In contrast to the 68000's CISC architecture, the MIPS family of microprocessors is one of the commercial pioneers of RISC. MIPS began as a 32-bit architecture with 32-bit instruction words and 32 general-purpose registers. In the 1990s the architecture was extended to 64 bits. MIPS instruction words are classified into three basic types: immediate (I-type), jump (J-type), and register (R-type). The original MIPS architecture supports four 32-bit addition instructions without any addressing mode permutations: add signed (ADD), add unsigned (ADDU), add signed immediate (ADDI), and add unsigned immediate (ADDIU). These instructions are represented by two types of instruction words, I-type and R-type, as shown in Table 7.4.

TABLE 7.4 MIPS Addition Instruction Words

I-type bits	31:26	25:21	20:16	15:0		
Field	Opcode	Source Register	Target Register	Immediate data		
ADDI	001000	Rn	Rn	Data		
ADDIU	001001	Rn	Rn	Data		
R-type bits	31:26	25:21	20:16	15:11	10:6	5:0
Field	Opcode	Source Register	Target Register	Destination Register	Shift Amount	Function
ADD	000000	Rn	Rn	Rn	00000	100000
ADDU	000000	Rn	Rn	Rn	00000	100001

The immediate operations specify two registers and a 16-bit immediate operand: $R_T = R_S$ + Immediate. The other instructions operate on registers only and allow the programmer to specify three registers: $R_D = R_S + R_T$. If you want to add data that is in memory, that data must first be loaded into a register. Whereas a single 68000 instruction can fetch a word from memory, increment the associated pointer register, add the word to another register, and then store the result back into memory, a MIPS microprocessor would require separate instructions for each of these steps. This is in keeping with RISC concepts: use more simpler instructions to get the job done.

Instruction decode logic for a typical RISC microprocessor can be much simpler than for a CISC counterpart, because there are fewer instructions to decode and fewer operand complexities to recognize and coordinate. Generally speaking, a RISC microprocessor accesses data memory only with dedicated load/store instructions. Data manipulation instructions operate solely on internal registers and immediate operands. Under these circumstances, microprocessor engineers are able to heavily optimize their design in favor of the reduced instruction set that is supported. It turns out that not all instructions in a CISC microprocessor are used with the same frequency. Rather, there is a core set of instructions that are called most of the time, and the rest are used infrequently. Those that are used less often impose a burden on the entire system, because they increase the permutations that the decode logic must handle in any given clock cycle. By removing the operations that are not frequently used, the microprocessor's control logic is simplified and can therefore be made to run faster. The result is improved throughput for the most commonly executed operations, which translates directly into greater performance overall.

The fundamental assumption that RISC microprocessors rely on to maintain their throughput is high memory bandwidth. For a RISC microprocessor to match or outperform a CISC microproces-

sor, it must be able to rapidly fetch instructions, because several RISC instructions are necessary to match the capabilities of certain CISC instructions. An older computer architecture with an asynchronous memory interface may not be able to provide sufficient instruction bandwidth to make a RISC microprocessor efficient. CISC architectures dominated off-the-shelf microprocessor offerings until low-latency memory subsystems became practical at a reasonable cost. Modern computer architectures implement very fast memory interfaces that are able to provide a steady stream of instructions to RISC microprocessors.

One fundamental technique for improving the instruction fetch bandwidth is to design a microprocessor with two memory interfaces—one for instructions and one for data. This is referred to as a *Harvard* architecture, as compared to a conventional *von Neumann* architecture in which instruction and data memory are unified. Using a Harvard architecture, instruction fetches are not disrupted by load/store operations. Unfortunately, a Harvard architecture presents numerous system-level problems of how to split program and data memory and how to load programs into memory that cannot be accessed by load/store operations. Most microprocessors that implement a Harvard architecture do so with smaller on-chip memory arrays that can store segments of program and data that are fetched from and written back to a unified memory structure external to the microprocessor chip. While this may sound so complex as to only be in the realm of serious number-crunchers, the small but powerful 8-bit PIC™ RISC microcontrollers from Microchip Technology implement a Harvard architecture with mutually exclusive program and data memory structures located on chip. This illustrates the point that advanced microprocessor concepts can be applied to any level of performance if a problem needs to be solved.

The RISC concept appears to have won the day in the realm of high-performance computing. With memory bandwidth not being much of a hindrance, streamlined RISC designs can be made fast and efficient. In embedded computing applications, the victor is less clear. CISC technology is still firmly entrenched in a market where slow memory subsystems are still common and core microprocessor throughput is not always a major design issue. What is clear is that engineers and marketers will continue to debate and turn out new products and literature to convince others why their approach is the best available.

7.2 CACHE STRUCTURES

Microprocessor and memory performance have improved asymmetrically over time, leading to a well recognized performance gap. In 1980, a typical microprocessor ran at under 10 MHz, and a typical DRAM exhibited an access time of about 250 ns. Two decades later, high-end microprocessors were running at several hundred megahertz, and a typical DRAM exhibited an access time of 40 ns. Microprocessors' appetites for memory bandwidth has increased by about two orders of magnitude over 20 years while main memory technology, most often DRAM, has improved by less than an order of magnitude during that same period. To make matters worse, many microprocessors shifted from CISC to RISC architectures during this same period, thereby further increasing their demand for instruction memory bandwidth. The old model of directly connecting main memory to a microprocessor has broken down and become a performance-limiting bottleneck.

The culprits for slow main memory include the propagation delays through deep address decoding logic and the high random access latency of DRAM—the need to assert a row address, wait some time, assert a column address, and wait some more time before data is returned. These problems can be partially addressed by moving to SRAM. SRAM does not exhibit the latency penalty of DRAM, but there are still the address decoding delays to worry about. It would be nice to build main memory with SRAM, but this is prohibitively expensive, as a result of the substantially lower den-