EE Core User's Manual

Publication date: April 2001

Sony Computer Entertainment Inc.
1-1, Akasaka 7-chome, Minato-ku
Tokyo 107-0052 Japan
Inquiries: ps2-system-manual@ml.scei.co.jp

About This Manual

The "EE Core User's Manual" describes the EE Core (the CPU core unit), which controls the entire Emotion Engine, and its general operations. It provides overviews of the configuration and instruction set and the functions of the main blocks. It gives only introductions to critical parts that are controlled chiefly by system software. For details of the instruction set, refer to the "EE Core Instruction Set Manual".

- Chapter 1 "Architecture Overview" describes the EE Core's features and configuration, pipeline operation, and the main blocks and functions.
- Chapter 2 "Instruction Set Overview" provides an overview of the instruction set, and describes data alignment and characteristic operations such as branch delay.
- Chapter 3 "Caches" describes the EE Core's instruction cache, data cache, and scratchpad RAM.
- Chapter 4 "Floating-Point Unit (FPU)" describes the floating-point operation unit connected as a coprocessor.

(This page is left blank intentionally)

Glossary

Term	Definition
EE	Emotion Engine. CPU of the PlayStation 2.
EE Core	Generalized computation and control unit of EE. Core of the CPU.
COP0	EE Core system control coprocessor.
COP1	EE Core floating-point operation coprocessor. Also referred to as FPU.
COP2	Vector operation unit coupled as a coprocessor of EE Core. VPU0.
GS	Graphics Synthesizer. Graphics processor connected to EE.
GIF	EE Interface unit to GS.
IOP	Processor connected to EE for controlling input/output devices.
SBUS	Bus connecting EE to IOP.
VPU (VPU0/VPU1)	Vector operation unit. EE contains 2 VPUs: VPU0 and VPU1.
VU (VU0/VU1)	VPU core operation unit.
VIF (VIF0/VIF1)	VPU data decompression unit.
VIFcode	Instruction code for VIF.
SPR	Quick-access data memory built into EE Core (Scratchpad memory).
IPU	EE Image processor unit.
word	Unit of data length: 32 bits
qword	Unit of data length: 128 bits
Slice	Physical unit of DMA transfer: 8 qwords or less
Packet	Data to be handled as a logical unit for transfer processing.
Transfer list	A group of packets transferred in serial DMA transfer processing.
Tag	Additional data indicating data size and other attributes of packets.
DMAtag	Tag positioned first in DMA packet to indicate address/size of data and address of the following packet.
GS primitive	Data to indicate image elements such as point and triangle.
Context	A set of drawing information (e.g. texture, distant fog color, and dither matrix) applied to two or more primitives uniformly. Also referred to as the drawing environment.
GIFtag	Additional data to indicate attributes of GS primitives.
Display list	A group of GS primitives to indicate batches of images.

(This page is left blank intentionally)

Contents

1. Architecture Overview

This chapter gives an overview of the EE Core architecture, focusing on the following items:

- Block diagram and functional block
- Superscalar pipeline operation
- Instruction set
- Registers
- Memory Management
- Cache Memory and Scratchpad RAM
- Bus interface
- Floating-Point Unit
- Performance Monitors
- Debug Support

1.1. Features of the EE Core

The EE Core is a superscalar implementation of the 64-bit MIPS IV Instruction Set Architecture. It implements a large extension to the instruction set specially tailored for multimedia applications.
It contains a CPU, a floating-point execution unit (Coprocessor 1), instruction and data caches, a scratchpad RAM and a tightly-coupled customer-specific coprocessor (Coprocessor 2).
It has two pipelines, and two instructions can be decoded each cycle. These instructions are executed and completed in order. However, since Data Cache misses are non-blocking and a single outstanding cache miss does not stall the pipeline, load misses or uncached loads may be retired out-of-order. Multiply, Multiply-Accumulate, Divide, Prefetch and Coprocessor instructions are also retired out-of-order.

These features are summarized as follows;

- 2-way superscalar pipeline
- 128-bit (two 64-bit) data path and 128-bit system bus
- Instruction set
  - 64-bit MIPS III instruction set
  - Selected MIPS IV instructions (Prefetch and Move conditional instructions)
  - Non-blocking load instructions
  - Three-operand Multiply and Multiply-Accumulate instructions
  - 128-bit multimedia instructions which configure the 128-bit data path as two 64-bit, four 32-bit, eight 16-bit or sixteen 8-bit paths
- On-chip caches and scratchpad RAM
  - Instruction cache: 16 KB, 2-way set associative
  - Data cache: 8 KB, 2-way set associative (with write-back protocol)
  - Data scratchpad RAM: 16 KB
  - Data cache line locking
  - Prefetch functions
- Fast integer Multiply and Multiply-Accumulate operations
- Memory management unit
  - 48 double entry full set associative address translation look-aside buffer (TLB)

1.2. Block Diagram and Functional Block Description

A block diagram of the EE Core is shown below.

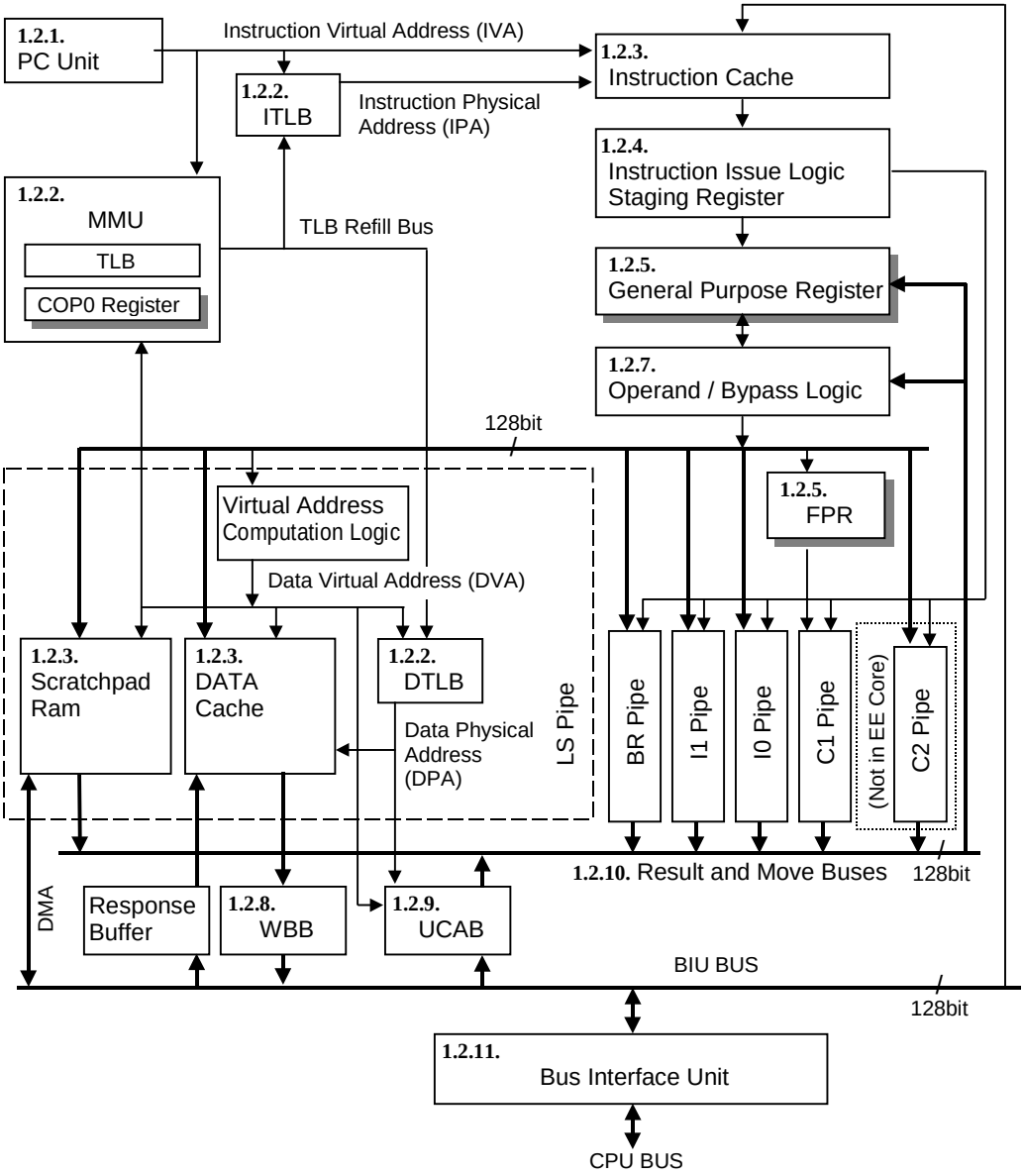


Figure 1-1 EE Core Block Diagram

1.2.1. PC Unit

The 32-bit Program Counter (PC) holds the address of the instruction which is being executed. It contains a 64-entry Branch Target Address Cache (BTAC), which is used during branch predictions.

1.2.2. MMU

The Memory Management Unit (MMU) supports the address translation functions of the CPU. It sends data to the DTLB (Data address Translation Lookaside Buffer) and ITLB (Instruction address Translation Lookaside

Buffer) via the TLB Refill Bus. Usage of these buffers is described in Section "1.3. Superscalar Pipeline Operation".

1.2.3. Caches and Scratchpad RAM

The instruction cache, the data cache and Scratchpad RAM are described in Chapter 4. For each branch instruction present in the Instruction Cache, two bits of branch history are stored in the Branch History Table (BHT).

1.2.4. Issue Logic and Staging Registers

The issue logic decides how to route instructions to appropriate pipes. Details are described later in this chapter.

1.2.5. GPR (General Purpose Registers) and FPR (Floating-Point Registers)

The General Purpose Registers and the Floating-Point Registers are described in Section "1.4. Registers".

1.2.6. Physical Pipes

The EE Core has 6 physical pipes, described below.

I0 and I1 Pipes

The I0 and I1 pipes contain logic to support integer arithmetic. Both are composed of a complete 64-bit ALU, Shifter and Multiply-Accumulate unit. The I0 pipeline contains the SA register used for funnel shift operations. The I1 pipeline contains a LZC (leading zero counting) unit. Furthermore, the two pipelines share a single 128-bit multimedia shifter.

These are configured dynamically into a single 128-bit execution pipe per instruction to execute the 128-bit Multimedia ALU, Shift and MAC instructions.

LS Pipe

The LS Pipe (Load/Store Pipe) contains logic to support 128-bit Load and Store instructions.

BR Pipe

The BR Pipe (Branch Pipe) contains logic to execute a Branch instruction.

C1 Pipe

The C1 Pipe contains logic to support a Floating-Point coprocessor unit (COP1=FPU).

C2 Pipe

The C2 Pipe contains logic to support a customer-specific coprocessor unit (COP2=VPU).

1.2.7. Operand/Bypass logic

The Operand/Bypass logic is a unit which takes data from the GPRs, Result Bus and Move Bus and routes the data to the pipelines and Scratchpad RAM.

1.2.8. Writeback Buffer

The Writeback Buffer (WBB) is an 8 entry by 16-byte (1-qword) FIFO storing data prior to accessing the CPU bus. It increases EE Core performance by decoupling the processor from the latencies of the CPU bus. The WBB also has a function for gathering uncached accelerated stores, that is, for gathering sequential stores less than a qword.

1.2.9. UCAB

The Uncached Accelerated Buffer (UCAB) is a 2 entry by 4-qword buffer. It caches 128 sequential bytes of data during an uncached accelerated load miss. If the address hits in the UCAB, the loads from the uncached accelerated space get the data from this buffer.

1.2.10. Result and Move Buses

The Result and Move Buses convey data between execution units, scratchpad RAM, the Data Cache and the Operand/Bypass logic unit.

1.2.11. Bus Interface Unit

The Bus Interface Unit (BIU) connects the EE Core to the rest of the system. It combines the Core's internal bus signals with the CPU Bus.

1.3. Superscalar Pipeline Operation

The EE Core has a six-stage superscalar pipeline. It can fetch, decode and execute a maximum of two instructions in parallel each cycle.
This section discusses in more detail the six physical pipelines described above. It also discusses how instructions are routed among pipes.

1.3.1. Integer Instruction Pipeline Stages

The EE Core contains four integer pipelines: the I0 and I1 pipes, and the LS and BR pipes. Each pipe consists of the following six stages, with each stage having 2 phases:

Symbol	Stage	Phase
1I	Instruction Fetch	Phase 1
2I	Instruction Fetch	Phase 2
1Q	Instruction Queue	Phase 1
2Q	Instruction Queue	Phase 2
1R	Register Fetch	Phase 1
2R	Register Fetch	Phase 2
1A	Execution	Phase 1
2A	Execution	Phase 2
1D	Data Fetch	Phase 1
2D	Data Fetch	Phase 2
1W	Write-back	Phase 1
2W	Write-back	Phase 2

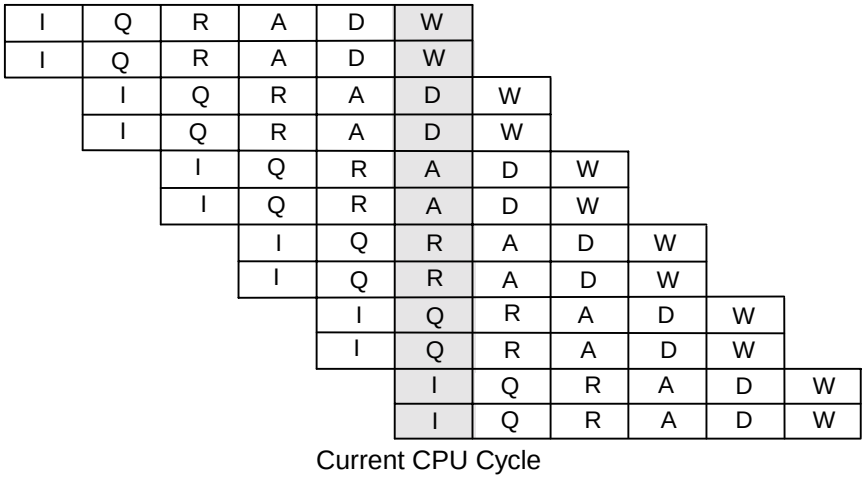


Figure 1-2 EE Core Integer Instruction Pipeline

Operations performed in each stage and phase are described as follows.

1I: Instruction Fetch, Phase 1

- The sequential address is calculated
- The branch address is calculated

2I: Instruction Fetch, Phase 2

- Selection of instruction addresses. Selects one of the following as the instruction address to be executed:
  - Sequential address
  - Branch/Jump address
  - Predicted Branch Target address from the BTAC
  - Exception vector address
  - EPC or Error EPC

1Q: Instruction Queue, Phase 1

- The instruction translation look-aside buffer (ITLB) does the virtual-to-physical address translation
- The Instruction Cache (data, Tag, S bits and BHT) fetch begins
- The BTAC read begins
- TLB read for instruction fetch starts

2Q: Instruction Queue, Phase 2

- The Instruction Cache fetch is completed
- TLB read completes
- The Instruction Cache Tag hit check is determined and either the cache or memory is selected
- The instructions are selected based on the S bit (SPRAM selection bit)
- The BTAC read is completed and, if there is a hit, an appropriate predicted target address is output

1R: Register Fetch, Phase 1

- Instructions are passed to the appropriate execution units
- The register file read is started
- Execution unit structural hazards are determined

2R: Register Fetch, Phase 2

- Instructions are decoded, data dependencies are determined and the appropriate instructions are issued
- The register file read is completed

1A: Execution, Phase 1

- Bypassing from D or W stage

2A: Execution, Phase 2

- The execution unit starts executing an instruction
- The integer arithmetic, logical, shift and multimedia instructions are completed
- The iterative steps of the Multiply, Multiply-Accumulate, or Divide instructions are executed
- The virtual address for load and store instructions is calculated
- The branch condition is determined
- The DTLB read starts
- The Data Cache or Scratchpad RAM read starts

1D: Data Fetch, Phase 1

- The DTLB read finishes
- The TLB read for a data access starts
- The Data Cache or Scratchpad RAM read is completed
- Data Cache Tag checking is completed
- Load or register data is obtained from COP1 and COP2
- COP0 registers are read

2D: Data Fetch, Phase 2

- The TLB read for a data access finishes
- Data alignment and way are selected for reading from the Data Cache
- Data alignment is done for reading from the Scratchpad RAM
- Data sign extension is done
- BHT bits and BTAC are updated
- Exceptions are detected

1W: Writeback, Phase 1

- Data is transferred to COP1 and COP2

2W: Writeback, Phase 2

- Results are written to the register file
- Data is written to the Data Cache or Scratchpad RAM
- The COP0, COP1 and COP2 registers are written

1.3.2. COP1 Pipeline

The COP1 pipeline consists of 8 stages and each stage has two phases, as shown in Figure 1-3.
COP1 instructions execute simultaneously in the integer pipeline I0 and the COP1 pipeline. With regard to descriptions such as "A/T", "D/X" and so on, as shown in Figure 1-3, the first letter identifies the main integer pipeline stage and the second letter identifies the COP1 pipeline stage.

Symbol	Stage	Phase
1I	Instruction Fetch	Phase 1
2I	Instruction Fetch	Phase 2
1Q	Instruction Queue	Phase 1
2Q	Instruction Queue	Phase 2
1R	Register Fetch	Phase 1
2R	Register Fetch	Phase 2
1T	COP1 Register Fetch	Phase 1
2T	COP1 Register Fetch	Phase 2
X	Execution 1	
Y	Arithmetic/ALU 1	
Z	Arithmetic/ALU 2	
1S	Result writing	Phase 1
2S	Result writing	Phase 2

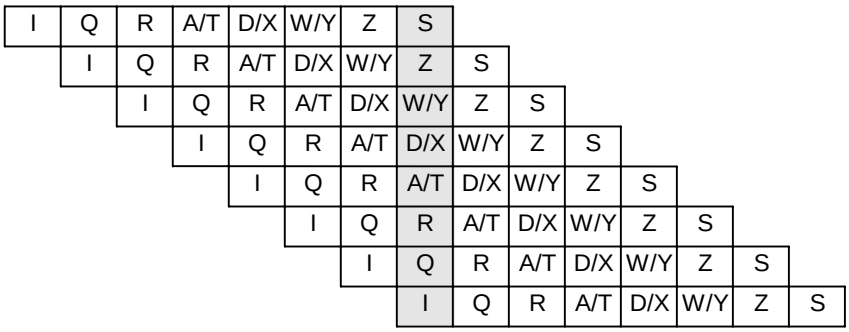


Figure 1-3 COP1 Pipeline

The operations of the I, Q and R stages are the same as those of the integer pipeline. The following describes operations in the stages specific to the COP1 pipeline.

1T: COP1 Register Fetch, Phase 1

- Register file read for operands

2T: COP1 Register Fetch, Phase 2

- Bypasses from the S Stage/W Stage for S/T overlap.

X: Execution 1

As the first phase for Multiply Operations, the following occurs. ALU, Conversion and Min/Max instructions have no operations.

- For sign, exclusive-OR is performed.
- For exponent, biasing is performed.
- For significand, the Booth function/Wallace multiplication is performed.

Y: Arithmetic / ALU Processing 1

As the second phase for Multiply Operations, the following occurs.

- Overflow/underflow on exponent is tested.
- Normalization for multiplication is done.

Also, as the first stage for ALU operations, the following occurs.

- Exponents are compared to determine the alignment.
- For floating-point to integer conversion, the alignment step is performed.
- For integer to floating-point conversion, the shift amount is determined.

Z: Arithmetic / ALU Processing 2

As the second phase of ALU operations, the following occurs. For Multiply Operations, this stage is no-op.

- Overflow/underflow detection
- Exponent readjustment
- Significand addition
- Exponent renormalization
- For floating-point to integer conversion and 2's complement instructions, overflow test is performed.
- For integer to floating-point conversion and shift instructions, exponent adjustment is performed.
- For the Compare instruction, the comparison is done.

1S: Result Writing, Phase 1

During the 1 S Stage, the results are available for computations.

2S: Result Writing, Phase 2

- FPR registers are written
- FCR31 is updated
- Bypass values are passed to the 2T stage

1.3.3. COP2 Pipeline

For COP2 pipeline operations, refer to the supplementary volume, "VU User's Manual". COP2 instructions execute simultaneously in the main integer pipeline I1 and the COP2 pipeline.

1.3.4. Classification and Routing of Instructions

Instruction routing, or what physical pipes are used for instructions of each category, is shown in Table 1-1. Instructions which require more than one execution pipeline are identified with *. For example, COP1 Move is executed in both the LS and the C1 pipelines. On the other hand, the ALU instructions are executed in either the I0 or the I1 pipeline. Figure 1-4 illustrates the contents of Table 1-1, adding the relation with logical pipes.

Instruction Categories	Instructions	Physical Pipeline					
		I1	I0	LS	BR	C1	C2
Load/Store	Load/Store, 128-bit Load/Store, Prefetch, CACHE			O			
SYNC	Synchronization	O					
LZC	Leading Zero Count	O					
ERET	Exception return	O					
SA Operate	Move to/from SA register		O				
COP0	COP0 move, COP0 operation	O					
COP1 Move	COP1 move, COP1 Load/Store			*		*	
COP2 Move	COP2 move, COP2 Load/Store			*			*
COP1 Operate	COP1 operation		*			*	
COP2 Operate	COP2 operation		*				*
ALU	Arithmetic, Shift, Logical, Trap, SYSCALL, BREAK	O	O				
MAC0	Multiply and Multiply-Accumulate for HI/LO register, Move to/from HI/LO		O				
MAC1	Multiply and Multiply-Accumulate for HI1/LO1 register, Move to/from HI1/LO1	O					
Branch	Branch and Jump				O		
Wide Operate	128-bit multimedia instructions	*	*				

Table 1-1 Categories of Instructions and Routing to Physical pipes

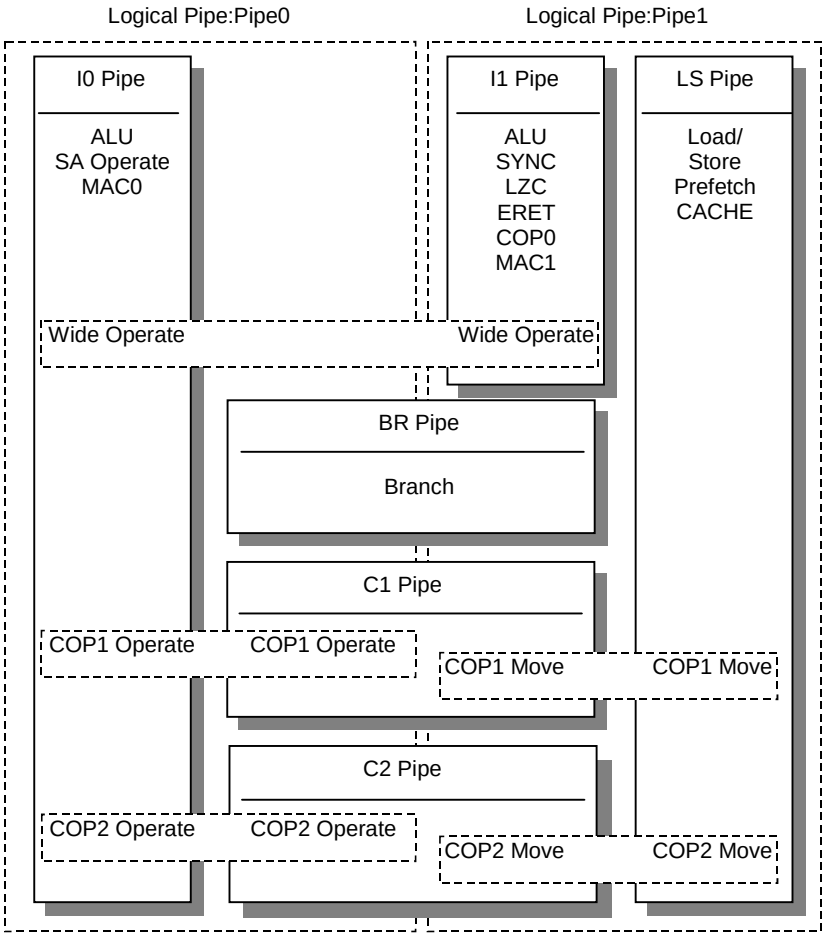


Figure 1-4 Instruction Routing in Logical and Physical Pipes

1.3.5. Instruction Issue Combinations

The EE Core always fetches 2 instructions and tries to issue 2 instructions in each cycle whenever it can. If an instruction can't be issued owing to data dependency or other reasons, a pair of staging registers is used as a buffer to connect between the Q and the R stage. If 2 instructions can't be issued in a particular cycle, 1 instruction is saved in the staging registers. In the next cycle, again fetches 2 instructions and tries to issue 2 instructions. The first one is what is left over in the staging register from the previous cycle; the other is what is newly fetched.

The instructions that get issued go to the R-stage of the pipeline and get associated with one of 2 logical pipes: Pipe 0 or Pipe 1. The instructions are then routed to an appropriate physical pipe for processing.

Instruction categories that can get issued to Pipe 0 and Pipe 1 are shown below. In other words, Pipe 0 and Pipe 1 are composed of the IO, C1, C2 and BR pipes and the I1, LS, C1, C2 and BR pipes respectively.

Instruction Categories	Logical Pipes Pipe 0	Pipe 1
Load/Store		O
SYNC		O
LZC		O
ERET		O
SA Operate	O	
COP0		O
COP1 Move		O
COP2 Move		O
COP1 Operate	O	
COP2 Operate	O	
ALU	O	O
MAC0	O	
MAC1		O
Branch	O	O
Wide Operate	O	

Table 1-2 Instruction categories and Routing to Logical Pipes

The ALU and Branch instruction categories can get issued to either Pipe 0 or Pipe 1. The binding of these 2 instructions is determined at instruction issue time.
In the MIPS ISA, placing an instruction from either the Branch or ERET category in the branch delay slot of the Branch category instruction is not allowed. That is, the following sequences are illegal and should not be issued.

1. Branch – Branch
2. Branch – ERET

The following sequences of instructions are also not allowed in the EE Core. (Though the Branch-Likely instruction category is a subset of the Branch instruction category, this limitation is restricted to the Branch-Likely instruction category.)

1. Branch – SYNC.P
2. Branch – SYNC.L
3. Branch-Likely – MTSA
4. Branch-Likely – MTSAB
5. Branch-Likely – MTSAH

Table 1-3 shows the instruction categories which can be issued concurrently to the 2 logical pipes.
"X" indicates a combination in which an instruction can not be issued concurrently. "Y" indicates a combination in which an instruction can be issued concurrently (i.e. enter the R stage), but it stalls for a single cycle in the A stage because of a resource hazard in the previous instruction.

		Pipe 0						
		SA Oper.	**COP1 Oper.**	**COP2 Oper.**	**ALU**	**MAC0**	**Branch**	**Wide Oper.**
Pipe 1	Load/Store	O	O	O	O	O	O	O
	ERET	O	O	O	O	O	**X**	O
	SYNC	O	O	O	O	O	O	O
	LZC	O	O	O	O	O	O	**Y**
	COP1 Move	O	**Y**	O	O	O	O	O
	COP2 Move	O	O	**Y**	O	O	O	O
	ALU	O	O	O	O	O	O	**Y**
	MAC1	O	O	O	O	O	O	**Y**
	Branch	O	O	O	O	O	**X**	O
	COP0	O	O	O	O	O	O	O

Table 1-3 Concurrently Issued Instruction Categories

1.4. Registers

The EE Core has a register set which extends the normal MIPS-compatible register set.
General purpose registers (GPRs), are extended from 64 bits to 128 bits, and a pair of HI/LO registers for the I1 pipe and the SA register for the funnel shift instructions are added.

1.4.1. CPU Registers

The EE Core has 128-bit wide general-purpose registers (GPRs). The upper 64 bits of the GPRs are only used by the EE Core-specific 128-bit Multimedia instructions (Parallel instructions).
HI1 and LO1 (which are the upper 64 bits of the 128-bit HI/LO registers respectively) are also used by the EE Core-specific multiply and divide instructions, such as MULT1, MULTU1, DIV1, DIVU1, MADD1, MADDU1, MFHI1, MFLO1, MTHI1 and MTLO1. These are not parallel instructions, but I1 pipe-specific instructions.

1.4.2. FPU Registers

The floating-point unit (FPU=COP1) has 32-bit wide floating-point registers, 2 floating-point control registers and a single 32-bit accumulator.

1.4.3. COP0 Registers

Coprocessor 0 (COP0) registers are shown in Table 1-4.

No.	Register Name	Description	Purpose
0	Index	Index of TLB entry	MMU
1	Random	Random index of TLB entry	MMU
2	EntryLo0	Low half of TLB for even PFN (Physical page number)	MMU
3	EntryLo1	Low half of TLB for odd PFN (Physical page number)	MMU
4	Context	Pointer to page table	Exception
5	PageMask	Mask that sets the TLB page size	MMU
6	Wired	Number of wired TLB entries	MMU
7	(Reserved)	Undefined	Undefined
8	BadVAddr	Virtual address for generating exception	Exception
9	Count	Timer count	Exception
10	EntryHi	High half (Virtual page number and ASID) of TLB entry	MMU
11	Compare	Timer compare value	Exception
12	Status	COP0 status	Exception
13	Cause	Cause of the last exception taken	Exception
14	EPC	Level 1 Exception Program Counter	Exception
15	PRId	Processor revision number	MMU
16	Config	Configuration Register	MMU
17	(Reserved)	Undefined	Undefined
18	(Reserved)	Undefined	Undefined
19	(Reserved)	Undefined	Undefined
20	(Reserved)	Undefined	Undefined
21	(Reserved)	Undefined	Undefined
22	(Reserved)	Undefined	Undefined
23	(Reserved)	Undefined	Undefined
24	Debug	Register for debug	Debug
25	Perf	Performance Counter and Control Register	Exception
26	(Reserved)	Undefined	Undefined
27	(Reserved)	Undefined	Undefined
28	TagLo	Cache Tag register (Low bits)	MMU
29	TagHi	Cache Tag register (High bits)	MMU
30	Error EPC	Level 2 Exception Program Counter	Exception
31	(Reserved)	Undefined	Undefined

Table 1-4 COP0 Registers

1.5. Memory Management

The EE Core processor provides a memory management unit (MMU), which uses an on-chip translation look-aside buffer (TLB) to translate virtual addresses into physical addresses.
The EE Core supports the MIPS-compatible 32-bit address and 64-bit data mode. Only 32-bit virtual and physical addresses have been implemented. There is no requirement for address sign extension. Address error exception checking will not be done on the upper 32 bits (which are ignored). The only condition that will generate the address error exception will be address alignment errors and segment protection errors.
Since there is only one addressing mode, four MIPS ISAs (I, II, III, IV) and an EE Core specific ISA are available without any restrictions in all of the three processor modes (with the appropriate MIPS ISA coprocessor usable restrictions).
The reserved instruction (RI) exception will occur only when the processor really tries to execute an undefined opcode.
Features of the memory management of the EE Core are as follows;

- MIPS III-compatible 32-bit MMU (with special bit defined for scratchpad RAM)
- Operation Modes: User, Supervisor and Kernel
- TLB: 48 entries of even/odd page pairs (96 pages)
  Full set associative
- Page Size: 4 KB, 16 KB, 64 KB, 256 KB, 1 MB, 4 MB, 16 MB
- ITLB: 2 entries
- DTLB: 4 entries
- Address Sizes: Virtual Address Size = 32
  Physical Address Size = 32

1.6. Cache Memory and Scratchpad RAM

The EE Core has an instruction cache and a separate data cache. It also has a scratchpad RAM for fast manipulation of large data structures.
The following are the main features of the caches:

- Separate Instruction Cache and Data Cache
- Virtually indexed cache and physically tagged cache
- Write-back policy for the Data Cache
- Data Cache and Instruction Cache burst read sequential ordering
- Cache Size: Instruction Cache: 16 KB
  Data Cache: 8 KB
  Scratchpad RAM: 16 KB
- Line size: 64 bytes
- Refill size: 64 bytes
- The number of way: 2-way set-associative
- Write Policy: Write-back
- Data order for block reads: Sequential ordering
- Data order for block writes: Sequential ordering
- Instruction cache miss restart: After all data received
- Data cache miss restart: Early restart on first quadword
- Cache parity: No
- Cache Locking: Data Cache Line Lock.
  Controlled by Cache instruction
- Cache Snooping: No
- Scratchpad RAM snooping: No
- Non-blocking load: Yes
- Hit Under Miss support: Yes
- Data Cache Prefetch: Yes

The following are the features of the Scratchpad RAM (SPRAM):

- 1024 x 128 bits (16 KB) static RAM
- External DMA read and write capability
- Accessible to software through load/store instructions

1.7. Bus Interface

The EE Core is connected to the rest of the system and to external devices, through the group of on-chip system bus signals called the CPU Bus. The features of the CPU Bus are listed below.

- Separate data and address buses (Demultiplexed operation)
- 128-bit data bus
- Clocked synchronous operations
- 8/16/32/64/128-bit burst access
- Multimaster capability
- Pipelined operations
- No turn-around or dead cycles between transfers

Note that cache coherency support and split transactions are not provided.

1.8. Floating-Point Unit

The floating-point (COP1) unit is a unit implementing a single-precision floating-point operation. This unit is not IEEE 754 compatible. The features are as follows:

- High-performance single-precision floating-point unit tightly coupled to the EE Core
- Supports single-precision format as defined in the IEEE 754 specification
- Plus/Minus "0" in line with the IEEE 754 specification are supported
- NaNs and plus/minus infinities are not supported
- No hardware exception mechanism to affect instruction execution
- Compatibility with COP2 (VPU)

1.9. Debug and Tracing Functions

The EE Core has the following debug support features:

- Instruction Address Breakpoint
- Data Address Breakpoint
- Data Value Breakpoint
- Each breakpoint is individually available and can set Mask
- Breakpoints can be available in different processor modes (User, Supervisor, Kernel and Exception modes)

2. Instruction Set Overview

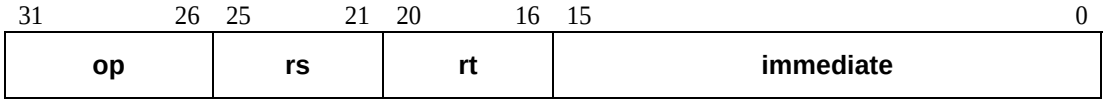
This chapter provides an overview of the EE Core instruction set. Refer to the supplementary volume "EE Core Instruction Set Manual" for detailed descriptions of individual instructions.
The EE Core supports all MIPS III instructions with the exception of 64-bit multiply, 64-bit divide, Load-Linked and Store Conditional instructions. It also implements additional EE Core-specific instructions, such as Multiply/Add and multimedia instructions.
The instruction set of the EE Core can be divided into the following groups:

- Load and Store instructions
- Computational instructions
- Branch and Jump instructions
- Special instructions
- Serialization instructions
- MIPS IV instructions
- System Control Coprocessor (COP0) instructions
- Coprocessor (COP1 / COP2) instructions
- EE Core-specific instructions

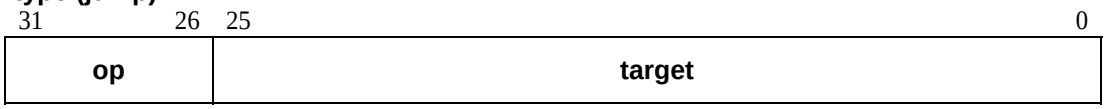
2.1. Binary Formats

There are three instruction formats in the EE Core: I-type (immediate), J-type (Jump), R-type (register), as shown below.

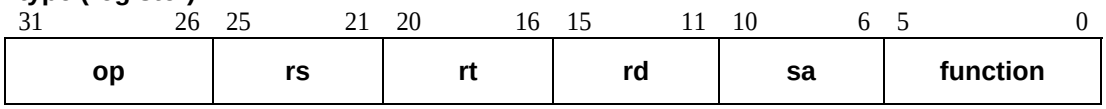
I-type (immediate)



J-type (jump)



R-type (register)



The contents of each field are as follows.

Field Name	Width	Contents
op	6 bits	Opcode
rs	5 bits	Source register specifier
rt	5 bits	Specifies target (source/destination) register or branch condition
immediate	16 bits	Immediate value, branch instruction offset or offset address
target	26 bits	Jump target address
rd	5 bits	Destination register specifier
sa	5 bits	Shift amount
function	6 bits	Function field

Refer to the supplementary volume "EE Core Instruction Set Manual" for the actual values.

2.2. Instruction Set Summary

The EE Core supports most of the MIPS III instructions and some of the MIPS IV instructions. In addition, EE Core-specific instructions (e.g. Multiply-Add) are implemented.
The EE Core instruction set is listed below.

Load/Store instructions

Mnemonic	Description	MIPS ISA
LB	Load Byte	MIPS I
LBU	Load Byte Unsigned	MIPS I
LD	Load Doubleword	MIPS III
LDL	Load Doubleword Left	MIPS III
LDR	Load Doubleword Right	MIPS III
LH	Load Halfword	MIPS I
LHU	Load Halfword Unsigned	MIPS I
LW	Load Word	MIPS I
LWL	Load Word Left	MIPS I
LWR	Load Word Right	MIPS I
LWU	Load Word Unsigned	MIPS III
SB	Store Byte	MIPS I
SD	Store Doubleword	MIPS III
SDL	Store Doubleword Left	MIPS III
SDR	Store Doubleword Right	MIPS III
SH	Store Halfword	MIPS I
SW	Store Word	MIPS I
SWL	Store Word Left	MIPS I
SWR	Store Word Right	MIPS I
SYNC	Synchronization	MIPS II

Computational Instructions

Mnemonic	Description	MIPS ISA
ADD	Add	MIPS I
ADDI	Add Immediate	MIPS I
ADDIU	Add Immediate Unsigned	MIPS I
ADDU	Add Unsigned	MIPS I
AND	AND	MIPS I
ANDI	AND Immediate	MIPS I
DADD	Doubleword Add	MIPS III
DADDI	Doubleword Add Immediate	MIPS III
DADDIU	Doubleword Add Immediate (Unsigned)	MIPS III
DADDU	Doubleword Add Unsigned	MIPS III
DIV	Divide	MIPS I
DIVU	Divide Unsigned	MIPS I
DSLL	Doubleword Shift Left Logical	MIPS III
DSLL32	Doubleword Shift Left Logical +32	MIPS III
DSLLV	Doubleword Shift Left Logical Variable	MIPS III
DSRA	Doubleword Shift Right Arithmetic	MIPS III
DSRA32	Doubleword Shift Right Arithmetic +32	MIPS III
DSRAV	Doubleword Shift Right Arithmetic Variable	MIPS III
DSRL	Doubleword Shift Right Logical	MIPS III

Mnemonic	Description	MIPS ISA
DSRL32	Doubleword Shift Right Logical +32	MIPS III
DSRLV	Doubleword Shift Right Logical Variable	MIPS III
DSUB	Doubleword Subtract	MIPS III
DSUBU	Doubleword Subtract Unsigned	MIPS III
LUI	Load Upper Immediate	MIPS I
MFHI	Move From HI	MIPS I
MFLO	Move From LO	MIPS I
MTHI	Move To HI	MIPS I
MTLO	Move To LO	MIPS I
MULT	Multiply	MIPS I
MULTU	Multiply Unsigned	MIPS I
NOR	NOR	MIPS I
OR	OR	MIPS I
ORI	OR Immediate	MIPS I
SLL	Shift Left Logical	MIPS I
SLLV	Shift Left Logical Variable	MIPS I
SLT	Set on Less Than	MIPS I
SLTI	Set on Less Than Immediate	MIPS I
SLTIU	Set on Less Than Immediate Unsigned	MIPS I
SLTU	Set on Less Than Unsigned	MIPS I
SRA	Shift Right Arithmetic	MIPS I
SRAV	Shift Right Arithmetic Variable	MIPS I
SRL	Shift Right Logical	MIPS I
SRLV	Shift Right Logical Variable	MIPS I
SUB	Subtract	MIPS I
SUBU	Subtract Unsigned	MIPS I
XOR	Exclusive OR	MIPS I
XORI	Exclusive OR Immediate	MIPS I

Branch/Jump Instructions

Mnemonic	Description	MIPS ISA
J	Jump	MIPS I
JAL	Jump And Link	MIPS I
JALR	Jump And Link	MIPS I
JR	Jump Register	MIPS I
BEQ	Branch on Equal	MIPS I
BEQL	Branch on Equal Likely	MIPS II
BGEZ	Branch on Greater Than or Equal to Zero	MIPS I
BGEZAL	Branch on Greater Than or Equal to Zero And Link	MIPS I
BGEZALL	Branch on Greater Than or Equal to Zero And Link Likely	MIPS II
BGEZL	Branch on Greater Than or Equal to Zero Likely	MIPS II
BGTZ	Branch on Greater Than Zero	MIPS I
BGTZL	Branch on Greater Than Zero Likely	MIPS II
BLEZ	Branch on Less Than or Equal to Zero	MIPS I
BLEZL	Branch on Less Than or Equal to Zero Likely	MIPS II
BLTZ	Branch on Less Than Zero	MIPS I
BLTZAL	Branch on Less Than Zero and Link	MIPS I
BLTZALL	Branch on Less Than Zero And Link Likely	MIPS II
BLTZL	Branch on Less Than Zero Likely	MIPS II
BNE	Branch on Not Equal	MIPS I

Mnemonic	Description	MIPS ISA
BNEL	Branch on Not Equal Likely	MIPS II

Special Instructions

Mnemonic	Description	MIPS ISA
SYSCALL	System Call	MIPS I
BREAK	Break	MIPS I
TGE	Trap if Greater Than or Equal	MIPS II
TGEU	Trap if Greater Than or Equal Unsigned	MIPS II
TLT	Trap if Less Than	MIPS II
TLTU	Trap if Less Than Unsigned	MIPS II
TEQ	Trap if Equal	MIPS II
TNE	Trap if Not Equal	MIPS II
TGEI	Trap if Greater Than or Equal Immediate	MIPS II
TGEIU	Trap if Greater Than or Equal Immediate Unsigned	MIPS II
TLTI	Trap if Less Than Immediate	MIPS II
TLTIU	Trap if Less Than Immediate Unsigned	MIPS II
TEQI	Trap if Equal Immediate	MIPS II
TNEI	Trap if Not Equal Immediate	MIPS II

MIPS IV Instruction

Mnemonic	Description	MIPS ISA
MOVN	Move on Register Not Equal to Zero	MIPS IV
MOVZ	Move on Register Equal to Zero	MIPS IV
PREF	Prefetch	MIPS IV

The EE Core does not support the following MIPS III instructions:

64-bit multiply and divide instructions (DMULT, DMULTU, DDIV, DDIVU) and Semaphore instructions (LL, LLD, SC, SCD)

2.3. Load/Store Instructions

Load/Store instructions transfer different sizes of data—bytes, halfwords, words and doublewords—between memory and registers. Signed and unsigned integers are supported by load instructions that are sign-extended or zero-extended.

The binary formats of load and store instructions are I-type (Immediate) and the only addressing mode is "base register plus 16-bit signed immediate offset" mode.

Note that the EE Core does not support Load-Linked and Store Conditional instructions, LL, LLD, SC and SCD in MIP III instructions.

The list of load/store instructions is shown below. In addition, refer to "2.10. Coprocessor Instructions (COP1/COP2)" about load/store instructions related to the coprocessor.

Mnemonic	Description	MIPS ISA
LB	Load Byte	MIPS I
LBU	Load Byte Unsigned	MIPS I
LD	Load Doubleword	MIPS III
LDL	Load Doubleword Left	MIPS III
LDR	Load Doubleword Right	MIPS III
LH	Load Halfword	MIPS I
LHU	Load Halfword Unsigned	MIPS I
LW	Load Word	MIPS I
LWL	Load Word Left	MIPS I
LWR	Load Word Right	MIPS I
LWU	Load Word Unsigned	MIPS III
SB	Store Byte	MIPS I
SD	Store Doubleword	MIPS III
SDL	Store Doubleword Left	MIPS III
SDR	Store Doubleword Right	MIPS III
SH	Store Halfword	MIPS I
SW	Store Word	MIPS I
SWL	Store Word Left	MIPS I
SWR	Store Word Right	MIPS I
SYNC	Synchronization	MIPS II

Table 2-1 Load/Store instructions

2.3.1. Data Formats and Alignment

The EE Core uses the following five data formats:

- 128-bit quadword
- 64-bit doubleword
- 32-bit word
- 16-bit halfword
- 8-bit byte

Byte ordering in data formats of halfword or greater is configured in little-endian order . Byte 0 is always the least significant (rightmost) byte, which is compatible with iAPX® x 86 and DEC VAX® conventions.

Bit ordering is configured in little endian and bit 0 always indicates the least significant (rightmost) bit (although no instructions explicitly designate bit positions within words).

Refer to Figure 2-1 and Figure 2-2 about the word/byte ordering and bit ordering respectively.

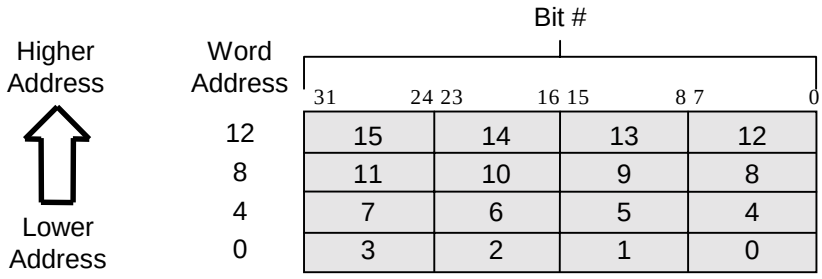


Figure 2-1 Little-Endian Byte Ordering

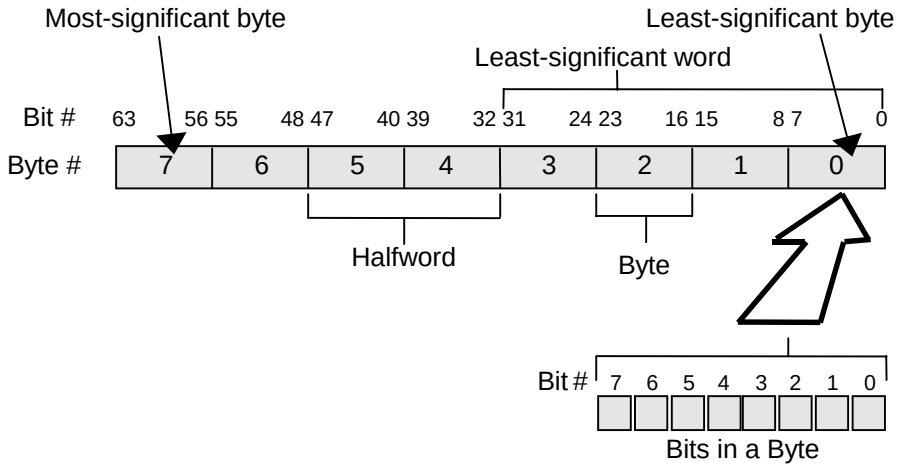


Figure 2-2 Little-Endian Data in a Doubleword

The EE Core uses byte addressing for all data access. Data of a halfword or greater has the following alignment constraints. Any access that does not satisfy these constraints will generate address error exceptions.

Data Formats	Condition
Halfword	Even address (0, 2, 4...)
Word	Address divisible by four (0, 4, 8...)
Doubleword	Address divisible by eight (0, 8, 16...)
Quadword	Address divisible by sixteen (0, 16, 32...)

The following special instructions load and store words, doublewords or quadwords that are not aligned according to the above constraints.

Mnemonic	Description
LWL	Load Word Left
LWR	Load Word Right
SWL	Store Word Left
SWR	Store Word Right
LDL	Load Doubleword Left
LDR	Load Doubleword Right
SDL	Store Doubleword Left
SDR	Store Dobuleword Right

In order to load/store misaligned data, these instructions have to be used in pairs. Therefore, one additional instruction cycle is necessary compared to aligned data. Refer to Figure 2-3.

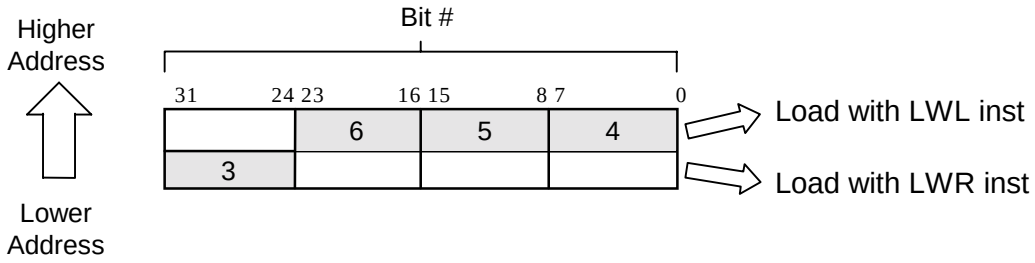


Figure 2-3 Little-Endian Misaligned Word Addressing

If LQ and SQ instructions are combined with a QFSRV instruction (funnel shift), extracting aligned data from a misaligned quadword is possible.

2.3.2. Load Delay

In general, when the data loaded by an instruction is not allowed to be used by the immediately following instruction, the load instruction is called a delayed load instruction; the delay until the data can be used is called the load delay slot.

In the EE Core, there is no absolute delayed load instruction. Loaded data is allowed to be used in the instruction following a load instruction. In such cases, however, hardware interlocks insert additional clock cycles. Consequently, taking a load delay into account in programming is not required. However, for software performance, it is desirable to place an instruction that does not use the data immediately following a load instruction.

2.4. Computational Instructions

EE Core computational instructions can be of either the R-type or I-type format. Both operands are registers in the R-type format, and one operand is a 16-bit immediate in the I-type format.
Computational instructions perform the following operations on register values:

- Arithmetic
- Logical
- Shift
- Multiply
- Divide

Computational instructions are divided into the following four categories:

- ALU immediate instructions
- Three-Operand Register-Type instructions
- Shift instructions
- Multiply and Divide instructions

The EE Core does not support the 64-bit Multiply and Divide instructions (DMULT, DMULTU, DDIV and DDIVU) in the MIPS III instruction set. In 64-bit operations, sign-extended or zero-extended 32-bit values are required. If using an incorrect 64-bit value, the result is unpredictable.

Mnemonic	Description	MIPS ISA
ADD	Add	MIPS I
ADDI	Add Immediate	MIPS I
ADDIU	Add Immediate Unsigned	MIPS I
ADDU	Add Unsigned	MIPS I
AND	AND	MIPS I
ANDI	AND Immediate	MIPS I
DADD	Doubleword Add	MIPS III
DADDI	Doubleword Add Immediate	MIPS III
DADDIU	Doubleword Add Immediate Unsigned	MIPS III
DADDU	Doubleword Add Unsigned	MIPS III
DIV	Divide	MIPS I
DIVU	Divide Unsigned	MIPS I
DSLL	Doubleword Shift Left Logical	MIPS III
DSLL32	Doubleword Shift Left Logical +32	MIPS III
DSLLV	Doubleword Shift Left Logical Variable	MIPS III
DSRA	Doubleword Shift Right Arithmetic	MIPS III
DSRA32	Doubleword Shift Right Arithmetic +32	MIPS III
DSRAV	Doubleword Shift Right Arithmetic Variable	MIPS III
DSRL	Doubleword Shift Right Logical	MIPS III
DSRL32	Doubleword Shift Right Logical +32	MIPS III
DSRLV	Doubleword Shift Right Logical Variable	MIPS III
DSUB	Doubleword Subtract	MIPS III
DSUBU	Doubleword Subtract Unsigned	MIPS III
LUI	Load Upper Immediate	MIPS I
MFHI	Move From HI	MIPS I
MFLO	Move From LO	MIPS I

Mnemonic	Description	MIPS ISA
MTHI	Move To HI	MIPS I
MTLO	Move To LO	MIPS I
MULT	Multiply	MIPS I
MULTU	Multiply Unsigned	MIPS I
NOR	NOR	MIPS I
OR	OR	MIPS I
ORI	OR Immediate	MIPS I
SLL	Shift Left Logical	MIPS I
SLLV	Shift Left Logical Variable	MIPS I
SLT	Set on Less Than	MIPS I
SLTI	Set on Less Than Immediate	MIPS I
SLTIU	Set on Less Than Immediate Unsigned	MIPS I
SLTU	Set on Less Than Unsigned	MIPS I
SRA	Shift Right Arithmetic	MIPS I
SRAV	Shift Right Arithmetic Variable	MIPS I
SRL	Shift Right Logical	MIPS I
SRLV	Shift Right Logical Variable	MIPS I
SUB	Subtract	MIPS I
SUBU	Subtract Unsigned	MIPS I
XOR	Exclusive OR	MIPS I
XORI	Exclusive OR Immediate	MIPS I

Table 2-2 Computational Instructions

2.5. Branch/Jump Instructions

The EE Core instruction set defines the following instructions to change the flow of control in programming: PC-relative conditional branches, a PC-region unconditional jump, an absolute register unconditional jump, and a corresponding subroutine call (which records the return link address in a general-purpose register). There are two different conditional branch instructions: Branch and Branch-Likely, as described below.

Mnemonic	Description	MIPS ISA
J	Jump	MIPS I
JAL	Jump And Link	MIPS I
JALR	Jump And Link Register	MIPS I
JR	Jump Register	MIPS I
BEQ	Branch on Equal	MIPS I
BEQL	Branch on Equal Likely	MIPS II
BGEZ	Branch on Greater Than or Equal to Zero	MIPS I
BGEZAL	Branch on Greater Than or Equal to Zero And Link	MIPS I
BGEZALL	Branch on Greater Than or Equal to Zero And Link Likely	MIPS II
BGEZL	Branch on Greater Than or Equal to Zero Likely	MIPS II
BGTZ	Branch on Greater Than Zero	MIPS I
BGTZL	Branch on Greater Than Zero Likely	MIPS II
BLEZ	Branch on Less Than or Equal to Zero	MIPS I
BLEZL	Branch on Less Than or Equal to Zero Likely	MIPS II
BLTZ	Branch on Less Than Zero	MIPS I
BLTZAL	Branch on Less Than Zero and Link	MIPS I
BLTZALL	Branch on Less Than Zero and Link Likely	MIPS II
BLTZL	Branch on Less Than Zero Likely	MIPS II
BNE	Branch on Not Equal	MIPS I
BNEL	Branch on Not Equal Likely	MIPS II

Table 2-3 Branch and Jump Instructions

2.5.1. Branch Delay Slot

All branch instructions have a delay of one instruction (the branch delay slot). That is, the instruction immediately following the branch instruction is executed while the target instruction is fetched.
If a branch is taken when an unconditional or conditional branch is established, the instruction in the branch delay slot (immediately following the branch instruction) is executed before the branch operation.
If the branch is not taken and execution falls through, branch instructions execute the instruction in the delay slot, but the branch-likely instructions do not. (They are said to nullify it.)
By convention, if an instruction in the branch delay slot is suspended by an exception or interrupt, the program can be continued by re-executing the immediately preceding branch instruction. To permit this, branches must be restartable. That is, in procedure calls, the branch target must not be determined by the register in which the return link is stored (usually register 31).

2.5.2. Overview of Jump Instructions

Subroutine calls in high-level languages are usually implemented with Jump or Jump-Link instructions, both of which are J-type instructions. The 26-bit target address shifts left 2 bits and combines with the high-order 4 bits of the program counter to form the branch target address.

Returns, dispatches and cross-page jumps are usually implemented with the Jump-Register or Jump-Link-Register instructions. Both are R-type instructions, in which the value of one of the general-purpose registers is the branch target address.

2.5.3. Overview of Branch Instructions

Conditional statements in high-level languages are usually implemented with branch or branch-likely instructions, both of which are I-type instructions. The 16-bit offset shifts left 2 bits and is sign-extended to 32 bits, which is added to the instruction address of the branch delay slot, to form the branch target address. Branch instructions are different from branch-likely instructions in whether they execute the instruction in the branch delay slot when a condition is not taken. Branch instructions execute the instruction in the branch delay slot; branch-likely instructions nullify it.

2.6. Special Instructions

Special instructions allow the software to cause traps. The list is shown below. Refer to the "EE Core Instruction Set Manual" for further information.

Mnemonic	Description	MIPS ISA
SYSCALL	System Call	MIPS I
BREAK	Break	MIPS I
TGE	Trap if Greater Than or Equal	MIPS II
TGEU	Trap if Greater Than or Equal Unsigned	MIPS II
TLT	Trap if Less Than	MIPS II
TLTU	Trap if Less Than Unsigned	MIPS II
TEQ	Trap if Equal	MIPS II
TNE	Trap if Not Equal	MIPS II
TGEI	Trap if Greater Than or Equal Immediate	MIPS II
TGEIU	Trap if Greater Than or Equal Immediate Unsigned	MIPS II
TLTI	Trap if Less Than Immediate	MIPS II
TLTIU	Trap if Less Than Immediate Unsigned	MIPS II
TEQI	Trap if Equal Immediate	MIPS II
TNEI	Trap if Not Equal Immediate	MIPS II

Table 2-4 Special Instructions

2.7. Serialization Instruction

The order in which memory accesses from load and store instructions appear outside the EE Core is not specified by the EE Core architecture.
The order of loads and stores can be guaranteed at a point in a program by using a SYNC or SYNC.L instruction. That is, load and store instructions executed before the SYNC or SYNC.L instruction are guaranteed to retire before load and store instructions following the SYNC or SYNC.L instruction are executed.
In addition to load and store instructions, a SYNC.P instruction can be used to guarantee the completion of an instruction. Instructions executed before a SYNC.P instruction are completed before an instruction following SYNC.P is executed.

2.8. MIPS IV Instructions

The EE Core supports a part of the MIPS IV instruction set: Conditional move instructions and the Prefetch instruction.

Mnemonic	Description	MIPS ISA
MOVN	Move on Register Not Equal to Zero	MIPS IV
MOVZ	Move on Register Equal to Zero	MIPS IV
PREF	Prefetch	MIPS IV

Table 2-5 MIPS IV instruction supported in EE Core

Conditional Move operations allow "IF" statements to be described without branches. "THEN" and "ELSE" clauses are always computed and the results are placed in a temporary register. Then, appropriate results are transferred to the target register depending on the conditions.
The Prefetch instruction fetches data expected to be used in the near future and places it in the Data Cache.

2.9. System Control Coprocessor (COP0) Instructions

COP0 instructions perform operations specifically on the System Control Coprocessor to manipulate the memory management and exception handling facilities of the processor.
COP0 instructions are enabled when the processor is in Kernel mode or when bit 28 (CU [0]) is set in the STATUS register. Otherwise, executing the following instructions generates a "Coprocessor Unusable" exception.
Refer to the "EE Core Instruction Set Manual" for details of each COP0 instruction.

Mnemonic	Description
MFBPC	Move From Breakpoint Control
MFDAB	Move From Data Address Breakpoint
MFDABM	Move From Data Address Breakpoint Mask
MFIAB	Move From Instruction Address Breakpoint
MFIABM	Move From Instruction Address Breakpoint Mask
MFDVB	Move From Data Value Breakpoint
MFDVBM	Move From Data Value Breakpoint Mask
MTBPC	Move To Breakpoint Control
MTDAB	Move To Data Address Breakpoint
MTDABM	Move To Data Address Breakpoint Mask
MTIAB	Move To Instruction Address Breakpoint
MTIABM	Move To Instruction Address Breakpoint Mask
MTDVB	Move To Data Value Breakpoint
MTDVBM	Move To Data Value Breakpoint Mask
MFC0	Move From COP0
MFPC	Move From Performance Counter
MFPS	Move From Performance Event Specifier
MTC0	Move To COP0
MTPC	Move To Performance Counter
MTPS	Move To Performance Event Specifier
BC0F	Branch on COP0 False
BC0FL	Branch on COP0 False Likely
BC0T	Branch on COP0 True
BC0TL	Branch on COP0 True Likely
TLBR	Read Indexed TLB Entry
TLBP	Probe TLB for Matching Entry
ERET	Exception Return
EI	Enable Interrupt
DI	Disable Interrupt

Table 2-6 Coprocessor 0 instructions

2.10. Coprocessor Instructions (COP1/COP2)

COP1 and COP2 instructions perform operations in their respective coprocessors. Loads and stores are I-type and other instructions have coprocessor-dependent formats.

2.10.1. COP1(FPU) Instructions

The following are COP1 instructions.

Mnemonic	Description
LWC1	Load Word to FPR
SWC1	Store Word from FPR
MTC1	Move Word to FCR
MFC1	Move Word from FPR
CTC1	Move Control Word to FCR
CFC1	Move Control Word from FCR
CVT.S.W	32-bit Fixed Point Floating-Point Convert to Single Floating-Point
CVT.W.S	Single Floating-Point Convert to 32-bit Fixed Point
ADD.S	Single Floating-Point Add
SUB.S	Single Floating-Point Subtract
MUL.S	Single Floating-Point Multiply
DIV.S	Single Floating-Point Divide
ABS.S	Single Floating-Point Absolute
MOV.S	Single Floating-Point Move
NEG.S	Single Floating-Point Negate
SQRT.S	Single Floating-Point Square Root
C.cond.S	Single Floating-Point Compare
BC1T	Branch on FPU True
BC1F	Branch on FPU False
BC1TL	Branch on FPU True Likely
BC1FL	Branch on FPU False Likely
ADDA.S	Single Floating-Point Add to Accumulator
SUBA.S	Single Floating-Point Subtract to Accumulator
MULA.S	Single Floating-Point Multiply to Accumulator
MADD.S	Single Floating-Point Multiply and Add
MADDA.S	Single Floating-Point Multiply and Add to Accumulator
MSUB.S	Single Floating-Point Multiply and Subtract
MSUBA.S	Single Floating-Point Multiply / Subtract from Accumulator
RSQRT.S	Single Floating-Point Reciprocal Square Root
MAX.S	Single Floating-Point Maximum
MIN.S	Single Floating-Point Minimum

Table 2-7 Coprocessor instructions

2.10.2. COP2 Instructions

The following are COP2 instructions. Refer to the "VU User's Manual" for the details of COP2 instructions.

Mnemonic	Description
LQC2	Load Quadword to COP2
SQC2	Store Quadword from COP2
QMFC2	Quadword Move From COP2
CFC2	Move Control From COP2
QMTC2	Quadword Move To COP2
CTC2	Move Control To COP2
CALLMS	Call Micro Subroutine
CALLMSR	Call Micro Subroutine Register
WAITQ	Wait Q Register
BC2F	Branch on COP2 False
BC2FL	Branch on COP2 False Likely
BC2T	Branch on COP2 True
BC2TL	Branch on COP2 True Likely

Table 2-8 COP2 Instructions

2.10.3. VU Macro instructions

COP2 (VPU0) provides EE Core programs with a macro instruction set, with almost the same functionality as the VU-specific instruction set (micro instructions). The list of macro instructions is shown below. Refer to the "VU User's Manual" for the details of each instruction.

Instruction	Description
VABS	Absolute
VADD	Addition
VADDi	ADD broadcast I register
VADDq	ADD broadcast Q register
VADDbc	ADD broadcast bc field
VADDA	ADD output to ACC
VADDAi	ADD output to ACC broadcast I register
VADDAq	ADD output to ACC broadcast Q register
VADDAbc	ADD output to ACC broadcast bc field
VSUB	Subtraction
VSUBi	SUB broadcast I register
VSUBq	SUB broadcast Q register
VSUBbc	SUB broadcast bc field
VSUBA	SUB output to ACC
VSUBAi	SUB output to ACC broadcast I register
VSUBAq	SUB output to ACC broadcast Q register
VSUBAbc	SUB output to ACC broadcast bc field
VMU	Multiply
VMULi	MUL broadcast I register
VMULq	MUL broadcast Q register
VMULbc	MUL broadcast bc field
VMULA	MUL output to ACC
VMULAi	MUL output to ACC broadcast I register
VMULAq	MUL output to ACC broadcast Q register
VMULAbc	MUL output to ACC broadcast bc field

Instruction	Description
VMADD	MUL and ADD (SUB)
VMADDi	MUL and ADD (SUB) broadcast I register
VMADDq	MUL and ADD (SUB) broadcast Q register
VMADDbc	MUL and ADD (SUB) broadcast bc field
VMADDA	MUL and ADD (SUB) output to ACC
VMADDAi	MUL and ADD (SUB) output to ACC broadcast I register
VMADDAq	MUL and ADD (SUB) output to ACC broadcast Q register
VMADDAbc	MUL and ADD (SUB) output to ACC broadcast bc field
VMSUB	Multiply and SUB
VMSUBi	Multiply and SUB broadcast I register
VMSUBq	Multiply and SUB broadcast Q register
VMSUBbc	Multiply and SUB broadcast bc field
VMSUBA	Multiply and SUB output to ACC
VMSUBAi	Multiply and SUB output to ACC broadcast I register
VMSUBAq	Multiply and SUB output to ACC broadcast Q register
VMSUBAbc	Multiply and SUB output to ACC broadcast bc field
VMAX	Maximum
VMAXi	Maximum broadcast I register
VMAXbc	Maximum broadcast bc field
VMINI	Minimum
VMINIi	Minimum broadcast I register
VMINIbc	Minimum broadcast bc field
VOPMULA	Outer product MULA
VOPMSUB	Outer product MSUB
VNOP	No Operation
VFTOI0	Float to Integer, fixed point 0-bit
VFTOI4	Float to Integer, fixed point 4-bit
VFTOI12	Float to Integer, fixed point 12-bit
VFTOI15	Float to Integer, fixed point 15-bit
VITOF0	Integer to Float, fixed point 0-bit
VITOF4	Integer to Float, fixed point 4-bit
VITOF12	Integer to Float, fixed point 12-bit
VITOF15	Integer to Float, fixed point 15-bit
VCLIP	Clipping
VDIV	Floating Divide
VSQRT	Floating Square-root
VRSQRT	Floating reciprocal Square-root
VIADD	Integer ADD
VIADDI	Integer ADD immediate
VIAND	Integer AND
VIOR	Integer OR
VISUB	Integer SUB
VMOVE	Move Floating register
VMFIR	Move From integer register
VMTIR	Move To integer register
VMR32	Rotate right 32 bits
VLQD	Load Quadword with pre-decrement
VLQI	Load Quadword with post-increment
VSQD	Store Quadword with pre-decrement
VSQI	Store Quadword with post-increment
VILWR	Integer load word register
VISWR	Integer store word register
VRINIT	Random-unit init R register

Instruction	Description
VRGET	Random-unit get R register
VRNEXT	Random-unit next M sequence
VRXOR	Random-unit XOR R register

Table 2-9 VU Macro Instruction

2.11. EE Core-Specific Instructions

The EE Core extends its instruction set from the original MIPS architecture. The following instructions are supported in particular:

- Three-operand Multiply and Multiply-Add instructions
- Multiply and divide instruction for Pipeline I1
- Multimedia instructions
- Enable interrupt and Disable interrupt instructions

Refer to the "EE Core Instruction Set Manual" for more information about each instruction.

2.11.1. EE Core-Specific Multiply / Divide Instructions

The standard MIPS instructions for multiply, divide and move to / from HI / LO registers execute on the I0 pipeline's MAC unit. A set of new instructions has also been defined to execute on the I1 pipeline's MAC unit:

Mnemonic	Description
MADD	Multiply/Add
MADDU	Multiply/Add Unsigned
MULT	Multiply (3-operand)
MULTU	Multiply Unsigned (3-operand)
MULT1	Multiply 1
MULTU1	Multiply Unsigned 1
DIV1	Divide 1
DIVU1	Divide Unsigned 1
MADD1	Multiply/Add 1
MADDU1	Multiply/Add Unsigned 1
MFHI1	Move From HI1
MFLO1	Move From LO1
MTHI1	Move To HI1
MTLO1	Move To LO1

Table 2-10 EE Core-Specific Multiply / Divide Instructions

The EE Core supports three-operand multiply instructions that store the multiply result to a general-purpose register in addition to the LO register. These instructions don't have to use the MFLO instruction to move data from the LO register to a general-purpose register.

- MULT rd, rs, rt HI | | LO = rs x rt (signed)
  rd = new LO contents
- MADDU rd, rs, rt HI | | LO += rs x rt (unsigned)
  rd = new LO contents

The EE Core also supports new multiply-add instructions, MADD and MADDU. These instructions execute multiply-accumulate operations using the HI and LO registers as accumulators.

- MADD rd, rs, rt HI | | LO += rs x rt (signed)
  rd = new LO contents
- MADDU rd, rs, rt HI | | LO += rs x rt (unsigned)
  rd = new LO contents

Latencies and throughputs of the integer multiply / divide instructions are shown in Table 2-11.

Mnemonic	Operand	Latency	Throughput
MULT, MULTU, MULT1, MULTU1	Two 32-bit operands	4	2
MADD, MADDU, MADD1, MADDU1	Two 32-bit operands	4	2
DIV, DIVU, DIV1, DIVU1	Two 32-bit operands	37	37
PMULTW, PMULTUW, PMADDW, PMADDUW, PMSUBW	Four 32-bit operands	4	2
PMULTH, PMADDH, PMSUBH, PHMADH, PHMSUBH	Eight 16-bit operands	4	2
PDIVW, PDIVUW	Two 32-bit operands	37	37
PDIVBW	Eight 32-bit & 16-bit operands	37	37

Table 2-11 Integer Multiply / Divide Operation Cycles

2.11.2. Multimedia Instructions

The EE Core implements a new set of instructions to support multimedia applications. (Table 2-12) Most of these instructions do parallel operations by combining the I0 and I1 pipelines. Instructions do parallel operations on either two 64-bit data items, four 32-bit data items, eight 16-bit data items or sixteen 8-bit data items to form a 128-bit path.

In order to support the 128-bit datapath, 128-bit load/store instructions are also implemented.

Category	Mnemonic	Description
Arithmetic	PADDB	Parallel Add Byte
	PSUBB	Parallel Subtract Byte
	PADDH	Parallel Add Halfword
	PSUBH	Parallel Subtract Halfword
	PADDW	Parallel Add Word
	PSUBW	Parallel Subtract Word
	PADSBH	Parallel Add/Subtract Halfword
	PADDSB	Parallel Add with Signed Saturation Byte
	PSUBSB	Parallel Subtract with Signed Saturation Byte
	PADDSH	Parallel Add with Signed Saturation Halfword
	PSUBSH	Parallel Subtract with Signed Saturation Halfword
	PADDSW	Parallel Add with Signed Saturation Word
	PSUBSW	Parallel Subtract with Signed Saturation Word
	PADDUB	Parallel Add with Signed Saturation Byte
	PSUBUB	Parallel Subtract with Signed Saturation Byte
	PADDUH	Parallel Add with Unsigned Saturation Halfword
	PSUBUH	Parallel Subtract with Unsigned Saturation Halfword
	PADDUW	Parallel Add with Unsigned Saturation Word
	PSUBUW	Parallel Subtract with Unsigned Saturation Word
Min/Max	PMAXH	Parallel Maximum Halfword
	PMINH	Parallel Minimum Halfword
	PMAXW	Parallel Maximum Word
	PMINW	Parallel Minimum Word
Absolute	PABSH	Parallel Absolute Halfword
	PABSW	Parallel Absolute Word
Multiply and Divide	PMULTW	Parallel Multiply Word
	PMULTUW	Parallel Multiply Unsigned Word
	PDIVW	Parallel Divide Word

Category	Mnemonic	Description
	PDIVUW	Parallel Divide Unsigned Word
	PMADDW	Parallel Multiply/Add Word
	PMADDUW	Parallel Multiply/Add Unsigned Word
	PMSUBW	Parallel Multiply/Subtract Word
	PMFHI	Parallel Move From HI
	PMFLO	Parallel Move From LO
	PMTHI	Parallel Move To HI
	PMTLO	Parallel Move To LO
	PMULTH	Parallel Multiply Halfword
	PMADDH	Parallel Multiply/Add Halfword
	PMSUBH	Parallel Multiply/Subtract Halfword
	PMFHL	Parallel Move From HI/LO
	PMTHL	Parallel Move To HI/LO
	PHMADH	Parallel Horizontal Multiply/Add Halfword
	PHMSBH	Parallel Horizontal Multiply/Subtract Halfword
	PDIVBW	Parallel Divide Broadcast Word
SA Operation	MFSA	Move From SA Register
	MTSA	Move To SA Register
	MTSAB	Move Byte Count to SA Register
	MTSAH	Move Halfword Count to SA Register
Shift	PSLLH	Parallel Shift Left Logical Halfword
	PSRLH	Parallel Shift Right Logical Halfword
	PSRAH	Parallel Shift Right Arithmetic Halfword
	PSLLW	Parallel Shift Left Logical Word
	PSLLVW	Parallel Shift Left Logical Variable Word
	PSRLW	Parallel Shift Right Logical Word
	PSRLVW	Parallel Shift Right Logical Variable Word
	PSRAW	Parallel Shift Right Arithmetic Word
	PSRAVW	Parallel Shift Right Arithmetic Variable Word
Logical	PAND	Parallel AND
	POR	Parallel OR
	PXOR	Parallel XOR
	PNOR	Parallel NOR
Compare	PCGTB	Parallel Compare for Greater Than Byte
	PCEQB	Parallel Compare for Equal Byte
	PCGTH	Parallel Compare for Greater Than Halfword
	PCEQH	Parallel Compare for Equal Halfword
	PCGTW	Parallel Compare for Greater Than Word
	PCEQW	Parallel Compare for Equal Word
	PLZCW	Parallel Leading Zero Count Word
128-bit Load Store	LQ	Load Quadword
	SQ	Store Quadword
Pack	PPACB	Parallel Pack To Byte
	PPACH	Parallel Pack To Halfword
	PINTEH	Parallel Interleave Even Halfword
	PPACW	Parallel Pack To Word
	PEXTUB	Parallel Extend Upper From Byte
	PEXTLB	Parallel Extend Lower From Byte
	PEXTUH	Parallel Extend Upper From Halfword
	PEXTLH	Parallel Extend Lower From Halfword
	PEXTUW	Parallel Extend Upper From Word
	PEXTLW	Parallel Extend Lower From Word
	PEXT5	Parallel Extend from 5 bits

Category	Mnemonic	Description
	PPAC5	Parallel Pack to 5 bits
Others	PCPYH	Parallel Copy Halfword
	PCPYLD	Parallel Copy Lower Doubleword
	PCPYUD	Parallel Copy Upper Doubleword
	PREVH	Parallel Reverse Halfword
	PINTH	Parallel Interleave Halfword
	PEXEH	Parallel Exchange Even Halfword
	PEXCH	Parallel Exchange Center Halfword
	PEXEW	Parallel Exchange Even Word
	PEXCW	Parallel Exchange Center Word
	PROT3W	Parallel Rotate 3 Word
	QFSRV	Quadword Funnel Shift Right Variable

Table 2-12 Multimedia Instructions

3. Caches

The EE Core contains both an instruction cache and a separate data cache. The processor also contains an embedded scratchpad RAM for fast manipulation of large data structures.
This chapter describes the cache structures, operations and control.

3.1. Cache and SPRAM Features

The two caches of the EE Core are configured as shown in Table 3-1:

Cache	Size	Organization	Line Size	Refill Size
Instruction Cache	16 KB	2-Way	64 bytes	64 bytes
Data Cache	8 KB	2-Way	64 bytes	64 bytes

Table 3-1 Cache Configuration

The following are the main features of the caches:

- Separate Instruction Cache and Data Cache
- Virtually indexed and physically tagged caches
- 64-byte line size
- 64-byte refill size
- 2-way set-associative cache
- Write-back policy (Data Cache)
- Missed quadword first sequential order refills
- Line Locking (Data Cache)
- Non-Blocking Loads
- Supports multiple hits under a single miss (Data Cache)

The following are the features of the Scratchpad RAM (SPRAM):

- 16 KB static RAM organized as 1K x 128 bits
- External DMA read and write capability
- Accessible to software through load/store instructions

No cache snoop capability has been provided. The user can choose to use CACHE instructions to keep coherency between caches and main memory.

3.2. Organization of the Caches

Organization of the Instruction and Data Caches is described below. Both are 2-way set-associative, composed of two sets of tags and 64-byte data.

3.2.1. Organization of the Data Cache

The Data Cache is connected to the CPU via a 128-bit bus. Therefore, the Data Cache can supply to the CPU or the coprocessors up to a quadword of data per access.
Organization of the Data Cache is illustrated in Figure 3-1. Tags are discussed in detail in a later section.

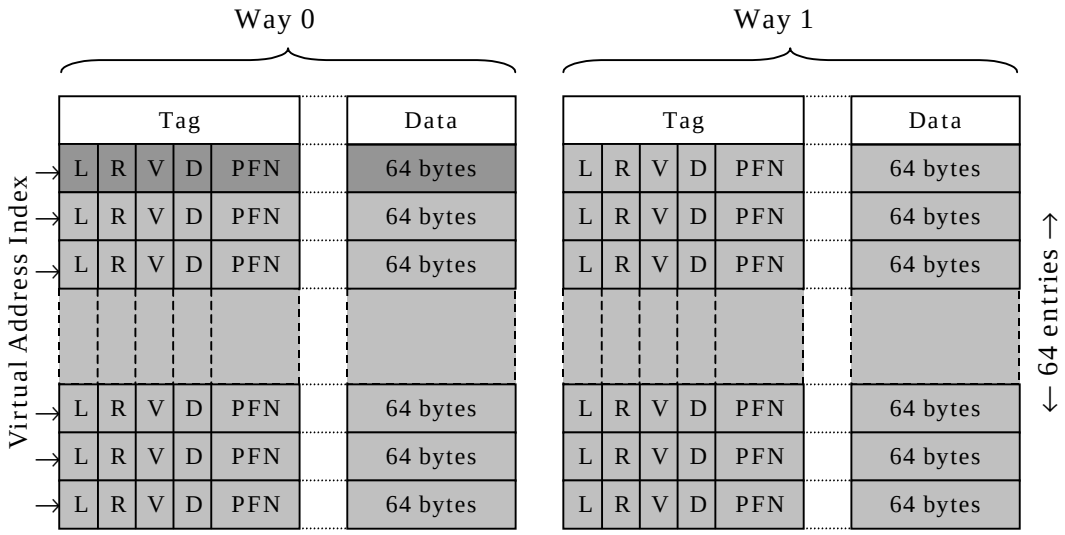


Figure 3-1 Organization of Data Cache

3.2.2. Organization of the Instruction Cache

The Instruction Cache is connected to the CPU pipeline via a 64-bit bus. This enables the CPU to fetch two instructions per cycle. Organization of the Instruction Cache is shown in Figure 3-2. Tags are discussed in detail in a later section.

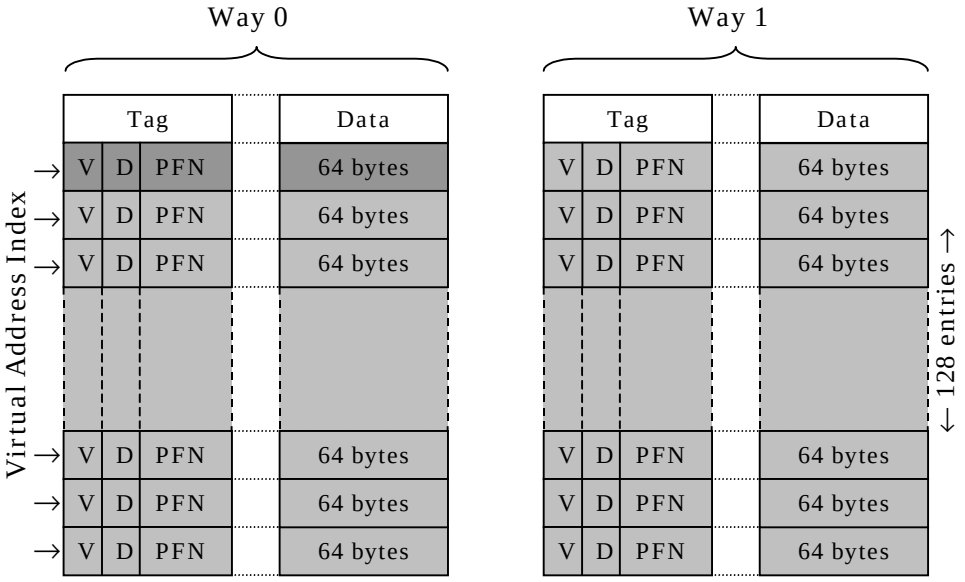


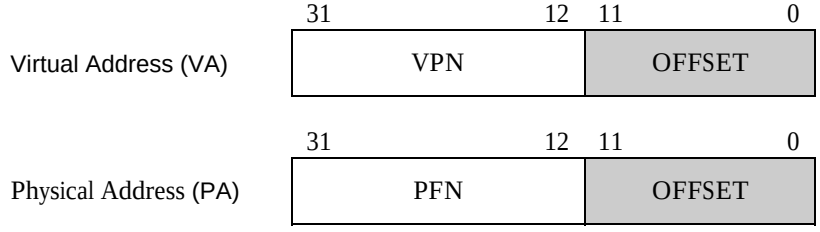
Figure 3-2 Organization of Instruction Cache

3.2.3. Tag Structure

The cache tag consists of a set of state bits and a physical page frame number or PFN field. The data and instruction caches have different numbers of state bits.

The size of the tag

The size of the tag and the number of virtual address bits are dependent on the size of the cache, address space and the number of the way. The EE Core supports 32-bit virtual and physical addresses as shown in the figure below:



Since the cache line size is fixed at 64 bytes, the tag cache has one tag for every 4 qwords. Table 3-2 shows cache sizes and the number of address bits.

Cache	Size	Way Organization	Virtual Address Index
Data	8 KB	64 x 64 bytes x 2-way	bit 11:6
Instruction	16 KB	128 x 64 bytes x 2-way	bit 12:6

Table 3-2 Cache Size and Address Bits

While the caches are indexed by the virtual address, the tag comparison is executed in a physical address. This is possible because the caches and the TLB translation look-aside buffer are accessed in parallel. When the tags have been accessed, the translated physical address is compared with the frame number and a cache hit or miss is determined.

Data Cache Tag Structure

Each Data Cache tag entry, as shown below, contains four state bits in addition to the physical page frame number (PFN).

Dirty (D)	Valid (V)	LRF (R)	Lock (L)	PFN

Figure 3-3 Data Cache Tag Fields

The Dirty bit and the Valid bit together identify the three states of the Data Cache Line (Valid Clean, Valid Dirty or Invalid).

Dirty (D)	Valid (V)	Cache Line State
X	0	Invalid
0	1	Valid Clean
1	1	Valid Dirty

Table 3-3 Data Cache Line States

The LRF bits indicate the Least-Recently-Filled line and control a replacement between the two ways. A refill access to a cache line in a way will flip the LRF bit to point to the other way as the least recently filled. For details of the LRF line update operation, refer to Section "3.3.1. Line Replacement Algorithm".
The lock bit is a flag which locks lines to keep data from being replaced. Refer to "3.3.7. Data Cache Lock Function" for more details.

Instruction Cache Tag Structure

Each Instruction Cache tag entry, as shown below, contains two state bits in addition to the physical page frame number (PFN).

Valid (V)	LRF (R)	PFN

Figure 3-4 Instruction Cache Tag Fields

The Valid bit indicates whether each line is in the Valid or Invalid states. The LRF bits, like those of the Data Cache tag, indicate the Least-Recently-Filled line and control a replacement. Refer to Section "3.3.1. Line Replacement Algorithm" for more information.

Initial Value of Cache Tags

State bits of all Data and Instruction Cache tags are initialized to 0 and the values of PFN fields are undefined upon reset.

3.3. Cache Operations

This section describes cache operations in regard to read/write policies, coherency, write-back and the lock function.

3.3.1. Line Replacement Algorithm

Based on the LRF algorithm for line replacement of the Instruction Cache and Data Cache, one of the 2 ways that was least recently refilled is replaced. For example, when a read from memory to each line occurs, the LRF bit is flipped. (Load and store accesses to the Data Cache do not modify the LRF bit.) XOR of the LRF bits indicate which way is the least recently filled and that result determines which way could be refilled. Refer to the following table.

Way 0 LRF	Way 1 LRF	XOR	Refill Way	New Way 0 LRF	New Way 1 LRF
0	0	0	0	1	0
1	0	1	1	1	1
1	1	0	0	0	1
0	1	1	1	0	0

Table 3-4 LRF Line Replacement Algorithm

If the cache line is locked, regardless of the state of the LRF bits, the data is refilled into the unlocked way. And when one of the ways is Invalid in the Instruction Cache, regardless of the state of the LRF bits, the data is refilled into the Invalidated way.

3.3.2. Non-blocking Loads and Hit Under Miss

The Data Cache supports non-blocking load and hit under miss to improve performance. Support for a non-blocking load allows the pipeline to continue instruction execution until one of the following occurs even when load instructions are pending due to a cache miss:

- An instruction which has data dependency with a load instruction that is pending due to a cache miss is issued
- A second miss occurs in the Data Cache
- An Uncached or Uncached Accelerated load instruction is issued.

Loads to GPRs and FPRs (COP1 data register) are non-blocking and loads to COP2 are always blocking. Support for hit under miss enables access to the Data Cache and continued execution of instructions in the pipeline, even when loads are pending due to a load miss (cached or uncached), a store miss (cached) or a prefetch miss (cached).

Uncached and uncached accelerated loads also do not stall the pipeline while they are pending due to misses. The pipeline continues instruction execution until one of the following occurs:

- An instruction which has data dependency with the load instruction that is pending is issued
- A Data Cache miss occurs
- A second Uncached or Uncached Accelerated load instruction is issued.

3.3.3. Cache Hit and Miss Operations

With a Data Cache hit, the cache sends data to the CPU in 128-bit (1-qword) units. In case of an Instruction Cache hit, the cache sends data (instruction code) in 64-bit units. To read or write data less than 128 bits is specified by the least significant 4 bits (bits 3:0) of the address.
With cache misses, cache refill is performed in one cache line (64-byte = 4-qword). Since the caches are connected to the system bus via a 128-bit bus, cache refill takes a burst of 4 bus cycles (8 CPU cycles). (Actual transfer time can be more due to bus arbitration, etc).
With a cache refill, both the Instruction and Data Cache always fetch first a missed quadword of a burst of four quadwords. The sequence in which four quadwords are read depending on the least significant 2 bits of the missed quadword is shown in the table below. Figure 3-5 illustrates the sequence in which the Data Cache is read from the memory when the second quadword misses.

Missed Address (PA[5:4])	Read Order (PA[5:4])
00	00 → 01 → 10 → 11
01	01 → 10 → 11 → 00
10	10 → 11 → 00 → 01
11	11 → 00 → 01 → 10

Table 3-5 Reread order in case of Cache Miss

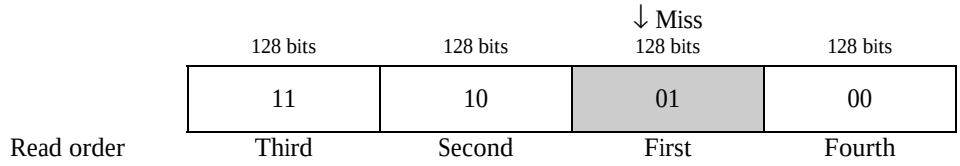


Figure 3-5 Reread Processing in case of Cache Miss

With a write miss to the Data Cache, the data is read from main memory in sequential order.
With cache misses in the Instruction Cache, just like with the Data Cache, a reread is performed in 4 quadwords and the pipeline starts in the same cycle the final qword is stored into the Instruction Cache.

3.3.4. Data Cache Writeback

Data cache lines are written back to memory before the missed data are read when the line data are dirty and a read or write miss occurs.

3.3.5. Data Cache State Transitions

As discussed previously, lines in the Data Cache can be in one of several states: Invalid, Valid Clean or Valid Dirty.
Invalid means the Data Cache line does not contain valid data. When a miss occurs, the data can be read in to the line immediately.
Valid Clean indicates that there are valid data in the Data Cache line and they are the same data as in memory.
Valid Dirty indicates that there are valid data in the Data Cache line and they are not the same data as in memory. That is, the data written into the cache have not been reflected yet in memory. The line has to be written back before reading the data.
The transition of the Data Cache is shown in Figure 3-6.

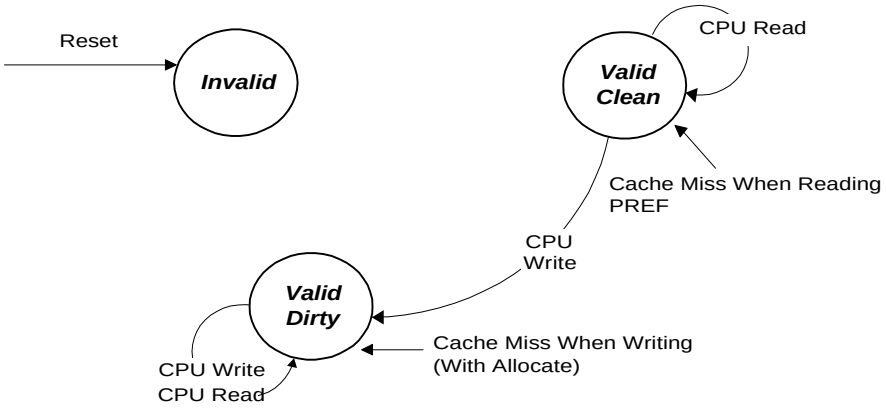


Figure 3-6 Data Cache Transition Diagram

3.3.6. Instruction Cache State Transitions

Cache lines in the Instruction Cache can be in either of two states: Invalid or Valid.
Invalid means the Instruction Cache line does not contain valid instructions. When a miss occurs, the line can read the instruction immediately.
Valid state indicates that there are valid instructions in the cache line.
The transition of the Data Cache is shown in Figure 3-7.

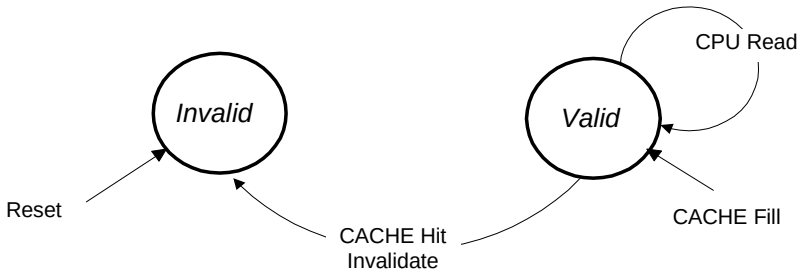


Figure 3-7 Instruction Cache Transition Diagram

3.3.7. Data Cache Lock Function

The contents of the Data Cache are replaced dynamically using the LRF algorithm. But a Data Cache lock function has been provided to retain important data in the cache.
Each line of the Data Cache has is a Lock bit. When it is set, the line is locked and the LRF bit is no longer meaningful. If one of the ways for a particular line is locked, the other way is the only way available for cache miss processing.
A write access to a locked line is performed only to the cache, not to memory. And the Dirty bit is not set then.
To lock the Data Cache, the following two instructions can be used:

CACHE DXSTG(DCACHE Index Store Tag)
CACHE DXSDT(DCACHE Index Store Data)

The following restrictions apply to line locking:

- The result of re-locking a locked line is undefined
- The results of locking both ways of a cache line are undefined

3.3.8. Uncached Load/Store

Store instructions with the uncached or uncached accelerated attributes bypass the Data Cache completely. Uncached and Uncached Accelerated load and store operations are always executed in order on the CPU bus. On the other hand, cached load operations can precede earlier cached store operations on the CPU bus. In order to avoid this, wait until the stored data has been sent to the Data Cache, SPRAM or CPU bus, using the SYNC or SYNC.L instructions.

3.3.9. Data Consistency

The capability to retain data consistency between address spaces, mapped into the instruction cache, data cache, and SPRAM is not provided.

3.4. Scratchpad RAM (SPRAM)

Certain applications require high-speed on-chip RAM that can be accessed by normal load and store instructions to handle data structures efficiently. To achieve this capability, a Scratchpad RAM (SPRAM) of 16 KB is provided, in addition to the locked Data Cache capability.

SPRAM operation is very similar to Data Cache operation. Both use the same access paths, which means the CPU can access only either SPRAM or the Data Cache in any given CPU cycle.

SPRAM can be accessed by the DMA controller as well as the CPU; the DMA controller has access priority.

SPRAM space pages are 16 KB in size. The least significant 14 bits of the virtual address indicate addresses in SPRAM. The upper 18 bits of the virtual address are used to access the TLB to determine if that particular 16 KB block is mapped into SPRAM or not. To differentiate between the memory spaces, that is, between the Data Cache and SPRAM, a bit (the S bit) in the TLB entry is used.

The SPRAM structure is configured as 1024 x 128 bits, and there are no tags associated with it. To read or write data to SPRAM, special DMA protocols are provided. Once the data reside in the SPRAM, load / store instructions can be used to access data.

3.4.1. SPRAM DMA Protocol

The DMA transfer between SPRAM and memory is performed as shown below.

For DMA writes to SPRAM, a special SPRAM write signal is provided to the CPU along with a 14-bit SPRAM address. Data is placed on the CPU Bus. The CPU samples the data from the CPU Bus and writes it into the indexed address in the SPRAM.

For DMA reads from SPRAM, the external DMA controller reads the contents of the SPRAM into memory. The CPU Bus address is used to index the SPRAM and the corresponding data is placed on the CPU Bus. Thus, by reading / writing with DMA, the CPU can execute programs concurrently, which can result in higher performance.

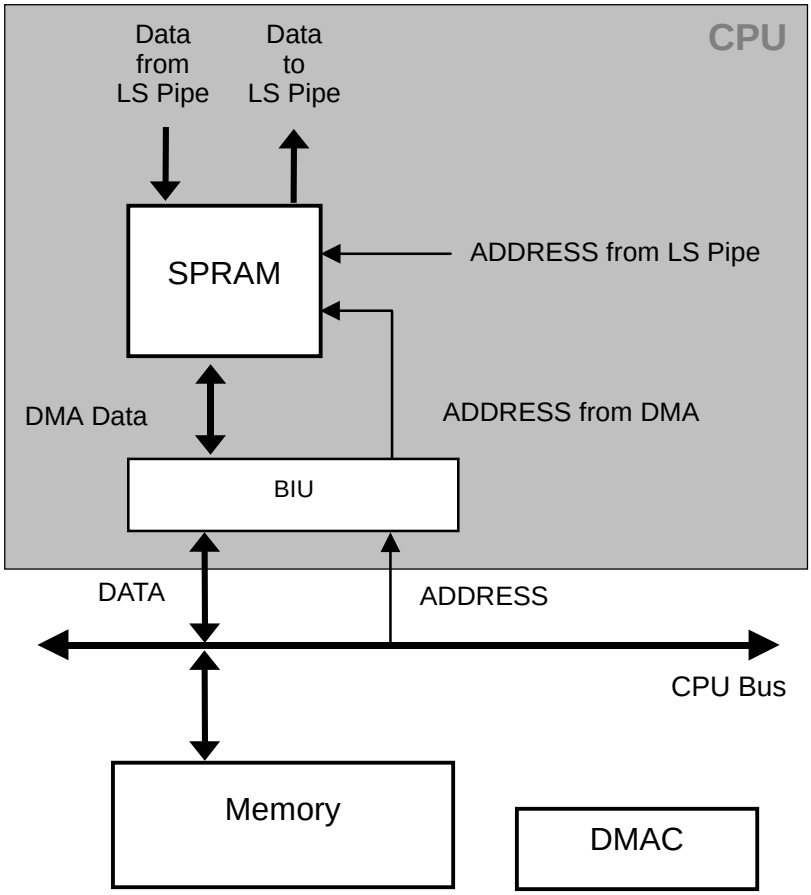


Figure 3-8 SPRAM Data Paths

Figure 3-8 illustrates how the SPRAM is embedded in the CPU and accessed by the DMA. Simultaneous access to the SPRAM by the CPU and the DMA controller results in alternate cycle accesses, with the DMA controller having the highest priority.

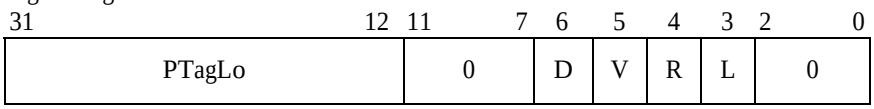
3.5. Cache Control Registers

The operations of the caches are controlled by bits in the Config register.

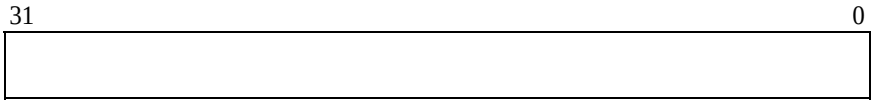
Bit	Description
ICE	Instruction Cache Enable
DCE	Data Cache Enable
IC	Instruction Cache Size
DC	Data Cache Size
IB	Instruction Cache Line Size
DB	Data Cache Line Size

The two cache tag registers TagLo and TagHi are 32-bit read/write registers that hold the tag and state of the cache line.

TagLo Register



TagHi Register



Name	Contents
PTagLo	Physical address bits 31:12
D	Dirty bit (Not used for the Instruction Cache)
V	Valid bit
L	Lock bit (Not used for the Instruction Cache)
R	LRF bit

The contents of the TagHi register have a variety of meanings depending on instructions.
These Tag registers are manipulated by MTC0 and CACHE instructions.

(This page is left blank intentionally)

4. Floating-Point Unit (FPU)

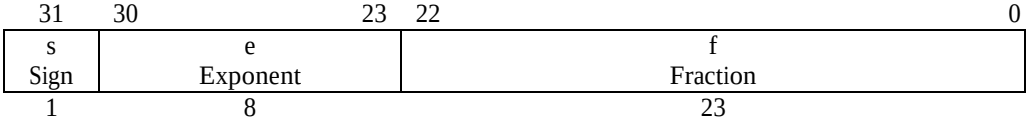
This chapter describes the floating-point unit (FPU=COP1) of the EE Core. The following is an overview of the FPU's features :

- High performance single-precision floating-point unit tightly coupled to the EE Core
- Supports single-precision format, as defined in the IEEE 754 specification
- No support for NaNs and plus/minus infinite is provided. Plus/minus "0" support is provided
- No hardware exception mechanism to affect the instruction stream
- Compatible with coprocessor 2 (VPU)

4.1. Data Formats

4.1.1. Floating-Point Formats

The FPU only supports 32-bit single-precision for floating-point numbers. The numeric value format is based on the IEEE754 standard, and has a 24-bit signed fraction and an 8-bit exponent.



Specification	
Size (Width)	32 bits
Exponent	8 bits
Integer bit	hidden
Fraction	24 bits
Emax (Maximum Exponent)	+128
Emin (Minimum Exponent)	-126
Biased Exponent	+127

Figure 4-1 Single-Precision Floating-Point Format

The true exponent E is calculated by subtracting the biased value from the exponent e. The range of E can be every integer value between Emin and Emax. The value of a single-precision floating-point number is shown below.

IF $0 < e <= 255$, then $(-1)^s \bullet 2^{e-127}(1.f)$

IF $e = 0$, then $(-1)^s \bullet 0$

Note that FPU does not support denormalized numbers, plus and minus infinities, and NaN(Not a Number) as defined in IEEE754.

4.1.2. Fixed-Point Format

Fixed-point values are in 2's complement format, shown in Figure 4-2, and unsigned fixed-point values are not directly supported.

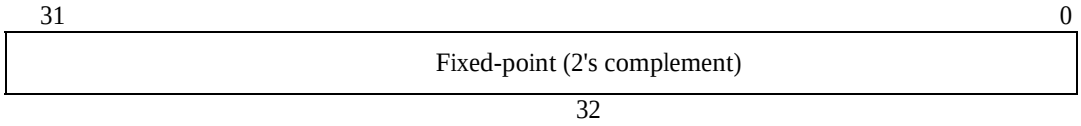


Figure 4-2 Fixed-Point Format

4.2. FPU Registers

The FPU has 32 general-purpose registers (FPRs), each of which is 32 bits wide. The CPU can access these registers through move (MFC1 and MTC1), load (LWC1) and store (SWC1) instructions.
There are 32 floating-point control registers (FCR), of which only two (FCR0 and FCR31) are implemented. Details of these are described later.
In addition, a 32-bit accumulator is used in the multiply-accumulate type instructions.

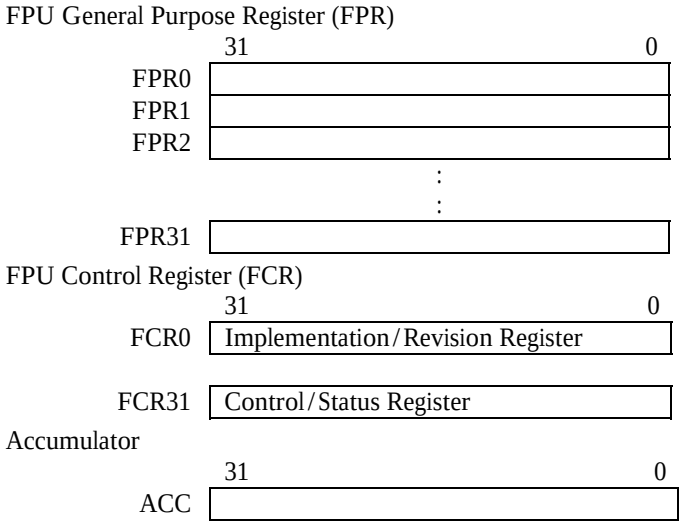


Figure 4-3 FPU Registers

4.3. FPU Control Registers

The FPU has 32 control registers (FCRs), which can only be accessed by move instructions (CFC1 and CTC1). However, FCR1 to FCR30 are reserved registers, and only FCR0 (Implementation/Revision register) and FCR31(Control/Status register) are implemented.

4.3.1. Implementation and Revision Register (FCR0)

The Implementation and Revision register (FCR0) is read-only and contains the implementation and revision number of the FPU. This information can determine the FPU capability and performance level, and can be used for diagnostic software.

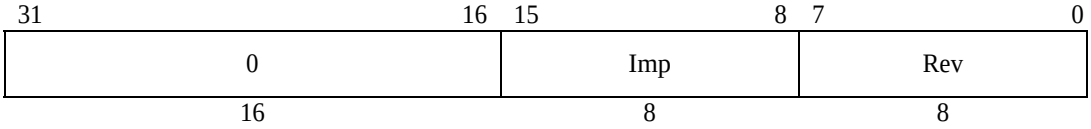


Figure 4-4 Implementation/Revision Register

Field	Bits	Description	r/w	Initial Value
0	31:16	Fixed as zero (reserved)	r /-	0
Imp	15:8	Implementation number	r /-	0x2E
Rev	7:0	Revision number bits 7:4 and bits 3:0 indicate a major and minor revision respectively.	r /-	0x00

Table 4-1 Implementation/Revision Register Fields

4.3.2. Control/Status Register (FCR31)

The Control/Status register (FCR31) contains FPU status information, such as results of arithmetic operations.

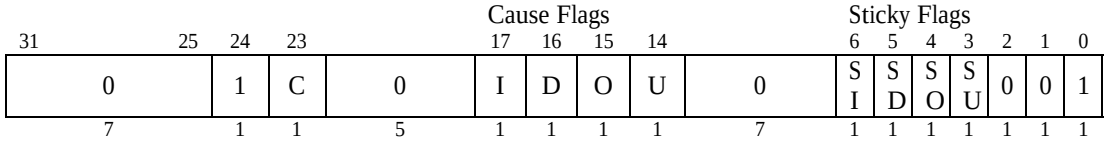


Figure 4-5 Control/Status Register

Four flags—I, D, O, and U—are collectively referred to as the Cause flags. Likewise, four flags—SI, SD, SO, and SU—are collectively referred to as the Sticky flags (Accumulation flags).

Cause flags indicate the result of the immediately prior arithmetic instruction. Sticky flags are set to 1 when the corresponding Cause flags become 1, and are never set to 0 unless software explicitly clears them, that is, part of the program indicates the result of arithmetic instructions.

Field	Bits	Description	r/w	Initial Value
C	23	Condition bit The bit is set to 1 when the result of a floating-point Compare operation is true and the bit is cleared to 0 when the result is false.	r/w	Undefined
I	17	Invalid Operation flag The bit is set to 1 when attempting to execute 0/0 division, square root of a negative number or reciprocal square root of a negative number. Otherwise, the bit is cleared to 0.	r/w	Undefined
D	16	Division by Zero flag The bit is set to 1 when attempting to execute division by zero. Otherwise, the bit is cleared to 0.	r/w	Undefined
O	15	Overflow flag The bit is set to 1 when the exponent of the computational result overflows. Otherwise, the bit is cleared to 0.	r/w	Undefined
U	14	Underflow flag The bit is set to 1 when the exponent of the computational result underflows. Otherwise, the bit is cleared to 0	r/w	Undefined
SI	6	Invalid Operation cumulative flag The bit is set to 1 when attempting to execute 0/0 division, square root of the negative number or reciprocal square root of the negative number.	r/w	Undefined
SD	5	Division by Zero cumulative flag The bit is set to 1 when attempting to execute division by zero.	r/w	Undefined
SO	4	Overflow cumulative flag The bit is set to 1 when the exponent of the computational result overflows.	r/w	Undefined
SU	3	Underflow cumulative flag The bit is set to 1 when the exponent of the computational result underflows.	r/w	Undefined

Table 4-2 Control/Status Register Fields

4.4. Instruction Set Overview

All FPU instructions are 32 bits long and aligned on a word boundary.
FPU instructions can be divided into the following categories:

- Move instructions: Instructions which move data between memory and the main processor and between FPU general-purpose registers and FPU control registers.
- Conversion instructions: Instructions which convert data formats of numeric data
- Computational instructions: Instructions which perform arithmetic operations on floating-point data in the FPU general-purpose register
- Compare instructions: Instructions which compare the contents of FPU general-purpose registers and reflect the result in the C bit of the status register
- Branch on FPU condition instructions: Instructions which branch according to the C bit of the status register

The list of FPU instructions is shown below. For details of each instruction, refer to the "EE Core Instruction Set Manual".

Category	Instruct.	Description
Move Instruction	LWC1	Load Word to FPR
	SWC1	Store Word from FPR
	MTC1	Move Word To FCR
	MFC1	Move Word From FCR
	MOV.S	Single Floating-point Move
	CTC1	Move Control Word to FCR
	CFC1	Move Control Word from FCR
Conversion Instruction	CVT.S.W	32-bit Fixed Point Convert to Single Floating-point
	CVT.W.S	Single Floating-point Convert to 32-bit Fixed Point
Computational Instruction	ADD.S	Single Floating-point Add
	SUB.S	Single Floating-point Subtract
	MUL.S	Single Floating-point Multiply
	DIV.S	Single Floating-point Divide
	ABS.S	Single Floating-point Absolute
	NEG.S	Single Floating-point Negate
	SQRT.S	Single Floating-point Square Root
	ADDA.S	Single Floating-point Add to Accumulator
	SUBA.S	Single Floating-point Subtract to Accumulator
	MULA.S	Single Floating-point Multiply to Accumulator
	MADD.S	Single Floating-point Multiply and Add
	MADDA.S	Single Floating-point Multiply and Add to Accumulator
	MSUB.S	Single Floating-point Multiply and Subtract
	MSUBA.S	Single Floating-point Multiply/Subtract from Accumulator
	RSQRT.S	Single Floating-point Reciprocal Square Root
Computational Instruction	MAX.S	Single Floating-point Maximum
	MIN.S	Single Floating-point Minimum
Comparison	C.cond.S	Single Floating-point Compare
Branch on FPU Condition	BC1T	Branch on FPU True
	BC1F	Branch on FPU False
	BC1TL	Branch on FPU True Likely
	BC1FL	Branch on FPU False Likely

Table 4-3 FPU Instructions

4.5. Results of Abnormal Computation

If abnormal computations such as "Divide by Zero" are performed or an overflow or underflow occurs, the resulting values and flags set in the status register are shown in the table below.

Computational Exception	Result Value	Cause Flag
Divide by Zero	+Fmax or –Fmax	D=1
0 / 0	+Fmax or –Fmax	I=1
Square root of a negative number	Square root of the absolute value of the parameter	I=1
Exponent Overflow	+Fmax or –Fmax	O=1
Exponent Underflow	+0 or –0	U=1
Conversion Overflow	+Fmax or –Fmax	None

*Fmax is the maximum number in a single-precision floating-point format.

Table 4-4 Results of Abnormal Computation

4.6. Sign of Zero

In a single-precision floating-point format, two zeros, +0 and –0 are present. The results when using signed 0 as both of the operands are shown in the table below. This is compatible with the IEEE 754 specification.

Operation	Result	I/SI flag	D/SD flag
(+0)+(+0)	+0	-/-	-/-
(+0)+(-0)	+0	-/-	-/-
(-0)+(+0)	+0	-/-	-/-
(-0)+(-0)	-0	-/-	-/-
(+0)-(+0)	+0	-/-	-/-
(+0)-(-0)	+0	-/-	-/-
(-0)-(+0)	-0	-/-	-/-
(-0)-(-0)	+0	-/-	-/-
(+0)×(+0)	+0	-/-	-/-
(+0)×(-0)	-0	-/-	-/-
(-0)×(+0)	-0	-/-	-/-
(-0)×(-0)	+0	-/-	-/-
(+0)/(+0)	7FFFFFFF	1/1	0/0
(+0)/(-0)	FFFFFFFF	1/1	0/0
(-0)/(+0)	FFFFFFFF	1/1	0/0
(-0)/(-0)	7FFFFFFF	1/1	0/0
Max(+0,+0)	+0	-/-	-/-
Max(+0,-0)	+0	-/-	-/-
Max(-0,+0)	+0	-/-	-/-
Max(-0,-0)	-0	-/-	-/-
Min(+0,+0)	+0	-/-	-/-
Min(+0,-0)	-0	-/-	-/-
Min(-0,+0)	-0	-/-	-/-
Min(-0,-0)	-0	-/-	-/-
(+0)/SQRT(+0)	7FFFFFFF	1/1	0/0
(+0)/SQRT(-0)	FFFFFFFF	1/1	0/0
(-0)/SQRT(+0)	FFFFFFFF	1/1	0/0
(-0)/SQRT(-0)	7FFFFFFF	1/1	0/0
(+fs)/(+0)	7FFFFFFF	0/0	1/1
(+fs)/(-0)	FFFFFFFF	0/0	1/1
(-fs)/(+0)	FFFFFFFF	0/0	1/1
(-fs)/(-0)	7FFFFFFF	0/0	1/1

Table 4-5 Computation of Signed Zero

4.7. Rounding

The FPU only supports "Rounding towards 0". "Rounding towards Nearest" and "Rounding towards +/- infinities" as defined by IEEE 754 are not supported.
Since "Rounding towards Nearest" is not supported, the FPU does not use the Guard, Round and Sticky bits during rounding. Also, since "Rounding toward 0" does not require full value prior to rounding unlike the definition of IEEE 754, the results may differ from the IEEE 754 Rounding to 0. This difference is usually restricted to the least significant bit only.
Since the rounding mode is not programmable in this FPU, the two least significant bits of the Control and Status registers (FCR31) are hardwired to "01".

4.8. IEEE 754 Compatibility

The FPU is not compatible with the IEEE 754 Floating-Point standard. Only single-precision operations are supported. Overflow and Underflow are detected only for overflow or underflow of the exponent. While the IEEE standard recommends trapping a computational exception, in the FPU, processing continues by just setting a flag. Since these flags can be sampled on an instruction by instruction basis, emulating this trap is possible if necessary.
In addition, the following shows the differences from the IEEE 754 specification. When computational results are different, refer to the table below.

- NaN (Not a Number), +∞, -∞ and denormalized number are not supported.
- The FPU does not use the Guard, Round and Sticky bits during computations. The computed result usually differs from IEEE 754 only in the least-significant bit. For saturating instructions, bits other than the least-significant bit can be different.
- Only "Rounding towards 0" is supported and "Rounding towards Nearest", "Rounding towards +/-∞" are not supported. The result of "Rounding towards 0" can differ from IEEE 754 in the least significant bit.
- IEEE 754-defined exceptions are not fully supported. In particular, Invalid Operation exceptions due to NaN, +/-∞ and Inexact exceptions are not supported.

Operation	IEEE 754	FPU
0/0	Result is a "NaN". Invalid Operation exception is taken.	Result is +Fmax or -Fmax. I bit (SI bit) is set.
Sqrt(x) (x<0)	Result is a "NaN". Invalid Operation exception is taken.	Result is Sqrt (|x|). I bit (SI bit) is set.
Division by zero	Result is +∞ or -∞ Division by Zero exception is taken	Result is +Fmax or -Fmax. D bit (SD bit) is set.
Exponent overflow	Result is +∞, -∞, +Fmax or -Fmax (determined by the rounding mode) Overflow exception is taken	Result is +Fmax or -Fmax. O bit (SO bit) is set.
Exponent underflow	Result is Denormalized value. Underflow exception is taken	Result is +0 or -0. U bit (SU bit) is set.
Conversion of Floating-point to Integer Overflow	Result is not defined. Invalid Operation exception is taken.	Result is 2^{31}-1 or -2^{31} I bit (SI bit) is set.

Table 4-6 Differences of the computational results between IEEE 754 and FPU