5. Systems Thinking

Enable people to follow the way, without them having to understand [the internals of] it.

— Confucius (551–479 BCE)

Inside every large program, there is ~~a small program~~ **an algorithm** trying to get out.
where algorithm also means specification or high-level design of a system

— Leslie Lamport, 2018

Computational processes execute on computing systems, including computer systems and computing application systems. Systems thinking is the way of thinking to make computational processes **practical**. Systems thinking must systematically and thoroughly address all necessary details and complexities. That is,

Being practical = Being thorough + Being systematic + Coping with complexity.

Without systems thinking, we would not have had the vibrant computing ecosystem today, with billions of users using millions of applications on various devices.

The main character of systems thinking is: using **abstractions** to compose **modules** into a system**,** to enable **seamless execution** of computational processes. This chapter discusses systems thinking with its three key concepts: abstraction, modularization, and seamless transition.

Abstraction is a creative process of abstracting the essentials, as well as the process's outcome. Computer science abstractions mainly consist of **data abstractions** and **control abstractions**. All abstractions have three properties: constrained, objective, and generalizable, i.e., the **COG** properties.

Modules are a special type of abstractions which enforce the **information hiding** principle. A system is comprised of modules by interconnecting their interfaces. We discuss a number of hardware and software abstractions. The discussions are ordered by levels of abstractions, from the lowest-level logic gates to the highest-level application software, including combinational circuits, sequential circuits, instruction pipeline, von Neumann architecture, and software stack.

Seamless execution is concerned with how to ensure smooth executions of computational processes. What is the first instruction to execute? Where to find the next instruction? How to transition from one instruction to the next? Is there any bottleneck? What if there is abnormality? We discuss four "laws" which make seamless execution possible: Yang's cycle principle, Postel's robustness principle, von Neumann's exhaustiveness principle, and Amdahl's law.

5.1. Systems Thinking Has Three Objectives

Let us consider computing without systems thinking, even having the benefits of logic thinking and algorithmic thinking. We can still theoretically solve all computable problems, by executing algorithms on Turing machines. But we quickly run into practical problems, as illustrated by the following example.

Example 38. Understand message storage in WeChat

The WeChat (微信) application system is a popular computer application service. In 2018, the WeChat community exceeded 1 billion users worldwide. Can we understand or design a WeChat system by executing algorithms on Turing machines? It is not practical to do so.

Let us consider a specific problem in the WeChat system, the **message storage problem**: when I send a chat message to my family, where should the WeChat service store my message? This storage problem is not a logic problem or an algorithmic problem. We cannot easily formulate and solve the storage problem and evaluate alternative solutions, as we do for the sorting problem. It is a systems problem involving thoughtful considerations and tradeoffs involving many issues of practical systems. A way of systems thinking is needed, which has three main objectives.

- Being **thorough**. The WeChat system design considers all necessary issues such as functionality, user experienced convenience, performance, fault tolerance, privacy, as well as system scalability.
- Being **systematic**. WeChat adopts a systematic approach called cloud computing. Consequently, messages are stored on the WeChat cloud datacenter.
- **Coping with complexity**. There are too many possibilities to consider. A message can be stored on many places: my smartphone, my family members' devices, the WeChat system, or a third-party platform. Which one is the correct or the better choice? How to go about answer such a question?

⚏

Systems thinking strives to simultaneously achieve all three objectives. This makes system thinking a synergy of science, engineering, and art. It is the reason why a system designer is often known as an *architect*.

5.1.1. Being Thorough

In understanding or designing a system such as WeChat, we need a *thorough* way to cover all the details of the integrated whole system from end to end, ignoring unnecessary details.

For instance, when considering where to store a message, we need to consider the entire path the message may traverse, from the message sender end to the message receiver end. We also need to consider the whole stack of systems components from high-level user interfaces, algorithmic descriptions, software and hardware, down to the lowest level of automatic execution on a computer.

We use three examples to illustrate how systems thinking emphasizes thoroughness. One example is representative of necessary but boring details. The other two illustrate how to take care of necessary details by using clever abstractions. Systems thinking requires that all necessary details should be considered, even those boring ones. At the same time, systems thinking strives to avoid simple-minded enumeration of all details, by using abstractions.

Example 39. Big endian versus little endian data representations

A necessary but boring detail is the ordering of bytes of a multi-byte number. We do not see such details in the design and analysis of an algorithm.

For instance, a 32-bit integer 1078018627_{10} can also be written as 0x40414243, which consists of four bytes, where Byte0 is 01000000=0x40, Byte1 is 01000001=0x41, Byte2 is 01000010=0x42, and Byte3 is 01000011=0x43. Most computers have a byte addressable memory. When storing this number in memory, we need four consecutive memory byte cells starting at address A.

The problem is: when storing this number in memory, in what order to place the four bytes 0x40, 0x41, 0x42, 0x43 in the four memory cells A, A+1, A+2, A+3? Two representations (orderings) are used in practice, as shown in Fig. 5.1.

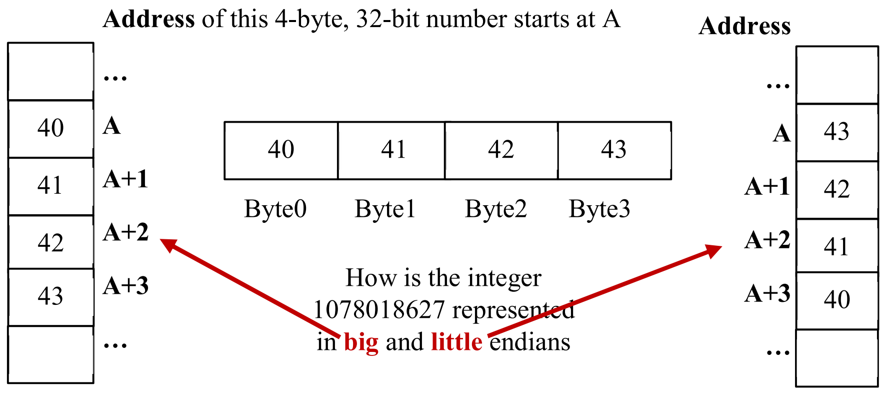


Fig. 5.1 Big endian versus little endian representations of integer 1078018627

The **little endian** ordering places the least significant byte, i.e., 0x43 of 0x40414243, in the smallest address A, and the most significant byte, i.e., 0x40 of 0x40414243, in the biggest address A+3. The **big endian** ordering simply reverses the order. Similarly, in computer communication, where a bit-stream is transmitted between two computers, the little endian ordering sends the least significant bit first, and the big endian ordering sends the most significant bit first.

The big endian versus little endian ordering issue got its name from *Gulliver's Travels*, a satirical novel by Jonathan Swift. In a fictional country Lilliput, two factions fought a holy war over from which end to break a boiled egg. In 1980, Danny

Cohen, at Information Sciences Institute, University of Southern California, applied the terms big endian and little endian to computer science. Cohen holds the viewpoint that "Agreement upon an order is more important than the order agreed upon." Forty years later, the computer science fields settled into a situation where different products and communities use different endians within themselves, and cross-community applications convert the data format whenever necessary. Some example communities follow:

- Big endian: TCP/IP networks, MIPS processors
- Little endian: x86 processors, ARM processors, RISC-V processors

☶

Example 40. A single abstraction for millions of I/O devices

Recall that the von Neumann model of computer has three families of components: the processors, the memory units, and the I/O devices. Of these three, the family of I/O devices is the largest family, with over a million different types of devices having been built and used.

The problem is: how do millions of applications deal with so many I/O devices?

For instance, a scratch pad is quite different from a keyboard, although both are input devices. A display monitor is quite different from a printer, although both are output devices.

We must be thorough, allowing a computer to interact with any I/O devices. A brute-force approach is to design a method for an application to interact with every device. This is not feasible, since there are million × million combinations.

Computer scientists have come up with an ingenious abstraction, called *device driver*, as illustrated in Fig. 5.2.

Applications only need to interact with the generic device driver interface, which is quite similar to the interface of accessing files. Applications only see two generic types of I/O devices: *block devices* and *character devices*. A computer application interacts with a character device one character at a time, such as printing out a program output on the display monitor in command-line mode, one character at a time. Interactions with a block device allow us to input or output a larger chunk of information (called a *block*) at a time, such as outputting an image on the display monitor in graphic mode.

When an I/O device product, say a new scratch pad called Device1, is developed, the device vendor also develops a unit of software, say Driver1, which is the device driver for Device1. This device driver software is developed, say, with 1 man-month cost. It implements the generic interface and realizes all particulars of detailed I/O operations, hiding all such details from applications.

Note that such a device driver abstraction drastically decreases the development cost, from $M \times N$ to $M+N$ where M is number of different applications and N is the number of different devices. Since N and M are numbered in millions, the total cost is reduced from trillions to millions of man-months. For each device, the development cost is reduced from millions of man-months to a few.

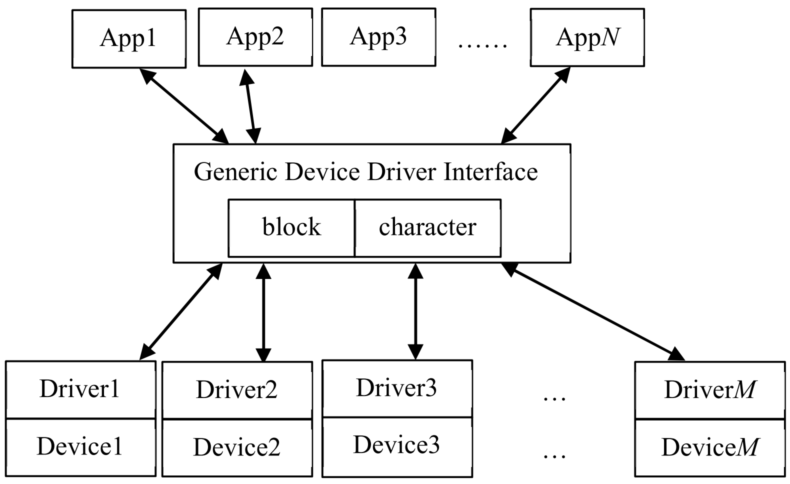


Fig. 5.2 Illustration of the device driver abstraction.

The above example shows that in achieving thoroughness, systems thinking avoids painful enumeration of all details, by using clever abstractions. We consider another case to further illustrate this point.

Example 41. Measure supercomputers by smartly designed benchmarks

Supercomputers are the fastest computers in the world. We want supercomputers to increase their speed a thousand-fold every ten years. Here lies a basic problem: How do we precisely define and measure this objective? This problem can be further divided into two more detailed questions:

- How to measure the speed of a supercomputer?
- How to be thorough, i.e., consider all applications when measuring the speed?

An answer to the first question is straightforward: run a small set of representative application programs, called **benchmarks**, and count the number of operations executed per second. Supercomputers measure speed with **FLOPS**, for 64-bit floating-point addition and multiplication operations executed per second.

The second question is more difficult. What are *representative* benchmark programs? A benchmark suite usually consists of no more than a dozen programs. How can it represent the thousands of supercomputing applications today? Systems thinking suggests a method shown in Fig. 5.3. Supercomputing speed is significantly influenced by a phenomenon called locality. **Temporal locality** refers to the fact that data and instructions currently used tend to be used again in near future. **Spatial locality** refers to a similar fact regarding address space.

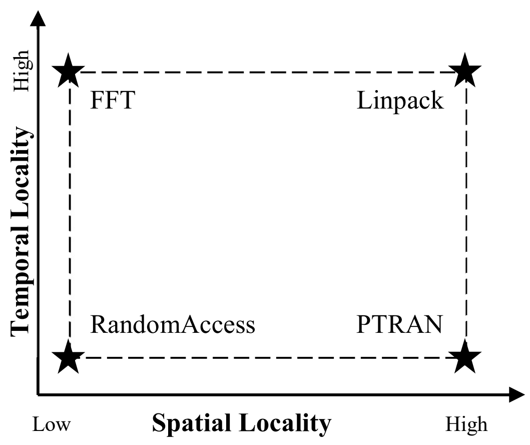


Fig. 5.3 Illustration of using benchmarks of various representative localities.

The supercomputing community designed a benchmark suite containing four benchmark programs, representing the four extreme combinations of temporal and spatial localities: low-low, low-high, high-low, and high-high. Any other application falls within the area enclosed by the dashed lines.

☷

5.1.2. Being Systematic

We have literally thousands of types of computers and millions of computer applications today. It is unthinkable to design or understand each of them in an arbitrary, ad hoc way. Fortunately, computer science has created a systematic, layered approach to support millions of applications for billions of users on their favorite computers. This approach is called the technology **stack** approach. Figure 5.4 shows a hardware-software stack for a desktop PC. Similar stacks are available for embedded computers, smartphones, servers, and supercomputers.

In Fig. 5.4, the colored parts represent hardware components of a computer. These components are formed from logic gates, combinational circuits, and sequential circuits, which in turn are constructed from transistors and wires, as well as capacitors and resistors. Magnetic parts are the main components of hard disks. There are millions of products of I/O devices.

The upper three layers of the stack are software. Instructions are the smallest units of software. System software (such as operating system and compiler) and application software are made of instructions. When any application program starts to execute, a **process** of the application is created. Thus, a process is a program in execution. The operating system manages the processes by scheduling them to execute on the computer hardware at certain times.

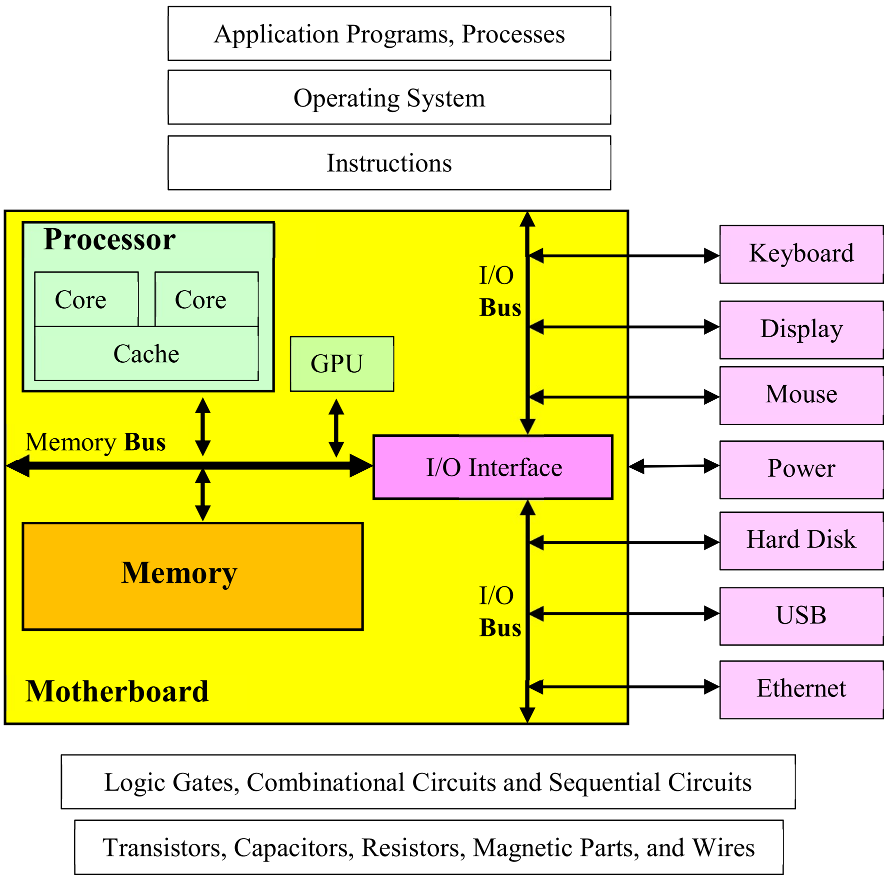


Fig. 5.4 Illustration of the hardware-software stack of a desktop personal computer

The stack of layers in Fig. 5.4 conveys two meanings of being systematic.

- First, the same stack is used to support millions of applications, instead of a million stacks, each for one application.
- Second, an upper layer provides a higher level of abstraction than the lower layers, and the upper layer abstraction utilizes primitive resources or abstractions from the lower layers. For instance, a processor is made of combinational circuits and sequential circuits, which in turn are made of transistors and wires. A processor has a much higher level of abstraction than transistors and wires.

5.1.3. Coping with Complexity

Complexity here refers to systems complexity, which is different from algorithmic complexity in Chapter 4. A system is either a computer system, such as a laptop computer, or a computer application system, such as WeChat. Complexity makes it difficult to understand, design, and use a computing system. Systems thinking is used to cope with complexity. We have already seen millions of practically working computing systems, including WeChat. These successful cases testify that systems thinking provides effective supports for coping with complexity.

Many factors contribute to systems complexity. We discuss four such factors which appear frequently.

The first factor is **system scale**, or system size, which refers to the number of components of a system. For instance, the main processor microchip in a smartphone has over 2 billion transistors. If we liken the microchip to Planet Earth, and compare transistors and wires to buildings and roads, the transistor layout diagram of a microchip is as complex as a country's meter-scale map, showing all buildings and roads. The hardware system used by WeChat is also as complex as a meter-scale world map. We cannot understand such a complex system as a set of 2 quintillion (billion billion) transistors.

The second factor is **system heterogeneity**, or diversity, which refers to the number of different types of components in a system. For instance, WeChat has over 1 billion users worldwide, where each user uses one or more computing devices to access the WeChat services. These devices have much heterogeneity and diversity. A device could be a smartphone, a pad, a PC, or even a robot on a server. The smartphones are made by thousands of different companies. It is a wonder that WeChat works at all with such device heterogeneity and diversity.

The third factor is **system organization**, i.e., how the components are connected and organized into a system. A haphazard mess of interconnected components is much more complex than a system with clear, principled organization.

The fourth factor is **system variation**, which refers to the fact that the system or its components are often not stable, but keep changing. Furthermore, they may change at different times and different rates. Sometimes, we also call system variation as system dynamicity. Examples of such changes include: deploying new products or services, upgrading an existing service or product, bug-fixing, etc.

In summary, to make computing practical, we need systems thinking which is thorough, systematic, and can cope with systems complexity. Computer science has accumulated a rich body of knowledge for systems thinking. The essence is using **abstractions** to compose **modules** into a system, to enable **seamless execution** of computational processes. In other words, systems thinking has three mental tools: abstraction, modularization, and seamless transition. The beauty of systems thinking is that even with seemingly impossible thoroughness and systematic requirements, these mental tools enable us to cope with such complexity presented in today's computer application systems consisting of billions of diverse users and devices.

5.2. Abstraction

Abstraction is the creative process of abstracting a high-level concept from low-level instances, which are full of irrelevant details and particularities. An abstraction is also the outcome of the creative process of abstracting. This book focuses on computing abstractions, namely, the abstracting process that produces abstractions of information transformation by digital symbol manipulations. From the systems viewpoint, this book studies four classes of abstractions, as listed in Table 5.1. They are (1) data representations, (2) software abstractions, (3) hardware abstractions, and (4) the von Neumann architecture model that bridges software and hardware.

Table 5.1 Four Classes of Abstractions

Data Type	bit (1 bit), hexadecimal number (4 bits), byte (8 bits), uint8 (8-bit unsigned integer), integer (64 bits); array (n elements of the same type), slice (a descriptor pointing to an array); text file, BMP image file; hypertext and hyperlink	
Software	Algorithm	Smart method of information transformation, such as quicksort, hiding text in a BMP file, etc.
	Program	Code realizing algorithms in computer language, such as hide.go in the Text Hider project
	Process	Program in execution, such as the “hide” process running in a Linux environment
	Instruction	The smallest unit of software, directly executable by computer hardware
von Neumann Architecture: a computer model bridging software and hardware		
Hardware	Instruction Pipeline	The basic hardware mechanism to automatically execute any instruction
	Sequential Circuit	More precisely, only consider Synchronous Sequential Circuit comprised of combinational circuits and state circuits and driven by a clock signal; equivalent to the automata concept
	Combinational Circuit	Aka Boolean circuit, realizing a Boolean function

5.2.1. Three Properties of Abstraction: COG

All abstractions have three properties, called the **COG properties** of abstraction. That is, any computing abstraction is constrained, objective, and generalizable.

- Constrained. An abstraction is a high-level concept specification constrained by hiding details. Abstraction focuses on the essential aspect when specifying the computing system (or a subsystem) from one perspective, while hiding or ignoring details from other perspectives and particularities of individual instances. The ability to hide and ignore, namely, to constrain, is why abstraction can cope with complexity.

- **Objective**. Abstraction does not imply up-in-the-air vagueness or ambiguity. A computing abstraction is a named, objective entity. It is a precisely defined concept, both syntactically and semantically, and cannot be arbitrarily interpreted or changed by any particular human's whim. Objectivity makes computing abstractions bit-accurate and automatically executable.
- **Generalizable**. An abstraction should be generalizable to unseen instances or unexpected scenarios. Computing abstraction is created by humans. The created abstraction should be able to handle existing abstractions or instances already seen, as well as unseen instances and unexpected scenarios. This capability of generalization is why we can use one set of abstractions of concepts and methods to solve all problems encountered, instead of treating each individual problem instance individually.

Example 42. The COG properties of Unicode

The 128-character ASCII set is sufficient to encode English text. Can we do the same for all the world's text? The computer science community has come up with a beautiful abstraction called **Unicode** to solve the problem. By the year 2020, Unicode already encodes over 143000 characters in 154 languages, including over 70,000 Chinese characters.

"Encoding the world's writing systems by a number of bits" is the process of abstracting. The Unicode abstraction is the outcome of this abstracting process. It has three properties of COG.

- Unicode is **constrained**. It focuses on one essential task: encoding the world's writing systems, or character sets. It ignores issues such as the font, the size, the alignment of the character, whether it is boldface or italic, etc.
- Unicode is **objective**. Unicode is precisely defined. The Chinese character '志' and the Euro sign '€' have encodings U+5FD7 and U+20AC, respectively, without ambiguity.
- Unicode is **generalizable**. Unicode uses a standard abstraction to solve the problem of "encoding text in the world's writing systems". It is not tied to any computer hardware, software, or application scenario. It is used on our PCs, smartphones, and the World Wide Web. Most computer devices and applications supported by Unicode did not exist when Unicode was designed.

☷

5.2.2. Data Abstractions

Computing abstractions mainly manifest as data abstractions and control abstractions. **Data abstractions**, also called **data types**, are abstractions of data, including operations on such data. Examples of data abstractions include number systems, bit, byte, character, string, integer, floating-point number, array, slice, text file, BMP image file, hypertext and hyperlink. **Control abstractions** are abstractions specifying the control structure of a system, that is, when and how to invoke which parts of a system, to give order to the totality of the system. Examples of control abstractions include operator precedence, sequence, selection, iteration, and function. Some abstractions exhibit features of both data abstractions and control abstractions. We discuss several examples in this section. Section 5.3 contains more examples of abstractions as hardware and software modules.

25. Positional Notation of Number Systems

Most modern number systems are **positional number systems**. That is, they use a positional notation to represent a number such that the value of the number is determined by its digits and the position of each digit. The following example shows why the positional notation is popular: it makes doing arithmetic easy, as we do with the usual decimal notation.

Example 43. Find a person's age using Roman and decimal numerals

A person was born in the year MCMLIV. What's his age in the year MMXXI?
The above question uses Roman numerals. We need to find out

MMXXI – MCMLIV = ?

Note the following mapping between Roman numbers and decimal numbers.

Roman	M	D	C	L	X	V	I	IV	IX	XL	XC	CD	CM
Decimal	1000	500	100	50	10	5	1	4	9	40	90	400	900

Using Roman numerals to do arithmetic is difficult. Converting the numbers into the decimal notation makes it easier: 2021 – 1954 = 67. The person is 67 years old, or LXVII years old in Roman numerals. That is, MMXXI – MCMLIV = LXVII.

Why is the decimal notation so much easier? Because it is a **positional notation**, as illustrated in Fig. 5.5. The decimal number $a = a_3a_2a_1a_0.a_{-1}a_{-2}a_{-3}a_{-4} = 2021.1954$ has eight decimal **digits**. Two digits have identical symbol 2. But, because of their different positions, they represent different values. The leftmost 2 represents two thousands, and the second 2 represents two tens. Note that the value of a is $a = \sum_{i=-4}^{3} a_i \times 10^i$, where i is the **index**.

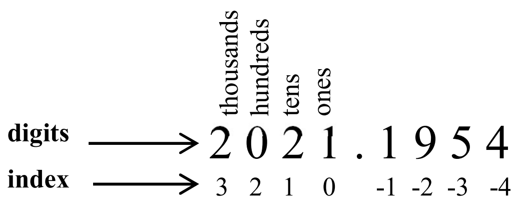


Fig. 5.5 Explaining the positional notation of decimal number 2021.1954

Example 44. Other positional number systems

In general, the value of a natural number *a* represented in a base-*b*, *n*-digit positional notation is evaluated by

$$a = \sum_{i=0}^{n-1} a_i \times b^i$$

where digit a_i takes a value from the **digit set** {0, 1, …, *b*-1}.

For binary notation, we have

$$a = \sum_{i=0}^{n-1} a_i \times 2^i$$

where digit a_i takes a value in the digit set {0, 1}.

With a hexadecimal positional notation, we have

$$a = \sum_{i=0}^{n-1} a_i \times 16^i$$

where digit a_i takes a value in the digit set {0, 1, …, 15}={0, 1, …, F}.

It is apparent that a particular positional number system is determined by three things: its word-length *n*, its base *b*, and its digit set.

To more clearly understand the concept of positional number system, let us ask some questions: What values are allowed for the base and the digit set? What is the relation between the base and the digit set? It appears that the base should be a positive integer *k*, and the digit set is {0, 1, …, *k*-1}. We have seen the cases for *k* = 2 (binary), 10 (decimal), and 16 (hexadecimal).

Can we have radically different positional number systems? Can we use Fibonacci numbers? Can we use an irrational base?

The answers are YES. In 1957, George Bergman, then a 12-year junior high school student, proposed to use {0, 1} as the digit set and the Golden ratio $\tau = (1 + \sqrt{5})/2 \approx 1.6180339$ as the base. He also showed how to do arithmetic in this *τ number system*. Later work on Fibonacci Number System (FNS) showed how to represent numbers using 0 and 1 as digits and Fibonacci numbers as positional weights.

Table 5.2 contrasts these number systems with the more familiar decimal, hexadecimal, and binary number systems. For instance, the decimal number 14 has hexadecimal and binary representations of E and 1110, respectively. With the τ number system, 14 is represented as the following:

$$14 = \mathbf{100100.110110} = \tau^5 + \tau^2 + \tau^{-1} + \tau^{-2} + \tau^{-4} + \tau^{-5}.$$

With the Fibonacci Number System (FNS), 14 is represented as follows:

$$14 = \mathbf{11001} = 1 \times 8 + 1 \times 5 + 0 \times 3 + 0 \times 2 + 1 \times 1.$$

Table 5.2 Decimal, hexadecimal, binary, tau, and Fibonacci number system representations

Decimal	Hexadecimal	Binary	The τ Number System	FNS
$10^1 10^0$	16^0	$2^3 2^2 2^1 2^0$	$\tau^5\tau^4\tau^3\tau^2\tau^1\tau^0\tau^{-1}\tau^{-2}\tau^{-3}\tau^{-4}\tau^{-5}\tau^{-6}$	8 5 3 2 1
0	0	0000	0	00000
1	1	0001	1	00001
2	2	0010	10.01	00010
3	3	0011	100.01	00100
4	4	0100	101.01	00101
5	5	0101	1000.1001	01000
6	6	0110	1010.0001	01001
7	7	0111	10000.0001	01010
8	8	1000	10001.0001	10000
9	9	1001	10010.0101	10001
10	A	1010	10100.0101	10010
11	B	1011	10101.0101	10100
12	C	1100	100000. 101001	10101
13	D	1101	100010.001001	11000
14	**E**	**1110**	**100100.110110**	**11001**
15	F	1111	100101.001001	11010

26. Representing real numbers

From binary-decimal number conversion in Section 2.1, we already know that a real number can be represented by using a pair of numbers, for the whole part and the fraction part, respectively. For instance, $\pi \approx 3.1415927$ can be represented in binary as 11.0010010000111111011011, after decimal-to-binary conversion.

What happens in reality is more sophisticated. Real numbers are represented in computers as **floating-point numbers**. Similar to the *scientific notation* of numbers, they use two fixed-point numbers for the exponent and the significant parts, respectively. For instance, a possible floating-point representation for π is

$$\pi \approx 31415927 \times 10^{-7}$$

where -7 is the exponent and 31415927 is the significant (also called mantissa).

The most widely used floating-point number representations are the *IEEE 754 floating-point standard*. The IEEE 754 32-bit format uses one bit (the leftmost bit) for the sign, 8 bits for the exponent, and 23 bits for the significant. It can represent $\pi \approx 3.1415927$ with a precision up to the 7th digit after the decimal point. The IEEE 754 64-bit format uses one bit for the sign, 11 bits for the exponent, and 52 bits for the significant. It can represent $\pi \approx 3.141592653589793$ with a precision up to the 15th digit after the decimal point. The 64-bit format doubles the precision of the 32-

bit format. Thus, the IEEE 754 32-bit format is also called the single-precision format, and the 64-bit format called the double-precision format.

Let us look at the representation $\pi \approx 31415927 \times 10^{-7}$ more carefully. This number could also be represented as $\pi \approx 3.1415927 \times 10^{0}$ or $\pi \approx 0.31415927 \times 10^{1}$. Such multiple representations of the same value may cause confusion.

We can achieve representation uniqueness with **normalized significant**, i.e., placing the binary point to the right of the leftmost non-zero bit of the significant. Thus, π is uniquely represented as $\pi \approx 1.10010010000111111011011 \times 2^{00000001}$ in binary notation, or $\pi \approx 1.5707964 \times 2^{1}$. Since the left-most bit is always 1 for any non-zero numbers, it can be omitted to save one bit of representation.

The IEEE 754 standard also uses **biased exponent** to enable fast exponent comparison operation. That is, an exponent bias of 127 is added to the exponent value for 32-bit representation, which is subtracted when interpreting a number's value. Thus, the exponent 00000001 becomes 1+127=128, or 10000000 in binary notation. The final IEEE 754 32-bit representation for π is shown in Fig. 5.6. Similarly, the 64-bit representation for π is

0**10000000000**1001001000011111101101010100010001000010110100011000.

Besides normalized significant and biased exponent, the standard has several additional clever designs, especially in representing and handling exceptions and errors. Besides normal values, the IEEE 754 standard also includes representations for not so normal floating-point numbers, such as positive and negative infinities ($\pm\infty$), subnormal numbers (representing underflowing values), and Not a Number values (NaNs, such as trying to find $\sqrt{-5}$).

All the above systematic thinking helps ensure the algebraically completeness of floating-point arithmetic, such that many arithmetic errors, one of which caused the loss of an Ariane 5 rocket in 1996, could be avoided. Being a sophisticated abstraction, IEEE 754 is widely used in smartphones, laptops, servers to supercomputers. Professor William Kahan was awarded the Turing Award for his fundamental contributions to numerical analysis, such as embodied in IEEE 754.

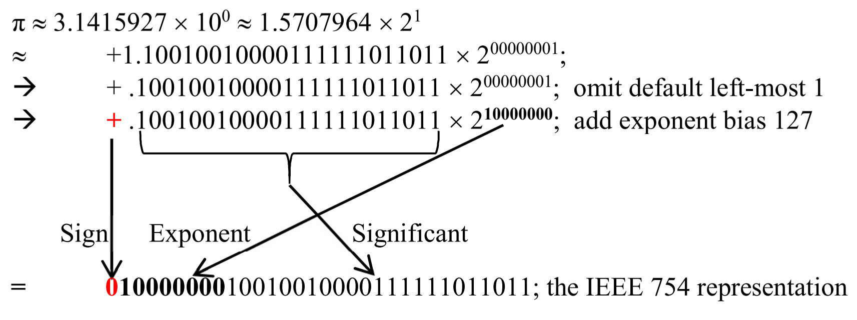

π ≈ 3.1415927 × 10⁰ ≈ 1.5707964 × 2¹
≈ +1.10010010000111111011011 × 2^00000001;
→ + .10010010000111111011011 × 2^00000001; omit default left-most 1
→ + .10010010000111111011011 × 2^10000000; add exponent bias 127
Sign
Exponent
Significant
= 01000000010010010000111111011011; the IEEE 754 representation

Fig. 5.6 Representation of π in IEEE 754 32-bit floating-point standard

27. Test if two floating-point numbers are equal

Table 1.2 already hints that in general, computer representations of integers and characters are exact, but computer representations of real numbers, i.e., floating-point numbers, are often approximate. Any computer has finite word lengths and finite memory capacity. It cannot hold the exact value of a digital symbol that requires an infinite number of bits, such as some real numbers.

The program in Fig 5.7 generates the following seemingly inconsistent outputs:

```
> go run ./testPoint123.go
0.1+0.2 == 0.3
0.1+0.2 != 0.3
0.1+0.2 == 0.3
>
```

The code shows that it is a wrong way to use the double-equal operator "==" to compare two floating-point expressions. Instead, we should compare the absolute difference against a small threshold value, e.g., $|(X+Y)-Z| < 10^{-12}$.

```
package main
import "fmt"
import "math"
func main() {
        if 0.1 + 0.2 == 0.3 {
                fmt.Println("0.1+0.2 == 0.3")
        } else {
                fmt.Println("0.1+0.2 != 0.3")
        }
        X := 0.1            // var X float64 = 0.1
        Y := 0.2
        Z := 0.3
        if X + Y == Z {
                fmt.Println("0.1+0.2 == 0.3")
        } else {
                fmt.Println("0.1+0.2 != 0.3")
        }
        if math.Abs(X+Y - Z) < math.Pow(10, -12) {
                fmt.Println("0.1+0.2 == 0.3")
        } else {
                fmt.Println("0.1+0.2 != 0.3")
        }
}
```

Fig. 5.7 Code testPoint123.go illustrating the strange phenomenon: 0.1+0.2 != 0.3.

28. ASCII, Unicode, and UTF-8.

ASCII is a US standard for encoding English text, covering 128 symbols. Unicode is an international standard that encodes over 143000 characters in 154 languages, including over 70,000 Chinese characters, also known as CJK Unified Ideographs. Unicode also encodes other symbols, such as format characters, control characters and emoji symbols. A dominant implementation of Unicode is **UTF-8** (8-bit Unicode Transformation Format), which is capable of encoding all characters in Unicode. By January 2020, 94.6% of character encodings for the websites world-wide use UTF-8, while fewer than 0.1% use ASCII.

UTF-8 uses one to four bytes when encoding different character sets. The first byte of UTF-8 (0X00 to 0X7F to be exact) has the same encodings as ASCII, as shown in Table 5.3. Thus, only one byte is needed for an ASCII character. The UTF-8 values for the 128 ASCII characters range from 0x00 for the NUL character to 0X7F for the DEL character. On the other hand, a Gothic character needs four bytes. Three bytes are needed for a Chinese character, also called Hanzi. For instance, the Chinese character 志 has a Unicode value of U+5FD7, and a UTF-8 value of 0XE5BF97 (3 bytes for E5 BF 97). For the Euro sign "€", the Unicode value is U+20AC and the UTF-8 value is 0XE282AC. For the Greek capital letter Omega "**Ω**", the Unicode value is U+03A9 and the UTF-8 value is 0XCEA9.

Table 5.3 ASCII, Unicode, and UTF-8 representations of five typical characters

Symbol	Description	ASCII	Unicode	UTF-8	Bytes needed by UTF-8
T	English capital letter T	0X54	U+0054	0X54	1
Ω	Greek letter Omega	N/A	U+03A9	0XCEA9	2
€	The Euro sign	N/A	U+20AC	0XE282AC	3
志	A Chinese character	N/A	U+5FD7	0XE5BF97	3
𐍈	A Gothic letter	N/A	U+10348	0XF0908D88	4

29. Review of bit, byte, character, integer, array, and slice

In Chapters 1 and 2, we introduced the rudiment concepts of six data types: bit, byte, character, integer, array, and slice. Here we put these data abstractions together in one place, to better see their distinctions and differences. We pay special attention to how these data abstractions are stored in memory and how they are represented when printing out. The six data types are illustrated in Fig. 5.8.

A quick way to understand these data types is to execute the following program.

```
X := byte(63)                              // X is a byte variable
fmt.Printf("Decimal: %d\n", X)             // Decimal: 63
fmt.Printf("Hex: %X\n", X)                 // Hex: 3F
fmt.Printf("Character: %c\n", X)           // Character: ?
fmt.Printf("Binary: %b\n", X)              // Binary: 111111
```

```
var S [5]byte = [5]byte{'h','e','l','l','o'}          // S=[104, 101, 108, 108, 111]
var byteSlice []byte = S[1:4]                          // byteSlice=[101, 108, 108]
fmt.Println("array S = ", S)               // Array S = [104 101 108 108 111]
fmt.Println("byteSlice = ", byteSlice)                 // byteSlice = [101 108 108]
```

Try to execute the same code and see what happens when "X := byte(63)" is replaced by "X := 63" and "X := 8364". The meanings of variables for byte, int and array are straightforward. The left-most bit is the sign bit in the int type.

The slice variable byteSlice is a data structure describing a section of an underlying array (S[1], S[2], S[3]), i.e., the array section S[1:4] starting at index 1 and has a length of 3. Note that S[4] is not part of S[1:4].

Slices can also be created with the built-in **make** function. The function call

make([]int, n+1)

allocates an array of length n+1 and returns a slice that refers to that array. All elements of the array are type int and initially set to zero.

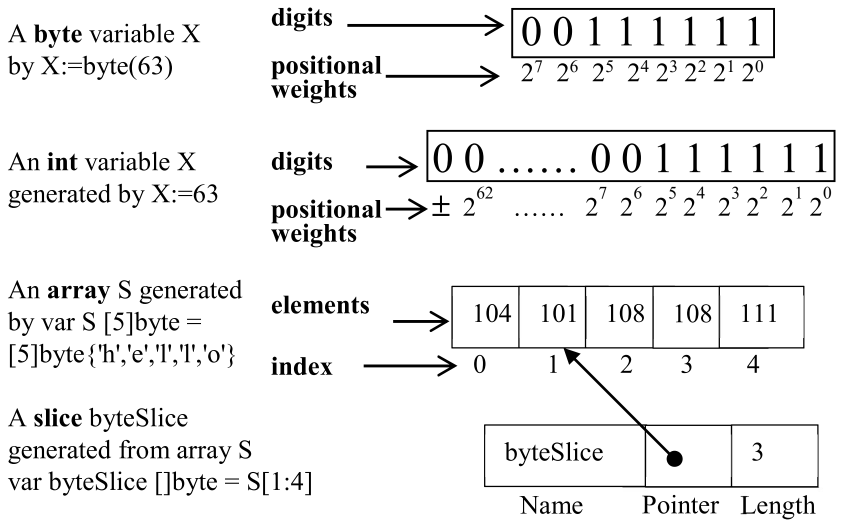

A **byte** variable X by X:=byte(63)
digits
0 0 1 1 1 1 1 1
positional weights
2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0
An **int** variable X generated by X:=63
digits
0 0 ...... 0 0 1 1 1 1 1 1
positional weights
$\pm$ 2^{62} ...... 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0
An **array** S generated by var S [5]byte = [5]byte{'h','e','l','l','o'}
elements
104 101 108 108 111
index
0 1 2 3 4
A **slice** byteSlice generated from array S var byteSlice []byte = S[1:4]
byteSlice
3
Name
Pointer
Length

Fig. 5.8 Differences among byte, int, array, and slice

Note that we only use four data types in the Go language practices, i.e., byte, int, array and slice. So how to treat character and bit? Two practices follow.

- A character such as the question mark '?' is represented as a byte value 00111111, which is equivalent to an 8-bit unsigned integer 00111111=63. The two data types **byte** and **uint8** are equivalent.
- To operate on a bit, we can operate on a byte or an integer containing that bit.

Example 45. Inverting the least significant bit of a byte

Suppose we want to invert the least significant bit (the right-most bit) of a byte 0011111**1**, to obtain 0011111**0**. We cannot directly do it with one operation but need a sequence of operations such as the following:

```
x := byte(63)     // assign 63$_{10}$=00111111$_2$ to variable x
v := ^x           // bitwise NOT of x, i.e., v=11000000
v = v & 0x1       // bitwise AND to retain the right-most bit of v, i.e.,
                        v= 11000000 & 00000001
x = x & 0xFE      // bitwise AND to clear the right-most bit of x, i.e.,
                        x= 00111111 & 11111110
x = x | v         // bitwise OR to invert the last bit of x, i.e.,
                        x=00111110 | 00000000=00111110
```

Let us use a sequence of equations to show how the above code works. To invert the least significant bit of 0011111**1**, the code executes as follows:

x = 0011111**1**	Given input
$v = \overline{0}\,\overline{0}\,\overline{1}\,\overline{1}\,\overline{1}\,\overline{1}\,\overline{1}\,\overline{1} = 1100000\mathbf{0}$	Bitwise NOT
v = 11000000 & 00000001 = 0000000**0**	Bitwise AND
x = 00111111 & 11111110 = 0011111**0**	Bitwise AND
x = 00111110 | 00000000 = 0011111**0**	Bitwise OR

☶

Example 46. Replacing the least significant 2 bits of a byte

Suppose we want to replace the least significant 2 bits of a byte 001111**11**, with the least significant 2 bits of another byte 001010**10,** to obtain 001111**10**. We can realize this with the following sequence of operations:

```
x := byte(63)     // assign 63$_{10}$=00111111$_2$ to variable x
v := byte(42)     // assign 42$_{10}$=00101010$_2$ to variable v
v = v & 0x3       // bitwise AND to retain the right-most 2 bits of v, i.e.,
                        v= 00101010 & 00000011 = 00000010
x = x & 0xFC      // bitwise AND to clear the right-most 2 bits of x, i.e.,
                        x= 00111111 & 11111100 = 00111100
x = x | v         // bitwise OR to replace the last 2 bits of x, i.e.,
                        x=00111100 | 00000010 = 00111110
```

The above code replaces the least significant 2 bits of variable x with the least significant 2 bits of variable v. This becomes clearer with the following sequence of equations to show how the above code works step by step.

x = 00111111	Given input
v = 00101010	Given input
v = 00101010 & 00000011 = 000000**10**	Bitwise AND
x = 00111111 & 11111100 = 001111**00**	Bitwise AND
x = 001111**00** | 000000**10** = 001111**10**	Bitwise OR

☶

30. Pointers and Addressing Modes

The array data type, such as **var a [100]int**, has an obvious advantage: it is a simple linear arrangement of 100 integers consecutively stored in memory, and an array element a[i] can be referenced by index i. Computer scientists realized a fact about data structure and data layout in memory: although the linear arrangement of an array is easy to understand and operate, allowing nonlinear arrangements, i.e., the elements can jump around, brings flexibility.

Such nonlinear arrangements are realized by a basic mechanism called **pointers**, which is implemented by the **indirect addressing** mode provided by hardware. A pointer holds the *address* of a value, not the value itself.

We already encountered several addressing modes, such as base+index+offset in Chapter 2. Three addressing modes are compared below:

• Immediate mode:	MOV 50, R1; assign the immediate value 50 to R1
• Direct mode:	MOV M[50], R1; assign M[50] to R1
• Indirect mode:	MOV M[M[50]], R1; assign M[M[50]] to R1

Registers are regarded as special memory cells.

In a high-level language such as Go, an ordinary variable holds a value, such as an integer or a byte value. A pointer variable holds the address of a variable.

Example 47. Contrasting a pointer variable to an ordinary variable

Figure 5.9 contrasts a normal variable b and pointer variable p.

Any variable has three attributes: a name, a data type, and a value. When a program is compiled to execute, the variable name is bound to a memory address, called the variable's address, which can be obtained with the '&' operator.

For instance, after the following declaration statement

```
var b bool = true
```

we have a variable named b of Boolean type, with an initial value of true.

The asterisk '*' symbol is used in a declaration statement to declare a pointer variable:

```
var p *bool = &b
```

Here, the pointer variable is named p, which points to a value of Boolean type, with an initial value as the address of b, which happens to be 0xc042058058.

The asterisk '*' symbol is also used in an expression as a **dereference** operator, to obtain the value at the address pointed to by a pointer variable.

For instance, p in an expression denotes the address of b, i.e., 0xc042058058. But *p denotes the value of b, i.e., the value at address 0xc042058058, which is initially true.

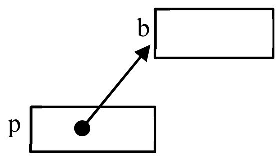
It is left as an exercise to print out the address of p. We can declare another pointer variable q to point to pointer variable p.

```
package main
import "fmt"
func main() {
  b := true          // Boolean variable b
  p := & b           // p holds b's address
  fmt.Println(p)     // Print b's address
  fmt.Println(*p)    // Print b's value
  *p =  false        // Modify b's value
  fmt.Println(b)     // Print b's value
  *p = !(*p)         // Use and modify b's value by negation
  fmt.Println(b)
}
```

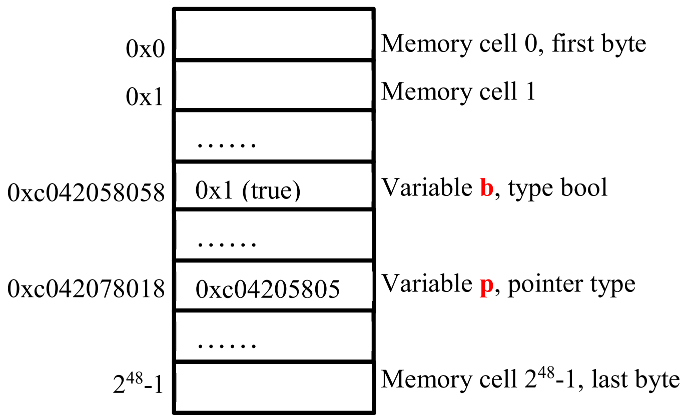
(a) Source code pointer.go

```
> go run ./pointer.go
0xc042058058
true
false
true
>
```

(b) Screen display



(c) Diagram showing that p points to b



(d) Memory layout in a (2^{48}-1)-byte byte addressable memory

Fig. 5.9 Illustration of the pointer concept

Example 48. Computing Fibonacci numbers of arbitrary word length

The fib.dp.big.go program introduced in Example 6 computes Fibonacci number F(n) for arbitrarily big integer n, using the dynamic programming method. A new integer data type big.Int from the math/big package is used to handle representation and arithmetic operations of big integers. However, the code is mostly provided by the Go libraries. It is powerful but has too many details for beginners to understand.

We use a simplified program fib.Uint.go to see how big integers are handled. This program is self-contained, and does not use the math/big package provided by the Go language. This fib.Uint.go program contains the definition of a new data type Uint, which is a slice of 64-bit unsigned integers, to represent an unsigned integer of arbitrarily big word length. An accumulator function Acc is defined to do a = a + b. A String function is defined for converting an unsigned integer into a string of decimal digits, to be printed out by fmt.Printf.

```
package main
import (
        "fmt"
        "math"
)
func main() {
        fmt.Printf("F(100) = %s\n", String(*(fibonacci(100))))
}
type Uint []uint64
func fibonacci(n int) *Uint {
        a := &Uint{0}                       // a = 0
        b := &Uint{1}                       // b = 1
        for i := 1; i < n+1; i++ {
                Acc(a, b)                   //a = a + b
                a, b = b, a
        }
        return a
}
// Code defining Acc and String functions
```

(a) Source code of program fib.Uint.go

```
> go run fib.Uint.go
F(100) = 354224848179261915075
>
```

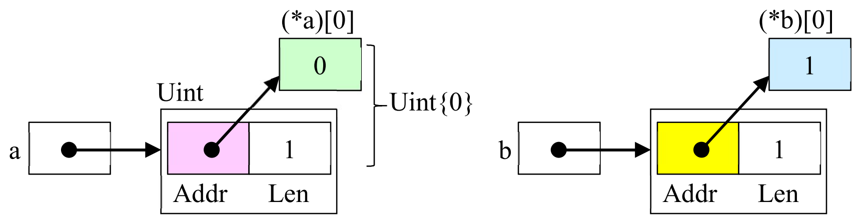
(b) Output by executing program fib.Uint.go

Fig. 5.10 Program fib.Uint.go and its output

We focus on the fibonacci function definition. After the first two statements:

a := &Uint{0}
b := &Uint{1}

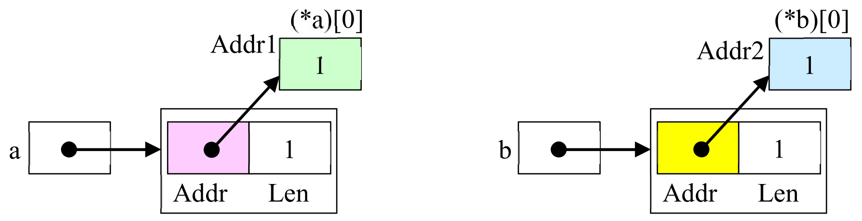
the program creates in memory the following two structures for variables a and b.



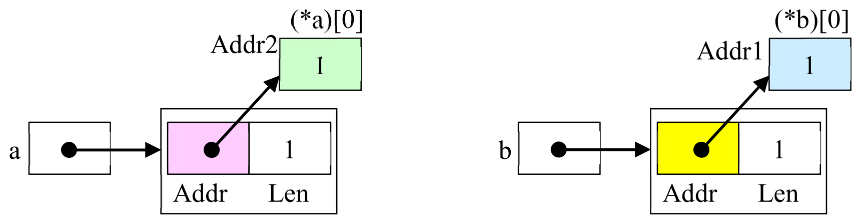
The data type Uint is a slice structure with two fields. One field holds the length. The other field holds the address pointing to the array of 64-bit unsigned integers. The notation Uint{0} denotes a slice of length 1, i.e., the array has one element holding the initial value of 0. The statement a:=&Uint{0} assigns the address of Uint{0} to a. That is, a is a pointer variable holding the address of Uint{0}. Thus, the one element of the array is denoted by (*a)[0].

Two pointer variables a and b represent two unsigned integers of arbitrary length, where each unsigned integer is implemented by a slice of unsigned integers.

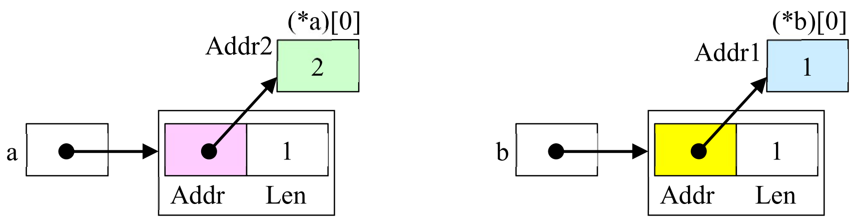
Let us look at the loop in Fig. 5.10a. Assume i is 1. The Acc(a, b) function realizes an accumulative addition a = a + b = 0 + 1 = 1. After the Acc(a, b) statement is executed, the memory contents change to the following configuration.



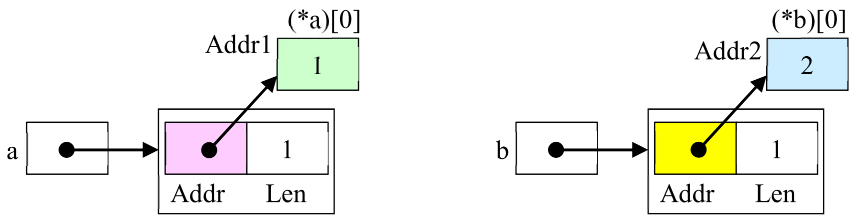
The a,b=b,a statement exchanges a and b to make sure that a is always the smaller of a and b. After the a,b=b,a statement is executed, the new memory configuration follows.



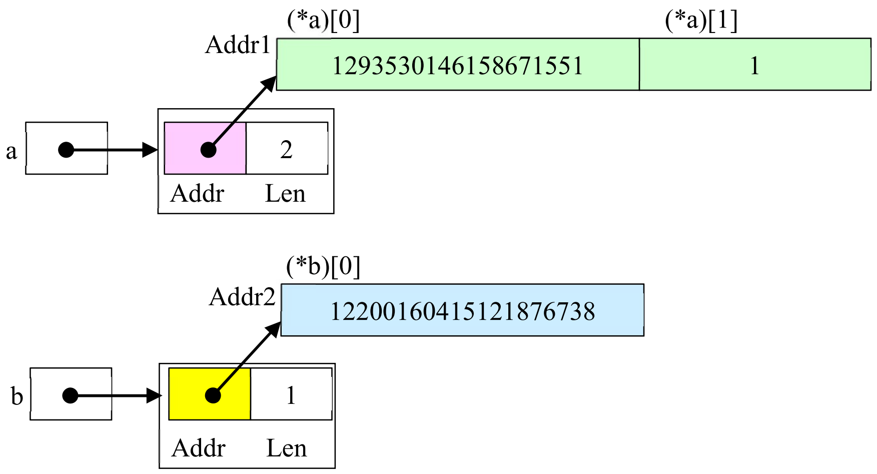
Assume i =2. After the Acc(a, b) statement is executed, the memory contents change to the following configuration.



After the a,b=b,a statement is executed, the new memory configuration follows.



Now assume i=93. This is the first time that the overflow occurs if we use a single 64-bit unsigned integer. After Acc(a, b) is executed, the memory becomes:



We avoid the overflow, which becomes a carry value into the second word, i.e., (*a)[1], of the slice of unsigned integers. Note that (*a)[1]=1 denotes 2^{64}. Change 100 to 93 in the program fib.Uint.go and see that it correctly outputs
F(93) = 12200160415121876738 and F(94) = 19740274219868223167.

☶

31. The File Abstraction

Files are used to organize and persistently store chunks of information. Here, **persistence** means files still exit when electrical power is turned off on a computer. A program **reads** a file from the hard disk into the memory for processing, and **stores** the modified results into the hard disk.

Files are stored in a **file system** of a computer. A file system organizes ordinary files and directories in a tree. A **directory** is a special file that contains other files. Each file, either an ordinary file or a directory, has a file name. Parts of the tree of file names of an example file system is shown in Fig. 5.11 and Table 5.4.

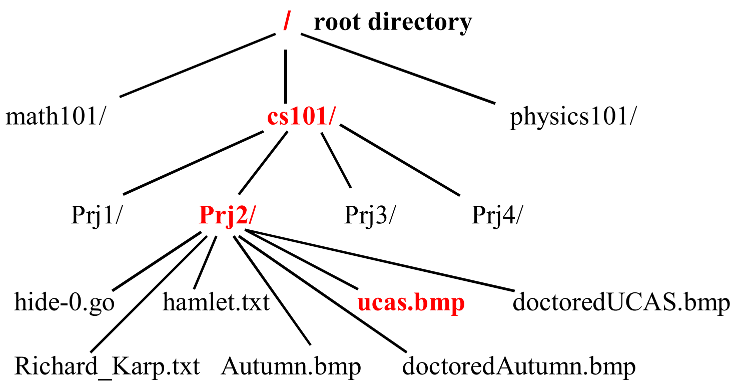


Fig. 5.11 An example tree of files and directories.

Table 5.4 Typical Directories and Files

Typical Directory and Files	Absolute Path Name	Relative File Name
Root directory	/	
Home directory	/cs101/	
Current directory		./
Parent directory		../
Program file to hide text in image	/cs101/Prj2/hide-0.go	hide-0.go
A text file	/cs101/Prj2/Richard_Karp.txt	Richard_Karp.txt
Another text file	/cs101/Prj2/hamlet.txt	hamlet.txt
An image file	/cs101/Prj2/ucas.bmp	ucas.bmp
A doctored image file	/cs101/Prj2/doctoredUCAS.bmp	doctoredUCAS.bmp
Another image file	/cs101/Prj2/Autumn.bmp	Autumn.bmp
Another doctored image file	/cs101/Prj2/doctoredAutumn.bmp	doctoredAutumn.bmp

The **absolute file name** starts from the **root directory** "/" and goes down the tree, adding a slash "/" at each level. Each file has a unique absolute name. The file ucas.bmp has the absolute name /cs101/Prj2/ucas.bmp.

Let us look at Fig. 5.11 and Table 5.4 more carefully. When a user logs into a computer, she/he is automatically at a system-specified default directory, called the **home directory**. The directory in which the user is currently working is called the **current directory** or *working directory*. A pwd (Print Working Directory) command is used to print out the current directory.

Assume the user is at the home directory /cs101/, but wants to work in the working directory /cs101/Prj2/, which contains files for the second project. The user can execute the change directory command "cd Prj2" to change the current directory to /cs101/Prj2/. The sequence of screen printouts follows.

```
The user logs into a computer
>pwd            //the current directory is the home directory
/cs101/
>cd Prj2        //change to directory /cs101/Prj2
>pwd
/cs101/Prj2
```

Once in directory /cs101/Prj2/, the user can access all the seven files there using their shorter **relative file names**. When the current directory is /cs101/Prj2/, the following three names identify the same image file, and all three commands display the same image.

```
>display Autumn.bmp
>display ./Autumn.bmp
>display /cs101/Prj2/Autumn.bmp
```

A file contains data and **metadata**. Data is the bits for the actual information provided by the file. Metadata is data about data, which provides additional information, such as the format and organization of data, the file name, the file size, the access permissions, the time of creation, etc. For instance, Autumn.bmp is a file containing 9144630 bytes and organized as shown in Fig. 5.12, where Pixel Array contains data, while BMP File Header and BMP Info Header contain metadata.

Such a **BMP** (bitmap) image file stores an image as an array of pixels. A **pixel** (picture element) represents a point of an image by three color depth values for the primary colors of red, green, and blue. Each color depth value of the RGB colors is represented as a number of uint8 byte type. Thus, each pixel needs three bytes.

A BMP image file's metadata includes information on the starting places and the sizes of various parts of the image, e.g., the width and the height of the image. Note that the first 54 byte-level addresses are used for metadata. The pixel array for actual image data starts at address 54. The first pixel uses addresses 54, 55, and 56 to store its three RGB color depth values.

The black color is represented when the RGB values are set to (0, 0, 0), that is, when color depth values are all zero. Similarly, the white color is represented when the RGB values are set to (255,255,255), and the red color is represented when the RGB values are set to (255, 0, 0).

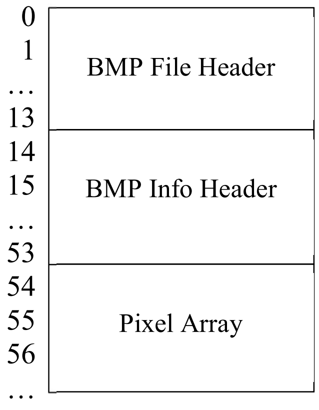


Fig. 5.12 Organization of data and metadata in a BMP image file.

Example 49. Hide text in a picture by doctoring the image file

Let us appreciate the file abstraction in more details by hiding hamlet.txt in Autumn.bmp. That is, we doctor Autumn.bmp such that the content of the text file hamlet.txt are hidden in the picture of the image file Autumn.bmp.

- A careless hiding program will result in a doctored image that is visibly different from the original image, as shown in Fig. 5.13.
- A carefully designed hiding program will result in a doctored image that looks the same as the original image, as is shown in Fig. 5.14.

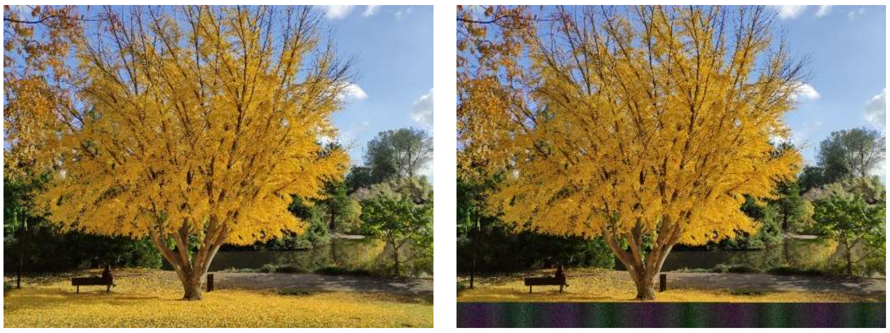


Fig. 5.13 Failure to hide text in a picture: the doctored image (the right picture) obviously differs from the original image (the left picture) Photo credit: Chundian Li

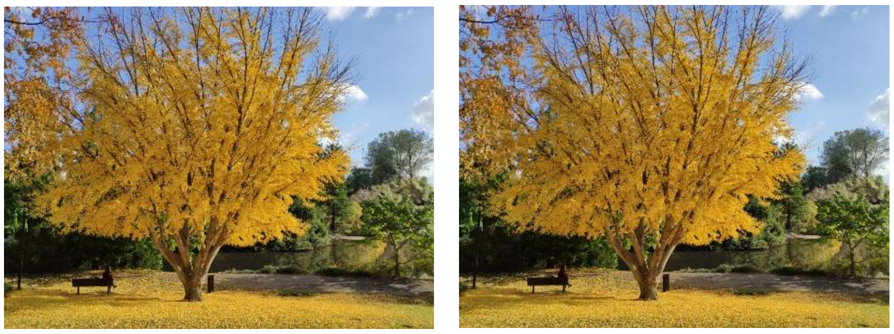


Fig. 5.14 Successfully hiding text in a picture: the doctored image (the right picture) shows no difference from the original image (the left picture) Photo credit: Chundian Li

This information hiding process is realized by the following algorithm, assuming the bmp file format of Fig. 5.12. The corresponding Go program is hide-0.go.

- **Input**: A text file hamlet.txt and an image file Autumn.bmp.
- **Output**: A doctored image file doctoredAutumn.bmp
- **Steps**:

```
1.  Read Autumn.bmp into variable p              // p for picture
2.  Read hamlet.txt into variable t              // t for text
3.  Hide the length of hamlet.txt in the first 32 bytes of the Pixel Array
4.  Hide hamlet.txt in variable p in the remaining bytes of the Pixel Array
5.  Write p to file doctoredAutumn.bmp
```

We now discuss how to develop a program hide-0.go to realize this algorithm. All information is hidden in the Pixel Array, not in the metadata area. Every byte of information is hidden in 4 consecutive bytes of Pixel Array, such that only the *least significant two bits* of each byte of Pixel Array are modified.

Note that variable p is a slice of bytes. Pixel Array starts at address 54. That is, p[54:] holds the Pixel Array, and p[0:54] holds the metadata information.

To read the image file Autumn.bmp into variable p, the program executes

```
p, _ := ioutil.ReadFile("./Autumn.bmp").
```

The function ioutil.ReadFile is provided by Golang in the package "io/ioutil". It accepts a text string of file name and returns the file contents as a byte slice. A Golang function call can return multiple values. If any value is no concern to us, we use an underscore "_" symbol as a placeholder.

Similarly, the contents of the text file hamlet.txt are read into a byte slice variable t by executing the following statement.

```
t, _ := ioutil.ReadFile("./hamlet.txt")
```

Modified contents of variable p are written to file doctoredAutumn.bmp by executing the following statement

```
ioutil.WriteFile("./doctoredAutumn.bmp", p, 0666)
```

where 0666 specifies the access permissions for file doctoredAutumn.bmp, according to the format of Table 5.5. The leading 0 bit indicates that this file is an ordinary file. A directory file should have a leading 1 bit.

Table 5.5 File access permissions (r: read; w: write; e: execute)

Owner			Group			Others		
r	w	e	r	w	e	r	w	e
-	-	-	-	-	-	-	-	-

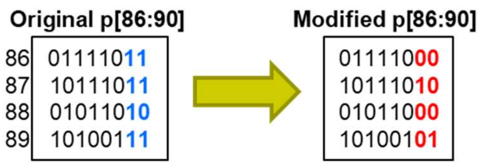
The above WriteFile function call says that variable p is written to file doctoredAutumn.bmp. Being a new file, doctoredAutumn.bmp needs to be created first. The access permissions 666=110110110 says that the file's owner, the group which the owner belongs to, and other users have the rights to read and write the file. But, no user can access the file for execution. Using the shell command "ls -l", we can see the read-write-execute permissions listed as follows:

```
>ls -l doctoredAutumn.bmp
-rw-rw-rw-    …   doctoredAutumn.bmp
>
```

Now let us go through the process of modifying variable p. This is done by repetitively calling a user-defined function modify(txt int, pix []byte, size int). The function saves an integer value *txt* into a byte slice variable *pix*, 2 bits at a time, for a total of *size* iterations.

```
func modify(txt int, pix []byte, size int) {
    for i := 0; i < size; i++ {
        replace last 2 bits of pix[i] with the last 2 bits of txt
        repeat with the next 2 bits of txt
    }
}
```

For instance, to hide character ‘H’ in p[86:90], we call modify(72, p[86:90], 4), and we have: txt is ‘H’ = 72 = 01001000; pix is p[86:90]; size is 4. The loop body is executed 4 times, and p[86:90] is modified as follows.



To hide hamlet.txt in Autumn.bmp and generate doctoredAutumn.bmp, we need to first save the length of the text file hamlet.txt, and then save the contents of hamlet.txt. This is illustrated in Fig. 5.15. The length needs to be saved in case we want to recover the text file from doctoredAutumn.bmp. This is done by executing

```
modify(len(t), p[S:S+T], T)
```

where S is the starting address of the pixel array, 54; and T is the length of the text file to be hidden. We assume the length len(t) is a 64-bit integer. Each byte of pixel array can hide 2 bits. Thus, to hide len(t) we need 64/2=32 bytes of Pixel Array. In other words, when T=32, slice p[54:86] is used for hiding len(t).

To hide the contents of hamlet.txt, we execute the following loop:

```
for i:=0; i<len(t); i++{
        offset := S+T+(i*4)
        modify(int(t[i]), p[offset:offset+C], C)
}
```

where each iteration of the loop body hides one character in four bytes of p at proper addresses. The first character of hamlet.txt is t[0]='H', which is hidden in p[86:90]. The second character is t[1]='A', hidden in p[90:94]. The third character is t[2]='M', hidden in p[94:98].

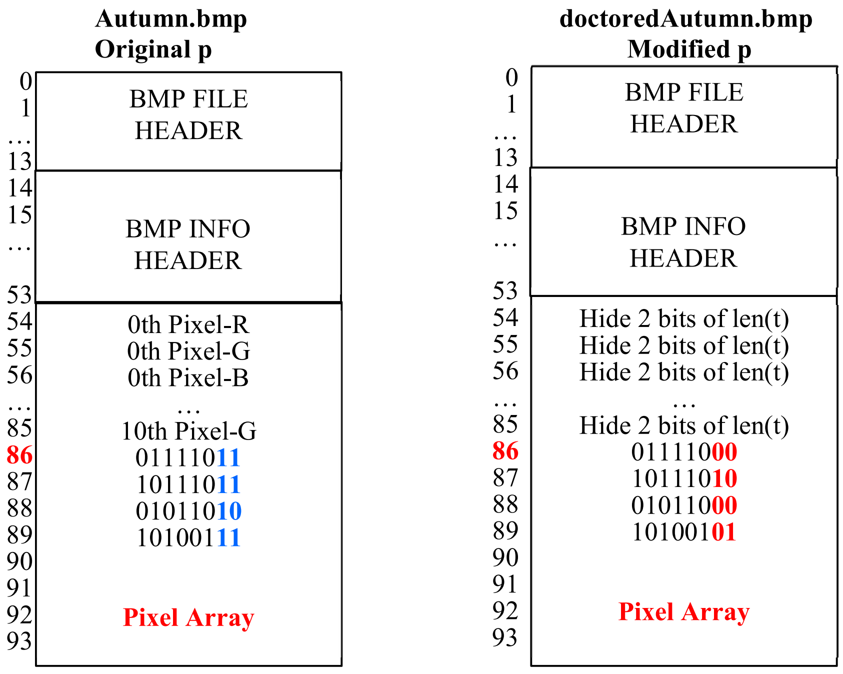


Fig. 5.15 Illustration of how the length and the first character 'H' of hamlet.txt are hidden

5.2.3. Control Abstractions

Five control abstractions are common in a high-level language program. Most of them are intuitive. We only elaborate loop and function.

- **Precedence** in an expression. For instance, in expression x*b+c || i < 7, the precedence ordering is ((x*b)+c) || (i < 7). When in doubt, use parentheses.
- **Sequencing**. By default, a sequence of statements is executed by the syntactic ordering of the sequence, one statement after another.
- **Selection**. An if-then-else statement, also called **conditional**. An example is

  ```
  if i<7 {
    fmt.Println(i)
  }
  ```
- **Loop** iteration. A loop repetitively executes a body of code. Each repetition is called an iteration. The body of code is called the loop body.
- **Function**. A function is defined once and can be called many times.

The following **for loop** statement summarizes the elements of an array.

```
for i := 0; i < n; i++ {
  sum = sum + x[i]
}
```

This loop statement has four parts, as shown in Fig. 5.16. The init statement i:=0 sets the initial value of index i. The loop then checks the condition expression i < n. If false, the loop finishes. If true, executes the loop body, and then execute the post statement i++. After an iteration, the loop repeats by checking the condition again.

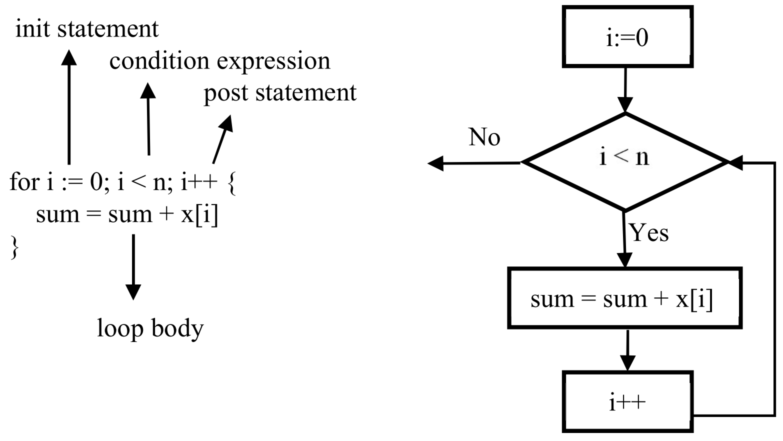
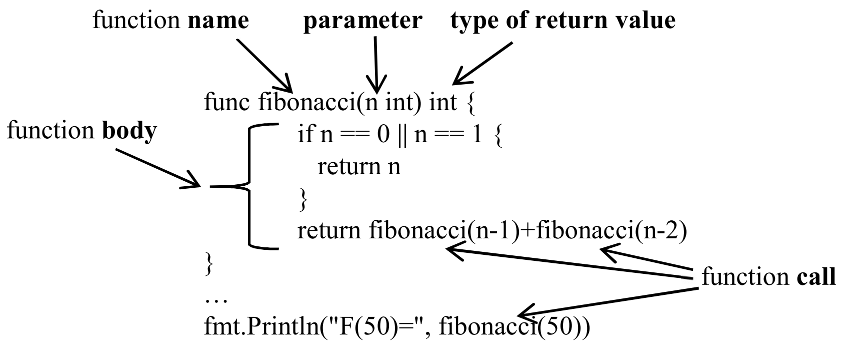


Fig. 5.16 Illustration of a loop statement: its four parts and its control flow

A **function** is a sub-program to be *called* by other statements in a program. An example function definition starts with the keyword func and has four parts, as shown in Fig. 5.17: (1) a function *name* fibonacci, (2) an input *parameter* n of type int, (3) the type int of the function's *return value*, and (4) a function *body* enclosed between { and }.

This fibonacci function call appears three times in Fig. 5.17.



```
func fibonacci(n int) int {
    if n == 0 || n == 1 {
      return n
    }
    return fibonacci(n-1)+fibonacci(n-2)
}
...
fmt.Println("F(50)=", fibonacci(50))
```

Fig. 5.17 A function is defined once and called three times in the above code

When a statement in a program calls a function, the execution environment (called **context**) is first saved before the program executes the function body, so that when the function returns, the program can resume execution with a proper context.

The operating system of a computer divides the memory space of the program (a process) into four segments, called **text**, **data**, **stack**, and **heap**, to hold program's code, static data variables and constants, function calls, and dynamic data, respectively. The context of a function call is saved in the stack segment.

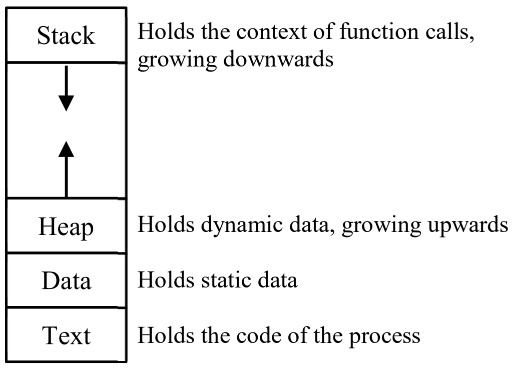

Stack
Holds the context of function calls, growing downwards
Heap
Holds dynamic data, growing upwards
Data
Holds static data
Text
Holds the code of the process

Fig. 5.18 Memory layout of the Text, Data, Stack, and Heap segments of a program (process)

5.3. Modularization

The divide-and-conquer methodology is discussed in algorithmic thinking. There is a similar methodology in systems thinking, called modularization. It has two facets: (1) dividing a system into multiple modules, and (2) composing modules into a higher-level abstraction, also known as system. When discussing modularization, the following points are noteworthy:

- Two modules may be interconnected, but they normally do not overlap.
- Modularization is a special form of abstraction where the information hiding principle is followed.
- Modularization, i.e., how to divide and compose a system, is an art, needing human imagination and creativity.

This section uses a number of progressively more complex examples to demonstrate modularization. Most examples are from the computer hardware.

5.3.1. Combinational Circuits

Combinational circuits are logic circuits using **gates** to realize propositional logic expressions. Figure 5.19 shows four gates realizing the four basic Boolean operators: AND, OR, NOT, and XOR.

X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

X	Z
0	1
1	0

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

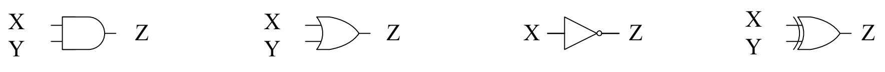
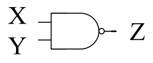


Fig. 5.19 Symbols of four basic gates AND, OR, NOT, and XOR, and their truth tables

Other gates can be realized by such basic gates. In reality, some basic gates are directly implemented by semiconductor circuitry. Figure 5.20 shows how a 2-input NAND gate is realized by a 4-transistor CMOS semiconductor circuitry, where CMOS stands for Complementary Metal Oxide Semiconductor.

When X and Y are both at HIGH voltage level (logic 1), the lower two transistors are both ON and the upper two complementary transistors are both OFF. The output Z is connected to the ground (Vss) at LOW voltage level (logic 0). For any other configuration, the output Z is connected to Vdd and thus at HIGH voltage level (logic 1). The CMOS semiconductor circuitry realizes a NAND gate.

X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0



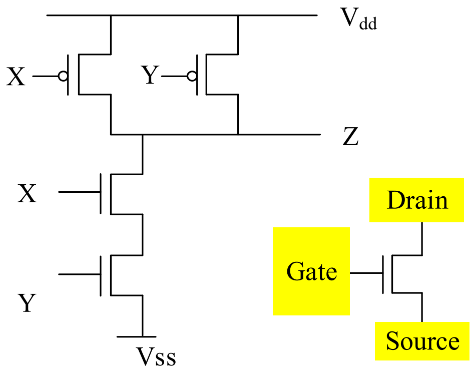


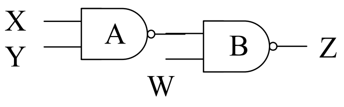
Fig. 5.20 NAND gate: truth table, symbol, and a CMOS implementation. Also shown are the three terminals of a transistor: source, drain, and gate.

The NAND gate is an abstraction with the COG properties:

- It is **c**onstrained by focusing on the Boolean logic functionality, ignoring details such as the voltage levels, number of transistors, power consumption, etc.
- It is **o**bjective. Its functionality is precisely defined by its truth table.
- It is **g**eneralizable. Its functionality can be implemented by a circuit other than the 4-transistor CMOS circuit.

In addition, the NAND gate symbol is much simpler than the 4-transistor CMOS circuit. It embodies the **information-hiding** principle: a module only exposes its interface and visible behaviors, but hides internal details and internal behavior. This is further illustrated in Fig. 5.21, which shows three equivalent representations of a combinational circuit: a Boolean expression, a logic circuit, and a CMOS circuit. The Boolean expression and the logic circuit hide the details of the CMOS circuit, and are much simpler.

$$Z = \overline{(\overline{X \cdot Y}) \cdot W}$$



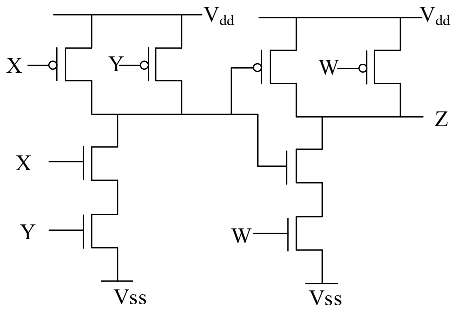


Fig. 5.21 A combinational circuit: Boolean expression, logic diagram, and CMOS circuit diagram

32. Various Adders

In this UKA unit, students are shown several examples to see how to compose logic gates into various systems to do addition. Then as an exercise, students are asked to design a subtractor. Assume X and Y are unsigned integers, we want to compute Z = X + Y using gates.

Example 50. Full adder and ripple-carry adder

A very simple adder is a 1-bit adder, called **full adder**. The "full" here means that it considers also the carry-in and the carry-out bits, in addition to the two addend bits X, Y and the resulting sum bit Z. Figure 5.22 shows the full-adder symbol and its implementing logic circuit. Students are asked to verify that the correct Boolean expressions for the two outputs are $Z = X \oplus Y \oplus C_{in}$ and $C_{out} = (X \cdot Y) + (X \oplus Y) \cdot C_{in}$, respectively.

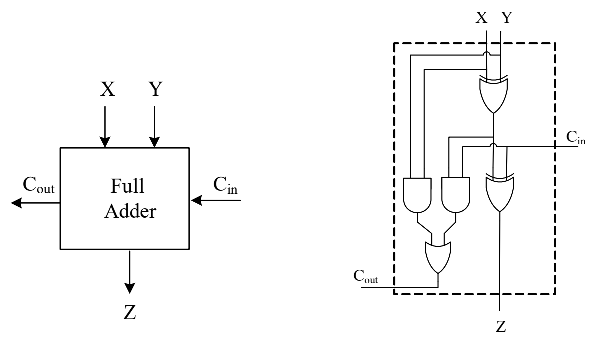


Fig. 5.22 Full adder: symbol and logic circuit diagram.

We can form an n-bit adder by cascading n full adders, as shown below for n=4.

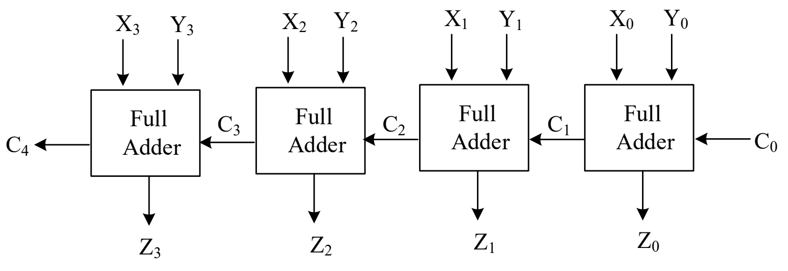
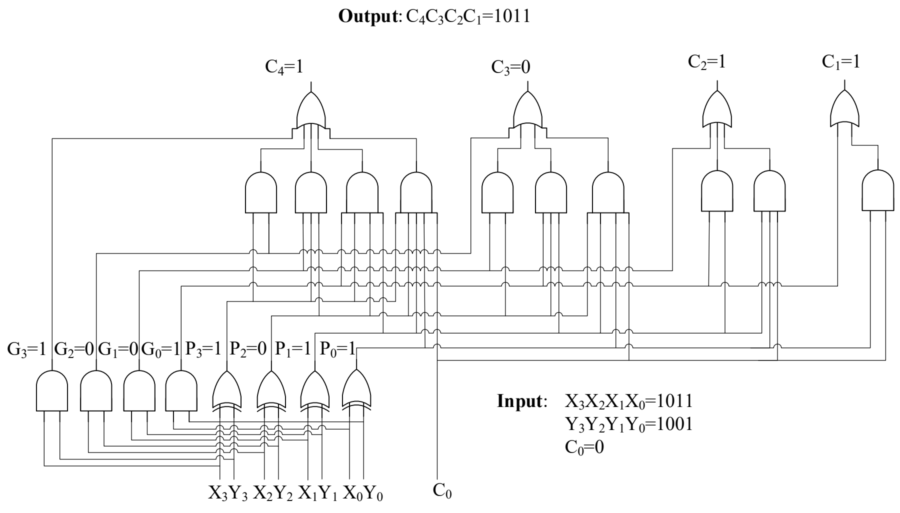


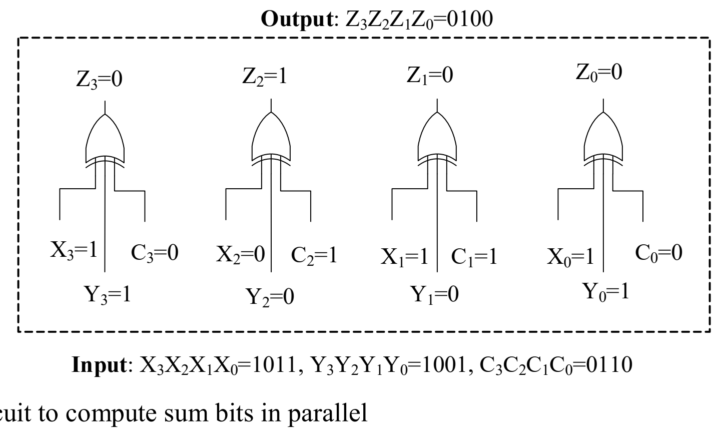
Fig. 5.23 A ripple-carry adder by cascading 4 full adders to form a 4-bit adder.

Example 51. A faster adder

A ripple-carry adder serially generates the carry bits and the sum bits. A more efficient adder generates the carry and the sum bits in parallel. Such a 4-bit parallel adder is shown below, which computes X+Y=1011+1001 = 10100, with an overflowing carry bit of C_4=1. The trick is to compute all carry bits in parallel. The overflowing carry bit C_4 does not depend on lower carry bits any more.



(a) Circuit to compute carry bits in parallel



(b) Circuit to compute sum bits in parallel

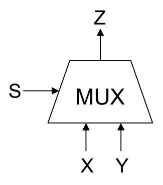
Fig. 5.24 A faster adder with parallel computing of carry bits.

☶

Example 52. An adder-subtractor controlled by multiplexers

The arithmetic-logic unit (ALU) of a processor supports many operations, not just the addition. How does it do that? By using control circuitry to select which operation to perform. The simplest control circuit is the 2-to-1 **multiplexer** in Fig. 5.25. The trapezoid is the multiplexer symbol, which selects one of the two input values X and Y as the output value Z, based on the selection value S. In other words, the multiplexer implements $Z = S \cdot Y + \overline{S} \cdot X$.

S	X	Y	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



S	Z
0	X
1	Y

Fig. 5.25 A 2-to-1 multiplexer: truth table, symbol, simplified truth table

Figure 5.26 shows a 4-bit adder-subtractor designed using four multiplexers. The circuit is easy to understand by noting that subtracting Y from X, i.e., X - Y, is equivalent to adding the two's complement of Y to X, i.e., X + (-Y). For instance, 5-5=5+(-5) → $0101 + \left(\overline{0}\,\overline{1}\,\overline{0}\,\overline{1} + 0001\right)$ =0101+1011=10000.

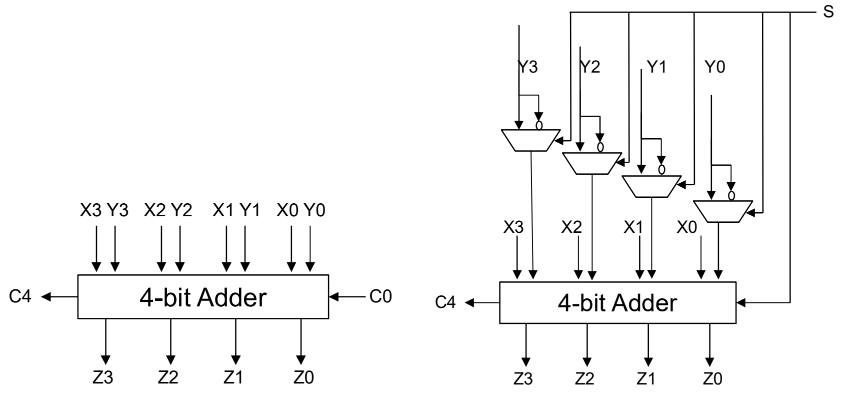


Fig. 5.26 A 4-bit adder and a 4-bit adder-subtractor, both in two's complement representation

5.3.2. Sequential Circuits

Compared to combinational circuits, sequential circuits have one more type of components: state circuits. Thus, **sequential circuit = combinational circuit + state circuit**. With states, a system can execute multi-step computational processes. Each step computes two types of values: the current output values and the next state values. Both are computed from the current input values and the current state values.

States in hardware circuits are implemented by two types of basic circuits: (1) memory cells, and (2) **flip-flops**, also known as latches, which are logic circuits with feedback wires. Many sequential circuits use flip-flops to hold state values.

33. Various types of memory cells

Four terms are often used for memory technology: DRAM, SRAM, NVM, ROM.

- Volatile memory means its contents are lost when power is turned off. Volatile memory is usually faster than non-volatile memory (NVM). There are two common types of volatile memory: DRAM and SRAM.
- Non-volatile memory means its contents are kept even when power is turned off. There are two common types of non-volatile memory: read-only memory (ROM) and read-write NVM.

A DRAM (dynamic random access memory) cell consists of a transistor and a capacitor and represents the state as the charge on the capacitor. This simplicity makes it inexpensive. However, capacitor leaks electricity. Thus, DRAM needs to constantly refresh its contents, once every 7.8-128 μs.

An SRAM (static random access memory) cell consists of six transistors. It is more expensive than DRAM but much faster. When the word line W is on, the bit line B is connected to the state Q to read or write. Suppose we want to write a 1 to the cell. Then B is set to 1, which causes Q to be HIGH (1) and the left-lower transistor to be on, turning $\overline{Q}$ to be LOW (0).

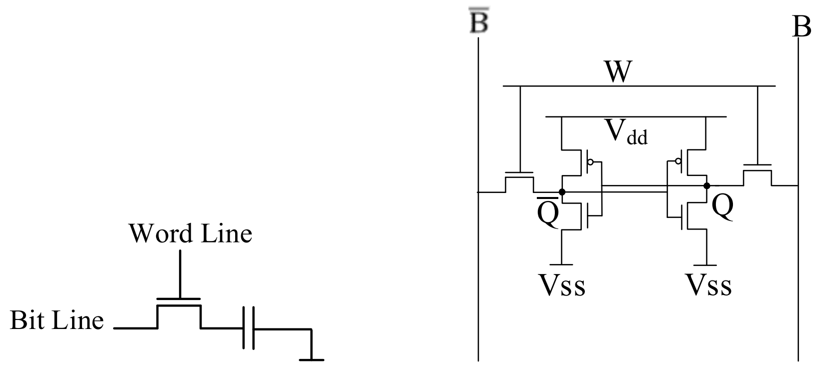


Fig. 5.27 A DRAM cell (left) and an SRAM cell (right) for storing one bit of state.

Read-only memory (ROM) often hardwires its contents into the memory hardware, thus does not lose its contents. The downside is that it cannot be written again. When the computer power is turned on, the computer usually fetches its first instruction from some ROM devices. Read-write NVM devices can be written again. A common example is the flash memory in a student's USB memory stick, also known as the USB flash drive or USB thumb drive.

34. A logic circuit with feedbacks: the delay flip-flop

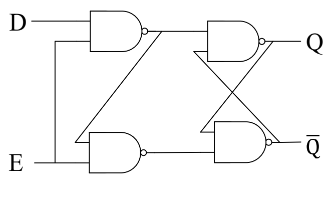
In logic thinking, we try to avoid circular reasoning such as the barber paradox. But it turns out that adding feedback wires to a normal combinational circuit provides a new capability: now logic circuits can support states. Such circuits are called flip flops. We show below a **D flip-flop**, for *delay flip-flop*, which is widely used in computer circuits.

The D flip-flop is implemented with four NAND gates, with three feedback wires that are absent in normal combinational circuits. Its behavior is characterized by its truth table. When the Enable signal E is OFF (0), the D flip-flop maintains its state, that is, $Q_{next} = Q$. When the Enable signal E is ON (1), the D flip-flop changes its state to the D input value, that is, $Q_{next} = D$.

In practice, the system clock signal is often used for the Enable signal E. The clock signal, often written as CLK, is a special signal that alternates its value between LOW (0) and HIGH (1). Each cycle of 0 and 1 is called a **clock cycle**. The number of clock cycles in a second is called the **clock frequency**. Suppose a computer's processor (CPU) has a clock frequency of 3 GHz, i.e., 3 giga cycles per second. This translates to a clock cycle of 1/(3 GHz) = 0.33 ns.

Let us look at the behavior of a D flip-flop during a clock cycle, when E is replaced by CLK. When CLK=0, the D flip-flop maintains its state. When CLK=1, the D flip-flop changes its state to the value of D. Thus, after one clock cycle, the state Q of the D flip-flop changes its value to that of D. This behavior is why the flip-flop is called the *delay* flip-flop: its state output value delays one clock cycle from the input value D.

E	D	Q	Q_{next}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



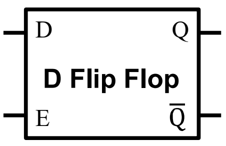


Fig. 5.28 The D flip-flop: its truth table, gates implementation, and symbol.

35. A general organization of sequential circuits

Any sequential circuit can be organized as shown in Fig. 5.29. It is a circuit comprised of three sub-circuits. The state circuit consists of one or more D flip-flops, where each flip-flop can hold one bit of state. The combinational circuit F generates the current output value Out(t) from the current input value In(t) and the current state value State(t). The combinational circuit G generates the next state value State(t+1) from the current input value In(t) and the current state value State(t). In equation form, we have

- Out(t) = F(In(t), State(t))
- State(t+1) = G(In(t), State(t))

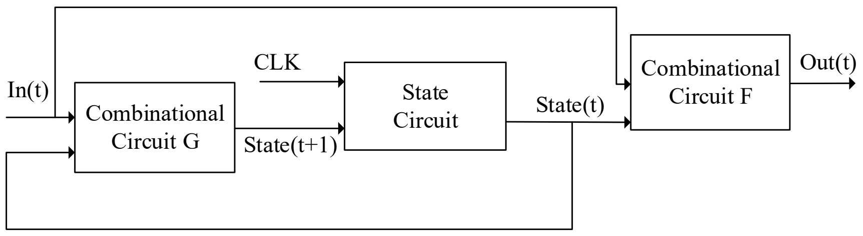


Fig. 5.29 A typical organization of sequential circuits.

Designing a sequential circuit (let's call it the system) can follow this simple procedure: (1) find out the number *n* of bits needed for holding the system's states, and then use *n* D flip-flops to form the state circuit; (2) according to the system's requirements, design the combinational circuits F and G.

36. Serial adder and subtractor

This UKA unit asks students to go through the multiple steps of a serial addition process, to see how to design a sequential circuit and how a sequential circuit works. In each step, this 4-bit adder does a 1-bit full addition. Then as exercises, students are asked to design a 4-bit serial subtractor and an n-bit serial subtractor.

Example 53. A 4-bit serial adder

Design a 4-bit serial adder of unsigned integers, which implements

$$Z_3Z_2Z_1Z_0 = X_3X_2X_1X_0 + Y_3Y_2Y_1Y_0$$

in 4 steps, where each step does a 1-bit full addition. Verify the correctness of the design by executing the following addition

$11_{10} + 9_{10} = 1011_2 + 1001_2 = 10100_2 = 20_{10} = 4_{10}$ and overflow.

To design a sequential circuit, the first question to ask is: how many bits of the state the sequential circuit should have? It turns out that we can use the state to denote the current carry bit. Doing a binary addition serially, i.e., one bit at a time, we only have one carry bit to remember. Thus, we only need one D flip-flop to hold one bit of state, which can have two state values, q_0 and q_1, to denote the current carry value to be 0 and 1, respectively.

From the semantics of the binary addition of unsigned integers, we can derive a state-transition table and the equivalent state transition diagram of the automaton for the 4-bit serial adder, as shown below. Note that X, Y, Z, Q, and Q_{next} denote the current bits of the input X, the input Y, the output Z, the state, and the next state, respectively. The notation XY/Z attached to an arrow needs a couple of examples to explain. 01/1 says that in state q_0, if XY=01, then output Z=1 and stays in state q_0. 11/0 says that in state q_0, if XY=11, then output Z=0 and transition to state q_1.

Q	X	Y	Z	Q_{next}
q_0	0	0	0	q_0
q_0	0	1	1	q_0
q_0	1	0	1	q_0
q_0	1	1	0	q_1
q_1	0	0	1	q_0
q_1	0	1	0	q_1
q_1	1	0	0	q_1
q_1	1	1	1	q_1

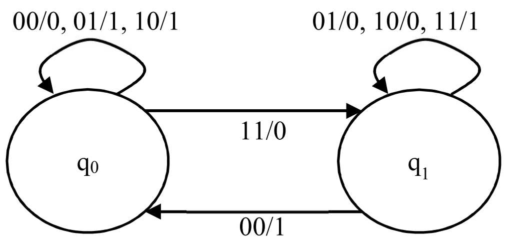


Fig. 5.30 Automaton for the 4-bit serial adder: state-transition table and state transition diagram

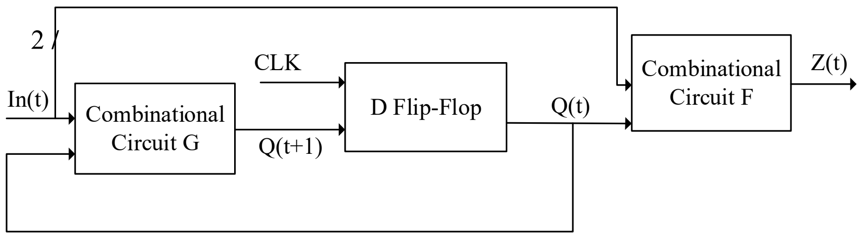
From the general organization of sequential circuits, i.e., Fig. 5.29, we can obtain the first diagram shown in Fig. 5.31 for the serial adder. The enable signal of the D flip-flop is the clock signal CLK. The input In(t) now denotes two variables X and Y. The second diagram in Fig. 5.31 further simplifies by using notations closer to the state-transition table. From the transition table, we can easily derive the following Boolean expressions for the two combinational circuits F and G.

- Z(t) = F(In(t), Q(t))
- Q(t+1) = G(In(t), Q(t))

which are equivalently rewritten as the following expressions:

- Combinational circuit F: $\mathrm{Z} = \mathrm{F}(\mathrm{X}, \mathrm{Y}, \mathrm{Q}) = \mathrm{X} \oplus \mathrm{Y} \oplus \mathrm{C}$
- Combinational circuit G: $\mathrm{Q}_{\mathrm{next}} = \mathrm{G}(\mathrm{X}, \mathrm{Y}, \mathrm{Q}) = (\mathrm{X} \cdot \mathrm{Y}) + (\mathrm{X} \oplus \mathrm{Y}) \cdot \mathrm{Q}$

These are the current output function and the next state function, respectively. They denote the current output bit Z and the carry-out bit C_{out}, respectively. That is, $Q = C_{in}$ and $Q_{next} = C_{out}$.



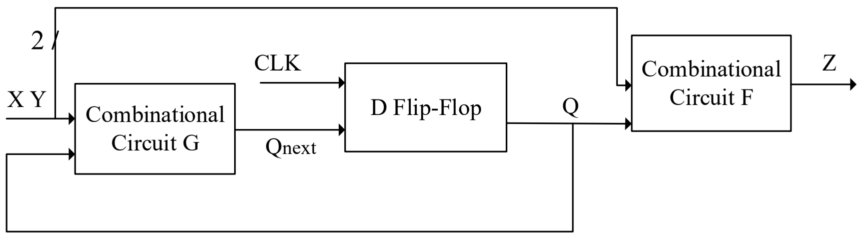


Fig. 5.31 A typical organization diagram of the serial adder

Now we verify the correctness of the serial adder with an example:

$11_{10} + 9_{10} = 1011_2 + 1001_2 = 10100_2 = 20_{10} = 4_{10}$ and overflow.

Note that we have as inputs

$$X_3X_2X_1X_0 = 1011$$
$$Y_3Y_2Y_1Y_0 = 1001$$

and we want the correct output to be

$$Z_3Z_2Z_1Z_0 = 0100$$

with an overflowing carry-out bit of value 1.

This can be done in a sequence of 4 steps, where each step does a 1-bit full addition. We immediately found an error in the above design. It did not set an initial value for the state.

Setting the initial state value (the lowest carry-in bit C_0) to 0, we have:

- Step 1: $X_0 = 1, Y_0 = 1, C_0 = 0$ (Automaton in q_0). $Z_0 = \mathbf{0}, C_1 = 1$ (goto q_1)
- Step 2: $X_1 = 1, Y_1 = 0, C_1 = 1$ (Automaton in q_1). $Z_1 = \mathbf{0}, C_2 = 1$ (stay q_1)
- Step 3: $X_2 = 0, Y_2 = 0, C_2 = 1$ (Automaton in q_1). $Z_2 = \mathbf{1}, C_3 = 0$ (goto q_0)
- Step 4: $X_3 = 1, Y_3 = 1, C_3 = 0$ (Automaton in q_0). $Z_3 = \mathbf{0}, C_4 = \mathbf{1}$ (goto q_1)

The final output is $Z_3Z_2Z_1Z_0 = \mathbf{0100}$, with the carry-out $C_4 = \mathbf{1}$, indicating that an overflow has occurred. The carry-out and the sum output bits are written as

$$C_4Z_3Z_2Z_1Z_0 = 10100.$$

The computational result is correct.

☶

5.3.3. Instruction Set and Instruction Pipeline

The sequential circuits discussed in the last section are actually **synchronous sequential circuits**, because each circuit is driven by a common clock signal CLK. All states of the circuit transition to their respective next states at each clock cycle. Such synchronous sequential circuits behave the same as the automata discussed in Section 3.2. The former implements the latter.

Automata and sequential circuits are basic concepts and widely used in computer systems and computer application systems. This section discusses how they are used in the processor (CPU) of a computer, to implement the instruction pipeline. Each stage of the instruction pipeline is implemented as a sequential circuit.

Every processor has an **instruction set**, which is the set of all possible instructions of the processor, organized in a systematic way. The **instruction pipeline** is the hardware that executes the instructions. A 3-stage instruction pipeline is shown in Fig. 5.32. The three stages are instruction fetch (IF), instruction decode (ID), and instruction execute (EX) stages.

An instruction is first fetched from the memory to an *instruction register* (IR) in the processor. It is then decoded to generate proper control signals. The control signals are then applied, together with the clock signal, to drive the multiplexers, thus to execute the instruction and produce the result. For instance, the processing of the MOV 0, R1 instruction stores an immediate value 0 to register R1. It goes through the following three stages.

• Instruction Fetch (IF):	IR $\leftarrow$ M[PC]
• Instruction Decode (ID):	Signals = Decode(IR)
• Instruction Execute (EX):	R1 $\leftarrow$ 0; PC $\leftarrow$ PC+1

Having more pipeline stages can help increase the clock frequency, thus making the processor runs faster. Modern processors have 5~31 pipeline stages.

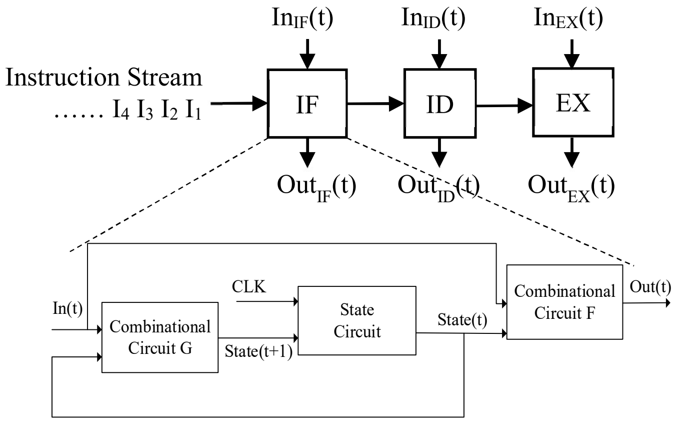


Fig. 5.32 A 3-stage instruction pipeline, each stage implemented by a sequential circuit.

Example 54. Design a simple instruction set

Recall Section 2.3, where we used an assembly language program to realize the following Go loop structure:

```
for i := 2; i < 51; i++ {
  fib[i] = fib[i-1] + fib[i-2]
}
```

The code snippets of the Go language program and the corresponding assembly language code are shown below side by side, to show their correspondences.

```
fib[0] = 0                              MOV 0, R1
                                        MOV R1, M[R0]  //R0=12 initially
fib[1] = 1                              MOV 1, R1
                                        MOV R1, M[R0+8]
for i := 2; i < 51; i++ {               MOV 2, R2       // i:=2
  fib[i] = fib[i-1] + fib[i-2]    Loop: MOV 0, R1       // label Loop
                                        ADD M[R0+R2*8-16], R1
                                        ADD M[R0+R2*8-8], R1
                                        MOV R1, M[R0+R2*8-0]
                                        INC R2          // i++
                                        CMP 51, R2      // i < 51?
}                                       JL Loop         // if Yes, goto Loop
```

Suppose this is all we want this **Fibonacci computer** to do. That is, it only needs to execute the above 11 instructions. Design an instruction set for this computer.

An instruction consists of *opcode* and *operands*. The design process can follow the following procedure: (1) Determine the types of instructions and decide the opcodes; (2) for each opcode, determine its operands. We may need to do tradeoffs to balance the design of the entire instruction set.

This **Fibonacci computer** has five registers visible to the user: FLAGS, PC, R0, R1, and R2. There are six different types of instructions, and each is assigned an opcode, as is shown in Table 5.6. Note that the three instructions (1) MOV 0, R1; (2) MOV 1, R1 and (3) MOV 2, R2 belong to one type of instruction: it moves an immediate value to a register. We only need three bits to hold the six opcodes.

Table 5.6 The opcodes of the instruction set of the Fibonacci computer

Instruction Type	Opcode	Semantics
MOV to Register	000	Assign an immediate value to a register
MOV to Memory	001	Assign the content of a register to M[Address]
ADD	010	R1 + M[Address] $\rightarrow$ R1
INC	011	R + 1 $\rightarrow$ R (R is a register)
CMP	100	Compare to a value, assign the result to FLAGS
JL	101	If FLAGS is '<' (less than), Loop $\rightarrow$ PC

The instructions, each needing 11 bits to represent, are organized in two groups as shown in Table 5.7. The JL Loop (jump to Loop if FLAGS is "less than") instruction needs only one operand to hold the value of Loop, which in this example is 5. The other instructions each need two operands.

Let us first look at the first group of instructions, where the first operand is an immediate value. The JL instruction (opcode 101) needs only 1 operand, which is an unsigned integer of 8 bits (11 – 3 = 8). The JL instruction can jump to a memory address from 0 to 255. For the other 5 instructions, Operand 1 is a 6-bit value, and Operand 2 specifies one of four registers. That is, 00, 01, 10, 11 specify R0, R1, R2, R3, respectively.

In the second group of instructions, Operand 1 is a 6-bit value specifying a memory address, and Operand 2 specifies one of four registers. Based on the base+index+offset addressing mode, the memory address is computed by the following relations:

Address = R0 + R2*I + J, where

I = 0, 1, 2, 4, 8

J = 0, ±4, ±8, ±16

Since the base register R0 and the index register R2 are given and fixed, there are 5×7 = 35 possible (I, J) pairs, and thus 35 distinct values for Operand 1. Since $35<2^6$, 6 bits are enough for Operand 1. For instance, given the initial values of R0 = 12 and R2 =2, R0+R2*8-8 = 12 + 2*8 -8 = 12 + 8 = 20.

Table 5.7 How the 11 instructions of the Fibonacci computer are represented

Opcode 3-bit	**Operand 1 Immediate Value, 6-bit**	**Operand 2 Register, 2-bit**	**Instruction**
000	000000	01	MOV 0, R1
000	000001	01	MOV 1, R1
000	000010	10	MOV 2, R2
011	111111	10	INC R2
100	110011	10	CMP 51, R2
101	00000101		JL Loop
Opcode 3-bit	**Operand 1 Memory Address, 6-bit**	**Operand 2 Register, 2-bit**	**Instruction**
001	R0+R2*0+0	01	MOV R1, M[R0]
001	R0+R2*0+8	01	MOV R1, M[R0+8]
001	R0+R2*0-0	01	MOV R1, M[R0+R2*8-0]
010	R0+R2*8-8	01	ADD M[R0+R2*8-8], R1
010	R0+R2*8-16	01	ADD M[R0+R2*8-16], R1

5.3.4. Software Stack on a von Neumann Computer

Now a computer can execute instructions by its instruction pipeline hardware, we go to see how the computer software is organized. Typically, the software is organized as a layered structure, called **software stack**, on top of a common abstraction of hardware, the von Neumann architecture, as shown in Table 5.8.

Software can be classified as **application software** and **infrastructure software**. The latter provides an infrastructure for applications and can be further divided into **middleware** and **system software**. Middleware is so called because it is between application software and system software. Examples of middleware include database management systems such as MySQL, Web servers such as Nginx, and Web Browsers such as Chrome. System software normally views middleware the same as application software.

We have already used the Linux operating system and the Go programming language with its associated Go compiler. In the Personal Artifact project, students are asked to use the HTML/CSS/JavaScript programming environment to create their dynamic webpages, to be run on a Web server and a Web browser.

The most basic system software is called **firmware**, often hardwired into a memory device. This way, when the power is turned on, the instructions and data in the firmware are ready to go. Thus, firmware is not as hard as hardware, but also not as flexible or soft as software. Firmware is used to realize functionality such as power-on checks and diagnostics, initializing the basic hardware configuration, and loading the operating system from the hard disk. An example firmware is **BIOS**, or the Basic Input/Output System in our personal computers.

Source code refers to programs written in human understandable high-level languages or assembly languages. *Binary code* (executable code) refers to the code of a string of 0's and 1's.

Table 5.8 Examples of software stack on top of a common von Neumann architecture

Software Type			**Example**
Application Software			Scientific computing, Business computing, Personal productivity software; PDF, Search Engine, TikTok, WeChat
Infrastructure Software	Middleware	Databases, Web servers, Web Browsers	MySQL, Nginx, WebServer.go Chrome, Safari
	System Software	Languages, Compilers, Interpreters	C, Go, JavaScript, Python Shell
		Operating Systems	Linux, Android, iOS, Windows
		Firmware	BIOS
von Neumann Architecture			
Hardware			

5.4. Seamless Transition

A modern personal computer can execute a billion instructions per second. How does the computer smoothly transition from one instruction to the next one in this billion-step computational process? We can refine this question by asking three more focused problems:

- How to identify the first instruction?
- How to ensure a single instruction's correct execution?
- How to find the next instruction and transition to it?

Computer systems capable of solving the above three problems are said to have the capability of **seamless transition**. It turns out that computer science has established four principles to support seamless transition, as listed in Box 10.

Box 10. Four Principles of Seamless Transition

- **Yang's Cycle Principle**. A computational process consists of cycles of diversity. The system finishes one cycle and automatically returns to the beginning (of the next cycle), so that processes preserve their kinds.
- **Postel's Robustness Principle**. Be tolerant of inputs, and strict on outputs.
- **von Neumann's Exhaustiveness Principle**. Instructions must be given to the computer in absolutely exhaustive detail, for the computer to execute completely without intelligent human intervention.
- **Amdahl's Law**. Suppose a computational process's execution time can be broken into two portions $(1-f)$ and f, such that $(1-f) > f$. Improve on the common case, i.e., the $(1-f)$ portion, but be aware that speedup is at most $1/f$.

It is not the case that one principle is proposed for a problem. The roles of the four principles can be roughly described as follows:

- Yang's cycle principle addresses part of the third problem as well as the mega question: how does the computer smoothly transition from one instruction to the next one?
- Postel's robustness principle mostly addresses the second problem.
- von Neumann's exhaustiveness principle involves all three problems.

While the above three principles target the functionality of a system, the fourth principle, Amdahl's law, addresses the performance of a system. Seamless transition does not just mean that the system correctly executes computational processes, it also implies that computational processes flow through the system smoothly and seamlessly. Amdahl's law addresses such performance issues.

5.4.1. Yang's Cycle Principle

Any computational process is a multi-step process. A fundamental question is the following: How to ensure the seamless transition from one step to the next step?

Computer science uses a single principle to solve this problem. This principle does not have a name. We call it Yang's cycle principle, or Yang Xiong's principle of cycles, because Yang Xiong presented a similar principle around year 2 BCE, in his classic *The Canon of Supreme Mystery* (太玄经).

《太玄经·周首》：𝌇 阳气周神而反乎始，物继其汇。

Head Zhou (Full Circle) 𝌇: Yang qi comes full circle. Divinely, it returns to the beginning. Things go on to preserve their kinds.

A system executes a computational process in a sequence of cycles. The system finishes one cycle and automatically returns to the beginning (of the next cycle), so that different computational processes preserve their respective kinds.

How can it be done to automatically return to the beginning of the next cycle? Because, at the basic level, a computing system is a sequential circuit. It uses the current state Q to generate the next state Q_{next}.

For instance, recall the typical organization of sequential circuit in Section 5.3.2, which we redraw below. At step *k*, the system is in state Q, which is the output of the D flip-flops. The system uses Q and the current input In to generate Q_{next} through circuit G, and to generate the current output Out through circuit F. This is the functionality of step *k*. When step *k* finishes, Q_{next} replaces Q to become the current state via the D flip-flops, and the system returns to the beginning of step *k* + 1.

The same mechanism works for all steps, enabling different steps to realize different functionalities. This is what "things go on to preserve their kinds" implies.

This support of diversity can be realized by using control signals. In Fig. 5.33, inputs (In) to the two circuits F and G actually consist of Data Inputs and Control Inputs. A control input signal can be used to select the functionality of a circuit, such as to add or to subtract, as shown by the selection signal S in Fig. 5.26.

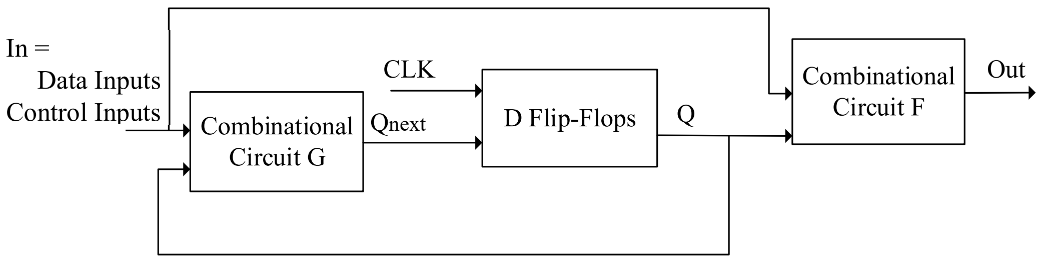


Fig. 5.33 A typical sequential circuit organization

The above principle is a general principle, working for step (or cycle) that could be at different granularities:

- A small granularity is **clock cycle**. For instance, a 1-GHz processor has the clock cycle of 1 ns. At this granularity, the multi-step computational process is a sequence of clock cycles. The system finishes one clock cycle and returns to the beginning of the next clock cycle.
- At the instruction granularity, a computer finishes a step by executing an **instruction cycle**. At this granularity, the multi-step computational process is a sequence of instruction cycles. The system finishes one instruction cycle and returns to the beginning of the next instruction cycle. Note that an instruction cycle is itself implemented by a sequence of clock cycles, to realize pipeline stages of Instruction Fetch, Instruction Decode, and Instruction Execution. Each stage of the instruction pipeline may need one or more clock cycles.
- At the program granularity, a computer finishes a step by executing a **program cycle**. At this granularity, the multi-step computational process is a sequence of program cycles. The system finishes one program cycle and returns to the beginning of the next program cycle. Note that a program cycle is itself implemented by a sequence of instruction cycles.

To recap: a task such as sending a WeChat text message involves the execution of a number of programs. A task is implemented by a sequence of program cycles. A program cycle is implemented by a sequence of instruction cycles. An instruction cycle is implemented by a sequence of clock cycles. In each of these cases, we observe a common principle: when a step finishes, the system returns to the beginning of the next step, until the entire computational process completes.

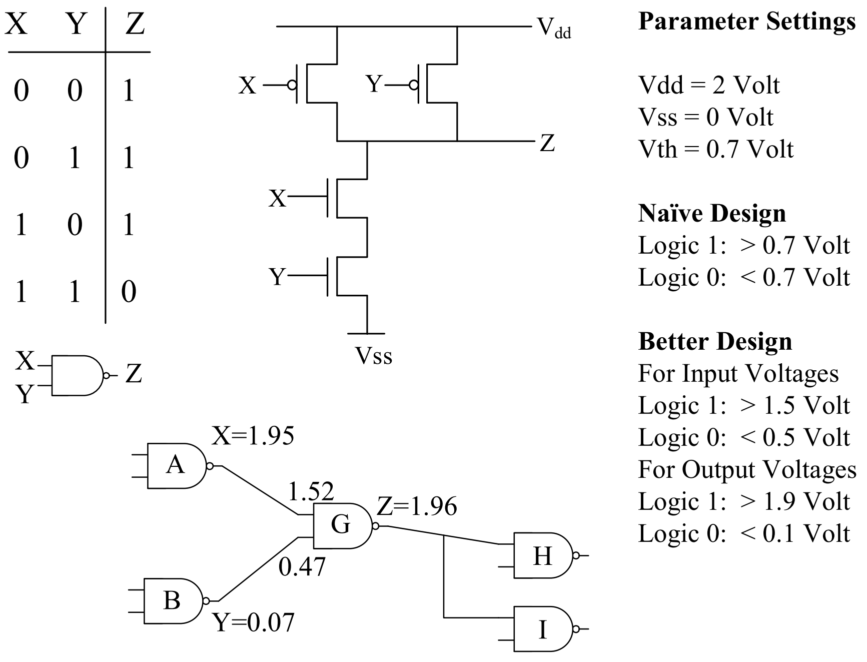
5.4.2. Postel's Robustness Principle

Postel's robustness principle was originally proposed in 1980 by Internet pioneer Jon Postel for robust communication on the Internet. Since then, it has been applied to other systems.

> TCP implementations should follow a general principle of robustness: be conservative in what you do, be liberal in what you accept from others.
>
> Jon Postel, 1980

This principle is often shortened to: be tolerant of input and strict on output. It has an important implication: accumulation of errors, drifts, and distortions can often be avoided. We will use an example of combinational circuit design to illustrate this principle.

Figure 5.34 shows a combinational circuit of five NAND gates. We focus on gate G. It receives inputs from gates A and B, and sends an output to gates H and I. We need to understand how the single step of gate G correctly works. That is, how gate G correctly accepts values from gates A, B and generates correct output value Z.



X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

Parameter Settings

Vdd = 2 Volt
Vss = 0 Volt
Vth = 0.7 Volt

Naïve Design
Logic 1: > 0.7 Volt
Logic 0: < 0.7 Volt

Better Design
For Input Voltages
Logic 1: > 1.5 Volt
Logic 0: < 0.5 Volt
For Output Voltages
Logic 1: > 1.9 Volt
Logic 0: < 0.1 Volt

Fig. 5.34 A combinational circuit demonstrating Postel's Robustness Principle.

Two designs of the CMOS circuit for a NAND gate are shown by contrasting their parameters. Suppose the CMOS transistors have the following characteristics:

- The HIGH voltage Vdd is 2 Volt.
- The LOW voltage Vss is 0 Volt.
- The threshold voltage Vth is 0.7 Volt. That is, the transistor will turn ON if the gate voltage rises over the threshold of 0.7 Volt, and turn OFF when the gate voltage drops below 0.7 Volt.

The naïve design has various problems and is difficult or impossible to implement. The main problem is that the specification "Logic 1: > 0.7 Volt" and "Logic 0: < 0.7 Volt" has no safe margin for deviations and leaves no minimal gap between voltages for logic 1 and logic 0. How many margins should there be? Specifically, should inputs and outputs of a logic gate be treated the same way?

Postel's robustness principle tells us No! We should leave more margin for the input than the output. The Better Design follows this principle.

- *Tolerance on inputs*. The Better Design leaves a 0.5 Volt margin for an input signal to a gate. Only when the input voltage is larger than 1.5 Volt, not 0.7 Volt, should the logic value correspond to logic 1. Only when the input voltage is less than 0.5 Volt, not 0.7 Volt, should the logic value correspond to logic 0. There is a minimal gap of 1 Volt between the voltages for logic 0 and logic 1.

- *Strictness on outputs*. The Better Design leaves a 0.1 Volt margin for an output signal from a gate. Only when the out voltage is larger than 1.9 Volt, not 0.7 Volt, should the logic value correspond to logic 1. Only when the output voltage is less than 0.1 Volt, not 0.7 Volt, should the logic value correspond to logic 0. There is a minimal gap of 1.8 Volt between the voltages for logic 0 and logic 1.

With this design, a gate can tolerate input deviations within 0.5 Volt, but accommodate output deviations within only 0.1 Volt. The design is indeed tolerant of input and strict on output.

Suppose the two inputs of gate G have logic values X=1 and Y=0. The output Z should be equal to 1. But Gate B actually outputs 0.07 Volt. The signal is later raised to 0.47 Volt when it travels through the wire to arrive at gate G. This is fine, because gate G views any input value between 0 Volt and 0.5 Volt as indicating a logic 0. Similar analysis applies to the output of gate A. Gate G views any input value between 2 Volt and 1.5 Volt as indicating a logic 1. For Z=1, the output voltage is not between 2 Volt and 1.5 Volt. The CMOS circuit implementation guarantees that it is a valued between 2 Volt and 1.9 Volt.

5.4.3. von Neumann's Exhaustiveness Principle

The exhaustiveness principle is due to John von Neumann. In the *First Draft of a Report on the EDVAC*, he stated right at the beginning of the document the following principle, when defining an automatic computing system (called *the device*) for problem-solving such as to solve a non-linear partial differential equation (called *this operation*).

> The instructions which govern this operation must be given to the device in absolutely exhaustive detail. They include all numerical information which is required to solve the problem under consideration …
>
> Once these instructions are given to the device, it must be able to carry them out completely and without any need for further intelligent human intervention.
>
> John von Neumann, 1945

Note that instructions in the above quote are not only binary instructions of program code, but *all numerical information*. Obviously, the computer must be given the input data and the processing program code. The computer must also be given information such as the library of functions, context information, etc.

In addition, a computer must be given the answers to the following three questions:

- Where and what is the first instruction, when the computer power is turned on?
- How to determine the next instruction to execute?
- What types of exceptions are there, to normal execution of programs?

We use three examples to demonstrate the exhaustive principle.

Example 55. First instruction to execute when the power is turned on

When the power is turned on in a computer with an x86 processor, the first instruction to execute is at memory address 0xFFFFFFF0, and it contains a jump instruction such as JUMP 000F0000. Address 000F0000 contains the entry instruction for the BIOS code.

Thus, the computer starts by executing the BIOS firmware code to initialize the system and to load the operating system. Adding a jump instruction upfront increases flexibility. For instance, if we want the computer to start by executing another firmware code BIOS-2 at address 000FA000, we can easily do so by changing the contents of address 0xFFFFFFF0 to JUMP 000FA000.

☶

Example 56. Three methods to determine the next instruction to execute

Historically, three methods have been used to determine the next instruction to execute. Modern computers mostly use the program counter (PC) mechanism: the address of the next instruction to execute is stored in the program counter.

Another simple method was used by the revised version of the ENIAC computer: the format of every instruction is expanded to have a field for holding the address of the next instruction.

The earliest method is linear sequencing. The Harvard Mark I computer was designed by graduate student Howard Aiken and built by IBM. It was an electromechanical *Automatic Sequence Controlled Calculator*, meaning that instructions are linearly sequenced. There is no jump or branch. The next instruction is located right after the current instruction on an instruction tape.

The Mark I machine treats instructions and data differently, by storing data in memory and instructions on tape. This **Harvard architecture** is still widely used in the cache units of modern computers. A processor normally has separate instruction cache and data cache. In contrast, the **Princeton architecture** uses a single cache or memory to store both data and instructions.

☶

Example 57. Three types of exceptions

Three types of **exceptions** are considered in almost all computers.

- *Interrupt*. For instance, when the user punches a key on the keyboard, the current instruction finishes and the computer jumps to an interrupt handling subprogram to handle the interrupt, such as writing the punched key value to memory.
- *Hardware error*. When the memory chip becomes faulty, we cannot use the interrupt mechanism, since we cannot fetch the current instruction from memory. An exception handling subprogram stored somewhere else should be executed.
- *Machine check*. This is the "all other" exception, for exhaustiveness.

☶

5.4.4. (***) Amdahl's Law

Amdahl's law was original proposed by Gene Amdahl, a designer of the famous IBM S/360 general-purpose computer. The modern form of this law can be stated succinctly as follows: After enhancing a portion of a system, the speedup obtained is upper bounded by the reciprocal of the other portion's time.

More precisely, suppose a system's execution time is broken into two portions $(1-f)$ and f, such that $(1-f) > f$. Enhancement on the $(1-f)$ portion can lead to a speedup no more than $1/f$.

Here, **speedup** is (time before enhancement) / (time after enhancement).

Example 58. How much speedup is possible by enhancing the processor?

Suppose executing a task, e.g., rendering a short movie, takes 600 seconds. CPU processing takes 90% of time, and accessing the memory and I/O devices takes the remaining10%. How much is the speedup, if the CPU is 9 times faster? How much is the speedup, if the CPU is 9999 times faster?

For the old system, $f = 0.1$, and the execution time of the task is

$$T_{old} = 0.9 \times 600 + 0.1 \times 600 = 540 + 60 = 600 \text{ seconds.}$$

For the enhanced system, the CPU is 9 times faster, implying that the CPU processing time is 1/10 of the old one. The execution time becomes

$$T_{enhanced} = 0.9 \times 600 \;/\; 10 + 0.1 \times 600 = 54 + 60 = 114 \text{ seconds.}$$

The speedup is $600 \;/\; 114 = 5.26$.

Now suppose the new CPU is 9999 times faster. In the enhanced system, the CPU processing time is 1/10000 of the old CPU processing time. The execution time of the task becomes

$$T_{enhanced} = \frac{0.9 \times 600}{10000} + 0.1 \times 600 = 0.054 + 60 = 60.054 \text{ seconds.}$$

The speedup becomes $600 \;/\; 60.054 = 9.99$. Note that as the CPU becomes faster, the speedup approaches but never exceeds $1/f = 1/0.1 = 10$.

☷

Amdahl's law offers two advices for system design.

- *Optimize the common case*. In the above example, the common case of the old system is CPU processing, which takes 90% of execution time. The other portion for accessing memory and I/O devices forms the uncommon case, which takes only 10% of execution time. So, system enhancement, or system optimization, should focus on the common case: CPU processing.
- *Chase the bottleneck*. After making the CPU 9 times faster, i.e., reducing the CPU processing time to one tenth of the old CPU processing time, the total time becomes 114 seconds. Furthermore, accessing memory and I/O devices becomes the common case (also called the **bottleneck**). We should change our optimization target to reducing the time for accessing memory and I/O devices. When the bottleneck changes, so does our target. This is called to *chase the bottleneck*.

37. Instruction Pipeline revisited

Consider again the Fibonacci Computer example in Section 2.3, especially the execution of the MOV 0, R1 instruction in a 3-stage instruction pipeline:

• Instruction Fetch (IF):	IR←M[PC]
• Instruction Decode (ID):	Signals = Decode(IR)
• Instruction Execute (EX):	R1 ← 0; PC ← PC+1

Before the instruction is executed, the computer has the configuration (also known as *state*) shown in Fig. 5.35. Note that several new components of the processor are exposed. These system components are invisible to programmers.

- **IR** is the **Instruction Register**, which holds the instruction being executed.
- **MAR** is the **Memory Address Register**, which holds the memory address to be used to access the memory.
- **MDR** is the **Memory Data Register**, which holds the data value for a memory access (load or store).
- Controller is the control circuitry used to generate control signals of instruction.

After the Instruction Fetch (IF) stage finishes, the computer transitions to the configuration in Fig. 5.36a. After the Instruction Decode (ID) stage finishes, IR guides the controller to generate control signals shown in red in Fig. 5.36a. Figure 5.36b shows the configuration after the EX stage finishes. Note that PC has changed to 6, ready to execute the next instruction, MOV R1, M[R0+R2*8-16].

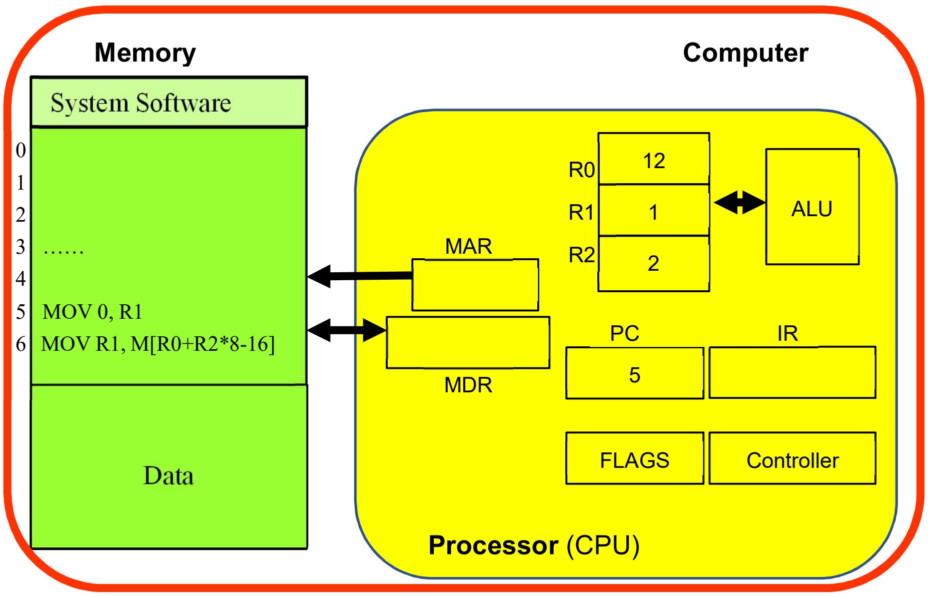
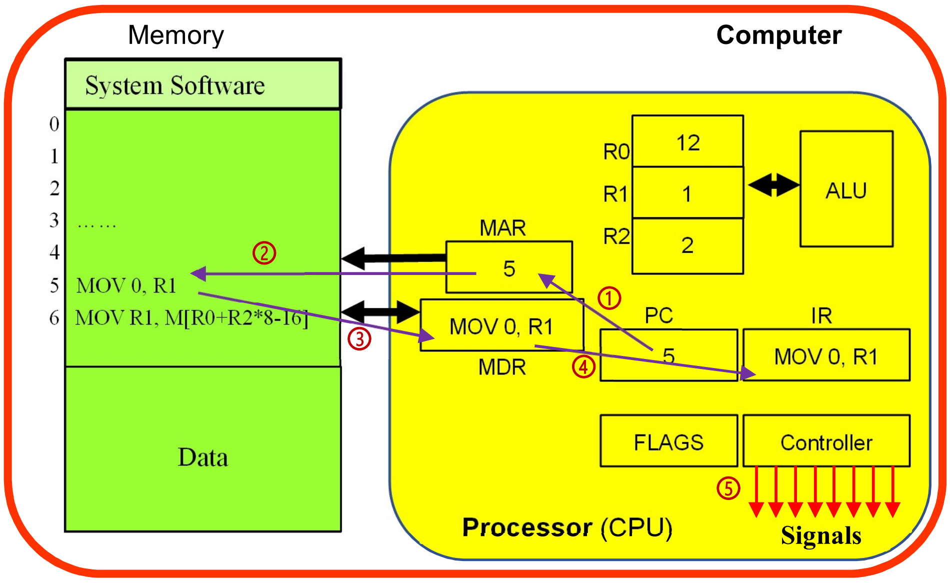
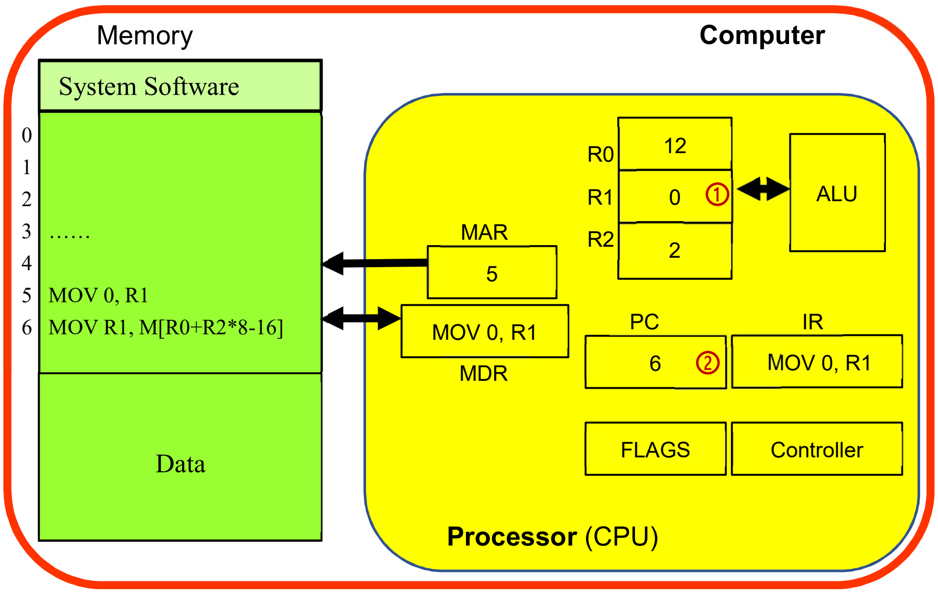


Fig. 5.35 Computer configuration right before the MOV 0, R1 instruction is executed.



(a) After the Instruction Fetch stage (micro operations ① ② ③ ④), and after the Instruction Decode stage (micro operation ⑤)



(b) After the Instruction Execute stage (micro operations ① ②)

Fig. 5.36 Computer configurations after the IF, ID, and EX stages finish, respectively.

The instruction pipeline hardware resource can be shared by multiple instruction executions, through the overlapping of pipeline stages, as shown in Fig. 5.37. When there is no overlapping, pipelining has no speed advantage. Suppose a processor has a clock frequency of 1 GHz without overlapping. It can execute 1 billion instructions per second. With overlapping, the processor's clock frequency can increase to 3 GHz. It can now execute 3 billion instructions per second.

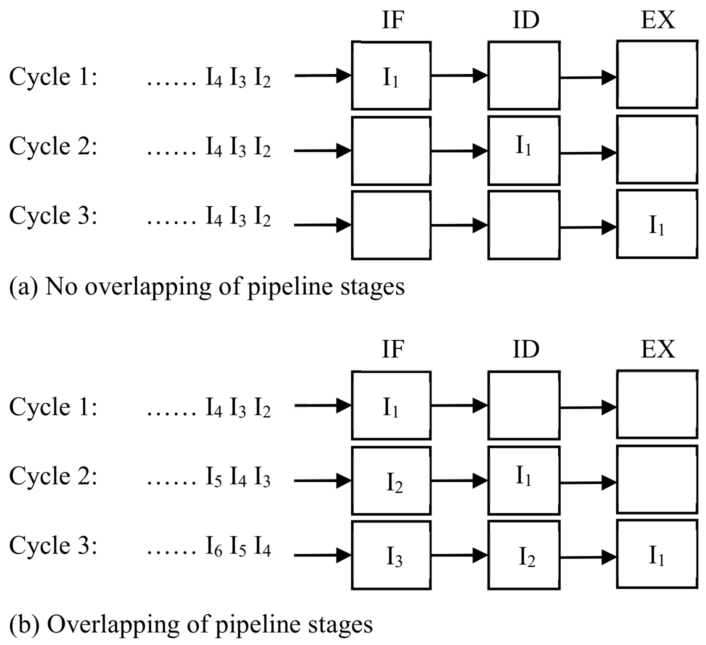


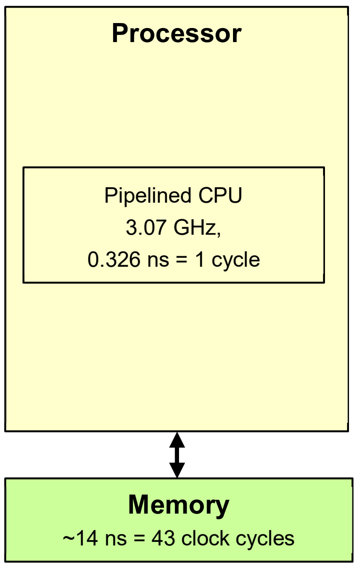
Fig. 5.37 A 3-stage instruction pipeline, without or with overlapping

38. Cache

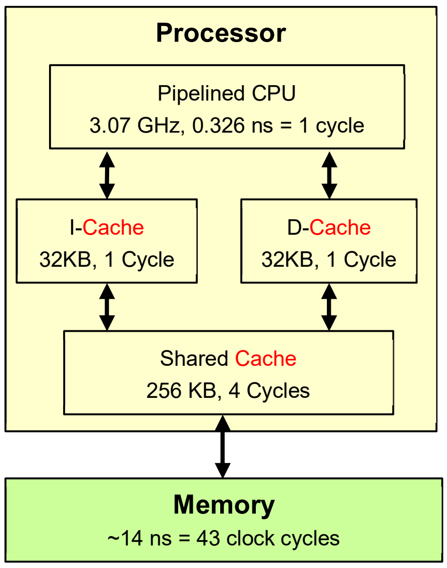
Caching is a method to enhance performance by adding a fast but small memory, called **cache**, between the processor and the memory, as shown in Fig. 5.38.

We note a phenomenon that is widely present in computers today. There is a big gap between the processor speed and the memory access speed. Theoretically, a pipelined processor can execute one instruction per clock cycle. However, to perform one memory access (load or store) needs 14 ns, or 43 clock cycles. This disparity is often called the **von Neumann bottleneck**: the memory is too slow to feed data to the processor.

Figure 5.38 uses a hypothetical Fibonacci Computer to show that caching can alleviate this bottleneck and increase performance by utilizing **locality**: data or instructions recently used or nearby are likely to be used again. We store and reuse frequently used data and instructions in the cache.



(a) A pipelined computer without cache



(b) A pipelined computer with cache

		No Caching	Caching
	MOV 0, R1	43	
	MOV R1, M[R0]	86	
	MOV 1, R1	43	
	MOV R1, M[R0+8]	86	
	MOV 2, R2	43	
Loop:	MOV 0, R1	43	1
	ADD M[R0+R2*8-16], R1	86	1
	ADD M[R0+R2*8-8], R1	86	2
	MOV R1, M[R0+R2*8-0]	86	2
	INC R2	43	1
	CMP 51, R2	43	2
	JL Loop	43	3

(c) Number of clock cycles needed to execute each instruction: without cache vs. with cache

Fig. 5.38 Using cache to enhance performance

Cache can itself be layered. In Fig. 5.38b, the cache is organized as two layers. Layer 1 consists of separate **instruction cache** and **data cache**, each holding 32KB information but is as fast as the CPU. Layer 2 is a single cache shared by both instructions and data. It is larger (256 KB) but slower (4 cycles) than layer 1.

Example 59. Caching enhances computer performance

Let us use the above Fibonacci computer to compute F(92). The loop code showing in red in Fig. 5.38c is executed 92 times, while the code in black only executes once. We thus focus on the 7 instructions in the loop code to estimate performance, ignoring all initialization overheads.

The first instruction of the loop, MOV 0, R1, needs to access the memory once, to load the instruction, which needs 14 ns, or 14/0.326 = 43 clock cycles. All other micro-operations are internal operations. We assume that altogether they can be accomplished in one clock cycle. Due to pipeline overlapping, the MOV 0, R1 instruction itself costs 43 clock cycles. The same analysis goes for the 5th, 6th, 7th instructions, each costing 43 clock cycles. Similar analysis applies to the 2nd, 3rd, and 4th instructions. They each need 86 clock cycles due to needing two memory accesses, one for fetching the instruction, and the other for loading/storing data.

The **peak speed** of a computer is the maximal theoretical number of instructions executed per second. As the joke goes, the vendor of the computer guarantees that you can never exceed this speed. The above Fibonacci computer has a clock frequency of 3.07 GHz, translating to a peak speed of 3.07 GIPS, or Giga instructions per second. The **sustained speed** of the computer is the actual number of instructions executed per second. For the loop code, each iteration of the 7 instructions needs 43*4 + 86*3 = 43*10 = 430 clock cycles = 430 * 0.326 = 140 ns. The sustained speed is 7/140 = 0.05 GIPS.

Efficiency is (sustained speed) / (peak speed), which in this case is 0.05 / 3.07 = 1.63%. In other words, the von Neumann bottleneck renders the computer inefficient, only achieving 1.63% of the processor's peak speed.

Now caching comes to the rescue. All instructions and data can be stored in the I-cache and the D-cache, respectively. Thus, accessing memory only need one clock cycle. If the instruction pipeline works smoothly in full overlapping, each instruction will cost only 1 clock cycle.

However, there are **dependencies**, which cause the pipeline to stall. The third instruction ADD M[R0+R2*8-8], R1 actually does R1 + M[R0+R2*8-8] $\rightarrow$ R1, which depends on the result data R1 from the previous instruction. This is called **data dependency**. The pipeline needs to stall for one extra cycle for data to become available. The 7th instruction JL Loop indicates a jump to the back of the Loop, to execute the first instruction MOV 0, R1 again for the next iteration. However, the first instruction MOV 0, R1 cannot be fetched, until the JL Loop finishes. This is called **control dependency**. The pipeline needs to stall for two extra cycles for the jump instruction to finish.

With caching, each iteration of the 7 instructions needs 1*3 + 2*3 + 3*1 = 12 cycles = 12 * 0.326 = 3.91 ns. The sustained speed is 7/3.91 = 1.79 GIPS, achieving a speedup of 1.79/0.05 = 35.8. The efficiency becomes 1.79 / 3.07 = 58.32%.

To summarize, caching does not necessarily increase peak speed, but can significantly increase sustained speed and efficiency.

☶

39. Parallel Computing

Another approach to enhancing performance is **parallel computing**, which employs multiple processors to execute a computational task. Each processor in such a **parallel computer** is called a **core**. Most examples of this book only use single-core computers, also known as **sequential computers**. However, students need to know that most real computers today, from smartphones to supercomputers, employ multi-core processors.

Example 60. Parallel computing enhances supercomputer performance

Top500 is a list of the world's fastest supercomputers, maintained since 1993. The performance of a supercomputer is tested by executing a benchmark program, called Linpack, for solving a system of linear equations using Gaussian elimination. In other words, it finds $\boldsymbol{x}$ in $\boldsymbol{Ax} = \boldsymbol{b}$, where $\boldsymbol{A}$ is an $N \times N$ matrix, and $\boldsymbol{x}, \boldsymbol{b}$ are two N-dimensional vectors. The equation $\boldsymbol{Ax} = \boldsymbol{b}$ can be explicitly written as the following, when $N = 3$:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Table 5.9. compares two parallel computers, i.e., the Top-1 systems in 1993 and in 2020, respectively. Note that the speed (performance) increased nearly 7 million times in 27 years, from 59.7 GFlop/s in 1993 to 415 PFlop/s in 2020. This is equivalent to an annual growth rate of more than 24%.

Faster computers can solve bigger problems. The problem size *N*, i.e., the size of matrix $\boldsymbol{A}$, increased 391 times. Since the number of operations is $O(N^3)$, the number of 64-bit floating-point operations increased more than 60 million times.

The most important factor contributing to this multi-million-fold enhancement is parallelism, which increased over seven thousand times. In contrast, the clock frequency only increased 68 times.

Table 5.9 Parallel computing enhances performance: Top500 Linpack test comparison

Time of Test	1993	2020	1993-2020 Growth Factor
Top-1 Name	Thinking Machine CM-5	Fujitsu Fugaku	N/A
Problem Size	N = 52,224	N = 20,459,520	392
Speed	59.7 GFlop/s	415,530 TFlop/s	6,960,302
Clock Frequency	32 MHz	2.2 GHz	69
Parallelism	1,024 cores	7,299,072 cores	7,128
Memory	32 GB	4,866,048 GB	152,064
Power	96.5 KW	28,334.5 KW	294
Cost	US $30 million	US $1 billion	33

☷

5.5. Exercises

1. Regarding the objectives of systems thinking, which of the following statements are/is correct?

   (a) Systems thinking aims to coping with complexity, meaning that it reduces complexity from $O(N^4)$ to $O(\log N)$, where N is the problem size.
   (b) Systems thinking aims to being thorough, meaning that it considers all necessary and unnecessary details of the system, leaving no stones unturned.
   (c) Systems thinking emphasize flexibility, by designing a specific system for a target application scenario in an ad hoc manner.
   (d) Systems thinking uses abstractions to cope with complexity.

2. Consider the big endian vs. little endian representations of numbers. Which of the following statements are/is correct?

   (a) Big endian places the least significant byte in the smallest address.
   (b) Big endian places the most significant byte in the smallest address.
   (c) Big endian is better than little endian.
   (d) Neither the big endian nor the little endian representation is better than the other representation.

3. Which of the following list orders abstractions from low-level to high-level?

   (a) Computer, transistor, motherboard, CPU microchip
   (b) Computer, CPU microchip, transistor, motherboard
   (c) Transistor, CPU microchip, motherboard, computer
   (d) Transistor, motherboard, CPU microchip, computer

4. Which of the following list orders abstractions from high-level to low-level?

   (a) Computer, transistor, logic gate, memory
   (b) Computer, logic gate, memory, transistor
   (c) Computer, memory, logic gate, transistor
   (d) Transistor, memory, logic gate, computer

5. Which of the following list orders abstractions from high-level to low-level?

   (a) Computer, transistor, combinational circuit, sequential circuit
   (b) Computer, sequential circuit, combinational circuit, transistor
   (c) Combinational circuit, computer, transistor, sequential circuit
   (d) Transistor, sequential circuit, combinational circuit, computer

6. The operating system of a computer uses one abstraction to manage all application software programs. What is it?

   (a) Instruction
   (b) Program
   (c) Code
   (d) Process

7. The operating system of a computer uses one abstraction to manage all application software programs. What is it?
   (a) High-level language program
   (b) Machine code program
   (c) Processor
   (d) Process

8. In a positional number system, the two 6's in 0x06F6 denote different values, as they are in different positions. Which of the following statements are/is correct?
   (a) The leftmost 6 denotes 6_{10} and the rightmost 6 denotes 1536_{10}.
   (b) The leftmost 6 denotes 1536_{10} and the rightmost 6 denotes 6_{10}.
   (c) The leftmost 6 denotes 6×16^2 and the rightmost 6 denotes 6×16^0.
   (d) The leftmost 6 denotes six hundreds and the rightmost 6 denotes six ones.

9. Consider the hexadecimal number a = 0x06F6 in a positional number system. Which of the following statements are/is correct?
   (a) The base is 16 and the digit set is {0, 1, 2, …, 14,15}.
   (b) The value of 0x06F6 is $\sum_{i=0}^{3}(a_i \times 16^i) = 6 \times 16^2 + 15 \times 16 + 6 = 1782$.
   (c) The base is 16 and the digit set is {0, 1, 2, …, E, F}.
   (d) The base is 10 and the digit set is {0, 1}, since computer only uses binary numbers of 0 and 1.

10. In IEEE 754 floating-point single precision format, the string of 32 bits 01000000010010010000111111011011 represents the decimal value 3.1415927. Assume a string S = 11000000010000000000000000000000 of 32 bits are given. Which of the following statements are/is correct?
   (a) String S is a negative number.
   (b) String S is a positive number.
   (c) String S has an exponent value of 10000000 = 128_{10}.
   (d) String S has an exponent value of 1.
   (e) String S has a significant value of 0.1 = 0.5_{10}.
   (f) String S has a significant value of 1.1 = 1.5_{10}.

11. In IEEE 754 floating-point single precision format, the string of 32 bits 01000000010010010000111111011011 represents the decimal value 3.1415927. Assume a string S = 11000000010000000000000000000000 of 32 bits are given. Which of the following statements are/is correct?
   (a) String S represents the decimal value 0.5×2^{128}.
   (b) String S represents the decimal value -0.5×2^{128}.
   (c) String S represents the decimal value 0.5×2^{1}.
   (d) String S represents the decimal value -0.5×2^{1}.
   (e) String S represents the decimal value 1.5×2^{128}.
   (f) String S represents the decimal value -1.5×2^{128}.

(g) String *S* represents the decimal value 1.5×2^1.
(h) String *S* represents the decimal value -1.5×2^1.

12. Consider representing 0 as an IEEE 754 floating-point number in single precision format. Which of the following statements are/is correct?

(a) The representation is 00000000000000000000000000000000.
(b) The representation is 10000000000000000000000000000000.
(c) The representation is 00000000010000000000000000000000.
(d) The representation is 10000000010000000000000000000000.

13. Floating-point numbers are often approximate values of real numbers. Suppose we want to test whether two floating-point variables X and Y have equal values. We may choose the following Go code:

```
(A) if X==Y { … }
(B) if math.Abs(X-Y) < math.Pow(10,-9) { … }
```

Which of the above code are/is correct?
(a) Only A.
(b) Only B.
(c) Either A or B.
(d) Neither A nor B.

14. Mathematically, $2\times2=4$ and $0.1\times0.1=0.01$. This is not always true in cyberspace. What does the following code output?

```
X := 2
Y := 0.1
fmt.Println(X*X == 4, Y*Y == 0.01)
```

(a) false false
(b) false true
(c) true false
(d) true true

15. Assume the following code is given.

```
var S [5]byte = [5]byte{'H','E','L','L','O'}
var byteSlice []byte = S[1:2]
fmt.Printf("%s %d", byteSlice, len(byteSlice))
```

The output is:
(a) HE 2
(b) EL 2
(c) H 1
(d) E 1
(e) L 1

16. Assume the following code is given.

```
X := 53
```

```
P := &X
fmt.Println(*P)
```

Which of the following statements are/is correct?

(a) X is an integer variable.
(b) Expression &X returns the address value of variable X.
(c) P is a pointer variable holding the address of variable X.
(d) Expression *P returns the value of variable X, i.e., 53.

17. Assume the following code is given.

```
X := 53
p := &X
fmt.Println(*p)
```

The output is:

(a) 5
(b) 3
(c) 53
(d) A hexadecimal number representing variable X's address

18. Write a Go program to invert the first bit of 00111111 to obtain 10111111.
19. Write a Go program to replace the most significant two bits of 00111111 with 01, to obtain 01111111.
20. Refer to the image file Autumn.bmp. Which of the following is NOT metadata?

(a) The photographer's name Li Chundian
(b) The access permission -rw-rw-rw-, i.e., 0666
(c) The size of the file
(d) The pixels of the image

21. Refer to the image file Autumn.bmp. Which of the following is NOT metadata?

(a) The time the file was last modified
(b) The RGB information of the picture elements
(c) The BMP File Header
(d) The BMP Info Header

22. Refer to Example 49. The textbook says: the first character is t[0]='H', which is hidden in p[86:90]. Now let us consider the fourth character, which of the following statement is correct?

(a) The fourth character is t[3]='L', which is hidden in p[89:93].
(b) The fourth character is t[3]='L', which is hidden in p[90:94].
(c) The fourth character is t[3]='L', which is hidden in p[98:102].
(d) The fourth character is t[4]='E', which is hidden in p[98:102].

23. Modify program hide-0.go to realize the effect of Fig. 5.13. That is, hide the contents of the text hamlet.txt in the most significant two bits of the pixel array of the image file Autumn.bmp.

24. Precedence, sequencing, selection, and loop are four control abstractions. Which of the following refers to loop?

    (a) Check a Boolean condition to determine which part of code to execute next.
    (b) Execute one statement after another in the syntactical order of the code.
    (c) Evaluate an expression in the order given by the precedence of operators or parentheses.
    (d) Repetitive execution of a body of code for a specific number of times.

25. Assume the following code is given, where array X=[1, 2, 3]

```
sum := 0
for i := 0; i < 3; i++ {
  sum = sum + X[i]
}
Fmt.Println(sum)
```

If the code "for i := 0; i < 3; i++" is replaced by the following code C, what is the output? Put the correct capital letter in the parentheses of each line below.

(a) When C is "for i := 0; i < 1; i++", the output is ()	V: compiling error
(b) When C is "for ; i < 3; i++", the output is ()	W: 1
(c) When C is "for i := 0; ; i++", the output is ()	X: runtime error
(d) When C is "for i := 0; i < 3; ", the output is ()	Y: no termination
(e) When C is "for ; ; ", the output is ()	Z: 6

26. Refer to the recursive program fib-5.go to compute F(5). How many times is the function fibonacci called in executing fib-5.go?

    (a) 11
    (b) 12
    (c) 13
    (d) 14
    (e) 15

27. Refer to modularization in Section 5.3, which of the following statements are/is correct?

    (a) The interior of a module is invisible to other parts of a system. They can only access a module through the module's interface.
    (b) The interior of a module is visible to other parts of a system, so that all modules can cooperate to maximize system performance.
    (c) Two modules are normally isolated. They do not have shared components.
    (d) Two modules should have shared components.
    (e) All of the above four can be used in designing a system. It is up to the system designer to decide.

28. Regarding combinational circuits discussed in Section 5.3.1, which of the following statements are/is correct?

(a) A combinational circuit is a number of interconnected logic gates.
(b) A combinational circuit is a number of logic gates interconnected by wires, with no feedback wires.
(c) A combinational circuit is driven by a clock signal.
(d) A flip flop is a combinational circuit, since it consists of a number of interconnected logic gates.

29. How many 1-input-1-output combinational circuits are there? If two combinational circuits realize the same Boolean expression, they are equivalent and counted as one circuit.

(a) 1
(b) 2
(c) 4
(d) 8

30. How many N-input-1-output combinational circuits are there? If two combinational circuits realize the same Boolean expression, they are equivalent and counted as one circuit.

(a) N
(b) N^2
(c) 2^N
(d) 2^{2^N}

31. Refer to the full adder in Fig. 5.22. Which of the following statements are/is correct?

(a) A full adder receives three inputs, two operand bits and one carry-in bit, to generate a sum output bit and a carry-out bit.
(b) A full adder is a 2-input-2-output combinational circuit.
(c) A full adder is a 3-input-2-output combinational circuit.
(d) The sum output bit is 1, if an odd number of the three input bits are 1.

32. Refer to the fast adder in Fig. 5.24. Denote $G_0 = X_0 \cdot Y_0$, $G_1 = X_1 \cdot Y_1$, and $P_0 = X_0 \oplus Y_0$, $P_1 = X_1 \oplus Y_1$. Which of the following equations are correct?

(a) $C_2 = C_0 \cdot P_0 \cdot P_1 + P_1 \cdot G_0 + G_1$
(b) $C_2 = C_0 \cdot G_0 \cdot G_1 + G_1 \cdot P_0 + P_1$
(c) $C_2 = X_1 \cdot Y_1 + (X_1 \oplus Y_1) \cdot C_1$
(d) $C_2 = C_0 \cdot X_0 \cdot Y_0 + X_1 \cdot Y_1 + C_1$

33. Refer to the fast adder in Fig. 5.24. Denote $G_0 = X_0 \cdot Y_0$, $G_1 = X_1 \cdot Y_1$, and $P_0 = X_0 \oplus Y_0$, $P_1 = X_1 \oplus Y_1$. Which of the following equations are correct?

(a) $C_2 = X_1 \cdot Y_1 + (X_1 \oplus Y_1) \cdot C_1$
(b) $C_2 = G_1 + P_1 \cdot C_1$
(c) $C_1 = G_0 + P_0 \cdot C_0$
(d) $C_2 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0$

34. Regarding sequential circuits discussed in Section 5.3.2, which of the following statements are/is correct?

    (a) A sequential circuit is comprised of combinational circuits and state circuits.
    (b) A state circuit is realized by one or more memory cells or flip flops.
    (c) A flip flop is a combinational circuit.
    (d) A flip flop is an augmented combinational circuit with feedback wires.

35. Regarding sequential circuits discussed in Section 5.3.2, which of the following statements are/is correct?

    (a) A sequential circuit is comprised of combinational circuits and state circuits.
    (b) A state circuit can be realized by one or more delay flip flops.
    (c) A sequential circuit in this book is actually a clock-synchronous sequential circuit, since the state circuit is driven by a clock signal.
    (d) A sequential circuit in this book is actually a clock-synchronous sequential circuit, since the combinational circuits are driven by a clock signal.

36. Regarding memory circuits discussed in Section 5.3.2, which of the following statements are/is correct?

    (a) DRAM needs constant refreshing, to compensate for electric leakage.
    (b) A DRAM cell needs six transistors.
    (c) A 1KB DRAM memory contains over 8000 capacitors.
    (d) A 1KB SRAM memory contains over 8000 capacitors.
    (e) A 1KB ROM memory contains over 8000 capacitors.
    (f) SRAM does not need refreshing, meaning its content is not lost when the power is turned off.

37. Consider the characteristics of memory circuits discussed in Section 5.3.2. Put the correct capital letter in the parentheses of each line below.

(a) Volatile, fast, expensive correspond to ()	U: DRAM
(b) Volatile, slow, inexpensive correspond to ()	V: NVM
(c) Nonvolatile, read-only correspond to ()	W: ROM
(d) Nonvolatile, read-write correspond to ()	X: SRAM

38. Refer to Example 53. Design a 4-bit serial subtractor and verify its correctness by executing 11 – 9 = 2 and 9 – 11 = -2.
39. Refer to Example 53. Design an n-bit serial subtractor and verify its correctness by executing 99…9 – 99…9 and see that the result is indeed 0.
40. Refer to Example 54. Design the instruction set architecture for an Accumulator Computer to execute the following computation, assuming X=[1, 2, 3,…, *N*]

```
sum := 0
for i := 0; i < N; i++ {
  sum = sum + X[i]
}
```

41. Regarding the software stack used in this book, which of the following statements are/is correct?
    (a) The Go compiler is a piece of firmware.
    (b) The Go compiler is a piece of system software.
    (c) The Linux Shell is a piece of application software.
    (d) The Linux Shell is a piece of middleware.
42. Regarding the software stack used in this book, which of the following statements are/is correct?
    (a) The Web browser is a piece of firmware.
    (b) The Web browser is a piece of middleware.
    (c) The Linux operating system is a piece of system software.
    (d) The Linux operating system is a piece of application software.
43. Regarding the cycle principle in Section 5.4, which of the following statements are/is correct?
    (a) An instruction cycle consists of multiple program cycles.
    (b) An instruction cycle consists of multiple clock cycles.
    (c) A program cycle consists of multiple instruction cycles.
    (d) A program cycle consists of multiple clock cycles.
44. Regarding the robustness principle in Section 5.4, which of the following statements are/is correct?
    (a) Be conservative in what you do to others, be liberal in what you accept from others.
    (b) Be conservative in what you accept from others, be liberal in what you do to others.
    (c) Be tolerant of inputs, be strict on outputs.
    (d) Be tolerant of outputs, be strict on inputs.
45. After a jump instruction finishes, the destination instruction should be executed. What happens when the jump instruction finishes?
    (a) The destination instruction is assigned to the program counter PC.
    (b) The address of the destination instruction is assigned to the program counter PC.
    (c) The destination instruction is assigned to the instruction register IR.
    (d) The address of the destination instruction is assigned to the instruction register IR.
46. After a jump instruction finishes, the destination instruction should be executed. What happens when jump instruction finishes?
    (a) The address of the jump instruction is assigned to the program counter PC.

(b) The address of the destination instruction is assigned to the program counter PC.
(c) The address of the destination instruction is assigned to the instruction register IR.
(d) The difference of the addresses of the destination instruction and of the jump instruction is assigned to the instruction register IR.

47. When the following event happens, what exception occurs? Put the correct capital letter in the parentheses of each line below. App exception denotes an exception that occurs in the application program that does not cause hardware error, interrupt or machine check.

(a) Execute a "divide by zero" operation ()	W: App exception
(b) ID stage sees an undefined opcode ()	X: Hardware Error
(c) CPU knows something is wrong but not what ()	Y: Interrupt
(d) The user moves the mouse ()	Z: Machine Check

48. When the Fibonacci Computer is executing code and the following situation happens, what exception occurs? Put the correct capital letter in the parentheses of each line below.

(a) The code does not terminate ()	W: Hardware Error
(b) ID stage sees opcode 111 ()	X: Interrupt
(c) The power is turned off ()	Y: Machine Check
(d) The user clicks the mouse ()	Z: No exception

49. Suppose the execution time of a program is 100 seconds, of which 80% is spent on CPU processing, and 20% on accessing memory and the hard disk. Assume the CPU clock frequency increases from 1 GHz to 1 THz, which is an enhancement of about 1000 times. What is the execution time of the program on this enhanced computer?

(a) 0.02 seconds
(b) 0.2 seconds
(c) 2 seconds
(e) 20 seconds
(f) 200 seconds

5.6. Bibliographic Notes

The chapter quotation by Confucius (孔子) is from the classic 论语, or *Analects*. The original Chinese text is “民可使由之不可使知之”. This thinking on abstraction has multiple interpretations. The quotation by Leslie Lamport is from [1]. Luszczek, *et al*, presents the idea of using a few smartly design benchmarks to represent many applications [2]. Bergman presents the Bergman number system [3]. Discussion on Fibonacci number systems can be found in [4]. The IEEE floating-point number standard is discussed in [5-6]. Alkhatib, *et al*, presents a vision of seamless intelligence of the IEEE Computer Society [7]. The English translation of Head Zhou, quoted in Section 5.4.1, is from [8]. Postel proposed the principle of robustness in [9]. John von Neumann proposed the exhaustiveness principle in 1945. A reprinting can be found in [9]. Amdahl presented the idea of Amdahl’s law in [11], although no formula can be found in that short paper. TOP500 is a website maintaining performance data on the world’s 500 fastest computers [12].

[1] Lamport, L. If you're not writing a program, don't use a programming language. https://www.heidelberg-laureate-forum.org/video/lecture-if-youre-not-writing-a-program-dont-use-a-programming-language.html. 2018.

[2] Luszczek, P. R., Bailey, D. H., Dongarra, J. J., Kepner, J., Lucas, R. F., Rabenseifner, R., & Takahashi, D. (2006, November). The HPC Challenge (HPCC) benchmark suite. In Proceedings of the 2006 ACM/IEEE conference on Supercomputing (Vol. 213, No. 10.1145, pp. 1188455-1188677).

[3] Bergman, G., A Number System with an Irrational Base, Mathematics Magazine, Vol. 31, No. 2 (Nov. - Dec., 1957), pp. 98-110.

[4] Ahlbach, C., Usatine, J., Frougny, C., Pippenger, N.: Efficient Algorithms for Zeckendorf Arithmetic. The Fibonacci Quarterly 51(3), 249–255 (2013).

[5] Severance, C. (1998). IEEE 754: an interview with William Kahan. Computer, 31(3), 114-115.

[6] Kahan, W. (1996). IEEE standard 754 for binary floating-point arithmetic. Lecture Notes on the Status of IEEE, 754(94720-1776), 11.

[7] Alkhatib, H., Faraboschi, P., Frachtenberg, E., Kasahara, H., Lange, D., Laplante, P., ... & Schwan, K. (2015). What will 2022 look like? The IEEE CS 2022 report. Computer, 48(3), 68-76.

[8] Nylan M. The Canon of Supreme Mystery by Yang Hsiung: A Translation with Commentary of the T'ai Hsuan Ching. SUNY Press, 1993.

[9] Postel, J. (1980). DoD Standard–Transmission Control Protocol–RFC 761. University of Southern California. https://datatracker.ietf.org/doc/rfc761/.

[10] Von Neumann, J. (1993). First Draft of a Report on the EDVAC. IEEE Annals of the History of Computing, 15(4), 27-75.

[11] Amdahl, G. M. (1967, April). Validity of the single processor approach to achieving large scale computing capabilities. In Proceedings of the April 18-20, 1967, spring joint computer conference (pp. 483-485).

[12] TOP500, https://www.top500.org Web site.

6. Network Thinking

[The users of the World Wide Web] felt they were part of a system, a new system of humanity collectively producing more and more value.
……
Everybody would find mutual understanding with each other across the Web, rather than fighting each other, sending each other's sons out to fight each other in the fields.

Tim Berners-Lee on the original spirit of the World Wide Web, 2019

This chapter extends the principles in the previous chapters to networks, or more precisely, computer networks. We use the familiar global Internet as the main example. The **Internet** is a global *internetwork* (network of networks) connecting many local networks and computers through the Internet Protocol (IP).

Network thinking studies and utilizes the networking aspect of computing to generate computational and societal values. It has two basic concepts: *connectivity* and *protocol stack*.

Connectivity refers to what and how nodes of a network are connected. We pay special attention to two knowledge units: *naming* and *topology*. What nodes are in the network? How to name them? How to find a specific node? How are the nodes interconnected? Does the network structure change over time? Is the network topology static, dynamic, or evolutionary? Students learn how to use a user-friendly Web URL name, to access a web resource on a specific computer.

A **protocol stack** is a stack of layers of rules governing communication. It is used to enable multiple nodes of a network to communicate with one another, in order to automatically collaborate in accomplishing a common computational task. We introduce how the Web over Internet protocol stack works. Key concepts involved include: message passing, packet switching, interfaces of protocol layers, switch and router. We also prepare students to do the Personal Artifact project by introducing Web programming examples.

In addition, this chapter introduces network laws and responsible computing, as the societal impact is most obvious on the Internet. We discuss *network effect* as exemplified by Metcalfe's law and the viral marketing phenomenon. Examples are used to illustrate issues on security, privacy, and professional code of conduct.

6.1. Network Terms

Often, a computational process involves not just one entity, but also a group of interconnected entities, which may refer to or communicate with one another. Here, an entity may be an abstract or real entity regarding a computer, a person, or a thing. A thing could be a physical thing, a document, an idea, or a concept.

The group of interconnected entities is called a **network**. The entities are called the **nodes** of the network. For instance, all the one billion WeChat users form a network with one billion nodes. As another example, all articles of computing form a network of computer science literature. The nodes (articles) are interconnected by citations. The nodes may refer to one another, but do not necessarily communicate with one another by sending or receiving messages.

A computational process may treat a network as an *object*. For instance, we may design a computational process to compute the topology of the network of WeChat users. Then, the network is an input object, and the topology of the network is an output object. A network may also be the *subject* of a computational process. For example, a network of computers may be used to execute the above task of topology computation: the network computes the topology.

Let us start with some terms of networking, by looking at parts of networks of three universities, shown in Fig. 6.1.

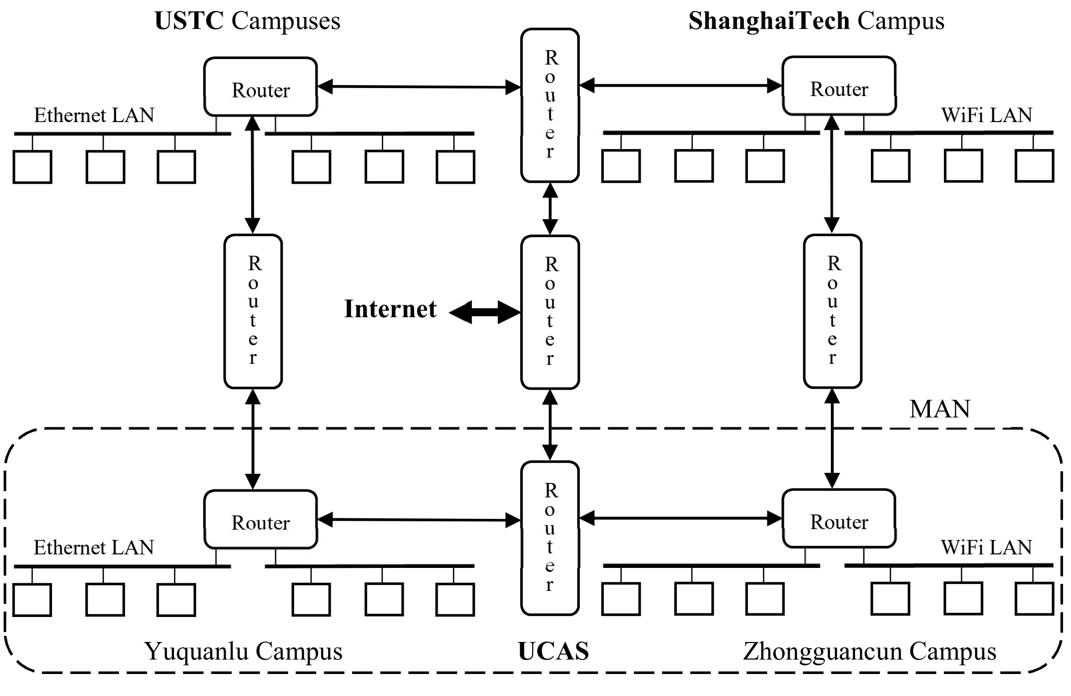


Fig. 6.1 Illustrating parts of networks of UCAS and sister universities

A **local area network** (LAN) connects devices (shown as squares in Fig. 6.1) in the same building or even on the same campus. A **metropolitan area network** (MAN) connects LANs in the same city. A **wide area network** (WAN) spans multiple cities or even multiple countries. These are demonstrated in Fig. 6.1.

The University of Chinese Academy of Sciences (UCAS), the University of Science and Technology of China (USTC) and the ShanghaiTech University (ShanghaiTech) have campuses in Beijing, Hefei and Shanghai, respectively. Their LANs and MANs are connected by a WAN called the China Science and Technology Network (CSTNet), which is connected to and forms a part of the global Internet.

An institution providing Internet connection services is called an Internet service provider (**ISP**). The institution running CSTNet, also called CSTNET, is the ISP for all students, faculties, and staffs of Chinese Academy of Sciences.

Suppose a student, Zhang Lei, is studying in a classroom on the Zhongguancun campus of UCAS in Beijing. She wants to use her laptop computer to access a server located in a machine room on the ShanghaiTech campus. Let us trace the connections and see the network terms involved.

- **Host**. Any device connected to a network is called a host (shown as a square in Fig. 6.1). It could be a laptop computer, a printer, a server, or an environment sensor. A host directly used by the user is called a **client**, which makes requests to be served by another host called **server**. Zhang's laptop computer is a client host, and the server on the ShanghaiTech campus is a server host. Note that hosts do NOT include **networking devices** discussed below, such as access point, hub, switch, router, and gateway devices.
- **Access Point** (AP). Zhang uses a wireless protocol called WiFi to access the Internet. Her laptop computer first needs to connect to an **access point** (AP). The AP device converts wireless signals to wired signals and vice versa. The AP device is part of the WiFi LAN and not explicitly shown in Fig. 6.1.
- **Hub** and **Switch**. A small LAN is often a bus structure realized by connecting a few hosts to a central device called a network *hub*. Multiple small LANs can be connected to form a bigger LAN to accommodate more hosts or to extend the distance covered. A network *switch* connects multiple hosts or LANs. The same protocol, e.g., Ethernet, must be used within a LAN, small or big. In such a *homogeneous* network, hosts are said to be *homogeneously connected*.
- **Router**. What if we want to connect several networks that use different protocols? Networking devices called **routers** are used to solve this *internetworking* problem, i.e., connecting multiple member networks into a bigger *heterogeneous* network, e.g., the Internet. Each member network of the Internet may internally use a different networking technology of its own choice, such as WiFi, Ethernet, or Infiniband. Figure 6.2 shows how multiple routers (the brown boxes) connect two heterogeneous member networks: the UCAS Beijing campus network using WiFi, and the ShanghaiTech campus network using Ethernet.

- A **gateway** of a member network is the router connecting the member network to a bigger network, e.g., Internet. In Fig. 6.2, the purple and green dots indicate gateways for the Beijing network and the Shanghai network, respectively.

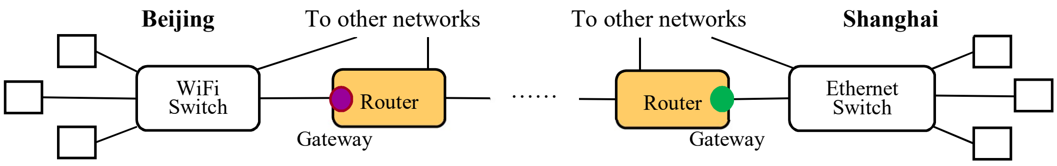


Fig. 6.2 Connecting two heterogeneous campus networks by routers

6.2. Connectivity

Connectivity refers to the structure of a network. Only when an entity is connected, i.e., having become a node of a network, can it be accessed. Connectivity studies problems related to *naming* and *topology*, such as the following:

- How to name the nodes of a network? How to find a specific node? How to refer to a specific node?
- How are the nodes interconnected? Does the network structure change over time? Is the network static, dynamic, or evolutionary?

It is customary to use a mathematical structure, called a **graph**, to represent a network. A graph $G =< V, E >$ is a pair of sets. Here, $V = \{v_1, v_2, \dots, v_n\}$ is the set of nodes. A node is also called a **vertex**. $E = \{e_1, e_2, \dots, e_m\}$ is the set of **edges** or *links*. Each edge connects two nodes. Figure 6.3 shows two networks. A directed edge shows a one-way connection. An undirected edge is a two-way connection. Note that node v_1 connects to itself, while v_3 does not connect to any node.

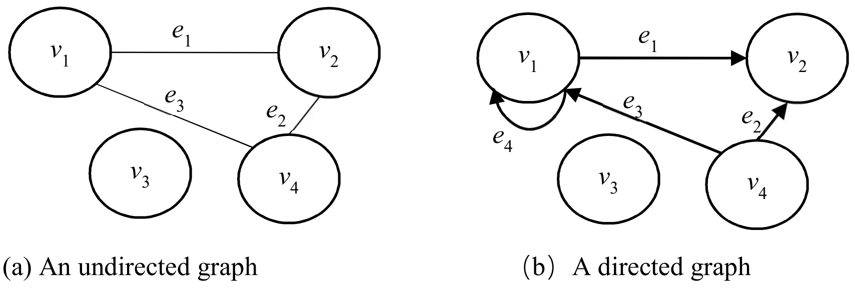


Fig. 6.3 Networks can be represented as directed and undirected graphs

6.2.1. Naming

We specify the nodes of a network by **naming**, i.e., giving a name to every node. As illustrated in Table 6.1, every network has a **namespace**, which consists of all names of the network specified by a naming scheme.

A name is usually a string of characters. Whether a string is legitimate is determined by a naming scheme, which is often specified by some technical **standard**. Three influential standards bodies are IEEE, IETF and W3C. They are all professional volunteer organizations. Ethernet names and protocols are designed by the Institute of Electrical and Electronics Engineers (IEEE). Internet standards (called **RFC**s) are designed by the Internet Engineering Task Force (IETF). Web names and protocols are designed by the World Wide Web Consortium (W3C).

Designing a namespace needs to consider the following issues:

- *Uniqueness*. Does a name map to a unique node? The email address namespace enjoys uniqueness, but the namespace of personal names of a country's population does not have uniqueness. There may be multiple persons named Joan Smith, causing *name conflicts*, which in turn may lead to wrong connections.
- *Autonomy*. Can a user create or change a name on his own? Such autonomy has the advantage of convenience, but may lead to chaos. One may change a URL, i.e., the name of a webpage. However, Web links to the old URL become invalid. For many naming schemes, creating or modifying a name need to go through a centralized process, involving an authority of name registry.
- *Friendliness*. Are the names user-friendly, i.e., understandable by humans? The eight name schemes in Table 6.1 have decreasing user friendliness. "Joan Smith" is much more understandable than "00-1E-C9-43-24-42", which is the name of the network interface circuitry in a computer, also called MAC address.
- *Name conversion*. An entity can have two namespaces. The Internet site with domain name www.ict.ac.cn has an Internet Protocol (IP) address 159.226.97.84. The Domain Name System (**DNS**) converts a domain name to its IP address.

Table 6.1 Examples of Naming Schemes

Namespace	**Instance**	**Remark**
Personal name	Joan Smith	Personal names in a country
WeChat user	ZhongguanVillager	Any legitimate string per WeChat standard
URL	www.ict.ac.cn/cs101	Universal Resource Locator of a webpage
Internet site	www.ict.ac.cn	Any domain name by IETF standards
Email address	zxu@ict.ac.cn	userName@domainName
IP address	159.226.97.84	Internet protocol address per IETF standards
Phone number	189-6666-8888	11 decimal digits by Telcom provider standards
MAC address	00-1E-C9-43-24-42	12 hexadecimal digits per IEEE standards

40. The Namespaces of the Internet: domain names vs. IP addresses

Recall that the **Internet** is a global network of networks, connecting many local networks and computers through a common set of protocols, particularly the Internet Protocol (IP).

Let's put ourselves in the shoes of early Internet designers and ask this question: How many names are needed to uniquely denote every node of the global Internet?

There are two types of nodes in the Internet:

- *Internet hosts* are computers connected to the Internet, such as our laptop PCs and smartphones. Here, computers included Internet-connected devices such as printers, TVs, and sensors, which can be viewed as embedded computers.
- *Networking devices* on the Internet, such as network routers and switches.

Every node of the Internet should be named. At a minimum, a node must have an address, called the Internet Protocol address, or IP address. See Box 11 for a comparison between names and addresses.

Two types of IP addresses are used today.

- **IPv4 addresses**. An Internet Protocol version 4 address (IPv4 address) occupies 32 bits, organized as a 4-field format xxx.xxx.xxx.xxx such as 159.226.97.84, where each field is normally shown as a decimal number from 0 to 255. This IPv4 address format can generate 2^{32}, or over 4 billion, different IP addresses.
- **IPv6 addresses**. An Internet Protocol version 6 address (IPv6 address) occupies 128 bits. This IPv6 address format can generate 2^{128} different IP addresses.

If we assign a unique IP address to a node of the Internet, the IPv4 addressing scheme can accommodate an Internet of over 4 billion nodes, including Internet hosts and networking devices. In reality, multiple IP addresses can map to one node (cf. RFC 791). Thus, the 32-bit IPv4 addressing scheme implies that the Internet can have fewer than 4 billion nodes, and even fewer host computers. Today, we already have more than 4 billion hosts on the Internet.

Longer address formats could have been used to accommodate more nodes. But, longer address formats also consume more bit resources. To send a message from node A to node B, the message needs to include the source address for A and the destination address for B, besides the payload data of the message. For a short message of 4-byte payload data with IPv4 addresses, we need to send at least 96 bits, of which only 1/3 is payload, and 2/3 is IP address metadata information.

The Internet pioneers did a tradeoff in 1981 (RFC 791) by choosing a 32-bit format for the IPv4 addressing scheme. Note that an IP address assignment to a node is not permanent, and may be recycled. Still, in 2019, all IPv4 addresses worldwide were assigned, leaving no free IPv4 addresses to allocate.

Fortunately, the IETF standards body created a much longer IPv6 address format in 1995 (RFC 1883). If each node of the Internet is assigned a unique IP address, the IPv6 address format can accommodate $2^{128} = 2^{32}\times2^{96}$ nodes, or 2^{96} times the capacity of the IPv4 scheme.

Box 11. Addresses vs. Names

A **name** is a string of characters used to denote an entity. **Addresses** are a special form of names. An address can be directly used to access an entity, while a non-address name needs to be first converted into an address before accessing the entity denoted by the name.

For instance, *www.ict.ac.cn* is a name: the **domain name** of the website for the Institute of Computing Technology, Chinese Academy of Sciences. To access this website from a Web browser, what really happens is that this domain name is converted by DNS into the corresponding **IP address** 159.226.97.84. This IP address is then used to access the website. It is difficult to remember an IP address such as 159.226.97.84. Users prefer to use the more user-friendly domain name www.ict.ac.cn.

A computer on the Internet can serve as a *client host* such as a user's PC, a *server host* such as a supercomputer, or a *networking device* such as a router. A computer on the Internet can have one or more IP addresses. Multiple domain names can be mapped to the same IP address. Conversely, the same domain name can be served by multiple computers. A popular website, sometimes called a *hub* node of the Internet, may need dozens of computers to serve requests during peak hours.

Domain names are organized in hierarchies. The structure of the hierarchies can be seen when reading a domain name from right to left. In www.ict.ac.cn, "cn" is the top-level domain by *country*, "ac" is the second-level domain, and "www" is at the lowest level. There are also *generic* top-level domains, such as "com" and "org" in github.com and w3.org. There are over 1000 top-level domain names today. A student named Fan Wang (范望) may register an Internet domain name as fan.wang.

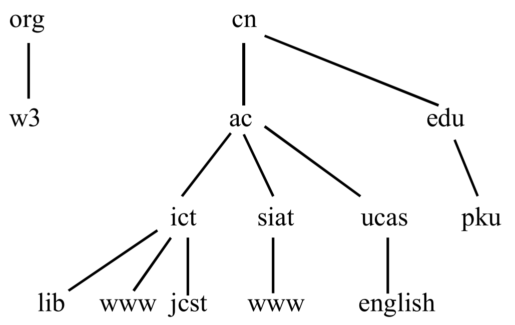


Fig. 6.4 Parts of the global domain name hierarchies

41. The Namespace of the World Wide Web: URL

A *uniform resource locator* (**URL**) is a string that specifies the location of a Web resource for a browser to easily access the resource. A **web resource** can be a website, a webpage, a picture file on the Web, etc. Sometimes, people calls a URL a **web address**. As illustrated in Fig. 6.5, a basic URL consists of three parts, separated by "://" and "/". There can be infinitely many web resources, thus infinitely many URLs.

The first part specifies the *scheme*, which indicates a protocol. The most popular protocol used on the Web is the *hypertext transport protocol* (**HTTP**), and its secure version *hypertext transport protocol secure* (**HTTPS**). This book contains examples of both protocols. Other protocol names include FTP for transferring a file from one computer to another computer, and MAILTO for email.

The second part specifies the *host*, which indicates the name of an Internet host computer. The host can be specified by its domain name or its IP address. When we enter in the browser http://www.ict.ac.cn or http://159.226.97.84, we get the same result, indicating the same host computer.

The third part specifies the *path*, which indicates where the resource is located on the host, by specifying a file path name. Note that the HTTP root directory may be different from the host computer's operating system root directory. In Fig. 6.5, the file path name is /cs101, e.g., indicating that the resource is located at cs101 in the HTTP root directory of the host.

Two special URLs are noteworthy. First, when we enter in the browser a URL of a website without a path (e.g., http://www.ict.ac.cn) or with only the HTTP root (e.g., http://www.ict.ac.cn/), we are accessing the **homepage** of the website.

Second, when we enter in the browser the URL http://localhost, or equivalently http://127.0.0.1/, this special host is reserved for the local computer (or local host), where the browser is executing. The string "localhost" indicates the **loopback domain name** of a computer, and 127.0.0.1 is its **loopback IP address**. Project 4 of this book will use local host extensively to develop and debug webpages.

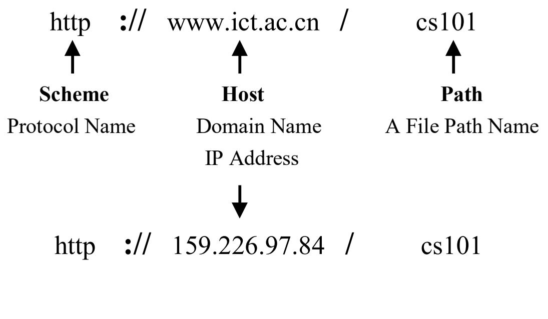


Fig. 6.5 Anatomy of a uniform resource locator (URL)

6.2.2. Network Topology

At any moment, the structure of a network is a graph, showing what nodes and edges are in the network, and how the nodes are interconnected by the edges. This graph is also called the **topology** of the network. The topology of a network may change with time. Depending on how the network topology changes, we can classify networks into three classes, as illustrated in Fig. 6.6.

- **Static networks**. Static networks do not change their nodes and edges. More precisely, a network $G =< V, E >$ is a static network, if the node set V and the edge set E do not change over time. In Fig. 6.6, the fully connected graph and the star network are both static networks.
- **Dynamic networks**. A dynamic network does not change its nodes, but may change its edges. In Fig. 6.6, the **bus** and the **crossbar** switch networks are both dynamic networks. At one moment, the bus may actually connect the processor (P) and the memory (M). At the next moment, the bus may actually connect the memory (M) and the input device (I). The bus supports a *shared-media network*, while the crossbar supports a *switching network*.
- **Evolutionary networks**. Evolutionary networks change both nodes and edges over time. An example is the network of all webpages on the World Wide Web. The node set V (the set of all webpages) and the edge set E (the set of Web links) both constantly change over time.

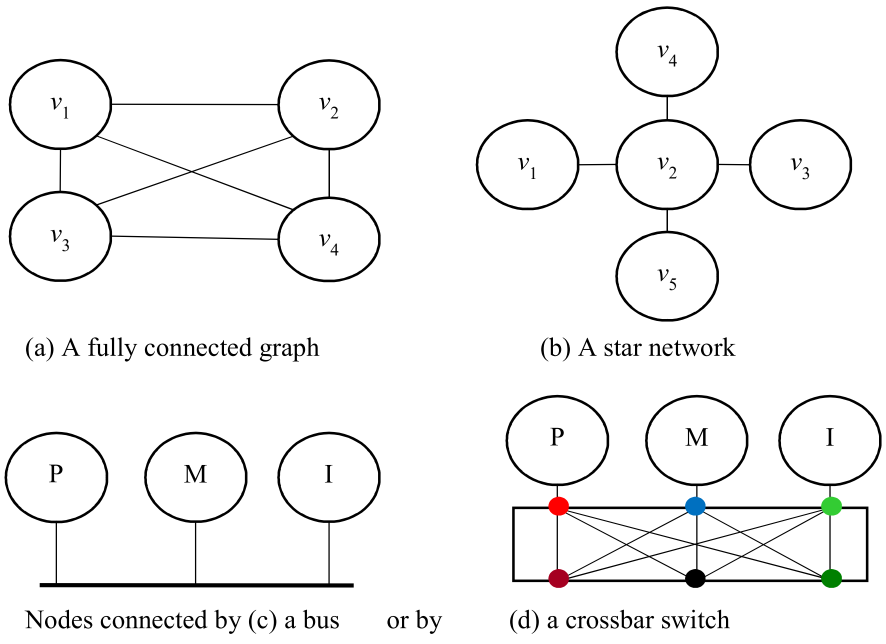


Fig. 6.6 Examples of static and dynamic networks

In Fig. 6.6d, three devices (circles) interconnect to one another through a crossbar switch (rectangle). The three devices are the processor (P), the memory (M) and the input device (I). Each device is connected to both an input port and an output port of the crossbar switch. For instance, the processor (P) is connected to the red input port and the purple output port in Fig. 6.6d. This is redrawn in Fig. 6.7, where both P nodes denote the same processor. A device may be able to send messages and receive messages at the same time. In practice, an input port and an output port can be implemented as one port. Such a port capable of simultaneous bidirectional communication is said to work in the **full-duplex** mode. That is, full-duplexing allows data to flow into and out of a port at the same time.

The crossbar switch is actually a fully connected graph, as shown in Fig. 6.7. At any moment, all or parts of the edges are active. A crossbar switch is more powerful than a bus. It enables all devices to connect to one another at the same time. A bus can only connect two devices at a time, except for **broadcasting**, where a device sends a message to all devices on the bus.

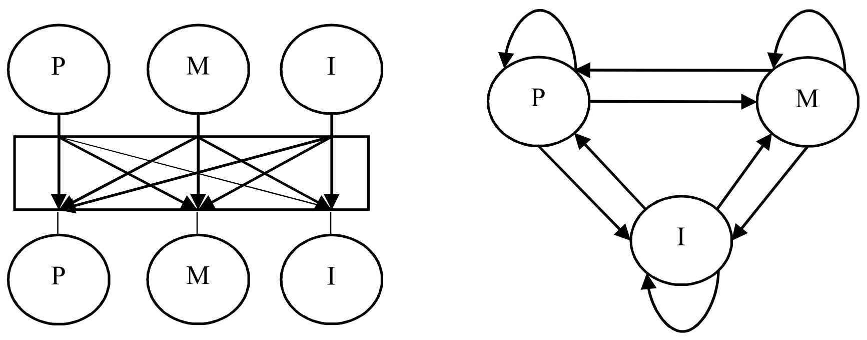


Fig. 6.7 Expanded illustrations of a network of three devices interconnected by a crossbar switch.

Example 61. How network thinking helped create modern search engines

Network thinking offers a new perspective to look at problems and can lead to innovative solutions. A case in point is search engine design. Early search engines computed search results by matching the key-words in search queries to the contents of webpages (*nodes*). These first-generation search engines only utilized nodes of the network of webpages.

Around 1996, Jon Kleinberg, Robin Li (李彦宏), and Larry Page, independently observed a phenomenon: Web links (*edges*) also significantly influence the relevance of search results. They utilized both nodes and edges to develop the second-generation search engines with much better results. This approach more fully utilizes network thinking and created Google and Baidu companies, serving billions of users and generating annual revenue over $100 billion.

☷

6.3. Protocol Stack

A **protocol** is a set of rules enabling two nodes of a network to communicate with each other, in order to automatically collaborate in accomplishing a common computational task. A **protocol stack** is a stack of layers of protocols which work together. This section introduces the Web over Internet protocol stack, to elaborate the basic concepts of communication in a computer network.

6.3.1. The Web over Internet Stack

A basic innovation of the Internet is to use *packet switching* to replace the *circuit switching* technology used in traditional telecommunication networks.

- **Circuit switching**. When two parties are having a telephone conversation lasting 10 minutes over a telecommunication network, a physical circuit is established and dedicated to this task during the entire time period of conversation. No other person can use the resource of the circuit for this period of 10 minutes.
- **Packet switching**. When two parties are having a telephone conversation lasting 10 minutes over the Internet, a person's utterances are viewed as digital **messages** communicated to the other person. Each message is divided into a number of small **packets**, which are sent together with packets from other people or devices, over the same communication channel (circuit). In other word, the same channel can accommodate multiple telephone conversations in this period of 10 minutes.

Example 62. Comparison of circuit switching and packet switching

Three students each want to download a file from the Internet over a shared communication channel, as shown in Fig. 6.8. Suppose the communication channel has a bandwidth of 10 Mbps (million bits per second) and all three students start downloading at time 0. How much time does each of these downloading tasks take?

Figure 6.8 shows the Internet as a cloud picture, where three files are stored. The circuit switching approach works as shown in Fig. 6.8a. Suppose student Smith's downloading request is first received by the system. A dedicated, end-to-end communication circuit is established between the file and Smith's PC. Then the system transmits file Autumn.bmp from its source server to the destination computer (Smith's PC). Let us simplify matters by further assume that all overheads are ignored. Then, because each byte has 8 bits, the first downloading task takes

$T_1 = 9.14 \times 8 \,/\, 10 = 7.31$ seconds.

In other words, the task to download Autumn.bmp finishes at $T_1 = 7.31$ second.

Similarly, the second task to download hamlet.txt only needs

$T_2 = 0.182 \times 8 \,/\, 10 = 0.146$ seconds.

However, it starts at 0 second and finishes at $T_1 + T_2 = 7.46$ second.

The third task to download ucas.bmp needs

$T_3 = 0.81 \times 8 \,/\, 10 = 0.648$ seconds.

The third task starts at 0 second and finishes at $T_1 + T_2 + T_3 = 8.11$ second.

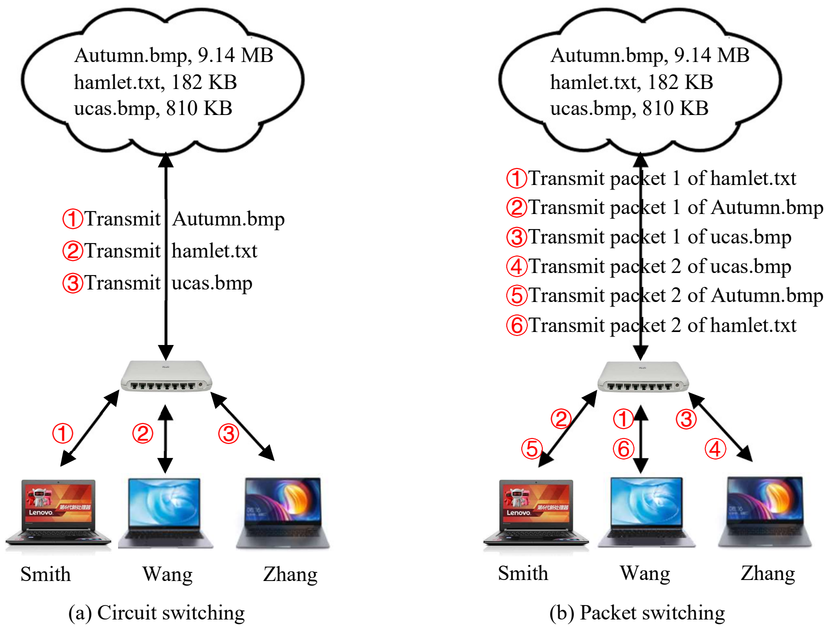


Fig. 6.8 Explaining circuit switching versus packet switching

With circuit switching, a dedicated circuit is established between the two parties of a communication session. Only at the end of the session is another dedicated circuit established (the circuit is "switched") to serve another session.

Circuit switching has the advantage of guaranteeing the quality of service of a communication session, which is not interfered by other communication sessions. This is especially important for real-time services such as telephone voice communication, where interference may cause voice sound quality degradation. Circuit switching also has disadvantages. Dedicated use of a channel implies that the channel capacity is not necessarily fully utilized. Furthermore, a communication session forces other sessions to wait till it finishes.

With packet switching, each of the three files (messages) is divided into a number of packets (e.g., each of 1 KB), and the channel statistically mixes and transmits the packets of all three messages, as illustrated in Fig. 6.8b. It is left as an exercise to show that the three downloading tasks starting at time 0 finish at $T_1 = 8.11$ second, $T_2 = 0.44$ second, and $T_3 = 1.44$ second, respectively. The three communication tasks proceed simultaneously, without waiting for one another to finish.

☶

The following table shows the format of an Ethernet packet. Each packet has a **body** and a **header**. The body (in bold) is the payload data of the packet. The header, parts of which may come after the payload data, includes control information, addresses, and Cyclic Redundancy Check (CRC) error check information.

Table 6.2 The Ethernet packet format, where 42-1500 types are used for the payload

7	1	6	6	2	**42-1500**	4
Preamble	Frame Delimiter	Destination MAC Address	Source MAC Address	Type	**Data (Payload)**	CRC

Another basic innovation of the Internet is to use multiple layers of protocols, stacked on top of one another, to form the widely used **TCP/IP protocol stack**. Each layer serves a particular purpose of networking by transferring different types of packets in different ways. Together, the stack of multiple layers of protocols effectively supports the global Internet we have today.

This idea was extended later by a Web layer on top of the Internet. This HTTP protocol enables accessing of hyperlinked hypermedia resources, such as webpages. Table 6.3 shows parts of the Web over the Internet protocol stack. Let us elaborate this five-layer stack from bottom up.

- Layer 1 is called the **physical layer**. It provides physical communication channels by wired electrical cables or optical fibers, as well as wireless waveforms. Layer 1 works on physical signals of individual bits of 0’s and 1’s.
- Layer 2 is called the **data link layer**. It reliably sends a layer-2 packet, called a *frame*, between two homogeneously connected devices, including hosts and networking devices. Two widely used protocols are Ethernet (as specified by the IEEE 802.3 standard) and WiFi (as specified by the IEEE 802.11 standard).
- Layer 3 is called the **network layer**, which has only a single protocol: IP (the Internet Protocol). It sends a layer-3 packet, called a *datagram*, between two Internet hosts, without guaranteeing reliable packet delivery. This is called the *best-effort* approach. The two Internet hosts are not necessarily homogeneously connected, but may be connected through one or more routers.
- Layer 4 is called the **transport layer**. A widely used protocol at this layer is TCP (the Transmission Control Protocol). It reliably sends a layer-4 packet, called a *TCP packet*, between two Internet hosts.
- Layer 5 is called the **application layer**. There may be several application layers stacked on top of one another. We only consider the Web application layer using HTTP (the Hypertext Transfer Protocol). Its purpose is to easily access hyperlinked hypertext resources, such as webpages, across the Internet.

Table 6.3 Parts of the Web over Internet protocol stack

Layer	Protocol	Purpose
Application Layer Layer 5	HTTP	Access hypertext resources on a Web server from a Web client
Transport Layer Layer 4	TCP	Reliably transfer packets between two Internet hosts
Network Layer Layer 3	IP	Transfer packets between two Internet hosts in the best-effort way
Data Link Layer Layer 2	Ethernet, WiFi	Reliably transfer packets between two homogeneously connected devices
Physical Layer Layer 1	Wired or wireless, electrical or optical, cables or waveforms	Provide physical communication channels Transfer signals of individual bits

42. How does the protocol stack work? Fetch a home page from a server

Let us return to the example of Zhang Lei's using her laptop computer in Beijing to access a server in Shanghai, via the Web over Internet protocol stack.

In her Web browser, Zhang enters the following information to access the server:

http://www.shanghaitech.edu.cn/

which has three fields. HTTP is the protocol name. This protocol is used to transfer Web requests and responses. The domain name www.shanghaitech.edu.cn identifies the website of ShanghaiTech, which is translated by DNS to the IP address 119.78.254.67 of the website server. The third part is a slash, indicating the **home page**, i.e., the introductory webpage of the website.

Upon receiving this HTTP request, the Web server at 119.78.254.67 sends back an HTTP response message, i.e., the contents of the home page. The message is divided into a number of HTTP packets.

In Fig. 6.9, an HTTP packet is indicated as a pink box, which is handed over to the TCP layer as the TCP packet body. The TCP layer adds a TCP header (shown as a blue box) to form a TCP packet. The TCP packet is handed over to the IP layer as the IP packet body. The IP layer adds an IP header (shown as a yellow box) to form an IP packet. Finally, the IP packet is handed over to the data link (Ethernet) layer as the packet body. The Ethernet layer adds an Ethernet header (shown as a red box) to form an Ethernet packet. We ignore the physical layer in this book.

Note that all these operations are automatically done in the server host. When a data link packet arrives at the destination host, i.e., Zhang's laptop computer, a reverse process takes place to recover the HTTP packet data.

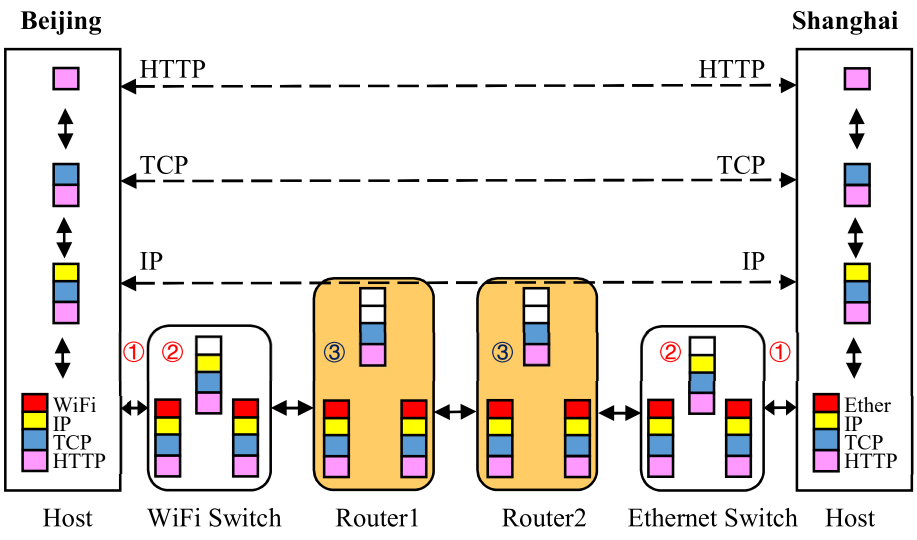


Fig. 6.9 Illustrating how a packet traverses the Web over Internet stack.

Let us ask and answer several questions to better understand the protocol stack approach with packet switching communication.

- Does Zhang need to worry about TCP/IP and Ethernet when surfing the Web?

The answer is NO. This is the beauty of the protocol stack approach.

A *user* at one layer does not have to worry about the layers below. The protocol stack provides two types of interfaces. When Zhang's Web browser in Beijing communicates with the Web server in Shanghai, both parties use a **peering interface** via the HTTP protocol, shown by horizontal dashed lines in Fig. 6.9. Here, *peering* means that the Web browser in Beijing acts as if it directly talks to its peer, the Web server in Shanghai, using a common HTTP protocol.

A software *developer*, however, implements the HTTP protocol using a **service interface** provided by the TCP layer, shown by vertical solid lines in Fig. 6.9. The TCP layer provides services to the HTTP layer via the service interface.

- Can the Web server in Shanghai send an HTTP packet to Zhang's Web browser in Beijing, without also sending a data link layer packet, e.g., an Ethernet frame?

The answer is NO.

One cannot send a high layer packet without also sending a packet of every layer below. When a packet enters a network, it is in a data link layer format and travels as wired and wireless signals.

An HTTP packet must be repackaged as a TCP packet, which is then repackaged as an IP packet, which is then repackaged as an Ethernet packet, before it can be sent to the network as physical signals. This is similar to how post offices send letters. Letters are wrapped in envelopes, adding header information such as addresses, whether the letters should be express mailed, etc. The letters in envelopes are then wrapped in post office's packages, and so on.

- Can the server computer in Shanghai send an Ethernet packet to Zhang's personal computer in Beijing?

The answer is YES, as long as there is a homogeneous network connecting the two hosts via the same Ethernet protocol.

But the reality is like Fig. 6.9, to accommodate heterogeneity of the Internet. The server host sends an HTTP packet, wrapped as an Ethernet packet, to the Ethernet switch (①). The switch opens the packet to reveal the Ethernet header (shown as a white box in Fig. 6.9), and then adds a new header (with a new MAC address) to form a new Ethernet packet (②). When the packet arrives at Router2, the router opens the packet to reveal both the Ethernet and the IP headers (the two white boxes), and then form a new Ethernet packet by reformatting the packet and adding a new Ethernet header (③). Similar steps take place at Router1 (③) and the WiFi Switch (②), and then a WiFi packet arrives at the laptop computer host (①). Note that when Router1 (③) reformats the packet, it converts an Ethernet packet into a WiFi packet by adding a WiFi header.

- What is actually sent over a network?

Bit string of 0's and 1's.

Any packet is eventually encapsulated as one or more physical layer packets, which travel as wired or wireless signals. A physical layer packet is sent through electrical cables, electromagnetic waveforms or optical fibers, in a bit string of 0's and 1's. A 0 may be represented as a LOW voltage pulse or a LIGHTOFF state, while a 1 may be represented as a HIGH voltage pulse or a LIGHTON state.

- When a message is sent from a source host computer A to a destination host computer B, do all packets of the message travel through the same physical path from host A to host B?

Not necessarily.

The global Internet has **redundancy** built in, to cope with traffic congestions and faults. This redundancy is a reason why we see many fault events reported every day worldwide, affecting parts of the Internet, but the global Internet as a whole has never gone down.

An example is shown in Fig. 6.10, where A, B, C, and D denote hosts, and the remaining nodes denote networking devices (switches and routers). There are multiple physical paths from host A to host B. When host A sends a 99-packet message to host B, it may well be the case that the first packet of the message travels along the physical path A-T-X-Y-W-B, the 49th packet traverses path A-T-U-V-W-B, the 99th packet traverses path A-T-X-Z-Y-W-B, etc.

It may happen that the 49th packet arrives at host B before the first packet. Each packet comes with a packet number. The complete message is reassembled from the packets by their numbers, when all packets arrive at hots B.

As long as there is a valid path from A to B, a message can be communicated, even though other nodes or edges may be broken. For instance, when node T is down, host A cannot communicate. But, the remaining hosts B, C and D can still communicate with one another.

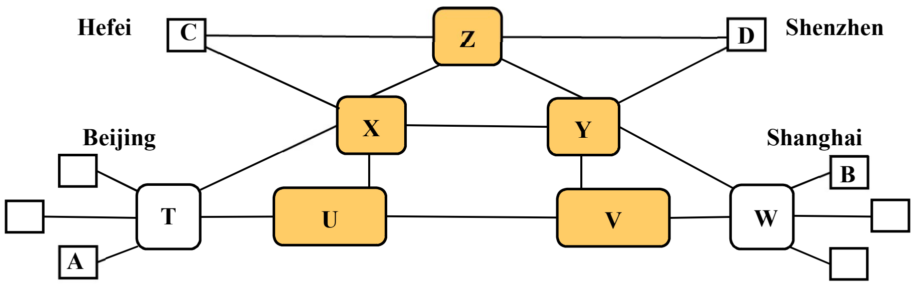


Fig. 6.10 Illustrating redundancy of the Internet: there are multiple paths from host A to host B

6.3.2. Elementary Web Programming

We introduce several Web programming examples to deepen the understanding of protocol stack. They also prepare students for the Personal Artifact project, which asks a student to create a **dynamic webpage**. Here, *dynamic* means that the contents or format of a webpage may change when displayed in a browser. In contrast, a static webpage does not change its contents or format.

Only the basic HTML/CSS/JavaScript knowledge is introduced. Students are suggested to transfer the Go programming knowledge learnt to Web programming. They are also encouraged to learn any additional knowledge needed, by referencing the library of Personal Artifacts examples in the Supplementary Material.

43. Create a Web server and static webpages

To create a Web server, add the following WebServer.go file in the working directory. The code uses the net/http package to create a Web server.

```
> cat WebServer.go
package main
import "net/http"
func main() {
  http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
    http.ServeFile(w, r, r.URL.Path[1:])
  })
  http.ListenAndServe(":8080", nil)
}
>
```

Then, run the program WebServer.go in background.

```
> go run WebServer.go &
[1] 72
>
```

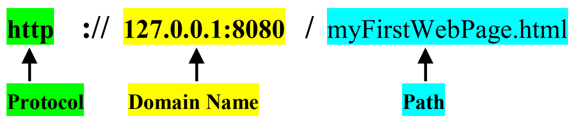
The '&' symbol at the end of a command indicates that the command runs in the background: Although control returns back to the shell, the WebServer program is still executing (with a process ID 72), and ready to accept Web requests.

As a second step, add the myFirstWebPage.html file to the working directory. This is probably the student's first webpage. The myFirstWebPage.html static webpage is explained in Example 63.

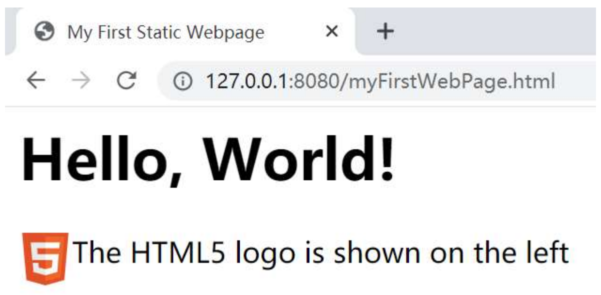
As a third step, open a Web browser and enter the following URL

http://127.0.0.1:8080/myFirstWebPage.html

Recall that a Web URL has three parts separated by "://" and "/". The domain name in this case is the loopback IP address of the student's laptop computer (the local host). We can also use the equivalent URL http://localhost:8080/myFirst-WebPage.html. Port number 8080 is a default port for HTTP.



The above browser command will display a webpage shown in Fig. 6.11.



Hello, World!

The HTML5 logo is shown on the left

Fig. 6.11 Brower display of myFirstWebPage.html.

Example 63. Look inside my first webpage: myFirstWebPage.html

A webpage, as shown in Fig. 6.12, is expressed as Hypertext Markup Language (**HTML**) code, which is enclosed by a <html> … </html> pair and consists of a **head** and a **body**. The most recent version of HTML is HTML5.

In the head part, <meta charset="utf-8"> specifies the character set as UTF-8, and the <title>…</title> pair specifies the title of the myFirstWebPage.html document as "My First Static Webpage". The title does not show up in the webpage itself when the HTML document is displayed.

```
<html>
  <head>
    <meta charset="utf-8">
    <title>My First Static Webpage</title>
  </head>
  <body>
    <h1> Hello, World! </h1>
    <p>
<img src="https://www.w3.org/html/logo/downloads/HTML5_Badge.svg"
style="float:left;width:30px;height:30px;">
The HTML5 logo is shown on the left
    </p>
    <script>
    </script>
  </body>
</html>
```

Fig. 6.12 A student's first webpage: myFirstWebPage.html

There are three types of code in the body part. It pays for the lecturer to encourage students to play with the first webpage by changing different elements and items and then redisplay, to quickly appreciate the HTML and CSS code.

- Pure HTML code. One example is the <h1> … </h1> pair which displays the enclosed content "Hello, World!" as level-1 heading. Another example is a *paragraph* enclosed in a <p> … </p> pair. There is only one paragraph in the webpage myFirstWebPage.html. It contains two items: an HTML5 logo image specified by a <img …> pair, and a text phrase "The HTML5 logo is shown on the left". The src="…" item specifies the source URL (hyperlink) of the image. The HTML5 logo image is retrieved from the specified source URL (hyperlink)

https://www.w3.org/html/logo/downloads/HTML5_Badge.svg. The protocol is "https", the domain name is "www.w3.org", and the local path name for the image file is "html/logo/downloads/HTML5_Badge.svg".
- Cascading style sheets (**CSS**) code specifying the format (presentation style) of the webpage. For instance, the item style="float:left;width:30px;height:30px;" in the paragraph says that the HTML5 logo image is displayed with the stated style: floating to the left of the text phrase, 30-pixel wide, and 30-pixel high.
- JavaScript code enclosed between a pair of <script> and </script>. In Fig. 6.12, this JavaScript part happens to be empty.

A Web server hosts the webpage files (called *resources*). The Web browser retrieves a Web resource, hyperlinked by a URL such as myFirstWebPage.html, to a student's laptop computer. The browser interprets it and displays the result. This is also called *rendering* the webpage. In this process, the browser automatically retrieves additional resources such as the HTML5 logo image by the hyperlink https://www.w3.org/html/logo/downloads/HTML5_Badge.svg.

☶

Example 64. A static webpage displaying the date of next Children's Day

We start to use JavaScript code in staticChildrensDay.html, as shown in Fig. 6.13. JavaScript is an *object-oriented* language. An **object** is a data structure with one or more components, each of which can be specified by the dot notation. An object may also contain a number of *methods* that operate on the data structure. Each method call is also specified by the dot notation. For instance, in the statement

document.getElementById("childrensDay");

document is an object, getElementById is a method, "childrensDay" is a parameter denoting the paragraph’s ID. The statement says to get the element by ID in the HTML document, where ID is specified by the string "childrensDay".

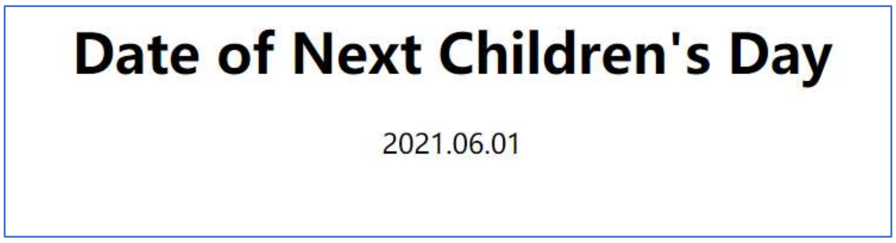
The example code contains a single *paragraph* element. Its behavior is not directly specified in the <p>…</p> pair, but rather via the id "childrensDay". The <p>…</p> pair only specifies the location of the paragraph element on the webpage, but does not specify the content of the paragraph. Four lines of JavaScript code are included in the pair <script> … </script>. They specify the style and the content of the paragraph element.

- var x = document.getElementById("childrensDay"); declares an *object* x and initializes it with the paragraph element denoted by ID childrensDay. Hereafter, x is the paragraph with id "childrensDay".
- x.style.fontSize = "60px"; sets the font size of paragraph x to be 60 pixels.
- x.style.color = "purple"; sets the color of paragraph x to be purple.
- x.innerHTML = "2021.06.01"; sets the content of paragraph x to be "2021.06.01".

```
<html>
   <head>
      <meta charset="utf-8">
      <title>Display the date of next Children's Day</title>
   </head>
   <body>
      <h1 style="text-align: center">Date of Next Children's Day</h1>
      <p style="text-align: center" id="childrensDay" ></p>
      <script>
         var x = document.getElementById("childrensDay");
         x.style.fontSize = "60px";
         x.style.color = "purple";
         x.innerHTML = "2021.06.01";
      </script>
   </body>
</html>
```

Element ID
Element Style
Element Content

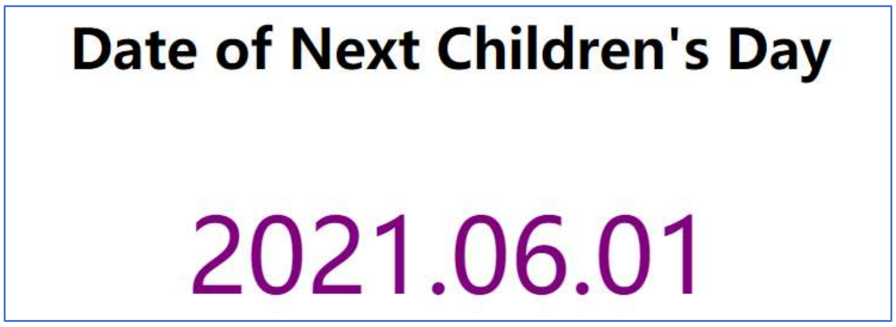
(a) Sample code for displaying the date of next Children's Day



Date of Next Children's Day

2021.06.01

(b) Output when the two lines of red code are deleted



Date of Next Children's Day

2021.06.01

(c) Output when the two lines of red code are present

Fig. 6.13 A static webpage staticChildrensDay.html: code and output

44. Create a dynamic webpage

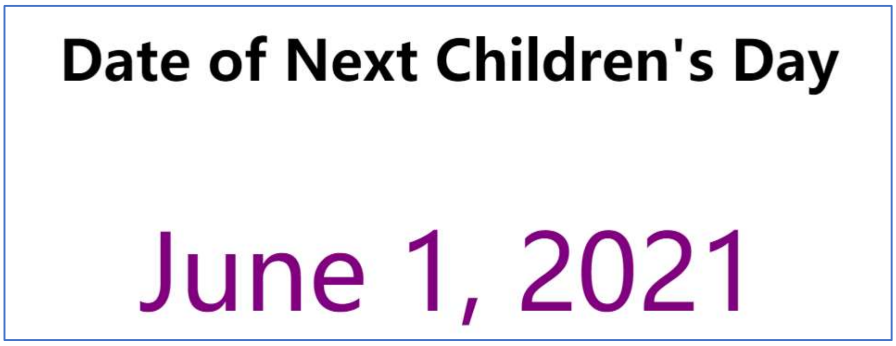
The previous example staticChildrensDay.html generates wrong outputs after June 1, 2021. To fix this error, we use the following JavaScript code to dynamically check and output the correct date, as shown in Fig.6.14.

The program's logic is simple: (1) get the current values of year and month using the system provided Date object; (2) if the month is June or later, increments year.

We add 1 to date.getMonth() because JavaScript counts months from 0 to 11.

```
<html>
  …
  <body>
    <h1 style="text-align: center">Date of Next Children's Day</h1>
    <p style="text-align: center" id="childrensDay" ></p>
    <script>
      var x = document.getElementById("childrensDay");
      x.style.fontSize = "60px";
      x.style.color = "purple";                              ID = Element Name
      var date = new Date;                                   Style
      var year = date.getFullYear();                         Content
      var month = date.getMonth() + 1;
      if (month >= 6) year = year + 1;
      x.innerHTML = "June 1, " + year;
    </script>
  </body>
</html>
```

(a) Code to display the dynamic date of next Children's Day



Date of Next Children's Day

June 1, 2021

(b) Browser output of ChildrensDay.html

Fig. 6.14 A dynamic webpage ChildrensDay.html: code and output

6.4. Network Laws and Responsible Computing

Computer networks, especially the Internet, have shown great impact on human society, including our work, life, and culture. During the past decades, notable phenomena and principles were observed, which continue to evolve and impact society. We discuss several such phenomena and principles, as well as practical guides to responsible computing.

6.4.1. Bandwidth, Latency, and User Experience

We want a computer network to have high bandwidth and low latency. **Bandwidth** is the bit rate by which messages are transmitted over a network, while **latency** is the time taken to transmit a message.

Exact meanings of bandwidth and latency vary. Two types are often used.

- *Minimal latency and maximal bandwidth*. The total transmission time *t* of a message of length m bits transmitted between two parties can be estimated by the following **Hockney's formula**, $t = t_0 + m/r_\infty$. Here, t_0 is the minimal **latency**, i.e., the time needed to transmit a 0-bit message; and r_∞ is the maximal **bandwidth** achieved when the message length m approaches infinity.
- *User-experienced latency and bandwidth*. For a given message of length m, define the user-experienced latency as the total transmission time t and user-experienced bandwidth as the ratio m/t.

Recall *Keck's law* when we discuss the Wonder of Exponentiation in Section 1.3.2. Technology advances have enabled the exponential growth of the data transmission rate of optical fibers. The maximal bandwidth achieved in the so called hero experiments, roughly equal to r_∞, increased about 100 times every decade, as shown in Table 6.4. Note that t_0 and r_∞ represent extreme values. Latency t_0 here means the startup overhead of a message transmission, i.e., the minimal latency. Bandwidth r_∞ is the maximal bandwidth.

Table 6.4 also shows the total transmission time t of a message of length m bits, assuming different values of t_0 and r_∞.

Table 6.4 Bandwidth growth of a single optical fiber in hero experiments
We thank Mr. Jeff Hecht of IEEE to provide Donald Keck's original data.

Time of Experiment	**Maximal Bandwidth Achieved r_∞**	**Time t to Transmit 1 GB**		**Time t to Transmit 1 KB**	
		$t_0 = 1\ \mu s$	$t_0 = 1\ ms$	$t_0 = 1\ \mu s$	$t_0 = 1\ ms$
1975	4.50E+07 bps, or 45 Mbps	178 s	178 s	0.2 ms	1.2 ms
1984	1.00E+09 bps, or 1 Gbps	8 s	8 s	9 μs	1 ms
1993	1.53E+11 bps, or 153 Gbps	52 ms	53 ms	1.1 μs	1 ms
2002	1.00E+13 bps, or 10 Tbps	0.8 ms	1.8 ms	1 μs	1 ms
2013	8.18E+14 bps, or 818 Tbps	11 μs	1 ms	1 μs	1 ms

45. Bandwidth and latency: extreme and user-experienced values

Let us look at Table 6.4 again. Assume a message of length m = 1 GB = 8 Gb is transmitted on a network with startup latency $t_0 = 1$ ms and a maximal bandwidth $r_\infty = 10$ Tbps. The total transmission time is

$$t = t_0 + m/\, r_\infty = 1 \text{ ms} + 8 \text{ Gb} \,/\, 10 \text{ Tbps} = 0.0018 \text{ s} \ = \ 1.8 \text{ ms} \,.$$

In contrast, the user-experienced latency is t = 1.8 ms, and the user-experienced bandwidth is m / t = 8 Gb / 1.8 ms = 4.44 Tbps.

Let us look at Table 6.5 below, which shows a network closer to the reality experienced by an ordinary home user with a 1-Gbps optic fiber subscription. For short messages of m=1 KB and a high startup latency $t_0 = 1$ second, the user actually gets only a bandwidth of 8 Kbps.

Table 6.5 Examples of user-experienced bandwidth and latency given $r_\infty = 1$Gbps

Minimal Latency t_0	**m = 1 GB**		**m = 1 MB**		**m = 1 KB**	
	t	**m/t**	**t**	**m/t**	**t**	**m/t**
0.000001 s	8 s	1 Gbps	8 ms	1 Gbps	9 μs	0.89 Gbps
0.001 s	8 s	1 Gbps	9 ms	0.89 Gbps	1 ms	7.9 Mbps
0.1 s	8.1 s	0.99 Gbps	0.108 s	74 Mbps	0.1 s	80 Kbps
1 s	9 s	0.89 Gbps	1.008 s	7.9 Mbps	1 s	8 Kbps

46. Data compression: lossless and lossy compressions

Data compression is the technique used to reduce the size of a file, such that a file can take less storage space and transmission time. **Lossless compression** can reduce the size of a file without losing information. **Lossy compression** could lose information when reducing the size of a file. With lossy compression, the original file cannot be recovered from a compressed file.

The Linux operating system comes with a lossless compression command called gzip. Try this compression command

```
> gzip Autumn.bmp
```

and compare Autumn.bmp to the compressed file Autumn.bmp.gz, to see how much the file size is reduced. The command gzip is lossless, in that no information is lost. We can execute a decompress command:

```
> gzip -d Autumn.bmp.gz
```

to recover the original file Autumn.bmp.

In general, when compressing a program file or a scientific data file, we should use lossless compression. This is due to the fact that a single faulty bit could invalidate the entire file. On the other hand, lossy compression can be used with multimedia such as image, video, and sound files. We can reduce file size while losing some resolution (quality) of the multimedia file.

6.4.2. Network Effect

Since the invention of ARPANET in the 1960s, the global Internet has grown significantly in its scale and impact to society. Some facts and projections are summarized in Table 6.6. The impact of a network can be beneficial, harmful, or both. When the impact is beneficial, we often call it the network value.

Table 6.6 The Internet's historic scale growth trend and exemplar techniques

Time	# Nodes	Exemplar Techniques
1960s	A few	Packet Switching Network
1970s	Thousands	TCP/IP, Ethernet, Router
1980s	100 thousand	Client-Server Computing
1990s	Million	World Wide Web
2000s	100 Million	Cloud Computing
2010s	Billions	Smartphones, Mobile Internet
2020-2050	Trillions	Internet of Human-Cyber-Physical Systems

Network effect refers to the phenomenon that a node derives value from the network, not from itself. An isolated node, e.g., a standalone laptop computer, has no impact to and receives no impact from the network. But when the laptop computer joins the Internet, it obtains much more value. This additional value is the network effect, in contrast to the standalone value. Network effect has two notable special cases: *superlinear total* and *viral growth*.

- *Superlinear total*. The total impact of a network is more than the linear summation of the impacts of its nodes. In other words, with a network of n nodes, the total impact of the network to society is super-linear, i.e., more than order of n.
- *Viral growth*. The size of the network can grow quickly like a virus spreading.

47. Metcalfe's law and Reed's law

The superlinear total feature of the network effect has been studied by scholars in computer science and other fields. People proposed observations and viewpoints, and some of which became known as "laws". Two of the most popular are Metcalfe's law which postulates a quadratic relationship, and Reed's law which postulates an exponential relationship.

These two laws are summarized in Box 12. They have caused extensive discussions not only in the scientific and technology circles, but also in social sciences and business communities.

Box 12. Network Effect Laws

Metcalfe's law ($V \propto n^2$) is an observation made in 1980 by Robert Metcalfe, an American engineer and the inventor of Ethernet. It says that the value *V* of a network of *n* nodes is proportional to n^2, that is, the effect of the network is proportional to the square of the number of nodes of the networked system.

Reed's law ($V \propto 2^n$) is an observation made in 2001 by David Reed, an American computer scientist and the designer of the User Datagram Protocol (UDP) of the Internet protocol stack. Reed's law says that the value *V* of a network of *n* nodes can scale exponentially with *n*, because the network can form 2^n subgroups.

A common criticism to these laws is that they lack support from real-case data, although they may have captured some intuitive truth.

Recently, researchers used real data from two companies to validate the network effect laws. The results show that Facebook and Tencent data indeed validate Metcalfe's law.

Let us make two assumptions: (1) the number of nodes, *n*, is measured by the Monthly Active Users (MAUs); and (2) the value of a network is measured by the annual revenue of the company. Then, the revenue and MAU data of Facebook and Tencent, from the year 2003 to the year 2019, fit the following equations:

$$\text{Facebook's Value} = 9.69 \times 10^{-9} \times n^2 \text{ USD, RMSD=4.58 Billion USD}$$
$$\text{Tencent's Value} = 9.67 \times 10^{-9} \times n^2 \text{ USD, RMSD} = 4.30 \text{ Billion USD}$$

In other words, real data do show that $V \propto n^2$. RMSD is an acronym for the Root Mean Square Deviation, a fitting error metric. They are in the range of 4.30-4.58 billion US dollars, small numbers comparing to Facebook's revenue of 70.7 billion US dollars in 2019.

In fact, real data fit a "cube law" even better, with smaller RMSD.

$$\text{Facebook's Value} = 4.44 \times 10^{-1} \times n^3 \text{ USD, RMSD=0.81 Billion USD}$$
$$\text{Tencent's Value} = 5.73 \times 10^{-18} \times n^3 \text{ USD, RMSD} = 3.38 \text{ Billion USD}$$

The value of a network falls somewhere between Metcalfe's law and Reed's law.

It is also interesting to note that the per-user revenue numbers of Facebook and Tencent follow a similar growth trend, shown in Fig. 6.15. In 2018, each user contributed about 24 US dollars to a company's revenue.

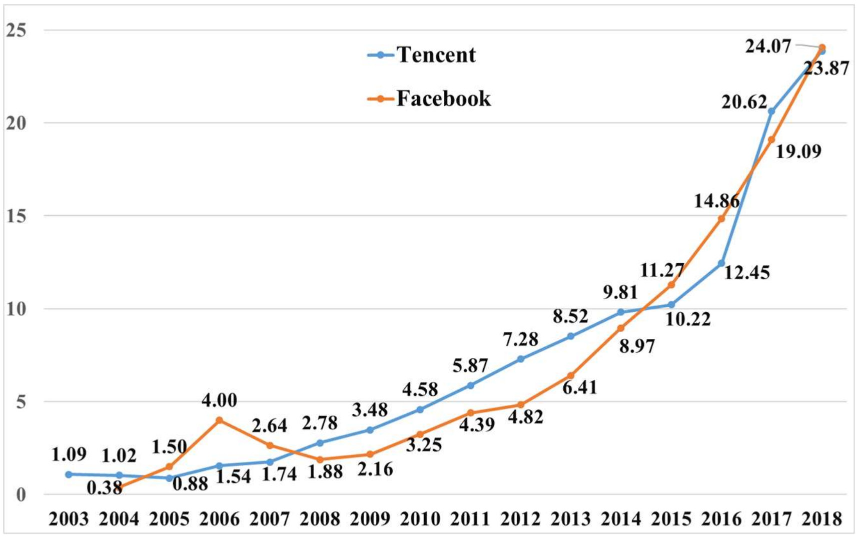


Fig. 6.15 Per-user revenue (amount in USD) growth trends of Facebook and Tencent

48. The viral marketing phenomenon

Viral marketing refers to the following phenomena and practices: (1) viruses, including biologic viruses and computer viruses, spread wide and fast; (2) people learn from how a virus spreads to grow the market of a desirable computing product or service; (3) some computing products or services do grow their markets wide and fast, often enhanced by the network effect. When the market of a product or service grows wide and fast, we say it *becomes viral*.

The term *viral marketing* has become popular since 1997, when an email Web service called Hotmail quickly grew its user base to pass 10 million. Viral marketing has since become a marketing strategy in the field of business management. People have summarized four viral marketing features to characterize why computer viruses spread so wide and fast. A computer virus is (1) connected to the Internet, and has (2) zero purchasing price, (3) zero usage cost, and (4) zero propagation cost.

- *Connected*. This is obvious. Isolated viruses cannot infect.
- *Zero purchasing price*. No one pays any money to buy a computer virus. The virus comes “free of charge”. Similarly, the Hotmail service is free of charge.
- *Easy to use, zero usage fee*. One does not need to learn any manual to “use” a computer virus. “Using” a virus is also free of charge. Similarly, Hotmail is easy to subscribe and use.

- *Easy to propagate, zero propagation fee*. When a product is adopted by a user, there is an easy way for the product to propagate to other users. For instance, the Hotmail company devised a viral marketing trick: at the bottom of every e-mail sent out by any Hotmail user, there is a line saying "Get your free e-mail at Hotmail". This line prompts a non-user to sign up as a Hotmail user.

Example 65. Social networks facilitate viral marketing

Social networks encourage free sharing of information, through various sharing tools such as blog followers, Twitter tags, WeChat moments, and TikTok likes. Not only can a network grow quickly, so can a subnetwork. An individual grassroot user can grow her followers substantially, to become an important influencer in a short time.

For instance, "李子柒 Liziqi", a rural youth based in Sichuan, China and a "Nature and Internet Celebrity", has over 13 million followers on YouTube, as of December 9th, 2020, with half a million new subscribers in a month. Her videos of rural beauty have not only entertained people, but also sparked academic studies in network influencers, digital divide, and urbanization.

6.4.3. Responsible Computing

Responsible computing refers to the ideas and practices to design and use computing products and services responsibly. Responsible computing issues include, among other things, (1) cybersecurity issues, (2) privacy awareness, and (3) respecting professional norms.

49. Cybersecurity issues

An obvious fact is that some information on the Internet is credible and some is not. A not so obvious fact is that not all information one receives from a trusted source, such as a trusted website, is credible. A main reason why we cannot always enjoy trusted Internet and Web services is that the global Internet is constantly under attack. According to a 2020 cybercrime report by the cybersecurity firm McAfee, cybercrime costed companies worldwide over 1 trillion US dollars.

Cybersecurity problems involve hardware, software and people. They come in many forms, including the following:

- *People*. Humans make mistakes, including users, developers, and administrators. Such errors create vulnerabilities which are exploited by attackers. Computing innovations, including hardware, software and services, are created by people. Such an innovation, e.g., viral marketing, can have both beneficial and harmful effects, even for the same person.
- *Malware*. Various malicious software enables an attacker to damage or gain unauthorized access to a computer. Malware examples include computer viruses, Trojan horses and spyware outlined in Box 13.
- *Hardware exploitation*. In 2018, researchers discovered an attack technique called Meltdown that "does not rely on any software vulnerabilities." Instead,

the attack exploits a feature of processor hardware called out-of-order execution, to enable an attacker to read privileged information such as passwords.

- *DoS attacks*. A **denial-of-service** attack overwhelms a computer by sending it a lot of messages in a short amount of time, so that the computer has no resource left to handle normal requests. A **distributed denial-of-service** (DDoS) attack utilizes many computer hosts on the Internet to mount the attack.
- *Spams*. Unwanted emails are often called **spams**. Some spams are easy to identify, such as mass advertising emails and phishing emails. Other times, it is difficult to correctly classify an email as a spam. Suppose a student, who is a student member of both ACM and IEEE, receives a Call for Participation email from the Publicity Chair of an ACM or IEEE international conference. Should this be considered a spam? Professional sources usually offer an option for the student to indicate "I do not wish to receive such emails anymore".
- *Phishing*. A **phishing website** or **phishing emails** ask users for sensitive information, such as a person's password or credit card number. Similar to fishing, phishing offers various types of baits, such as "You are about to receive a subpoena. Click the attachment for details", or "Your account has been frozen and we need your PIN to unfreeze it".

Box 13. Computer Bugs, Malware, Viruses, and Trojan Horses

Computer **bugs** are the term used to refer to all errors in a computer or a network of computers, including software bugs and hardware bugs. This term was attributed to Grace Hopper, a computer pioneer who recorded in 1947 the first bug in computer history: a moth stuck in a relay in a Mark II Computer at Harvard University.

Computer bugs are unintentional errors. In contrast, **malware** is malicious software that is designed to intentionally damage or control a computer. Malware includes viruses, Trojan horses, spyware, etc. **Trojan horses**, like the one in the Greek story, are computer programs which hide their true intentions by disguising as ordinary computer files. **Spyware** collects information about an intruded computer and reports the information back to the attacker.

A **computer virus** is a self-replicating program able to infect computers, like biologic viruses infecting people. A computer virus needs to embed in another program (infecting the program). In contrast, a **computer worm** is a *standalone* self-replicating program that can spread to other computers.

Fred Cohen, an American computer scientist, demonstrated the first computer viruses in 1983 while he was a graduate student at the University of Southern California. He demonstrated that a virus-infected Unix command could let the virus gain control of a computer in five minutes.

Various cybersecurity techniques and processes have been utilized to counter cybercrimes. These counter measures have overheads and are not foolproof.

- *Physical isolation*. For instance, core computing systems in some financial institutions are not connected to the Internet.
- *Firewalls*. A computing system connected to the Internet can be protected by a **firewall** installed between the system and the Internet, to block or filter out undesirable messages. When an institution has multiple computers at multiple sites, they can be interconnected by virtual private networks (**VPNs**), which are secure networks built on top of the public Internet. Such an institution system, protected by firewalls and VPNs, offers an approximation to physical isolation, without the cost of building a dedicated system.
- *Antivirus software*. As the name implies, **antivirus software** is used to detect and kill computer viruses. In fact, modern antivirus tools can detect and remove various malware, including viruses, Trojan horses, and spyware.
- *Cryptography*. Many cybersecurity tools utilize results from **cryptography**, a professional field that studies secure message communication in the presence of adversaries. The basic idea is **encryption**: turn the original message (*plaintext*) into an encrypted message (*ciphertext*). **Decryption** is the reverse process of recovering the plaintext from the ciphertext.

Two types of cryptography are popular. They both use **keys** for encryption and decryption. A key is a specially designed piece of information. In **symmetric-key encryption**, the two parties of communication share the same secrete key for both encryption and decryption. The sender side computer uses an *encryption algorithm* to encrypts plaintext into ciphertext with the key. Then the ciphertext is sent out. The receiver side computer uses the same key and a *decryption algorithm* to decrypt the ciphertext received, to recover the original plaintext.

For instance, the **Caesar cipher** uses a value, e.g., 3, as the key. Its encryption algorithm shifts each plaintext letter 3 positions down the alphabet, and the decryption algorithm shifts each ciphertext letter 3 positions up. Thus, given the plaintext message “HELLO”, the sender encrypts it to create and send out the ciphertext “KHOOR”. Upon receiving the ciphertext, the receiver decrypts it to recover the original message “HELLO”, using the same key (3).

In contrast to symmetric-key encryption, the encryption key and the decryption key are different in **public-key encryption**. Let us use a simplified example to demonstrate how public-key encryption works.

Example 66. Public-key encryption using the RSA method

Suppose a sender computer wants to send the receiver computer a message. The receiver publishes her encryption key for the world to know. That is why it is called the receiver’s **public key**. However, the receiver holds her decryption key secrete (private). Thus, it is called the receiver’s **private key**. Note that only the *receiver’s* keys are involved.

We use below an example from the original paper of the most popular public-key encryption method, known as **the RSA method**. Here, RSA refers to Ron Rivest, Adi Shamir, and Leonard Adleman, who are now professors at the Massachusetts Institute of Technology, University of Southern California, and the Weizmann Institute of Science, respectively. They invented the RSA method when they were at MIT, and received Turing Award in 2002.

1. As a preparation, first encode a message into a sequence of numbers. Then it suffices to show how to securely communicate one number.
2. The sender uses his encryption algorithm F to generate the ciphertext. F takes the plaintext number M and the *receiver's* public key K_P as inputs, to generate the ciphertext number C. In other words, $C = F(M, K_P)$.
3. The ciphertext number C is transmitted from the sender to the receiver, over the Internet. An eavesdropper may intercept C. However, C is enciphered and does not reveal the plaintext M.
4. The receiver uses her private key K_S and the decryption algorithm G to convert the ciphertext number C into the original plaintext M. That is, $M = G(C, K_S)$.

Suppose the plaintext is a 20-character message "ITS ALL GREEK TO ME " (note that there are five space characters). The process goes as follows.

- Each character of the plaintext is first turned into a 2-digit number, by the following converting scheme: space=00, A=01, B=02, …, Z=26. The 20-character plaintext "ITS ALL GREEK TO ME " will result in a large number of 40-digit length: 0920190001121200071805051100201500130500, which is broken into ten 4-digit numbers: 0920 1900 0112 1200 0718 0505 1100 2015 0013 0500. We only need to consider how to securely communicate one number, e.g., 0920.
- The sender uses the following encryption algorithm: $C = M^e \bmod n$, where M is the plaintext number, C is the ciphertext number, and the pair (e, n) is the public key K_P. In other words, $K_P = (e, n)$. Now let us make *the critical magic assumption*: $n = 2773, d = 157, e = 17$. Then, when the plaintext number is $M = 0920 = 920$, the corresponding ciphertext number is easily computed:

$$C = M^e \bmod n = 920^{17} \bmod 2773 = 948 = 0948.$$

- The ciphertext number 0948 is transmitted from the sender to the receiver.
- The receiver uses the following decryption algorithm: $M = C^d \bmod n$, where M is the plaintext number, C is the ciphertext number, and the pair (d, n) is the private key K_S. With the ciphertext number 0948, the corresponding plaintext number is again easily computed:

$$M = C^d \bmod n = 948^{157} \bmod 2773 = 920 = 0920.$$

The entire message is a 20-character plaintext "ITS ALL GREEK TO ME ", which is converted into ten numbers: 0920 1900 0112 1200 0718 0505 1100 2015 0013 0500. These numbers are encrypted as 0948 2342 1084 1444 2663 2390 0778 0774 0219 1655. This encrypted number sequence is transmitted over the Internet.

Note that an eavesdropper knows many things about the communication, because all this information is public, including:

- The encryption algorithm F, which is $C = M^e \bmod n$, and the decryption algorithm G, which is $M = C^d \bmod n$.
- The public key $K_P = (e, n) = (17, 2773)$.
- The character-to-number converting scheme: space=00, A=01, B=02, …, Z=26.
- The ciphertext number sequence 0948 2342 1084 1444 2663 2390 0778 0774 0219 1655, which is transmitted over the open Internet.

The eavesdropper can use the character-to-number converting scheme space=00, A=01, B=02, …, Z=26, to try to recover the plaintext number sequence, which yields the following gibberish message:

I?W?J?N?Z?W?G?G?BSP?, (symbol '?' is for an undefined number)

instead of the plaintext message:

ITS ALL GREEK TO ME .

Why cannot the eavesdropper decode the ciphertext? He lacks the private key $K_P = (d, n) = (157, 2773)$, which only the receiver holds.

Let us return to the *magic assumption*: $n = 2773, d = 157, e = 17$. How are these values determined? There are profound mathematical insights underlying this assumption. We go through the basic idea without involving mathematic details. The receiver decides the values as follows.

- Randomly choose two large prime numbers p and q, and set $n = p \times q$. For this example, the receiver chooses $n = p \times q = 47 \times 59 = 2773$.
- Compute the Euler number $(p - 1) \times (q - 1) = 46 \times 58 = 2668$.
- Randomly choose a large integer d such that $\mathrm{GCD}(d, 2668) = 1$. For this example, she chooses $d = 157$ which satisfies $\mathrm{GCD}(157, 2668) = 1$. Now the receiver has the complete private key information: $K_S = (d, n) = (157, 2773)$.
- Find value e satisfying $(d \times e) \bmod 2668 = 1$. For this example, $e = 17$. Now the receiver can publish the public key: $K_P = (e, n) = (17, 2773)$.

Note that the eavesdropper knows $n = p \times q = 2773$. However, he does not know the two large prime numbers p and q. Consequently, he cannot compute the Euler number $(p - 1) \times (q - 1) = 46 \times 58 = 2668$, or the number $d = 157$.

One may wonder once we know n, if we can determine the two prime numbers p and q by a clever algorithm. This is called the **prime factorization problem**, which has no known efficient algorithm. *RSA relies on this fact.*

As of the year 2020, the largest RSA integer factored is RSA-250, which has 250 decimal digits. A French-US team accomplished the prime factorization task utilizing a network of parallel computers in Europe and the USA. The total computing resources used are roughly 2700 core-years.

A computer network or application can use symmetric-key encryption, public-key encryption, or both. Let us look at **HTTPS**, the secure version of HTTP, which is the most popular protocol for accessing websites today. According to W3Techs, a Web technology survey service, HTTPS is the default protocol for 68% of the top 10 million websites worldwide.

HTTPS uses an encryption layer called *Transport Layer Security* (**TLS**), for secure communication between a Web browser and a website. TLS, thus HTTPS, uses both symmetric-key encryption and public-key encryption techniques. For the long term, HTTPS (and TLS) uses public and private keys between a browser and a server. For the short term, e.g., for each HTTPS GET session, a onetime symmetric session key is automatically generated from the public and private keys, and used by the browser and the server for this session.

When a browser gets a public key from a website, say, ccf.org.cn, how does the browser know for sure that the website is indeed for China Computer Federation? HTTPS also uses **digital certificate**, a bit string issued by a trusted institution called **certificate authority**, to authenticate that a website is who it claims to be. When the website sends the public key to the browser, the website actually sends the digital certificate which encloses the public key. The browser interacts with the certificate authority to make sure that the certificate is indeed from the website of China Computer Federation. Then the browser can use the pubic key, contained in the certificate, to securely communicate with ccf.org.cn.

50. Privacy awareness

Security and privacy are related. Some people consider privacy issues a subset of cybersecurity issues. To differentiate, security is about safeguarding a user's systems and data from attacks, while privacy emphasizes keeping a user's identity and personally identifiable information (PII) *private*.

Personal information is any information relates to a natural person's identity or personally identifiable information, but does not include anonymized personal information. Such information is broad, including personal names, ID numbers, personal photos or videos, website clicks records, voice signals collected by a smart speaker, financial records, medical conditions, and much more.

Students should be aware that privacy can be violated not only when one's obviously sensitive personal data, such as name and credit card number, are explicitly exposed. Technical and social means exist to reveal one's personal data, such as by utilizing metadata, data mining, and artificial intelligence techniques.

For instance, smart meter technology is used by some communities to care for senior citizens who have medical conditions and live alone. When the water usage pattern of a senior citizen's apartment falls below a certain threshold curve, it is likely that the senior citizen is incapacitated and a health worker should go to check up. This is especially beneficial in a pandemic, where more senior citizens are staying at home alone. However, such technology should not be misused.

Websites collect personal data to provide better services and personalized advertisements. Security cameras are installed in residential areas and city blocks to fight crimes. However, they also raise privacy concerns. Where and how to draw the line is still an on-going research area, in the computer science field as well as in the legal field. *IEEE Security and Privacy* is a professional magazine exploring security and privacy issues. On the legal side, the European Union enacted a law framework, called *General Data Protection Regulation* (**GDPR**), which went into effect in 2018. The Chinese law-making body, the National People's Congress of the People's Republic of China, published a request for comments of a Personal Information Protection Act Draft (**PIPA**), in October 2020.

We list below several items of the laws, in a non-legal language:

- The laws facilitate the protection, as well as legal and fair use, of personal information (personal data).
- A person has basic rights to his/her personal information, such as:
  - Right to permit a third party to collect and use personal data
  - Right to timely rectification of personal data
  - Right to be forgotten
  - Right to port one's personal data to another website
- These rights are protected by law, even when a piece of personal data is not owned by the person. A person's cellphone number is protected, even though the number belongs to the telecom company, and the person only "rents" it.
- Another person or institution can collect, store, process, and otherwise use a person's data in a legal and fair way (PIPA used 合法、正当、必要).

51. Respecting professional norms

Computing innovations have **beneficial and harmful impact** on society. The even more challenging part is that the same computing innovation could be both beneficial and harmful, sometimes even to the same person. For instance, the personalized recommendation algorithm used in a video website can recommend relevant videos to a targeted user, which is often beneficial. On the other hand, the same algorithm can create a somewhat closed, even biased, information world to the same user, which may be harmful.

Another challenge relates to **collaboration**. A modern worker seldom works in complete isolation. The Human Sorter project of this book specifically asks students to form groups to do team work. Computing innovations include collaboration tools, such as emails, social network groups, document-sharing software, video conference software, and peer-programming tools. Collaboration often increases productivity and result quality, by synergizing experiences, perspectives, talents, and skills from the team members. However, collaboration without respecting professional norms could decrease productivity or even cause harm. Sometimes, the line of right and wrong is not clear, requiring thoughtful judgement.

One way to deal with these dilemmas is to obtain help from professional communities. As users and developers, students need to be aware of and respect professional norms, which are often codified in a professional society's bylaws. For instance, the Association for Computing Machinery maintains Bylaw 15: *ACM Code of Ethics and Professional Conduct*. Its seven principles are shown in Box 14.

Box 14. Seven Principles of the ACM Code of Conduct

1. Contribute to society and to human well-being, acknowledging that all people are stakeholders in computing.
2. Avoid harm.
3. Be honest and trustworthy.
4. Be fair and take action not to discriminate.
5. Respect the work required to produce new ideas, inventions, creative works, and computing artifacts.
6. Respect privacy.
7. Honor confidentiality.

Let us look at Principle 5 in more details. In plain language, this principle says that the creator and the user of a piece of intellectual work should respect each other. A new computer innovation, with big or small impact, is built using prior work. A creator is also a user. Respect comes in many forms, including the following:

- Acknowledgement. The user should credit and cite prior work used.
- Respecting various protections. The user should respect not only laws, but also regulations, non-disclosure agreement, and code of conduct, which protect the intellectual property rights of the creator.
- Contribution to public good. The creator should not oppose fair use of his work, and should contribute to open source or public domain work. The world's most popular operating system is Linux, not any proprietary one.

Whether a conduct is professional (i.e., right or wrong in plain language) can sometimes be difficult to determine. Thus, not only awareness, but also thoughtful personal judgement, are needed. We use three examples to illustrate the difficulty.

Example 67. Free flow versus professionally sharing of scientific data

Should scientific data flow freely, or in some constrained way?

It seems obvious that scientific data should flow freely. For instance, genome sequence data of COVID-19 viruses are freely available at three databases hosted in USA, Germany and China: GenBank, GISAID, and 2019nCoVR. Anyone can go to these websites and download a genome sequence data file. This helps scientists from all over the world to fight against the COVID-19 pandemic.

However, scientific data may contain sensitive information. Simple-minded insistence on free flow of scientific data may generate privacy, confidentiality, safety, and security concerns. A better practice is to share scientific data following professional norms, which address these concerns.

In fact, this point was made explicit by those scientists who create the GISAID initiative. Look at https://www.gisaid.org/ to find out how they created a model of free access, while “overcoming disincentive hurdles and restrictions”, by requiring users to identify themselves and uphold the GISAID Database Access Agreement.

☶

Example 68. Full disclosure versus responsible disclosure

Suppose a worker in a company finds a vulnerability in a product of the company. The product is used by billions of users. Hackers could exploit the vulnerability to cause harm to society. What should the worker do? Two of the choices are listed below:

- *Responsible disclosure*. Report the vulnerability to the company, without disclosure to the public. This gives the company time to fix the vulnerability, without hackers knowing and exploiting it.
- *Full disclosure*. Announce the vulnerability to the public. The public pressure will force the company to fix the bug before hackers can exploit it.

It is left as an exercise for students to decide which choice better matches the spirit of the ACM Code of Conduct.

☶

Example 69. Morris worm

In 1988, Robert T. Morris, then a first-year graduate student in computer science at Cornell University, released a computer worm (later called **Morris worm**) to the Internet, while doing his research work. Morris intended to count the number of computers on the Internet. He did his work by utilizing unauthorized accesses to spread the worm on those computers.

Due to a bug in the program, the worm caused damages in millions of dollars on thousands of computers. Morris was sentenced to three years of probation plus community work and a fine. Later, Robert T. Morris earned a Ph.D. degree from Harvard University, became a professor at MIT, and was elected to the US National Academy of Engineering, for his contributions in the computer network.

Given the above facts, determine if releasing the Morris worm is wrong, according to the ACM Code of Conduct. This is left as an exercise.

☶

6.5. Exercises

1. Which of the following statements is NOT true regarding network thinking?
   (a) A node in a network does not have to be a computer.
   (b) A node in a network is an abstract or real entity.
   (c) Nodes of a network do not have to send messages to one another.
   (d) There can be no isolated node in a network. Every node has to connect to at least one other node.

2. Which of the following statements is NOT true?
   (a) An accessing point (AP) is not a host, but a networking device, which converts wired and wireless signals to each other.
   (b) A network switch connects multiple hosts running the same protocol, to form a homogeneous network.
   (c) A network switch can be used to connect a LAN running Ethernet and another LAN running WiFi, to form a heterogeneous network.
   (d) Several functions can be packed into a product. For instance, a WiFi device can combine the AP, switch, and router functions into the same the product, called a **WiFi router**.

3. What are the sizes of the following networks? Fill out the following table.

Namespace	**Example**	**Network Size (Number of Nodes)**
Personal name	Joan Smith	Billions
WeChat user	ZhongguanVillager	
URL	www.ict.ac.cn/cs101	
Internet site	www.ict.ac.cn	
Email address	zxu@ict.ac.cn	
IP address	159.226.97.84	
Phone number	189-6666-8888	
MAC address	00-1E-C9-43-24-42	

4. Determine uniqueness and user-friendliness of each of the following naming schemes of networks. Fill out the blank cells in the table.

Namespace	**Example**	**Uniqueness**	**User-Friendliness**
Personal name	Joan Smith	Not unique	Friendly
WeChat user	ZhongguanVillager		
URL	www.ict.ac.cn/cs101		
Internet site	www.ict.ac.cn		
Email address	zxu@ict.ac.cn		
IP address	159.226.97.84		
Phone number	189-6666-8888		
MAC address	00-1E-C9-43-24-42	Unique	Not user friendly

5. When accessing a website by entering https://www.ict.ac.cn/cs101 in a browser, what is the top-level domain (the highest level domain)?
   (a) https
   (b) www
   (c) cn
   (d) cs101
6. Which of the following is NOT a legitimate IP address when using IPv4?
   (a) 0.0.0.0
   (b) 127.0.0.1
   (c) 159.226.97.84
   (d) 159.279.97.84
7. Which of the following is a legitimate IP address when using IPv4?
   (a) 159.226.97.0
   (b) 389.226.97.84
   (c) 159.389.97.84
   (d) 159.279.389.84
   (e) 159.226.97.389
8. What is the role of the Domain Name Service (DNS)?
   (a) Converting an Internet domain name into an Internet Protocol address.
   (b) Converting an Internet Protocol address into an Internet domain name.
   (c) Converting an IP address into a domain name.
   (d) Converting a cellphone number into an Internet domain name.
9. IPv6 has a 128-bit address format. In contrast, IPv4 has only a 32-bit address format. More IP addresses can be provided by IPv6, compared to IPv4. But how many more?
   (a) 32 times as many IP address as IPv4.
   (b) 96 times as many IP address as IPv4.
   (c) 128 times as many IP address as IPv4.
   (d) 2^{32} times as many IP address as IPv4.
   (e) 2^{96} times as many IP address as IPv4.
   (f) 2^{128} times as many IP address as IPv4.
10. All scientific literature of the world forms a graph. Let us call this graph the *scientific literature graph* (SLG), where a paper (or a book) is a node (vertex), and a citation is an edge pointing from the citing work to the cited work. According to network thinking, is the SLG a network?
   (a) No. A computer network is used to pass messages. No message is communicated in the SLG. The citations are just marks.
   (b) Yes. The SLG depicts the connectivity of the network of scientific literature.

(c) No. Network thinking must utilize both abstractions of connectivity and protocol stack.
(d) No. The SLG is not network because scientific literature keeps growing.

11. According to network thinking, is the scientific literature graph (SLG) a static network, a dynamic network, or an evolving network?

(a) The SLG is not a network.
(b) The SLG is a static network.
(c) The SLG is a dynamic network.
(d) The SLG is an evolving network.

12. Albert Einstein published in 1905 a paper on special relativity, with the title *On the Electrodynamics of Moving Bodies* when translated into English. This paper contains no citation. Recall that in the SLG, a citation is an edge pointing from the citing work to the cited work. Which of the following statements is correct?

(a) In the SLG, Einstein's paper is a node with no incoming edges.
(b) In the SLG, Einstein's paper is a node with no outgoing edges.
(c) In the SLG, Einstein's paper is an isolated node, with neither incoming nor outgoing edges.
(d) Einstein was wrong not citing prior work.

13. The second-generation search engines generated much better results than the first-generation search engines. Why?

(a) The second-generation search engines used much better computer systems.
(b) The second-generation search engines collected user data, such as a user's click history data, to improve search results.
(c) The second-generation search engines utilized artificial intelligence techniques and were smarter.
(d) These first-generation search engines only utilized nodes of the network of webpages. The second-generation search engines practiced network thinking better by utilizing both nodes and edges.

14. Let us define a search network as follows: the nodes are the search engine and all search engine users, and there exists an edge if a user submits at least one search query to the search engine and gets a result back. Which of the following statements is NOT correct?

(a) At any moment, the search network is a star network.
(b) At any moment, the search network is a static network.
(c) At any moment, the search network is a dynamic network.
(d) The search network is an evolving network.

15. Which of the following statements is NOT correct?

(a) A bus is equivalent to a fully connected graph with at most one edge being active at any moment.

(b) Suppose *n* nodes are connected by a crossbar switch with *n* fully-duplex ports. Such a network is equivalent to a fully connected graph of *n* nodes with at most *n* edges being active at any moment.
(c) A network of *n* nodes connected by a crossbar switch is a dynamic network, because it does not change its nodes but may change its edges.
(d) A network of *n* nodes connected by a crossbar switch is an evolving network, because it may change its nodes and edges.

16. Which of the following statements is correct regarding packet switching?

(a) It splits multiple users' multiple messages into packets, and statistically transmits the packets in turn.
(b) It splits one user's multiple messages into packets, and statistically transmits the packets in turn.
(c) It packs one user's multiple messages into one packet and transmits it.
(d) It packs multiple users' multiple messages into one packet and transmits it.

17. Refer to Example 62. Verify that the three downloading tasks finish at T_1=8.11 second, T_2=0.44 second, and T_3=1.44 second, respectively.
18. Refer to Example 62. Observe that for both circuit switching and packet switching, all three downloading tasks finish at the moment of 8.11 second. In other words, packet switching does not save time. Then, why bother with inventing packet switching? Select all reasonable explanations.

(a) With packet switching, the three communication tasks proceed simultaneously, without waiting for one another to finish.
(b) With circuit switching, user Wang feels as if his computer is frozen. He needs to wait for 7.31 seconds before seeing any bits transmitted.
(c) With circuit switching, user Zhang feels as if her computer is frozen. She needs to wait for 7.46 seconds before seeing any bits transmitted.
(d) In many applications using circuit switching, e.g., two people having a telephone conversation over a circuit, the channel capacity of the circuit is often not fully utilized. Packet switching can more efficiently utilize the channel capacity, by having multiple communication tasks sharing the same circuit.

19. In general, a packet contains four types of information: payload data, addresses, control information, and error-handling information. What information is contained in the packet header and the packet body, respectively?

(a) The packet header contains addresses, control information, and error-handling information; the packet body contains payload data.
(b) The packet header contains addresses and control information; the packet body contains payload data as well as error-handling information.
(c) The packet header contains addresses; the packet body contains payload data as well as control information and error-handling information.
(d) The packet header contains control information and error-handling information; the packet body contains payload data and addresses.

20. Which of the following statements is NOT correct?
    (a) The Web over Internet protocol stack is a technical foundation of data communication for the World Wide Web.
    (b) The HTTP peering interface is used between two peers: a Web browser and a Web server. This peering interface provides an abstraction, such that the two peers do not need to worry about the layers of protocols below.
    (c) The service interface between HTTP and TCP is used for the TCP layer to support the HTTP layer.
    (d) All packets of an HTTP message from a browser to a server travels along the same physical path.
21. Which of the following statements is NOT correct?
    (a) When an HTTP packet is sent from a browser to a Web server, at least one TCP packet is also sent from the browser computer to the server computer.
    (b) When an HTTP packet is sent from a browser to a Web server, at least one IP packet is also sent from the browser computer to the server computer.
    (c) When an HTTP packet is sent from a browser to a Web server, at least one datalink layer packet is also sent from the browser computer to the server computer.
    (d) When an HTTP packet is sent from a browser to a Web server, a physical layer binary string of 0's and 1's is also sent from the browser computer to the server computer.
    (e) An HTTP packet can be sent from a browser to a Web server, without sending any TCP, IP, or datalink layer packets.
22. Refer to Fig. 6.10. Suppose only routers (shown as brown boxes) may become faulty. What is the minimal number of faulty routers required to disable communication between host A to host B?
    (a) 1
    (b) 2
    (c) 3
    (d) 4
23. Refer to Fig. 6.10. Suppose only inter-router edges may become faulty. What is the minimal number of faulty inter-router edges required to disable communication between host A to host B?
    (a) 1
    (b) 2
    (c) 3
    (d) 4
24. A student subscribes to an optical fiber plan from a reputable ISP, which connects his apartment to the Internet with a 1-Gbps bandwidth connection. How-

ever, he often only experiences a 5-Mbps bandwidth when accessing the Internet. Why is there this huge disparity? Which of the following is NOT a reasonable explanation?

(a) The 1-Gbps bandwidth optic connection is only a portion of the full path from the student's laptop to the accessed website. The rest of the path could be much slower.
(b) The student may be sharing a network switch with neighbors.
(c) Assume the rest of the Internet is fast enough and there is no sharing. The 1-Gbps bandwidth is the maximal bandwidth, i.e., r_∞ in Hockney's formula. User-experienced bandwidth can be much lower.
(d) The student has a fast laptop computer.

25. Which of the following statements is correct regarding data compression?

(a) Lossless compression often generates larger compressed files than lossy compression.
(b) Lossy compression can be used to compress a Go program file.
(c) Lossy compression can be used to compress a binary program file, i.e., an executable program file.
(d) To compress a picture file such as Autumn.bmp, one must use a lossless compression tool, such as the gzip command.

26. Which of the following statements is true about network effect?

(a) A laptop computer connected to the Internet has more value to the user than a standalone computer, because it benefits from network effect.
(b) The total value of a network is the linear sum of the values of its nodes.
(c) Reed's law is wrong because Facebook and Tencent data do not support it.
(d) Viral marketing is absolutely harmful, like biologic viruses hurting people.

27. Cybersecurity protection needs to consider (select one answer)

(a) Software
(b) Software and people
(c) Software and hardware
(d) Software, hardware, and people

28. Which of the following statements is true?

(a) Software bugs are a form of malware.
(b) Malware is a form of software bugs.
(c) Computer viruses are a form of malware.
(d) Spam is a form of software bugs.

29. Which of the following statements is true?

(a) A website has installed firewall, antivirus software and antispam software. It should be considered a trusted website.

(b) All information from a trusted website is credible.
(c) I access a website through HTTPS, the secure version of HTTP. Thus, the information I receives from the website is credible.
(d) There is no perfect cybersecurity, nor absolutely trustworthy website.

30. I receive an email where the Subject part says a famous charity is asking for donation. Which of the following actions is proper?

(a) I should ignore the email since it is a phishing email.
(b) I should donate by clicking the URL in the email and fill out the form with my credit card information and donation amount.
(c) I should find out more details by clicking the attachment of the email.
(d) I should double check before the donation action.

31. Which of the following statements is true about the Caesar cipher?

(a) It uses public-key encryption.
(b) It uses symmetric-key encryption.
(c) It uses a combination of public-key and symmetric-key encryption.
(d) It uses no encryption.

32. Which of the following statements is true about HTTPS?

(a) It uses public-key encryption.
(b) It uses symmetric-key encryption.
(c) It uses a combination of public-key and symmetric-key encryption.
(d) It uses no encryption.

33. Example 66 is mistaken. The eavesdropper CAN decode the ciphertext to get the plaintext. The eavesdropper knows the following public facts: (1) $n = p \times q = 47 \times 59 = 2773$; (2) $e = 17$; (3) the relation between the pair (d, e). He can use the relation $(d \times e)$ mod $2668 = 1$ to find d, and then generates the private key (d, e).

What is wrong with this reasoning?
(a) Fact (1) does not hold, because the eavesdropper does not know $n = 2773$.
(b) Fact (1) does not hold. The eavesdropper only knows $n = 2773$, but not $47 \times 59 = 2773$. Consequently, he cannot use $(d \times e)$ mod $2668 = 1$, because he cannot compute the Euler number $(p - 1) \times (q - 1) = 46 \times 58 = 2668$.
(c) Fact (2) does not hold, because $e = 17$ is not public knowledge.

34. A student has found a bug in Example 66. The eavesdropper CAN decode the ciphertext by using a computer program to try all pairs of prime numbers smaller than $n = 2773$, to discover that the crucial hidden fact that $n = p \times q = 47 \times 59 = 2773$.
Suggest a way for the lecturer to fix this “bug”?

(a) Inform the class that this Example is from an authority, i.e., from RSA, the famous Turing Award winners.
(b) Inform the class that no such computer program exists, because prime factorization is a hard problem.
(c) Inform the class that Example 66 uses small numbers to illustrate the principle of the RSA method. Real applications use much larger *n*, much harder to break. For instance, breaking RSA-250 into two prime numbers needs 1000 years on a laptop computer. HTTPS uses larger numbers.

35. Refer to Example 66. Write a Go program to compute and verify that indeed, $920^{17} \bmod 2773 = 948$ and $948^{157} \bmod 2773 = 920$.
36. Which of the following is NOT an example of personally identifiable information (PII), when discussing privacy protection?

(a) The password of a student's personal computer
(b) A student's full name
(c) A student's university ID number
(d) A student's full face photo

37. Zhang Lei is a product designer at a company which produced a product that is used by millions of users. Zhang discovers a bug in the product, which hackers could exploit to cause harm. What should she do according to the ACM Code of Conduct?

(a) Do nothing, since she can work with colleagues to fix the bug in a couple of weeks.
(b) Follow the practice of *responsible disclosure*.
(c) Follow the practice of *full disclosure*. That is, announce the bug to the public without the consent of the company.
(d) Report the bug to governmental regulators.

38. Refer to Example 69. Is the act of releasing the Morris worm wrong, according to the ACM Code of Conduct?

(a) No, because Morris was just doing his research work, intending no harm.
(b) No, because he was an outstanding academic, as demonstrated by his later achievements.
(c) No, because the damage-causing bug was an accident. Nobody can guarantee that a sophisticated program is bug free.
(d) Yes, because he was convicted and had served his sentence.
(e) Yes, because his action did not do enough to avoid harm, contravening Principle 2 of the ACM Code of Conduct.

6.6. Bibliographic Notes

The chapter quotation is from an invited talk given by Tim Berners-Lee at the 2019 EmTech China Conference [1]. An accessible source of introductory Web programming is [2]. Hockney's formula on latency and bandwidth is discussed by Roger Hockney in [3]. A recent report on bandwidth growth trends of optical fibers can be found in [4]. Metcalfe's law, Reed's law, and evidence based on real data can be found in [5-6]. References [7-8] discussed the social network example of a rural video influencer. References [9-11] discussed software-based and hardware-based cybersecurity examples. The RSA method example is based on material from [12-13]. This book's description of professional norms referenced the ACM Code of Ethics and Professional Conduct, the entry webpage of which is [14]. Three databases offer open access to genomic data on COVID-19 viruses [15-17].

[1] Berners-Lee T. Invited Talk at the EmTech China Conference hosted by MIT Technology Review, Beijing, 2019.

[2] https://www.w3schools.com/.

[3] Hockney, R. W. (1996). The science of computer benchmarking. Society for Industrial and Applied Mathematics, Philadelphia, USA.

[4] Hecht J. (2016). Great leaps of light. IEEE Spectrum, 53(2):28-53.

[5] Metcalfe, B. (2013). Metcalfe's law after 40 years of Ethernet. Computer, 46(12), 26-31.

[6] Zhang, X. Z., Liu, J. J., Xu, Z. W. (2015). Tencent and Facebook Data Validate Metcalfe's Law. Journal of Computer Science and Technology, 30(2):246-251.

[7] Jewell, J R. (2020). YouTube Commentary: "李子柒 Liziqi" — Nature and Internet Celebrity in the Time of the Coronavirus. The Arts Fuse. https://artsfuse.org/194752/youtube-commentary-李子柒-liziqi-nature-and-internet-celebrity-in-the-time-of-the-coronavirus/

[8] Zhang, X. (2020). The rural video influencers in China: on the new edge of urbanization. Master of Arts Thesis, Cornell University.

[9] Smith, Z. M. and Lostri, E. (2020). The Hidden Costs of Cybercrime. McAfee Report. https://www.mcafee.com/enterprise/en-us/assets/reports/rp-hidden-costs-of-cyber-crime.pdf.

[10] Cohen, F. (1987). Computer viruses: theory and experiments. Computers & security, 6(1), 22-35.

[11] Lipp M, Schwarz M, Gruss D, et al. Meltdown: reading kernel memory from user space. Communications of the ACM, 2020, 63(6): 46-56.

[12] Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2), 120-126.

[13] Zimmermann, P. (2020). Factorization of RSA-250. https://lists.gforge.inria.fr/pipermail/cado-nfs-discuss/2020-February/001166.html.

[14] https://www.acm.org/code-of-ethics.

[15] https://www.ncbi.nlm.nih.gov/sars-cov-2/.

[16] https://www.gisaid.org/.

[17] https://bigd.big.ac.cn/ncov/?lang=en.

7. Projects

This chapter describes four practice projects, which offer students the opportunity to design an abstract computer, a real computer, and two computer applications. One of the applications creates a Web application. The overall design of the four projects is illustrated in the following table. The project order can be adjusted. For instance, Project 3 can go before Project 2.

Table 7.1 Overall design of the four projects

Project	Content	Purpose	Work Mode
Project 1	Turing Adder	Design an abstract computer Appreciate Turing machine	Mostly work alone
Project 2	Text Hider	Design a computer application Appreciate programming	Mostly work alone
Project 3	Human Sorter	Design a real computer Appreciate how algorithm, software and hardware work together	Team work
Project 4	Personal Artifact	Appreciate creative expression Design a network application of dynamic webpage	Mostly work alone

The projects also involve responsible computing practices, based on the principles of the ACM Code of Conduct. These projects emphasize independent work, collaboration, and acknowledgment. All used intellectual work should be acknowledged. A student can interact with others in doing the project, but the final product should be his/her own work.

More detailed projects information, based on the practice of the CS101 course at UCAS, can be found in Supplementary Material at the companion website.

The Supplementary Material also includes a sample learning/teaching platform, which includes the following resources:

- Software for students to do homework assignments (exercises) and have them checked and scored.
- Software for students to hand in project reports.
- Instructions and resources for students to download to their personal computers. Students are instructed to create a Linux environment for the course, including all necessary code and data files.
- A software tool to do the Turing Adder project.
- A software tool for the Text Hider project. It includes skeleton code, which serves as a starting point for the students to work on. The code follows good programming practices.
- A website for the Personal Artifact project, which includes a library of dozens of dynamic webpages from the teaching team and previous students.

7.1. Turing Adder: Turing Machine for Serial Additions

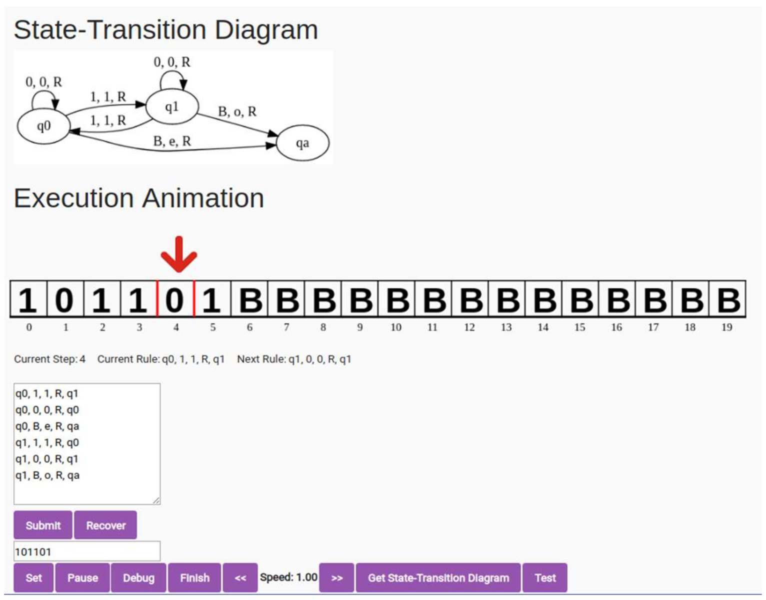
The Turing Adder project augments students' understanding of an abstract computer: the Turing machine. It asks a student to do unary addition of 4-bit numbers, binary addition of 4-bit numbers, and binary addition of arbitrary-bit numbers. They are all bit-serial adders of unsigned positive numbers.

- **Objective**. Use the software tool provided by the teaching/learning platform, design a Turing machine by entering its state-transition table, for each of the following three computational tasks. A snapshot of the software tool used at UCAS is shown in Fig. 7.1.
  - Unary addition of two unary numbers, each of which is at most 4-bit long. For instance, 11 + 111 = 11111. That is, 2 + 3 = 5. Note that the result can be longer than 4 bits.
    Binary addition of two binary numbers, each of which is 4-bit long. For instance, 1011 + 0111 = 11010, or 11 + 7 = 18. Note that the result can be longer than 4 bits.
  - Binary addition of two binary numbers, each of which is of an arbitrary length between 4 and 64 bits. The platform tool automatically selects three lengths and checks the Turing machine for correctness.
- **Material and Method**. The main material to reference is Section 3.2.3 and the software tool from the teaching/learning platform.

For each of the three tasks, we suggest students to adopt the following design and development procedure.

- A student first develops the Turing machine on his/her personal computer, which is connected to the platform.
- The student can see the corresponding state transition diagram of the Turing machine. Some students prefer such a graphic representation of a Turing machine, especially when the number of states is small.
- The student can see execution animation of a Turing machine, as well as debug the design, either in a step-by-step mode or in a run-to-completion mode.
- After the student is sure about the correctness of the Turing machine, he/she can submit the work via the platform tool. Submission cannot be reversed.

- **Project Report**. Students do not need to turn in any project report. The project completes once all three Turing machines are submitted. The platform tool automatically checks for duplications.
- **Scoring**. Each student is scored for correctness of the three Turing machines.
  - 50% for the 4-bit unary adder
  - 20% for the 4-bit binary adder
  - 30% for the arbitrary-bit binary adder



Current Step: 4 Current Rule: q0, 1, 1, R, q1 Next Rule: q1, 0, 0, R, q1

```
q0, 1, 1, R, q1
q0, 0, 0, R, q0
q0, B, e, R, qa
q1, 1, 1, R, q0
q1, 0, 0, R, q1
q1, B, o, R, qa
```

```
101101
```

Fig. 7.1 Snapshot of an example platform tool for the Turing Adder project

7.2. Text Hider: Program to Hide Text in Picture

The Text Hider project represents a computer application. It hides the content of the text file hamlet.txt in the picture of the image file Autumn.bmp. This is done by a Go program hide-0.go, which stores the modified picture in another image file doctoredAutumn.bmp. Students also need to develop a Go program show-0.go, to recover the content of the text file hamlet.txt from doctoredAutumn.bmp.

- **Objective**. Develop a Go program to hide the content of a text file in an image file, as well as a Go program to recover the content of the text file from the doctored image file. The basic principle of hiding is to replace the least significant two bits of one byte of the Pixel Array by two bits of a character, as shown in Example 49. The objective has four detailed interpretations:
  - The doctored image file must show no visible difference from the original image file, as demonstrated in Example 49.
  - The two programs should work for other text and image files, assuming the BMP image file format. In other words, it should work if we want to hide the content of the text file aMiddleSummersDream.text in the picture of the image file Spring.bmp.

– This project emphasizes good programming practice, as illustrated in UKA Unit 6 in Section 2.2. Both hide-0.go and show-0.go should follow such practices.
– This project emphasizes independent work. Each student should complete this project on his/her own. To help check for work independence, students must use the two files, hamlet.txt and Autumn.bmp, which are provided by the platform. Use by other means, such as borrowing a file from a fellow student through a USB stick, could cause the project to fail, even without the student being alerted.

Material and Method. Most material is already provided in Example 49. Recall that the algorithm for hide-0.go is as follows:

Algorithm for hide-0.go
- **Input**: A text file hamlet.txt and an image file Autumn.bmp.
- **Output**: A doctored image file doctoredAutumn.bmp
- **Steps**:

```
1. Read Autumn.bmp into variable p                // p for picture
2. Read hamlet.txt into variable t                // t for text
3. Hide the length of hamlet.txt in the first 32 bytes of Pixel Array
4. Hide hamlet.txt in variable p in the remaining bytes of Pixel Array
5. Write p to file doctoredAutumn.bmp
```

Students need to pay attention to the following items:
– Design and develop the hide-0.go program to follow good programming practices. The code in the textbook is incomplete and does not pay much attention to good programming practices.
– Design an algorithm for show-0.go.
– Develop the show-0.go program and follow good programming practices.
– Verify that the two programs indeed hide and recover a text file in an image file, by executing them on the baseline files hamlet.txt and Autumn.bmp.

- **Project Report**. Every student needs to turn in a project report, including:

– Description of the student's own design on how to hide and show
– Source code of hide-0.go and show-0.go
– Evaluation of programs' executions
– Reflection and discussion, including any unusual happenings
– Acknowledgements, if any

- **Scoring**. Each student is scored for programming (including good programming practices) and independent work.

– 85% for the student's design and code, if achieving the project objective
– 15% for communications clarity of the project report

7.3. Human Sorter: Team Computer for Quicksort

The Human Sorter project invites students to design a real computer: a computer made of a team of students to do quicksort. By designing and testing this team computer, students learn team work. Each student also reports to the team how computer hardware, instructions, and program execution work as a whole.

- **Objective**. Design a team computer of students to execute a quicksort program, to sort the students in the team from order by name to order by height (Fig. 7.2).

  Each student needs to produce a design, including:
  - The team computer organization
  - The instruction set of the team computer
  - The quicksort program made of a sequence of such instructions
  - The evaluation record of program executions

  The evaluation must show that the design satisfies three correctness properties:
  - *Result correctness*: the students are indeed ordered by height.
  - *Algorithmic correctness*: the execution implements the quicksort algorithm.
  - *Systems correctness*: the team computer executes the program sequentially, i.e., step-by-step, one instruction after another.

- **Material and Method**. A CS101 course usually enrolls hundreds of students. This large class is divided into small teams. Each team needs a team leader and should consist of no more than 30 students. Remember that the quicksort algorithm takes at least $O(n \log n)$ steps.

  - Data, memory, processor are all made of humans (students). The team computer is a human computer, not an electronic computer.
  - The team computer hardware organization must consist of memory and processor. No I/O devices are necessary.
  - For tips on computer hardware design, refer to Sections 2.3 and 5.3.3. But, forget not that this is a human computer. Each student and each team can use their imagination in the design. It does not have to be a computer following the von Neumann architecture. For instance, the instructions of the quicksort code do not have to be stored in the memory of the team computer.
  - For tips on instruction set design and the quicksort program implementation, refer to Sections 2.3, 4.3.3 and 5.3.3.
  - It is best that program executions are conducted on a sufficiently large open space, such as on a big lawn that can accommodate hundreds of students.
  - Each team must keep meticulous records of execution, down to the execution of every instruction step. For each execution of the quicksort program, the initial configuration, the final configuration and the number of steps are reported. That is one reason why the team computer in Fig. 7.2 has a *monitor*, who keeps the record of every execution step. The *stepper* serves as the system clock, to count the steps and to make sure that the rest of the system follows the step-by-step rule. *Humans tend to violate this rule*.

- **Project Report**. Every student needs to report his/her design to the team. Each team comes up with its design by selecting and combining designs from the team members, through a process of *rough consensus and running code* (the IETF mantra). At the end of the project after the team's execution runs, each student hands in a project report, which contains the following contents:
  - The student's own design, which could be a revised version built on the team's design
  - Evaluation of the team's design, with sufficient evidence
  - Reflection and discussion, including any unusual or funny happenings
  - Acknowledgements

Note that there is no project report handed in by a team. However, each team should present its design to the entire class of hundreds of students, by class presentation, posters, or webpages.

In the UCAS CS101 course, this project normally takes 3-4 sessions:

- Session 1. Individual students go through their designs
- Session 2. Teams fix their designs
- Session 3. Teams run their quicksort programs on their team computers
- Session 4. Teams report their projects to the entire class

- **Scoring**. Each student is scored for design, team work, and communication.
  - 80% for the student's design, if it achieves the project objective
  - 10% for the team leader's report
  - 10% for communications clarity of the project report

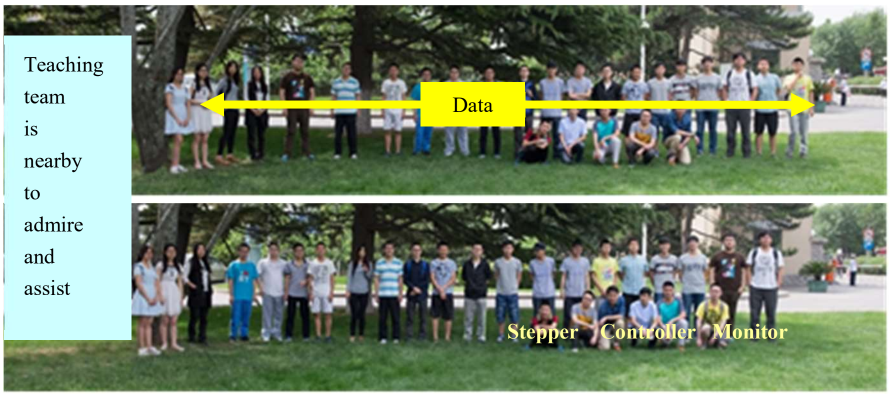


Fig. 7.2 A team computer sorts a team of students: from order by name to order by height
Photos are blurred for privacy Photos credits: Haoming Qiu

7.4. Personal Artifact: Web Page of Creative Expression

This Personal Artifact project encourage students to demonstrate their creative expression by designing a dynamic webpage. Students are also encouraged to learn any additional knowledge needed by themselves.

- **Objective**. Design a dynamic webpage of creative expression. Successful completion of this personal artifact needs students to demonstrate their self-learning capability, as the textbook or the lecturers do not cover much Web programming material, except that in Section 6.3.2.

  Each student needs to produce and show to the class a webpage:
  – A dynamic webpage including HTML, CSS, and JavaScript code
  – A webpage showing creative expression

- **Material and Method**. The students each use the material in Section 6.3.2 as a starting point, create a dynamic webpage, and upload the files and the project report to the class website. In the process, students may need to reference the following sources of resources.

  – The Internet. For instance, additional Web programming knowledge can be found at https://www.w3schools.com/.
  – The library of Personal Artifacts examples in the Supplementary Material. Previous students have created a dozen genres. The top three are scientific artifacts, games, and artistic communications.

  Three points should be emphasized in producing the artifact.
  – Most of the artifact should be made by the student.
  – Used material from other sources should be properly acknowledged.
  – Make an effort to control the program size of your webpage. 100-200 lines of HTML/CSS/JavaScript code can go a long way. Avoid copying existing code into your product.

  Some students created webpages for "My Beautiful Homeland". The idea is wonderful. Some students created a webpage with links to beautiful photos enticing people to visit their homeland as tourists. However, such projects could get a low score for two problems: (1) the webpage is not dynamic, and (2) most photos are creations of other people.

- **Project Report**. The project report has a free form, including:

  – Articulation of the artifact, including reflection and discussion
  – Source code of the webpage program
  – Acknowledgements

- **Scoring**. Each student is scored for design and communication.

  – 90% for the student's design, if it achieves the project objective
  – 10% for communications clarity of the project report

8. Appendices

8.1. Multiples and Fractions

Base 10	Base 2	Symbol	Prefix	Example
10E24	2E80	Y	Yotta	
10E21	2E70	Z	Zetta	ZB, zettabytes, the world's data volume
10E18	2E60	E	Exa	EFLOPS, Exa floating-point operations per second, speed of supercomputers
10E15	2E50	P	Peta	PB, petabytes, a server's storage capacity
10E12	2E40	T	Tera	TB, terabytes, a disk's storage capacity
10E9	2E30	G	Giga	Gbps, giga bits per second, bandwidth of a local area network
10E6	2E20	M	Mega	MW, Mega Watt, power consumption of a supercomputer
10E3	2E10	K	Kilo	Kg, Kilogram, weight of a laptop computer
10E2		H	Hecta	
10E1		da	deca	
10E-1		d	deci	100 ms, 100 milliseconds, good interaction time when using a computer
10E-2		c	centi	
10E-3		m	milli	mm, millimeter, size of a semiconductor die
10E-6		µ	micro	µs, microsecond, communication latency between two nodes of a supercomputer
10E-9		n	nano	nm, nanometer, feature size of a semiconductor technology
10E-12		p	pico	pJ, picojoule, energy consumption of an arithmetic operation in a computer
10E-15		f	femto	fs, femtosecond, laser wavelength
10E-18		a	atto	
10E-21		z	zepto	Landauer's principle: Erasing one bit needs zeptoJoule energy

Note that the same symbol may imply different values using base 10 or base 2.

T = 1.00 × 2E40 = 1,099,511,627,776 ≈ 1.10 × 10E12 ≠ 1.00 × 10E12

G = 1.00 × 2E30 = 1,073,741,824 ≈ 1.07 × 10E9 ≠ 1.00 × 10E9

M = 1.00 × 2E20 = 1,048,576 ≈1.05 × 10E6 ≠ 1.00 × 10E6

K = 1.00 × 2E10 = 1,024 ≈1.02 × 10E3 ≠ 1.00 × 10E3

This difference is the reason why some users thought that a vendor may give them fewer resources in a computer product. For instance, a 1-TB hard disk may actually have only a storage capacity of 10E12=1,000,000,000,000 bytes, not 2E40=1,099,511,627,776 bytes, a shortage of about 100GB.

8.2. Programming Basics

Programming in the Go programming language (Golang) is explained in preceding chapters. For ease of reference, we summarize the programming constructs in four tables: shell commands, packages, statements, and data types.

Table 8.1 Commands in a Linux shell used in this book

Command	Purpose	Example
cat	Print file to standard output	>cat hello.go print hello.go to screen
cd	Change directory	>cd .. change to the parent directory
display	Display a picture	>display ucas.bmp display ucas.bmp on screen
./hello	Execute binary code hello in current directory	
go build	Compile Go source file into executable file	>go build hello.go compile hello.go into executable file hello
go run	Compile and execute Go program	>go run hello.go compile and execute hello.go
ls	List files in current directory	>ls . list files of current directory
Tab key	Automatically complete a command	
↑	Execute the previous command	
Ctr-C	Exit the current command	
Ctr-S	Save the file in editing	

Table 8.2 Golang packages used in this book

Package	Example
fmt	For input and output functions, e.g., fmt.Println("Hello")　　//print "Hello" to display fmt.Printf("One=%d",1)　　//print "One=1" to display fmt.Scanf("%d",&A)　　//enable user to enter integer to variable A
io/ioutil	For reading/writing files, e.g., p, _ := ioutil.ReadFile("./ucas.bmp")　//read data in ucas.bmp to variable p ioutil.WriteFile("./mucas.bmp", p, 0666)　//write p to mucas.bmp
math	For mathematical functions, e.g., math.Pow(2,3)　//returns 2^3=8
os	For interaction with operating system, e.g., V := os.Args[0]　//os.Args is an array of command parameters

Table 8.3 Golang statements used in this book

Statement	Example
Assignment	v := 1 // assign 1 to v
	a,b := true,false // assign true to a and false to b
Break	Terminate a loop, e.g.,
	for i:=0;i<5;i++{
	if(i>=3) break
	fmt.Printf("%d ",i)
	}
	The code above will print 0 1 2, because the loop is terminated by break when i==3
Continue	Go to beginning of the next loop iteration, e.g.,
	for i:=0;i<5;i++{
	if(i<3) continue
	fmt.Println("%d ",i)
	}
	The code above will print 3 4, because when i<3, fmt.Println() is skipped
Declaration	v := 3 // declare a variable v and assign 3 to it
	var v int = 3
	const c = 3 // declare a constant c and assign 3 to it
	const c int = 3
For loop	// Compute sum of an integer array
	sum := 0
	var arr [5]int = [5]int{0,1,2,3,4}
	for i:=0; i<len(arr); i++{ // i:=0; is init statement, i<len(arr); is
	sum += arr[i] // condition and i++ is post statement
	}
Function definition	// Define an addition function
	func Add(a int, b int) int { // Add is the function name
	return a+b // a,b are parameters of the function
	}
Function call	c := Add(1,2) // call Add function to obtain c = 3
If statement	if a<b {
	fmt.Println("Smaller") // if a<b, this statement will be executed
	}else{
	fmt.Println("Not smaller") // if a>=b, this statement will be executed
	}
Return	Specify the return value of a function, e.g., "return a+b" in Add

Table 8.4 Go data types used in this book

Data Type	Example
array	var a [10]int // define an array consisting of 10 integers
	var primes [3]int = [3]int {2,3,5} // define an array consisting of 2,3,5
	var p0 int = primes[0] // primes[0] is the zero-th element of primes
bool	var a bool = true // define a bool variable named “a”, whose value is true
	b,c := true,false // define 2 bool variables whose values are true, false
byte	var X byte = ‘a’ // define byte variable X, assign ASCII encoding of ‘a’ to it
int	var y int = 1 // define a signed integer variable whose value is 1
slice	var prime_array [3]int = [6]int {2,3,5} // prime_array is an array
	var s []int = primes[0:2] // s is a slice representing the first 2 elements of primes
	var u []int = make([]int,3) // u is a slice representing a nameless array consisting of 3 integers
string	var str1 string = “directly declaration” // define a string
	var n int = len(str1) // len() returns the number of characters in str1
uint	var i uint = 1 // define an unsigned integer variable whose value is 1

In many Go programs of this book, e.g., fib.dp.bu.go in Example 34 of Section 4.3.1, input data are hardwired into the code, to simplify the code. This is bad programming practice. The fib.dp.bu.go code only works for F(5). It is better to change

```
func main() {
  fmt.Println("F(5)=", fibonacci(5))
}
```

in fib.dp.bu.go to the following code, which enables user to enter a number.

```
func main() {
  var n int = 0
  fmt.Printf("Please enter a natural number between 0 and 92: ")
  _,err := fmt.Scanf("%d", &n)          // n = user-entered integer
  if err != nil {
    fmt.Println("Input Error: Not a number")
    return
  }else if n < 0 {
    fmt.Println("Input Error: Please enter a non-negative integer.")
    return
  }else if n >92 {
    fmt.Println("Input Error: The number is too large. The program overflows.")
    return
  }
  fmt.Printf("F(%d) = %d\n", n, fibonacci(n))
}
```

8.3. Pointers to Supplementary Material

The companion website cs101.ucas.edu.cn provides supplementary material for (1) lecture and projects slides, (2) text, graphics, sound, and video files, and (3) sample programs and tools which help provide a teaching and learning platform.

The website also contains solutions to even-numbered homework exercises, as well as source code of all programs in this book. The following are included.

- Go programs:

  binary.search.go
  fib-10.go
  fib-5.go
  fib-50.go
  fib.binet-50.go
  fib.dp-5.go
  fib.dp-50.go
  fib.dp.big.go
  fib.dp.go
  fib.dp.bu.go
  fib.go
  fib.matrix.go
  fib.Uint.go
  hash.search.go
  hello-1.go
  hello.go
  hide-0.go
  linear.search.go
  name_to_number-0.go
  name_to_number-1.go
  name_to_number.go
  null.go
  parity.go
  pi.go
  pointer.go
  replace.go
  symbols.go
  testPoint123.go
  WebServer.go

- Web programs:

myFirstWebPage.html
staticChildrensDay.html
ChildrensDay.html

Index

A

ABC features, 25
absolute file name, 222
abstract computer, 3, 5
Abstraction, 8, 206
Access Point, 268
accumulator, 80
Acu-Exams., 31
adapt, 146
address, 80, 200
Addressing Mode, 216
algorithm, 18, 147
Algorithmic thinking, 31, 146
ALU, 76
Amdahl's Law, 249
analog values, 57
antivirus software, 295
application layer, 278
application software, 242
array, 68, 69, 213
ASCII, 63
assembly language, 79
assignment, 69
asymptotic notation, 151
Automata, 120
Automatic execution, 8, 26
automaton, 121
average case, 177
average time complexity, 183
axiomatic systems, 119
axioms, 119

B

Babbage's problem, 42
bandwidth, 288
barber paradox, 136
base, 80
base register, 80
basic verb, 71
Batch, 45
bees metaphor, 8
Bell's law, 44
benchmarks, 202
beneficial and harmful impact, 299
best case, 177
biased exponent, 211
big endian, 200
Big endian, 200
binary, 58
Binary, 76
binary search, 179
Binet's formula, 27
BIOS, 242
bit, 18, 27
Bit accuracy, 27
bit-shift, 112
BMP, 222
body, 278
Boolean algebra axioms, 101
Boolean expressions, 101
Boolean function, 102
Boolean Logic, 98
bottleneck, 249
broadcasting, 275
Bubble Sort Algorithm, 149
Bugs, 23
bus, 274
Bush's problem, 42, 44
business computing., 46
byte, 63, 69, 214
byte-addressable, 58

C

cache, 78, 252
Caesar cipher, 295
camel notation, 12
Caryl Rusbult's investment model, 29
certificate authority, 298
character, 213
Church-Turing Hypothesis, 132
Church-Turing Thesis, 133
Circuit switching, 276
client, 43, 268
clock cycle, 235, 245
clock frequency, 235
cloud computing, 45
code, 18, 21
COG properties, 206
collaboration, 299
combinational circuit, 234
Combinational Circuits, 229
command-line prompt., 65
comments, 22

Compilation, 21
compile-time errors, 23
Completeness, 137
complexity, 5, 8
Complexity class NP, 191
Complexity class P, 190
Computability, 132
computable problem, 132
computational lens, 47
computational process, 18
Computational Thinking, 25, 47
computer application, 3, 5
Computer Science, 18, 34
Computer simulation, 41
Computer System, 21
computer virus, 292
conclusion, 119
conditional, 227
conjunction, 100
conjunctive normal form, 103
Connectivity, 266, 269
Consistency, 137
constant, 68
constrained, 207
Constructive abstraction, 29
constructs, 73
consumer computing, 46
context, 228
contrapositive, 111
control abstractions, 198, 207, 227
control dependency, 254
converse, 110
Coping with complexity, 199
correctness, 97
Correctness, 8
CPU, 76
creative expression, 5
crossbar, 274
cryptography, 295
current directory, 222
curse of exponentiation, 41
Cyberspace, 18

D

D flip-flop, 235
data, 18, 228
data abstractions, 198
Data Abstractions, 207
data cache, 253
Data compression, 289
data dependency, 254
data link layer, 278
data representation, 200
data types, 67, 207
Decidability, 137
decimal, 58
Decryption, 295
define, 146
denial-of-service, 294
dependencies, 254
dereference, 216
design, 146
dictionary, 178
digit set, 209
digital certificate, 298
digital economy, 36
digital infinity, 8
digital symbol, 18
Digital Symbol Manipulation, 57
digits, 208
directory, 221
discretization, 57
disjunction, 100
disjunctive normal form, 102
distributed denial-of-service, 294
Divide-and-Conquer Algorithms, 153
DNS, 270
documentation, 72
domain, 116
domain name, 272
dot notation, 12
Dynamic networks, 274
dynamic programming, 74, 166
dynamic webpage, 282

E

edges, 269
Effectiveness, 8
Efficiency, 254
Eight Understandings Within, 31
Elementary Web Programming, 282
Embedded, 43
encoding, 19
encryption, 295
enterprise computing, 46
Entscheidungsproblem, 133
Evolutionary networks, 274
exceptions, 248
exclusive-or, 100
Explicit and implicit input/output, 131
exponent, 210
expression, 69

F

Fibonacci Computer, 79, 240
Fibonacci numbers, 26
File Abstraction, 221
file system, 221
files, 18, 221
finite automaton,, 123
Finite states, 131
firewall, 295
firmware, 242
flip-flops, 234
floating-point number, 212
floating-point numbers, 210
FLOPS, 202
for loop, 69, 227
formatting verbs, 67
Free flow, 300
full adder, 106, 231
Full disclosure, 301
full-duplex, 275
function, 21, 73, 228
Function, 227

G

gates, 229
gateway, 269
GDPR, 299
generality, 97
generalizable, 207
Gödel’s Incompleteness Theorems, 136
Gödel's first incompleteness theorem, 137
Gödel's second incompleteness theorem, 138
gold metaphor, 38
Goodstein theorem, 139
graph, 269
graph coloring problem, 192
greatest common divisor, 148
Greedy Strategy, 169
GUI, 45

H

halting problem, 133
Hamiltonian path problem, 193
Harvard architecture, 248
hash function, 183
hash index, 183
Hash search, 183
hash table, 183
hashing, 183
header, 278
heap, 228
hexadecimal, 61
high-level formatting verb, 71
high-level language program, 21
Hockney's formula, 288
home directory, 222
homepage, 273
Host, 268
HTTP, 273
HTTPS, 273, 298
Hub, 268
hypertext, 44

I

I/O devices, 76, 78, 201
implication, 99
Incomputable Problems, 133
index, 69, 80, 208
index register, 80
indirect addressing, 216
inference rules, 119
information and communication technology (ICT), 36
information hiding, 198
information society, 37
information technology, 36
information-hiding, 230
infrastructure software, 242
Input, 146, 147
Insertion Sort Algorithm, 153
Instance, 270
instruction cache, 253
instruction cycle, 245
Instruction driven, 76
instruction pipeline, 239
Instruction Register, 250
instruction set, 79, 239
instructions, 76
integer, 213
Integer Multiplication, 162
Interactive, 45
Internet, 266, 271
interpreter, 23
inverse, 110
IP address, 270
IPv4 addresses, 271
IPv6 addresses, 271
IR, 250

ISP, 268
IT industry, 36
IT professionals, 38

K

Keck’s law, 39
keys, 295
Kleene algebra axioms, 107
Kleene expressions, 107
Kleene function, 109
Kleene logic, 107
Knuth's Characterization of Algorithm, 147
Knuth's Test, 10, 34

L

latency, 288
length, 69
liar paradox, 136
linear search, 179
little endian, 200
local area network, 268
locality, 252
logic, 47
Logic thinking, 31, 97
loop, 68
loop body, 69
loopback domain name, 273
loopback IP address, 273
Lossless compression, 289
Lossy compression, 289
low-level language program, 23

M

machine code, 21
Magic numbers, 72
main memory, 76
maintain the code, 72
make, 214
Manipulation, 18
mantissa, 210
MAR, 250
Matrix Multiplication, 163
MDR, 250
measure, 146
Mechanical Theorem Proving, 120
memory, 76
Memory Address Register, 250
memory cell, 234
Memory Data Register, 250
Merge Sort Algorithm, 155
message storage problem, 199
messages, 276
metadata, 222
Metcalfe’s law, 290
metropolitan area network, 268
middleware, 242
Mobile Internet, 45
modeling, 19
Modularity, 8
modules, 198, 205
Modularization, 229
Moore’s law, 39
Morris worm, 301
Motherboard, 78
multicore, 78
Multimedia, 45
multiplexer, 233

N

namespace, 270
Namespace, 270
naming, 270
negation, 111
negation properties, 116
network, 45, 267
Network effect, 290
Network Laws, 288
network layer, 278
Network Thinking, 266
Network Topology, 274
networking devices, 268
No stop in the middle, 131
nodes, 267
Nordhaus’s law, 39
Normal forms, 102
normalized significant, 211
NP-complete, 193

O

object, 285
objective, 207
offset, 80
operations, 18
order, 116
Output, 146, 147
overflow, 33

P

P versus NP problem, 192
Packet switching, 276
packets, 276
Palindromes, 123
Paradoxes, 133
parallel computer, 255
parallel computing, 255
parity, 112
peak speed, 254
Peano Arithmetic, 137
peering interface, 280
PEPS, 18
persistence, 221
Personal, 45
phishing emails, 294
phishing website, 294
physical layer, 278
PIPA, 299
pixel, 222
P-M-I/O, 76
Pointer, 216
Polynomial Church-Turing Thesis, 133
Portable, 45
positional notation, 208
positional number systems, 208
Postel's Robustness Principle, 245
practical, 198
Precedence, 227
predicate, 115
Predicate refinement, 117
Predicative Logic, 115
premise, 99
prime factorization problem, 297
Princeton architecture, 248
Principle of Harmony, 47
Privacy awareness, 298
private key, 295
Problem, 19
Problem Solving, 18
process, 203
processor, 76
professionally sharing of scientific data, 300
program, 18
program counter, 80
program cycle, 245
program status register, 80
Programming, 19
proposition, 99
Propositional Logic, 99
protocol, 276
protocol stack, 266, 276
public key, 295
public-key encryption, 295
Public-key encryption, 295
Pumping Lemma, 144

Q

quantifiers, 115
quicksort, 175
quotation marks, 12

R

Randomization Strategy, 175
reads, 221
real computer, 3, 5
reducible, 133
redundancy, 281
Reed's law, 290
relative file names, 222
Remark, 270
representation, 58
Responsible Computing, 288, 293
responsible disclosure, 301
RFCs, 270
right-moving Turing machine, 144
ripple-carry adder, 231
root directory, 222
rounding error, 30
routers, 268
RSA method, 296
runtime errors, 23
Russell's paradox, 136

S

scientific computing, 45
seamless execution, 198, 205
seamless transition, 243
Seamless transition, 8
search, 178
Search Algorithms, 178
search engine, 275
Selection, 227
Sequencing, 227
sequential circuit, 234
sequential circuits, 236
sequential computers, 255
Serial adder, 236

Serial execution, 76
server, 268
service interface, 280
shell, 23, 66
significant, 210
simple signed integer, 62
Simulation, 41
Single Factor Optimization, 158
slash notation, 12
slice, 73, 75, 213
software stack, 242
Sorting Problem, 149
spams, 294
Spatial locality, 202
Specific Algorithm, 146
speedup, 249
spyware, 295
stable matching problem, 170
stack, 203, 228
standard, 270
Standard Error, 66
Standard Input, 66
Standard Output, 66
state circuit, 234
state transition, 79
statement, 21
state-transition diagram, 121
state-transition table, 122
Static networks, 274
Steps, 146, 147
Stored program, 76
stored program architecture, 76
stores, 221
Strassen's algorithm, 164
string, 68, 69
subtractor, 236
supercomputers, 202
sustained speed, 254
Switch, 268
Symbol-Manipulation System, 76
symmetric-key encryption, 295
synchronous sequential circuits, 239
System, 20
system heterogeneity, 205
system organization, 205
system scale, 205
system software, 242
system variation, 205
systematic, 199
Systems thinking, 31, 198

T

TCP/IP protocol stack, 278
Temporal locality, 202
text, 228
thorough, 199
Three Features Without, 25
Time complexity, 152
TLS, 298
topology, 274
transport layer, 278
Trojan horses, 295
Truth table, 100
Turing computable, 132
Turing machine, 126
Turing Machines, 120
Turing Test, 46
Turing's problem, 45
two's complement, 62
type cast, 70

U

uint8, 69, 214
UKA units, 34
undesirable, 295
Unicode, 207
Universality, 8
unsigned integer, 62
URL, 273
usage mode, 44
User Experience, 288
UTF-8, 213

V

variable, 68
vertex, 269
Viral marketing, 292
virtual things, 41
virtual worlds, 41
von Neumann architecture, 76
von Neumann bottleneck, 252
von Neumann computer, 79
von Neumann's Exhaustiveness Principle, 247
VPNs, 295

W

web address, 273

Web over Internet Stack, 276
web resource, 273
wide area network, 268
WiFi router, 302
Wonder of Cyberspace, 41
Wonder of exponentiation, 39
Wonder of Simulation, 41
Word, 58
word length, 34, 58
World Wide Web, 273
worst case, 177

Y

Yang's Cycle Principle, 244