CPU Instruction Set Summary

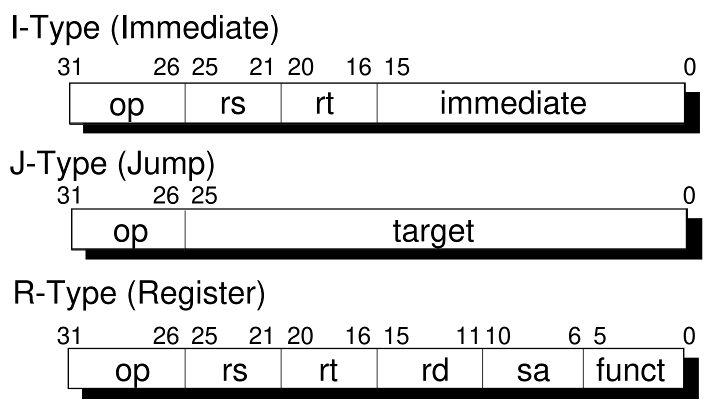
2

This chapter is an overview of the central processing unit (CPU) instruction set; refer to Appendix A for detailed descriptions of individual CPU instructions.

An overview of the floating-point unit (FPU) instruction set is in Chapter 6; refer to Appendix B for detailed descriptions of individual FPU instructions.

2.1 CPU Instruction Formats

Each CPU instruction consists of a single 32-bit word, aligned on a word boundary. There are three instruction formats—immediate (I-type), jump (J-type), and register (R-type)—as shown in Figure 2-1. The use of a small number of instruction formats simplifies instruction decoding, allowing the compiler to synthesize more complicated (and less frequently used) operations and addressing modes from these three formats as needed.



op	6-bit operation code
rs	5-bit source register specifier
rt	5-bit target (source/destination) register or branch condition
immediate	16-bit immediate value, branch displacement or address displacement
target	26-bit jump target address
rd	5-bit destination register specifier
sa	5-bit shift amount
funct	6-bit function field

Figure 2-1 CPU Instruction Formats

In the MIPS architecture, coprocessor instructions are implementation-dependent; see Appendix A for details of individual Coprocessor 0 instructions.

Load and Store Instructions

Load and store are immediate (I-type) instructions that move data between memory and the general registers. The only addressing mode that load and store instructions directly support is *base register plus 16-bit signed immediate offset*.

Scheduling a Load Delay Slot

A load instruction that does not allow its result to be used by the instruction immediately following is called a *delayed load instruction*. The instruction slot immediately following this delayed load instruction is referred to as the *load delay slot*.

In the R4000 processor, the instruction immediately following a load instruction can use the contents of the loaded register, however in such cases hardware interlocks insert additional real cycles. Consequently, scheduling load delay slots can be desirable, both for performance and R-Series processor compatibility. However, the scheduling of load delay slots is not absolutely required.

Defining Access Types

Access type indicates the size of an R4000 processor data item to be loaded or stored, set by the load or store instruction opcode. Access types are defined in Appendix A.

Regardless of access type or byte ordering (endianness), the address given specifies the low-order byte in the addressed field. For a big-endian configuration, the low-order byte is the most-significant byte; for a little-endian configuration, the low-order byte is the least-significant byte.†

The access type, together with the three low-order bits of the address, define the bytes accessed within the addressed doubleword (shown in Table 2-1). Only the combinations shown in Table 2-1 are permissible; other combinations cause address error exceptions. See Appendix A for individual descriptions of CPU load and store instructions.

† Data formats are described in Chapter 1.

Table 2-1 Byte Access within a Doubleword

Access Type Mnemonic (*Value*)	**Low Order Address Bits**			**Bytes Accessed**															
				Big endian (63-----------31------------0) Byte								**Little endian (63-----------31------------0) Byte**							
	2	**1**	**0**																
Doubleword (*7*)	0	0	0	0	1	2	3	4	5	6	7	7	6	5	4	3	2	1	0
Septibyte (*6*)	0	0	0	0	1	2	3	4	5	6			6	5	4	3	2	1	0
	0	0	1		1	2	3	4	5	6	7	7	6	5	4	3	2	1	
Sextibyte (*5*)	0	0	0	0	1	2	3	4	5					5	4	3	2	1	0
	0	1	0			2	3	4	5	6	7	7	6	5	4	3	2		
Quintibyte (*4*)	0	0	0	0	1	2	3	4							4	3	2	1	0
	0	1	1				3	4	5	6	7	7	6	5	4	3			
Word (*3*)	0	0	0	0	1	2	3									3	2	1	0
	1	0	0					4	5	6	7	7	6	5	4				
Triplebyte (*2*)	0	0	0	0	1	2											2	1	0
	0	0	1		1	2	3									3	2	1	
	1	0	0					4	5	6			6	5	4				
	1	0	1						5	6	7	7	6	5					
Halfword (*1*)	0	0	0	0	1													1	0
	0	1	0			2	3									3	2		
	1	0	0					4	5					5	4				
	1	1	0							6	7	7	6						
Byte (*0*)	0	0	0	0															0
	0	0	1		1													1	
	0	1	0			2											2		
	0	1	1				3									3			
	1	0	0					4							4				
	1	0	1						5					5					
	1	1	0							6			6						
	1	1	1								7	7							

Computational Instructions

Computational instructions can be either in register (R-type) format, in which both operands are registers, or in immediate (I-type) format, in which one operand is a 16-bit immediate.

Computational instructions perform the following operations on register values:

- arithmetic
- logical
- shift
- multiply
- divide

These operations fit in the following four categories of computational instructions:

- ALU Immediate instructions
- three-Operand Register-Type instructions
- shift instructions
- multiply and divide instructions

64-bit Operations

When operating in 64-bit mode, 32-bit operands must be sign extended. The result of operations that use incorrect sign-extended 32-bit values is unpredictable.

Cycle Timing for Multiply and Divide Instructions

Any multiply instruction in the integer pipeline is transferred to the multiplier as remaining instructions continue through the pipeline; the product of the multiply instruction is saved in the HI and LO registers.

If the multiply instruction is followed by an MFHI or MFLO before the product is available, the pipeline interlocks until this product does become available.

Table 2-2 gives the execution time for integer multiply and divide operations. The "Total Cycles" column gives the total number of cycles required to execute the instruction. The "Overlap" column gives the number of cycles that overlap other CPU operations; that is, the number of cycles required between the present instruction and a subsequent MFHI or MFLO without incurring an interlock. If this value is zero, the operation is not performed in parallel with any other CPU operation.

Table 2-2 Multiply/Divide Instruction Cycle Timing

Instruction	Total Cycles	Overlap
MULT	12	10
MULTU	12	10
DIV	75	0
DIVU	75	0
DMULT	20	18
DMULTU	20	18
DDIV	139	0
DDIVU	139	0

For more information about computational instructions, refer to the individual instruction as described in Appendix A.

Jump and Branch Instructions

Jump and branch instructions change the control flow of a program. All jump and branch instructions occur with a delay of one instruction: that is, the instruction immediately following the jump or branch (this is known as the instruction in the *delay slot*) always executes while the target instruction is being fetched from storage.†

Overview of Jump Instructions

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

Returns, dispatches, and large cross-page jumps are usually implemented with the Jump Register or Jump and Link Register instructions. Both are R-type instructions that take the 32-bit or 64-bit byte address contained in one of the general purpose registers.

For more information about jump instructions, refer to the individual instruction as described in Appendix A.

Overview of Branch Instructions

All branch instruction target addresses are computed by adding the address of the instruction in the delay slot to the 16-bit *offset* (shifted left 2 bits and sign-extended to 32 bits). All branches occur with a delay of one instruction.

If a conditional branch likely is not taken, the instruction in the delay slot is nullified.

For more information about branch instructions, refer to the individual instruction as described in Appendix A.

† Taken branches have a 3 cycle penalty in this implementation. See Chapter 3 for more information.

Special Instructions

Special instructions allow the software to initiate traps; they are always R-type. For more information about special instructions, refer to the individual instruction as described in Appendix A.

Exception Instructions

Exception instructions are extensions to the MIPS ISA. For more information about exception instructions, refer to the individual instruction as described in Appendix A.

Coprocessor Instructions

Coprocessor instructions perform operations in their respective coprocessors. Coprocessor loads and stores are I-type, and coprocessor computational instructions have coprocessor-dependent formats.

Individual coprocessor instructions are described in Appendices A (for CP0) and B (for the FPU, CP1).

CP0 instructions perform operations specifically on the System Control Coprocessor registers to manipulate the memory management and exception handling facilities of the processor. Appendix A details CP0 instructions.