J. Webster (ed.), *Wiley Encyclopedia of Electrical and Electronics Engineering*
Copyright © 1999 John Wiley & Sons, Inc.

HARVARD ARCHITECTURES

Improving the speed of arithmetic computations has been a desire of human beings ever since arithmetic itself was invented. The abacus, a frame containing parallel rods with sliding beads, was used to simplify computing for several thousand years. The invention of logarithms by John Napier in the seventeenth century is perhaps the next significant invention in fast arithmetic. Several centuries were spent studying and designing calculating aids, and developing numerical analysis techniques, before the advent of modern computers.

Figure 1 illustrates the basic structure of all modern computers. The arithmetic and logic unit (*ALU*) is the "brain" of the computer, and it obtains both the programs and the data from a memory unit. Data or programs can be input into the computer system through various input-output devices. The operations of all the units are controlled by the control unit. The control unit together with the ALU referred to as the *central processing unit* (*CPU*).

Computers consisting of a CPU and memory to store programs and data are omnipresent in the modern world, and many of us take this structure of computers for granted. However, no devices with any similarity to the modern computer were built prior to World War II. It indeed took a long path of evolution from abacus to the first computer, although progress has been relatively fast in the past 50 years.

Historical Perspective

In this section, we describe two early computers, one from Harvard University, the Harvard Mark I, and one from Princeton University, the Princeton IAS computer. We also describe the fundamentals of the von Neumann model of computing.

Harvard Mark I. Howard Aiken (1900–1973), while an instructor and graduate student in the Department of Physics at Harvard University in 1937 (1), proposed the development of a machine to evaluate formulae, tabulate results, compute infinite series, solve differential equations, and perform integration and differentiation of numerical data. IBM not only funded his proposal, but also sent skilled IBM engineers to build the machine. IBM's experience with tabulators (machines that used punched cards to perform limited calculations under plugboard control) was applied to the construction of the machine. The result was the sequence-controlled automatic calculator (*SCAC*), later known as the Harvard Mark I. Aiken obtained a doctorate from Harvard University in 1939, and was appointed a faculty instructor (equivalent to assistant professor) in physics and communications engineering at Harvard. He continued to work on the Harvard Mark I until it was completed in 1943.

The Harvard Mark I computer had disjoint instruction and data memories. The program was stored on a 24-channel punched paper tape, from which it was read into the processor. Programming constructs such as looping were not supported. Data were input into the machine using punched-paper-tape readers identical to those that read the program in. There were sixty 24-bit decimal registers, each of which acted as an accumulator. There was no central arithmetic unit. The computations were distributed in the accumulators. Arithmetic operations supported included addition, subtraction, multiplication, division, log, exponent with base 10, and

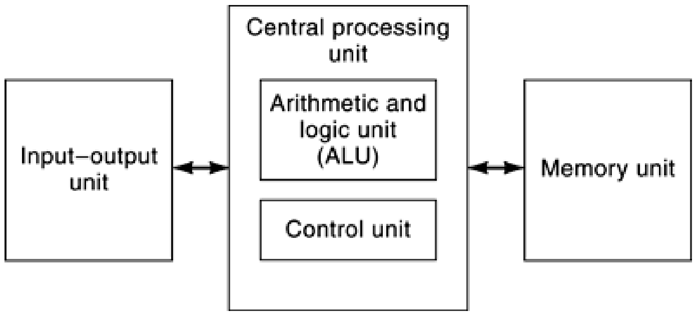

Fig. 1. Basic organization of all modern computers.

sin x. The computer consisted of stepping switches, rotating shafts, and cam-driven pulse generators. The machine became operational in 1943 and was in use until 1959. Some original description of this computer can be found in its manual of operation written by the staff of the Computation Laboratory (2).

Princeton IAS Computer. Von Neumann and his colleagues at the Institute for Advanced Studies (*IAS*) at Princeton University built a computer around 1946, which was called the Princeton IAS computer. There are three major subsystems in this architecture: the instruction processing unit, arithmetic unit, and memory. It employed a random-access cathode-ray-tube main memory, which stored both programs and data. The instruction processing unit sends instruction or data addresses to the memory. The output of the memory is sent to the instruction processing unit (if it is an instruction) or to the arithmetic unit (if it is a datum). The most distinguishing feature of the IAS computer with respect to previous machines was the use of a single memory to store both programs and data. It should be remembered here that many earlier computers, such as the Babbage analytical engine, Zuse Z3, and Harvard Mark I, had separate instruction and data memories. The Princeton IAS computer is the embodiment of the von Neumann model of computing, described in the following subsection.

Von Neumann Computers. The most unifying concept in all computer designs built since the 1950s is that they are all *stored-program computers*. This term indicates that the programs are stored in some kind of memory, wherefrom it is fed to the processing unit. Although that may seem obvious to many in the current generation, early computing devices did not have their programs stored.

John Von Neumann is the person most credited for the development of the structure of the stored-program computer. He is considered by many as the first to posit that instructions and data are not fundamentally different, and that they both can be stored in the same memory. During World War II, the University of Pennsylvania developed the first electronic computer, called ENIAC (Electronic Numerical Integrator Calculator). J. P. Eckert and J. Mauchly from the Moore School of Electrical Engineering at the University of Pennsylvania were the principal engineers involved in the development of the machine, and Von Neumann was helping the team to improve the way programs are entered and stored. Von Neumann put the ideas together and wrote a memo proposing a stored-program computer in 1944. Later, in 1946, von Neumann wrote a landmark paper with Burks and Goldstine (3), and the concepts in it are embodied in the Princeton IAS computer.

The stored-program concept had a major impact on the advancement of computers. The concept of treating instructions and data alike and the ability to manipulate instructions just like data helped the development of compilers, assemblers, linkers, and so on. A *compiler* translates a program written in a high-level language such as C, PASCAL, or FORTRAN to an intermediate format, called assembly language, corresponding to the machine on which the program will be run. An *assembler* takes this assembly-language program and converts it into the binary or machine language of the machine. *Linkers* are programs that help to create a single executable program (binary) from multiple files that are part of a single application. As far as the compiler

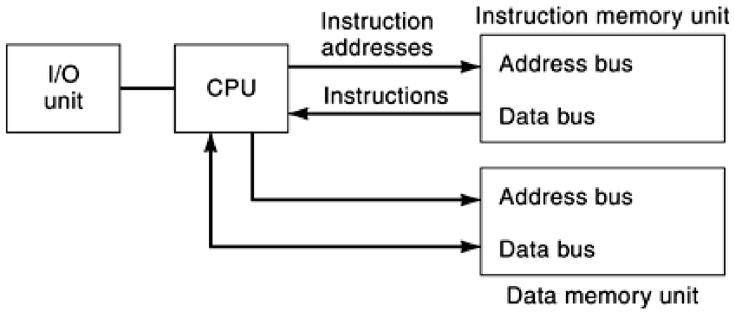

Fig. 2. Harvard-style architecture where data and instructions are stored in separate memory units.

is concerned, the high-level language code is its data. Similarly, the data for assemblers and linkers are all actually programs or code. Thus data and code are treated alike in many respects.

Definition of Harvard Architecture

Acknowledging the disjoint instruction and data memories in the Mark I machine built at Harvard University, the computer design style where data and instructions are stored in separate modules is called *Harvard* architecture. Figure 2 illustrates the concept. Instruction addresses are sent to the instruction memory, which responds with instructions to the processor. Data addresses are fed to the data memory, and it sends data back to the processor. Instruction and data can be accessed simultaneously. As opposed to this, machines that use unified instruction and data memories are called *Princeton* architectures, acknowledging the Princeton IAS project. Figure 3 illustrates the basic structure of the Princeton architecture. If the processor needs an instruction, the instruction address is sent to the memory unit, which sends the instruction to the instruction register (*IR*) and instruction decoder of the processor. If a datum is needed, the address is sent to the same memory unit, and the memory sends the datum to the memory data register (*MDR*) of the processor. Unless the memory unit is multiported, only one access can be happening at any time. Typically the term *von Neumann architecture* is used to imply a stored-program computer, and the term *Princeton architecture* is often used to specify distinctly that there is a unified memory instead of separate instruction and data memories. Some description of the early computers and the Harvard architecture concept can be found in computer architecture textbooks (4,5,6,7,8,9,10).

Modern Architectures with Harvard Style

Split Instruction and Data Caches. Modern computers use a hierarchical memory, typically consisting of two or three levels of cache memories, a physical memory, and the hard disk or virtual memory, as illustrated in Fig. 4. It is very common to use separate (split) caches for instructions and data. Processors with split instruction and data caches, as shown in Fig. 5, can be said to have the Harvard architecture as opposed to the Princeton architecture [at least at the level-1 (L1) cache].

Pipelining, a commonly employed technique to get high performance, makes it essential to have split instruction and data caches, at least at L1. A simple pipeline has separate stages for instruction fetching, instruction decoding, instruction execution, and memory access. When one instruction is being fetched, a prior instruction is being decoded and another previous instruction is being executed. A fourth previous

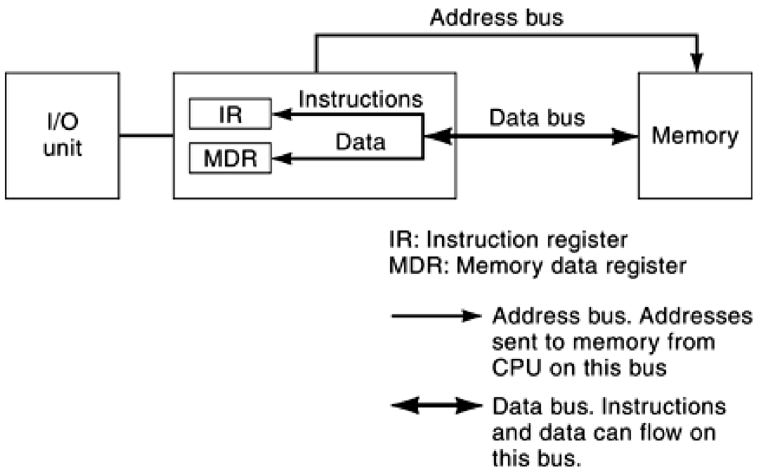
Address bus
I/O unit
IR
MDR
Instructions
Data
Data bus
Memory
IR: Instruction register
MDR: Memory data register
Address bus. Addresses sent to memory from CPU on this bus
Data bus. Instructions and data can flow on this bus.

Fig. 3. Princeton-style architecture where instructions and data are stored in the same memory unit.

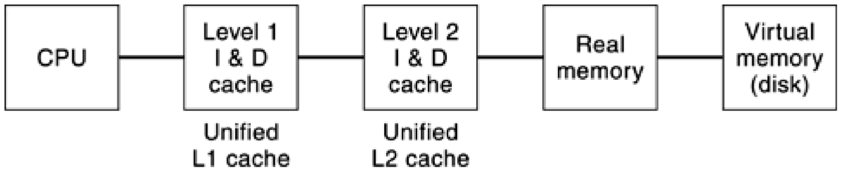

Fig. 4. Memory hierarchy with unified L1 and L2 caches.

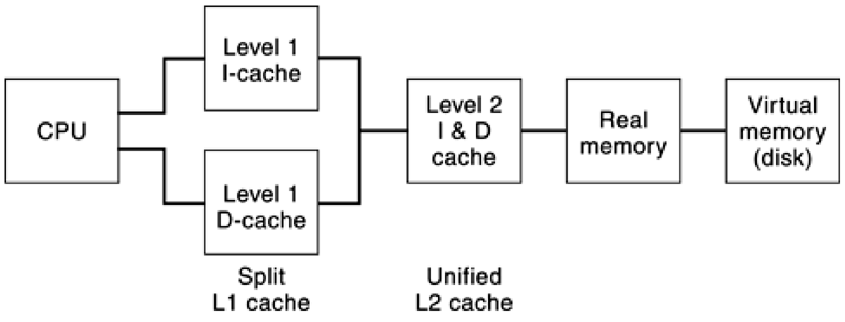

Fig. 5. Memory hierarchy with split L1 caches and unified L2 caches.

instruction will be in the memory access stage. Consider the datapath in Fig. 6. Instructions and data are stored in one memory unit, and they are fetched into the instruction register or memory data register as appropriate. Such a datapath cannot be used for a pipelined processor, because there will be a structural hazard (resource dependency) in every cycle. Pipelining necessitates a datapath such as that in Fig. 7, which supports simultaneous access to both the instruction memory unit and the data memory unit in every cycle. Instruction x can be accessing data while instruction x+4 is being fetched. Unified L1 caches were thus ruled out with the advent of pipelined processors.

Most modern microprocessors have split L1 caches as illustrated in Table 1. It may be noted that in most processors with split caches, the cache is evenly divided into the instruction cache (I cache) and the data cache

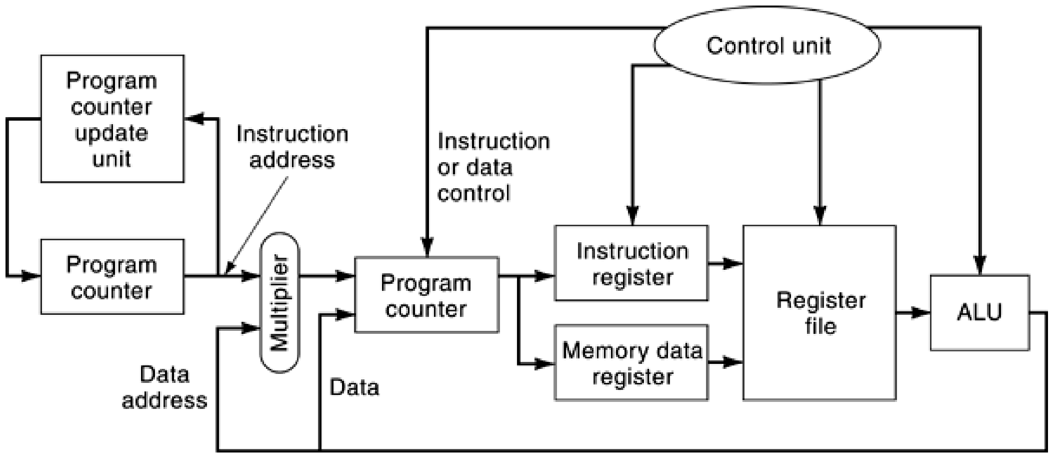

Fig. 6. A datapath with unified memory. Such a datapath will not work well with a pipelined processor, because instruction and data memory accesses will result in resource conflict.

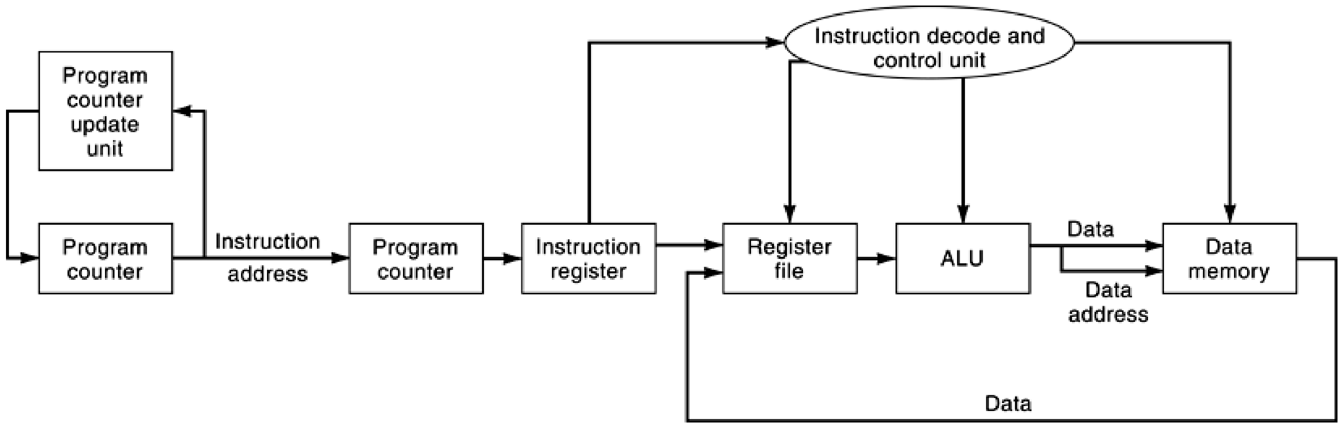

Fig. 7. A datapath with split L1 caches. This kind of a datapath works well with pipelining.

(D cache). An exception to this is the HP PA7200. The only processors with unified L1 caches are the Intel i486 processors. Many of the early processors in this list did not have a level-2 (L2) cache, but if they had one, it was unified.

Separating instructions and data into separate units provides several benefits:

(1) One can obtain larger memory bandwidth by having separate instruction and data caches. One instruction and one datum can be accessed simultaneously. Most microprocessors of today make a reference to data memory for 30% to 40% of all instructions. This means that one needs 1.3 to 1.4 accesses to the cache per instruction if it is a unified cache. In superscalar processors (which issue more than one instruction in the same cycle), this would amount to accessing a unified cache several times during the same cycle.
(2) At least the L1 caches need to have a split topology in order to support pipelining. Pipelining is a very important technique for high performance, and hence L1 caches in all modern processors are split into instruction and data caches.

Table 1. L1 Caches in a Few Commercial Microprocessors

Processor	Topology	Cache Size (bytes) L1 I-Cache	Cache Size (bytes) L1 D-Cache
Intel i486DX	Unified	8K	—
Intel i486DX2	Unified	8K	—
Intel i486DX4	Unified	16K	—
Intel P5	Split	8K	8K
Motorola M68030	Split	256	256
Motorola M68040	Split	4K	4K
Motorola M68060	Split	8K	8K
HP PA7200	Split	2K	1M
Alpha21164	Split	8K	8K
Alpha21264	Split	64K	64K
UltraSparc II	Split	16K	16K
HAL SPARC64	Split	64K	64K
Pentium III (IA32)	Split	16K	16K
AMD Athlon	Split	64K	64K
PowerPC620	Split	8K	8K
MIPS R10000	Split	32K	32K

(3) Split caches can be tuned to suit the behavior of instructions or data as needed. For instance, spatial locality of code is often good, whereas spatial locality of data is good only in certain kinds of programs. Instruction caches can be designed to have larger block sizes in such cases, with slightly smaller block sizes for data caches. Such optimizations are not possible if unified caches are used.

(4) Most processors do not use self-modifying code. Under that condition, instruction memory is read-only, and the designer does not need to worry about write policies such as writeback or writethrough. The control circuitry for the instruction cache can be simplified accordingly.

(5) Conflicts between instructions and data in the cache can be avoided by using separate instruction and data caches.

Disadvantages of Split Instruction and Data Memories.. Split instruction and data caches are not without demerits. The most important disadvantages are the following:

(1) Double caching may result from having separate instruction and data caches. Data to one application may be instructions to another application. For instance a program is data to the compiler. The executable (machine-language file) is output data for the compiler; however, it is instructions when you execute the user program. Similarly, during Java program execution, the user code is data to the interpreter or just-in-time (*JIT*) compiler, whereas it is actually a program. In many such instances, separate instruction and data memories (caches) will mean storing the same information in both places.

(2) In split caches, the apportioning of the cache area into instruction area and data area gets fixed at design time. But partitioning resources dynamically (at run time) has been generally seen to be more effective than

performing the partitioning at design time. For instance, even if a program does not use all of the instruction cache space, but needs more cache space for data, data cannot share unused space in the instruction cache if split caches are used. In the case of unified caches, program or data can occupy more space, depending on the need. And this partitioning can be different in different programs or during different phases of execution of the same program.

Performance of Unified and Split Caches.. All modern microprocessors use split instruction and data caches at L1. When the available chip area is split into instruction and data caches, each cache is smaller than the unified cache. There have been simulation studies investigating the performance of different cache configurations (11). Table 2 illustrates the miss ratios of unified and split caches for the *SPEC* benchmarks. (SPEC stands for System Performance Evaluation Cooperative, a consortium of companies that have joined to create benchmark suites for computer performance evaluation.) The SPEC benchmarks consist of integer and floating-point programs. Separate averages are shown for integer and floating-point programs, due to the difference in their cache behavior. Four different configurations, with a particular cache used either as a unified cache or split equally into instruction or data caches, are presented. The miss ratios of the split instruction caches are seen to be better than those of unified caches; however, the split data caches incur higher miss rates than the unified ones, because they are only half as big. The instruction cache will be accessed once for every instruction, and the data cache will be accessed for every load and store instruction (approximately 30% of all instructions). Hence the overall miss rate of the split caches is comparable to that of the unified caches. It may also be remembered that cache miss ratios are not the sole metric of cache performance. The split caches have twice the bandwidth of unified caches. The unified cache is at times attractive from the point of view of miss ratios alone, however, considering that simultaneous access of instructions and data is possible only in the split caches, the latter are often advantageous.

Split L1 and Unified L2 caches. It is beneficial in many ways to have disjoint instruction and data memories at any level of the memory hierarchy, but in order to make full use of the split memories, a system needs separate buses to the memory modules. Use of disjoint buses external to the chip necessitates duplicate pins for instruction and data addresses and for instruction bits and data bits. For buses that appear inside a chip, this will not be a problem. Since L1 caches in microprocessors are on chip, it is fairly simple to have separate instruction and data caches and separate address and data lines to them. However, L2 caches in many processors are situated outside the core die, and hence external buses are involved. Hence in most microprocessors, the L1 cache is split into instruction and data caches, and the L2 cache is unified. The L1 caches, which are accessed very frequently, benefit from the increased bandwidth of the split architecture. There is an access going to the instruction cache in every cycle, and there is an access going to the data cache in approximately 30% of the cycles (assuming 30% of instructions are loads or stores). L2 caches get accessed only when there is a miss in the L1 cache, and hence the need to access it simultaneously for both instructions and data is not as acute as it is for L1 caches. Hence unified L2 caches perform satisfactorily.

Future Directions

Split instruction and data caches are the order of the day. L1 caches in all modern microprocessors are split, and hence we can say that all these processors are exploiting the Harvard design style at least partially. Philosophically, instructions and data are alike, and one program's instruction is another program's data. Hence, philosophically, the notion of single memory housing both data and instruction is very sound. In practice, however, split instruction and data caches are advantageous for high performance. Pipelining cannot be supported without splitting at least the L1 caches. Splitting memory contents into instructions and data is a very good way to achieve higher bandwidth at lower costs, even irrespective of pipelining. However L2 and L3 caches are typically unified. In a couple of years, the integration densities will be high enough to permit

Table 2. Cache Miss Rates of Unified and Split Caches for SPEC 92 Benchmarks[a]

Cache Configuration	SPEC INT Average	SPEC FP Average	SPEC Overall Average
1K Instruction	0.0882	0.0782	0.0822
1K Data	0.1663	0.2639	0.2249
2K Unified	0.1069	0.1503	0.1330
2K Instruction	0.0599	0.0709	0.0665
2K Data	0.1247	0.2315	0.1888
4K Unified	0.0792	0.1225	0.1052
4K Instruction	0.0429	0.0604	0.0534
4K Data	0.0948	0.1898	0.1518
8K Unified	0.0492	0.0921	0.0749
8K Instruction	0.0229	0.0488	0.0384
8K Data	0.0725	0.1368	0.1111
16K Unified	0.0305	0.0655	0.0515

[a] Based on Gee et. al. (11).

L2 caches on the same die as the processor, and Harvard-style organization can then be easily adopted for L2 caches. However, bandwidth requirement at L2 is not as high as at L1 and the dynamic partitioning of unified caches into instruction and data areas at run time has several advantages. Hence L2 caches may continue to be unified for years to come.

BIBLIOGRAPHY

1. I. B. Cohen *Howard Aiken: Portrait of a Computer Pioneer*, Cambridge, MA, and London: MIT Press, 1999.
2. The Staff of the Computation laboratory, *A Manual of Operation for the Automatic Sequence Controlled Calculator*, Cambridge, MA: Harvard University Press, 1946.
3. A. W. Burkes H. H. Goldstine J. von Neumann *Preliminary discussions of the logical design of an electronic computing instrument, Report*, U.S. Army Ordnance Department, 1946.
4. H. G. Cragon *Computer Architecture and Implementation*, New York: Cambridge Univ. Press, 2000.
5. V. C. Hamacher Z. G. Vranesic S. G. Zaky *Computer Organization*, 4th ed., New York: McGraw-Hill, 1996.
6. J. P. Hayes *Computer Architecture and Organization*, New York: McGraw-Hill, 1988.
7. M. J. Murdocca V. P. Heuring *Principles of Computer Architecture*, Upper Saddle River, NJ: Prentice-Hall, 2000.
8. D. A. Patterson J. L. Hennessy *Computer Organization and Design: The Hardware/Software Interface*, San Francisco: Morgan Kaufmann, 1998.

9. W. Stallings *Computer Organization and Architecture, Designing for Performance*, 4th ed., Upper Saddle River, NJ: Prentice-Hall, 1996.
10. M. R. Zargham *Computer Architecture: Single and Parallel Systems*, Upper Saddle River, NJ: Prentice-Hall, 1996.
11. J. D. Gee *et al. Cache performance of the SPEC benchmark suite, Technical Report UCB/CSD 91/648*, Computer Sciences Division, University of California, Berkeley.

LIZY KURIAN JOHN
The University of Texas at Austin