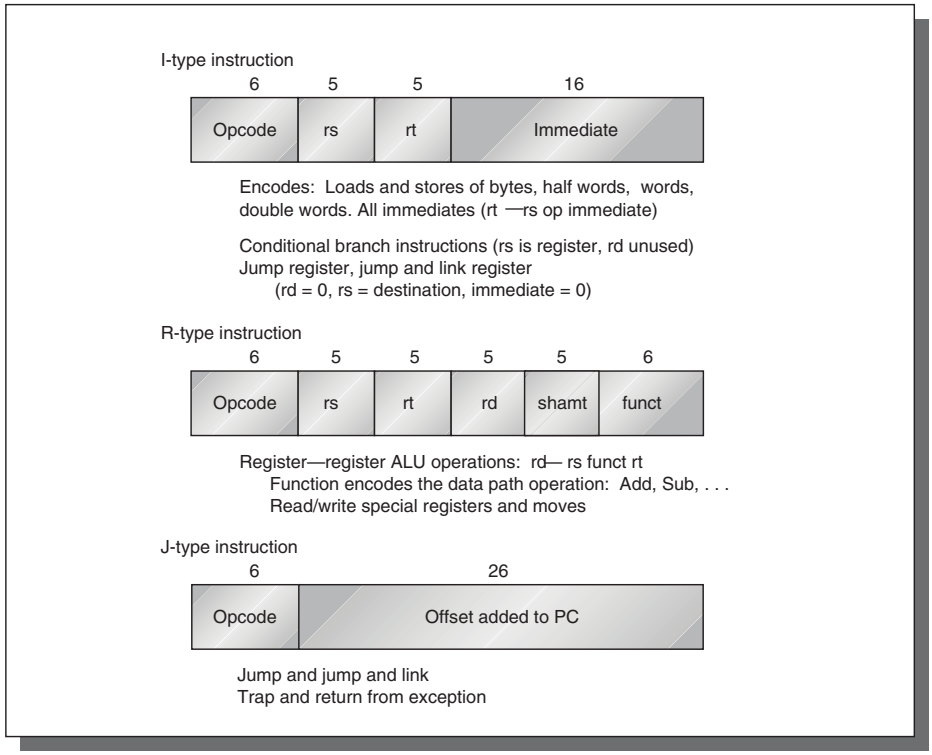

FIGURE 2.27 Instruction layout for MIPS. All instructions are encoded in one of three types, with common fields in the same location in each format.

Addressing modes for MIPS data transfers

The only data addressing modes are immediate and displacement, both with 16-bit fields. Register indirect is accomplished simply by placing 0 in the 16-bit displacement field, and absolute addressing with a 16-bit field is accomplished by using register 0 as the base register. Embracing zero gives us four effective modes, although only two are supported in the architecture.

MIPS memory is byte addressable in Big Endian mode with a 64-bit address. As it is a load-store architecture, all references between memory and either GPRs or FPRs are through loads or stores. Supporting the data types mentioned above, memory accesses involving GPRs can be to a byte, half word, word, or double word. The FPRs may be loaded and stored with single-precision or double-precision numbers. All memory accesses must be aligned.

MIPS Instruction Format

Since MIPS has just two addressing modes, these can be encoded into the opcode. Following the advice on making the processor easy to pipeline and decode,

all instructions are 32 bits with a 6-bit primary opcode. Figure 2.27 shows the instruction layout. These formats are simple while providing 16-bit fields for displacement addressing, immediate constants, or PC-relative branch addresses.

Appendix B shows a variant of MIPS—called MIPS16—which has 16-bit and 32-bit instructions to improve code density for embedded applications. We will stick to the traditional 32-bit format in this book.

MIPS Operations

MIPS supports the list of simple operations recommended above plus a few others. There are four broad classes of instructions: loads and stores, ALU operations, branches and jumps, and floating-point operations.

Example instruction	Instruction name	Meaning
LD R1,30(R2)	Load double word	Regs[R1]$\leftarrow_{64}$ Mem[30+Regs[R2]]
LD R1,1000(R0)	Load double word	Regs[R1]$\leftarrow_{64}$ Mem[1000+0]
LW R1,60(R2)	Load word	Regs[R1]$\leftarrow_{64}$ (Mem[60+Regs[R2]]$_0$)^{32} ## Mem[60+Regs[R2]]
LB R1,40(R3)	Load byte	Regs[R1]$\leftarrow_{64}$ (Mem[40+Regs[R3]] $_0$)^{56} ## Mem[40+Regs[R3]]
LBU R1,40(R3)	Load byte unsigned	Regs[R1]$\leftarrow_{64}$ 0^{56} ## Mem[40+Regs[R3]]
LH R1,40(R3)	Load half word	Regs[R1]$\leftarrow_{64}$ (Mem[40+Regs[R3]]$_0$)^{48} ## Mem[40+Regs[R3]]##Mem[41+Regs[R3]]
L.S F0,50(R3)	Load FP single	Regs[F0]$\leftarrow_{64}$ Mem[50+Regs[R3]] ## 0^{32}
L.D F0,50(R2)	Load FP double	Regs[F0]$\leftarrow_{64}$ Mem[50+Regs[R2]]
SD R3,500(R4)	Store double word	Mem[500+Regs[R4]]$\leftarrow_{64}$ Regs[R3]
SW R3,500(R4)	Store word	Mem[500+Regs[R4]]$\leftarrow_{32}$ Regs[R3]
S.S F0,40(R3)	Store FP single	Mem[40+Regs[R3]]$\leftarrow_{32}$ Regs[F0]$_{0..31}$
S.D F0,40(R3)	Store FP double	Mem[40+Regs[R3]]$\leftarrow_{64}$ Regs[F0]
SH R3,502(R2)	Store half	Mem[502+Regs[R2]]$\leftarrow_{16}$ Regs[R3]$_{48..63}$
SB R2,41(R3)	Store byte	Mem[41+Regs[R3]]$\leftarrow_{8}$ Regs[R2]$_{56..63}$

FIGURE 2.28 The load and store instructions in MIPS. All use a single addressing mode and require that the memory value be aligned. Of course, both loads and stores are available for all the data types shown.

Any of the general-purpose or floating-point registers may be loaded or stored, except that loading R0 has no effect. Figure 2.28 gives examples of the load and store instructions. Single-precision floating-point numbers occupy half a floating-point register. Conversions between single and double precision must be done explicitly. The floating-point format is IEEE 754 (see Appendix G). A list of the all the MIPS instructions in our subset appears in Figure 2.31 (page 146).

To understand these figures we need to introduce a few additional extensions to our C description language presented initially on page 107:

- A subscript is appended to the symbol ← whenever the length of the datum being transferred might not be clear. Thus, $\leftarrow_n$ means transfer an n-bit quantity. We use $x, y \leftarrow z$ to indicate that z should be transferred to x and y.
- A subscript is used to indicate selection of a bit from a field. Bits are labeled from the most-significant bit starting at 0. The subscript may be a single digit (e.g., $\texttt{Regs[R4]}_0$ yields the sign bit of R4) or a subrange (e.g., $\texttt{Regs[R3]}_{56..63}$ yields the least-significant byte of R3).
- The variable `Mem`, used as an array that stands for main memory, is indexed by a byte address and may transfer any number of bytes.
- A superscript is used to replicate a field (e.g., 0^{48} yields a field of zeros of length 48 bits).
- The symbol ## is used to concatenate two fields and may appear on either side of a data transfer.

A summary of the entire description language appears on the back inside cover. As an example, assuming that R8 and R10 are 64-bit registers:

$$\texttt{Regs[R10]}_{32..63} \leftarrow_{32} (\texttt{Mem[Regs[R8]]}_0)^{24} \texttt{ ## Mem[Regs[R8]]}$$

means that the byte at the memory location addressed by the contents of register R8 is sign-extended to form a 32-bit quantity that is stored into the lower half of register R10. (The upper half of R10 is unchanged.)

All ALU instructions are register-register instructions. Figure 2.29 gives some examples of the arithmetic/logical instructions. The operations include simple arithmetic and logical operations: add, subtract, AND, OR, XOR, and shifts. Immediate forms of all these instructions are provided using a 16-bit sign-extended immediate. The operation `LUI` (load upper immediate) loads bits 32 to 47 of a register, while setting the rest of the register to 0. `LUI` allows a 32-bit constant to be built in two instructions, or a data transfer using any constant 32-bit address in one extra instruction.

As mentioned above, R0 is used to synthesize popular operations. Loading a constant is simply an add immediate where one source operand is R0, and a register-register move is simply an add where one of the sources is R0. (We sometimes use the mnemonic `LI`, standing for load immediate, to represent the former and the mnemonic `MOV` for the latter.)

MIPS Control Flow Instructions

MIPS provides compare instructions, which compare two registers to see if the first is less than the second. If the condition is true, these instructions place a

Example instruction	Instruction name	Meaning
DADDU R1,R2,R3	Add unsigned	Regs[R1]←Regs[R2]+Regs[R3]
DADDIU R1,R2,#3	Add immediate unsigned	Regs[R1]←Regs[R2]+3
LUI R1,#42	Load upper immediate	Regs[R1]←0^{32}##42##0^{16}
SLL R1,R2,#5	Shift left logical	Regs[R1]←Regs[R2]<<5
SLT R1,R2,R3	Set less than	if (Regs[R2]<Regs[R3]) Regs[R1]←1 else Regs[R1]←0

FIGURE 2.29 Examples of arithmetic/logical instructions on MIPS, both with and without immediates.

1 in the destination register (to represent true); otherwise they place the value 0. Because these operations "set" a register, they are called set-equal, set-not-equal, set-less-than, and so on. There are also immediate forms of these compares.

Example instruction	Instruction name	Meaning
J name	Jump	$PC_{36..63}$←name
JAL name	Jump and link	Regs[R31]←PC+4; $PC_{36..63}$←name; ((PC+4)-2^{27}) ≤ name < ((PC+4)+2^{27})
JALR R2	Jump and link register	Regs[R31]←PC+4; PC←Regs[R2]
JR R3	Jump register	PC←Regs[R3]
BEQZ R4,name	Branch equal zero	if (Regs[R4]==0) PC←name; ((PC+4)-2^{17}) ≤ name < ((PC+4)+2^{17})
BNE R3,R4,name	Branch not equal zero	if (Regs[R3]!= Regs[R4]) PC←name; ((PC+4)-2^{17}) ≤ name < ((PC+4)+2^{17})
MOVZ R1,R2,R3	Conditional move if zero	if (Regs[R3]==0) Regs[R1]←Regs[R2]

FIGURE 2.30 Typical control-flow instructions in MIPS. All control instructions, except jumps to an address in a register, are PC-relative. Note that the branch distances are longer than the address field would suggestion; since MIPS instructions are all 32-bits long, the byte branch address is multiplied by 4 to get a longer distance.

Control is handled through a set of jumps and a set of branches. Figure 2.30 gives some typical branch and jump instructions. The four jump instructions are differentiated by the two ways to specify the destination address and by whether or not a link is made. Two jumps use a 26-bit offset shifted two bits and then replaces the lower 28 bits of the program counter (of the instruction sequentially following the jump) to determine the destination address. The other two jump instructions specify a register that contains the destination address. There are two flavors of jumps: plain jump, and jump and link (used for procedure calls). The latter places the return address—the address of the next sequential instruction—in R31.

Instruction type/opcode	Instruction meaning
Data transfers	**Move data between registers and memory, or between the integer and FP or special registers; only memory address mode is 16-bit displacement + contents of a GPR**
LB,LBU,SB	Load byte, load byte unsigned, store byte (to/from integer registers)
LH,LHU,SH	Load half word, load half word unsigned, store half word (to/from integer registers)
LW,LWU,SW	Load word, Load word unsigned, store word (to/from integer registers)
LD,SD	Load double word, store double word (to/from integer registers)
L.S,L.D,S.S,S.D	Load SP float, load DP float, store SP float, store DP float
MFC0,MTC0	Move from/to GPR to/from a special register
MOV.S,MOV.D	Copy one SP or DP FP register to another FP register
MFC1,MTC1	Move 32 bits from/to FP registers to/from integer registers
Arithmetic/logical	**Operations on integer or logical data in GPRs; signed arithmetic trap on overflow**
DADD,DADDI,DADDU, DADDIU	Add, add immediate (all immediates are 16 bits); signed and unsigned
DSUB,DSUBU	Subtract, subtract immediate; signed and unsigned
DMUL,DMULU,DDIV,DDIVU	Multiply and divide, signed and unsigned; all operations take and yield 64-bit values
AND,ANDI	And, and immediate
OR,ORI,XOR,XORI	Or, or immediate, exclusive or, exclusive or immediate
LUI	Load upper immediate—loads bits 32 to 47 of register with immediate; then sign extends
DSLL, SDRL, DSRA, DSLLV, DSRLV, DSRAV	Shifts: both immediate (DS__) and variable form (DS__V); shifts are shift left logical, right logical, right arithmetic
SLT,SLTI,SLTU,SLTIU	Set less than, set less than immediate; signed and unsigned
Control	**Conditional branches and jumps; PC-relative or through register**
BEQZ,BNEZ	Branch GPR equal/not equal to zero; 16-bit offset from PC+4
BC1T,BC1F	Test comparison bit in the FP status register and branch; 16-bit offset from PC+4
J, JR	Jumps: 26-bit offset from PC+4 (J) or target in register (JR)
JAL, JALR	Jump and link: save PC+4 in R31, target is PC-relative (JAL) or a register (JALR)
TRAP	Transfer to operating system at a vectored address
ERET	Return to user code from an exception; restore user mode
Floating point	**FP operations on DP and SP formats**
ADD.D,ADD.S,ADD.PS	Add DP, SP numbers, an d pairs of SP numbers
SUB.D,SUB.S,ADD.PS	Subtract DP, SP numbers, an d pairs of SP numbers
MUL.D,MUL.S,MUL.PS	Multiply DP, SP floating point, an d pairs of SP numbers
DIV.D,DIV.S,DIV.PS	Divide DP, SP floating point, an d pairs of SP numbers
CVT._._	Convert instructions: CVT.x.y converts from type x to type y, where x and y are L (64-bit integer), W (32-bit integer), D (DP), or S (SP). Both operands are FPRs.
C.__.D,C.__.S	DP and SP compares: "__" = LT,GT,LE,GE,EQ,NE; sets bit in FP status register

FIGURE 2.31 Subset of the instructions in MIPS64. Figure 2.27 lists the formats of these instructions. SP = single precision; DP = double precision. This list can also be found on the page preceding the back inside cover.