Introduction

1

Historically, the evolution of computer architectures has been dominated by families of increasingly complex central processors. Under market pressures to preserve existing software, complex instruction set computer (CISC) architectures evolved by the accretion of microcode and increasingly intricate instruction sets. This intricacy in architecture was itself driven by the need to support high-level languages and operating systems, as advances in semiconductor technology made it possible to fabricate integrated circuits of greater and greater complexity. And at that time it seemed self-evident to designers that architectures should continue to become more and more complex as technological advances made such VLSI designs possible.

In recent years, however, reduced instruction set computer (RISC) architectures are implementing a different model for the interaction between hardware, firmware, and software. RISC concepts emerged from a statistical analysis of the way in which software actually uses processor resources: dynamic measurement of system kernels and object modules generated by optimizing compilers showed that the simplest instructions were used most often—even in the code for CISC machines. Correspondingly, complex instructions often went unused because their single way of performing a complex operation rarely matched the precise needs of a high-level language.

RISC architecture eliminates microcode routines and turns low-level control of the machine over to software. The RISC approach is not new, but its application has become more prevalent in recent years, due to the increasing use of high-level languages, the development of compilers that are able to optimize at the microcode level, and dramatic advances in semiconductor memory and packaging. It is now feasible to replace relatively slow microcode ROM with faster RAM that is organized as an instruction cache. Machine control resides in this instruction cache that is, in effect, customized on-the-fly: the instruction stream generated by system- and compiler-generated code provides a precise fit between the requirements of high-level software and the low-level capabilities of the hardware.

Reducing or simplifying the instruction set was not the primary goal of RISC architecture; it is a pleasant side effect of techniques used to gain the highest performance possible from available technology. Thus, the term *reduced instruction set computer*s is a bit misleading; it is the push for performance that really drives and shapes RISC designs.

1.1 Benefits of RISC Design

Some benefits that result from RISC design techniques are not directly attributable to the drive to increase performance, but are a result of the basic reduction in complexity—a simpler design allows both chip-area resources and human resources to be applied to features that enhance performance. Some of these benefits are described below.

Shorter Design Cycle

The architectures of RISC processors can be implemented more quickly than their CISC counterparts: it is easier to fabricate and debug a streamlined, simplified architecture with no microcode than a complex architecture that uses microcode. CISC processors have such a long design cycle that they may not be completely debugged by the time they are technologically obsolete. The shorter time required to design and implement RISC processors allows them to make use of the best available technologies.

Effective Utilization of Chip Area

The simplicity of RISC processors also frees scarce chip geography for performance-critical resources such as larger register files, translation lookaside buffers (TLBs), coprocessors, and fast multiply and divide units. Such resources help RISC processors obtain an even greater performance edge.

User (Programmer) Benefits

Simplicity in architecture also helps the user by providing a uniform instruction set that is easier to use. This allows a closer correlation between the instruction count and the cycle count, making it easier to measure code optimization activities.

Advanced Semiconductor Technologies

Each new VLSI technology is introduced with tight limits on the number of transistors that fit on each chip. Since the simplicity of a RISC processor allows it to be implemented in fewer transistors than its CISC counterpart, the first computers capable of exploiting these new VLSI technologies have been using and will continue to use RISC architecture.

Optimizing Compilers

RISC architecture is designed so that the compilers, not assembly languages, have the optimal working environment. RISC philosophy assumes that high-level language programming is used, which contradicts the older CISC philosophy that assumes assembly language programming is of primary importance.

The trend toward high-level language instructions has led to the development of more efficient compilers to convert high-level language instructions to machine code. Primary measures of compiler efficiency are the compactness of its generated code and the shortness of its execution time.

During the development of more efficient compilers, analysis of instruction streams revealed that the greatest amount of time was spent executing simple instructions and performing load and store operations, while the more complex instructions were used less frequently. It was also learned that compilers produce code that is often a narrow subset of the processor instruction set architecture (ISA). A compiler works more efficiently with instructions that perform simple, well-defined operations and generate minimal side-effects. Compilers do not use complex instructions and features; the more complex, powerful instructions are either too difficult for the compiler to employ or those instructions do not precisely fit high-level language requirements.

Thus, a natural match exists between RISC architectures and efficient, optimizing compilers. This match makes it easier for compilers to generate the most effective sequences of machine instructions to accomplish tasks defined by the high-level language.

MIPS RISCompiler Language Suite

Some compiler products are derived from disparate sources and consequently do not fit together very well. Instead of treating each language's compiler as a separate entity, the MIPS RISCompiler™ language suite shares common elements across the entire family of compilers. In this way the language suite offers both tight integration and broad language coverage.

The MIPS language suite supports:

- industry-standard front ends for the following languages (C, FORTRAN, Pascal)
- a common intermediate language, offering an efficient way to add language front ends over time
- all of the back end optimization and code generation
- the same object format and calling conventions
- mixed-language programs
- debugging of programs written in all languages, including mixtures

This language suite approach yields high-quality compilers for all languages, since common elements make up the majority of each of the language products. In addition, this approach provides the ability to develop and execute multi-language programs, promoting flexibility in development, avoiding the necessity of recoding proven program segments, and protecting the user's software investment. The common back-end also exports optimizing and code-generating improvements immediately throughout the language suite, thereby reducing maintenance.

1.2 Compatibility

The R4000 processor provides complete application software compatibility with the MIPS R2000, R3000, and R6000 processors. Although the MIPS processor architecture has evolved in response to a compromise between software and hardware resources in the computer system, the R4000 processor implements the MIPS ISA for user-mode programs. This guarantees that user programs conforming to the ISA execute on any MIPS hardware implementation.

1.3 Processor General Features

This section briefly describes the programming model, the memory management unit (MMU), and the caches in the R4000 processor. A more detailed description is given in succeeding sections.

- **Full 32-bit and 64-bit Operations.** The R4000 processor contains 32 general purpose 64-bit registers. (When operating as a 32-bit processor, the general purpose registers are 32-bits wide.) All instructions are 32 bits wide.
- **Efficient Pipeline.** The superpipeline design of the processor results in an execution rate approaching one instruction per cycle. Pipeline stalls and exceptional events are handled precisely and efficiently.
- **MMU.** The R4000 processor uses an on-chip TLB that provides rapid virtual-to-physical address translation.
- **Cache Control.** The R4000 primary instruction and data caches reside on-chip, and can each hold 8 Kbytes. In the R4400 processor, the primary caches can each hold 16 Kbytes. Architecturally, each primary cache can be increased to hold up to 32 Kbytes. An off-chip secondary cache (R4000SC and R4000MC processors only) can hold from 128 Kbytes to 4 Mbytes. All processor cache control logic, including the secondary cache control logic, is on-chip.
- **Floating-Point Unit**. The FPU is located on-chip and implements the ANSI/IEEE standard 754-1985.

1.4 R4000 Processor Configurations

The R4000 processor† is packaged in three different configurations. All processors are implemented in sub-1-micron CMOS technology.

- **R4000PC** is designed for cost-sensitive systems such as inexpensive desktop systems and high-end embedded controllers. It is packaged in a 179-pin PGA, and does not support a secondary cache.
- **R4000SC** is designed for high-performance uniprocessor systems. It is packaged in a 447-pin LGA/PGA and includes integrated control for large secondary caches built from standard SRAMs.
- **R4000MC** is designed for large cache-coherent multiprocessor systems. It is packaged in a 447-pin LGA/PGA and, in addition to the features of R4000SC, includes support for a wide variety of bus designs and cache-coherency mechanisms.

Table 1-1 lists the features in each of the three configurations (**X** indicates the feature is present). R4400 processor enhancements are described in the section following.

1.5 R4400 Processor Enhancements

In addition to the features contained in the R4000 processor, the R4400 processor has the following enhancements:

- fully functional Status pins (described in Chapter 10)
- Master/Checker mode (described in Chapter 16)
- larger primary caches (described in Processor General Features, in this chapter)
- uncached store buffer (described in Chapter 3)
- divide-by-6 and divide-by-8 modes (described in Chapter 10)
- cache error bit, *EW*, added to the *CacheErr* register (described in Chapter 5).

† Features of the R4400 processor that differ from the R4000 processor are noted throughout this book; for instance, R4400 processor enhancements are listed in the next section. Otherwise, references to the R4000 processor may be taken to include the R4400 processor.

Table 1-1 R4000 Features

Feature	R4000PC	R4000SC	R4000MC
Primary Cache States			
Valid	X	X	X
Shared			X
Clean Exclusive		X	X
Dirty Exclusive	X	X	X
Secondary Cache Interface		X	X
Secondary Cache States			
Valid	X	X	X
Shared			X
Dirty Shared			X
Clean Exclusive		X	X
Dirty Exclusive	X	X	X
Multiprocessing			X
Cache Coherency Attributes			
Uncached	X	X	X
Noncoherent	X	X	X
Sharable			X
Update			X
Exclusive			X
Packages			
PGA (179-pin)	X		
PGA (447-pin)		X	X

1.6 R4000 Processor

This section describes the following:

- the 64-bit architecture of the R4000 processor
- the superpipeline design of the CPU instruction pipeline (described in detail in Chapter 3)
- an overview of the System interface (described in detail in Chapter 12)
- an overview of the CPU registers (detailed in Chapters 4 and 5) and CPU instruction set (detailed in Chapter 2 and Appendix A)
- data formats and byte ordering
- the System Control Coprocessor, CP0, and the floating-point unit, CP1
- caches and memory, including a description of primary and secondary caches, the memory management unit (MMU), the translation lookaside buffer (TLB), and the Secondary Cache interface (described in more detail in Chapters 4 and 11). The Secondary Cache interface is detailed in Chapter 13.

64-bit Architecture

The natural mode of operation for the R4000 processor is as a 64-bit microprocessor; however, 32-bit applications maintain compatibility even when the processor operates as a 64-bit processor.

The R4000 processor provides the following:

- 64-bit on-chip floating-point unit (FPU)
- 64-bit integer arithmetic logic unit (ALU)
- 64-bit integer registers
- 64-bit virtual address space
- 64-bit system bus

Figure 1-1 is a block diagram of the R4000 processor internals.

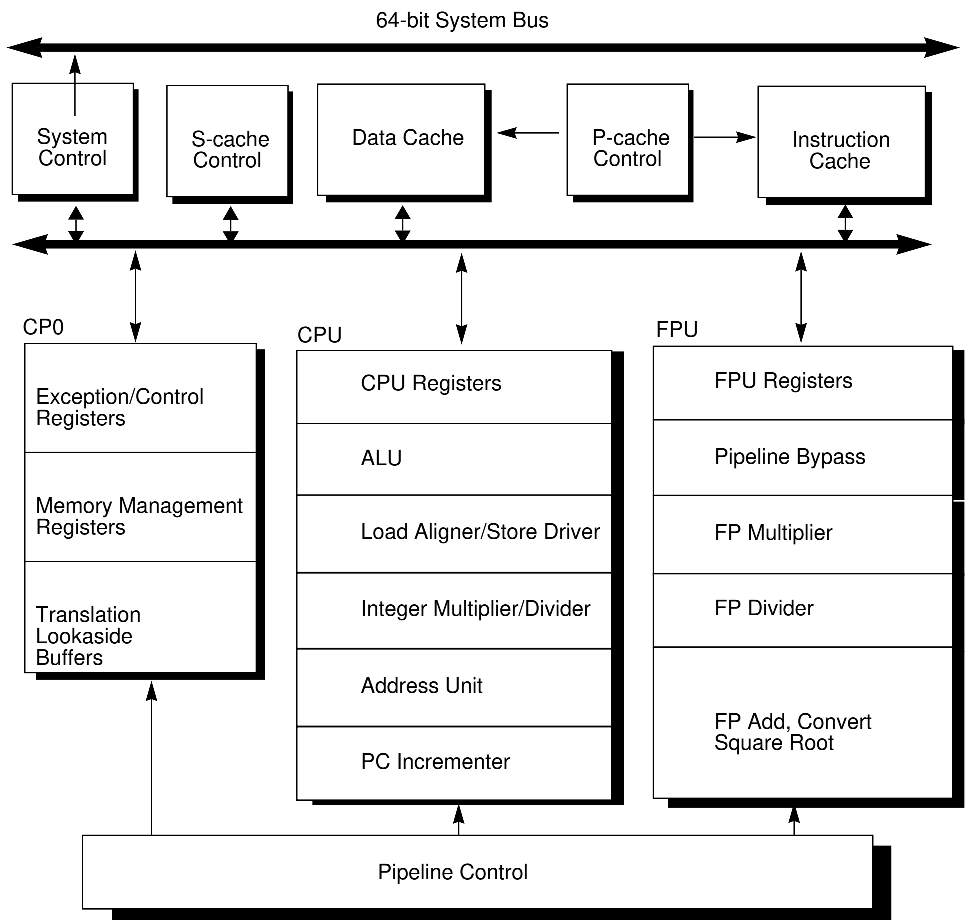

Figure 1-1 R4000 Processor Internal Block Diagram

Superpipeline Architecture

The R4000 processor exploits instruction parallelism by using an eight-stage superpipeline which places no restrictions on the instruction issued. Under normal circumstances, two instructions are issued each cycle.

The internal pipeline of the R4000 processor operates at twice the frequency of the master clock, as discussed in Chapter 3. The processor achieves high throughput by pipelining cache accesses, shortening register access times, implementing virtual-indexed primary caches, and allowing the latency of functional units to span more than one pipeline clock cycles.

System Interface

The R4000 processor supports a 64-bit System interface that can construct uniprocessor systems with a direct DRAM interface—with or without a secondary cache—or cache-coherent multiprocessor systems. The System interface includes:

- a 64-bit multiplexed address and data bus
- 8 check bits
- a 9-bit parity-protected command bus
- 8 handshake signals

The interface is capable of transferring data between the processor and memory at a peak rate of 400 Mbytes/second, when running at 50 MHz.

CPU Register Overview

The central processing unit (CPU) provides the following registers:

- 32 general purpose registers
- a Program Counter (*PC*) register
- 2 registers that hold the results of integer multiply and divide operations (*HI* and *LO*).

Floating-point unit (FPU) registers are described in Chapter 6.

CPU registers can be either 32 bits or 64 bits wide, depending on the R4000 processor mode of operation.

Figure 1-2 shows the CPU registers.

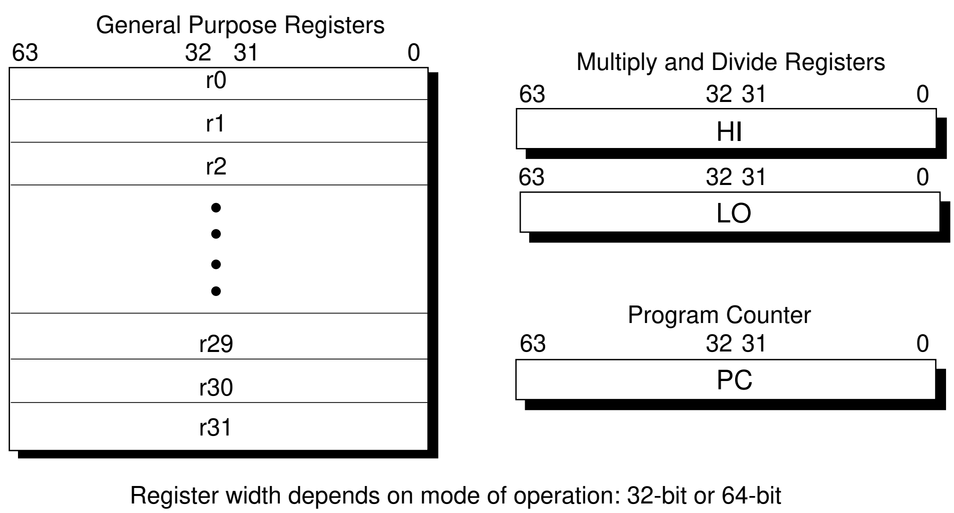

Figure 1-2 CPU Registers

Two of the CPU general purpose registers have assigned functions:

- *r0* is hardwired to a value of zero, and can be used as the target register for any instruction whose result is to be discarded. *r0* can also be used as a source when a zero value is needed.
- *r31* is the link register used by Jump and Link instructions. It should not be used by other instructions.

The CPU has three special purpose registers:

- *PC* — Program Counter register
- *HI* — Multiply and Divide register higher result
- *LO* — Multiply and Divide register lower result

The two Multiply and Divide registers (*HI*, *LO*) store:

- the product of integer multiply operations, or
- the quotient (in *LO*) and remainder (in *HI*) of integer divide operations

The R4000 processor has no *Program Status Word* (PSW) register as such; this is covered by the *Status* and *Cause* registers incorporated within the System Control Coprocessor (CP0). CP0 registers are described later in this chapter.

CPU Instruction Set Overview

Each CPU instruction is 32 bits long. As shown in Figure 1-3, there are three instruction formats:

- immediate (I-type)
- jump (J-type)
- register (R-type)

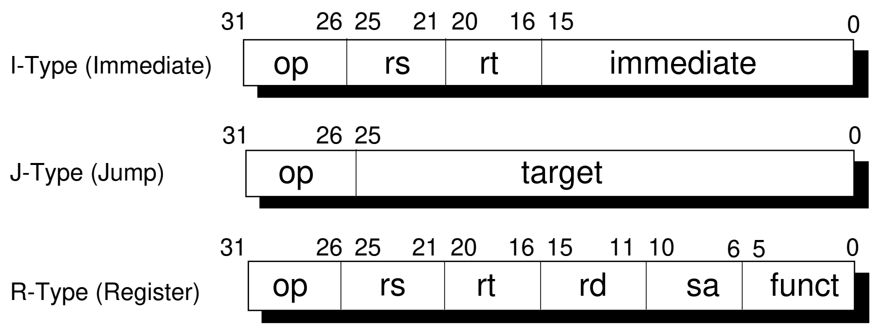

Figure 1-3 CPU Instruction Formats

Each format contains a number of different instructions, which are described further in this chapter. Fields of the instruction formats are described in Chapter 2.

Instruction decoding is greatly simplified by limiting the number of formats to these three. This limitation means that the more complicated (and less frequently used) operations and addressing modes can be synthesized by the compiler, using sequences of these same simple instructions.

The instruction set can be further divided into the following groupings:

- **Load and Store** instructions move data between memory and general registers. They are all immediate (I-type) instructions, since the only addressing mode supported is base register plus 16-bit, signed immediate offset.
- **Computational** instructions perform arithmetic, logical, shift, multiply, and divide operations on values in registers. They include register (R-type, in which both the operands and the result are stored in registers) and immediate (I-type, in which one operand is a 16-bit immediate value) formats.
- **Jump and Branch** instructions change the control flow of a program. Jumps are always made to a paged, absolute address formed by combining a 26-bit target address with the high-order bits of the Program Counter (J-type format) or register address (R-type format). Branches have 16-bit offsets relative to the program counter (I-type). Jump And Link instructions save their return address in register 31.
- **Coprocessor** instructions perform operations in the coprocessors. Coprocessor load and store instructions are I-type.
- **Coprocessor 0** (system coprocessor) instructions perform operations on CP0 registers to control the memory management and exception handling facilities of the processor. These are listed in Table 1-18.
- **Special** instructions perform system calls and breakpoint operations. These instructions are always R-type.
- **Exception** instructions cause a branch to the general exception-handling vector based upon the result of a comparison. These instructions occur in both R-type (both the operands and the result are registers) and I-type (one operand is a 16-bit immediate value) formats.

Chapter 2 provides a more detailed summary and Appendix A gives a complete description of each instruction.

Tables 1-2 through 1-17 list CPU instructions common to MIPS R-Series processors, along with those instructions that are extensions to the instruction set architecture. The extensions result in code space reductions, multiprocessor support, and improved performance in operating system kernel code sequences—for instance, in situations where run-time bounds-checking is frequently performed. Table 1-18 lists CP0 instructions.

Table 1-2 CPU Instruction Set: Load and Store Instructions

OpCode	Description
LB	Load Byte
LBU	Load Byte Unsigned
LH	Load Halfword
LHU	Load Halfword Unsigned
LW	Load Word
LWL	Load Word Left
LWR	Load Word Right
SB	Store Byte
SH	Store Halfword
SW	Store Word
SWL	Store Word Left
SWR	Store Word Right

Table 1-3 CPU Instruction Set: Arithmetic Instructions (ALU Immediate)

OpCode	Description
ADDI	Add Immediate
ADDIU	Add Immediate Unsigned
SLTI	Set on Less Than Immediate
SLTIU	Set on Less Than Immediate Unsigned
ANDI	AND Immediate
ORI	OR Immediate
XORI	Exclusive OR Immediate
LUI	Load Upper Immediate

Table 1-4 CPU Instruction Set: Arithmetic (3-Operand, R-Type)

OpCode	Description
ADD	Add
ADDU	Add Unsigned
SUB	Subtract
SUBU	Subtract Unsigned
SLT	Set on Less Than
SLTU	Set on Less Than Unsigned
AND	AND
OR	OR
XOR	Exclusive OR
NOR	NOR

Table 1-5 CPU Instruction Set: Multiply and Divide Instructions

OpCode	Description
MULT	Multiply
MULTU	Multiply Unsigned
DIV	Divide
DIVU	Divide Unsigned
MFHI	Move From HI
MTHI	Move To HI
MFLO	Move From LO
MTLO	Move To LO

Table 1-6 CPU Instruction Set: Jump and Branch Instructions

OpCode	Description
J	Jump
JAL	Jump And Link
JR	Jump Register
JALR	Jump And Link Register
BEQ	Branch on Equal
BNE	Branch on Not Equal
BLEZ	Branch on Less Than or Equal to Zero
BGTZ	Branch on Greater Than Zero
BLTZ	Branch on Less Than Zero
BGEZ	Branch on Greater Than or Equal to Zero
BLTZAL	Branch on Less Than Zero And Link
BGEZAL	Branch on Greater Than or Equal to Zero And Link

Table 1-7 CPU Instruction Set: Shift Instructions

OpCode	Description
SLL	Shift Left Logical
SRL	Shift Right Logical
SRA	Shift Right Arithmetic
SLLV	Shift Left Logical Variable
SRLV	Shift Right Logical Variable
SRAV	Shift Right Arithmetic Variable

Table 1-8 CPU Instruction Set: Coprocessor Instructions

OpCode	Description
LWCz	Load Word to Coprocessor z
SWCz	Store Word from Coprocessor z
MTCz	Move To Coprocessor z
MFCz	Move From Coprocessor z
CTCz	Move Control to Coprocessor z
CFCz	Move Control From Coprocessor z
COPz	Coprocessor Operation z
BCzT	Branch on Coprocessor z True
BCzF	Branch on Coprocessor z False

Table 1-9 CPU Instruction Set: Special Instructions

OpCode	Description
SYSCALL	System Call
BREAK	Break

Table 1-10 Extensions to the ISA: Load and Store Instructions

OpCode	Description
LD	Load Doubleword
LDL	Load Doubleword Left
LDR	Load Doubleword Right
LL	Load Linked
LLD	Load Linked Doubleword
LWU	Load Word Unsigned
SC	Store Conditional
SCD	Store Conditional Doubleword
SD	Store Doubleword
SDL	Store Doubleword Left
SDR	Store Doubleword Right
SYNC	Sync

Table 1-11 Extensions to the ISA: Arithmetic Instructions (ALU Immediate)

OpCode	Description
DADDI	Doubleword Add Immediate
DADDIU	Doubleword Add Immediate Unsigned

Table 1-12 Extensions to the ISA: Multiply and Divide Instructions

OpCode	Description
DMULT	Doubleword Multiply
DMULTU	Doubleword Multiply Unsigned
DDIV	Doubleword Divide
DDIVU	Doubleword Divide Unsigned

Table 1-13 Extensions to the ISA: Branch Instructions

OpCode	Description
BEQL	Branch on Equal Likely
BNEL	Branch on Not Equal Likely
BLEZL	Branch on Less Than or Equal to Zero Likely
BGTZL	Branch on Greater Than Zero Likely
BLTZL	Branch on Less Than Zero Likely
BGEZL	Branch on Greater Than or Equal to Zero Likely
BLTZALL	Branch on Less Than Zero And Link Likely
BGEZALL	Branch on Greater Than or Equal to Zero And Link Likely
BCzTL	Branch on Coprocessor z True Likely
BCzFL	Branch on Coprocessor z False Likely

Table 1-14 Extensions to the ISA: Arithmetic Instructions (3-operand, R-type)

OpCode	Description
DADD	Doubleword Add
DADDU	Doubleword Add Unsigned
DSUB	Doubleword Subtract
DSUBU	Doubleword Subtract Unsigned

Table 1-15 Extensions to the ISA: Shift Instructions

OpCode	Description
DSLL	Doubleword Shift Left Logical
DSRL	Doubleword Shift Right Logical
DSRA	Doubleword Shift Right Arithmetic
DSLLV	Doubleword Shift Left Logical Variable
DSRLV	Doubleword Shift Right Logical Variable
DSRAV	Doubleword Shift Right Arithmetic Variable
DSLL32	Doubleword Shift Left Logical + 32
DSRL32	Doubleword Shift Right Logical + 32
DSRA32	Doubleword Shift Right Arithmetic + 32

Table 1-16 Extensions to the ISA: Exception Instructions

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

Table 1-17 Extensions to the ISA: Coprocessor Instructions

OpCode	Description
DMFCz	Doubleword Move From Coprocessor z
DMTCz	Doubleword Move To Coprocessor z
LDCz	Load Double Coprocessor z
SDCz	Store Double Coprocessor z

Table 1-18 CP0 Instructions

OpCode	Description
DMFC0	Doubleword Move From CP0
DMTC0	Doubleword Move To CP0
MTC0	Move to CP0
MFC0	Move from CP0
TLBR	Read Indexed TLB Entry
TLBWI	Write Indexed TLB Entry
TLBWR	Write Random TLB Entry
TLBP	Probe TLB for Matching Entry
CACHE	Cache Operation
ERET	Exception Return

Data Formats and Addressing

The R4000 processor uses four data formats: a 64-bit doubleword, a 32-bit word, a 16-bit halfword, and an 8-bit byte. Byte ordering within each of the larger data formats—halfword, word, doubleword—can be configured in either big-endian or little-endian order. Endianness refers to the location of byte 0 within the multi-byte data structure. Figures 1-4 and 1-5 show the ordering of bytes within words and the ordering of words within multiple-word structures for the big-endian and little-endian conventions.

When the R4000 processor is configured as a big-endian system, byte 0 is the most-significant (leftmost) byte, thereby providing compatibility with MC 68000® and IBM 370® conventions. Figure 1-4 shows this configuration.

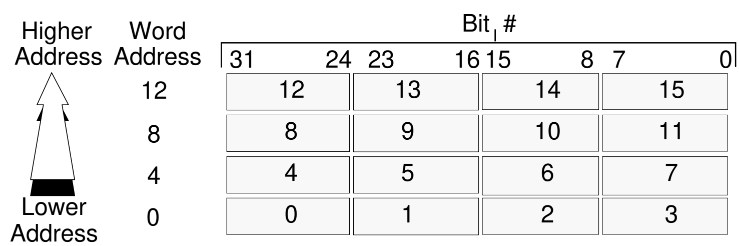

Figure 1-4 Big-Endian Byte Ordering

When configured as a little-endian system, byte 0 is always the least-significant (rightmost) byte, which is compatible with iAPX® x86 and DEC VAX® conventions. Figure 1-5 shows this configuration.

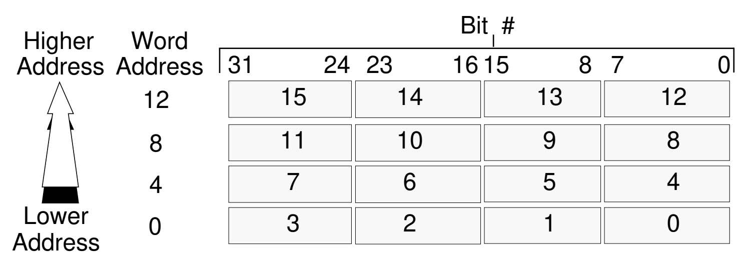

Figure 1-5 Little-Endian Byte Ordering

In this text, bit 0 is always the least-significant (rightmost) bit; thus, bit designations are always little-endian (although no instructions explicitly designate bit positions within words).

Figures 1-6 and 1-7 show little-endian and big-endian byte ordering in doublewords.

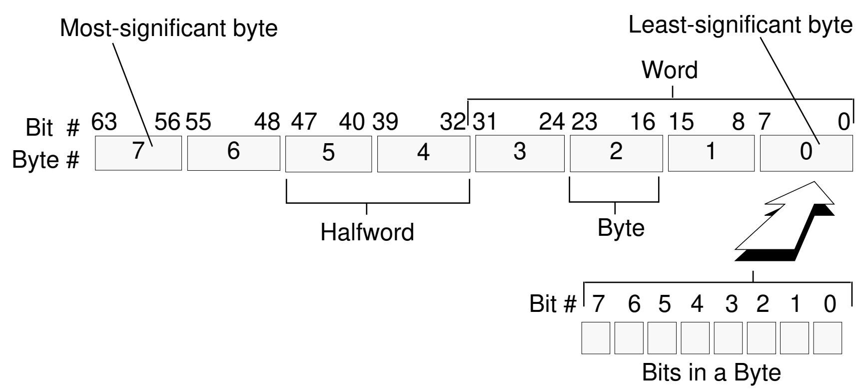

Figure 1-6 Little-Endian Data in a Doubleword

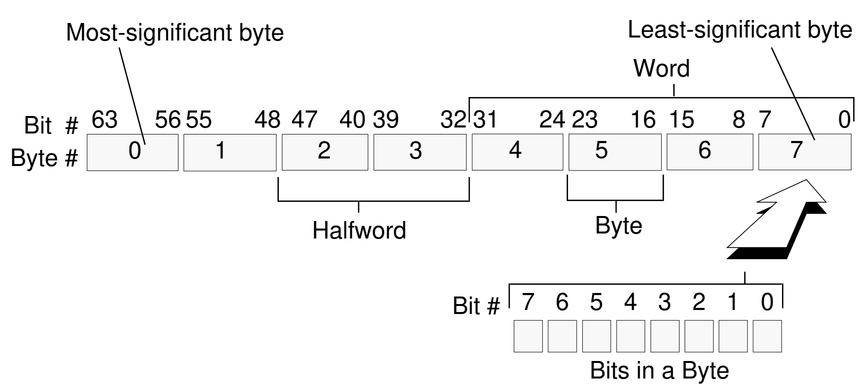

Figure 1-7 Big-Endian Data in a Doubleword

The CPU uses byte addressing for halfword, word, and doubleword accesses with the following alignment constraints:

- Halfword accesses must be aligned on an even byte boundary (0, 2, 4...).
- Word accesses must be aligned on a byte boundary divisible by four (0, 4, 8...).
- Doubleword accesses must be aligned on a byte boundary divisible by eight (0, 8, 16...).

The following special instructions load and store words that are not aligned on 4-byte (word) or 8-word (doubleword) boundaries:

LWL	**LWR**	**SWL**	**SWR**
LDL	**LDR**	**SDL**	**SDR**

These instructions are used in pairs to provide addressing of misaligned words. Addressing misaligned data incurs one additional instruction cycle over that required for addressing aligned data.

Figures 1-8 and 1-9 show the access of a misaligned word that has byte address 3.

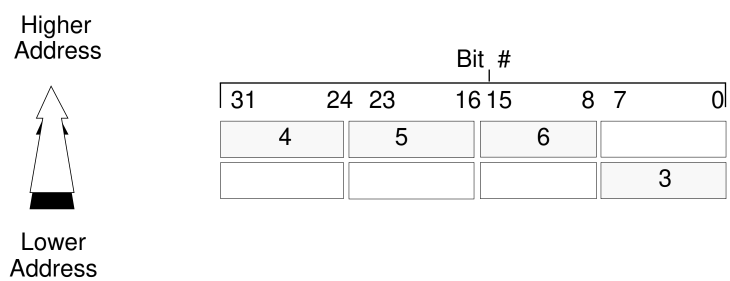

Figure 1-8 Big-Endian Misaligned Word Addressing

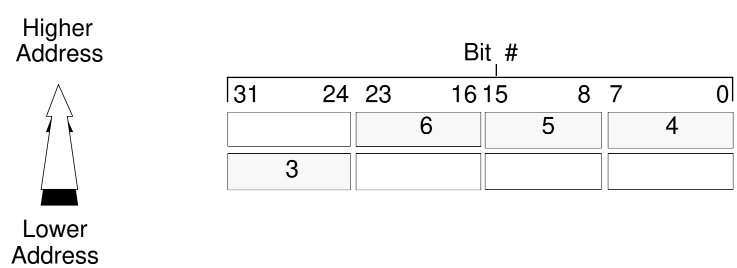

Figure 1-9 Little-Endian Misaligned Word Addressing

Coprocessors (CP0-CP2)

The MIPS ISA defines three coprocessors (designated CP0 through CP2):

- Coprocessor 0 (**CP0**) is incorporated on the CPU chip and supports the virtual memory system and exception handling. CP0 is also referred to as the *System Control Coprocessor*.
- Coprocessor 1 (**CP1**) is reserved for the on-chip, floating-point coprocessor, the FPU.
- Coprocessor 2 (**CP2**) is reserved for future definition by MIPS.

CP0 and CP1 are described in the sections that follow.

System Control Coprocessor, CP0

CP0 translates virtual addresses into physical addresses and manages exceptions and transitions between kernel, supervisor, and user states. CP0 also controls the cache subsystem, as well as providing diagnostic control and error recovery facilities.

The CP0 registers shown in Figure 1-10 and described in Table 1-19 manipulate the memory management and exception handling capabilities of the CPU.

Register Name	Reg. #	Register Name	Reg. #
Index	*0*	*Config*	*16*
Random	*1*	*LLAddr*	*17*
EntryLo0	*2*	*WatchLo*	*18*
EntryLo1	*3*	*WatchHi*	*19*
Context	*4*	*XContext*	*20*
PageMask	*5*		*21*
Wired	*6*		*22*
	7		*23*
BadVAddr	*8*		*24*
Count	*9*		*25*
EntryHi	*10*	*ECC*	*26*
Compare	*11*	*CacheErr*	*27*
SR	*12*	*TagLo*	*28*
Cause	*13*	*TagHi*	*29*
EPC	*14*	*ErrorEPC*	*30*
PRId	*15*		*31*

Exception Processing *Memory Management* *Reserved*

Figure 1-10 R4000 CP0 Registers

Table 1-19 System Control Coprocessor (CP0) Register Definitions

Number	Register	Description
0	Index	Programmable pointer into TLB array
1	Random	Pseudorandom pointer into TLB array *(read only)*
2	EntryLo0	Low half of TLB entry for even virtual address (VPN)
3	EntryLo1	Low half of TLB entry for odd virtual address (VPN)
4	Context	Pointer to kernel virtual page table entry (PTE) in 32-bit addressing mode
5	PageMask	TLB Page Mask
6	Wired	Number of wired TLB entries
7	—	Reserved
8	BadVAddr	Bad virtual address
9	Count	Timer Count
10	EntryHi	High half of TLB entry
11	Compare	Timer Compare
12	SR	Status register
13	Cause	Cause of last exception
14	EPC	Exception Program Counter
15	PRId	Processor Revision Identifier
16	Config	Configuration register
17	LLAddr	Load Linked Address
18	WatchLo	Memory reference trap address low bits
19	WatchHi	Memory reference trap address high bits
20	XContext	Pointer to kernel virtual PTE table in 64-bit addressing mode
21–25	—	Reserved
26	ECC	Secondary-cache error checking and correcting (ECC) and Primary parity
27	CacheErr	Cache Error and Status register
28	TagLo	Cache Tag register
29	TagHi	Cache Tag register
30	ErrorEPC	Error Exception Program Counter
31	—	Reserved

Floating-Point Unit (FPU), CP1

The MIPS floating-point unit (FPU) is designated CP1; the FPU extends the CPU instruction set to perform arithmetic operations on floating-point values. The FPU, with associated system software, fully conforms to the requirements of ANSI/IEEE Standard 754–1985, *IEEE Standard for Binary Floating-Point Arithmetic*.

The FPU features include:

- **Full 64-bit Operation**. The FPU can contain either 16 or 32 64-bit registers to hold single-precision or double-precision values. The FPU also includes a 32-bit *Status/Control* register that provides access to all IEEE-Standard exception handling capabilities.
- **Load and Store Instruction Set**. Like the CPU, the FPU uses a load- and store-based instruction set. Floating-point operations are started in a single cycle and their execution overlaps other fixed-point or floating-point operations.
- **Tightly-coupled Coprocessor Interface**. The FPU is on the CPU chip, and appears to the programmer as a simple extension of the CPU (accessed as CP1). Together, the CPU and FPU form a tightly-coupled unit with a seamless integration of floating-point and fixed-point instruction sets. Since each unit receives and executes instructions in parallel, some floating-point instructions can execute at the same rate (two instructions per cycle) as fixed-point instructions.

Memory Management System (MMU)

The R4000 processor has a 36-bit physical addressing range of 64 Gbytes. However, since it is rare for systems to implement a physical memory space this large, the CPU provides a logical expansion of memory space by translating addresses composed in the large virtual address space into available physical memory addresses. The R4000 processor supports the following two addressing modes:

- 32-bit mode, in which the virtual address space is divided into 2 Gbytes per user process and 2 Gbytes for the kernel.
- 64-bit mode, in which the virtual address is expanded to 1 Tbyte (2^{40} bytes) of user virtual address space.

A detailed description of these address spaces is given in Chapter 4.

The Translation Lookaside Buffer (TLB)

Virtual memory mapping is assisted by a translation lookaside buffer, which caches virtual-to-physical address translations. This fully-associative, on-chip TLB contains 48 entries, each of which maps a pair of variable-sized pages ranging from 4 Kbytes to 16 Mbytes, in multiples of four.

Instruction TLB

The R4000 processor has a two-entry instruction TLB (ITLB) which assists in instruction address translation. The ITLB is completely invisible to software and exists only to increase performance.

Joint TLB

An address translation value is tagged with the most-significant bits of its virtual address (the number of these bits depends upon the size of the page) and a per-process identifier. If there is no matching entry in the TLB, an exception is taken and software refills the on-chip TLB from a page table resident in memory; this TLB is referred to as the joint TLB (JTLB) because it contains both data and instructions jointly. The JTLB entry to be rewritten is selected at random.

Operating Modes

The R4000 processor has three operating modes:

- User mode
- Supervisor mode
- Kernel mode

The manner in which memory addresses are translated or *mapped* depends on the operating mode of the CPU; this is described in Chapter 4.

Cache Memory Hierarchy

To achieve a high performance in uniprocessor and multiprocessor systems, the R4000 processor supports a two-level cache memory hierarchy that increases memory access bandwidth and reduces the latency of load and store instructions. This hierarchy consists of on-chip instruction and data caches, together with an optional external secondary cache that varies in size from 128 Kbytes to 4 Mbytes.

The secondary cache is assumed to consist of one bank of industry-standard static RAM (SRAM) with output enables, arranged as a quadword (128-bit) data array, with a 25-bit-wide tag array. Check fields are added to both data and tag arrays to improve data integrity.

The secondary cache can be configured as a joint cache, or split into separate instruction and data caches. The maximum secondary cache size is 4 Mbytes; the minimum secondary cache size is 128 Kbytes for a joint cache, or 256 Kbytes total for split instruction/data caches. The secondary cache is direct mapped, and is addressed with the lower part of the physical address.

Primary and secondary caches are described in more detail in Chapter 11.

Primary Caches

The R4000 processor incorporates separate on-chip primary instruction and data caches to fill the high-performance pipeline. Each cache has its own 64-bit data path, and each can be accessed in parallel.

The R4000 processor primary caches hold from 8 Kbytes to 32 Kbytes; the R4400 processor primary caches are fixed at 16 Kbytes.

Cache accesses can occur up to twice each cycle. This provides the integer and floating-point units with an aggregate bandwidth of 1.6 Gbytes per second at a MasterClock frequency of 50 MHz.

Secondary Cache Interface

The R4000SC (secondary cache) and R4000MC (multiprocessor) versions of the processor allow connection to an optional secondary cache. These processors provide all of the secondary cache control circuitry, including error checking and correcting (ECC) protection, on chip.

The Secondary Cache interface includes:

- a 128-bit data bus
- a 25-bit tag bus
- an 18-bit address bus
- SRAM control signals

The 128-bit-wide data bus is designed to minimize cache miss penalties, and allow the use of standard low-cost SRAM in secondary cache.