register Number	Sel	Register Name	Function	Compliance Level	Reference: MIPS64 Specification
23	0	Debug	EJTAG Debug register	Implemented in SB-1	EJTAG v2.5 Specification Also, refer to the Debug Architecture Chapter in this manual.
23	3	EDebug	Extended Debug register	Implemented in SB-1	EJTAG v2.5 Specification Also, refer to the Debug Architecture Chapter in this manual.
24	0	DEPC	Program counter at last EJTAG debug exception	Implemented in SB-1	EJTAG v2.5 Specification Also, refer to the Debug Architecture Chapter in this manual.
25	0-n	PerfCnt	Performance counter interface	Implemented in SB-1	Section 4.9.24, pg. 137 Also, refer to the Performance Monitoring Architecture Chapter in this manual.
26	0	ErrCtl	Parity/ECC error control and status	Implemented in SB-1	Section 4.9.25, pg. 140 Also, refer to the Error Handling Chapter in this manual.
26	1	BusErr_DPA	Data Bus Error Physical Address	Implemented in SB-1	Section 4.9.25, pg. 140 Also, refer to the Error Handling Chapter in this manual.
27	0	CacheErrI	Instruction Cache error control and status	Implemented in SB-1	Section 4.9.26, pg. 140 Also, refer to the Error Handling Chapter in this manual.
27	1	CacheErrD	Data Cache error control and status	Implemented in SB-1	Section 4.9.26, pg. 140 Also, refer to the Error Handling Chapter in this manual.
27	3	CacheErr_DPA	Data Cache Error Physical Address	Implemented in SB-1	Section 4.9.26, pg. 140 Also, refer to the Error Handling Chapter in this manual.
28	0	TagLoI	Low-order portion of instruction cache tag interface	Required	Section 4.9.27, pg. 142
28	1	DataLo	Low-order portion of cache data interface	Not Implemented in SB-1	Section 4.9.28, pg. 143
28	2	TagLoD	Low-order portion of data cache tag interface	Required	Section 4.9.27, pg. 142

TABLE 8-1 **List of CP0 Registers in SB-1**

Register Number	Sel	Register Name	Function	Compliance Level	Reference: MIPS64 Specification
29	0	TagHiI	High-order portion of instruction cache tag interface	Required	Section 4.9.29, pg. 143
29	1	DataHi	High-order portion of cache data interface	Not Implemented in SB-1	Section 4.9.30, pg. 144
29	2	TagHiD	High-order portion of data cache tag interface	Required	Section 4.9.29, pg. 143
30	0	ErrorEPC	Program counter at last error	Required	Section 4.9.31, pg. 144 Also, refer to the Error Handling Chapter in this manual.
31	0	DESAVE	EJTAG debug exception save register	Implemented in SB-1	EJTAG v2.5 Specification Also, refer to the Debug Architecture Chapter in this manual.

8.2.1 Processor Status and Control (Status, CP0 Register 12, sel0)

Figure 8-1 shows the SB-1 Status and Control Register.

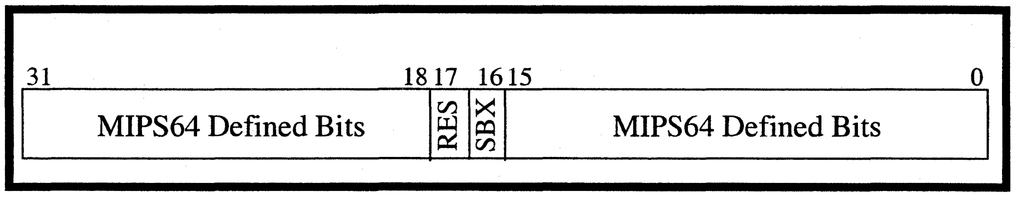

FIGURE 8-1 **SB-1 Status and Control Register**

The SBX bit allows the execution of SiByte specific extensions to the standard MIPS64 instruction set architecture. Upon reset, this bit is set to 0, disabling the execution of extended instructions. If an extended instruction is executed with this bit set to 0, the processor will generate a Reserved Instruction exception. The list of extended instructions in SB-1 follows:

MDMX: PAVG, PABSDIFF, PABSDIFFC. Refer to Chapter 3 for additional details.

Floating Point: DIV.PS, RECIP.PS, RSQRT.PS, SQRT.PS. Refer to Chapter 4 for additional details.

8.2.2 Processor Identification and Revision (PRId, CP0 Register 15, sel0)

Figure 8-2 shows the format of PRId register in MIP64 architecture.

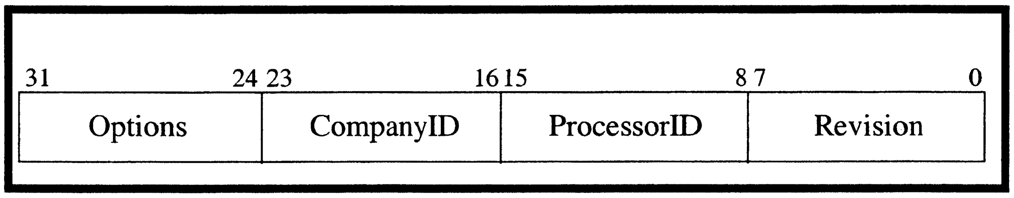

FIGURE 8-2 **PRId Register Format**

This is a 32 bit read-only register, factory preset, that contains information identifying the manufacturer, manufacturer options, processor identification and revision level of the processor. Table 8-2 presents the value of this register in SB-1.

TABLE 8-2 **PRId Register Fields in SB-1**

Field	**Description**	**Setting**
Revision	Specifies the revision number of the processor. This field allows software to distinguish between one revision and another of the same processor type.	**0x1** (Initial Value)
ProcessorID	Identifies the type of processor. This field allows software to distinguish between various processor implementations within a single company, and is qualified by the CompanyID field, described above. The combination of the CompanyID and ProcessorID fields creates a unique number assigned to each processor implementation.	**0x1**

TABLE 8-2 **PRId Register Fields in SB-1**

Field	Description	Setting
CompanyID	Identifies the company that designed or manufactured the processor. Software can distinguish a MIPS32 or MIPS64 processor from one implementing an earlier MIPS ISA by checking this field for zero. If it is nonzero the processor implements the MIPS32 or MIPS64 Architecture. Company IDs are assigned by MIPS Technologies when a MIPS32 or MIPS64 license is acquired. The encodings in this field are: 0x0: Not a MIPS32 or MIPS64 processor 0x1: MIPS Technologies, Inc. 0x4 : SiByte, Inc.	**0x4**
Company Options	Available to the designer or manufacturer of the processor for company-dependent options. The value in this field is not specified by the architecture.	**PRId<24> = MP Bit** PRId<24> = 0x0 for Uniprocessor PRId<24> = 0x1 for Multiprocessors **PRId<27:25> = Processor Number** 0x0 through 0x7, up to 8 processors If PRId<24> == 0, then the only valid value for PRId<27:25> is 0x0 **PRId<31:28> = SiByte Reserved**

8.2.3 Configuration Register (Config, CP0 Register 16, sel0)

Bits 16 through 30 of this register are reserved for implementation by MIPS64 ISA. SB-1 uses bits [19:16] of Config register to implement the multiprocessor vector offset bits, called MPV. This allows each processor to "shift" its exception block by 64KB, i.e. bits [19:16] of the exception vector are selected directly from this register field. At Reset, these bits are set to 0. These bits are read/write.

Figure 8-3 shows the bits taken by this field in the Config Register.

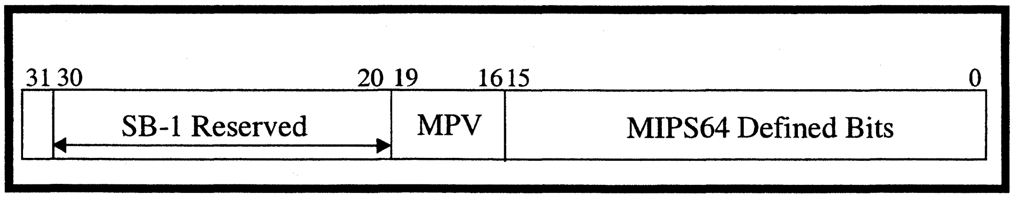

FIGURE 8-3 **MPV Field in SB-1 Config Register**

Exceptions that occur to the boot block, e.g. vectors with 0xBxxx_xxxx such as reset, NMI, debug, etc., are not affected by this offset, and as such, multiple copies do not need to be present in the ROM.

8.2.4 Load Linked Address (LLAddr, CP0 Register 17, sel0)

The *LLAddr* register contains relevant bits of the physical address read by the most recent Load Linked instruction.

This register is for diagnostic purposes only and serves no function during normal operation. The format of this register in SB-1 is shown in Figure 8-4.

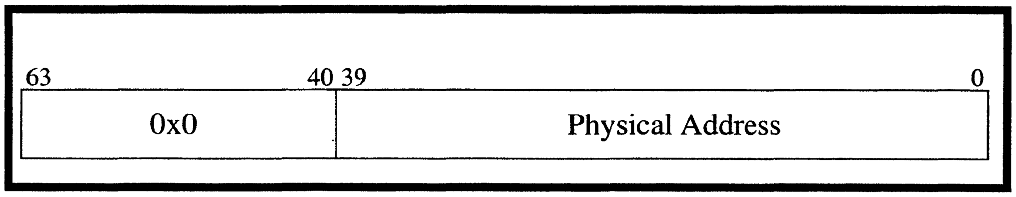

FIGURE 8-4 LLAddr Register Format in SB-1

Table 8-3 describes the *LLAddr* register fields.

TABLE 8-3 **LLAddr Register Field Descriptions**

Fields		Description	Read/Write	Reset State
Name	Bits			
PAddr	39:0	This field encodes the physical address read by the most recent Load Linked instruction.	R/W	Undefined
--	63:40	All Zeros	R/W	0x0

8.2.5 Watchpoint Address (WatchLo, CP0 Register 18, sel0-n)

For details, refer to the chapter on Debug Architecture in this manual.

8.2.6 Watchpoint Control (WatchHi, CP0 Register 19, sel0-n)

For details, refer to the chapter on Debug Architecture in this manual.

8.2.7 EJTAG Debug Register (Debug, CP0 Register 23, sel0)

For details, refer to the chapter on Debug Architecture in this manual.

8.2.8 Program Counter at Last EJTAG Debug Exception (DEPC, CP0 Register 24, sel0)

For details, refer to the chapter on Debug Architecture in this manual.

8.2.9 Performance Counter Interface (PerfCnt, CP0 Register 25, sel0)

For details, refer to the chapter on Performance Monitoring Architecture in this manual.

8.2.10 Parity/ECC Error Control and Status (ErrCtl, CP0 Register 26, sel0)

For details, refer to the chapter on Error Handling in this manual.

8.2.11 Cache Error Control and Status (CacheErr, CP0 Register 27, sel0-3)

For details, refer to the chapter on Error Handling in this manual.

8.2.12 Low-order Portion of Cache Data Interface (DataLo, CP0 Register 28, sel1)

This CP0 register is not implemented in SB-1.

8.2.13 High-order Portion of Cache Data Interface (DataHi, CP0 Register 29, sel1)

This CP0 register is not implemented in SB-1.

8.2.14 EJTAG Debug Exception Save Register (DESAVE, CP0 Register 31, sel0)

For details, refer to the chapter on Debug Architecture in this manual.

8.3 Privileged Resource Hazards

This section details the hazards surrounding the SB-1 privileged resources. Specifically, the use of privileged resources, such as CP0 registers, the TLB, and cache state, and the execution of privileged instructions, such as MTC0 and MFC0, are covered.

8.3.1 Privileged Resources and Instructions

The SB-1 privileged resources include the Coprocessor 0 registers, the Translation Lookaside Buffer, and the Instruction and Data Cache Tags.

Some CP0 registers serve as the interface between software and the hardware resource. For TLB access, the following CP0 registers are used: Index (0), Random (1), EntryLo0 (2), EntryLo1 (3), PageMask (5), and EntryHi (10). For cache tag access, the TagLo-I (28, sel. 0) and TagHi-I (29, sel. 0) interface with the instruction cache, and TagLo-D (28, sel. 2) and TagHi-D (29, sel. 2) interface to the data cache.

The following table outlines the resources required by the privileged instructions defined by the MIPS64 ISA. The Inst column indicates the type of instruction, while the Source and Destination columns list the required resources and the resources updated by the instruction.

TABLE 8-4 Resources Required by MIPS64 Privileged Instructions

Inst	Source	Destination
CACHE	TLB, Cache Tags, TagLo/TagHi	Cache Tags, TagLo/TagHi
ERET	Status, EPC, ErrorEPC	Status, PC
DERET	Debug, DEPC	Debug, PC
DMFC0/MFC0	CP0 Register	
DMTC0/MTC0		CP0 Register
MTC0 EntryHi		TLB, EntryHi
TLBP	TLB, EntryHi	Index
TLBR	TLB, Index	TLB, PageMask, EntryHi/EntryLo
TLBWI	Index, PageMask, EntryHi/EntryLo	TLB
TLBWR	Random, PageMask, EntryHi/EntryLo	TLB

In addition, some processor activities implicitly require privileged resources to operate. These are listed below:

TABLE 8-5 Processor Activities Requiring Privileged Resources

Action	Source	Destination
Inst Fetches	Cache Tags (TLB on miss)	
Loads	TLB, Cache Tags	
Stores	TLB, Cache Tags	

8.3.2 Privileged Resource Hazards

Whenever an instruction writes a result to a resource required by a subsequent instruction or action, a hazard exists. This is commonly known as a Read-After-Write (RAW) hazard. (Other types of hazards, such as WAW and WAR, are not visible in the SB-1's implementation of the privileged resources.) In the SB-1, all privileged instructions and memory actions such as loads and stores are interlocked by serializing the dependent operations. As a result, no SSNOPs are required between these types of operations.

The privileged instructions and other operations can be classified into several groups, as shown below:

TABLE 8-6 Operation Grouping of Privileged and Miscellaneous CPU Operations

Group	Instruction/Action	Description
TLBOp	TLBWI, TLBWR, TLBR, TLBP	TLB Instructions
EntryHi	MTC0 CP0.EntryHi	Move to CP0.EntryHi
Cache	CACHE	Cache Operation Instructions
CP0Wr	MTC0, DMTC0	CP0 Register Write Operations
CP0Rd	MFC0, DMFC0	CP0 Register Read Operations
MemOp	Loads and Stores	Load and Store Operations

Note that in this classification, a move to the CP0.EntryHi register is a special case due to its effects on the implementation, namely the TLB.

The following matrix outlines the implemented interlocks for each pair of operation groups. To eliminate the hazards, the SB-1 effectively serializes the instructions from each group that may result in a hazard. Note that no hazards exist for a group followed by a CP0Wr or EntryHi (these are WA* hazards).

TABLE 8-7 **Implemented Interlocks for Each Pair of Operation Groups**[a]

First/Second	TLBOp	Cache	CP0Rd	MemOp
TLBOp	HW Interlock	HW Interlock	HW Interlock	HW Interlock
EntryHi	HW Interlock	HW Interlock	HW Interlock	HW Interlock
Cache	HW Interlock	HW Interlock	HW Interlock	HW Interlock
CP0Wr	HW Interlock	HW Interlock	HW Interlock	HW Interlock
CP0Rd	**CP0 Hazard**	HW Interlock	**CP0 Hazard**	**CP0 Hazard**
MemOp	**CP0 Hazard**	HW Interlock	**CP0 Hazard**	**CP0 Hazard**

a. Hardware interlock implies serialization

Instruction cache operations are not interlocked with respect to the instruction fetch. As a result, care must be taken when executing these operations since they may have unpredictable results. It is recommended that instruction cache operations only be executed from uncacheable space.

8.3.3 CP0 Register Side-Effects

Although the above operations are interlocked, side-effects of writing CP0 registers are not interlocked. In general, the CP0 write takes effect in stage 8 of the pipeline; however, the usage of the CP0 data may take place earlier in the pipeline. Thus, there is a "shadow" of some number of instructions between the update and the update's being observed. These hazards are divided into fetch hazards and execution hazards.

(Note that updates of CP0 state that are invisible to software, such as exceptions, or that are implicit in the execution of certain instructions, such as an ERET, are handled by the hardware, so no SSNOPs are required for proper operation.)

8.3.3.1 Fetch Hazards

Because the instruction fetch is decoupled from the execution of instructions, there is no way to guarantee the timing between the CP0 register write and the instruction fetch. To eliminate this type of hazard, an ERET instruction must be executed between the CP0 write and the first instruction that should observe the update. In the SB-1, there are no cases that require placing SSNOPs before the ERET instruction.

Writes to CP0 registers that affect instruction fetch are listed below by register:

TABLE 8-8 **CP0 Registers that Affect Instruction Fetch**

Register	Field	Action
EntryHi	ASID	Inst Cache Lookup/Inst Watch Exception
Status	CU	Coprocessor Usable Exceptions
	RE	Instruction Fetch Endinanness
	MX	MDMX Usable Exceptions
	PX	Rsvd Inst for 64b User Instructions
	KX/SX/UX	Address Error/TLB Refill Exceptions
	KSU	Address Error Exceptions
	EXL/ERL	Address Error/Inst Watch Exceptions
Config	K0	Instruction Cache Lookup
WatchLo-I	All	Instruction Watch Exceptions
WatchHi-I	All	Instruction Watch Exceptions

In addition, installing a new TLB entry for the instruction fetch requires executing an ERET to ensure that the TLB state has been updated properly.

8.3.3.2 Execution Hazards

Execution hazards occur between the time of the CP0 register write and the time of the CP0 register use after the issue stage. The following table indicates when certain instructions or actions require and/or generate CP0 state. Note that TLB operations and registers, as well as CACHE instructions, are omitted since they are fully interlocked.

TABLE 8-9 **Required/Generated CP0 States by SB-1 Instructions and Activities**

	Operands		Results	
Inst/Event	**What**	**Pipestage**	**What**	**Pipestage**
MTC0/DMTC0			CP0 Register	8
MFC0/DMFC0	CP0 Register	Interlock		
Load/Store	EntryHi.ASID	Interlock		
	Status.RE	Interlock		
	Status.KX/SX/UX	Interlock		
	Status.KSU	Interlock		
	Status.ERL	Interlock		

TABLE 8-9 Required/Generated CP0 States by SB-1 Instructions and Activities

	Operands		Results	
Inst/Event	**What**	**Pipestage**	**What**	**Pipestage**
	Status.EXL	Interlock		
	Config.K0	Interlock		
	WatchLo-D	Interlock		
	WatchHi-D	Interlock		
FP 64b Registers	Status.FR	0		
SiByte Ext	Status.SBX	0		
Exception	Status.BEV	7	Status.ERL	8
	Cause.IV	7	Status.EXL	8
	Config.MPV	7	Cause.BD	10
			Cause.CE	8
			Cause.ExcCode	8
			EPC	10
			ErrorEPC	10
Mem Exception (including above)			Context	8
			BadVAddr	8/10
			EntryHi	8
			XContext	8
Interrupt	Status.IM	4	See Exception Above	
	Status.ERL	4		
	Status.EXL	4		
	Status.IE	4		
	Cause.IP	4		
Timer Int	Count	3	Cause.IP7	4
	Compare	3		
Def Watch Exc (Data)	Status.EXL	4	Cause.WP	8
	Status.ERL	4	See Exception Above	
	Cause.WP	4		
ERET	EPC	7	Status.ERL	8
	ErrorEPC	7	Status.EXL	8
Debug Exception			Debug.DBD	10
			Debug.DM	8

TABLE 8-9 **Required/Generated CP0 States by SB-1 Instructions and Activities**

	Operands		**Results**	
Inst/Event	**What**	**Pipestage**	**What**	**Pipestage**
			Debug.IEXI	8
			Debug.DExcCode	8
			Debug.DINT	8
			Debug.DIB	8
			Debug.DDBS	8
			Debug.DDBL	8
			Debug.DBp	8
			Debug.DSS	8
			DEPC	10
Impr Debug Exc	Debug.MCheckEP	4	See Debug Exception Above	
	Debug.CacheEP	4		
	Debug.DBusEP	4		
	Debug.IEXI	4		
DERET	DEPC	7	Debug.DM	8
			Debug.IEXI	8

The pipestages for each event can be used to calculate the required separation (in cycles) between two events. Note that the SB-1 is a superscalar machine, so cycles do not necessarily equal instructions. To make the number of instructions equal the number of cycles, the SSNOP instruction can be used since this instruction forces single-issue for itself.

To calculate the separation, the following formula can be used:

Separation = Pipestage(Result) - Pipestage(Operand) - 1

For example, a MTC0 instruction that modifies the RE bit must occur a certain number of cycles before a subsequent load or store. The number of cycles, or SSNOPs, is 8 (MTC0 write) - 0 (Load/Store use) - 1, or 7. Likewise, a MTC0 that enables interrupts may cause an interrupt to be taken on an instruction issued four cycles later:

Separation = 8 (MTC0 write) - 4 (IE use) - 1 = 3

```
T        MTC0  r1, CP0.Status
```

T+1	SSNOP
T+2	SSNOP
T+3	SSNOP
T+4	(interrupt seen here)

In some cases, the separation may be 0 cycles, such as between a MTC0 to the EPC register followed by an ERET. In the SB-1, privileged instructions are always the oldest instruction issued in a particular cycle. As a result, privileged instructions are effectively pipelined, so SSNOPs do not need to be used to ensure that two privileged instructions are issued in different cycles. With this behavior and with interlocks, privileged instruction sequences need not include intervening instructions.

The cycle separation calculated by using the above table indicates the maximum shadow resulting from a particular pair of operations. Use of the dependent operation within that shadow may result in UNPREDICTABLE behavior.

CHAPTER 9 The Debug Architecture

9.1 Introduction

This document covers the SB-1 core debug implementation. The debug features of the SB-1 are mostly software in nature and allow customers programming or integrating the SB-1 core to debug their software and hardware.

9.2 Debug Features

The SB-1 core provides customers several debug features that aid in the development of hardware and software systems. Included in the basic SB-1 debug functionality are MIPS64 compliant Watch registers and a MIPS compliant EJTAG subset. To provide additional debug flexibility, an enhanced debug mode is implemented which includes an external signaling interface and an alternate debug vector which enables loading code over the TAP.

9.2.1 Watch Registers

Two Watch register pairs (WatchHi and WatchLo) are implemented by the SB-1 core to trap on software-specified addresses. The first pair, selected when sel equals zero, can be used to break on instruction addresses while the second pair, selected when sel equals one, traps load or store accesses.

The Watch registers are implemented as specified in the MIPS64 document with the following exceptions. The WatchLo register, corresponding to select zero, always reads bits [1:0] as zero and ignores writes to those bits, because they are used exclusively for instruction references. Likewise, bit [2] of WatchLo register one is always read zero and ignored on writes, and bits [1:0] are used as enables since that register corresponds to data references only.

TABLE 9-1 WatchLo/Hi Register Specifics

Bits	Field	Description	Access	Reset
WatchLo, Sel = 0				
[63:3]	VAddr	Instruction Virtual Address	R/W	X
[2]	I	Instruction Watch Enable	R/W	0
[1:0]	0	Reserved	R	0
WatchHi, Sel = 0				
[31]	M	More Watch Pairs Implemented	R	1
[30]	G	Global Bit	R/W	X
[29:24]	0	Reserved	R	0
[23:16]	ASID	Application Space ID	R/W	X
[15:12]	0	Reserved	R	0
[11:3]	Mask	Mask Bits	R/W	X
[2:0]	0	Reserved	R	0
WatchLo, Sel = 1				
[63:3]	VAddr	Data Virtual Address	R/W	X
[2]	0	Reserved	R	0
[1]	R	Load Watch Enable	R/W	0
[0]	W	Store Watch Enable	R/W	0
WatchHi, Sel = 1				
[31]	M	More Watch Pairs Implemented	R	0
[30]	G	Global Bit	R/W	X
[29:24]	0	Reserved	R	0
[23:16]	ASID	Application Space ID	R/W	X
[15:12]	0	Reserved	R	0
[11:3]	Mask	Mask Bits	R/W	X
[2:0]	0	Reserved	R	0

9.3 EJTAG

The SB-1 core features a compliant subset of the MIPS EJTAG, Version 2.5, functionality. EJTAG extends the operating modes, the ISA, and the CP0 registers of a MIPS processor.

In addition to the normal kernel, supervisor, and user modes, the EJTAG specification defines a special debug mode. In debug mode, several types of debug exceptions may be serviced, including single step instruction breaks and debug interrupts signalled by the external agent via the DINT pin. The debug mode may also be entered via the Software Debug Breakpoint instruction (SDBBP).

EJTAG defines several CP0 registers to hold debug state when a debug exception is encountered. The Debug register contains information about how the debug handler was entered, while the DEPC register holds the PC of the instruction that was executing when the debug exception occurred. To provide consistent state between debug exceptions, the DESAVE register is implemented and acts as a scratch register for the debug handler. When the handler is complete, a DERET instruction is executed, resuming the original program at the PC stored in DEPC.

The EJTAG spec outlines the behavior of debug mode and single step instruction break (which is enabled for all modes except debug when the SSt bit is set) as well as the register definitions for the extended CP0 registers. Since the SB-1 does not implement some EJTAG features, the Debug register is defined as follows (for an explanation of EDM, see the next section):

TABLE 9-2 Debug Register: CP0 Register 23, Sel = 0, EDM = 0

Bits	Field	Description	Access	Reset
[31]	DBD	Debug Branch Delay Slot	R	X
[30]	DM	Debug Mode Status	R	0
[29]	NoDCR	No Debug Control Register	R	1
[28]	LSNM	Load/Store Normal Memory[a]	R	0
[27]	Doze	Doze Status	R	0
[26]	Halt	Halt Status	R	0
[25]	CountDM	Count Register in Debug Mode	R	0
[24]	0	Reserved	R	0
[23]	MCheckP	Machine Check Exception Pending	R/W1[b]	0
[22]	CacheEP	Cache Error Exception Pending	R/W1[b]	0
[21]	BusDP	Bus Error Exception Pending	R/W1[b]	0
[20]	IEXI	Imprecise Error Exception Inhibit	R/W	0
[19]	DDBSImpr	Debug Data Brk St Impr Stat[a]	R	0

TABLE 9-2 **Debug Register: CP0 Register 23, Sel = 0, EDM = 0**

Bits	Field	Description	Access	Reset
[18]	DDBLImpr	Debug Data Brk Ld Impr Stat[a]	R	0
[17:15]	EJTAGver	EJTAG Version (Version 2.5)	R	001
[14:10]	DExcCode	Debug Exception Code	R	X
[9]	NoSSt	No Single Step Implemented	R	0
[8]	SSt	Single Step Enable	R/W	0
[7:6]	Rsvd	Reserved	R	0
[5]	DINT	Debug Interrupt Status	R	X
[4]	DIB	Debug Instruction Break Exception Status[c]	R	0
[3]	DDBS	Debug Data Break Store Exception Status[c]	R	0
[2]	DDBL	Debug Data Break Load Exception Status[c]	R	0
[1]	DBp	Debug Breakpoint Exception Status	R	X
[0]	DSS	Debug Single Step Exception Status	R	X

a. These Debug Register bits are forced to 0 since the EJTAG memory region and break registers are not implemented
b. R/W1 indicates that software may read the state of the bit but can only modify it by writing a one.
c. These bits are always read as zero in standard debug mode, but when extended debug mode is enabled, they are used to indicate watch exception conditions. See below.

9.4 Extended Debug Mode

In addition to EJTAG, the SB-1 core implements a SiByte defined extended debug mode to enhance the processor's debug capabilities. Extended debug mode uses the existing EJTAG and Watch registers and defines an extended debug mode register (EDebug) that controls the additional debug features. Addressed via the Debug register number with select equal to 3, the EDebug register contains the following fields:

TABLE 9-3 **EDebug Register(CP0 Register 23, Sel = 3)**

Bits	Field	Description	Access	Reset
[31]	EDM	Extended Debug Mode Enable	R/W	0
[30]	AltVec	Alternate Debug Vector Select	R/W	a
[29:9]	Rsvd	Reserved	R	0
[8]	SStPrv	SSt Enable in Privileged Modes	R/W	X
[7:6]	Rsvd	Reserved	R	0
[5]	EDE	Extended Debug Event	R/W	0
[4]	EDIWT	EDIW triggers EDE	R/W	X
[3]	EDDWST	EDDWS triggers EDE	R/W	X
[2]	EDDWLT	EDDWL triggers EDE	R/W	X
[1:0]	Rsvd	Reserved	R	0

a. See discussion about the DBBOOT signal below

Extended debug mode is controlled through the EDM bit in the EDebug register. When EDM is enabled, the Watch register pairs are used to generate debug exceptions rather than normal Watch exceptions. As a result, Watch register matches may no longer be deferred (the WP bit will never be set) and cause entry into the debug handler if the core is not already in debug mode. Upon entering debug mode, bits [4:2] of the Debug register are set depending on the type of Watch register match:

TABLE 9-4 **Debug, Sel = 0, EDM = 1**

Bits	Field	Description	Access	Reset
[4]	EDIW	Extended Debug Instruction Watch Status	R	X
[3]	EDDWS	Extended Debug Data Watch St Status	R	X
[2]	EDDWL	Extended Debug Data Watch Ld Status	R	X

The EDebug register also allows software to select an alternate exception vector for debug exceptions. When the AltVec bit is set in the EDebug register, the processor jumps to instruction address 0xB000_0480 instead of the normal EJTAG debug exception vector. An external agent, like the SB-1250 SCD, may map the memory at the physical address that results (0x00_1000_0480) to a JTAG probe so the debug handler and data can be delivered by the probe. Servicing the debug exception through the probe allows flexible implementation of the handler.

Extended debug mode enables software to control single step more finely. The SStPrv bit enables single step in non-user modes (kernel, supervisor, EXL, and ERL) and may be cleared when EDM is enabled so single step is only active for user mode software. If EDM is disabled, or SSt is off, the state of this bit has no effect on single step.

Finally, the EDE bit in the EDebug register allows software and certain hardware events to signal that a debug event has occurred. The state of this bit is reflected in the EDEN signal, an SB-1 core output, which may be driven to an external agent to trigger an outside action. The EDE bit is always settable from software; however, if EDM is enabled, watchpoint events may set the bit as well, as long as the corresponding trigger bits are set (bits [4:2] of the EDebug register). Clearing the EDE bit can only be done in software.

Debug mode can be entered directly from reset if the DBBOOT signal is asserted during reset. This signal forces the SB-1 core to begin fetching from the alternate debug vector in debug mode when reset is deasserted. The table below indicates what state is set by the DBBOOT signal during reset:

TABLE 9-5 Debug Reset Behavior

DBBOOT	DM	AltVec	Vector
0	0	0	0xBFC0_0000
1	1	1	0xB000_0480

DBBOOT can also be used to force the state of the AltVec bit in the EDebug register. Asserting DBBOOT during normal operation will set the AltVec bit but will not cause the processor to enter debug mode (DINT must be used if entry into debug mode is desired.) The bit can only be cleared by software, and the DBBOOT signal must be deasserted for the clearing write to take effect.

Enabling some features in extended debug mode places certain restrictions on software. Because EDM uses the Watch registers as breakpoints, setting EDM when the Cause[WP] bit is one results in UNDEFINED processor behavior. In addition, the behavior of the processor is UNDEFINED if the EDM and SStKS bits are modified while not in debug mode, since single step behavior cannot be guaranteed. Software should check or clear the WP bit before setting the EDM bit, and handlers should be careful not to modify the state of the EDM and SStKS bits outside of debug mode. The EDE and AltVec bits may, however, be modified as long as the previous restrictions are honored.

9.5 Debug Signal Pins

The following table summarizes the external debug signals implemented on the SB-1 core:

TABLE 9-6 Debug Signal Pins

Name	I/O	Description
DINT	I	Debug Interrupt: causes the processor to take a debug exception and enter debug mode
DBBOOT	I	Debug Boot: forces the processor into debug mode after reset and initiates the instruction fetch from the alternate debug vector. Sets AltVec immediately. On Reset, causes immediate entry to DM at Alternate Vector.
EDEN	O	Extended Debug Event Notification: asserts to notify an external agent of a core debug event; initiated by the debug handler or hardware watchpoints

DBBOOT can also be used to force the state of the AltVec bit in the EDebug register. Asserting DBBOOT during normal operation will set the AltVec bit but will not cause the processor to enter debug mode. (DINT must be used if entry into debug mode is desired.) The bit can only be cleared by software, and the DBBOOT signal must be deasserted for the clearing write to take effect.

CHAPTER 10 Error Handling

10.1 Introduction

This chapter covers the error handling capabilities of the SB-1 core. Errors are classified into several groups depending on their relationship to the instruction stream. Each of these classes are defined here:

Precise - An exception is precise if the EPC or ErrorEPC indicates the PC of the exact instruction that caused or detected the error.

Imprecise - An exception is imprecise if the EPC or ErrorEPC indicates the PC of an instruction that cannot be guaranteed to have caused or detected the error.

Deterministic - A deterministic error is imprecise but the instruction that caused or detected the error can be determined by interpreting the program flow from the instruction indicated by the EPC or ErrorEPC. In general, instructions that cause or detect deterministic errors are located within four instructions of that PC.

Recoverable - A recoverable error is imprecise, but the instruction stream may be restarted from the instruction indicated by the EPC or ErrorEPC. In addition, recoverable errors are corrected by hardware, which completes the correction before the error is signaled.

Table 10-1 classifies the type of errors that are detectable in the SB-1 core and describes the type of exception taken by each:

TABLE 10-1 **SB-1 Error Types**

Error Type	Exception Type
Instruction Cache	
Instruction Cache Tag Address Parity Error	Cache Error
Instruction Cache Data Parity Error	Cache Error
Data Cache	
Data Cache Tag State Parity Error	Cache Error
Data Cache Tag Address Parity Error	Cache Error
Data Cache Data Single-Bit ECC Error	Cache Error
Data Cache Data Double-Bit ECC Error	Cache Error
TLB	
TLB Multiple Entry Match (TLB Shutdown)	Machine Check
BIU	
External Cache/Memory Error (external cache miss request ends in an uncorrectable parity/ECC error response)	Cache Error
External Cache/Memory Error (external cache miss request ends in an uncorrectable parity/ECC error response)	Cache Error
External Bus Error (external request ends in a bus error response)	Bus Error
Duplicate Tag State Parity Error	Cache Error
Duplicate Tag Address Parity Error	Cache Error
General	
Time Out Counter Expiration	Machine Check

All other errors are undetectable and may manifest themselves in unpredictable processor behavior. The subsequent sections detail the various error detecting and correcting properties of the different units on the CPU.

10.2 Instruction Cache

In general, the instruction cache is parity protected in both the tag and data arrays. The tag fields and their protection are listed in Table 10-2.

TABLE 10-2 **Instruction Cache Tag Field Protection**

Field	TagLo Bits	Size	Protection	TagLo Bit
R	[63:62]	2b	Parity Bit (P1)	[11]
VTag1	[43:25]	19b	Parity Bit (P1)	[11]
VTag0	[24:13]	12b	Parity Bit (P0)	[10]
G	[8]	1b	Parity Bit (P0)	[10]
ASID	[7:0]	8b	Parity Bit (P0)	[10]
V	[29]	1b	Parity Bit (P)	[27]

Even parity is implemented in the instruction cache tag; that is, the number of ones in the protected data and the parity bit is an even number. The valid (V) bit is covered by a single parity bit (P). In the rest of the tag, P0 covers the ASID, the G bit, and bits [24:13] of the address. P1 covers bits [43:25] of the address and the region bits.

The instructions and the predecode bits are protected by a parity bit for every byte of data. As a result, one cacheline contains 32b of parity for the instructions and 4b of parity to cover the predecode bits (one bit of predecode parity covers two instructions' predecode bits). Again, the parity for the data array is even parity, although this parity calculation is not visible to the user.

Note that with a parity protection scheme, single bit errors can be detected, but not corrected, by the hardware. Also, some double bit errors cannot be detected.

10.2.1 Implementation Notes:

The valid bit parity calculation factors into the hit signal. As a result, the V and P bits must match for the line to be valid in the cache. Tag address parity errors are calculated for each way and are signaled if any of the ways in the index contains an error, regardless of the hit/miss determination. Instruction and predecode parity errors are only reported if there is an error in the instructions being fetched.

The LRU bits are not parity protected, but invalid combinations are flushed to a known valid format when an invalid state is detected. Errors detected in the valid and parity bits are never reported, although they are scrubbed to the invalid state (V==0, P==0).

Table 10-3 indicates the types of errors that can occur at different stages of an instruction access.

TABLE 10-3 Instruction Access Error Types

Action	Error-Exception	Type
Icache Lookup	Tag Address Parity-Cache Error	Precise
Icache Hit	Data Parity-Cache Error	Precise
Icache Miss	External Cache Error	Precise
	External Bus Error	Precise

During an instruction cache miss, uncorrectable errors signaled on the return of the cacheline cause a cache or bus error exception to be taken. In either case, the data are not filled into the cache.

10.3 Data Cache

The Data Cache address in the tag array is protected by even parity, as shown in Table 10-4.

TABLE 10-4 Data Cache Tag Protection

Field	TagLo Bits	Size	Protection	TagLo Bit
PTag1	[39:26]	14b	Parity Bit (P1)	[11]
PTag0	[25:13]	13b	Parity Bit (P0)	[10]

The state bits in the tag array are protected by a sparse encoding. This encoding is detailed in TagLo/TagHi descriptions in Chapter 6.

In the data array, error correcting code (ECC) is implemented to correct single bit data errors and detect double bit errors. Both types of errors signal exceptions, although the single bit errors are corrected by hardware before the error handler is invoked. Single bit errors are imprecise, but the program may be restarted from the instruction address stored in the ErrorEPC register. An exception is taken on single bit errors so that software may log the error if desired.

Double bit ECC errors are imprecise, but for load hits, the instruction that caused the error can be determined by interpreting the instruction stream from the instruction address stored in the ErrorEPC register. In general, the load instruction that caused the error is within four instructions in the dynamic instruction stream.

10.3.1 Implementation Notes

Data cache errors may be detected by speculative loads, although the errors themselves will be reported on non-speculative instructions.

The LRU bits are not parity protected, but invalid combinations are flushed to a known valid format when an invalid state is detected. In addition, the stream bit in the tag array is not protected, so errors in this bit are not detected.

Table 10-5 indicates the types of errors that can occur at different stages of a load access:

TABLE 10-5 Load Errors

Action	Error-Exception	Type
Load Lookup	Tag State Parity-Cache Error	Deterministic
	Tag Address Parity-Cache Error	Deterministic
Load Hit	Single Bit ECC-Cache Error	Recoverable
	Double Bit ECC-Cache Error	Deterministic
Load Miss	External Cache Error	Imprecise
	External Bus Error	Imprecise

Table 10-6 indicates the types of errors that can occur at different stages of a store access, with the final column indicating whether the store data are written into the data cache.

TABLE 10-6 Store Errors

Action	Error-Exception	Type	Data Written
Store Lookup	Tag State Parity-Cache Error	Imprecise	No
	Tag Address Parity-Cache Error	Imprecise	No
Store Hit	Single Bit ECC-Cache Error	Recoverable	Yes
	Double Bit ECC-Cache Error	Imprecise	No
Store Miss	External Cache Error	Imprecise	No
	External Bus Error	Imprecise	No

During a data cache miss, uncorrectable errors signaled on the return of the cacheline cause either a cache or bus error exception to be taken. In both cases, the fill proceeds as it normally would, i.e. the data and address are written, but the tag state is marked with an error code. Note that this behavior may result in additional cache errors if the cache is subsequently accessed.

A fill may result in the eviction of a cacheline. If that cacheline contains an error, the processor may inhibit the writeback or complete the writeback by signaling an error during the data phase of the bus. Table 10-7 indicates what happens in each case:

TABLE 10-7 **Cacheline Errors due to Evicts**

Error-Exception	**Type**	**Processor Behavior**
Tag State Parity-Cache Error	Imprecise	Inhibit Writeback
Tag Address Parity-Cache Error	Imprecise	Inhibit Writeback
Single Bit ECC-Cache Error	Recoverable	Writeback-Data Corrected
Double Bit ECC-Cache Error	Imprecise	Writeback-Data Uncorrected

Finally, the data cache is checked for errors on coherency requests, and both invalidate and intervention requests cause the processor to take a cache error exception if the processor detects an error in any of the tags at the target index. Invalidates that detect tag errors do not change the tag state, and interventions always reply to the requestor with an error indication, as shown in Table 10-8.

TABLE 10-8 **Error-Exception Types and Interventions**

Error-Exception	**Type**	**Invalidate**	**Intervention**
Tag State Parity	Imprecise	Error Only	Reply-Tag Uncorrected
Tag Address Parity	Imprecise	Error Only	Reply-Tag Uncorrected
Single Bit ECC	Recoverable	N/A	Reply-Data Corrected
Double Bit ECC	Imprecise	N/A	Reply-Data Uncorrected

10.4 TLB

The processor TLB is protected from multiple entry matches by detecting such cases on TLB writes. When a TLB write is executed and a match occurs that would result in multiple matching entries in the TLB, the processor takes a machine check exception and sets the TS bit in the Status register. This exception is imprecise.

When matching entries are detected, the processor does not write the TLB with the conflicting entry. Software may try to correct the situation by flushing the TLB, but before the error handler returns, it must clear the TS bit.

10.4.1 Implementation Notes

The TLB actually implements a hidden "valid" bit which is cleared by reset and only set on a TLB write. Matches cannot occur for an entry whose valid bit is cleared. Because the valid bit prevents matches after reset and because the processor prevents matching writes, there is no need to "shutdown" the TLB when an error condition is detected as multiple entries will never be present. Software's clearing the TS bit is simply based on architectural convention since the bit only indicates status. Note that even though the TS bit may be set, TLB lookups will continue as no harm will result, and TLB faults may occur if the handler uses mapped addresses.

10.5 BIU

The BIU forwards errors detected on the system bus to the exception unit. If an instruction request ends in an uncorrectable cache error or bus error, the fetch unit signals the appropriate error for that request. All instruction bus and cache errors are precise. A data request that ends in an error causes the processor to take the cache error or bus error exception at the earliest available instruction. Because the primary data cache is non-blocking, these errors are imprecise. In addition, the imprecise errors are held pending until the detected exception is taken. On all requests that end in an external error, the data returned have no meaning and are never written into the cache.

The BIU contains no timeout mechanism for bus requests. The external agent is responsible for returning a bus error on a processor request that cannot be serviced so the BIU request entry may be deallocated.

The primary data cache duplicate tags also detect parity errors on all coherent bus transactions. Because the duplicate tags are basically shadow copies of the main tags, they have the same error properties and may detect errors in the state bits or address bits. Any way that contains an error causes a cache error to be taken. Each case is outlined in Table 10-9.

TABLE 10-9 Duplicate Tag State/Address Parity Cache Errors

Error-Exception	Type	Processor Behavior
Tag State Parity-Cache Error	Imprecise	Error (Unowned)
Tag Address Parity-Cache Error	Imprecise	Error (Unowned)

Any error in the duplicate tags causes the processor to indicate an error response. From the point of view of the bus protocol, the processor does not own the line.

10.6 General

The CPU implements a 29b timeout counter to detect when the processor is no longer executing instructions. The counter is reset to zero every cycle an instruction graduates and increments whenever zero instructions graduate. If the counter overflows, the processor takes a machine check exception, which releases the processor to begin execution at the exception handler. Processor state may be saved by the handler, but it is likely that the core needs to be initialized by a reset sequence to behave correctly.

Software can detect a machine check due to a timeout by reading the TO bit in the ErrCtl register. When the timeout counter expires, this bit is set, and it can only be cleared by reset.

10.6.1 Implementation Notes:

A 29b counter will cause a machine check after approximately 500ms at 1GHz (2^29 cycles elapse before a carry out of the counter is generated).

10.7 Error Reporting Registers

In general, the first source of error information can be found in the CP0 ErrCtl register (number 26, select 0). For cache errors, this register indicates which cache, instruction or data, detected the error and whether the error is recoverable, i.e., corrected by the hardware. In addition, the register indicates when multiple bus errors have occurred or what action has resulted in a machine check.

ErrCtl (Register 26, Select 0):

TABLE 10-10 **CP0 Err Ctl Register Fields**

Bits	Field	Description	Access	Reset
[31]	R	Recoverable Cache Error	R	0
[30]	DC	Data Cache Error	R	0
[29]	IC	Instruction Cache Error	R	0
[28:24]	0	Reserved	R	0
[23]	MB	Multiple Bus Errors Detected	R	0
[22:16]	0	Reserved	R	0
[15]	TS	TLB Shutdown Machine Check (Copy of Status TS Bit)	R	0
[14]	TO	Timeout Machine Check	R	0
[13]	0	Reserved	R	0

The ERL bit modifies the behavior of the DC and IC bits when cache errors are detected. If ERL=0 and a cache error exception is taken, these bits log the cache in which the error occurred, and only one bit is set. If ERL=1, these bits become "sticky" so cache errors can accumulate while the processor is executing the cache error handler. In this way, data cache errors cannot mask instruction cache errors, or vice versa.

Note that if a Cache Error exception is taken and the R bit is set, then the handler may return immediately as the error has been corrected by hardware. Software may, however, log the error if desired since the CacheErr registers contain valid information about the error. In addition, if the IC bit is set, the R bit is masked so that instruction cache errors are always serviced.

The TS and TO bits may both be set in the ErrCtl register. In this case, the timeout error takes higher precedence than the TLB shutdown.

The CacheErr-I register (number 27, select 0) indicates the cause and location of errors detected in the instruction cache. The TA bit indicates that the processor detected a parity error in the tag address array, while the D bit indicates that the parity error was detected in the fetched instructions. The E bit indicates that the error occurred on an external access. For tag errors, the Idx field is valid and indicates the cache index where the error was detected, but the Way field is unpredictable. For data errors, both the Idx and Way fields are valid and point where the error instructions are located in the cache. Neither the Idx nor the Way field is defined for external errors, so the EPC register should be used for address information.

The following table specifies the CacheErr-I format and indicates when the Idx and Way fields are valid:

TABLE 10-11 CacheErr-I Format

Bits	Field	Description	Access	Reset
[31:30]	0	Reserved	R	0
[29]	TA	Tag Address Parity Error	R	X
[28]	D	Data Array Parity Error	R	X
[27]	0	Reserved	R	0
[26]	E	External Cache Error	R	X
[25:16]	0	Reserved	R	0
[15]	0	Reserved for Cache Index/Way	R	0
[14:13]	Way	Instruction Cache Way	R	X
[12:5]	Idx	Instruction Cache Index	R	X
[4:0]	0	Reserved	R	0

TABLE 10-12 Validity of Idx and Way Fields in ICache Errors

Error	Idx	Way
Tag Address	Valid	Invalid
Data Array	Valid	Valid
External	Invalid	Invalid

CacheErr-I (Register 27, Select 0):

The CacheErr-D register (number 27, select 1) indicates the cause and location of errors detected in the data cache. The TS bit indicates that the processor detected a parity error in the tag state array, while the TA bit signals a parity error in the tag address. If a single-bit ECC error is detected, then the DS bit is set, and the DD bit is set when a double-bit ECC error is detected. Finally, the E bit is set when a data cache access ends in an external error.

Five bits in the CacheErr-D register indicate the type of access that caused the error to be detected: L for loads, S for stores, WB for writebacks, C for coherency requests (such as invalidates or interventions), and DT for duplicate tag accesses.

In addition to the CacheErr-D register, another CP0 register, CacheErr-DPA (register 27, select 3), captures the entire 40b physical address for data cache accesses. It is always written when an error is detected in the data cache.

Table 10-13 specifies the CacheErr-D format and indicates when the Idx and Way fields are valid and when the PA is valid in the CacheErr-DPA register.

TABLE 10-13 CacheErr- D Format

Bits	Field	Description	Access	Reset
[31]	M	Multiple Data Cache Errors	R	0
[30]	TS	Tag State Parity Error	R	X
[29]	TA	Tag Address Parity Error	R	X
[28]	DS	Data Array Single-Bit ECC Err	R	0
[27]	DD	Data Array Double-Bit ECC Err	R	X
[26]	E	External Cache Error	R	0
[25]	L	Error on Load Access	R	0
[24]	S	Error on Store Access	R	X
[23]	FWB	Error on Fill/Writeback	R	X
[22]	C	Error on Coherency Access	R	X
[21]	DT	Error on Duplicate Tag Access	R	X
[20:16]	0	Reserved	R	0
[15]	0	Reserved for Cache Index/Way	R	0
[14:13]	Way	Data Cache Way	R	X
[12:5]	Idx	Data Cache Index	R	X
[4:0]	0	Reserved	R	0

TABLE 10-14 **Validity of Idx, Way and PA Fields in DCache Errors**

Error	Idx	Way	PA
Load/Store:			
Tag Address	Valid	Invalid	Valid
Tag State	Valid	Invalid	Valid
Data Single-Bit ECC	Valid	Valid	Valid
Data Double-Bit ECC	Valid	Valid	Valid
External	Invalid	Invalid	Valid
Writeback/Coherency:			
Tag Address	Invalid	Invalid	Valid
Tag State	Invalid	Invalid	Valid
Data Single-Bit ECC	Invalid	Invalid	Valid
Data Double-Bit ECC	Invalid	Invalid	Valid
Duplicate Tag:			
Tag Address	Invalid	Invalid	Valid
Tag State	Invalid	Invalid	Valid

The CacheErr-D and CacheErr-DPA always capture the first signaled cache error, and the information stored in these registers is "locked" until software performs a write to the CacheErr-D register. If another cache error is detected before the lock is cleared, the M bit is set, indicating that multiple cache errors have been detected. Like the lock bit, the M bit is cleared by a write to the CacheErr-D register.

When a data cache access ends in a bus error, the physical address of that request is stored in a separate register, BusErr-DPA (number 26, select 1). Like the CacheErr-D and CacheErr-DPA registers, this register contains the first detected error, and the lock for the register may be cleared by a write to that register. Multiple bus errors are also indicated, and the MB bit in the ErrCtl register serves this purpose.

Table 10-15 provides a summary of error reporting registers.

TABLE 10-15 **Error Reporting Registers**

Name	Reg	Sel
ErrCtl	26	0
BusErr-DPA	26	1
CacheErr-I	27	0
CacheErr-D	27	1
CacheErr-DPA	27	3

CHAPTER 11 The Performance Monitor Architecture

11.1 Introduction

This document describes the architecture states, features, and performance events of the performance monitor mechanism in SB1. The architecture features and states are pretty much final and mostly compliant with MIPS architecture specification, but with additional counters, event registers, and larger counter width. The architecture states include four pairs of counters and control registers, cache and branch event registers. The performance events are a list of machine dependent events which are interesting to monitor for analyzing performance. Some events may be changed due to implementation in the future.

11.2 Architecture State and Features

The performance monitor mechanism uses a total of 11 registers. The traditional counter control and data registers are mapped to CP0 Register 25 with Select from 0 to 7. The added event control and address registers are mapped under CP0 Register 22 with Select from 0 to 2. The mapping of the register is shown in Table 1. Their functionality and contents are described in the following sessions.

TABLE 11-1 Performance Counter Register Mapping

Register	Select	Counter Index	Register Usage
25	0x00	0	Event control register 0
	0x01		Event control register 0
	0x02	1	Event control register 1
	0x03		Event control register 1
	0x04	2	Event control register 2
	0x05		Event control register 2
	0x06	3	Event control register 3
	0x07		Event control register 3
	0x08-0x0F	NA	Reserved
22	0x00	Event	Event control register
	0x01		Event instruction address register
	0x02		Event memory address register

11.2.1 Event Counter and Control Registers (Register = 25, Select = 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07)

SB1 implements four 41-bit count up counters. They are readable and writable by software and updated implicitly by hardware event specified in the corresponding control register. The MSB (bit 40) is the overflow bit when there is a carry out from bit 39. It can be used to cause counter overflow interrupt and freeze the update of all the counters and event registers. Forty bits should be sufficient enough to last for about 18 minutes of execution time to count a 1-bit event at a frequency of 1 GHz.

Each counter register is paired with a control register. The following figure shows the format of the counter control register. Table 2 describes the control register fields. The control register is both readable and writeable by software.

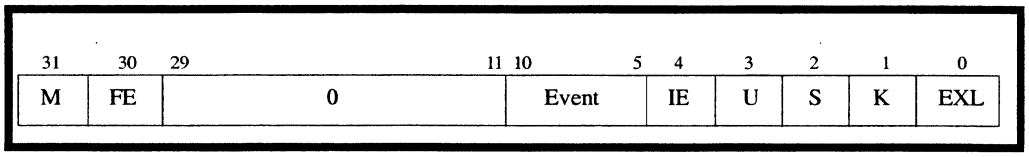

FIGURE 11-1 Performance Counter Control Register

TABLE 11-2 Performance Counter Control Register Field Description

Field Name	Bits	Description	SW Read/Write	Reset State	Compliance
M	31	If this bit is '1, another pair of performance counter control and counter registers are implemented at n+2 and n+3.	R	Preset	Yes
FE	30	Freeze enable: Once a counter overflows, it freezes the other counters and registers, preventing further updates if this bit is set.	R/W	1	New
0	29:11	Must be set to 0. Return 0 on read. No exception is incurred if a 1 is written.	R/W	0	Yes
Event	10:5	ID of the event being monitored: Events are symmetrical to each counter.	R/W	Undefined	Yes
IE	4	Overflow interrupt enable: Once a counter overflows, an overflow interrupt is triggered if this bit is set.	R/W	0	Yes
U	3	Enable event counting in User mode.	R/W	Undefined	Yes
S	2	Enable event counting in Supervisor mode.	R/W	Undefined	Yes

TABLE 11-2 **Performance Counter Control Register Field Description**

Field Name	Bits	Description	SW Read/Write	Reset State	Compliance
K	1	Enable event counting in Kernel mode.	R/W	Undefined	Yes
EXL	0	Enable event counting when EXL bit is set.	R/W	Undefined	Yes

A counter is 41-bit wide. It is incremented by the value of the input event specified in the control register each core cycle. The register format is shown in the following figure. When a counter generates a carry out of the bit 39, its overflow bit (bit 40) is set. In addition, a counter overflow interrupt is incurred if the counter's IE bit is set. The overflow freezes the update of all performance counter registers and event registers when the counter's FE bit is set. This helps retain the counter values precisely to the point of counter overflow. If the FE bit is not set, the counter continues incrementing.

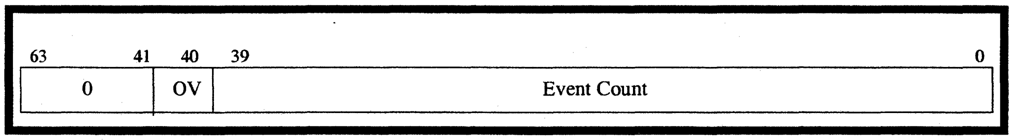

FIGURE 11-2 **Performance Counter Register**

TABLE 11-3 **Performance Counter Register Field Description**

Field Name	Bits	Description	SW Read/Write	Reset State	Compliance
0	63:41	Write is ignored. Read is 0.	R/W	0	New
OV	40	Overflow bit from bit 39.	R/W	1	New
Event Count	39:0	Incremented by the input values of the specified event each CPU cycle.	R/W	Undefined	New

11.2.2 Counter Overflow Interrupt

Once a counter 's overflow bit is set, if the IE bit is set, the hardware interrupt 5 should be raised and the interrupt bit IP(7) should be set in the Cause register.

11.2.3 Event Control and Address Registers (Select = 0x10, 0x11, 0x12)

Instruction cache, data cache, and branch prediction performance are the most important factors in deciding overall machine performance. The interesting events for performance analysis are the instruction cache misses, data cache misses, and branch mispredictions. With detailed information about these events such as the address of the event and the status of the requested data, one can optimize the code to get around the potential performance problems.

SB1 has a set of event address registers to assist the performance analysis: one for specifying the detailed conditions of the event to be captured, one for storing the event's data address, and the other for storing the instruction address of the operations that cause the event. SB1 captures the virtual address of the instruction. It captures the physical address of the data address, due to the unavailability of virtual addresses at the point of capture.

The formats of the registers are shown in the following figures. Table 4 lists the description of the control register fields. The fields in the control register are used to qualify four major sources of events:

- Branch execution
- Instruction cache misses
- Data cache misses for loads
- Data cache misses for stores.

The control register is 32-bit wide. Fields EXL, K, S, and U are common to qualify all four sources of events. They specify the execution modes in which the events can be captured. Fields Cc, Cw, Pt, Pn, Ot, On, Bc, Br, Bi, Bu, Bc, and Bwi (bits 4 to 15) are used only to qualify the branch execution events. These fields are divided into five group of filters as follows:

- Prediction result filter (Cc and Cw): they are masks to select the branches with correct or incorrect prediction results which include taken/not-taken predictions and indirect target predictions. When both bits are set, all the branches regardless of their prediction outcomes are included. When only one bit is set, only the branches with the selected outcome are included. When both bits are cleared, no branches are included.
- Prediction filter (Pn and Pt): they are masks to select the branches with taken or not-taken predictions. When both bits are set, all the branches regardless of their prediction outcomes of taken/not-taken are included. When only one bit is set, only the branches with the selected prediction are included. When both bits are cleared, no branches are included.
- Outcome filter (Ot and On): they are masks to select the branches with taken or not-taken outcomes. When both bits are set, all the branches regardless of their execution outcomes of taken/not-taken are included. When only one bit is set, only the branches with the selected outcome are included. When both bits are cleared, no branches are included.
- Branch type filter (Bc, Br, Bi, Bu, and Bc): they are masks to select various types of branch instructions as specified in the table.

- Instruction watch filter (Bwi): it indicates whether or not the branch event needs to be qualified with instruction watch register match.

The final qualification of a branch event is the ANDing result of the execution modes, and the qualification from each of the above five categories. With all these fields, we have a powerful filtering logic to select the interesting sets of branch events for capturing. The logic expression of the filtering is shown as follows:

Branch Event Qualification = (Execution mode filter) & (Prediction result filter) & (Prediction filter) &

Execution outcome filter) & (Branch type filter) & (Instruction watch match filter)

Note that unconditional, indirect, call, and return branches are always predicted taken, because their outcomes are always taken. In addition, since the predicted targets of unconditional branches are calculated, the prediction is always correct.

The rest of the fields in the control register (W, R, I, Cwd, and Cwi) are used for qualifying cache miss events which include instruction cache misses, data cache misses for loads, and data cache misses for stores. They are used in the following ways:

- Access type filter (W, R, and I): they are masks to select various types of cache accesses. When 'W' is set, the store accesses to the data cache are included. When 'R' is set, the load accesses to the data cache are included. When 'I' is set, the instruction cache miss accesses are included. With various bits being set, various types of cache accesses are included.
- Data watch register match filter (Cwd): it indicates the cache access event needs to be qualified with the match result of data watch register. This filter is available only for data cache accesses for loads (when R is set). It is a don't care for all the other types of events.
- Instruction watch register match filter (Cwi): it indicates the cache access event needs to be qualified with the match result of instruction watch register. This filter is available only for the instruction cache miss requests and data cache misses for loads (when I or R is set). It is a don't care for data cache misses from stores.

The final qualification of a cache access event is the ANDing result of the execution modes and the qualification from each of the above three categories. With all these fields, there is a powerful filtering logic to select the intersecting sets of cache miss events to capture. The logic expression of the filtering of each access type is shown below:

Cache Event Qualification = (Execution mode filter) & (Access type filter) &

(Data watch match filter) & (Instruction watch match filter)

Note that each cache access type should have independent qualification logic since they each use different filters.

31...21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Cwi	Cwd	I	R	W	Bwi	Bc	Bu	Bi	Br	Bl	On	Ot	Pn	Pt	Cw	Cc	U	S	K	EXL

FIGURE 11-3 Cache Event Control Register

TABLE 11-4 Cache Event Control Register Field Description

Field Name	Bits	Description	SW Read/Write	Reset State	Compliance
0	31:21	Write is ignored. Return 0 on read. No exception is incurred if a 1 is written	R	0	New
Cwi	20	Cache event capturing should be qualified by the match result of instruction watch register	R/W	Undefined	New
Cwd	19	Cache event capturing should be qualified by the match result of data watch register	R/W	Undefined	New
I	18	Select instruction fetch to be captured	R/W	Undefined	New
R	17	Select data reads to be captured (for loads)	R/W	Undefined	New
W	16	Select data writes to be captured (for stores)	R/W	Undefined	New
Bwi	15	Branch event capturing should be qualified by the match result of instruction watch register	R/W	Undefined	New
Bc	14	Conditional branch mask; this bit selects all the conditional branches	R/W	Undefined	New
Bu	13	Unconditional branch mask; this bit selects all the unconditional jumps	R/W	Undefined	New
Bi	12	Indirect branch mask: this bit selects all the indirect branches.	R/W	Undefined	New
Br	11	Procedure return mask: this bit selects all the procedure returns.	R/W	Undefined	New
Bl	10	Procedure call mask: this bit selects all the procedure calls.	R/W	Undefined	New
On	9	Not-Taken outcome mask: this bit selects the branches with not-taken outcome.	R/W	Undefined	New
Ot	8	Taken outcome mask: this bit selects the branches with taken outcome.	R/W	Undefined	New
Pn	7	Not-Taken prediction mask: this bit selects the branches with not-taken predictions.	R/W	Undefined	New
Pt	6	Taken prediction mask: this bit selects the branches with taken predictions.	R/W	Undefined	New
Cw	5	Wrong prediction mask: this bit selects the branches with wrong predictions.	R/W	Undefined	New

TABLE 11-4 Cache Event Control Register Field Description

Field Name	Bits	Description	SW Read/Write	Reset State	Compliance
Cc	4	Correct prediction mask: this bit selects the branches with correct predictions.	R/W	Undefined	New
U	3	Enable event capturing in User mode.	R/W	Undefined	New
S	2	Enable event capturing in Supervisor mode.	R/W	Undefined	New
K	1	Enable event capturing in Kernel mode.	R/W	Undefined	New
EXL	0	Enable event capturing when EXL bit is set.	R/W	Undefined	New

Each of the two event address registers is 64-bit wide. The instruction address register stores the virtual instruction address of either the load that causes a qualified data cache event or the branch that causes a qualified branch event. For instruction cache miss events and data cache miss events from stores, the instruction address register does not latch the instruction addresses, because the addresses are not available when the event is captured. Since each instruction is 32-bit wide, the lowest 2 address bits are always 0. Therefore, the lowest two bits in the instruction address register are then used to indicate the type of instruction address being captured, as shown in Table 5.

The format of the instruction address register is shown in the following figure. The description of the fields can be seen in Table 5.

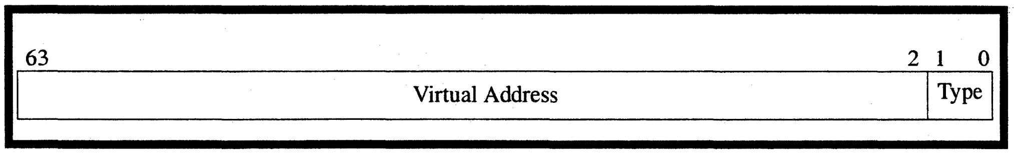

FIGURE 11-4 Event Instruction Address Register

TABLE 11-5 Event Instruction Address Register Field Description

Field Name	Bits	Description	SW Read/Write	Reset State	Compliance
Virtual Address	63:2	The virtual instruction address of the instruction that is qualified for the branch events or data cache events.	R/W	Undefined	New
Type	1:10	Type of address captured: 00 is for not-taken branch, 01 is for taken branch, 10 is for data load, and 11 is reserved.	R	Undefined	New

The data address register, on the other hand, stores either the missing data line addresses or missing instruction line addresses, because all the data cache read/write accesses and instruction misses must go through the same pipeline before they are sent to the bus. Because of the timing of the event capturing, we store the physical address of the data/instruction line being referenced, instead of its virtual address. Since the physical line address is stored, the lowest five address bits are always 0. The physical address takes up bits 5 to 39 in the register. The higher order bits are used to indicate the status of the captured event. The format of the register is shown in Table 6.

Bits I, R, and W indicate the type of the event as instruction cache miss, data cache miss from load, and data cache miss from store respectively. When a miss event is first captured, the pending bit is set until the data returns. Once the data actually returns, the pending bit is cleared and the source of the data return is recorded in the status bits as follows:

- Dirty (D) bit: indicates the returned data is dirty
- Other (O) bit: indicates the data is returned by the bus agents other than processors, secondary cache, and main memory
- Main memory (M) bit: indicate the data is returned by main memory
- Secondary Cache (C) bit: indicate the data is returned by the secondary cache
- Processor (P) bit: indicate the data is returned by other processors

63	62	61	60	59	58	57	56	55	54 40	39 5	4 0
Pd	I	R	W	P	C	M	O	D	0	Physical Address	0

FIGURE 11-5 Event Data Address Register

TABLE 11-6 Event Data Address Register Field Description

Field Name	Bits	Description	SW Read/Write	Reset State	Compliance
Pd	63	Cache miss event captured is still pending for the return.	R	0	New
I	62	Captured event is an instruction cache miss request.	R	Undefined	New
R	61	Captured event is a data cache access from read request.	R	Undefined	New
W	60	Captured event is a data cache access from write request.	R	Undefined	New
P	59	Captured request is serviced by the processor.	R	Undefined	New

TABLE 11-6 Event Data Address Register Field Description

Field Name	Bits	Description	SW Read/Write	Reset State	Compliance
C	58	Captured request is serviced by the secondary cache.	R	Undefined	New
M	57	Captured request is serviced by main memory.	R	Undefined	New
O	56	Captured request is serviced by agents other than the processors, secondary cache, and main memory	R	Undefined	New
D	55	Data returned as dirty.	R	Undefined	New
0	54:40	Reserved. Write is ignored. Read returns 0.	R	0	New
Physical Address	39.5	The memory physical address of data cache miss event or instruction fetch miss event.	R	Undefined	New
0	4:0	Reserved. Write is ignored. Read returns 0.	R	0	New

11.3 Performance Events

The following table lists the performance events being designed in SB1. The 'event description' field shows the meaning of the event being generated. The 'count description' field shows what the intended count is. The 'type' field shows how an event is counted in the machine pipeline. It could be counted speculatively (Spec), non-speculatively (past the branch validation) (Non-S), or post graduation (Grd). The 'Max' field shows the maximum count of an event in one cycle. The 'Src' field shows the generator of the event signal. The event ID has not been assigned. They will be assigned based on the physical implementation later. The total number of events is currently 50.

TABLE 11-7 Instruction Count

Event #	Event Description	Count Description	Type	Max	Src
Instruction Frequency					
	Clock is high	# of cycles			PC
	# of instructions retired	total number of instructions retired	Grd	1	PC
	a BR instruction executed	# of BR instructions executed	Non-S	4	PC
	# of LD instructions executed	total # of LD instructions executed (after speculative point)	Non-S	1	PC
	# of ST instructions executed	total # of ST instructions executed (after speculative point)	Non-S	2	PC
	a CP0 instruction executed	# of CP0 instruction executed (after speculative point)	Non-S	2	PC
	# of FLOPS executed	# of FLOPS executed	Non-S	1	PC
	# of MOPS executed	# of MOPS executed	Non-S	8	PC
	a store conditional executed	# of store conditionals executed (after speculative point)	Grd	2	PC
	a successful store conditional executed	# of successful store conditionals executed (after speculative point)	Grd	1	PC
Cache Access Events					
	a cache event captured	# of cache events captured	Spec	1	M
	a fetch results in I-Cache miss	# of fetch results in I-Cache miss	Spec	1	PC
	a valid I-Cache fill	# of valid I-Cache fill	Spec	1	PC
	a D-cache read results in a miss	# of D-cache read misses	Spec	2	DC
	D-cache is filled	# D-cache fills	Spec	1	DC
	a read hits in DCFIFO	# of read hits in DCFIFO	Spec	2	DC
	a read hits in read queue	# of merged read accesses	Spec	2	M
	a load/store hits prefetch in read queue	# of load/store hits prefetch in read queue	Spec	2	M
	a prefetch hits in cache or read queue	# of canceled prefetch requests	Spec	2	M

TABLE 11-7 Instruction Count

Event #	Event Description	Count Description	Type	Max	Src
	# of valid entries in read queue	average life time of request in read queue	Spec	8	M
	# of valid uncached entries in read queue	average life time of uncaches request in read queue	Spec	8	M
	a request hits in write buffer	# of request hit in write buffer	Non-S	1	M
	a writeback occurs due to replacement	# of write-backs due to replacement	Non-S	1	M
Bus Interface					
	a bus request is sent to ZB bus	# of bus requests	Non-S	1	B
	a bus read request is sent to ZB bus	# of bus read requests	Non-S	1	B
	a bus write request is sent to ZB bus	# of bus write requests	Non-S	1	B
	BIU stalls due to address bus busy	# of stall cycles for waiting for address bus	Non-S	1	B
	BIU stalls due to data bus busy	# of stall cycles for waiting for data bus	Non-S	1	B
Multiprocessor					
	a snoop request comes	# of snoops	Non-S	1	B
	a snoop hits in write buffer	# of snoop hits in write buffer	Non-S	1	M
	a shared snoop hits on a shared line	# of read shared snoop hits on a shared line (no action)	Non-S	1	B
	a shared snoop hits on an exclusive line	# of read shared snoop hits on an exclusive line (intervention shared)	Non-S	1	B
	an exclusive snoop hits on shared line	# of read exclusive snoop hits on shared line (invalidate)	Non-S	1	B
	an exclusive snoop hits on exclusive line	# of read exclusive snoop hits on exclusive line (intervention exclusive)	Non-S	1	B
	an invalidate snoop hits on shared line	# of invalidate snoop hits on shared line (invalidate or write-invalidate)	Non-S	1	B
	an invalidate snoop hits on exclusive line	# of invalidate snoop hits on exclusive line (invalidate or write-invalidate)	Non-S	1	B
	snoop address queue is full	# of cycles when snoop address queue is full	Non-S	1	B

TABLE 11-8 Microarchitectural Events

Event #	Event Description	Count Description	Type	Max	Src
	FE pipeline efficiency				
	taken branch bubble visible (a bubble reaches issue when there is no valid instruction to issue)	taken branch bubble visible (a bubble reaches issue when there is no valid instruction to issue)	Spec	1	PC

TABLE 11-8 Microarchitectural Events

Event #	Event Description	Count Description	Type	Max	Src
	instruction cache miss bubble (a bubble reaches issue when there is no valid instruction to issue)	instruction cache miss bubble (a bubble reaches issue when there is no valid instruction to issue)	Spec	1	PC
	Issue efficiency				
	# of instructions issued	# of instructions issued	Spec	4	I
	No valid instructions available for issuing	# of cycles when no valid instructions are available for issuing	Spec	4	1
	Valid instructions are available for issuing but stopped by resource constraints	# of cycles when valid instructions are available for issuing but stopped by resource constraints	Spec	4	1
	Valid instructions are available for issuing but stopped by dependency constraints	# of cycles when valid instructions are available for issuing but stopped by dependency constraints	Spec	4	1
	Issue stopped by width limit	# of cycles when maximum issue is achieved	Spec	4	1
	Replay and Misprediction				
	a replay is signaled (data dependency, RQ full, DCFIFO, fill)	# of replays signaled (data dependency, RQ full, DCFIFO, fill)	Non-S	1	I
	a replay caused by data dependency is signaled	# of replays caused by data dependency is signaled	Non-S	1	I
	a replay caused by RQ full is signaled	# of replays caused by RQ full	Non-S	1	I
	a replay caused by DCFIFO is signaled	# of replays caused by DCFIFO full	Non-S	1	I
	Branch prediction/execution details				
	a branch event is captured	# of branch events captured	Non-S	1	PC
	Bus interface				
	BIU is busy	# of MBOX requests to BIU when BIU is busy	Non-S	1	M

11.4 Pending Issues

- The assignment of the events to each counter with Mux minimization in mind
- The assignment of only a subset of events to the first counter for denominators
- Events

CHAPTER 12 Multiprocessing Support

12.1 Introduction

This chapter provides a high level overview of the multiprocessing support features provided by SB-1 processor core. The material presented here is intended to be used in conjunction with the more detailed system configuration specifics provided by the SB-1250 User Manual[1].

12.2 Support for Atomic Operations

The SB-1 supports the following standard MIPS atomic operation pairs:

- Load Linked - Store Conditional (LL-SC)
- Load Linked Double - Store Conditional Double (LLD-SCD)

These instructions provide a fast and simple alternative to the Dekker or Peterson algorithms for mutual exclusion. When used properly, these instructions provide support for an atomic read-modify-write sequence, upon which standard mutual exclusion mechanisms can be easily built.

The common usage for a busy-wait memory lock is as follows:

1. SB-1250 is the first Multiprocessing System On a Chip (SOC) product that utilizes the SB-1 core.

Register $1 contains the address of a memory lock/semaphore, where 1 represents a locked state and 0 a clear state.

```
.align 32
.set noreorder
TryAgain:          LL         $2, 0($1)           # get the lock
                   BNE        $2, $0, TryAgain    # if lock==1, spin
                   ADDIU      $2, $0, 1           # lock==0, so $2=1
                   SC         $2, 0($1)           # lock=1
                   BEQ        $2, $0, TryAgain    # if r-m-w fails ($2==0), spin
                   NOP
==== critical section =====
                   ADD        $2, $0, $0          # $2=0
                   SW         $2, 0($1)           # lock=0
```

If a 1 is written to $2 by the Store Conditional (SC), it indicates that the SC successfully updated the architectural view of memory location ($1). Otherwise, a 0 is written to $2. The memory lock must exist in cached coherent memory space, otherwise the results are UNPREDICTABLE.

Although not necessary for correct behavior, aligning the LL-SC sequence into the same 32 bytes can reduce spin time by ensuring that an Icache miss never occurs between the Load Linked and the Store Conditional.

There are several events which will cause the failure of Store Conditional instruction if they occur between a Load Linked and Store Conditional instruction pair. These include the following:

- Completion of a coherent memory access to the same 32-byte aligned block of memory by another processor,
- The occurrence of an exception on the processor executing the LL/SC instruction pair,
- A line fill which forces the locked line out of the cache.

In addition, the results of the Store Conditional are UNPREDICTABLE if:

- The Store Conditional instruction is not preceded by a Load Linked instruction.
- The Store Conditional instruction is preceded by a Load Linked instruction to a different physical or virtual address.

Any of the above conditions may cause a Store Conditional to indicate success without actually guaranteeing atomic access to the memory block in question.

The Load Linked-Store Conditional pairings do not explicitly guarantee fairness, only mutual exclusion.

12.3 Processor Synchronization

The following sections elaborate on processor synchronization schemes available for SB-1 multiprocessing.

12.3.1 Test and Set

For an example of how this synchronization method operates in SB-1, refer to Section 12.2.

12.3.2 Counter Based Synchronization

The common usage for a counting semaphore is shown below.

The memory lock contains the number of processes allowed in the critical section, and its address is specified by register 1 ($1) below:

```
.set noat
.set noreorder
.align 32
TryAgain1:  LL          $2, 0($1)           # Get memory lock
            BEQ         $2, $0, TryAgain1   # Check for non-zero result
            DADDIU      $2, $2, -1          # Decrement Semaphore (delay slot)
            SC          $2, 0($1)           # Attempt to store
            BEQ         $2, $0, TryAgain1   # If failed, loop back
            NOP
=== Critical Section ===
TryAgain2:  LL          $2, 0($1)           # Get lock again
            DADDIU      $2, $2, 1           # Increment Semaphore
            SC          $2, 0($1)           # Attempt store
            BEQ         $2, $0, TryAgain2   # If failed, loop back
            NOP
```

12.4 Coherency

The following sections provide an overview of the supported memory model and cache organization in systems that use the SB-1 processor.

12.4.1 Memory Model

The SB-1 supports a weakly ordered memory model. This is an important consideration when using non-atomic multi-programming techniques such as producer-consumer structures and shared memory states.

The following rules apply to external visibility of memory accesses in either a cached coherent or cached noncoherent region of memory:

- The order between a load and a store on the same processor is not guaranteed, and
- The order between multiple loads is not guaranteed.

12.5 Cache Organization and Coherency in SB-1

To ensure code efficiency and correctness, the following need to be considered with regard to the cache organization in SB-1.

12.5.1 Instruction Stream Modifications

The SB-1 has split instruction and data caches. Any program that requires modification to its own instruction stream must obey the following sequence of events and must guarantee that all these events occur in the order shown on a single processor before attempting to execute the new code:

1. Store the new instructions
2. Flush the L1 data cache
3. Flush the L1 instruction cache
4. Execute a SYNC instruction

In addition, all other processors in the system must be forced to flush their instruction caches and sync before attempting to execute the new code. Failure to do so may result in the execution of older cached code.

12.5.2 Caching Attributes

The SB-1 implements 4 different caching attributes at page granularity (refer to Chapter 6 for more detail):

- Cached coherent,
- Cached noncoherent,
- Uncached, and
- Uncached accelerated

For cached coherent pages, a snoop-based protocol can be implemented by the encompassing system to maintain coherency across the agents. This protocol is highly efficient and, as such, manual optimization of the cached state of highly-volatile exclusive regions is not recommended (refer to SB-1250 Users Manual for specific details for this system level implementation.)

Cached noncoherent regions allow memory regions outside the range of the coherence protocol to be cached to improve performance. For this class of regions, such as a bus device with memory-mapped configuration data, the operating system is responsible for ensuring that memory writes to the region are coherent across processors. As a minor optimization, memory pages containing only instructions may also be placed in cached noncoherent regions.

Multiprocessor memory contention should generally be avoided in all other modes.

12.6 Processor Bringup

After system reset, processor 1 in SB-1250 is held in reset mode to allow critical system initializations occur in a uniprocessor environment under processor 0. After system initialization is complete, processor 0 may "release" processor 1 by writing a 0 to bit 1 of the system_cfg register.

Processor 1, when "released," begins executing at the normal reset vector. Typically the reset vector code includes a branch based on a read of the PRId register.

Refer to SB-1250 User Manual for further documentation on supported MP, L2, and memory coherency protocols.

CHAPTER 13 SB-1 Implementation Specific Details

13.1 Introduction

This chapter clarifies SB-1 implementation specific details. In particular, implementation-dependent comments in the MIPS64 Manual are referenced and clarified (Each bullet bellow specifies a page number in MIPS64 Manual followed by SB-1 implementation specific comments).

13.2 Clarifications on Implementation-Dependent Non-Privileged Instructions

- P 4: SB-1 implements Reverse Endianness.
- P 13: SB-1 does not implement CP2.
- P 15: SB-1 implements DERET and SDBBP as the only EJTAG instructions.
- P 41: Refer to Prefetch Description in Chapter 6 for supported prefetch hint bits in SB-1.
- P 44: Refer to Prefetch Description in Chapter 6 for supported prefetch hint bits in SB-1.
- P 46: For those Cache Operations that require an index, no translation of the effective address occurs in SB-1.
- P 47: An Address Error Exception (with cause code equal AdEL) may occur if the effective address references a portion of the kernel address space which would normally result in such an exception.

- P 47: A data watch is not triggered by a cache instruction whose address matches the Watch register address match conditions.
- P 48: DataLo and DataHi registers are not implemented in SB-1.
- P 49: Code 011 is not implemented in SB-1.
- P 50: "Fetch and Lock" Cache Op is not implemented in SB-1.
- P 53: SB-1 includes the standard TLB MMU.
- P 54-59: For all TLB instructions (TLBR, TLBWI, TLBWR), no masking is involved in the VPN2 and PFN fields of EntryHi, EntryLo0 and EntryLo1 registers. All bits are preserved after a TLB entry is written and then read.
- P 60: SB-1 does not implement the WAIT instruction; it is treated as a noop.
- P 61: The format of FIR register is described in Chapter 4 of this document.
- P 62: SB-1 will flush all denormals to zero if flush to zero is enabled. It will also flush all underflow results to zero. If flush to zero is disabled, the SB-1 will cause an unimplemented operation exception for denormal inputs and underflowing results for arithmetic operations.

13.3 Clarifications on Implementation-Dependent Privileged Instructions

- P 70: SB-1 does not implement CP0 Reg22.
- P 73: In SB-1, SEGBITS = 44 and PABITS = 40.
- P 76: SB-1 implements 64-bit addressing.
- P 76: SB-1 implements Supervisor Mode.
- P 79: Refer to the next two bullets for implementation-dependent behavior when $\text{Status}_{\text{ERL}}$ =1.
- P 83: For kuseg segment when $\text{Status}_{\text{ERL}}$ =1, the lower 2^{31} byte segment of kuseg is treated as an ummapped uncached segment. For 64-bit addressing mode, when the UX bit is set in CP0 register, for range of addresses between $2^{31}$and 2^{44}, the following address translation occurs: bits 39 to 32 of the translated PA are all zeros, bits 31 to bit 0 of the translated PA are the same as the corresponding bits of the virtual address; the cache attribute is that of uncached type.
- P 85: Refer to Chapter 7 for the TLB format in SB-1.

- P 94: The *ErrorEPC* register is loaded with PC-4 if the state of the processor indicates that it was executing an instruction in the delay slot of a branch. Otherwise, the *ErrorEPC* register is loaded with PC. Note that this value may or may not be predictable if the Reset Exception was taken as the result of power being applied to the processor because PC may not have a valid value in that case. In SB-1, the value loaded into *ErrorEPC* register is not predictable on a Reset.
- P 94: Soft Reset exception is not implemented in SB-1.
- P 95: NMI exception is implemented in SB-1.
- P 96: Machine check exception is implemented in SB-1 for TLB/Time out.
- P 96: In SB-1, detection of multiple matching entries in the TLB occurs on the TLB write that creates multiple matching entries.
- P 99: In SB-1, a cache error exception resulting from an access to the data cache is generally reported *imprecisely* with respect to the instruction that caused the cache error.
- P 100: In SB-1, a data bus error exception is reported *imprecisely* with respect to the instruction that caused the bus error.
- P 102: From the MIPS64 Manual: "Some implementations of previous ISAs reported this case as a Floating Point Exception, setting the Unimplemented Operation bit in the Cause field of the *FCSR* register." SB-1 *does not* do this.
- P 103: If the EXL or ERL bits are one in the *Status* register and a single instruction generates both a watch exception (which is deferred by the state of the EXL and ERL bits) and a lower-priority exception, the lower priority exception is taken. In SB-1, the WP bit is not set in this case.
- P 103: In SB-1, a data watch exception is not triggered by a prefetch or cache instruction whose address matches the Watch register address match conditions.
- P 105: In SB-1, the width of the index field matches the size of the TLB.
- P 106: The random CP0 register is incremented by one for each cycle that has more than zero intruction(s) graduated, except for the cycles that have a TLBWI or TLBWR graduated. For every 3rd of such cycles, the random CP0 register is not incremented.
- P 108: Refer to Chapter 6 for a full listing of SB-1 implemented cache coherency attributes.
- P 112: SB-1 Implements 4K, 16K, 64K, 256K, 1M, 4M, 16M, and 64M[1] page sizes.
- P 114: The Count register acts as a timer, incrementing at a constant rate, whether or not an instruction is executed, retired, or any forward progress is made through the pipeline. For SB-1, the rate at which the counter increments is *once per cycle*.

1. 64M support will be in Pass2 of SB-1.

- P 116: When the value of the *Count* register equals the value of the *Compare* register, an interrupt request is ORed with hardware interrupt 5 to set interrupt bit IP(7) in the *Cause* register. This causes an interrupt as soon as the interrupt is enabled.
- P 117: In SB-1, the RP bit is not implemented.
- P 119: The TS bit indicates that the TLB has detected a match on multiple entries. In SB-1, this detection occurs on a write to the TLB.
- P 120: Supervisor Mode is implemented.
- P 128: For a description of PRId Register format in SB-1, refer to Chapter 8 in this document.
- P 132: For a description of LLAddr Register format in SB-1, refer to Chapter 8 in this document.
- P 133: SB-1 provides two pairs of WatchLo and WatchHi registers, referencing them via the select field of the MTC0/MFC0 and DMTC0/DMFC0 instructions. Refer to Chapter 9 in this document.
- P 133: In SB-1, a data watch is not triggered by a prefetch or a cache instruction whose address matches the Watch register address match conditions.
- P 137: For a list of performance counters implemented in SB-1, refer to Chapter 11 in this document.
- P 140: For the exact format and operation of the ErrCtl register, refer to Chapter 10 in this document.
- P 140: For the exact format and operation of the CacheErr register, refer to Chapter 10 in this document.
- P 142: For the exact format of the TagLo and TagHi registers, refer to Chapter 6 in this document.
- P 145: For a list of CP0 Hazards in SB-1 refer to Chapter 8.