Overview of the CPU Instruction Set

This chapter gives an overview of the CPU instructions, including a description of CPU instruction formats. An overview of the FPU instructions is given in Chapter 5.

4.1 CPU Instructions, Grouped By Function

CPU instructions are organized into the following functional groups:

- Load and store
- Computational
- Jump and branch
- Miscellaneous
- Coprocessor

Each instruction is 32 bits long.

4.1.1 CPU Load and Store Instructions

MIPS processors use a load/store architecture; all operations are performed on operands held in processor registers and main memory is accessed only through load and store instructions.

4.1.1.1 Types of Loads and Stores

There are several different types of load and store instructions, each designed for a different purpose:

- Transferring variously-sized fields (for example, LB, SW)
- Trading transferred data as signed or unsigned integers (for example, LHU)
- Accessing unaligned fields (for example, LWR, SWL)
- Selecting the addressing mode (for example, SDXC1, in the FPU)
- Atomic memory update (read-modify-write: for instance, LL/SC)

Regardless of the byte ordering (big- or little-endian), the address of a halfword, word, or doubleword is the lowest byte address among the bytes forming the object:

- For big-endian ordering, this is the most-significant byte.
- For a little-endian ordering, this is the least-significant byte.

Refer to "Byte Ordering and Endianness" on page 21 for more information on big-endian and little-endian data ordering.

4.1.1.2 Load and Store Access Types

Table 4-1 lists the data sizes that can be accessed through CPU load and store operations. These tables also indicate the particular ISA within which each operation is defined.

Table 4-1 Load and Store Operations Using Register + Offset Addressing Mode

Data Size	CPU			Coprocessors 1 and 2	
	Load Signed	Load Unsigned	Store	Load	Store
Byte	MIPS32	MIPS32	MIPS32		
Halfword	MIPS32	MIPS32	MIPS32		
Word	MIPS32	MIPS64	MIPS32	MIPS32	MIPS32
Doubleword (FPU)				MIPS32	MIPS32
Unaligned word	MIPS32		MIPS32		
Linked word (atomic modify)	MIPS32		MIPS32		

4.1.1.3 List of CPU Load and Store Instructions

The following data sizes (as defined in the *AccessLength* field) are transferred by CPU load and store instructions:

- Byte
- Halfword
- Word

Signed and unsigned integers of different sizes are supported by loads that either sign-extend or zero-extend the data loaded into the register.

Table 4-2 lists aligned CPU load and store instructions, while unaligned loads and stores are listed in Table 4-3. Each table also lists the MIPS ISA within which an instruction is defined.

Table 4-2 Aligned CPU Load/Store Instructions

Mnemonic	Instruction	Defined in MIPS ISA
LB	Load Byte	MIPS32
LBU	Load Byte Unsigned	MIPS32
LH	Load Halfword	MIPS32
LHU	Load Halfword Unsigned	MIPS32
LW	Load Word	MIPS32
SB	Store Byte	MIPS32
SH	Store Halfword	MIPS32
SW	Store Word	MIPS32

Unaligned words and doublewords can be loaded or stored in just two instructions by using a pair of the special instructions listed in Table 4-3. The load instructions read the left-side or right-side bytes (left or right side of register) from an aligned word and merge them into the correct bytes of the destination register.

Unaligned CPU load and store instructions are listed in Table 4-3, along with the MIPS ISA within which an instruction is defined.

Table 4-3 Unaligned CPU Load and Store Instructions

Mnemonic	Instruction	Defined in MIPS ISA
LWL	Load Word Left	MIPS32
LWR	Load Word Right	MIPS32
SWL	Store Word Left	MIPS32
SWR	Store Word Right	MIPS32

4.1.1.4 Loads and Stores Used for Atomic Updates

The paired instructions, Load Linked and Store Conditional, can be used to perform an atomic read-modify-write of word or doubleword cached memory locations. These instructions are used in carefully coded sequences to provide one of several synchronization primitives, including test-and-set, bit-level locks, semaphores, and sequencers and event counts. Table 4-4 lists the LL and SC instructions, along with the MIPS ISA within which an instruction is defined.

Table 4-4 Atomic Update CPU Load and Store Instructions

Mnemonic	Instruction	Defined in MIPS ISA
LL	Load Linked Word	MIPS32
SC	Store Conditional Word	MIPS32

4.1.1.5 Coprocessor Loads and Stores

If a particular coprocessor is not enabled, loads and stores to that processor cannot execute and the attempted load or store causes a Coprocessor Unusable exception. Enabling a coprocessor is a privileged operation provided by the System Control Coprocessor, CP0.

Table 4-5 lists the coprocessor load and store instructions.

Table 4-5 Coprocessor Load and Store Instructions

Mnemonic	Instruction	Defined in MIPS ISA
LDCz	Load Doubleword to Coprocessor-z, z = 1 or 2	MIPS32
LWCz	Load Word to Coprocessor-z, z = 1 or 2	MIPS32
SDCz	Store Doubleword from Coprocessor-z, z = 1 or 2	MIPS32
SWCz	Store Word from Coprocessor-z, z = 1 or 2	MIPS32

Table 4-6 lists the specific FPU load and store instructions;[1] it also lists the MIPS ISA within which an instruction was first defined.

Table 4-6 FPU Load and Store Instructions Using Register + Register Addressing

Mnemonic	Instruction	Defined in MIPS ISA
LWXC1	Load Word Indexed to Floating Point	MIPS64 MIPS32 Release 2
SWXC1	Store Word Indexed from Floating Point	MIPS64 MIPS32 Release 2
LDXC1	Load Doubleword Indexed to Floating Point	MIPS64 MIPS32 Release 2
SDXC1	Store Doubleword Indexed from Floating Point	MIPS64 MIPS32 Release 2
LUXC1	Load Doubleword Indexed Unaligned to Floating Point	MIPS64 MIPS32 Release 2
SUXC1	Store Doubleword Indexed Unaligned from Floating Point	MIPS64 MIPS32 Release 2

4.1.2 Computational Instructions

This section describes the following:

- "ALU Immediate and Three-Operand Instructions"
- "ALU Two-Operand Instructions"
- "Shift Instructions"
- "Multiply and Divide Instructions"

2's complement arithmetic is performed on integers represented in 2's complement notation. These are signed versions of the following operations:

- Add
- Subtract
- Multiply
- Divide

The add and subtract operations labelled "unsigned" are actually modulo arithmetic without overflow detection.

There are also unsigned versions of *multiply* and *divide*, as well as a full complement of *shift* and *logical* operations. Logical operations are not sensitive to the width of the register.

MIPS32 provided 32-bit integers and 32-bit arithmetic.

[1] FPU loads and stores are listed here with the other coprocessor loads and stores for convenience.

4.1.2.1 ALU Immediate and Three-Operand Instructions

Table 4-7 lists those arithmetic and logical instructions that operate on one operand from a register and the other from a 16-bit *immediate* value supplied by the instruction word. This table also lists the MIPS ISA within which an instruction is defined.

The *immediate* operand is treated as a signed value for the arithmetic and compare instructions, and treated as a logical value (zero-extended to register length) for the logical instructions.

Table 4-7 ALU Instructions With an Immediate Operand

Mnemonic	Instruction	Defined in MIPS ISA
ADDI	Add Immediate Word	MIPS32
ADDIU[1]	Add Immediate Unsigned Word	MIPS32
ANDI	And Immediate	MIPS32
LUI	Load Upper Immediate	MIPS32
ORI	Or Immediate	MIPS32
SLTI	Set on Less Than Immediate	MIPS32
SLTIU	Set on Less Than Immediate Unsigned	MIPS32
XORI	Exclusive Or Immediate	MIPS32

1. The term “unsigned” in the instruction name is a misnomer; this operation is 32-bit modulo arithmetic that does not trap on overflow.

Table 4-8 describes ALU instructions that use three operands, along with the MIPS ISA within which an instruction is defined.

Table 4-8 Three-Operand ALU Instructions

Mnemonic	Instruction	Defined in MIPS ISA
ADD	Add Word	MIPS32
ADDU[1]	Add Unsigned Word	MIPS32
AND	And	MIPS32
NOR	Nor	MIPS32
OR	Or	MIPS32
SLT	Set on Less Than	MIPS32
SLTU	Set on Less Than Unsigned	MIPS32
SUB	Subtract Word	MIPS32
SUBU[1]	Subtract Unsigned Word	MIPS32
XOR	Exclusive Or	MIPS32

1. The term “unsigned” in the instruction name is a misnomer; this operation is 32-bit modulo arithmetic that does not trap on overflow.

4.1.2.2 ALU Two-Operand Instructions

Table 4-8 describes ALU instructions that use two operands, along with the MIPS ISA within which an instruction is defined.

Table 4-9 Two-Operand ALU Instructions

Mnemonic	Instruction	Defined in MIPS ISA
CLO	Count Leading Ones in Word	MIPS32
CLZ	Count Leading Zeros in Word	MIPS32

4.1.2.3 Shift Instructions

The ISA defines two types of shift instructions:

- Those that take a fixed shift amount from a 5-bit field in the instruction word (for instance, SLL, SRL)
- Those that take a shift amount from the low-order bits of a general register (for instance, SRAV, SRLV)

Shift instructions are listed in Table 4-10, along with the MIPS ISA within which an instruction is defined.

Table 4-10 Shift Instructions

Mnemonic	Instruction	Defined in MIPS ISA
ROTR	Rotate Word Right	MIPS32 Release 2
ROTRV	Rotate Word Right Variable	MIPS32 Release 2
SLL	Shift Word Left Logical	MIPS32
SLLV	Shift Word Left Logical Variable	MIPS32
SRA	Shift Word Right Arithmetic	MIPS32
SRAV	Shift Word Right Arithmetic Variable	MIPS32
SRL	Shift Word Right Logical	MIPS32
SRLV	Shift Word Right Logical Variable	MIPS32

4.1.2.4 Multiply and Divide Instructions

The multiply and divide instructions produce twice as many result bits as is typical with other processors. With one exception, they deliver their results into the *HI* and *LO* special registers. The MUL instruction delivers the lower half of the result directly to a GPR.

- **Multiply** produces a full-width product twice the width of the input operands; the low half is loaded into *LO* and the high half is loaded into *HI*.
- **Multiply-Add** and **Multiply-Subtract** produce a full-width product twice the width of the input operations and adds or subtracts the product from the concatenated value of *HI* and *LO*. The low half of the addition is loaded into *LO* and the high half is loaded into *HI*.
- **Divide** produces a quotient that is loaded into *LO* and a remainder that is loaded into *HI*.

The results are accessed by instructions that transfer data between *HI*/*LO* and the general registers.

Table 4-11 lists the multiply, divide, and *HI*/*LO* move instructions, along with the MIPS ISA within which an instruction is defined.

Table 4-11 Multiply/Divide Instructions

Mnemonic	Instruction	Defined in MIPS ISA
DIV	Divide Word	MIPS32
DIVU	Divide Unsigned Word	MIPS32
MADD	Multiply and Add Word	MIPS32
MADDU	Multiply and Add Word Unsigned	MIPS32
MFHI	Move From HI	MIPS32
MFLO	Move From LO	MIPS32
MSUB	Multiply and Subtract Word	MIPS32
MSUBU	Multiply and Subtract Word Unsigned	MIPS32
MTHI	Move To HI	MIPS32
MTLO	Move To LO	MIPS32
MUL	Multiply Word to Register	MIPS32
MULT	Multiply Word	MIPS32
MULTU	Multiply Unsigned Word	MIPS32

4.1.3 Jump and Branch Instructions

This section describes the following:

- "Types of Jump and Branch Instructions Defined by the ISA"
- "Branch Delays and the Branch Delay Slot"
- "Branch and Branch Likely"
- "List of Jump and Branch Instructions"

4.1.3.1 Types of Jump and Branch Instructions Defined by the ISA

The architecture defines the following jump and branch instructions:

- PC-relative conditional branch
- PC-region unconditional jump
- Absolute (register) unconditional jump
- A set of procedure calls that record a return link address in a general register.

4.1.3.2 Branch Delays and the Branch Delay Slot

All branches have an architectural delay of one instruction. The instruction immediately following a branch is said to be in the **branch delay slot**. If a branch or jump instruction is placed in the branch delay slot, the operation of both instructions is undefined.

By convention, if an exception or interrupt prevents the completion of an instruction in the branch delay slot, the instruction stream is continued by re-executing the branch instruction. To permit this, branches must be restartable; procedure calls may not use the register in which the return link is stored (usually GPR *31*) to determine the branch target address.

4.1.3.3 Branch and Branch Likely

There are two versions of conditional branches; they differ in the manner in which they handle the instruction in the delay slot when the branch is not taken and execution falls through.

- **Branch** instructions execute the instruction in the delay slot.
- **Branch likely** instructions do not execute the instruction in the delay slot if the branch is not taken (they are said to *nullify* the instruction in the delay slot).

  Although the Branch Likely instructions are included in this specification, software is strongly encouraged to avoid the use of the Branch Likely instructions, as they will be removed from a future revision of the MIPS Architecture.

4.1.3.4 List of Jump and Branch Instructions

Table 4-12 lists instructions that jump to a procedure call within the current 256 MB-aligned region, or to an absolute address held in a register.

Table 4-12 lists the unconditional jump instructions within a given 256 MByte region. Table 4-13 lists branch instructions that compare two registers before conditionally executing a PC-relative branch. Table 4-14 lists branch instructions that test a register—compare with zero—before conditionally executing a PC-relative branch. Table 4-15 lists the deprecated Branch Likely Instructions.

Each table also lists the MIPS ISA within which an instruction is defined.

Table 4-12 Unconditional Jump Within a 256 Megabyte Region

Mnemonic	Instruction	Location to Which Jump Is Made	Defined in MIPS ISA
J	Jump	256 Megabyte Region	MIPS32
JAL	Jump and Link	256 Megabyte Region	MIPS32
JALR	Jump and Link Register	Absolute Address	MIPS32
JALR.HB	Jump and Link Register with Hazard Barrier	Absolute Address	MIPS32 Release 2
JALX	Jump and Link Exchange	Absolute Address	MIPS16e
JR	Jump Register	Absolute Address	MIPS32
JR.HB	Jump Register with Hazard Barrier	Absolute Address	MIPS32 Release 2

Table 4-13 PC-Relative Conditional Branch Instructions Comparing Two Registers

Mnemonic	Instruction	Defined in MIPS ISA
BEQ	Branch on Equal	MIPS32
BNE	Branch on Not Equal	MIPS32

Table 4-14 PC-Relative Conditional Branch Instructions Comparing With Zero

Mnemonic	Instruction	Defined in MIPS ISA
BGEZ	Branch on Greater Than or Equal to Zero	MIPS32
BGEZAL	Branch on Greater Than or Equal to Zero and Link	MIPS32
BGTZ	Branch on Greater Than Zero	MIPS32
BLEZ	Branch on Less Than or Equal to Zero	MIPS32
BLTZ	Branch on Less Than Zero	MIPS32
BLTZAL	Branch on Less Than Zero and Link	MIPS32

Table 4-15 Deprecated Branch Likely Instructions

Mnemonic	Instruction	Defined in MIPS ISA
BEQL	Branch on Equal Likely	MIPS32
BGEZALL	Branch on Greater Than or Equal to Zero and Link Likely	MIPS32
BGEZL	Branch on Greater Than or Equal to Zero Likely	MIPS32
BGTZL	Branch on Greater Than Zero Likely	MIPS32
BLEZL	Branch on Less Than or Equal to Zero Likely	MIPS32
BLTZALL	Branch on Less Than Zero and Link Likely	MIPS32
BLTZL	Branch on Less Than Zero Likely	MIPS32
BNEL	Branch on Not Equal Likely	MIPS32

4.1.4 Miscellaneous Instructions

Miscellaneous instructions include:

- "Instruction Serialization (SYNC and SYNCI)"
- "Exception Instructions"
- "Conditional Move Instructions"
- "Prefetch Instructions"
- "NOP Instructions"

4.1.4.1 Instruction Serialization (SYNC and SYNCI)

In normal operation, the order in which load and store memory accesses appear to a viewer *outside* the executing processor (for instance, in a multiprocessor system) is not specified by the architecture.

The SYNC instruction can be used to create a point in the executing instruction stream at which the relative order of some loads and stores can be determined: loads and stores executed before the SYNC are completed before loads and stores after the SYNC can start.

The SYNCI instruction synchronizes the processor caches with previous writes or other modifications to the instruction stream.

Table 4-16 lists the synchronization instructions, along with the MIPS ISA within which it is defined.

Table 4-16 Serialization Instruction

Mnemonic	Instruction	Defined in MIPS ISA
SYNC	Synchronize Shared Memory	MIPS32
SYNCI	Synchronize Caches to Make Instruction Writes Effective	MIPS32 Release 2

4.1.4.2 Exception Instructions

Exception instructions transfer control to a software exception handler in the kernel. There are two types of exceptions, *conditional* and *unconditional*. These are caused by the following instructions:

Trap instructions, which cause conditional exceptions based upon the result of a comparison

System call and breakpoint instructions, which cause unconditional exceptions

Table 4-17 lists the system call and breakpoint instructions. Table 4-18 lists the trap instructions that compare two registers. Table 4-19 lists trap instructions, which compare a register value with an *immediate* value.

Each table also lists the MIPS ISA within which an instruction is defined.

Table 4-17 System Call and Breakpoint Instructions

Mnemonic	Instruction	Defined in MIPS ISA
BREAK	Breakpoint	MIPS32
SYSCALL	System Call	MIPS32

Table 4-18 Trap-on-Condition Instructions Comparing Two Registers

Mnemonic	Instruction	Defined in MIPS ISA
TEQ	Trap if Equal	MIPS32
TGE	Trap if Greater Than or Equal	MIPS32
TGEU	Trap if Greater Than or Equal Unsigned	MIPS32
TLT	Trap if Less Than	MIPS32
TLTU	Trap if Less Than Unsigned	MIPS32II
TNE	Trap if Not Equal	MIPS32

Table 4-19 Trap-on-Condition Instructions Comparing an Immediate Value

Mnemonic	Instruction	Defined in MIPS ISA
TEQI	Trap if Equal Immediate	MIPS32
TGEI	Trap if Greater Than or Equal Immediate	MIPS32
TGEIU	Trap if Greater Than or Equal Immediate Unsigned	MIPS32

Table 4-19 Trap-on-Condition Instructions Comparing an Immediate Value

Mnemonic	Instruction	Defined in MIPS ISA
TLTI	Trap if Less Than Immediate	MIPS32
TLTIU	Trap if Less Than Immediate Unsigned	MIPS32
TNEI	Trap if Not Equal Immediate	MIPS32

4.1.4.3 Conditional Move Instructions

MIPS32 includes instructions to conditionally move one CPU general register to another, based on the value in a third general register. For floating point conditional moves, refer to Chapter 4.

Table 4-20 lists conditional move instructions, along with the MIPS ISA within which an instruction is defined.

Table 4-20 CPU Conditional Move Instructions

Mnemonic	Instruction	Defined in MIPS ISA
MOVF	Move Conditional on Floating Point False	MIPS32
MOVN	Move Conditional on Not Zero	MIPS32
MOVT	Move Conditional on Floating Point True	MIPS32
MOVZ	Move Conditional on Zero	MIPS32

4.1.4.4 Prefetch Instructions

There are two prefetch advisory instructions:

- One with register+offset addressing (PREF)
- One with register+register addressing (PREFX)

These instructions advise that memory is likely to be used in a particular way in the near future and should be prefetched into the cache. The PREFX instruction is encoded in the FPU *opcode* space, along with the other operations using register+register addressing

Table 4-21 Prefetch Instructions

Mnemonic	Instruction	Addressing Mode	Defined in MIPS ISA
PREF	Prefetch	Register+Offset	MIPS32
PREFX	Prefetch Indexed	Register+Register	MIPS64

4.1.4.5 NOP Instructions

The NOP instruction is actually encoded as an all-zero instruction. MIPS processors special-case this encoding as performing no operation, and optimize execution of the instruction. In addition, SSNOP instruction, takes up one issue cycle on any processor, including super-scalar implementations of the architecture.

Table 4-22 lists conditional move instructions, along with the MIPS ISA within which an instruction is defined.

Table 4-22 NOP Instructions

Mnemonic	Instruction	Defined in MIPS ISA
NOP	No Operation	MIPS32
SSNOP	Superscalar Inhibit NOP	MIPS32

4.1.5 Coprocessor Instructions

This section contains information about the following:

- "What Coprocessors Do"
- "System Control Coprocessor 0 (CP0)"
- "Floating Point Coprocessor 1 (CP1)"
- "Coprocessor Load and Store Instructions"

4.1.5.1 What Coprocessors Do

Coprocessors are alternate execution units, with register files separate from the CPU. In abstraction, the MIPS architecture provides for up to four coprocessor units, numbered 0 to 3. Each level of the ISA defines a number of these coprocessors, as listed in Table 4-23.

Table 4-23 Coprocessor Definition and Use in the MIPS Architecture

Coprocessor	MIPS32	MIPS64
CP0	Sys Control	Sys Control
CP1	FPU	FPU
CP2	implementation specific	
CP3	See Footnote	FPU (COP1X)

Coprocessor 0 is always used for system control and coprocessor 1 and 3 are used for the floating point unit. Coprocessor 2 is reserved for implementation-specific use.

A coprocessor may have two different register sets:

- Coprocessor general registers
- Coprocessor control registers

Each set contains up to 32 registers. Coprocessor computational instructions may use the registers in either set.