CHAPTER 1

Introduction to Computer Systems

The technological advances witnessed in the computer industry are the result of a long chain of immense and successful efforts made by two major forces. These are the academia, represented by university research centers, and the industry, represented by computer companies. It is, however, fair to say that the current technological advances in the computer industry owe their inception to university research centers. In order to appreciate the current technological advances in the computer industry, one has to trace back through the history of computers and their development. The objective of such historical review is to understand the factors affecting computing as we know it today and hopefully to forecast the future of computation. A great majority of the computers of our daily use are known as *general purpose* machines. These are machines that are built with no specific application in mind, but rather are capable of performing computation needed by a diversity of applications. These machines are to be distinguished from those built to serve (tailored to) specific applications. The latter are known as *special purpose* machines. A brief historical background is given in Section 1.1.

Computer systems have conventionally been defined through their interfaces at a number of layered abstraction levels, each providing functional support to its predecessor. Included among the levels are the application programs, the high-level languages, and the set of machine instructions. Based on the interface between different levels of the system, a number of computer architectures can be defined. The interface between the application programs and a high-level language is referred to as a *language architecture.* The *instruction set architecture* defines the interface between the basic machine instruction set and the *runtime* and *I/O control.* A different definition of computer architecture is built on four basic viewpoints. These are the structure, the organization, the implementation, and the performance. In this definition, the structure defines the interconnection of various hardware components, the organization defines the dynamic interplay and management of the various components, the implementation defines the detailed design of hardware components, and the performance specifies the behavior of the computer system. Architectural development and styles are covered in Section 1.2.

Fundamentals of Computer Organization and Architecture, by M. Abd-El-Barr and H. El-Rewini
ISBN 0-471-46741-3

A number of technological developments are presented in Section 1.3. Our discussion in this chapter concludes with a detailed coverage of CPU performance measures.

1.1. HISTORICAL BACKGROUND

In this section, we would like to provide a historical background on the evolution of cornerstone ideas in the computing industry. We should emphasize at the outset that the effort to build computers has not originated at one single place. There is every reason for us to believe that attempts to build the first computer existed in different geographically distributed places. We also firmly believe that building a computer requires teamwork. Therefore, when some people attribute a machine to the name of a single researcher, what they actually mean is that such researcher may have led the team who introduced the machine. We, therefore, see it more appropriate to mention the machine and the place it was first introduced without linking that to a specific name. We believe that such an approach is fair and should eliminate any controversy about researchers and their names.

It is probably fair to say that the first program-controlled (mechanical) computer ever build was the Z1 (1938). This was followed in 1939 by the Z2 as the first operational program-controlled computer with fixed-point arithmetic. However, the first recorded university-based attempt to build a computer originated on Iowa State University campus in the early 1940s. Researchers on that campus were able to build a small-scale special-purpose electronic computer. However, that computer was never completely operational. Just about the same time a complete design of a fully functional programmable special-purpose machine, the Z3, was reported in Germany in 1941. It appears that the lack of funding prevented such design from being implemented. History recorded that while these two attempts were in progress, researchers from different parts of the world had opportunities to gain first-hand experience through their visits to the laboratories and institutes carrying out the work. It is assumed that such first-hand visits and interchange of ideas enabled the visitors to embark on similar projects in their own laboratories back home.

As far as general-purpose machines are concerned, the University of Pennsylvania is recorded to have hosted the building of the Electronic Numerical Integrator and Calculator (ENIAC) machine in 1944. It was the first operational general-purpose machine built using vacuum tubes. The machine was primarily built to help compute artillery firing tables during World War II. It was programmable through manual setting of switches and plugging of cables. The machine was slow by today's standard, with a limited amount of storage and primitive programmability. An improved version of the ENIAC was proposed on the same campus. The improved version of the ENIAC, called the Electronic Discrete Variable Automatic Computer (EDVAC), was an attempt to improve the way programs are entered and explore the concept of stored programs. It was not until 1952 that the EDVAC project was completed. Inspired by the ideas implemented in the ENIAC, researchers at the Institute for Advanced Study (IAS) at Princeton built (in 1946) the IAS machine, which was about 10 times faster than the ENIAC.

In 1946 and while the EDVAC project was in progress, a similar project was initiated at Cambridge University. The project was to build a stored-program computer, known as the Electronic Delay Storage Automatic Calculator (EDSAC). It was in 1949 that the EDSAC became the world's first full-scale, stored-program, fully operational computer. A spin-off of the EDSAC resulted in a series of machines introduced at Harvard. The series consisted of MARK I, II, III, and IV. The latter two machines introduced the concept of separate memories for instructions and data. The term *Harvard Architecture* was given to such machines to indicate the use of separate memories. It should be noted that the term Harvard Architecture is used today to describe machines with separate cache for instructions and data.

The first general-purpose commercial computer, the UNIVersal Automatic Computer (UNIVAC I), was on the market by the middle of 1951. It represented an improvement over the BINAC, which was built in 1949. IBM announced its first computer, the IBM701, in 1952. The early 1950s witnessed a slowdown in the computer industry. In 1964 IBM announced a line of products under the name IBM 360 series. The series included a number of models that varied in price and performance. This led Digital Equipment Corporation (DEC) to introduce the first *minicomputer*, the PDP-8. It was considered a remarkably low-cost machine. Intel introduced the first *microprocessor*, the Intel 4004, in 1971. The world witnessed the birth of the first *personal computer* (PC) in 1977 when Apple computer series were first introduced. In 1977 the world also witnessed the introduction of the VAX-11/780 by DEC. Intel followed suit by introducing the first of the most popular microprocessor, the 80×86 series.

Personal computers, which were introduced in 1977 by Altair, Processor Technology, North Star, Tandy, Commodore, Apple, and many others, enhanced the productivity of end-users in numerous departments. Personal computers from Compaq, Apple, IBM, Dell, and many others, soon became pervasive, and changed the face of computing.

In parallel with small-scale machines, supercomputers were coming into play. The first such supercomputer, the CDC 6600, was introduced in 1961 by Control Data Corporation. Cray Research Corporation introduced the best cost/performance supercomputer, the Cray-1, in 1976.

The 1980s and 1990s witnessed the introduction of many commercial parallel computers with multiple processors. They can generally be classified into two main categories: (1) shared memory and (2) distributed memory systems. The number of processors in a single machine ranged from several in a shared memory computer to hundreds of thousands in a massively parallel system. Examples of parallel computers during this era include Sequent Symmetry, Intel iPSC, nCUBE, Intel Paragon, Thinking Machines (CM-2, CM-5), MsPar (MP), Fujitsu (VPP500), and others.

One of the clear trends in computing is the substitution of centralized servers by networks of computers. These networks connect inexpensive, powerful desktop machines to form unequaled computing power. Local area networks (LAN) of powerful personal computers and workstations began to replace mainframes and minis by 1990. These individual desktop computers were soon to be connected into larger complexes of computing by wide area networks (WAN).

TABLE 1.1 Four Decades of Computing

Feature	Batch	Time-sharing	Desktop	Network
Decade	1960s	1970s	1980s	1990s
Location	Computer room	Terminal room	Desktop	Mobile
Users	Experts	Specialists	Individuals	Groups
Data	Alphanumeric	Text, numbers	Fonts, graphs	Multimedia
Objective	Calculate	Access	Present	Communicate
Interface	Punched card	Keyboard & CRT	See & point	Ask & tell
Operation	Process	Edit	Layout	Orchestrate
Connectivity	None	Peripheral cable	LAN	Internet
Owners	Corporate computer centers	Divisional IS shops	Departmental end-users	Everyone

CRT, cathode ray tube; LAN, local area network.

The pervasiveness of the Internet created interest in network computing and more recently in grid computing. Grids are geographically distributed platforms of computation. They should provide dependable, consistent, pervasive, and inexpensive access to high-end computational facilities.

Table 1.1 is modified from a table proposed by Lawrence Tesler (1995). In this table, major characteristics of the different computing paradigms are associated with each decade of computing, starting from 1960.

1.2. ARCHITECTURAL DEVELOPMENT AND STYLES

Computer architects have always been striving to increase the performance of their architectures. This has taken a number of forms. Among these is the philosophy that by doing more in a single instruction, one can use a smaller number of instructions to perform the same job. The immediate consequence of this is the need for fewer memory read/write operations and an eventual speedup of operations. It was also argued that increasing the complexity of instructions and the number of addressing modes has the theoretical advantage of reducing the "semantic gap" between the instructions in a high-level language and those in the low-level (machine) language. A single (machine) instruction to convert several binary coded decimal (BCD) numbers to binary is an example for how complex some instructions were intended to be. The huge number of addressing modes considered (more than 20 in the VAX machine) further adds to the complexity of instructions. Machines following this philosophy have been referred to as *complex instructions set computers* (CISCs). Examples of CISC machines include the Intel Pentium™, the Motorola MC68000™, and the IBM & Macintosh PowerPC™.

It should be noted that as more capabilities were added to their processors, manufacturers realized that it was increasingly difficult to support higher clock rates that would have been possible otherwise. This is because of the increased

complexity of computations within a single clock period. A number of studies from the mid-1970s and early-1980s also identified that in typical programs more than 80% of the instructions executed are those using assignment statements, conditional branching and procedure calls. It was also surprising to find out that simple assignment statements constitute almost 50% of those operations. These findings caused a different philosophy to emerge. This philosophy promotes the optimization of architectures by speeding up those operations that are most frequently used while reducing the instruction complexities and the number of addressing modes. Machines following this philosophy have been referred to as *reduced instructions set computers* (RISCs). Examples of RISCs include the Sun SPARC™ and MIPS™ machines.

The above two philosophies in architecture design have led to the unresolved controversy as to which architecture style is "best." It should, however, be mentioned that studies have indicated that RISC architectures would indeed lead to faster execution of programs. The majority of contemporary microprocessor chips seems to follow the RISC paradigm. In this book we will present the salient features and examples for both CISC and RISC machines.

1.3. TECHNOLOGICAL DEVELOPMENT

Computer technology has shown an unprecedented rate of improvement. This includes the development of processors and memories. Indeed, it is the advances in technology that have fueled the computer industry. The integration of numbers of transistors (a transistor is a controlled on/off switch) into a single chip has increased from a few hundred to millions. This impressive increase has been made possible by the advances in the fabrication technology of transistors.

The scale of integration has grown from small-scale (SSI) to medium-scale (MSI) to large-scale (LSI) to very large-scale integration (VLSI), and currently to wafer-scale integration (WSI). Table 1.2 shows the typical numbers of devices per chip in each of these technologies.

It should be mentioned that the continuous decrease in the minimum devices feature size has led to a continuous increase in the number of devices per chip,

TABLE 1.2 Numbers of Devices per Chip

Integration	Technology	Typical number of devices	Typical functions
SSI	Bipolar	10–20	Gates and flip-flops
MSI	Bipolar & MOS	50–100	Adders & counters
LSI	Bipolar & MOS	100–10,000	ROM & RAM
VLSI	CMOS (mostly)	10,000–5,000,000	Processors
WSI	CMOS	>5,000,000	DSP & special purposes

SSI, small-scale integration; MSI, medium-scale integration; LSI, large-scale integration; VLSI, very large-scale integration; WSI, wafer-scale integration.

which in turn has led to a number of developments. Among these is the increase in the number of devices in RAM memories, which in turn helps designers to trade off memory size for speed. The improvement in the feature size provides golden opportunities for introducing improved design styles.

1.4. PERFORMANCE MEASURES

In this section, we consider the important issue of assessing the performance of a computer. In particular, we focus our discussion on a number of performance measures that are used to assess computers. Let us admit at the outset that there are various facets to the performance of a computer. For example, a user of a computer measures its performance based on the time taken to execute a given job (program). On the other hand, a laboratory engineer measures the performance of his system by the total amount of work done in a given time. While the user considers the program execution time a measure for performance, the laboratory engineer considers the throughput a more important measure for performance. A metric for assessing the performance of a computer helps comparing alternative designs.

Performance analysis should help answering questions such as how fast can a program be executed using a given computer? In order to answer such a question, we need to determine the time taken by a computer to execute a given job. We define the clock cycle time as the time between two consecutive rising (trailing) edges of a periodic clock signal (Fig. 1.1). Clock cycles allow counting unit computations, because the storage of computation results is synchronized with rising (trailing) clock edges. The time required to execute a job by a computer is often expressed in terms of clock cycles.

We denote the number of CPU clock cycles for executing a job to be the cycle count (CC), the cycle time by CT, and the clock frequency by $f = 1/CT$. The time taken by the CPU to execute a job can be expressed as

$$CPU\ time = CC \times CT = CC/f$$

It may be easier to count the number of instructions executed in a given program as compared to counting the number of CPU clock cycles needed for executing that

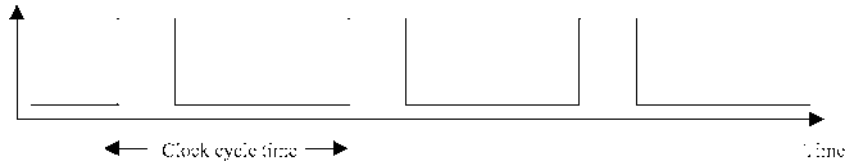

Figure 1.1 Clock signal

program. Therefore, the average number of clock cycles per instruction (CPI) has been used as an alternate performance measure. The following equation shows how to compute the CPI.

$$CPI = \frac{CPU\ clock\ cycles\ for\ the\ program}{Instruction\ count}$$

$$CPU\ time = Instruction\ count \times CPI \times Clock\ cycle\ time$$

$$= \frac{Instruction\ count \times CPI}{Clock\ rate}$$

It is known that the instruction set of a given machine consists of a number of instruction categories: *ALU* (simple assignment and arithmetic and logic instructions), *load*, *store*, *branch*, and so on. In the case that the CPI for each instruction category is known, the overall CPI can be computed as

$$CPI = \frac{\sum_{i=1}^{n} CPI_i \times I_i}{Instruction\ count}$$

where I_i is the number of times an instruction of type i is executed in the program and CPI_i is the average number of clock cycles needed to execute such instruction.

Example Consider computing the overall CPI for a machine A for which the following performance measures were recorded when executing a set of benchmark programs. Assume that the clock rate of the CPU is 200 MHz.

Instruction category	Percentage of occurrence	No. of cycles per instruction
ALU	38	1
Load & store	15	3
Branch	42	4
Others	5	5

Assuming the execution of 100 instructions, the overall CPI can be computed as

$$CPI_a = \frac{\sum_{i=1}^{n} CPI_i \times I_i}{Instruction\ count} = \frac{38 \times 1 + 15 \times 3 + 42 \times 4 + 5 \times 5}{100} = 2.76$$

It should be noted that the CPI reflects the organization and the instruction set architecture of the processor while the instruction count reflects the instruction set architecture and compiler technology used. This shows the degree of interdependence between the two performance parameters. Therefore, it is imperative that both the

CPI and the instruction count are considered in assessing the merits of a given computer or equivalently in comparing the performance of two machines.

A different performance measure that has been given a lot of attention in recent years is *MIPS* (million instructions-per-second (the rate of instruction execution per unit time)), which is defined as

$$MIPS = \frac{Instruction\ count}{Execution\ time \times 10^6} = \frac{Clock\ rate}{CPI \times 10^6}$$

Example Suppose that the same set of benchmark programs considered above were executed on another machine, call it machine B, for which the following measures were recorded.

Instruction category	Percentage of occurrence	No. of cycles per instruction
ALU	35	1
Load & store	30	2
Branch	15	3
Others	20	5

What is the *MIPS* rating for the machine considered in the previous example (machine A) and machine B assuming a clock rate of 200 MHz?

$$CPI_a = \frac{\sum_{i=1}^{n} CPI_i \times I_i}{Instruction\ count} = \frac{38 \times 1 + 15 \times 3 + 42 \times 4 + 5 \times 5}{100} = 2.76$$

$$MIPS_a = \frac{Clock\ rate}{CPI_a \times 10^6} = \frac{200 \times 10^6}{2.76 \times 10^6} = 70.24$$

$$CPI_b = \frac{\sum_{i=1}^{n} CPI_i \times I_i}{Instruction\ count} = \frac{35 \times 1 + 30 \times 2 + 20 \times 5 + 15 \times 3}{100} = 2.4$$

$$MIPS_b = \frac{Clock\ rate}{CPI_a \times 10^6} = \frac{200 \times 10^6}{2.4 \times 10^6} = 83.67$$

Thus $MIPS_b > MIPS_a$.

It is interesting to note here that although *MIPS* has been used as a performance measure for machines, one has to be careful in using it to compare machines having different instruction sets. This is because *MIPS* does not track execution time. Consider, for example, the following measurement made on two different machines running a given set of benchmark programs.

Instruction category	No. of instructions (in millions)	No. of cycles per instruction
Machine (A)		
ALU	8	1
Load & store	4	3
Branch	2	4
Others	4	3
Machine (B)		
ALU	10	1
Load & store	8	2
Branch	2	4
Others	4	3

$$CPI_a = \frac{\sum_{i=1}^{n} CPI_i \times I_i}{Instruction\ count} = \frac{(8 \times 1 + 4 \times 3 + 4 \times 3 + 2 \times 4) \times 10^6}{(8 + 4 + 4 + 2) \times 10^6} \cong 2.2$$

$$MIPS_a = \frac{Clock\ rate}{CPI_a \times 10^6} = \frac{200 \times 10^6}{2.2 \times 10^6} \cong 90.9$$

$$CPU_a = \frac{Instruction\ count \times CPI_a}{Clock\ rate} = \frac{18 \times 10^6 \times 2.2}{200 \times 10^6} = 0.198\,\text{s}$$

$$CPI_b = \frac{\sum_{i=1}^{n} CPI_i \times I_i}{Instruction\ count} = \frac{(10 \times 1 + 8 \times 2 + 4 \times 4 + 2 \times 4) \times 10^6}{(10 + 8 + 4 + 2) \times 10^6} = 2.1$$

$$MIPS_b = \frac{Clock\ rate}{CPI_a \times 10^6} = \frac{200 \times 10^6}{2.1 \times 10^6} = 95.2$$

$$CPU_b = \frac{Instruction\ count \times CPI_a}{Clock\ rate} = \frac{20 \times 10^6 \times 2.1}{200 \times 10^6} = 0.21\,\text{s}$$

$$MIPS_b > MIPS_a \qquad \text{and} \qquad CPU_b > CPU_a$$

The example shows that although machine B has a higher *MIPS* compared to machine A, it requires longer CPU time to execute the same set of benchmark programs.

Million floating-point instructions per second, MFLOP (rate of floating-point instruction execution per unit time) has also been used as a measure for machines' performance. It is defined as

$$MFLOPS = \frac{Number\ of\ floating\text{-}point\ operations\ in\ a\ program}{Execution\ time \times 10^6}$$

While MIPS measures the rate of average instructions, MFLOPS is only defined for the subset of floating-point instructions. An argument against MFLOPS is the fact that the set of floating-point operations may not be consistent across machines and therefore the actual floating-point operations will vary from machine to machine. Yet another argument is the fact that the performance of a machine for a given program as measured by MFLOPS cannot be generalized to provide a single performance metric for that machine.

The performance of a machine regarding one particular program might not be interesting to a broad audience. The use of arithmetic and geometric means are the most popular ways to summarize performance regarding larger sets of programs (e.g., benchmark suites). These are defined below.

$$Arithmetic\ mean = \frac{1}{n}\sum_{i=1}^{n} Execution\ time_i$$

$$Geometric\ mean = \sqrt[n]{\prod_{i=1}^{n} Execution\ time_i}$$

where *execution time*$_i$ is the execution time for the ith program and n is the total number of programs in the set of benchmarks.

The following table shows an example for computing these metrics.

Item	CPU time on computer A (s)	CPU time on computer B (s)
Program 1	50	10
Program 2	500	100
Program 3	5000	1000
Arithmetic mean	1835	370
Geometric mean	500	100

We conclude our coverage in this section with a discussion on what is known as the Amdahl's law for speedup (SU_o) due to enhancement. In this case, we consider speedup as a measure of how a machine performs after some enhancement relative to its original performance. The following relationship formulates Amdahl's law.

$$SU_o = \frac{Performance\ after\ enhancement}{Performance\ before\ enhancement}$$

$$Speedup = \frac{Execution\ time\ before\ enhancement}{Execution\ time\ after\ enhancement}$$

Consider, for example, a possible enhancement to a machine that will reduce the execution time for some benchmarks from 25 s to 15 s. We say that the speedup resulting from such reduction is $SU_o = 25/15 = 1.67$.

In its given form, Amdahl's law accounts for cases whereby improvement can be applied to the instruction execution time. However, sometimes it may be possible to achieve performance enhancement for only a fraction of time, Δ. In this case a new formula has to be developed in order to relate the speedup, SU_{Δ} due to an enhancement for a fraction of time Δ to the speedup due to an overall enhancement, SU_{o}. This relationship can be expressed as

$$SU_{\mathrm{o}} = \frac{1}{(1-\Delta) + (\Delta / SU_{\Delta})}$$

It should be noted that when $\Delta = 1$, that is, when enhancement is possible at all times, then $SU_{\mathrm{o}} = SU_{\Delta}$, as expected.

Consider, for example, a machine for which a speedup of 30 is possible after applying an enhancement. If under certain conditions the enhancement was only possible for 30% of the time, what is the speedup due to this partial application of the enhancement?

$$SU_{\mathrm{o}} = \frac{1}{(1-\Delta) + (\Delta / SU_{\Delta})} = \frac{1}{(1-0.3) + \dfrac{0.3}{30}} = \frac{1}{0.7 + 0.01} = 1.4$$

It is interesting to note that the above formula can be generalized as shown below to account for the case whereby a number of different independent enhancements can be applied separately and for different fractions of the time, $\Delta_1, \Delta_2, \ldots, \Delta_n$, thus leading respectively to the speedup enhancements $SU_{\Delta_1}, SU_{\Delta_2}, \ldots, SU_{\Delta_n}$.

$$SU_{\mathrm{o}} = \frac{1}{[1 - (\Delta_1 + \Delta_2 + \cdots + \Delta_n)] + \dfrac{(\Delta_1 + \Delta_2 + \cdots + \Delta_n)}{(SU_{\Delta_1} + SU_{\Delta_2} + \cdots + SU_{\Delta_n})}}$$

1.5. SUMMARY

In this chapter, we provided a brief historical background for the development of computer systems, starting from the first recorded attempt to build a computer, the Z1, in 1938, passing through the CDC 6600 and the Cray supercomputers, and ending up with today's modern high-performance machines. We then provided a discussion on the RISC versus CISC architectural styles and their impact on machine performance. This was followed by a brief discussion on the technological development and its impact on computing performance. Our coverage in this chapter was concluded with a detailed treatment of the issues involved in assessing the performance of computers. In particular, we have introduced a number of performance measures such as CPI, MIPS, MFLOPS, and Arithmetic/Geometric performance means, none of them defining the performance of a machine consistently. Possible

ways of evaluating the speedup for given partial or general improvement measurements of a machine were discussed at the end of this Chapter.

EXERCISES

1. What has been the trend in computing from the following points of view?
   (a) Cost of hardware
   (b) Size of memory
   (c) Speed of hardware
   (d) Number of processing elements
   (e) Geographical locations of system components
2. Given the trend in computing in the last 20 years, what are your predictions for the future of computing?
3. Find the meaning of the following:
   (a) Cluster computing
   (b) Grid computing
   (c) Quantum computing
   (d) Nanotechnology
4. Assume that a switching component such as a transistor can switch in zero time. We propose to construct a disk-shaped computer chip with such a component. The only limitation is the time it takes to send electronic signals from one edge of the chip to the other. Make the simplifying assumption that electronic signals can travel at 300,000 kilometers per second. What is the limitation on the diameter of a round chip so that any computation result can by used anywhere on the chip at a clock rate of 1 GHz? What are the diameter restrictions if the whole chip should operate at 1 THz $= 10^{12}$ Hz? Is such a chip feasible?
5. Compare uniprocessor systems with multiprocessor systems in the following aspects:
   (a) Ease of programming
   (b) The need for synchronization
   (c) Performance evaluation
   (d) Run time system
6. Consider having a program that runs in 50 s on computer A, which has a 500 MHz clock. We would like to run the same program on another machine, B, in 20 s. If machine B requires 2.5 times as many clock cycles as machine A for the same program, what clock rate must machine B have in MHz?
7. Suppose that we have two implementations of the same instruction set architecture. Machine A has a clock cycle time of 50 ns and a CPI of 4.0 for some program, and machine B has a clock cycle of 65 ns and a CPI of 2.5 for the same program. Which machine is faster and by how much?

8. A compiler designer is trying to decide between two code sequences for a particular machine. The hardware designers have supplied the following facts:

Instruction class	CPI of the instruction class
A	1
B	3
C	4

For a particular high-level language, the compiler writer is considering two sequences that require the following instruction counts:

Code sequence	Instruction counts (in millions)		
	A	B	C
1	2	1	2
2	4	3	1

What is the CPI for each sequence? Which code sequence is faster? By how much?

9. Consider a machine with three instruction classes and CPI measurements as follows:

Instruction class	CPI of the instruction class
A	2
B	5
C	7

Suppose that we measured the code for a given program in two different compilers and obtained the following data:

Code sequence	Instruction counts (in millions)		
	A	B	C
Compiler 1	15	5	3
Compiler 2	25	2	2

Assume that the machine's clock rate is 500 MHz. Which code sequence will execute faster according to MIPS? And according to execution time?

10. Three enhancements with the following speedups are proposed for a new machine: Speedup(a) = 30, Speedup(b) = 20, and Speedup(c) = 15. Assume that for some set of programs, the fraction of use is 25% for enhancement (a), 30% for enhancement (b), and 45% for enhancement (c). If only one enhancement can be implemented, which should be chosen to maximize the speedup? If two enhancements can be implemented, which should be chosen, to maximize the speedup?

REFERENCES AND FURTHER READING

J.-L. Baer, Computer architecture, *IEEE Comput.*, 17(10), 77–87, (1984).

S. Dasgupta, *Computer Architecture: A Modern Synthesis*, John Wiley, New York, 1989.

M. Flynn, *Some computer organization and their effectiveness*, *IEEE Trans Comput.*, C-21, 948–960 (1972).

D. Gajski, V. Milutinovic, H. Siegel, and B. Furht, *Computer Architecture: A Tutorial*, Computer Society Press, Los Alamitos, Calif, 1987.

W. Giloi, Towards a taxonomy of computer architecture based on the machine data type view, Proceedings of the 10th International Symposium on Computer Architecture, 6–15, (1983).

W. Handler, The impact of classification schemes on computer architecture, Proceedings of the 1977 International Conference on Parallel Processing, 7–15, (1977).

J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, 2nd ed., Morgan Kaufmann, San Francisco, 1996.

K. Hwang and F. Briggs, *Computer Architecture and Parallel Processing*, 2nd ed., McGraw-Hill, New York, 1996.

D. J. Kuck, *The Structure of Computers and Computations*, John Wiley, New York, 1978.

G. J. Myers, *Advances in Computer Architecture*, John Wiley, New York, 1982.

L. Tesler, Networked computing in the 1990s, reprinted from the Sept. 1991 Scientific American, The Computer in the 21st Century, 10–21, (1995).

P. Treleaven, Control-driven data-driven and demand-driven computer architecture (abstract), *Parallel Comput.*, 2, (1985).

P. Treleaven, D. Brownbridge, and R. Hopkins, Data drive and demand driven computer architecture, *ACM Comput. Surv.*, 14(1), 95–143, (1982).

Websites

http://www.gigaflop.demon.co.uk/~wasel