

CHAPTER

2

COMPUTER EVOLUTION AND PERFORMANCE

2.1 A Brief History of Computers

The First Generation: Vacuum Tubes
The Second Generation: Transistors
The Third Generation: Integrated Circuits
Later Generations

2.2 Designing for Performance

Microprocessor Speed
Performance Balance
Improvements in Chip Organization and Architecture

2.3 The Evolution of the Intel x86 Architecture

2.4 Embedded Systems and the ARM

Embedded Systems
ARM Evolution

2.5 Performance Assessment

Clock Speed and Instructions per Second
Benchmarks
Amdahl's Law

2.6 Recommended Reading and Web Sites

2.7 Key Terms, Review Questions, and Problems

KEY POINTS

- ◆ The evolution of computers has been characterized by increasing processor speed, decreasing component size, increasing memory size, and increasing I/O capacity and speed.
- ◆ One factor responsible for the great increase in processor speed is the shrinking size of microprocessor components; this reduces the distance between components and hence increases speed. However, the true gains in speed in recent years have come from the organization of the processor, including heavy use of pipelining and parallel execution techniques and the use of speculative execution techniques (tentative execution of future instructions that might be needed). All of these techniques are designed to keep the processor busy as much of the time as possible.
- ◆ A critical issue in computer system design is balancing the performance of the various elements so that gains in performance in one area are not handicapped by a lag in other areas. In particular, processor speed has increased more rapidly than memory access time. A variety of techniques is used to compensate for this mismatch, including caches, wider data paths from memory to processor, and more intelligent memory chips.

We begin our study of computers with a brief history. This history is itself interesting and also serves the purpose of providing an overview of computer structure and function. Next, we address the issue of performance. A consideration of the need for balanced utilization of computer resources provides a context that is useful throughout the book. Finally, we look briefly at the evolution of the two systems that serve as key examples throughout the book: the Intel x86 and ARM processor families.

2.1 A BRIEF HISTORY OF COMPUTERS

The First Generation: Vacuum Tubes

ENIAC The ENIAC (Electronic Numerical Integrator And Computer), designed and constructed at the University of Pennsylvania, was the world's first general-purpose electronic digital computer. The project was a response to U.S. needs during World War II. The Army's Ballistics Research Laboratory (BRL), an agency responsible for developing range and trajectory tables for new weapons, was having difficulty supplying these tables accurately and within a reasonable time frame. Without these firing tables, the new weapons and artillery were useless to gunners. The BRL employed more than 200 people who, using desktop calculators, solved the necessary ballistics equations. Preparation of the tables for a single weapon would take one person many hours, even days.

John Mauchly, a professor of electrical engineering at the University of Pennsylvania, and John Eckert, one of his graduate students, proposed to build a general-purpose computer using vacuum tubes for the BRL's application. In 1943, the Army accepted this proposal, and work began on the ENIAC. The resulting machine was enormous, weighing 30 tons, occupying 1500 square feet of floor space, and containing more than 18,000 vacuum tubes. When operating, it consumed 140 kilowatts of power. It was also substantially faster than any electro-mechanical computer, capable of 5000 additions per second.

The ENIAC was a decimal rather than a binary machine. That is, numbers were represented in decimal form, and arithmetic was performed in the decimal system. Its memory consisted of 20 "accumulators," each capable of holding a 10-digit decimal number. A ring of 10 vacuum tubes represented each digit. At any time, only one vacuum tube was in the ON state, representing one of the 10 digits. The major drawback of the ENIAC was that it had to be programmed manually by setting switches and plugging and unplugging cables.

The ENIAC was completed in 1946, too late to be used in the war effort. Instead, its first task was to perform a series of complex calculations that were used to help determine the feasibility of the hydrogen bomb. The use of the ENIAC for a purpose other than that for which it was built demonstrated its general-purpose nature. The ENIAC continued to operate under BRL management until 1955, when it was disassembled.

THE VON NEUMANN MACHINE The task of entering and altering programs for the ENIAC was extremely tedious. The programming process could be facilitated if the program could be represented in a form suitable for storing in memory alongside the data. Then, a computer could get its instructions by reading them from memory, and a program could be set or altered by setting the values of a portion of memory.

This idea, known as the *stored-program concept*, is usually attributed to the ENIAC designers, most notably the mathematician John von Neumann, who was a consultant on the ENIAC project. Alan Turing developed the idea at about the same time. The first publication of the idea was in a 1945 proposal by von Neumann for a new computer, the EDVAC (Electronic Discrete Variable Computer).

In 1946, von Neumann and his colleagues began the design of a new stored-program computer, referred to as the IAS computer, at the Princeton Institute for Advanced Studies. The IAS computer, although not completed until 1952, is the prototype of all subsequent general-purpose computers.

Figure 2.1 shows the general structure of the IAS computer (compare to middle portion of Figure 1.4). It consists of

- A main memory, which stores both data and instructions¹
- An arithmetic and logic unit (ALU) capable of operating on binary data

¹In this book, unless otherwise noted, the term *instruction* refers to a machine instruction that is directly interpreted and executed by the processor, in contrast to an instruction in a high-level language, such as Ada or C++, which must first be compiled into a series of machine instructions before being executed.

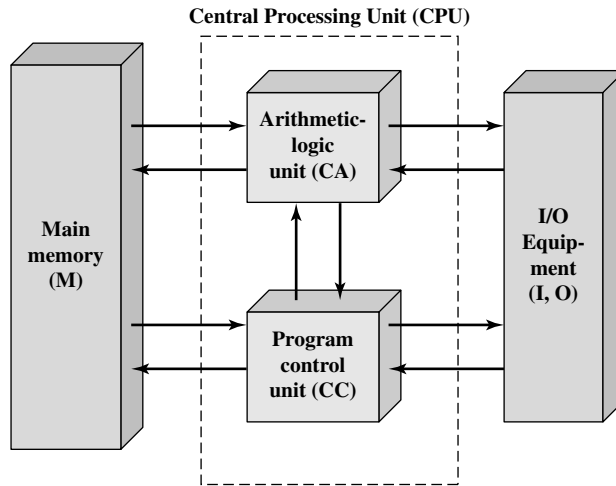


Figure 2.1 Structure of the IAS Computer

- A control unit, which interprets the instructions in memory and causes them to be executed
- Input and output (I/O) equipment operated by the control unit

This structure was outlined in von Neumann's earlier proposal, which is worth quoting at this point [VONN45]:

2.2 First: Because the device is primarily a computer, it will have to perform the elementary operations of arithmetic most frequently. These are addition, subtraction, multiplication and division. It is therefore reasonable that it should contain specialized organs for just these operations.

It must be observed, however, that while this principle as such is probably sound, the specific way in which it is realized requires close scrutiny. At any rate a *central arithmetical* part of the device will probably have to exist and this constitutes *the first specific part: CA*.

2.3 Second: The logical control of the device, that is, the proper sequencing of its operations, can be most efficiently carried out by a central control organ. If the device is to be *elastic*, that is, as nearly as possible *all purpose*, then a distinction must be made between the specific instructions given for and defining a particular problem, and the general control organs which see to it that these instructions—no matter what they are—are carried out. The former must be stored in some way; the latter are represented by definite operating parts of the device. By the *central control* we mean this latter function only, and the organs which perform it form *the second specific part: CC*.

2.4 **Third:** Any device which is to carry out long and complicated sequences of operations (specifically of calculations) must have a considerable memory . . .

(b) The instructions which govern a complicated problem may constitute considerable material, particularly so, if the code is circumstantial (which it is in most arrangements). This material must be remembered.

At any rate, the total *memory* constitutes the *third specific part of the device: M*.

2.6 The three specific parts CA, CC (together C), and M correspond to the *associative* neurons in the human nervous system. It remains to discuss the equivalents of the *sensory* or *afferent* and the *motor* or *efferent* neurons. These are the *input* and *output* organs of the device.

The device must be endowed with the ability to maintain input and output (sensory and motor) contact with some specific medium of this type. The medium will be called the *outside recording medium of the device: R*.

2.7 **Fourth:** The device must have organs to transfer . . . information from R into its specific parts C and M. These organs form its *input*, the *fourth specific part: I*. It will be seen that it is best to make all transfers from R (by I) into M and never directly from C.

2.8 **Fifth:** The device must have organs to transfer . . . from its specific parts C and M into R. These organs form its *output*, the *fifth specific part: O*. It will be seen that it is again best to make all transfers from M (by O) into R, and never directly from C.

With rare exceptions, all of today's computers have this same general structure and function and are thus referred to as von Neumann machines. Thus, it is worthwhile at this point to describe briefly the operation of the IAS computer [BURK46]. Following [HAYE98], the terminology and notation of von Neumann are changed in the following to conform more closely to modern usage; the examples and illustrations accompanying this discussion are based on that latter text.

The memory of the IAS consists of 1000 storage locations, called *words*, of 40 binary digits (bits) each.² Both data and instructions are stored there. Numbers are represented in binary form, and each instruction is a binary code. Figure 2.2 illustrates these formats. Each number is represented by a sign bit and a 39-bit value. A word may also contain two 20-bit instructions, with each instruction consisting of an 8-bit operation code (opcode) specifying the operation to be performed and a 12-bit address designating one of the words in memory (numbered from 0 to 999).

The control unit operates the IAS by fetching instructions from memory and executing them one at a time. To explain this, a more detailed structure diagram is

²There is no universal definition of the term *word*. In general, a word is an ordered set of bytes or bits that is the normal unit in which information may be stored, transmitted, or operated on within a given computer. Typically, if a processor has a fixed-length instruction set, then the instruction length equals the word length.

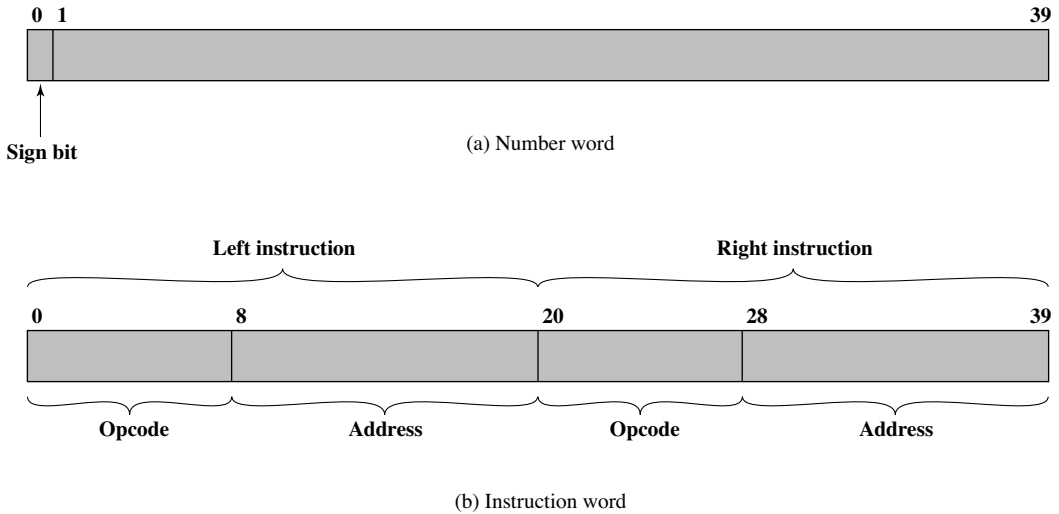


Figure 2.2 IAS Memory Formats

needed, as indicated in Figure 2.3. This figure reveals that both the control unit and the ALU contain storage locations, called *registers*, defined as follows:

- **Memory buffer register (MBR):** Contains a word to be stored in memory or sent to the I/O unit, or is used to receive a word from memory or from the I/O unit.
- **Memory address register (MAR):** Specifies the address in memory of the word to be written from or read into the MBR.
- **Instruction register (IR):** Contains the 8-bit opcode instruction being executed.
- **Instruction buffer register (IBR):** Employed to hold temporarily the right-hand instruction from a word in memory.
- **Program counter (PC):** Contains the address of the next instruction-pair to be fetched from memory.
- **Accumulator (AC) and multiplier quotient (MQ):** Employed to hold temporarily operands and results of ALU operations. For example, the result of multiplying two 40-bit numbers is an 80-bit number; the most significant 40 bits are stored in the AC and the least significant in the MQ.

The IAS operates by repetitively performing an *instruction cycle*, as shown in Figure 2.4. Each instruction cycle consists of two subcycles. During the *fetch cycle*, the opcode of the next instruction is loaded into the IR and the address portion is loaded into the MAR. This instruction may be taken from the IBR, or it can be obtained from memory by loading a word into the MBR, and then down to the IBR, IR, and MAR.

Why the indirection? These operations are controlled by electronic circuitry and result in the use of data paths. To simplify the electronics, there is only one

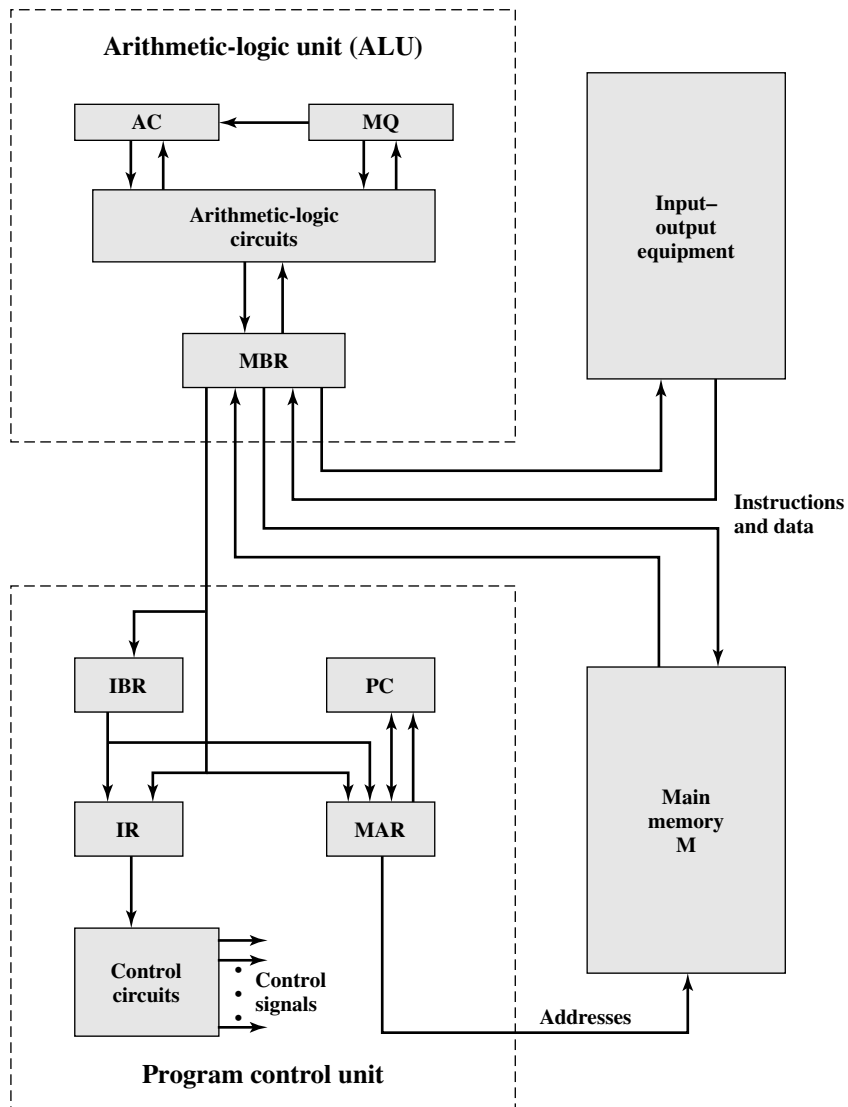


Figure 2.3 Expanded Structure of IAS Computer

register that is used to specify the address in memory for a read or write and only one register used for the source or destination.

Once the opcode is in the IR, the *execute cycle* is performed. Control circuitry interprets the opcode and executes the instruction by sending out the appropriate control signals to cause data to be moved or an operation to be performed by the ALU.

The IAS computer had a total of 21 instructions, which are listed in Table 2.1. These can be grouped as follows:

- **Data transfer:** Move data between memory and ALU registers or between two ALU registers.

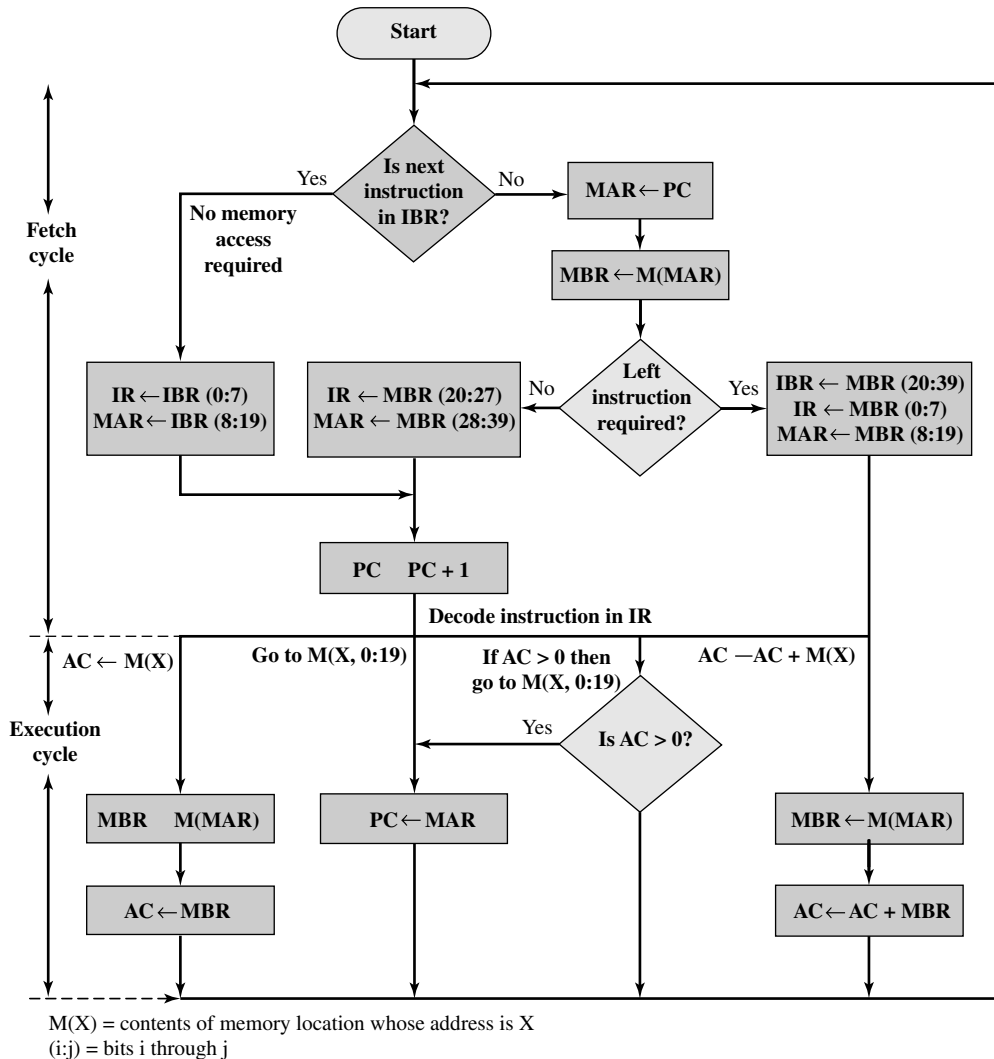


Figure 2.4 Partial Flowchart of IAS Operation

- **Unconditional branch:** Normally, the control unit executes instructions in sequence from memory. This sequence can be changed by a branch instruction, which facilitates repetitive operations.
- **Conditional branch:** The branch can be made dependent on a condition, thus allowing decision points.
- **Arithmetic:** Operations performed by the ALU.
- **Address modify:** Permits addresses to be computed in the ALU and then inserted into instructions stored in memory. This allows a program considerable addressing flexibility.

Table 2.1 The IAS Instruction Set

Instruction Type	Opcode	Symbolic Representation	Description
Data transfer	00001010	LOAD MQ	Transfer contents of register MQ to the accumulator AC
	00001001	LOAD MQ,M(X)	Transfer contents of memory location X to MQ
	00100001	STOR M(X)	Transfer contents of accumulator to memory location X
	00000001	LOAD M(X)	Transfer M(X) to the accumulator
	00000010	LOAD -M(X)	Transfer -M(X) to the accumulator
	00000011	LOAD M(X)	Transfer absolute value of M(X) to the accumulator
	00000100	LOAD - M(X)	Transfer - M(X) to the accumulator
Unconditional branch	00001101	JUMP M(X,0:19)	Take next instruction from left half of M(X)
	00001110	JUMP M(X,20:39)	Take next instruction from right half of M(X)
Conditional branch	00001111	JUMP+ M(X,0:19)	If number in the accumulator is nonnegative, take next instruction from left half of M(X)
	00010000	JUMP+ M(X,20:39)	If number in the accumulator is nonnegative, take next instruction from right half of M(X)
Arithmetic	00000101	ADD M(X)	Add M(X) to AC; put the result in AC
	00000111	ADD M(X)	Add M(X) to AC; put the result in AC
	00000110	SUB M(X)	Subtract M(X) from AC; put the result in AC
	00001000	SUB M(X)	Subtract M(X) from AC; put the remainder in AC
	00001011	MUL M(X)	Multiply M(X) by MQ; put most significant bits of result in AC, put least significant bits in MQ
	00001100	DIV M(X)	Divide AC by M(X); put the quotient in MQ and the remainder in AC
	00010100	LSH	Multiply accumulator by 2; i.e., shift left one bit position
	00010101	RSH	Divide accumulator by 2; i.e., shift right one position
Address modify	00010010	STOR M(X,8:19)	Replace left address field at M(X) by 12 rightmost bits of AC
	00010011	STOR M(X,28:39)	Replace right address field at M(X) by 12 rightmost bits of AC

Table 2.1 presents instructions in a symbolic, easy-to-read form. Actually, each instruction must conform to the format of Figure 2.2b. The opcode portion (first 8 bits) specifies which of the 21 instructions is to be executed. The address portion (remaining 12 bits) specifies which of the 1000 memory locations is to be involved in the execution of the instruction.

Figure 2.4 shows several examples of instruction execution by the control unit. Note that each operation requires several steps. Some of these are quite elaborate. The multiplication operation requires 39 suboperations, one for each bit position except that of the sign bit.

COMMERCIAL COMPUTERS The 1950s saw the birth of the computer industry with two companies, Sperry and IBM, dominating the marketplace.

In 1947, Eckert and Mauchly formed the Eckert-Mauchly Computer Corporation to manufacture computers commercially. Their first successful machine was the UNIVAC I (Universal Automatic Computer), which was commissioned by the Bureau of the Census for the 1950 calculations. The Eckert-Mauchly Computer Corporation became part of the UNIVAC division of Sperry-Rand Corporation, which went on to build a series of successor machines.

The UNIVAC I was the first successful commercial computer. It was intended for both scientific and commercial applications. The first paper describing the system listed matrix algebraic computations, statistical problems, premium billings for a life insurance company, and logistical problems as a sample of the tasks it could perform.

The UNIVAC II, which had greater memory capacity and higher performance than the UNIVAC I, was delivered in the late 1950s and illustrates several trends that have remained characteristic of the computer industry. First, advances in technology allow companies to continue to build larger, more powerful computers. Second, each company tries to make its new machines *backward compatible*³ with the older machines. This means that the programs written for the older machines can be executed on the new machine. This strategy is adopted in the hopes of retaining the customer base; that is, when a customer decides to buy a newer machine, he or she is likely to get it from the same company to avoid losing the investment in programs.

The UNIVAC division also began development of the 1100 series of computers, which was to be its major source of revenue. This series illustrates a distinction that existed at one time. The first model, the UNIVAC 1103, and its successors for many years were primarily intended for scientific applications, involving long and complex calculations. Other companies concentrated on business applications, which involved processing large amounts of text data. This split has largely disappeared, but it was evident for a number of years.

IBM, then the major manufacturer of punched-card processing equipment, delivered its first electronic stored-program computer, the 701, in 1953. The 701 was intended primarily for scientific applications [BASH81]. In 1955, IBM introduced the companion 702 product, which had a number of hardware features that suited it to business applications. These were the first of a long series of 700/7000 computers that established IBM as the overwhelmingly dominant computer manufacturer.

The Second Generation: Transistors

The first major change in the electronic computer came with the replacement of the vacuum tube by the transistor. The transistor is smaller, cheaper, and dissipates less heat than a vacuum tube but can be used in the same way as a vacuum tube to construct computers. Unlike the vacuum tube, which requires wires, metal plates, a glass capsule, and a vacuum, the transistor is a *solid-state device*, made from silicon.

The transistor was invented at Bell Labs in 1947 and by the 1950s had launched an electronic revolution. It was not until the late 1950s, however, that fully transistorized computers were commercially available. IBM again was not the first

³Also called *downward compatible*. The same concept, from the point of view of the older system, is referred to as *upward compatible*, or forward compatible.

Table 2.2 Computer Generations

Generation	Approximate Dates	Technology	Typical Speed (operations per second)
1	1946–1957	Vacuum tube	40,000
2	1958–1964	Transistor	200,000
3	1965–1971	Small and medium scale integration	1,000,000
4	1972–1977	Large scale integration	10,000,000
5	1978–1991	Very large scale integration	100,000,000
6	1991–	Ultra large scale integration	1,000,000,000

company to deliver the new technology. NCR and, more successfully, RCA were the front-runners with some small transistor machines. IBM followed shortly with the 7000 series.

The use of the transistor defines the *second generation* of computers. It has become widely accepted to classify computers into generations based on the fundamental hardware technology employed (Table 2.2). Each new generation is characterized by greater processing performance, larger memory capacity, and smaller size than the previous one.

But there are other changes as well. The second generation saw the introduction of more complex arithmetic and logic units and control units, the use of high-level programming languages, and the provision of *system software* with the computer.

The second generation is noteworthy also for the appearance of the Digital Equipment Corporation (DEC). DEC was founded in 1957 and, in that year, delivered its first computer, the PDP-1. This computer and this company began the mini-computer phenomenon that would become so prominent in the third generation.

THE IBM 7094 From the introduction of the 700 series in 1952 to the introduction of the last member of the 7000 series in 1964, this IBM product line underwent an evolution that is typical of computer products. Successive members of the product line show increased performance, increased capacity, and/or lower cost.

Table 2.3 illustrates this trend. The size of main memory, in multiples of 2^{10} 36-bit words, grew from 2K ($1K = 2^{10}$) to 32K words,⁴ while the time to access one word of memory, the *memory cycle time*, fell from 30 μ s to 1.4 μ s. The number of opcodes grew from a modest 24 to 185.

The final column indicates the relative execution speed of the central processing unit (CPU). Speed improvements are achieved by improved electronics (e.g., a transistor implementation is faster than a vacuum tube implementation) and more complex circuitry. For example, the IBM 7094 includes an Instruction Backup Register, used to buffer the next instruction. The control unit fetches two adjacent words

⁴A discussion of the uses of numerical prefixes, such as kilo and giga, is contained in a supporting document at the Computer Science Student Resource Site at WilliamStallings.com/StudentSupport.html.

Table 2.3 Example members of the IBM 700/7000 Series

Model Number	First Delivery	CPU Technology	Memory Technology	Cycle Time (μs)	Memory Size (K)	Number of Opcodes	Number of Index Registers	Hardwired Floating-Point	I/O Overlap (Channels)	Instruction Fetch Overlap	Speed (relative to 701)
701	1952	Vacuum tubes	Electrostatic tubes	30	2–4	24	0	no	no	no	1
704	1955	Vacuum tubes	Core	12	4–32	80	3	yes	no	no	2.5
709	1958	Vacuum tubes	Core	12	32	140	3	yes	yes	no	4
7090	1960	Transistor	Core	2.18	32	169	3	yes	yes	no	25
7094 I	1962	Transistor	Core	2	32	185	7	yes (double precision)	yes	yes	30
7094 II	1964	Transistor	Core	1.4	32	185	7	yes (double precision)	yes	yes	50

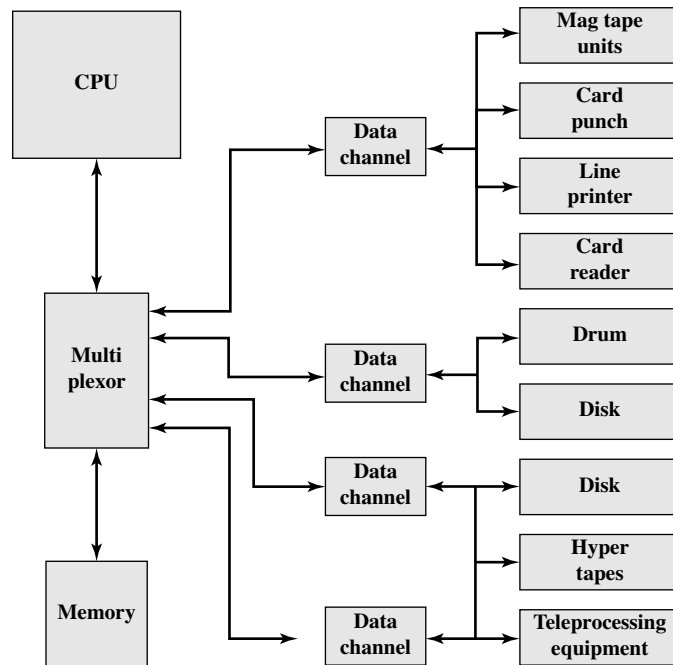


Figure 2.5 An IBM 7094 Configuration

from memory for an instruction fetch. Except for the occurrence of a branching instruction, which is typically infrequent, this means that the control unit has to access memory for an instruction on only half the instruction cycles. This prefetching significantly reduces the average instruction cycle time.

The remainder of the columns of Table 2.3 will become clear as the text proceeds.

Figure 2.5 shows a large (many peripherals) configuration for an IBM 7094, which is representative of second-generation computers [BELL71]. Several differences from the IAS computer are worth noting. The most important of these is the use of *data channels*. A data channel is an independent I/O module with its own processor and its own instruction set. In a computer system with such devices, the CPU does not execute detailed I/O instructions. Such instructions are stored in a main memory to be executed by a special-purpose processor in the data channel itself. The CPU initiates an I/O transfer by sending a control signal to the data channel, instructing it to execute a sequence of instructions in memory. The data channel performs its task independently of the CPU and signals the CPU when the operation is complete. This arrangement relieves the CPU of a considerable processing burden.

Another new feature is the *multiplexor*, which is the central termination point for data channels, the CPU, and memory. The multiplexor schedules access to the memory from the CPU and data channels, allowing these devices to act independently.

The Third Generation: Integrated Circuits

A single, self-contained transistor is called a *discrete component*. Throughout the 1950s and early 1960s, electronic equipment was composed largely of discrete

components—transistors, resistors, capacitors, and so on. Discrete components were manufactured separately, packaged in their own containers, and soldered or wired together onto masonite-like circuit boards, which were then installed in computers, oscilloscopes, and other electronic equipment. Whenever an electronic device called for a transistor, a little tube of metal containing a pinhead-sized piece of silicon had to be soldered to a circuit board. The entire manufacturing process, from transistor to circuit board, was expensive and cumbersome.

These facts of life were beginning to create problems in the computer industry. Early second-generation computers contained about 10,000 transistors. This figure grew to the hundreds of thousands, making the manufacture of newer, more powerful machines increasingly difficult.

In 1958 came the achievement that revolutionized electronics and started the era of microelectronics: the invention of the integrated circuit. It is the integrated circuit that defines the third generation of computers. In this section we provide a brief introduction to the technology of integrated circuits. Then we look at perhaps the two most important members of the third generation, both of which were introduced at the beginning of that era: the IBM System/360 and the DEC PDP-8.

MICROELECTRONICS Microelectronics means, literally, “small electronics.” Since the beginnings of digital electronics and the computer industry, there has been a persistent and consistent trend toward the reduction in size of digital electronic circuits. Before examining the implications and benefits of this trend, we need to say something about the nature of digital electronics. A more detailed discussion is found in Chapter 20.

The basic elements of a digital computer, as we know, must perform storage, movement, processing, and control functions. Only two fundamental types of components are required (Figure 2.6): gates and memory cells. A gate is a device that implements a simple Boolean or logical function, such as IF *A AND B ARE TRUE THEN C IS TRUE* (AND gate). Such devices are called gates because they control data flow in much the same way that canal gates do. The memory cell is a device that can store one bit of data; that is, the device can be in one of two stable states at any time. By interconnecting large numbers of these fundamental devices, we can construct a computer. We can relate this to our four basic functions as follows:

- **Data storage:** Provided by memory cells.
- **Data processing:** Provided by gates.

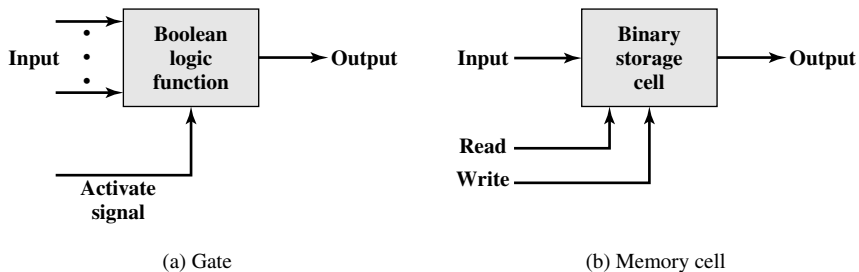


Figure 2.6 Fundamental Computer Elements

- **Data movement:** The paths among components are used to move data from memory to memory and from memory through gates to memory.
- **Control:** The paths among components can carry control signals. For example, a gate will have one or two data inputs plus a control signal input that activates the gate. When the control signal is ON, the gate performs its function on the data inputs and produces a data output. Similarly, the memory cell will store the bit that is on its input lead when the WRITE control signal is ON and will place the bit that is in the cell on its output lead when the READ control signal is ON.

Thus, a computer consists of gates, memory cells, and interconnections among these elements. The gates and memory cells are, in turn, constructed of simple digital electronic components.

The integrated circuit exploits the fact that such components as transistors, resistors, and conductors can be fabricated from a semiconductor such as silicon. It is merely an extension of the solid-state art to fabricate an entire circuit in a tiny piece of silicon rather than assemble discrete components made from separate pieces of silicon into the same circuit. Many transistors can be produced at the same time on a single wafer of silicon. Equally important, these transistors can be connected with a process of metallization to form circuits.

Figure 2.7 depicts the key concepts in an integrated circuit. A thin *wafer* of silicon is divided into a matrix of small areas, each a few millimeters square. The identical circuit pattern is fabricated in each area, and the wafer is broken up into *chips*. Each chip consists of many gates and/or memory cells plus a number of input and output attachment points. This chip is then packaged in housing that protects it and provides pins for attachment to devices beyond the chip. A number of these packages can then be interconnected on a printed circuit board to produce larger and more complex circuits.

Initially, only a few gates or memory cells could be reliably manufactured and packaged together. These early integrated circuits are referred to as *small-scale integration* (SSI). As time went on, it became possible to pack more and more components on the same chip. This growth in density is illustrated in Figure 2.8; it is one of the most remarkable technological trends ever recorded.⁵ This figure reflects the famous Moore's law, which was propounded by Gordon Moore, cofounder of Intel, in 1965 [MOOR65]. Moore observed that the number of transistors that could be put on a single chip was doubling every year and correctly predicted that this pace would continue into the near future. To the surprise of many, including Moore, the pace continued year after year and decade after decade. The pace slowed to a doubling every 18 months in the 1970s but has sustained that rate ever since.

The consequences of Moore's law are profound:

1. The cost of a chip has remained virtually unchanged during this period of rapid growth in density. This means that the cost of computer logic and memory circuitry has fallen at a dramatic rate.

⁵Note that the vertical axis uses a log scale. A basic review of log scales is in the math refresher document at the Computer Science Student Support Site at WilliamStallings.com/StudentSupport.html.

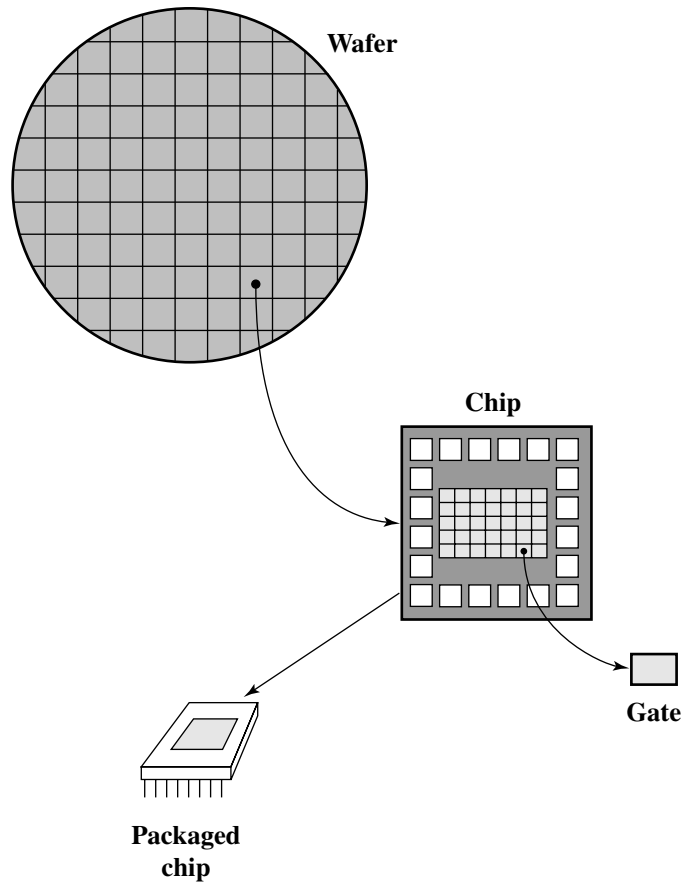


Figure 2.7 Relationship among Wafer, Chip, and Gate

2. Because logic and memory elements are placed closer together on more densely packed chips, the electrical path length is shortened, increasing operating speed.
3. The computer becomes smaller, making it more convenient to place in a variety of environments.
4. There is a reduction in power and cooling requirements.
5. The interconnections on the integrated circuit are much more reliable than solder connections. With more circuitry on each chip, there are fewer interchip connections.

IBM SYSTEM/360 By 1964, IBM had a firm grip on the computer market with its 7000 series of machines. In that year, IBM announced the System/360, a new family of computer products. Although the announcement itself was no surprise, it contained some unpleasant news for current IBM customers: the 360 product line was incompatible with older IBM machines. Thus, the transition to the 360 would be difficult for the current customer base. This was a bold step by IBM, but one IBM felt

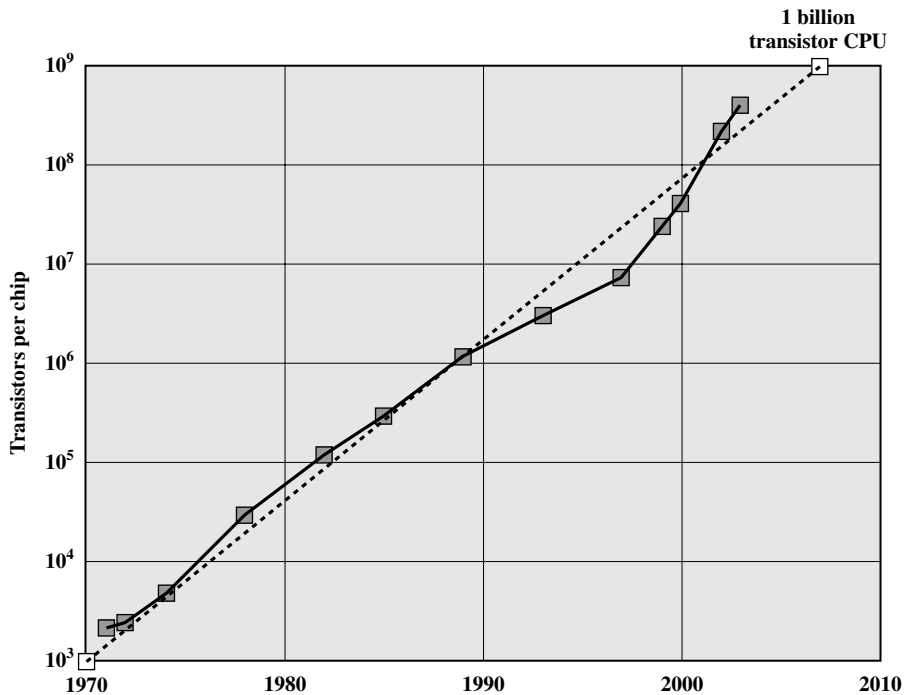


Figure 2.8 Growth in CPU Transistor Count [BOHR03]

was necessary to break out of some of the constraints of the 7000 architecture and to produce a system capable of evolving with the new integrated circuit technology [PADE81, GIFF87]. The strategy paid off both financially and technically. The 360 was the success of the decade and cemented IBM as the overwhelmingly dominant computer vendor, with a market share above 70%. And, with some modifications and extensions, the architecture of the 360 remains to this day the architecture of IBM's mainframe⁶ computers. Examples using this architecture can be found throughout this text.

The System/360 was the industry's first planned family of computers. The family covered a wide range of performance and cost. Table 2.4 indicates some of the key characteristics of the various models in 1965 (each member of the family is distinguished by a model number). The models were compatible in the sense that a program written for one model should be capable of being executed by another model in the series, with only a difference in the time it takes to execute.

The concept of a family of compatible computers was both novel and extremely successful. A customer with modest requirements and a budget to match could start with the relatively inexpensive Model 30. Later, if the customer's needs grew, it was possible to upgrade to a faster machine with more memory without

⁶The term *mainframe* is used for the larger, most powerful computers other than supercomputers. Typical characteristics of a mainframe are that it supports a large database, has elaborate I/O hardware, and is used in a central data processing facility.

Table 2.4 Key Characteristics of the System/360 Family

Characteristic	Model 30	Model 40	Model 50	Model 65	Model 75
Maximum memory size (bytes)	64K	256K	256K	512K	512K
Data rate from memory (Mbytes/sec)	0.5	0.8	2.0	8.0	16.0
Processor cycle time μ s)	1.0	0.625	0.5	0.25	0.2
Relative speed	1	3.5	10	21	50
Maximum number of data channels	3	3	4	6	6
Maximum data rate on one channel (Kbytes/s)	250	400	800	1250	1250

sacrificing the investment in already-developed software. The characteristics of a family are as follows:

- **Similar or identical instruction set:** In many cases, the exact same set of machine instructions is supported on all members of the family. Thus, a program that executes on one machine will also execute on any other. In some cases, the lower end of the family has an instruction set that is a subset of that of the top end of the family. This means that programs can move up but not down.
- **Similar or identical operating system:** The same basic operating system is available for all family members. In some cases, additional features are added to the higher-end members.
- **Increasing speed:** The rate of instruction execution increases in going from lower to higher family members.
- **Increasing number of I/O ports:** The number of I/O ports increases in going from lower to higher family members.
- **Increasing memory size:** The size of main memory increases in going from lower to higher family members.
- **Increasing cost:** At a given point in time, the cost of a system increases in going from lower to higher family members.

How could such a family concept be implemented? Differences were achieved based on three factors: basic speed, size, and degree of simultaneity [STEV64]. For example, greater speed in the execution of a given instruction could be gained by the use of more complex circuitry in the ALU, allowing suboperations to be carried out in parallel. Another way of increasing speed was to increase the width of the data path between main memory and the CPU. On the Model 30, only 1 byte (8 bits) could be fetched from main memory at a time, whereas 8 bytes could be fetched at a time on the Model 75.

The System/360 not only dictated the future course of IBM but also had a profound impact on the entire industry. Many of its features have become standard on other large computers.

DEC PDP-8 In the same year that IBM shipped its first System/360, another momentous first shipment occurred: PDP-8 from Digital Equipment Corporation

(DEC). At a time when the average computer required an air-conditioned room, the PDP-8 (dubbed a minicomputer by the industry, after the miniskirt of the day) was small enough that it could be placed on top of a lab bench or be built into other equipment. It could not do everything the mainframe could, but at \$16,000, it was cheap enough for each lab technician to have one. In contrast, the System/360 series of mainframe computers introduced just a few months before cost hundreds of thousands of dollars.

The low cost and small size of the PDP-8 enabled another manufacturer to purchase a PDP-8 and integrate it into a total system for resale. These other manufacturers came to be known as original equipment manufacturers (OEMs), and the OEM market became and remains a major segment of the computer marketplace.

The PDP-8 was an immediate hit and made DEC's fortune. This machine and other members of the PDP-8 family that followed it (see Table 2.5) achieved a production status formerly reserved for IBM computers, with about 50,000 machines sold over the next dozen years. As DEC's official history puts it, the PDP-8 "established the concept of minicomputers, leading the way to a multibillion dollar industry." It also established DEC as the number one minicomputer vendor, and, by the time the PDP-8 had reached the end of its useful life, DEC was the number two computer manufacturer, behind IBM.

In contrast to the central-switched architecture (Figure 2.5) used by IBM on its 700/7000 and 360 systems, later models of the PDP-8 used a structure that is now virtually universal for microcomputers: the bus structure. This is illustrated in Figure 2.9. The PDP-8 bus, called the Omnibus, consists of 96 separate signal paths, used to carry control, address, and data signals. Because all system components share a common set of signal paths, their use must be controlled by the CPU. This architecture is highly flexible, allowing modules to be plugged into the bus to create various configurations.

Later Generations

Beyond the third generation there is less general agreement on defining generations of computers. Table 2.2 suggests that there have been a number of later generations, based on advances in integrated circuit technology. With the introduction of large-scale integration (LSI), more than 1000 components can be placed on a single integrated circuit chip. Very-large-scale integration (VLSI) achieved more than 10,000 components per chip, while current ultra-large-scale integration (ULSI) chips can contain more than one million components.

With the rapid pace of technology, the high rate of introduction of new products, and the importance of software and communications as well as hardware, the classification by generation becomes less clear and less meaningful. It could be said that the commercial application of new developments resulted in a major change in the early 1970s and that the results of these changes are still being worked out. In this section, we mention two of the most important of these results.

SEMICONDUCTOR MEMORY The first application of integrated circuit technology to computers was construction of the processor (the control unit and the arithmetic and logic unit) out of integrated circuit chips. But it was also found that this same technology could be used to construct memories.

Table 2.5 Evolution of the PDP-8 [VOEL88]

Model	First Shipped	Cost of Processor + 4K 12-bit Words of Memory (\$1000s)	Data Rate from Memory (words/μsec)	Volume (cubic feet)	Innovations and Improvements
PDP-8	4/65	16.2	1.26	8.0	Automatic wire-wrapping production
PDP-8/5	9/66	8.79	0.08	3.2	Serial instruction implementation
PDP-8/1	4/68	11.6	1.34	8.0	Medium scale integrated circuits
PDP-8/L	11/68	7.0	1.26	2.0	Smaller cabinet
PDP-8/E	3/71	4.99	1.52	2.2	Omnibus
PDP-8/M	6/72	3.69	1.52	1.8	Half-size cabinet with fewer slots than 8/E
PDP-8/A	1/75	2.6	1.34	1.2	Semiconductor memory; floating-point processor

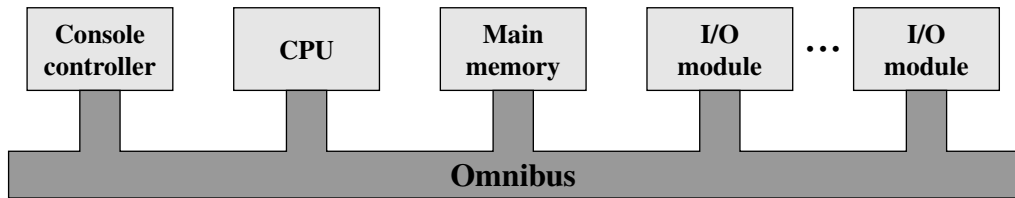


Figure 2.9 PDP-8 Bus Structure

In the 1950s and 1960s, most computer memory was constructed from tiny rings of ferromagnetic material, each about a sixteenth of an inch in diameter. These rings were strung up on grids of fine wires suspended on small screens inside the computer. Magnetized one way, a ring (called a *core*) represented a one; magnetized the other way, it stood for a zero. Magnetic-core memory was rather fast; it took as little as a millionth of a second to read a bit stored in memory. But it was expensive, bulky, and used destructive readout: The simple act of reading a core erased the data stored in it. It was therefore necessary to install circuits to restore the data as soon as it had been extracted.

Then, in 1970, Fairchild produced the first relatively capacious semiconductor memory. This chip, about the size of a single core, could hold 256 bits of memory. It was nondestructive and much faster than core. It took only 70 billionths of a second to read a bit. However, the cost per bit was higher than for that of core.

In 1974, a seminal event occurred: The price per bit of semiconductor memory dropped below the price per bit of core memory. Following this, there has been a continuing and rapid decline in memory cost accompanied by a corresponding increase in physical memory density. This has led the way to smaller, faster machines with memory sizes of larger and more expensive machines from just a few years earlier. Developments in memory technology, together with developments in processor technology to be discussed next, changed the nature of computers in less than a decade. Although bulky, expensive computers remain a part of the landscape, the computer has also been brought out to the “end user,” with office machines and personal computers.

Since 1970, semiconductor memory has been through 13 generations: 1K, 4K, 16K, 64K, 256K, 1M, 4M, 16M, 64M, 256M, 1G, 4G, and, as of this writing, 16 Gbits on a single chip ($1K = 2^{10}$, $1M = 2^{20}$, $1G = 2^{30}$). Each generation has provided four times the storage density of the previous generation, accompanied by declining cost per bit and declining access time.

MICROPROCESSORS Just as the density of elements on memory chips has continued to rise, so has the density of elements on processor chips. As time went on, more and more elements were placed on each chip, so that fewer and fewer chips were needed to construct a single computer processor.

A breakthrough was achieved in 1971, when Intel developed its 4004. The 4004 was the first chip to contain *all* of the components of a CPU on a single chip: The microprocessor was born.

The 4004 can add two 4-bit numbers and can multiply only by repeated addition. By today’s standards, the 4004 is hopelessly primitive, but it marked the beginning of a continuing evolution of microprocessor capability and power.

This evolution can be seen most easily in the number of bits that the processor deals with at a time. There is no clear-cut measure of this, but perhaps the best measure is the data bus width: the number of bits of data that can be brought into or sent out of the processor at a time. Another measure is the number of bits in the accumulator or in the set of general-purpose registers. Often, these measures coincide, but not always. For example, a number of microprocessors were developed that operate on 16-bit numbers in registers but can only read and write 8 bits at a time.

The next major step in the evolution of the microprocessor was the introduction in 1972 of the Intel 8008. This was the first 8-bit microprocessor and was almost twice as complex as the 4004.

Neither of these steps was to have the impact of the next major event: the introduction in 1974 of the Intel 8080. This was the first general-purpose microprocessor. Whereas the 4004 and the 8008 had been designed for specific applications, the 8080 was designed to be the CPU of a general-purpose microcomputer. Like the 8008, the 8080 is an 8-bit microprocessor. The 8080, however, is faster, has a richer instruction set, and has a large addressing capability.

About the same time, 16-bit microprocessors began to be developed. However, it was not until the end of the 1970s that powerful, general-purpose 16-bit microprocessors appeared. One of these was the 8086. The next step in this trend occurred in 1981, when both Bell Labs and Hewlett-Packard developed 32-bit, single-chip microprocessors. Intel introduced its own 32-bit microprocessor, the 80386, in 1985 (Table 2.6).

Table 2.6 Evolution of Intel Microprocessors

(a) 1970s Processors

	4004	8008	8080	8086	8088
Introduced	1971	1972	1974	1978	1979
Clock speeds	108 kHz	108 kHz	2 MHz	5 MHz, 8 MHz, 10 MHz	5 MHz, 8 MHz
Bus width	4 bits	8 bits	8 bits	16 bits	8 bits
Number of transistors	2,300	3,500	6,000	29,000	29,000
Feature size (μm)	10		6	3	6
Addressable memory	640 Bytes	16 KB	64 KB	1 MB	1 MB

(b) 1980s Processors

	80286	386TM DX	386TM SX	486TM DX CPU
Introduced	1982	1985	1988	1989
Clock speeds	6 MHz–12.5 MHz	16 MHz–33 MHz	16 MHz–33 MHz	25 MHz–50 MHz
Bus width	16 bits	32 bits	16 bits	32 bits
Number of transistors	134,000	275,000	275,000	1.2 million
Feature size (μm)	1.5	1	1	0.8–1
Addressable memory	16 MB	4 GB	16 MB	4 GB
Virtual memory	1 GB	64 TB	64 TB	64 TB
Cache	—	—	—	8 kB

Table 2.6 Continued

(c) 1990s Processors

	486TM SX	Pentium	Pentium Pro	Pentium II
Introduced	1991	1993	1995	1997
Clock speeds	16 MHz–33 MHz	60 MHz–166 MHz,	150 MHz–200 MHz	200 MHz–300 MHz
Bus width	32 bits	32 bits	64 bits	64 bits
Number of transistors	1.185 million	3.1 million	5.5 million	7.5 million
Feature size (μm)	1	0.8	0.6	0.35
Addressable memory	4 GB	4 GB	64 GB	64 GB
Virtual memory	64 TB	64 TB	64 TB	64 TB
Cache	8 kB	8 kB	512 kB L1 and 1 MB L2	512 kB L2

(d) Recent Processors

	Pentium III	Pentium 4	Core 2 Duo	Core 2 Quad
Introduced	1999	2000	2006	2008
Clock speeds	450–660 MHz	1.3–1.8 GHz	1.06–1.2 GHz	3 GHz
Bus width	64 bits	64 bits	64 bits	64 bits
Number of transistors	9.5 million	42 million	167 million	820 million
Feature size (nm)	250	180	65	45
Addressable memory	64 GB	64 GB	64 GB	64 GB
Virtual memory	64 TB	64 TB	64 TB	64 TB
Cache	512 kB L2	256 kB L2	2 MB L2	6 MB L2

2.2 DESIGNING FOR PERFORMANCE

Year by year, the cost of computer systems continues to drop dramatically, while the performance and capacity of those systems continue to rise equally dramatically. At a local warehouse club, you can pick up a personal computer for less than \$1000 that packs the wallop of an IBM mainframe from 10 years ago. Thus, we have virtually “free” computer power. And this continuing technological revolution has enabled the development of applications of astounding complexity and power. For example, desktop applications that require the great power of today’s microprocessor-based systems include

- Image processing
- Speech recognition
- Videoconferencing
- Multimedia authoring
- Voice and video annotation of files
- Simulation modeling

Workstation systems now support highly sophisticated engineering and scientific applications, as well as simulation systems, and have the ability to support image and video applications. In addition, businesses are relying on increasingly powerful servers to handle transaction and database processing and to support massive client/server networks that have replaced the huge mainframe computer centers of yesteryear.

What is fascinating about all this from the perspective of computer organization and architecture is that, on the one hand, the basic building blocks for today's computer miracles are virtually the same as those of the IAS computer from over 50 years ago, while on the other hand, the techniques for squeezing the last iota of performance out of the materials at hand have become increasingly sophisticated.

This observation serves as a guiding principle for the presentation in this book. As we progress through the various elements and components of a computer, two objectives are pursued. First, the book explains the fundamental functionality in each area under consideration, and second, the book explores those techniques required to achieve maximum performance. In the remainder of this section, we highlight some of the driving factors behind the need to design for performance.

Microprocessor Speed

What gives Intel x86 processors or IBM mainframe computers such mind-boggling power is the relentless pursuit of speed by processor chip manufacturers. The evolution of these machines continues to bear out Moore's law, mentioned previously. So long as this law holds, chipmakers can unleash a new generation of chips every three years—with four times as many transistors. In memory chips, this has quadrupled the capacity of dynamic random-access memory (DRAM), still the basic technology for computer main memory, every three years. In microprocessors, the addition of new circuits, and the speed boost that comes from reducing the distances between them, has improved performance four- or fivefold every three years or so since Intel launched its x86 family in 1978.

But the raw speed of the microprocessor will not achieve its potential unless it is fed a constant stream of work to do in the form of computer instructions. Anything that gets in the way of that smooth flow undermines the power of the processor. Accordingly, while the chipmakers have been busy learning how to fabricate chips of greater and greater density, the processor designers must come up with ever more elaborate techniques for feeding the monster. Among the techniques built into contemporary processors are the following:

- **Branch prediction:** The processor looks ahead in the instruction code fetched from memory and predicts which branches, or groups of instructions, are likely to be processed next. If the processor guesses right most of the time, it can prefetch the correct instructions and buffer them so that the processor is kept busy. The more sophisticated examples of this strategy predict not just the next branch but multiple branches ahead. Thus, branch prediction increases the amount of work available for the processor to execute.
- **Data flow analysis:** The processor analyzes which instructions are dependent on each other's results, or data, to create an optimized schedule of instructions.

In fact, instructions are scheduled to be executed when ready, independent of the original program order. This prevents unnecessary delay.

- **Speculative execution:** Using branch prediction and data flow analysis, some processors speculatively execute instructions ahead of their actual appearance in the program execution, holding the results in temporary locations. This enables the processor to keep its execution engines as busy as possible by executing instructions that are likely to be needed.

These and other sophisticated techniques are made necessary by the sheer power of the processor. They make it possible to exploit the raw speed of the processor.

Performance Balance

While processor power has raced ahead at breakneck speed, other critical components of the computer have not kept up. The result is a need to look for performance balance: an adjusting of the organization and architecture to compensate for the mismatch among the capabilities of the various components.

Nowhere is the problem created by such mismatches more critical than in the interface between processor and main memory. Consider the history depicted in Figure 2.10. While processor speed has grown rapidly, the speed with which data can be transferred between main memory and the processor has lagged badly. The interface between processor and main memory is the most crucial pathway in the entire computer because it is responsible for carrying a constant flow of program instructions and data between memory chips and the processor. If memory or the pathway fails to keep pace with the processor's insistent demands, the processor stalls in a wait state, and valuable processing time is lost.

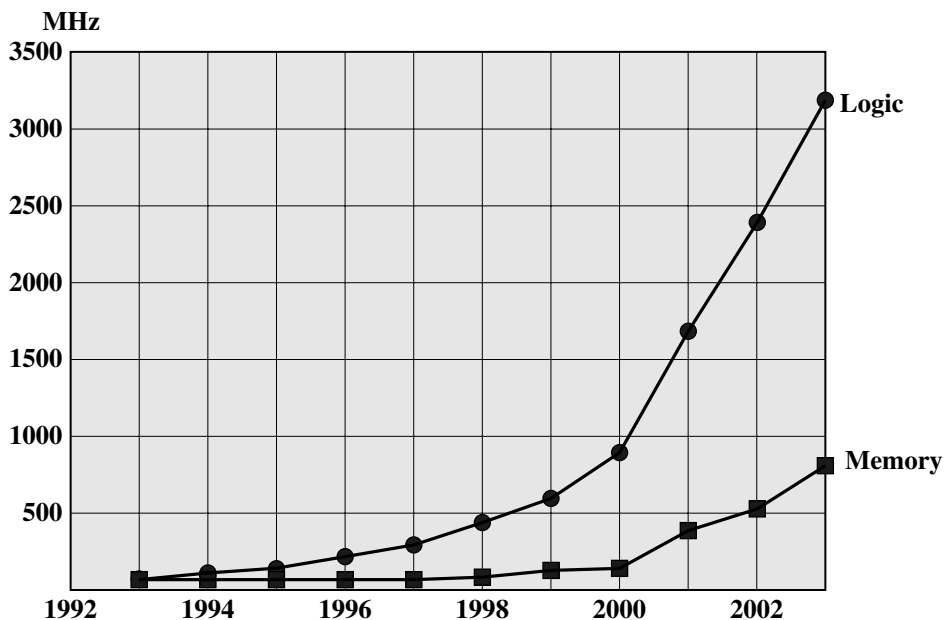


Figure 2.10 Logic and Memory Performance Gap [BORK03]

There are a number of ways that a system architect can attack this problem, all of which are reflected in contemporary computer designs. Consider the following examples:

- Increase the number of bits that are retrieved at one time by making DRAMs “wider” rather than “deeper” and by using wide bus data paths.
- Change the DRAM interface to make it more efficient by including a cache⁷ or other buffering scheme on the DRAM chip.
- Reduce the frequency of memory access by incorporating increasingly complex and efficient cache structures between the processor and main memory. This includes the incorporation of one or more caches on the processor chip as well as on an off-chip cache close to the processor chip.
- Increase the interconnect bandwidth between processors and memory by using higher-speed buses and by using a hierarchy of buses to buffer and structure data flow.

Another area of design focus is the handling of I/O devices. As computers become faster and more capable, more sophisticated applications are developed that support the use of peripherals with intensive I/O demands. Figure 2.11 gives some

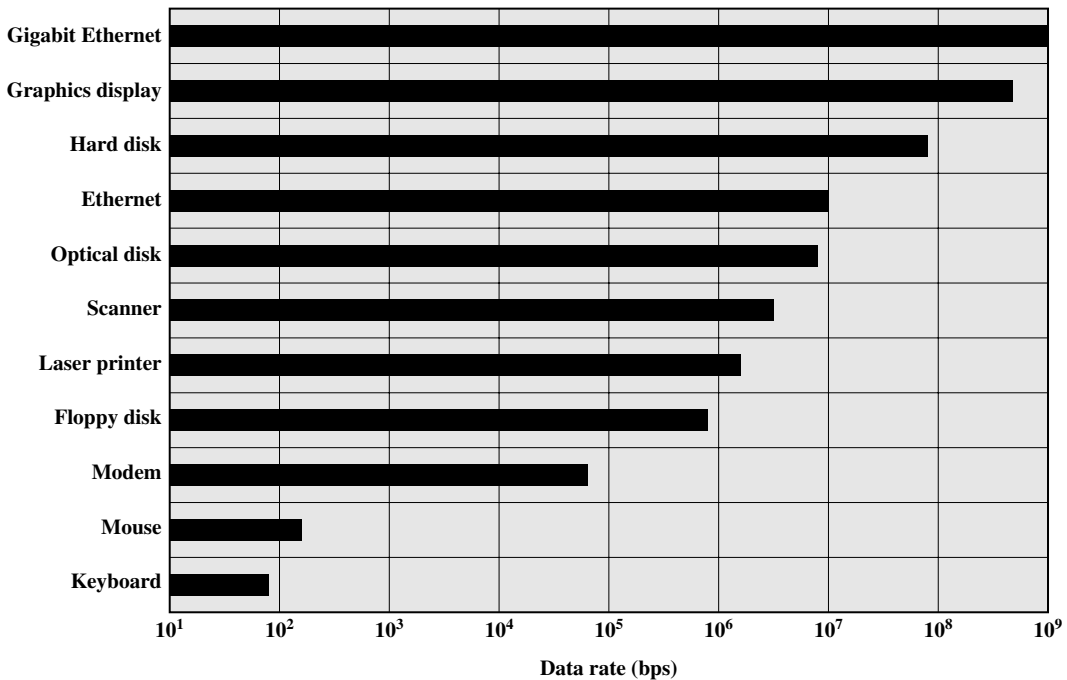


Figure 2.11 Typical I/O Device Data Rates⁷

⁷A cache is a relatively small fast memory interposed between a larger, slower memory and the logic that accesses the larger memory. The cache holds recently accessed data, and is designed to speed up subsequent access to the same data. Caches are discussed in Chapter 4.

examples of typical peripheral devices in use on personal computers and workstations. These devices create tremendous data throughput demands. While the current generation of processors can handle the data pumped out by these devices, there remains the problem of getting that data moved between processor and peripheral. Strategies here include caching and buffering schemes plus the use of higher-speed interconnection buses and more elaborate structures of buses. In addition, the use of multiple-processor configurations can aid in satisfying I/O demands.

The key in all this is balance. Designers constantly strive to balance the throughput and processing demands of the processor components, main memory, I/O devices, and the interconnection structures. This design must constantly be rethought to cope with two constantly evolving factors:

- The rate at which performance is changing in the various technology areas (processor, buses, memory, peripherals) differs greatly from one type of element to another.
- New applications and new peripheral devices constantly change the nature of the demand on the system in terms of typical instruction profile and the data access patterns.

Thus, computer design is a constantly evolving art form. This book attempts to present the fundamentals on which this art form is based and to present a survey of the current state of that art.

Improvements in Chip Organization and Architecture

As designers wrestle with the challenge of balancing processor performance with that of main memory and other computer components, the need to increase processor speed remains. There are three approaches to achieving increased processor speed:

- Increase the hardware speed of the processor. This increase is fundamentally due to shrinking the size of the logic gates on the processor chip, so that more gates can be packed together more tightly and to increasing the clock rate. With gates closer together, the propagation time for signals is significantly reduced, enabling a speeding up of the processor. An increase in clock rate means that individual operations are executed more rapidly.
- Increase the size and speed of caches that are interposed between the processor and main memory. In particular, by dedicating a portion of the processor chip itself to the cache, cache access times drop significantly.
- Make changes to the processor organization and architecture that increase the effective speed of instruction execution. Typically, this involves using parallelism in one form or another.

Traditionally, the dominant factor in performance gains has been in increases in clock speed due and logic density. Figure 2.12 illustrates this trend for Intel processor chips. However, as clock speed and logic density increase, a number of obstacles become more significant [INTE04b]:

- **Power:** As the density of logic and the clock speed on a chip increase, so does the power density (Watts/cm²). The difficulty of dissipating the heat generated

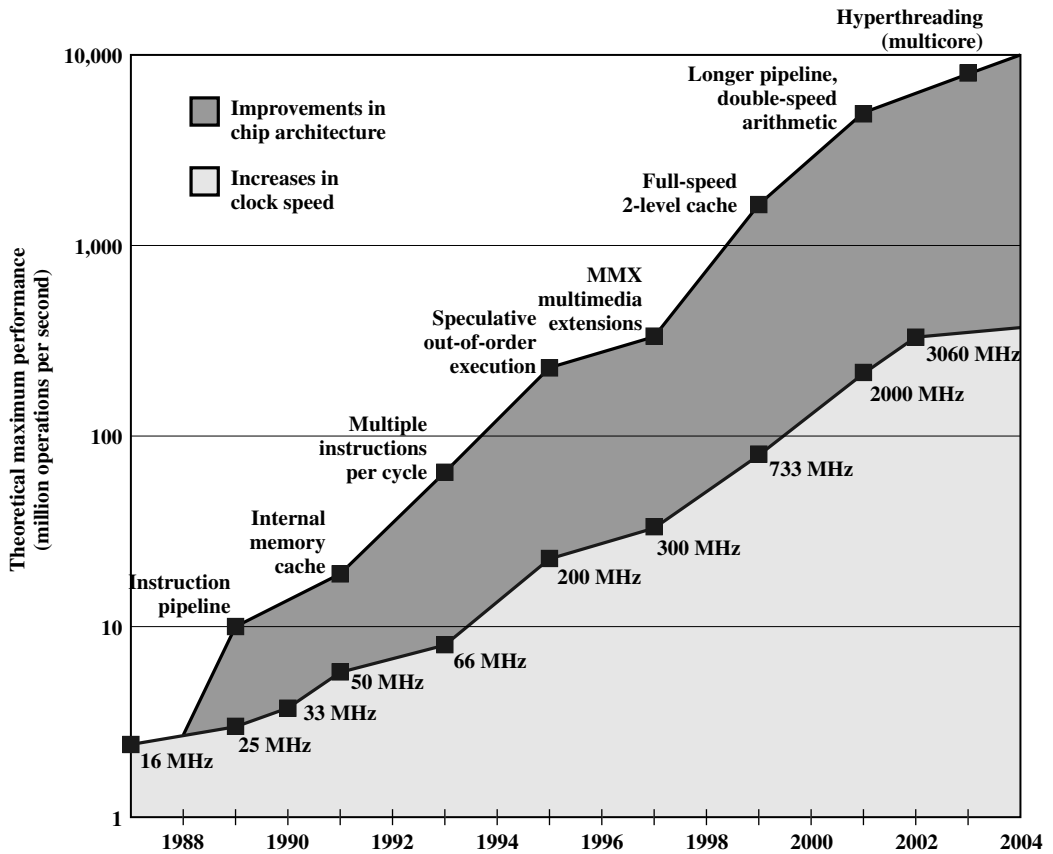


Figure 2.12 Intel Microprocessor Performance [GIBB04]

on high-density, high-speed chips is becoming a serious design issue ([GIBB04], [BORK03]).

- **RC delay:** The speed at which electrons can flow on a chip between transistors is limited by the resistance and capacitance of the metal wires connecting them; specifically, delay increases as the RC product increases. As components on the chip decrease in size, the wire interconnects become thinner, increasing resistance. Also, the wires are closer together, increasing capacitance.
- **Memory latency:** Memory speeds lag processor speeds, as previously discussed.

Thus, there will be more emphasis on organization and architectural approaches to improving performance. Figure 2.12 highlights the major changes that have been made over the years to increase the parallelism and therefore the computational efficiency of processors. These techniques are discussed in later chapters of the book.

Beginning in the late 1980s, and continuing for about 15 years, two main strategies have been used to increase performance beyond what can be achieved simply

by increasing clock speed. First, there has been an increase in cache capacity. There are now typically two or three levels of cache between the processor and main memory. As chip density has increased, more of the cache memory has been incorporated on the chip, enabling faster cache access. For example, the original Pentium chip devoted about 10% of on-chip area to a cache. The most recent Pentium 4 chip devotes about half of the chip area to caches.

Second, the instruction execution logic within a processor has become increasingly complex to enable parallel execution of instructions within the processor. Two noteworthy design approaches have been pipelining and superscalar. A pipeline works much as an assembly line in a manufacturing plant enabling different stages of execution of different instructions to occur at the same time along the pipeline. A superscalar approach in essence allows multiple pipelines within a single processor so that instructions that do not depend on one another can be executed in parallel.

Both of these approaches are reaching a point of diminishing returns. The internal organization of contemporary processors is exceedingly complex and is able to squeeze a great deal of parallelism out of the instruction stream. It seems likely that further significant increases in this direction will be relatively modest [GIBB04]. With three levels of cache on the processor chip, each level providing substantial capacity, it also seems that the benefits from the cache are reaching a limit.

However, simply relying on increasing clock rate for increased performance runs into the power dissipation problem already referred to. The faster the clock rate, the greater the amount of power to be dissipated, and some fundamental physical limits are being reached.

With all of these difficulties in mind, designers have turned to a fundamentally new approach to improving performance: placing multiple processors on the same chip, with a large shared cache. The use of multiple processors on the same chip, also referred to as multiple cores, or **multicore**, provides the potential to increase performance without increasing the clock rate. Studies indicate that, within a processor, the increase in performance is roughly proportional to the square root of the increase in complexity [BORK03]. But if the software can support the effective use of multiple processors, then doubling the number of processors almost doubles performance. Thus, the strategy is to use two simpler processors on the chip rather than one more complex processor.

In addition, with two processors, larger caches are justified. This is important because the power consumption of memory logic on a chip is much less than that of processing logic. In coming years, we can expect that most new processor chips will have multiple processors.

2.3 THE EVOLUTION OF THE INTEL x86 ARCHITECTURE

Throughout this book, we rely on many concrete examples of computer design and implementation to illustrate concepts and to illuminate trade-offs. Most of the time, the book relies on examples from two computer families: the Intel x86 and the ARM architecture. The current x86 offerings represent the results of decades of

design effort on complex instruction set computers (CISCs). The x86 incorporates the sophisticated design principles once found only on mainframes and supercomputers and serves as an excellent example of CISC design. An alternative approach to processor design in the reduced instruction set computer (RISC). The ARM architecture is used in a wide variety of embedded systems and is one of the most powerful and best-designed RISC-based systems on the market.

In this section and the next, we provide a brief overview of these two systems.

In terms of market share, Intel has ranked as the number one maker of microprocessors for non-embedded systems for decades, a position it seems unlikely to yield. The evolution of its flagship microprocessor product serves as a good indicator of the evolution of computer technology in general.

Table 2.6 shows that evolution. Interestingly, as microprocessors have grown faster and much more complex, Intel has actually picked up the pace. Intel used to develop microprocessors one after another, every four years. But Intel hopes to keep rivals at bay by trimming a year or two off this development time, and has done so with the most recent x86 generations.

It is worthwhile to list some of the highlights of the evolution of the Intel product line:

- **8080:** The world's first general-purpose microprocessor. This was an 8-bit machine, with an 8-bit data path to memory. The 8080 was used in the first personal computer, the Altair.
- **8086:** A far more powerful, 16-bit machine. In addition to a wider data path and larger registers, the 8086 sported an instruction cache, or queue, that prefetches a few instructions before they are executed. A variant of this processor, the 8088, was used in IBM's first personal computer, securing the success of Intel. The 8086 is the first appearance of the x86 architecture.
- **80286:** This extension of the 8086 enabled addressing a 16-MByte memory instead of just 1 MByte.
- **80386:** Intel's first 32-bit machine, and a major overhaul of the product. With a 32-bit architecture, the 80386 rivaled the complexity and power of minicomputers and mainframes introduced just a few years earlier. This was the first Intel processor to support multitasking, meaning it could run multiple programs at the same time.
- **80486:** The 80486 introduced the use of much more sophisticated and powerful cache technology and sophisticated instruction pipelining. The 80486 also offered a built-in math coprocessor, offloading complex math operations from the main CPU.
- **Pentium:** With the Pentium, Intel introduced the use of superscalar techniques, which allow multiple instructions to execute in parallel.
- **Pentium Pro:** The Pentium Pro continued the move into superscalar organization begun with the Pentium, with aggressive use of register renaming, branch prediction, data flow analysis, and speculative execution.
- **Pentium II:** The Pentium II incorporated Intel MMX technology, which is designed specifically to process video, audio, and graphics data efficiently.

- **Pentium III:** The Pentium III incorporates additional floating-point instructions to support 3D graphics software.
- **Pentium 4:** The Pentium 4 includes additional floating-point and other enhancements for multimedia.⁸
- **Core:** This is the first Intel x86 microprocessor with a dual core, referring to the implementation of two processors on a single chip.
- **Core 2:** The Core 2 extends the architecture to 64 bits. The Core 2 Quad provides four processors on a single chip.

Over 30 years after its introduction in 1978, the x86 architecture continues to dominate the processor market outside of embedded systems. Although the organization and technology of the x86 machines has changed dramatically over the decades, the instruction set architecture has evolved to remain backward compatible with earlier versions. Thus, any program written on an older version of the x86 architecture can execute on newer versions. All changes to the instruction set architecture have involved additions to the instruction set, with no subtractions. The rate of change has been the addition of roughly one instruction per month added to the architecture over the 30 years [ANTH08], so that there are now over 500 instructions in the instruction set.

The x86 provides an excellent illustration of the advances in computer hardware over the past 30 years. The 1978 8086 was introduced with a clock speed of 5 MHz and had 29,000 transistors. A quad-core Intel Core 2 introduced in 2008 operates at 3 GHz, a speedup of a factor of 600, and has 820 million transistors, about 28,000 times as many as the 8086. Yet the Core 2 is in only a slightly larger package than the 8086 and has a comparable cost.

2.4 EMBEDDED SYSTEMS AND THE ARM

The ARM architecture refers to a processor architecture that has evolved from RISC design principles and is used in embedded systems. Chapter 13 examines RISC design principles in detail. In this section, we give a brief overview of the concept of embedded systems, and then look at the evolution of the ARM.

Embedded Systems

The term *embedded system* refers to the use of electronics and software within a product, as opposed to a general-purpose computer, such as a laptop or desktop system. The following is a good general definition:⁹

Embedded system. A combination of computer hardware and software, and perhaps additional mechanical or other parts, designed to perform a dedicated function. In many cases, embedded systems are part of a larger system or product, as in the case of an antilock braking system in a car.

⁸With the Pentium 4, Intel switched from Roman numerals to Arabic numerals for model numbers.

⁹Michael Barr, *Embedded Systems Glossary*. Netrino Technical Library. <http://www.netrino.com/Publications/Glossary/index.php>

Table 2.7 Examples of Embedded Systems and Their Markets [NOER05]

Market	Embedded Device
Automotive	Ignition system Engine control Brake system
Consumer electronics	Digital and analog televisions Set-top boxes (DVDs, VCRs, Cable boxes) Personal digital assistants (PDAs) Kitchen appliances (refrigerators, toasters, microwave ovens) Automobiles Toys/games Telephones/cell phones/pagers Cameras Global positioning systems
Industrial control	Robotics and controls systems for manufacturing Sensors
Medical	Infusion pumps Dialysis machines Prosthetic devices Cardiac monitors
Office automation	Fax machine Photocopier Printers Monitors Scanners

Embedded systems far outnumber general-purpose computer systems, encompassing a broad range of applications (Table 2.7). These systems have widely varying requirements and constraints, such as the following [GRIM05]:

- Small to large systems, implying very different cost constraints, thus different needs for optimization and reuse
- Relaxed to very strict requirements and combinations of different quality requirements, for example, with respect to safety, reliability, real-time, flexibility, and legislation
- Short to long life times
- Different environmental conditions in terms of, for example, radiation, vibrations, and humidity
- Different application characteristics resulting in static versus dynamic loads, slow to fast speed, compute versus interface intensive tasks, and/or combinations thereof
- Different models of computation ranging from discrete-event systems to those involving continuous time dynamics (usually referred to as hybrid systems)

Often, embedded systems are tightly coupled to their environment. This can give rise to real-time constraints imposed by the need to interact with the environment. Constraints, such as required speeds of motion, required precision of measurement, and required time durations, dictate the timing of software operations.

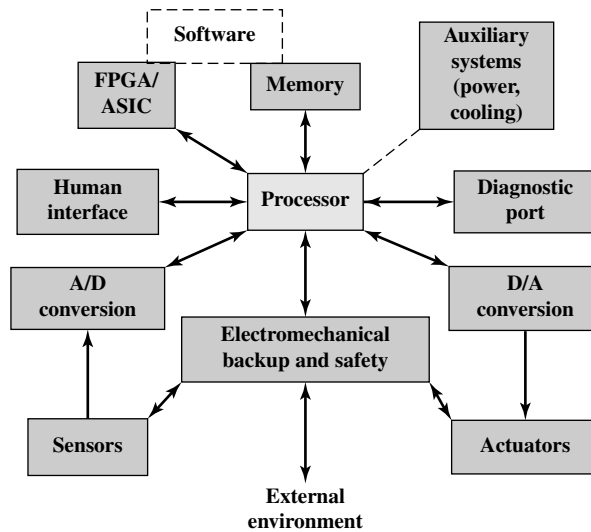


Figure 2.13 Possible Organization of an Embedded System

If multiple activities must be managed simultaneously, this imposes more complex real-time constraints.

Figure 2.13, based on [KOOP96], shows in general terms an embedded system organization. In addition to the processor and memory, there are a number of elements that differ from the typical desktop or laptop computer:

- There may be a variety of interfaces that enable the system to measure, manipulate, and otherwise interact with the external environment.
- The human interface may be as simple as a flashing light or as complicated as real-time robotic vision.
- The diagnostic port may be used for diagnosing the system that is being controlled—not just for diagnosing the computer.
- Special-purpose field programmable (FPGA), application specific (ASIC), or even nondigital hardware may be used to increase performance or safety.
- Software often has a fixed function and is specific to the application.

ARM Evolution

ARM is a family of RISC-based microprocessors and microcontrollers designed by ARM Inc., Cambridge, England. The company doesn't make processors but instead designs microprocessor and multicore architectures and licenses them to manufacturers. ARM chips are high-speed processors that are known for their small die size and low power requirements. They are widely used in PDAs and other handheld devices, including games and phones as well as a large variety of consumer products. ARM chips are the processors in Apple's popular iPod and iPhone devices. ARM is probably the most widely used embedded processor architecture and indeed the most widely used processor architecture of any kind in the world.

The origins of ARM technology can be traced back to the British-based Acorn Computers company. In the early 1980s, Acorn was awarded a contract by the

Table 2.8 ARM Evolution

Family	Notable Features	Cache	Typical MIPS @ MHz
ARM1	32-bit RISC	None	
ARM2	Multiply and swap instructions; Integrated memory management unit, graphics and I/O processor	None	7 MIPS @ 12 MHz
ARM3	First use of processor cache	4 KB unified	12 MIPS @ 25 MHz
ARM6	First to support 32-bit addresses; floating-point unit	4 KB unified	28 MIPS @ 33 MHz
ARM7	Integrated SoC	8 KB unified	60 MIPS @ 60 MHz
ARM8	5-stage pipeline; static branch prediction	8 KB unified	84 MIPS @ 72 MHz
ARM9		16 KB/16 KB	300 MIPS @ 300 MHz
ARM9E	Enhanced DSP instructions	16 KB/16 KB	220 MIPS @ 200 MHz
ARM10E	6-stage pipeline	32 KB/32 KB	
ARM11	9-stage pipeline	Variable	740 MIPS @ 665 MHz
Cortex	13-stage superscalar pipeline	Variable	2000 MIPS @ 1 GHz
XScale	Applications processor; 7-stage pipeline	32 KB/32 KB L1 512 KB L2	1000 MIPS @ 1.25 GHz

DSP = digital signal processor

SoC = system on a chip

British Broadcasting Corporation (BBC) to develop a new microcomputer architecture for the BBC Computer Literacy Project. The success of this contract enabled Acorn to go on to develop the first commercial RISC processor, the Acorn RISC Machine (ARM). The first version, ARM1, became operational in 1985 and was used for internal research and development as well as being used as a coprocessor in the BBC machine. Also in 1985, Acorn released the ARM2, which had greater functionality and speed within the same physical space. Further improvements were achieved with the release in 1989 of the ARM3.

Throughout this period, Acorn used the company VLSI Technology to do the actual fabrication of the processor chips. VLSI was licensed to market the chip on its own and had some success in getting other companies to use the ARM in their products, particularly as an embedded processor.

The ARM design matched a growing commercial need for a high-performance, low-power-consumption, small-size and low-cost processor for embedded applications. But further development was beyond the scope of Acorn's capabilities. Accordingly, a new company was organized, with Acorn, VLSI, and Apple Computer as founding partners, known as ARM Ltd. The Acorn RISC Machine became the Advanced RISC Machine.¹⁰ The new company's first offering, an improvement on the ARM3, was designated ARM6. Subsequently, the company has introduced a number of new families, with increasing functionality and performance. Table 2.8

¹⁰The company dropped the designation *Advanced RISC Machine* in the late 1990s. It is now simply known as the ARM architecture.

shows some characteristics of the various ARM architecture families. The numbers in this table are only approximate guides; actual values vary widely for different implementations.

According to the ARM Web site arm.com, ARM processors are designed to meet the needs of three system categories:

- **Embedded real-time systems:** Systems for storage, automotive body and power-train, industrial, and networking applications
- **Application platforms:** Devices running open operating systems including Linux, Palm OS, Symbian OS, and Windows CE in wireless, consumer entertainment and digital imaging applications
- **Secure applications:** Smart cards, SIM cards, and payment terminals

2.5 PERFORMANCE ASSESSMENT

In evaluating processor hardware and setting requirements for new systems, performance is one of the key parameters to consider, along with cost, size, security, reliability, and, in some cases power consumption.

It is difficult to make meaningful performance comparisons among different processors, even among processors in the same family. Raw speed is far less important than how a processor performs when executing a given application. Unfortunately, application performance depends not just on the raw speed of the processor, but on the instruction set, choice of implementation language, efficiency of the compiler, and skill of the programming done to implement the application.

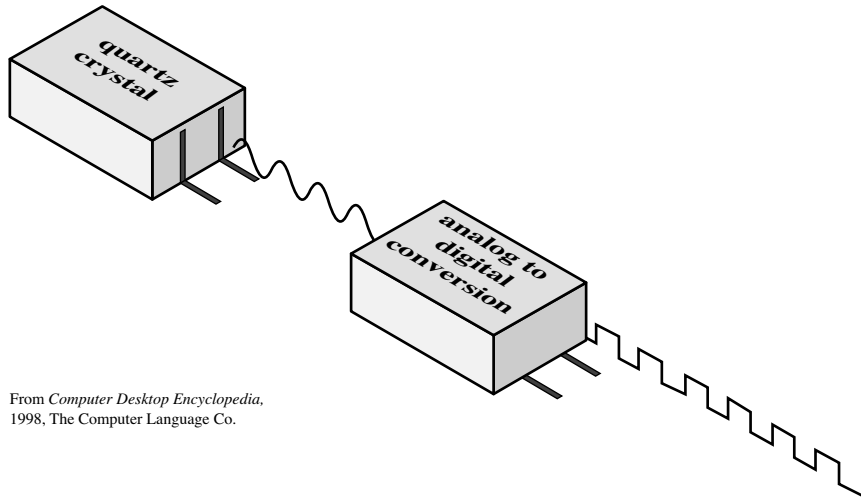
We begin this section with a look at some traditional measures of processor speed. Then we examine the most common approach to assessing processor and computer system performance. We follow this with a discussion of how to average results from multiple tests. Finally, we look at the insights produced by considering Amdahl's law.

Clock Speed and Instructions per Second

THE SYSTEM CLOCK Operations performed by a processor, such as fetching an instruction, decoding the instruction, performing an arithmetic operation, and so on, are governed by a system clock. Typically, all operations begin with the pulse of the clock. Thus, at the most fundamental level, the speed of a processor is dictated by the pulse frequency produced by the clock, measured in cycles per second, or Hertz (Hz).

Typically, clock signals are generated by a quartz crystal, which generates a constant signal wave while power is applied. This wave is converted into a digital voltage pulse stream that is provided in a constant flow to the processor circuitry (Figure 2.14). For example, a 1-GHz processor receives 1 billion pulses per second. The rate of pulses is known as the **clock rate**, or **clock speed**. One increment, or pulse, of the clock is referred to as a **clock cycle**, or a **clock tick**. The time between pulses is the **cycle time**.

The clock rate is not arbitrary, but must be appropriate for the physical layout of the processor. Actions in the processor require signals to be sent from one processor element to another. When a signal is placed on a line inside the processor,



From *Computer Desktop Encyclopedia*,
1998, The Computer Language Co.

Figure 2.14 System Clock

it takes some finite amount of time for the voltage levels to settle down so that an accurate value (1 or 0) is available. Furthermore, depending on the physical layout of the processor circuits, some signals may change more rapidly than others. Thus, operations must be synchronized and paced so that the proper electrical signal (voltage) values are available for each operation.

The execution of an instruction involves a number of discrete steps, such as fetching the instruction from memory, decoding the various portions of the instruction, loading and storing data, and performing arithmetic and logical operations. Thus, most instructions on most processors require multiple clock cycles to complete. Some instructions may take only a few cycles, while others require dozens. In addition, when pipelining is used, multiple instructions are being executed simultaneously. Thus, a straight comparison of clock speeds on different processors does not tell the whole story about performance.

INSTRUCTION EXECUTION RATE A processor is driven by a clock with a constant frequency f or, equivalently, a constant cycle time τ , where $\tau = 1/f$. Define the instruction count, I_c , for a program as the number of machine instructions executed for that program until it runs to completion or for some defined time interval. Note that this is the number of instruction executions, not the number of instructions in the object code of the program. An important parameter is the average cycles per instruction CPI for a program. If all instructions required the same number of clock cycles, then CPI would be a constant value for a processor. However, on any give processor, the number of clock cycles required varies for different types of instructions, such as load, store, branch, and so on. Let CPI_i be the number of cycles required for instruction type i , and I_i be the number of executed instructions of type i for a given program. Then we can calculate an overall CPI as follows:

$$CPI = \frac{\sum_{i=1}^n (CPI_i \times I_i)}{I_c} \quad (2.1)$$

Table 2.9 Performance Factors and System Attributes

	I_c	p	m	k	τ
Instruction set architecture	X	X			
Compiler technology	X	X	X		
Processor implementation		X			X
Cache and memory hierarchy				X	X

The processor time T needed to execute a given program can be expressed as

$$T = I_c \times CPI \times \tau$$

We can refine this formulation by recognizing that during the execution of an instruction, part of the work is done by the processor, and part of the time a word is being transferred to or from memory. In this latter case, the time to transfer depends on the memory cycle time, which may be greater than the processor cycle time. We can rewrite the preceding equation as

$$T = I_c \times [p + (m \times k)] \times \tau$$

where p is the number of processor cycles needed to decode and execute the instruction, m is the number of memory references needed, and k is the ratio between memory cycle time and processor cycle time. The five performance factors in the preceding equation (I_c , p , m , k , τ) are influenced by four system attributes: the design of the instruction set (known as *instruction set architecture*), compiler technology (how effective the compiler is in producing an efficient machine language program from a high-level language program), processor implementation, and cache and memory hierarchy. Table 2.9, based on [HWAN93], is a matrix in which one dimension shows the five performance factors and the other dimension shows the four system attributes. An X in a cell indicates a system attribute that affects a performance factor.

A common measure of performance for a processor is the rate at which instructions are executed, expressed as millions of instructions per second (MIPS), referred to as the **MIPS rate**. We can express the MIPS rate in terms of the clock rate and CPI as follows:

$$\text{MIPS rate} = \frac{I_c}{T \times 10^6} = \frac{f}{CPI \times 10^6} \quad (2.2)$$

For example, consider the execution of a program which results in the execution of 2 million instructions on a 400-MHz processor. The program consists of four major types of instructions. The instruction mix and the CPI for each instruction type are given below based on the result of a program trace experiment:

Instruction Type	CPI	Instruction Mix
Arithmetic and logic	1	60%
Load/store with cache hit	2	18%
Branch	4	12%
Memory reference with cache miss	8	10%

The average CPI when the program is executed on a uniprocessor with the above trace results is $CPI = 0.6 + (2 \times 0.18) + (4 \times 0.12) + (8 \times 0.1) = 2.24$. The corresponding MIPS rate is $(400 \times 10^6)/(2.24 \times 10^6) \approx 178$.

Another common performance measure deals only with floating-point instructions. These are common in many scientific and game applications. Floating-point performance is expressed as millions of floating-point operations per second (MFLOPS), defined as follows:

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

Benchmarks

Measures such as MIPS and MFLOPS have proven inadequate to evaluating the performance of processors. Because of differences in instruction sets, the instruction execution rate is not a valid means of comparing the performance of different architectures. For example, consider this high-level language statement:

```
A = B + C    /* assume all quantities in main memory */
```

With a traditional instruction set architecture, referred to as a complex instruction set computer (CISC), this instruction can be compiled into one processor instruction:

```
add    mem(B) , mem(C) , mem (A)
```

On a typical RISC machine, the compilation would look something like this:

```
load   mem(B) , reg(1) ;
load   mem(C) , reg(2) ;
add     reg(1) , reg(2) , reg(3) ;
store  reg(3) , mem (A)
```

Because of the nature of the RISC architecture (discussed in Chapter 13), both machines may execute the original high-level language instruction in about the same time. If this example is representative of the two machines, then if the CISC machine is rated at 1 MIPS, the RISC machine would be rated at 4 MIPS. But both do the same amount of high-level language work in the same amount of time.

Further, the performance of a given processor on a given program may not be useful in determining how that processor will perform on a very different type of application. Accordingly, beginning in the late 1980s and early 1990s, industry and academic interest shifted to measuring the performance of systems using a set of benchmark programs. The same set of programs can be run on different machines and the execution times compared.

[WEIC90] lists the following as desirable characteristics of a benchmark program:

1. It is written in a high-level language, making it portable across different machines.
2. It is representative of a particular kind of programming style, such as systems programming, numerical programming, or commercial programming.

3. It can be measured easily.
4. It has wide distribution.

SPEC BENCHMARKS The common need in industry and academic and research communities for generally accepted computer performance measurements has led to the development of standardized benchmark suites. A benchmark suite is a collection of programs, defined in a high-level language, that together attempt to provide a representative test of a computer in a particular application or system programming area. The best known such collection of benchmark suites is defined and maintained by the System Performance Evaluation Corporation (SPEC), an industry consortium. SPEC performance measurements are widely used for comparison and research purposes.

The best known of the SPEC benchmark suites is SPEC CPU2006. This is the industry standard suite for processor-intensive applications. That is, SPEC CPU2006 is appropriate for measuring performance for applications that spend most of their time doing computation rather than I/O. The CPU2006 suite is based on existing applications that have already been ported to a wide variety of platforms by SPEC industry members. It consists of 17 floating-point programs written in C, C++, and Fortran; and 12 integer programs written in C and C++. The suite contains over 3 million lines of code. This is the fifth generation of processor-intensive suites from SPEC, replacing SPEC CPU2000, SPEC CPU95, SPEC CPU92, and SPEC CPU89 [HENN07].

Other SPEC suites include the following:

- **SPECjvm98:** Intended to evaluate performance of the combined hardware and software aspects of the Java Virtual Machine (JVM) client platform
- **SPECjbb2000 (Java Business Benchmark):** A benchmark for evaluating server-side Java-based electronic commerce applications
- **SPECweb99:** Evaluates the performance of World Wide Web (WWW) servers
- **SPECmail2001:** Designed to measure a system's performance acting as a mail server

AVERAGING RESULTS To obtain a reliable comparison of the performance of various computers, it is preferable to run a number of different benchmark programs on each machine and then average the results. For example, if m different benchmark program, then a simple **arithmetic mean** can be calculated as follows:

$$R_A = \frac{1}{m} \sum_{i=1}^m R_i \quad (2.3)$$

where R_i is the high-level language instruction execution rate for the i th benchmark program.

An alternative is to take the **harmonic mean**:

$$R_H = \frac{m}{\sum_{i=1}^m \frac{1}{R_i}} \quad (2.4)$$

Ultimately, the user is concerned with the execution time of a system, not its execution rate. If we take arithmetic mean of the instruction rates of various benchmark programs, we get a result that is proportional to the sum of the inverses of

execution times. But this is not inversely proportional to the sum of execution times. In other words, the arithmetic mean of the instruction rate does not cleanly relate to execution time. On the other hand, the harmonic mean instruction rate is the inverse of the average execution time.

SPEC benchmarks do not concern themselves with instruction execution rates. Rather, two fundamental metrics are of interest: a speed metric and a rate metric. The **speed metric** measures the ability of a computer to complete a single task. SPEC defines a base runtime for each benchmark program using a reference machine. Results for a system under test are reported as the **ratio** of the reference run time to the system run time. The ratio is calculated as follows:

$$r_i = \frac{T_{ref_i}}{T_{sut_i}} \quad (2.5)$$

where T_{ref_i} is the execution time of benchmark program i on the reference system and T_{sut_i} is the execution time of benchmark program i on the system under test.

As an example of the calculation and reporting, consider the Sun Blade 6250, which consists of two chips with four cores, or processors, per chip. One of the SPEC CPU2006 integer benchmark is 464.h264ref. This is a reference implementation of H.264/AVC (Advanced Video Coding), the latest state-of-the-art video compression standard. The Sun system executes this program in 934 seconds. The reference implementation requires 22,135 seconds. The ratio is calculated as: $22136/934 = 23.7$.

Because the time for the system under test is in the denominator, the larger the ratio, the higher the speed. An overall performance measure for the system under test is calculated by averaging the values for the ratios for all 12 integer benchmarks. SPEC specifies the use of a **geometric mean**, defined as follows:

$$r_G = \left(\prod_{i=1}^n r_i \right)^{1/n} \quad (2.6)$$

where r_i is the ratio for the i th benchmark program. For the Sun Blade 6250, the SPEC integer speed ratios were reported as follows:

Benchmark	Ratio
400.perlbench	17.5
401.bzip2	14.0
403.gcc	13.7
429.mcf	17.6
445.gobmk	14.7
456.hmmer	18.6

Benchmark	Ratio
458.sjeng	17.0
462.libquantum	31.3
464.h264ref	23.7
471.omnetpp	9.23
473.astar	10.9
483.xalancbmk	14.7

The speed metric is calculated by taking the twelfth root of the product of the ratios:

$$(17.5 \times 14 \times 13.7 \times 17.6 \times 14.7 \times 18.6 \times 17 \times 31.3 \times 23.7 \times 9.23 \times 10.9 \times 14.7)^{1/12} = 18.5$$

The **rate metric** measures the throughput or rate of a machine carrying out a number of tasks. For the rate metrics, multiple copies of the benchmarks are run simultaneously. Typically, the number of copies is the same as the number of processors on the machine. Again, a ratio is used to report results, although the calculation

is more complex. The ratio is calculated as follows:

$$r_i = \frac{N \times Tref_i}{T_{sut_i}} \quad (2.7)$$

where $Tref_i$ is the reference execution time for benchmark i , N is the number of copies of the program that are run simultaneously, and T_{sut_i} is the elapsed time from the start of the execution of the program on all N processors of the system under test until the completion of all the copies of the program. Again, a geometric mean is calculated to determine the overall performance measure.

SPEC chose to use a geometric mean because it is the most appropriate for normalized numbers, such as ratios. [FLEM86] demonstrates that the geometric mean has the property of performance relationships consistently maintained regardless of the computer that is used as the basis for normalization.

Amdahl's Law

When considering system performance, computer system designers look for ways to improve performance by improvement in technology or change in design. Examples include the use of parallel processors, the use of a memory cache hierarchy, and speedup in memory access time and I/O transfer rate due to technology improvements. In all of these cases, it is important to note that a speedup in one aspect of the technology or design does not result in a corresponding improvement in performance. This limitation is succinctly expressed by Amdahl's law.

Amdahl's law was first proposed by Gene Amdahl in [AMDA67] and deals with the potential speedup of a program using multiple processors compared to a single processor. Consider a program running on a single processor such that a fraction $(1 - f)$ of the execution time involves code that is inherently serial and a fraction f that involves code that is infinitely parallelizable with no scheduling overhead. Let T be the total execution time of the program using a single processor. Then the speedup using a parallel processor with N processors that fully exploits the parallel portion of the program is as follows:

$$\begin{aligned} \text{Speedup} &= \frac{\text{time to execute program on a single processor}}{\text{time to execute program on } N \text{ parallel processors}} \\ &= \frac{T(1 - f) + Tf}{T(1 - f) + \frac{Tf}{N}} = \frac{1}{(1 - f) + \frac{f}{N}} \end{aligned}$$

Two important conclusions can be drawn:

1. When f is small, the use of parallel processors has little effect.
2. As N approaches infinity, speedup is bound by $1/(1 - f)$, so that there are diminishing returns for using more processors.

These conclusions are too pessimistic, an assertion first put forward in [GUST88]. For example, a server can maintain multiple threads or multiple tasks to handle multiple clients and execute the threads or tasks in parallel up to the limit of the number of processors. Many database applications involve computations on massive amounts of data that can be split up into multiple parallel tasks. Nevertheless,

Amdahl's law illustrates the problems facing industry in the development of multi-core machines with an ever-growing number of cores: The software that runs on such machines must be adapted to a highly parallel execution environment to exploit the power of parallel processing.

Amdahl's law can be generalized to evaluate any design or technical improvement in a computer system. Consider any enhancement to a feature of a system that results in a speedup. The speedup can be expressed as

$$\text{Speedup} = \frac{\text{Performance after enhancement}}{\text{Performance before enhancement}} = \frac{\text{Execution time before enhancement}}{\text{Execution time after enhancement}} \quad (2.8)$$

Suppose that a feature of the system is used during execution a fraction of the time f , before enhancement, and that the speedup of that feature after enhancement is SU_f . Then the overall speedup of the system is

$$\text{Speedup} = \frac{1}{(1 - f) + \frac{f}{SU_f}}$$

For example, suppose that a task makes extensive use of floating-point operations, with 40% of the time is consumed by floating-point operations. With a new hardware design, the floating-point module is speeded up by a factor of K . Then the overall speedup is:

$$\text{Speedup} = \frac{1}{0.6 + \frac{0.4}{K}}$$

Thus, independent of K , the maximum speedup is 1.67.

2.6 RECOMMENDED READING AND WEB SITES

A description of the IBM 7000 series can be found in [BELL71]. There is good coverage of the IBM 360 in [SIEW82] and of the PDP-8 and other DEC machines in [BELL78a]. These three books also contain numerous detailed examples of other computers spanning the history of computers through the early 1980s. A more recent book that includes an excellent set of case studies of historical machines is [BLAA97]. A good history of the microprocessor is [BETK97].

[OLUK96], [HAMM97], and [SAKA02] discuss the motivation for multiple processors on a single chip.

[BREY09] provides a good survey of the Intel microprocessor line. The Intel documentation itself is also good [INTE08].

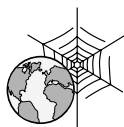
The most thorough documentation available for the ARM architecture is [SEAL00].¹¹ [FURB00] is another excellent source of information. [SMIT08] is an interesting comparison of the ARM and x86 approaches to embedding processors in mobile wireless devices.

For interesting discussions of Moore's law and its consequences, see [HUTC96], [SCHA97], and [BOHR98].

[HENN06] provides a detailed description of each of the benchmarks in CPU2006. [SMIT88] discusses the relative merits of arithmetic, harmonic, and geometric means.

¹¹Known in the ARM community as the "ARM ARM."

- BELL71** Bell, C., and Newell, A. *Computer Structures: Readings and Examples*. New York: McGraw-Hill, 1971.
- BELL78A** Bell, C.; Mudge, J.; and McNamara, J. *Computer Engineering: A DEC View of Hardware Systems Design*. Bedford, MA: Digital Press, 1978.
- BETK97** Betker, M.; Fernando, J.; and Whalen, S. "The History of the Microprocessor." *Bell Labs Technical Journal*, Autumn 1997.
- BLAA97** Blaauw, G., and Brooks, F. *Computer Architecture: Concepts and Evolution*. Reading, MA: Addison-Wesley, 1997.
- BOHR98** Bohr, M. "Silicon Trends and Limits for Advanced Microprocessors." *Communications of the ACM*, March 1998.
- BREY09** Brey, B. *The Intel Microprocessors: 8086/8066, 80186/80188, 80286, 80386, 80486, Pentium, Pentium Pro Processor, Pentium II, Pentium III, Pentium 4 and Core2 with 64-bit Extensions*. Upper Saddle River, NJ: Prentice Hall, 2009.
- FURB00** Furber, S. *ARM System-On-Chip Architecture*. Reading, MA: Addison-Wesley, 2000.
- HAMM97** Hammond, L.; Nayfay, B.; and Olukotun, K. "A Single-Chip Multiprocessor." *Computer*, September 1997.
- HENN06** Henning, J. "SPEC CPU2006 Benchmark Descriptions." *Computer Architecture News*, September 2006.
- HUTC96** Hutcheson, G., and Hutcheson, J. "Technology and Economics in the Semiconductor Industry." *Scientific American*, January 1996.
- INTE08** Intel Corp. Intel® 64 and IA-32 *Intel Architectures Software Developer's Manual (3 volumes)*. Denver, CO, 2008. intel.com/products/processor/manuals
- OLUK96** Olukotun, K., et al. "The Case for a Single-Chip Multiprocessor." *Proceedings, Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, 1996.
- SAKA02** Sakai, S. "CMP on SoC: Architect's View." *Proceedings. 15th International Symposium on System Synthesis*, 2002.
- SCHA97** Schaller, R. "Moore's Law: Past, Present, and Future." *IEEE Spectrum*, June 1997.
- SEAL00** Seal, D., ed. *ARM Architecture Reference Manual*. Reading, MA: Addison-Wesley, 2000.
- SIEW82** Siewiorek, D.; Bell, C.; and Newell, A. *Computer Structures: Principles and Examples*. New York: McGraw-Hill, 1982.
- SMIT88** Smith, J. "Characterizing Computer Performance with a Single Number." *Communications of the ACM*, October 1988.
- SMIT08** Smith, B. "ARM and Intel Battle over the Mobile Chip's Future." *Computer*, May 2008.



Recommended Web sites:

- **Intel Developer's Page:** Intel's Web page for developers; provides a starting point for accessing Pentium information. Also includes the Intel Technology Journal.
- **ARM:** Home page of ARM Limited, developer of the ARM architecture. Includes technical documentation.

- **Standard Performance Evaluation Corporation:** SPEC is a widely recognized organization in the computer industry for its development of standardized benchmarks used to measure and compare performance of different computer systems.
- **Top500 Supercomputer Site:** Provides brief description of architecture and organization of current supercomputer products, plus comparisons.
- **Charles Babbage Institute:** Provides links to a number of Web sites dealing with the history of computers.

2.7 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

accumulator (AC) Amdahl's law arithmetic and logic unit (ALU) benchmark chip data channel embedded system execute cycle fetch cycle input-output (I/O) instruction buffer register (IBR)	instruction cycle instruction register (IR) instruction set integrated circuit (IC) main memory memory address register (MAR) memory buffer register (MBR) microprocessor multicore multiplexor	opcode original equipment manufacturer (OEM) program control unit program counter (PC) SPEC stored program computer upward compatible von Neumann machine wafer word
--	--	---

Review Questions

- 2.1. What is a stored program computer?
- 2.2. What are the four main components of any general-purpose computer?
- 2.3. At the integrated circuit level, what are the three principal constituents of a computer system?
- 2.4. Explain Moore's law.
- 2.5. List and explain the key characteristics of a computer family.
- 2.6. What is the key distinguishing feature of a microprocessor?

Problems

- 2.1. Let $\mathbf{A} = A(1), A(2), \dots, A(1000)$ and $\mathbf{B} = B(1), B(2), \dots, B(1000)$ be two vectors (one-dimensional arrays) comprising 1000 numbers each that are to be added to form an array \mathbf{C} such that $C(I) = A(I) + B(I)$ for $I = 1, 2, \dots, 1000$. Using the IAS instruction set, write a program for this problem. Ignore the fact that the IAS was designed to have only 1000 words of storage.
- 2.2.
 - a. On the IAS, what would the machine code instruction look like to load the contents of memory address 2?
 - b. How many trips to memory does the CPU need to make to complete this instruction during the instruction cycle?
- 2.3. On the IAS, describe in English the process that the CPU must undertake to read a value from memory and to write a value to memory in terms of what is put into the MAR, MBR, address bus, data bus, and control bus.