Chapter

1 RISCs and MIPS Architectures

MIPS is the most elegant among the effective RISC architectures; even the competition thought so, as evidenced by the strong MIPS influence to be seen in later architectures like DEC's Alpha and HP's Precision. Elegance by itself doesn't get you far in a competitive marketplace, but MIPS microprocessors have generally managed to be among the most efficient of each generation by remaining among the simplest.

Relative simplicity was a commercial necessity for MIPS Computer Systems Inc., which spun off in 1985 from an academic project to make and market the chips. As a result, the architecture had (and perhaps still has) the largest range of active manufacturers in the industry—working from ASIC cores (MIPS Technologies, Philips) through low-cost CPUs (IDT, AMD/Alchemy) to the only 64-bit CPUs in widespread embedded use (PMC-Sierra, Toshiba, Broadcom).

At the low end the CPU has practically disappeared from sight in the "system on a chip"; at the high end Intrinsity's remarkable processor ran at 2 GHz—a speed unmatched outside the unlimited power/heat budget of contemporary PCs.

ARM gets more headlines, but MIPS sales volumes remain healthy enough: 100 M MIPS CPUs were shipped in 2004 into embedded applications.

The MIPS CPU is one of the RISC CPUs, born out of a particularly fertile period of academic research and development. RISC (Reduced Instruction Set Computing) is an attractive acronym that, like many such, probably obscures reality more than it reveals it. But it does serve as a useful tag for a number of new CPU architectures launched between 1986 and 1989 that owe their remarkable performance to ideas developed a few years earlier in a couple of seminal research projects. Someone commented that "a RISC is any computer architecture defined after 1984"; although meant as a jibe at the industry's use

of the acronym, the comment is also true for a technical reason—no computer defined after 1984 can afford to ignore the RISC pioneers' work.

One of these pioneering projects was the MIPS project at Stanford. The project name MIPS (named for the key phrase "microcomputer without interlocked pipeline stages") is also a pun on the familiar unit "millions of instructions per second." The Stanford group's work showed that pipelining, although a well-known technique, had been drastically underexploited by earlier architectures and could be much better used, particularly when combined with 1980 silicon design.

1.1 Pipelines

Once upon a time in a small town in the north of England, there was Evie's fish and chip shop. Inside, each customer got to the head of the queue and asked for his or her meal (usually fried cod, chips, mushy peas,[1] and a cup of tea). Then each customer waited for the plate to be filled before going to sit down.

Evie's chips were the best in town, and every market day the lunch queue stretched out of the shop. So when the clog shop next door shut down, Evie rented it and doubled the number of tables. But they couldn't fill them! The queue outside was as long as ever, and the busy townsfolk had no time to sit over their cooling tea.

They couldn't add another serving counter; Evie's cod and Bert's chips were what made the shop. But then they had a brilliant idea. They lengthened the counter and Evie, Bert, Dionysus, and Mary stood in a row. As customers came in, Evie gave them a plate with their fish, Bert added the chips, Dionysus spooned out the mushy peas, and Mary poured the tea and took the money. The customers kept walking; as one customer got the peas, the next was already getting chips and the one after that fish. Less hardy folk don't eat mushy peas—but that's no problem; those customers just got nothing but a vacant smile from Dionysus.

The queue shortened and soon they bought the shop on the other side as well for extra table space.

That's a pipeline. Divide any repetitive job into a number of sequential parts and arrange them so that the work moves past the workers, with each specialist doing his or her part for each unit of work in turn. Although the total time any customer spends being served has gone up, there are four customers being served at once and about three times as many customers being served in that market day lunch hour. Figure 1.1 shows Evie's organization, as drawn by her son Einstein in a rare visit to nonvirtual reality.[2]

Seen as a collection of instructions in memory, a program ready to run doesn't look much like a queue of customers. But when you look at it from

1. Non-English readers should probably not inquire further into the nature of this delicacy.

2. It looks to me as if Einstein has been reading books on computer science.

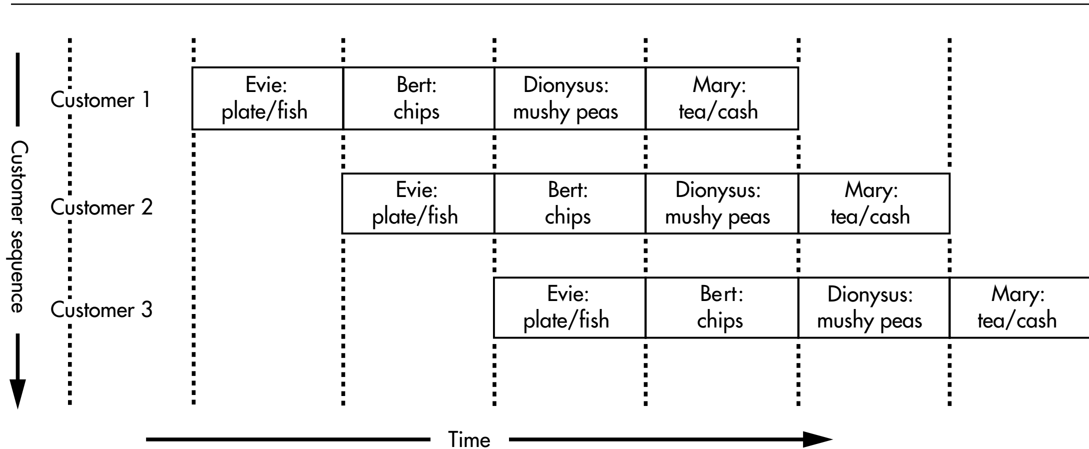

FIGURE 1.1 Evie's fish and chip shop pipeline.

the CPU's point of view, things change. The CPU fetches each instruction from memory, decodes it, finds any operands it needs, performs the appropriate action, and stores any results—and then it goes and does the same thing all over again. The program waiting to be run is a queue of instructions waiting to flow through the CPU one at a time.

The various different jobs required to deal with each instruction already require different specialized chunks of logic inside the CPU, so building a pipeline doesn't even make the CPU much more complicated; it just makes it work harder.

The use of pipelining is not new with RISC microprocessors. What makes the difference is the redesign of everything—starting with the instruction set—to make the pipeline more efficient.[3] So how do you make a pipeline efficient? Actually, that's probably the wrong question. The right question is this: What makes a pipeline inefficient?

1.1.1 *What Makes a Pipeline Inefficient?*

It's not good if one stage takes much longer than the others. The organization of Evie's shop depends on Mary's ability to pour tea with one hand while giving change with the other—if Mary takes longer than the others, the whole queue will have to slow down to match her.

3. The first RISC in this sense was probably the CDC6600, designed by Seymour Cray in the 1970s, but the idea didn't catch on at that time. However, this is straying into the history of computer architecture, and if you like this subject you'll surely want to read Hennessy and Patterson, 1996.

In a pipeline, you try to make sure that every stage takes roughly the same amount of time. A circuit design often gives you the opportunity to trade off the complexity of logic; against its speed, and designers can assign work to different stages: with care, the pipeline is balanced.

The hard problem is not difficult actions, it's awkward customers. Back in the chip shop Cyril is often short of cash, so Evie won't serve him until Mary has counted his money. When Cyril arrives, he's stuck at Evie's position until Mary has finished with the three previous customers and can check his pile of old bent coins. Cyril is trouble, because when he comes in he needs a resource (Mary's counting) that is being used by previous customers. He's a *resource conflict*.

Daphne and Lola always come in together (in that order) and share their meals. Lola won't have chips unless Daphne gets some tea (too salty without something to drink). Lola waits on tenterhooks in front of Bert until Daphne gets to Mary, and so a gap appears in the pipeline. This is a *dependency* (and the gap is called a *pipeline bubble*).

Not all dependencies are a problem. Frank always wants exactly the same meal as Fred, but he can follow him down the counter anyway—if Fred gets chips, Frank gets chips.

If you could get rid of awkward customers, you could make a more efficient pipeline. This is hardly an option for Evie, who has to make her living in a town of eccentrics. Intel is faced with much the same problem: The appeal of its CPUs relies on the customer being able to go on running all that old software. But with a new CPU you get to define the instruction set, and you can define many of the awkward customers out of existence. In section 1.2 we'll show how MIPS did that, but first we'll come back to computer hardware in general with a discussion of caching.

1.1.2 *The Pipeline and Caching*

We said earlier that efficient pipeline operation requires every stage to take the same amount of time. But a 2006 CPU can add two 64-bit numbers 50 to 100 times quicker than it can fetch a piece of data from memory.

So effective pipelining relies on another technique to speed most memory accesses by a factor of 50—the use of *caches*. A cache is a small, very fast, local memory that holds copies of memory data. Each piece of data is kept with a record of its main memory address (the *cache tag*) and when the CPU wants data the cache gets searched and, if the requisite data is available, it's sent back quickly. Since we've no way to guess what data the CPU might be about to use, the cache merely keeps copies of data the CPU has had to fetch from main memory in the recent past; data is discarded from the cache when its space is needed for more data arriving from memory.

Even a simple cache will provide the data the CPU wants more than 90 percentage of the time, so the pipeline design needs only to allow enough time to

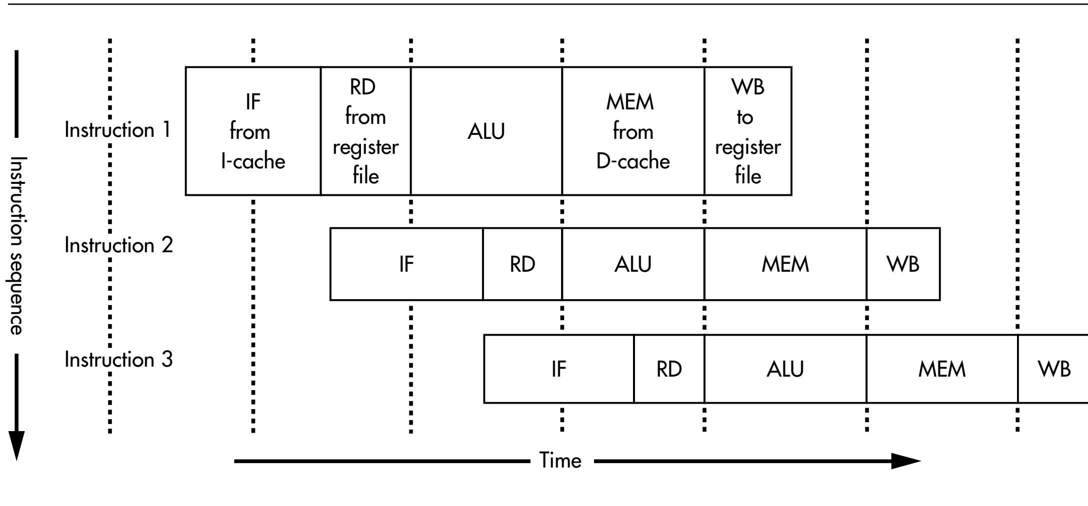

FIGURE 1.2 MIPS five-stage pipeline.

fetch data from the cache; a cache miss is a relatively rare event and we can just stop the CPU when it happens (though cleverer CPUs find more useful things to do).

The MIPS architecture was planned with separate instruction and data caches, so it can fetch an instruction and read or write a memory variable simultaneously.

CISC architectures have caches too, but they're most often afterthoughts, fitted in as a feature of the memory system. A RISC architecture makes more sense if you regard the caches as very much part of the CPU and tied firmly into the pipeline.

1.2 The MIPS Five-Stage Pipeline

The MIPS architecture is made for pipelining, and Figure 1.2 is close to the earliest MIPS CPUs and typical of many. So long as the CPU runs from the cache, the execution of every MIPS instruction is divided into five phases, called *pipestages*, with each pipestage taking a fixed amount of time. The fixed amount of time is usually a processor clock cycle (though some actions take only half a clock, so the MIPS five-stage pipeline actually occupies only four clock cycles).

All instructions are rigidly defined so they can follow the same sequence of pipestages, even where the instruction does nothing at some stage. The net result is that, so long as it keeps hitting the cache, the CPU starts an instruction every clock cycle.

Let's look at Figure 1.2 and consider what happens in each pipestage.

IF (instruction fetch) Gets the next instruction from the instruction cache (*I-cache*).

RD (read registers) Fetches the contents of the CPU registers whose numbers are in the two possible source register fields of the instruction.

ALU (arithmetic/logic unit) Performs an arithmetical or logical operation in one clock cycle (floating-point math and integer multiply/divide can't be done in one clock cycle and are done differently, but that comes later).

MEM Is the stage where the instruction can read/write memory variables in the data cache (*D-cache*). On average, about three out of four instructions do nothing in this stage, but allocating the stage for each instruction ensures that you never get two instructions wanting the data cache at the same time. (It's the same as the mushy peas served by Dionysus.)

WB (write back) Stores the value obtained from an operation back to the register file.

You may have seen other pictures of the MIPS pipeline that look slightly different; it has been common practice to simplify the picture by drawing each pipestage as if it takes exactly one clock cycle. Some later MIPS CPUs have longer or slightly different pipelines, but the pipeline with five stages in four cycles is where the architecture started, and something very like it is still used by the simpler MIPS CPUs.

The tyranny of the rigid pipeline limits the kinds of things instructions can do. First, it forces all instructions to be the same length (exactly one machine word of 32 bits), so that they can be fetched in a constant time. This itself discourages complexity; there are not enough bits in the instruction to encode really complicated addressing modes, for example. And the fixed-size instructions directly cause one problem; in a typical program built for an architecture like x86, the average size of instructions is only just over three bytes. MIPS code will use more memory space.

Second, the pipeline design rules out the implementation of instructions that do any operation on memory variables. Data from cache or memory is obtained only in phase 4, which is much too late to be available to the ALU. Memory accesses occur only as simple load or store instructions that move the data to or from registers (you will see this described as a *load/store architecture*).

The RISC CPUs launched around 1987 worked because the instruction sets designed around those restrictions prove just as useful (particularly for compiled code) as the complicated ones that give so much more trouble to the

hardware. A 1987 or later RISC is characterized by an instruction set designed for efficient pipelining and the use of caches.

The MIPS project architects also attended to the best thinking of the time about what makes a CPU an easy target for efficient optimizing compilers. Many of those requirements are quite compatible with the pipeline requirements, so MIPS CPUs have 32 general-purpose registers and three-operand arithmetical/logical instructions. Happily, the complicated special-purpose instructions that particularly upset pipelines are often those that compilers are unwilling to generate.

The RISC pioneers' judgment has stood the test of time. More recent instruction sets have pushed the hardware/software line back even further; they are called VLIW (very long instruction word) and/or EPIC (explicitly parallel instruction computing). The most prominent is Intel's IA64 architecture, but it has not succeeded despite massive investment; it appears to have got the hardware/software boundary wrong.

1.3 RISC and CISC

We can now have a go at defining what we mean by these overused terms. For me, RISC is an adjective applied to machine architectures/instruction sets. In the mid-1980s, it became attached to a group of relatively new architectures in which the instruction set had been cunningly and effectively specified to make pipelined implementations efficient and successful. It's a useful term because of the great similarity of approach apparent in SPARC, MIPS, PowerPC, HP Precision, DEC Alpha, and (to a lesser extent) in ARM.

By contrast to this rather finely aimed description, CISC (Complex Instruction Set Computing) is used negatively to describe architectures whose definition has not been shaped by those insights about pipelined implementations. The RISC revolution was so successful that no post-1985 architecture has abandoned the basic RISC principles;[4] thus, CISC architectures are inevitably those born before 1985. In this book you can reasonably assume that something said about CISC is being said to apply to both Intel's x86 family and Motorola's 680x0.

Both terms are corrupted when they are applied not to instruction sets but to implementations. It's certainly true that Intel accelerated the performance of its far-from-RISC x86 family by applying implementation tricks pioneered by RISC builders. But to describe these implementations as having a RISC core is misleading.

4. Even Intel's complex and innovation-packed IA64 shares some RISC pipeline-friendly features. But the adjective EPIC—as used by Intel—nicely captures both their boundless ambition and the possibility of a huge flop.

1.4 Great MIPS Chips of the Past and Present

It's time to take a tour through the evolution of MIPS processors and the systems that use them, over the span of the past 20 years or so. We'll look at events in the order they occurred, roughly speaking, with a few scenic detours. Along the way, we'll see that although the MIPS architecture was originally devised with UNIX workstations in mind, it has since found its way into all sorts of other applications—many of which could hardly have been foreseen during the early years. You'll get to know some of these names much better in the chapters that follow.

And although much has happened to the instruction set as well as the silicon, the user-level software from a 1985 R2000 would run perfectly well and quite efficiently on any modern MIPS CPU. That's possibly the best backward-compatibility achievement of any popular architecture.

1.4.1 *R2000 to R3000 Processors*

MIPS Becomes a Corporation

MIPS Computer Systems Inc. was formed in 1984 to commercialize the work of Stanford University's MIPS CPU group; we'll abbreviate the name to "MIPS Inc." Stanford MIPS was only one of several U.S. academic projects that were bringing together chip design, compiler optimization, and computer architecture in novel ways with great success. The commercial MIPS CPU was enhanced with memory management hardware and first appeared late in 1985 as the R2000.

Chip fabrication plants were very expensive to set up even during the mid-1980s; they were certainly beyond the means of a small start-up company. MIPS got its designs into production by licensing them to existing semiconductor vendors who'd already committed the sizable investments required. Early licensees included Integrated Device Technology (IDT), LSI Logic, Performance Semiconductor, and NEC.

An ambitious external math coprocessor chip (the R2010 floating-point accelerator, or FPU) first shipped in mid-1987. Since MIPS was intended to serve the vigorous market for engineering workstations, good floating-point performance was important, and the R2010 delivered it.

MIPS itself bought some of the devices produced by those vendors, incorporating them into its own small servers and workstations. The vendors were free under their licensing agreements to supply the devices to other customers.

The R3000 Processor

First shipped in 1988–1989, this took advantage of a more advanced manufacturing process along with some well-judged hardware enhancements, which

combined to give a substantial boost to performance. From the programmer's point of view, the R3000 was almost indistinguishable from the R2000, which meant the speed of this new design could be unleashed immediately on the rapidly growing base of MIPS software. It was soon joined by the R3010 FPU—a similarly improved version of its predecessor.

By the beginning of the 1990s, a few pioneers were using the R3000 in embedded applications, beginning with high-performance laser printers and typesetting equipment.

The R2000/R3000 chips include cache controllers—to get a cache, just add commodity static RAMs. The FPU shared the cache buses to read instructions (in parallel with the integer CPU) and to transfer operands and results. At 1986 speeds, this division of function was ingenious, practical, and workable; importantly, it held the number of signal connections on each device within the pin count limitations of the pin-grid array packages commonly used at the time. This made it possible to produce the devices at reasonable cost and also to assemble them into systems using existing manufacturing equipment.

The Challenges of Raising the Clock Rate

Although it made good sense at the time of its introduction, difficulties eventually arose centering on the partitioning of functions among the R3000, the R3010 FPU, and the external caches.

First, the R3000's external cache implementation led indirectly to some tricky problems for system designers. To squeeze as much performance as possible from the external cache RAMs, their control signals had to be switched at very short, very precisely defined time intervals. The responsibility for implementing the precision delays was passed along to the system designer: the R3000 required four externally generated copies of the input clock, separated by phase shifts that defined the time intervals essential to correct management of the cache control signals. At 20 MHz that was manageable, but as clock speeds rose through 30 MHz and above, the relentless tightening of the accuracy requirements made the task much harder.

Second, the pressure to increase system clock rates also led to problems for the RAM vendors: To keep pace with shrinking cycle times at the processor pipeline, they had to find ways to achieve corresponding improvements in the access time of the memory devices.

All these difficulties became increasingly apparent as the 1980s drew to a close and limited the designs of this generation to a modest rate of improvement. Starting at 25 MHz in 1988, R3000 systems eventually reached 40 MHz in 1991—and they weren't going any faster.

1.4.2 *The R6000 Processor: A Diversion*

The late 1980s saw lively debates among processor designers about the best way to increase microprocessor clock rates. Two subjects in particular came

to the fore. First: Would it be better for future processor designs to keep the cache implementation external, or to bring the caches on-chip? Second: Which logic technology would be the most advantageous choice for future designs?

The first-generation RISC CPUs were built using CMOS chips. They ran cool and packed a lot of logic into a small space, and all low-cost (pre-RISC) microprocessors used CMOS. CMOS proponents thought they had an advantage for many years to come. Yes, CMOS logic was not the fastest, but that would get fixed—the necessary investment would certainly be forthcoming from companies like Intel. And they argued that CMOS would get even better at the things it already did well—packing even more logic into a given silicon area and switching at even higher frequency within a given power budget.

Other designers knew how compelling speed was for CPUs, and they concluded that high-end processors would be better off using ECL chips like those that were already used for mainframe and supercomputer CPUs. Simple ECL logic gates were faster, and it was much faster at sending signals between chips. But you got less logic into a single chip, and it ran much hotter.

Since the two technologies faced such different challenges, it was very difficult to predict which one was the more likely to emerge as the eventual winner. Among the ECL champions was Bipolar Integrated Technology (BIT), and in 1988 it started work on a MIPS CPU called R6000. The project was ambitious, and BIT hoped to redefine the performance of "super-minicomputers" in the same way that CMOS RISC microprocessors had redefined workstation performance.

There were problems. Because of ECL's density limitations, the processor design had to be partitioned into multiple devices. And customers were anxious about a complete shift to ECL's chip-to-chip signaling standards. BIT built BiCMOS hybrids that sought to mix the best of both worlds.

In the end, the problems overwhelmed the project. The R6000 was delayed by one problem after another, and slipped to be later than the R4000: the first of a new generation of CMOS processors that used their greater density to move the caches on-chip, gaining clock rate by a different route.

BiCMOS CPUs didn't die along with BIT: A few years later, a company named Exponential Technology made another valiant attempt, creating a PowerPC implementation around 1996 that achieved a very impressive clock rate for its time of over 500 MHz. Like BIT, however, the company was eventually thwarted by a combination of technical and contractual difficulties and went out of business.

In a really big argument, both sides are often wrong. In the end, several more years were to pass before on-chip implementation of the caches became essential to achieving the highest clock rate. Hewlett Packard stuck with CMOS chips and a (large) external primary cache for its rather MIPS-like Precision architecture. HP eventually pushed its clock rate to around 120 MHz—three times the fastest R3000—*without* using ECL or BiCMOS. HP was its own

customer for these processors, using them in its own workstations; the company felt this market was best served by an evolutionary approach and could bear the costs of high pin-count packages and careful high-speed system-level design. This strategy put HP at the top of the performance stakes for a long, long time; the winner is not always the most revolutionary.

1.4.3 *The First CPU Cores*

In the early 1980s, LSI Logic pioneered the idea of adapting high-volume chip design and manufacturing techniques so that systems companies could create devices specifically tailored to the needs of their own products. Those chips were called Application-Specific Integrated Circuits (ASICs); by around 1990, they could contain up to several thousand gates, equivalent to a large board full of 1970s-era logic devices. The unit cost was very low, and development costs were manageable.

We've seen already that LSI took a very early interest in MIPS and made some R2000/R3000 chips. A couple of years later, it was a natural move for the company to create an implementation of the MIPS architecture that used its own in-house ASIC technology; that move opened the door for customers to include a MIPS processor within a chip that also incorporated other logic. Other MIPS licensees, such as IDT, also began to offer products that integrated simple peripheral functions alongside a MIPS CPU.

Even at the very beginning of the 1990s, you could easily put the basic logic of an R3000-class CPU on an ASIC; but ASICs didn't have very efficient RAM blocks, so integrating the caches was a problem. But ASIC technology progressed rapidly, and by 1993 it was becoming realistic to think of implementing an entire microprocessor system on a chip—not just the CPU and caches, but also the memory controllers, the interface controllers, and any small miscellaneous blocks of supporting logic.

The ASIC business depended on customers being able to take a design into production in a relatively short time—much less than that needed to create a chip using "custom" methods. While it was obviously attractive to offer customers the idea of integrating a complete system on a chip, ASIC vendors had to strike a balance: How could the inevitable increase in complexity still be accommodated within the design cycles that customers had come to expect?

The ASIC industry's answer was to offer useful functional elements—such as an entire MIPS processor—in the form of cores: ready-made building blocks that conveniently encapsulated all the necessary internal design work and verification, typically presented as a set of machine-readable files in the formats accepted by ASIC design software. Systems designs of the future would be created by connecting several ASIC cores together on a chip; in comparison with existing systems—created by connecting together devices on a circuit board—the new systems implemented as core-based ASICs would be smaller, faster, and cheaper.

Until this time, ASIC designers had naturally thought in terms of combining fairly small logic blocks—state machines, counters, decoders, and so forth. With the advent of ASIC cores, designers were invited to work with a broader brush on a much larger canvas, bringing together processors, RAMs, memory controllers, and on-chip buses.

If you suspect that it can't have been that easy, you have good instincts. It sounded compelling—but in practice, creating cores and connecting them together both turned out to be very difficult things to do well. Nevertheless, these early ASIC cores are of great historical significance; they're the direct ancestors of the system-on-a-chip (SoC) designs that have become pervasive during the early 2000s. We'll take up the SoC story again a bit later on, after we've followed several threads of MIPS development through the 1990s.

1.4.4 *The R4000 Processor: A Revolution*

The R4000, introduced in 1991, was a brave and ground-breaking development. Pioneering features included a complete 64-bit instruction set, the largest possible on-chip caches (dual 8 KB), clock rates that seemed then like science fiction (100 MHz on launch), an on-chip secondary cache controller, a system interface running at a fraction of the internal CPU clock, and on-chip support for a shared-memory multiprocessor system. The R4000 was among the first devices to adopt a number of the engineering developments that were to become common by around 1995, though it's important to note that it didn't take on the complexity of superscalar execution.

The R4000 wasn't perfect. It was an ambitious chip and the design was hard to test, especially the clever tricks used for multiprocessor support. Compared with the R3000, it needs more clock cycles to execute a given instruction sequence—those clock cycles are so much shorter that it ends up well in front, but you don't like to give performance away. To win on clock speed the primary caches are on-chip: To keep the cost of each device reasonable, the size of the caches had to be kept relatively small. The R4000 has a longer pipeline, mainly to spread the cache access across multiple clock cycles. Longer pipelines are less efficient, losing time when the pipeline is drained by a branch.

1.4.5 *The Rise and Fall of the ACE Consortium*

Around the time of the R4000's introduction, MIPS had high hopes that the new design would help it to become an important participant in the market for workstations, desktop systems, and servers.

This wasn't mere wishful thinking on the part of MIPS. During the early 1990s, many observers predicted that RISC processors would take an increasing portion of the market away from their CISC competitors; the bolder

prognosticators even suggested that the CISC families would die away entirely within a few years.

In 1991, a group of about 20 companies came together to form a consortium named the Advanced Computing Environment (ACE) initiative. The group included DEC (minicomputers), Compaq (PCs), Microsoft, and SCO (then responsible for UNIX System V). ACE's goal was to define specifications and standards to let future UNIX or Windows software drop straight onto any of a range of machines powered by either Intel x86 or MIPS CPUs. Even in 1991, a small percentage of the PC business would have meant very attractive sales for MIPS CPUs and MIPS system builders.

If hype could create a success, ACE would have been a big one. But looking back on it, Microsoft was more interested in proving that its new Windows NT system was portable (and perhaps giving Intel a fright) than in actually breaking up their PC market duopoly. For MIPS, the outcome wasn't so good; chip volumes wouldn't sustain it and its systems business entered into a decline, which before long became so serious that the future of the company was called into question.

1.4.6 *SGI Acquires MIPS*

As 1992 progressed, the hoped-for flock of new ACE-compliant systems based on MIPS processors was proving slow to materialize, and DEC—MIPS's highest-profile workstation user—decided that future generations of its systems would instead use its own Alpha processor family.

That left workstation company Silicon Graphics, Inc. (SGI) as by far the leading user of MIPS processors for computer systems. So in early 1993, SGI was the obvious candidate to step in and rescue MIPS Inc., as a way of safeguarding the future of the architecture on which its own business depended. By the end of 1994, late-model R4400 CPUs (a stretched R4000 with bigger caches and performance tuning) were running at 200–250 MHz and keeping SGI in touch with the RISC performance leaders.

1.4.7 *QED: Fast MIPS Processors for Embedded Systems*

Some of MIPS Inc.'s key designers left to start a new company named Quantum Effect Design (QED). The QED founders had been deeply involved in the design of MIPS processors from the R2000 through R4000.

With IDT as a manufacturing partner and early investor, QED's small team set out to create a simple, fast 64-bit MIPS implementation. The plan was to create a processor that would offer good performance for a reasonable selling price, so that the device could find a home in many applications, ranging from low-end workstations, through small servers, to embedded systems like top-of-the-range laser printers and network routers.

There were determined people who'd applied R4000 chips to embedded systems, but it was a fight. QED made sure that the R4600 was much more appealing to embedded systems designers, and the device soon became a success. It went back to a simple five-stage pipeline and offered very competitive performance for a reasonable price. Winning a place in Cisco routers as well as SGI's Indy desktops led to another first: The R4600 was the first RISC CPU that plainly turned in a profit.

The QED design team continued to refine its work, creating the R4650 and R4700 during the mid-1990s. We'll take up the QED story again a little further on, when we talk about the R5000.

1.4.8 *The R10000 Processor and Its Successors*

During the mid-1990s, SGI placed very high importance on high-end workstations and supercomputers. Absolute performance was a very important selling point, and the MIPS division was called upon to meet this challenge with its next processor design.

The SGI/MIPS R10000 was launched in early 1996. It was a major departure for MIPS from the traditional simple pipeline; it was the first CPU to make truly heroic use of out-of-order execution, along with multiple instruction issue. Within a few years, out-of-order designs were to sweep all before them, and all really high-end modern CPUs are out-of-order. But the sheer difficulty of verifying and debugging the R10000 convinced both participants and observers to conclude that it had been a mistake for SGI to undertake such an ambitious design in-house.

SGI's workstation business began to suffer during the latter half of the 1990s, leading inevitably to a decline in its ability to make continuing investments in the MIPS architecture. Even as this took place, the market for mainstream PCs continued to expand vigorously, generating very healthy revenue streams to fund the future development of competing architectures—most notably Intel's Pentium family and its descendants and, to a lesser extent, the PowerPC devices designed by Motorola and IBM.

Against this backdrop, SGI started work on MIPS CPUs beyond the R10000; but, because of mounting financial pressures, the projects were canceled before the design teams were able to complete their work. In 1998, SGI publicly committed itself to using the Intel IA-64 architecture in its future workstations, and the last MIPS design team working on desktop/server products was disbanded. In 2006 (as I write) some SGI machines are still dependent on the R16000 CPU; while it takes advantage of advances in process technology to achieve higher clock rates, the internal design has scarcely been enhanced beyond that of the 1996 R10000. Meanwhile, IA-64 CPUs have sold well below Intel's most pessimistic projections, and the fastest CPUs in the world are all variants of the x86. SGI seems to be unlucky when it comes to choosing CPUs!

1.4.9 *MIPS Processors in Consumer Electronics*

LSI Logic and the Sony PlayStation

In 1993, Sony contracted with LSI Logic for the development of the chip that was to form the heart of the first PlayStation. Based on LSI's CW33000 processor core, it was clocked at 33 MHz and incorporated a number of peripheral functions, such as a DRAM controller and DMA engine. The PlayStation's highly integrated design made it cheap to produce and its unprecedented CPU power made for exciting gaming. Sony rapidly overtook more established vendors to become the leading seller of video game consoles.

The Nintendo64 and NEC's Vr4300 Processor

Nintendo game consoles lost considerable market share to Sony's PlayStation. In response, Nintendo formed an alliance with Silicon Graphics and decided to leapfrog 32-bit CPU architectures and go straight for a 64-bit chip—in a $199 games machine.

The chip at its heart—the NEC Vr4300—was a cut-down R4000, but not *that* cut-down. It did have a 32-bit external bus, to fit in a cheaper package with fewer pins, and it shared logic between integer and floating-point maths. But it was a lot of power for a $199 box.

The Vr4300's computing power, low price, and frugal power consumption made it very successful elsewhere, particularly in laser printers, and helped secure another niche for the MIPS architecture in "embedded" applications.

But the Vr4300 was the last really general-purpose CPU to storm the games market; by the late 1990s, the CPU designs intended for this market had become increasingly specialized, tightly coupled with dedicated hardware accelerators for 3D rendering, texture mapping, and video playback. When Sony came back with the PlayStation 2, it had a remarkable 64-bit MIPS CPU at its heart. Built by Toshiba, it features a floating-point coprocessor whose throughput wouldn't have disgraced a 1988 supercomputer (though its accuracy would have been a problem). It has proven too specialized to find applications outside the games market, but a version of the same CPU is in Sony's PSP handheld games console, which will certainly be with us for a few years to come.

Cumulative sales of these video game consoles worldwide is well into the tens of millions, accounting for a larger volume of MIPS processors than any other application—and also causing them to outsell a good many other CPU architectures.

1.4.10 *MIPS in Network Routers and Laser Printers*

The R5000 Processor

Following the success of the R4600 and its derivatives, QED's next major design was the R5000. Launched in 1995—the same year as SGI's R10000—this

was also a superscalar implementation, though in terms of general design philosophy and complexity, the two designs stood in stark contrast to each other.

The R5000 used the classic five-stage pipeline and issued instructions in-order. It was capable, however, of issuing one integer instruction and one floating-point instruction alongside each other. The MIPS architecture makes this scheme relatively easy to implement; the two instruction categories use separate register sets and execution units, so the logic needed to recognize opportunities for dual issue doesn't have to be very complicated.

Of course, the other side of the same coin is that the performance gain is relatively modest. Unless the R5000 is used in a system that runs a significant amount of floating-point computation, the superscalar ability goes unused. Even so, the R5000 incorporated other improvements that made it appealing to system designers as an easy upgrade from the R4600 generation.

QED Becomes a Fabless Semiconductor Vendor

During the first few years of its life, QED had operated purely as a seller of intellectual property, licensing its designs to semiconductor device vendors who then produced and sold the chips. In 1996, the company decided it could do better by selling chips under its own name. The manufacturing was still carried out by outside partners—the company remained "fabless" (that is, it didn't have any fabrication plants under its direct ownership)—but now QED took charge of testing the chips and handled all of its own sales, marketing, and technical support.

Around this time, QED embarked on a project to develop a PowerPC implementation in the same lean, efficient style as the R4600. Unfortunately, business and contractual difficulties with the intended customer reared their heads, with the result that the device was never brought to market. After this brief excursion into PowerPC territory, QED resumed its exclusive focus on the MIPS architecture.

QED's RM5200 and RM7000 Processors

QED's first "own-brand" CPU was the RM5200 family, a direct descendant of the R5000. With a 64-bit external bus it played well in network routers, while a 32-bit bus and cheaper package was good for laser printers.

QED built on the RM5200's success, launching the RM7000 in 1998. This device marked several important milestones for MIPS implementations: It was the first to bring the (256 Kbyte) secondary cache on-chip.[5] RM7000 was also a serious superscalar design, which could issue many pairings of integer instructions besides the integer/floating-point combination inherited from the R5000.

5. QED originally hoped to use a DRAM-like memory to make the RM7000's secondary cache very small, but it turned out that an adequate compromise between fast logic and dense DRAM on a single chip was not then possible. It still isn't.

The RM5200 and RM7000 processor families sold well during the mid to late 1990s into many high-end embedded applications, finding especially widespread use in network routers and laser printers. QED wisely ensured that the RM7000 was very closely compatible with the RM5200, both from the programmer's and system designer's points of view. This made it fairly straightforward to give aging RM5200 systems a quick midlife boost by upgrading them to the RM7000, and many customers found it useful to follow this path.

SandCraft

Around 1998, the design team that had created the Vr4300 for Nintendo incorporated as SandCraft, and set out to produce embedded CPUs intended for the high-end embedded applications then served by QED's RM5200 and RM7000 families.

SandCraft's designs were architecturally ambitious and took time to bring to market. Despite several years of continued efforts to build a large enough customer base, the company eventually went out of business. Its assets were acquired by Raza Technologies, and it remains to be seen whether any significant portion of SandCraft's legacy will eventually find its way into production.

1.4.11 *MIPS Processors in Modern Times*

Alchemy Semiconductor: Low-Power MIPS Processors

By 1999, the markets for cellphones, personal organizers, and digital cameras were growing rapidly. The priority for such processors is low power consumption: Since these appliances need to be small, light, and have to run from internal batteries, they must be designed to operate within very tight power budgets. At the same time, competitive pressures require each generation of appliances to offer more features than its predecessor. Manufacturers sought 32-bit computing power to meet the growing applications' hungry demands.

Taken together, these requirements present a moving target for processor designers: Within a power budget that grows only gradually (with advances in battery chemistry and manufacturing), they're called upon to deliver a significant boost in performance with every design generation.

It is really just a matter of historical accident that nobody had implemented a fast, low-power MIPS processor. But DEC had built a 200-MHz low-power ARM ("StrongARM") and the ARM company was willing to build less exalted machines that would eke out your battery life even longer. When DEC engaged in unwise litigation with Intel over CPU patents, they lost, big-time. Among the things Intel picked up was the StrongARM development. Amazingly, it seems

to have taken Intel a couple of years to notice this jewel, and by that time all the developers had left.

In 1999, Alchemy Semiconductor was founded precisely to exploit this opportunity. With backing from Cadence, a vendor of chip design tools, some members of the design team that had created StrongARM now turned their ingenuity to the design of a very low power 32-bit MIPS CPU. It works very well, but their designs were too high-end, and perhaps just a bit late, to break the ARMlock on cellphones.

Alchemy pinned its hopes on the market for personal organizers, which certainly needed faster CPUs than the phones did. But the market didn't boom in the same way. Moreover, the organizer market seemed to be one in which every innovator lost money; and finally Microsoft's hot-then-cold support of MIPS on Windows CE made the MIPS architecture a handicap in this area.

SiByte

This company was also founded in 1999 around an experienced design team, again including some members who had worked on DEC's Alpha and StrongARM projects.[6]

SiByte built a high-performance MIPS CPU design—it aimed for efficient dual-issue at 1 GHz. Moreover, this was to be wrapped up for easy integration into a range of chip-level products; some variants were to emphasize computational capacity, featuring multiple CPU cores, while others laid the stress on flexible interfacing by integrating a number of controllers.

SiByte's design found considerable interest from networking equipment makers who were already using QED's RM5200 and RM7000 devices; as the company began to put the device into production, however, manufacturing difficulties caused long delays, and the 1-GHz target proved difficult.

Consolidation: PMC-Sierra, Broadcom, AMD

The last years of the 1990s saw the infamous "dotcom bubble." Many small technology companies went public and saw their stock prices climb to dizzying heights within weeks.

Networking companies were among the darlings of the stock market and with their market capitalizations rising into tens of billions, they found it easy to buy companies providing useful technology—and that sometimes meant MIPS CPUs.

This was the climate in which Broadcom acquired SiByte, and PMC-Sierra acquired QED—both in mid-2000. It seemed that the future of high-end MIPS

---

6. In the U.S. market, a canceled project can have as seminal an effect as a successful one.

designs for embedded systems was now doubly safeguarded by the deep pockets of these two new parent companies.

The collapse of the technology bubble came swiftly and brutally. By late 2001, the networking companies saw their stock prices showing unexpected weakness, and orders from their customers slowing alarmingly; by 2002, the entire industry found itself in the grip of a savage downturn. Some companies saw their market capitalizations drop to less than a tenth of their peak values over one year.

The resulting downdraft inevitably affected the ability of PMC-Sierra and Broadcom to follow through with their plans for the QED and SiByte processor designs. It wasn't just a matter of money; it became extremely difficult for these companies even to find a reasonable strategic direction, as sales for many established product lines slowed to a trickle.

Alchemy Semiconductor also felt the cold wind of change, and despite the impressively low power consumption of its designs, the company had difficulty finding high-volume customers. Finally, in 2002, Alchemy was acquired by Advanced Micro Devices (AMD), which continued to market the Au1000 product line for a couple of years. As we go to press, we hear that the Alchemy product line has been acquired by Raza Technologies.

Highly Integrated Multiprocessor Devices

Broadcom had initially announced plans for an ambitious evolution of SiByte's 1250 design from two CPU cores to four, along with an extra memory controller and much faster interfaces. This project became a casualty of the downturn, and the evolutionary future of the 1250 product line fell into uncertainty.

Meanwhile, the QED design team—now operating as PMC-Sierra's MIPS processor division—created its own dual-CPU device, the RM9000x2. This also integrated an SDRAM controller and various interfaces. Due in part to the chilly market conditions, the RM9000 family was slow to find customers, though it did surpass the 1-GHz milestone for CPU clock rate. Subsequent derivatives added further interfaces, including Ethernet controllers, but the difficulties in securing large design wins persisted.

In 2006, the future for such highly integrated devices appears doubtful. As the transistors in chips shrink, the amount of logic you can get for the production-cost dollar increases. But the one-off cost of getting a new design into production keeps going up: For the most recent 90-nanometer generation, it's $1 M or more. If you fill the space available with logic to maximize what your chip will do, design and test costs will be tens of millions.

To get that back, you need to sell millions of units of each chip over its product lifetime. A particular version of a chip like the RM9000x2 or Broadcom's 1250 can sell tens or hundreds of thousands: It isn't enough. It's not clear what sort of device may attract enough volume to fund full-custom embedded-CPU design in future.

Intrinsity: Taking a MIPS Processor to 2 GHz

Alert readers will have noticed the overall arc of the MIPS story to date: The early R2000/R3000 implementations were performance leaders among microprocessors, but competing families eventually caught up and overtook MIPS.

So you might be wondering: Has anyone tried to make a really fast MIPS processor in the last few years? The answer is yes: Intrinsity Semiconductor announced in 2002 its FastMath processor. Using careful design techniques to extract high performance from essentially standard process technologies, Intrinsity was able to produce a lean 32-bit MIPS design with an impressive clock rate of 2 GHz.

While this was widely recognized as a fine technical achievement, the device has struggled to find a market. It's still not nearly as fast as a modern PC processor, and its power consumption and heat dissipation is relatively high by consumer standards.

1.4.12 *The Rebirth of MIPS Technologies*

In 1998, SGI—facing mounting cash-flow problems—decided to spin off its CPU design group, restoring it to independence as MIPS Technologies. The new company was chartered to create core CPUs to be used as part of a system-on-a-chip (SoC). You might recall that we encountered the idea of an SoC much earlier in this section, when we described the appearance of the first ASIC cores.

In the early days of SoCs, CPU vendors found that it was very difficult to guarantee a core's performance—for example, the CPU clock rate—unless they provided their customers with a fixed silicon layout for the core internals, predefined for each likely target chip "foundry"—a "hard core."

MIPS Technologies originally intended to build high-performance hard cores and built and shipped fast 64-bit designs (20 Kc and later 25 Kf). But that was the wrong horse. During the last few years, the market has increasingly preferred its cores to be synthesizable (originally called "soft core").

A synthesizable core is a set of design files (usually in Verilog) that describes the circuit and can be compiled into a real hardware design. A synthesizable core product consists of a lot more than a Verilog design, since the customer must be able to incorporate it in a larger SoC design and validate the CPU and its connections well enough that the whole chip will almost certainly work.

MIPS Technologies' first synthesizable core was the modest 32-bit 4-K family; since then, it has added the 64-bit 5 K, the high-performance 32-bit 24 K, and (launched in early 2006) the multithreading 34 K.

1.4.13 *The Present Day*

MIPS CPUs in use today come in four broad categories:

- *SoC cores*: MIPS CPUs still benefit in size and power consumption from the simplicity that came from the Stanford project, and an architecture with a long history spanning handhelds to supercomputers is an attractive alternative to architectures tailored for the low end. MIPS was the first "grown up" CPU to be available as an ASIC core—witness its presence in the Sony PlayStation games console. The most prominent supplier is MIPS Technologies Inc., but Philips retains their own designs.
- *Integrated embedded 32-bit CPUs*: From a few dollars upward, these chips contain CPU, caches, and substantial application-oriented blocks (network controllers are popular). There's considerable variation in price, power consumption, and processing power. Although AMD/Alchemy has some very attractive products, this market seems to be doomed, with devices in the target marketplace finding that an SoC heart does a better job of maximizing integration, saving vital dollars and milliwatts.
- *Integrated embedded 64-bit CPUs*: These chips offer a very attractive speed/power-consumption trade-off for high-end embedded applications: Network routers and laser printers are common applications. But it doesn't look as though they can sell in sufficient numbers to go on paying for chip development costs.

  But somewhere in this category are companies that are trying radically new ideas, and it's a tribute to the MIPS architecture's clean concepts that it often seems the best base for leading-edge exploration. Raza's XLR series of multicore, multithreaded processors represent a different kind of embedded CPU, which aims to add more value (and capture more revenue per unit) than a "traditional" embedded CPU. Cavium's Octium is also pretty exciting.
- *Server processors*: Silicon Graphics, the workstation company that was the adoptive parent of the MIPS architecture, continued to ship high-end MIPS systems right up to its insolvency in 2006, even though that was seven years after it committed to a future with Intel IA-64. But it's the end of the road for these systems: MIPS is destined to be "only" in the vast consumer and embedded markets.

The major distinguishing features of some milestone products are summarized in Table 1.1. We haven't discussed the instruction set revision levels from MIPS I through MIPS64, but there'll be more about them in section 2.7, where you'll also find out what happened to MIPS II.

TABLE 1.1 Milestones in MIPS CPUs

Year	Designer/model/ clock rate (MHz)	Instruction set	Cache (I+D)	Notes
1987	MIPS R2000-16	MIPS I	External: 4 K+4 K to 32 K+32 K	External (R2010) FPU.
1990	IDT R3051-20		4 K+1 K	The first embedded MIPS CPU with on-chip cache and progenitor of a family of pin-compatible parts.
1991	MIPS R4000-100	MIPS III	8 K+8 K	Integrates FPU and L2 cache controller with pinout option. Full 64-bit CPU—but five years later, few MIPS CPUs were exploiting their 64-bit instruction set. Long pipeline and half-speed interface help achieve high clock rates.
1993	IDT/QED R4600-100		16 K+16 K	QED's brilliantly tuned redesign is much faster than R4000 or R4400 at the same clock rate—partly because it returned to the classic MIPS five-stage pipeline. Important to SGI's fast and affordable low-end Indy workstation and Cisco's routers.
1995	NEC/MIPS Vr4300-133		16 K+8 K	Low cost, low power but full-featured R4000 derivative. Initially aimed at Nintendo 64 games console, but embedded uses include HP's LJ4000 laser printers.
1996	MIPS R10000-200	MIPS IV	32 K+32 K	Bristling with microprocessor innovations, the R10000 is not at all simple. The main MIPS tradition it upholds is that of taking a principle to extremes. The result was hot, unfriendly, but with unmatched performance/MHz.
1998	QED RM7000		16 K+16 K+256 K L2	The first MIPS CPU with on-chip L2 cache, this powered generations of high-end laser printers and Internet routers.
2000	MIPS 4 K core family	MIPS32	16 K+16 K (typ)	The most successful MIPS core to date—synthesizable and frugal.
2001	Alchemy AU-1000		16 K+16 K	If you wanted 400 MHz for 500 mW, this was the only show in town. But it lost markets.
2001	Broadcom BCM1250	MIPS64	32 K+32 K+256 K L2	Dual-CPU design at 600 MHz+ (the L2 is shared).
2002	PMC-Sierra RM9000x2	MIPS64	16 K+16 K+256 K L2	Dual-CPU design at 1 GHz (the L2 is NOT shared; each CPU has its own 256 K). First MIPS CPU to reach 1 GHz.
2003	Intrinsity FastMath	MIPS32	16 K+16 K+1 M L2	Awesome 2-GHz CPU with vector DSP did not find a market.
2003	MIPS 24 K core	MIPS32 R2	At 500 MHz in synthesizable logic, a solidly successful core design.	
2005	MIPS 34 K core	MIPS32+MT ASE	32 K+32 K (typ)	MIPS multithreading pioneer.

1.5 MIPS Compared with CISC Architectures

Programmers who have some assembly-language-level knowledge of earlier architectures—particularly those brought up on x86 or 680x0 CISC instruction sets—may get some surprises from the MIPS instruction set and register model. We'll try to summarize them here, so you don't get sidetracked later into doomed searches for things that don't quite exist, like a stack with push/pop instructions!

We'll consider the following: constraints on MIPS operations imposed to make the pipeline efficient; the radically simple load/store operations; possible operations that have been deliberately omitted; unexpected features of the instruction set; and the points where the pipelined operation becomes visible to the programmer.

The Stanford group that originally dreamed up MIPS was paying particular attention to the short, simple pipeline it could afford to build. But it's a testament to the group's judgment that many of the decisions that flowed from that have proven to make more ambitious implementations easier and faster, too.

1.5.1 *Constraints on MIPS Instructions*

- *All instructions are 32 bits long*: That means that no instruction can fit into only two or three bytes of memory (so MIPS binaries are typically 20 percent to 30 percent bigger than for 680x0 or 80x86) and no instruction can be bigger.

  It follows that it is impossible to incorporate a 32-bit constant into a single instruction (there would be no instruction bits left to encode the operation and the target register). The MIPS architects decided to make space for a 26-bit constant to encode the target address of a jump or jump to subroutine; but that's only for a couple of instructions. Other instructions find room only for a 16-bit constant. It follows that loading an arbitrary 32-bit value requires a two-instruction sequence, and conditional branches are limited to a range of 64-K instructions.

- *Instruction actions must fit the pipeline*: Actions can only be carried out in the right pipeline phase and must be complete in one clock. For example, the register write-back phase provides for just one value to be stored in the register file, so instructions can only change one register.

  Integer multiply and divide instructions are too important to leave out but can't be done in one clock. MIPS CPUs have traditionally provided them by dispatching these operations into a separately pipelined unit we'll talk about later.

- *Three-operand instructions*: Arithmetical/logical operations don't have to specify memory locations, so there are plenty of instruction bits to define two independent sources and one destination register. Compilers love

three-operand instructions, which give optimizers much more scope to improve code that handles complex expressions.

- *The 32 registers*: The choice of the number of registers is largely driven by software requirements, and a set of 32 general-purpose registers is easily the most popular in modern architectures. Using 16 would definitely not be as many as modern compilers like, but 32 is enough for a C compiler to keep frequently accessed data in registers in all but the largest and most intricate functions. Using 64 or more registers requires a bigger instruction field to encode registers and also increases context-switch overhead.

- *Register zero*: `$0` always returns zero, to give a compact encoding of that useful constant.

- *No condition codes*: One feature of the MIPS instruction set that is radical even among the 1985 RISCs is the lack of any condition flags. Many architectures have multiple flags for "carry," "zero," and so on. CISC architectures typically set these flags according to the result written by any or a large subset of machine instructions, while some RISC architectures retain flags (though typically they are only set explicitly, by compare instructions).

  The MIPS architects decided to keep all this information in the register file: Compare instructions *set* general-purpose registers and conditional branch instructions *test* general-purpose registers. That does benefit a pipelined implementation, in that whatever clever mechanisms are built in to reduce the effect of dependencies on arithmetical/logical operations will also reduce dependencies in compare/branch pairs.

  We'll see later that efficient conditional branching (at least in one favorite simple pipeline organization) means that the decision about whether to branch or not has to be squeezed into only half a pipeline stage; the architecture helps out by keeping the branch decision tests very simple. So MIPS conditional branches test a single register for sign/zero or a pair of registers for equality.

1.5.2 *Addressing and Memory Accesses*

- *Memory references are always plain register loads and stores*: Arithmetic on memory variables upsets the pipeline, so it is not done. Every memory reference has an explicit load or store instruction. The large register file makes this much less of a problem than it sounds.

- *Only one data-addressing mode*: Almost all loads and stores select the memory location with a single base register value modified by a 16-bit signed displacement (a limited register-plus-register address mode is available for floating-point data).

- *Byte-addressed*: Once data is in a register of a MIPS CPU, all operations always work on the whole register. But the semantics of languages such as C fit badly on a machine that can't address memory locations down to byte granularity, so MIPS gets a complete set of load/store operations for 8- and 16-bit variables (we will say *byte* and *halfword*). Once the data has arrived in a register it will be treated as data of full register length, so partial-word load instructions come in two flavors—sign-extend and zero-extend.
- *Load/stores must be aligned*: Memory operations can only load or store data from addresses aligned to suit the data type being transferred. Bytes can be transferred at any address, but halfwords must be even-aligned and word transfers aligned to four-byte boundaries. Many CISC microprocessors will load/store a four-byte item from any byte address, but the penalty is extra clock cycles.

  However, the MIPS instruction set architecture (ISA) does include a couple of peculiar instructions to simplify the job of loading or storing at improperly aligned addresses.
- *Jump instructions*: The limited 32-bit instruction length is a particular problem for branches in an architecture that wants to support very large programs. The smallest opcode field in a MIPS instruction is 6 bits, leaving 26 bits to define the target of a jump. Since all instructions are four-byte aligned in memory, the two least significant address bits need not be stored, allowing an address range of 2^{28} = 256 MB. Rather than make this branch PC relative, this is interpreted as an absolute address within a 256-MB segment. That's inconvenient for single programs larger than this, although it hasn't been much of a problem yet!

  Branches out of segment can be achieved by using a jump register instruction, which can go to any 32-bit address.

  Conditional branches have only a 16-bit displacement field—giving a 2^{18}-byte range, since instructions are four-byte aligned—which is interpreted as a signed PC-relative displacement. Compilers can only code a simple conditional branch instruction if they know that the target will be within 128 KB of the instruction following the branch.

1.5.3 *Features You Won't Find*

- *No byte or halfword arithmetic*: All arithmetical and logical operations are performed on 32-bit quantities. Byte and/or halfword arithmetic requires significant extra resources and many more opcodes, and it is rarely really useful. The C language's semantics cause most calculations to be carried out with `int` precision, and for MIPS `int` is a 32-bit integer.

  However, where a program explicitly does arithmetic as `short` or `char`, a MIPS compiler must insert extra code to make sure that the

results wrap and overflow as they would on a native 16- or 8-bit machine.

- *No special stack support*: Conventional MIPS assembly usage does define one of the registers as a stack pointer, but there's nothing special to the hardware about `sp`. There is a recommended format for the stack frame layout of subroutines, so that you can mix modules from different languages and compilers; you should almost certainly stick to these conventions, but they have no relationship to the hardware.

  A stack pop wouldn't fit the pipeline, because it would have two register values to write (the data from the stack and the incremented pointer value).

- *Minimal subroutine support*: There is one special feature: jump instructions have a jump and link option, which stores the return address into a register. `$31` is the default, so for convenience and by convention `$31` becomes the return address register.

  This is less sophisticated than storing the return address on a stack, but it has some significant advantages. Two examples will give you a feeling for the argument: First, it preserves a pure separation between branch and memory-accessing instructions; second, it can aid efficiency when calling small subroutines that don't need to save the return address on the stack at all.

- *Minimal interrupt handling*: It is hard to see how the hardware could do less. It stashes away the restart location in a special register, modifies the machine state just enough to let you find out what happened and to disallow further interrupts, then jumps to a single predefined location in low memory. Everything else is up to the software.

- *Minimal exception handling*: Interrupts are just one sort of exception (the MIPS word *exception* covers all sorts of events where the CPU may want to interrupt normal sequential processing and invoke a software handler). An exception may result from an interrupt, an attempt to access virtual memory that isn't physically present, or many other things. You go through an exception, too, on a deliberately planted trap instruction like a system call that is used to get into the kernel in a protected OS. All exceptions result in control passing to the same fixed entry point.[7]

  On any exception, a MIPS CPU *does not* store anything on a stack, write memory, or preserve any registers for you.

  By convention, two general-purpose registers are reserved so that exception routines can bootstrap themselves (it is impossible to do anything

7. I exaggerate slightly; these days there are quite a few different entry points, and there were always at least two. Details will be given in section 5.3.

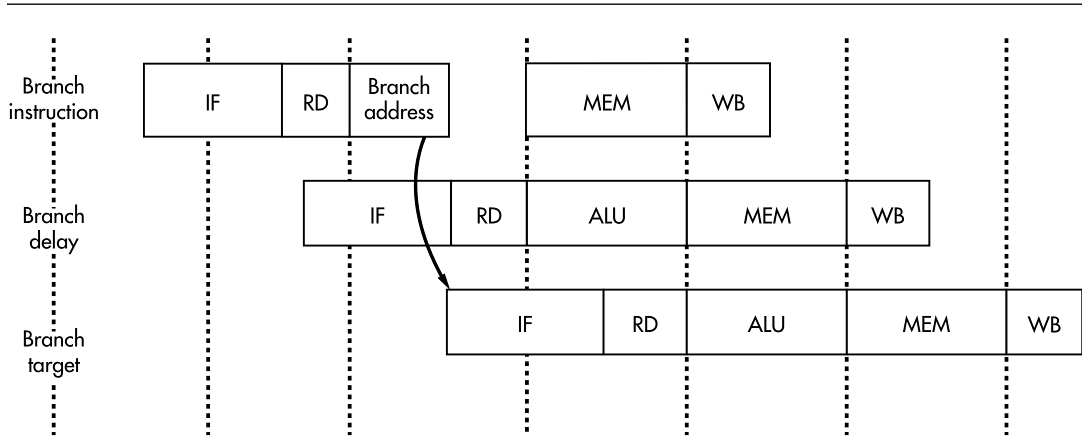

FIGURE 1.3 The pipeline and branch delays.

on a MIPS CPU without using some registers). For a program running in any system that takes interrupts or traps, the values of these registers may change at any time, so you'd better not use them.

1.5.4 *Programmer-Visible Pipeline Effects*

So far, this has all been what you might expect from a simplified CPU. However, making the instruction set pipeline friendly has some stranger effects as well, and to understand them we're going to draw some pictures.

- *Delayed branches*: The pipeline structure of the MIPS CPU (Figure 1.3) means that when a jump/branch instruction reaches the execute phase and a new program counter is generated, the instruction after the jump will already have been started. Rather than discard this potentially useful work, the architecture dictates that the instruction after a branch must always be executed before the instruction at the target of the branch. The instruction position following any branch is called the *branch delay slot*.

  If nothing special was done by the hardware, the decision to branch or not, together with the branch target address, would emerge at the end of the ALU pipestage—by which time, as Figure 1.3 shows, you're too late to present an address for an instruction in even the next-but-one pipeline slot.

  But branches are important enough to justify special treatment, and you can see from Figure 1.3 that a special path is provided through the ALU to make the branch address available half a clock cycle early. Together with the odd half-clock-cycle shift of the instruction fetch stage, that means that the branch target can be fetched in time to become the next but one,

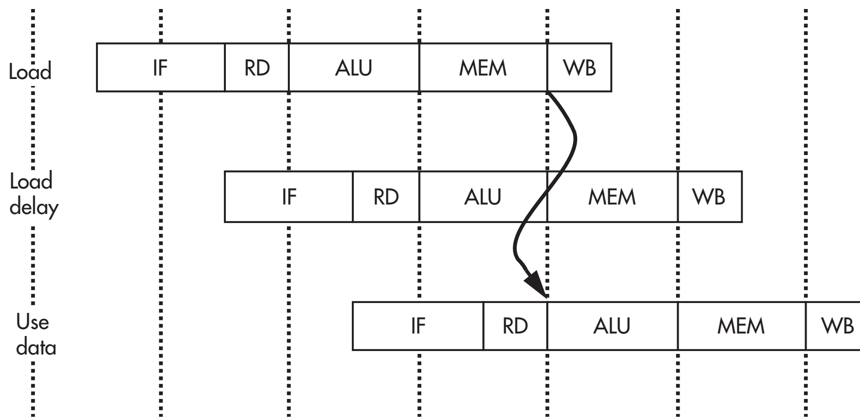

FIGURE 1.4 The pipeline and load delays.

so the hardware runs the branch instruction, then the branch delay slot instruction, and then the branch target—with no other delays.

It is the responsibility of the compiler system or the assembly programming wizard to allow for and even to exploit the branch delay; it turns out that it is usually possible to arrange that the instruction in the branch delay slot does useful work. Quite often, the instruction that would otherwise have been placed before the branch can be moved into the delay slot.

This can be a bit tricky on a conditional branch, where the branch delay instruction must be (at least) harmless on both paths. Where nothing useful can be done, the delay slot is filled with a **nop** instruction.

Many MIPS assemblers will hide this odd feature from you unless you explicitly ask them not to.

- *Late data from load (load delay slot)*: Another consequence of the pipeline is that a load instruction's data arrives from the cache/memory system *after* the next instruction's ALU phase starts—so it is not possible to use the data from a load in the following instruction. (See Figure 1.4 for how this works.)

  The instruction position immediately after the load is called the *load delay slot*, and an optimizing compiler will try to do something useful with it. The assembler will hide this from you but may end up putting in a **nop**.

  On modern MIPS CPUs the load result is interlocked: If you try to use the result too early, the CPU stops until the data arrives. But on early MIPS CPUs, there were no interlocks, and the attempt to use data in the load delay slot led to unpredictable results.