CHAPTER 1

Introduction

1.1 Computers

A *computer* is an electronic device that accepts input, stores data, processes data according to a set of instructions (called program), and produces output in desired form. As shown in Figure 1.1, it can be abstracted as a black box that accepts input and produces output. The output depends on the current program in the block box.

Computer *input* is the information that is submitted to a computer by a human, by another computer, or by its environment. *Output* is the result produced by the computer, such as texts, audio, graphs, and pictures.

According to the Merriam-Webster Dictionary, data is factual information (as measurements or statistics) used as a basis for reasoning, discussion, or calculation[2]. In computer science, *data* is such factual information in a form suitable for use with a computer. Since a computer is an electronic device, the way to store data in the computer is in the form of electrical signals. These electrical signals typically have two states represented by 0 or 1. Thus, all data from outside a computer are transformed into a representation that uses 0 or 1 when stored or processed in the computer. The representation is transformed back to a desired form for output as a result of the computation.

Figure 1.1: Computer as a black box.

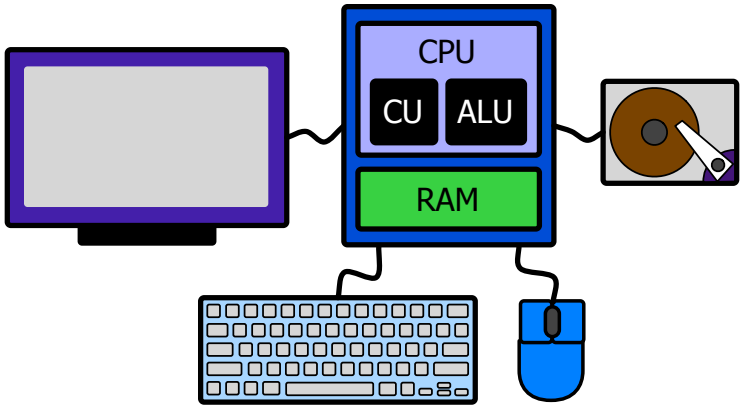

Figure 1.2: The von Neumann architecture.

A *bit* (a contraction of binary digit) is the fundamental unit of information in computer science that has two possible distinct values, 0 or 1. Inside a computer, data is encoded as patterns of bits (i.e., 0s and 1s). A *bit pattern* is a sequence (or string) of bits. The meaning of a bit pattern depends on the interpretation. Sometimes it is used to represent numeric values; sometimes it represents other symbols such as characters in an alphabet; sometimes an image or audio. A *byte* is another unit of information. It consists of 8 bits. Historically, a byte was used to encode a single character of text.

While early computers were built for specific problems and solved mathematical and engineering problems, modern computers are aimed at more general problems in a variety of domains. Today, computers are even embedded in automobiles, airplanes, robots, refrigerators, microwave ovens, mobile phones, MP3 players, etc.

A computer system consists of hardware and software. *Hardware* is the physical components of the system. *Software* is the set of programs that instructs the hardware to obtain the output. A computer *program* is a set of instructions telling the computer what to do with the input in order to produce the output. It controls the actions of a machine (i.e., computer). The task of writing a program for a computer is called *programming*. The person who writes the program is called a *programmer*.

1.2 The von Neumann Architecture

Most of modern computers follow the *von Neumann architecture*. The term is derived from a proposal (*First Draft of a Report on the EDVAC*[3]) by the early computer scientist John von Neumann and others in 1945. They proposed the idea of a general-purpose electronic computer with a *stored program*. Von Neumann is not the first person who proposed the stored program concept for

computers. There are other pioneers in computer science, such as Alan Turing, J. Presper Eckert, and John Mauchly, who proposed the same concept. However, von Neumann receives the principal credit because he instructed the rest of the world about it.

A computer that follows the von Neumann architecture consists of four basic hardware components (Figure 1.2): input devices, output devices, main memory, and central processing unit (CPU).

There are many different types of *input devices*, including keyboards, mice, hard disk drives, bar code readers, etc. They transmit information from the outside world into main memory. For example, when you press a key on a computer keyboard, you send a character to the main memory of the computer. The character is encoded in a sequence of electrical signals. Then, the signals are transmitted to and stored in main memory.

Output devices transmit information from main memory to the outside world. They include screens, printers, hard disk drives, etc. For example, a character stored in main memory is transmitted to a monitor in a sequence of electrical signals. The monitor decodes the signals to display the character on its screen.

Main memory stores both the program and the data being processed. Data can be both read and written in main memory as bit patterns and the location of data does not affect the access speed. This type of memory is often referred to as *RAM* (random-access memory). In addition, main memory is typically *volatile*. That is, when the power is turned off, the information stored in main memory is lost. Main memory does not distinguish the type of data stored. Programs or input/output devices are responsible for interpreting the stored bit pattern as a number, a alphabetical symbol, or some other type.

Machine code is the representation of a program that is actually read, interpreted, and executed by the computer. A program in machine code consists of a sequence of *machine instructions*. A machine instruction is represented as a finite bit string.

The CPU carries out the instructions of a computer program. Two major components of a CPU are the *arithmetic logic unit* (ALU) and the *control unit* (CU). The ALU performs arithmetic and logical operations, and the CU fetches instructions from main memory, decodes them, and executes them. The CU uses the ALU to execute the instructions when necessary.

Main memory is treated as *primary storage*. Computers have *secondary storage* (alternatively referred to as *auxiliary storage* or *external memory*) that is non-volatile. It does not lose data stored when the power is down. It is typically used as both input and output devices, and for storing programs and data. The computer usually accesses its secondary storage through an intermediate space in main memory.

In modern computers, hard disk drives or solid-state disk drives are used as secondary storage. The capacity of secondary storage is typically two orders of magnitude bigger than that of main memory. Some other examples of secondary storage are flash memory (e.g., USB sticks), floppy disks, magnetic tape, punched cards, and paper tape.

1.3 The Stored Program Concept

Early computers were designed to perform a specific computing task. They had fixed programs. To use them for a different task, it was necessary to rewire, restructure, or redesign the hardware. This requires human intervention. For example, ENIAC (Electronic Numerical Integrator And Computer) constructed by the University of Pennsylvania in 1946 was programmed by setting switches and modifying wiring to route data and control signals.

Conceptually, programs and data are very different. The stored program concept means that we treat programs as data, and they can both be stored in main memory. With this, the program for a specific task is loaded into main memory (e.g., from secondary storage), and instructions in the program are executed one after another without any human intervention. The program is easily replaced by another program for a different task when necessary. By storing a program in main memory, the hours of tedious labor required to reprogram computers can be eliminated. A modern computer can solve almost an infinite variety of problems by just switching between different programs.

1.4 Files

According to the Merriam-Webster Dictionary, a *file* is a complete collection of data treated by a computer as a unit especially for purposes of input and output[2]. Computer files can be considered as the counterpart of traditional files that are kept in offices. The primary purpose of a computer file is to store data in a more permanent form. Files are typically stored in a secondary storage device.

The contents of a file are encoded in a sequence of bits. The meaning of the bits totally depends on the interpretation by the program that accesses the file. In general, there are two kinds of computer files: *text files* and *binary files*. The contents of a text file are interpreted as character symbols. Other files than text files are binary files. A binary file contains any type of data encoded in bits. Text files are considered to be different from binary files in general because binary files contain more than just textual data, such as formatting information encoded in bits.

A character is encoded as a bit string in a text file. *ASCII* (American Standard Code for Information Interchange) developed by ANSI (American National Standards Institute) is the most common character coding scheme for English-language text files. ASCII uses 7 bits for each character symbol. Thus, 128 (2^7) different character symbols can be defined with ASCII. However, each byte in an ASCII text file contains a single character. Figure 1.3 shows the ASCII character table. ASCII was developed a long time ago and designed actually for use with teletypes. The first 32 non-printing characters are rarely used now.

Recently, the Unicode Consortium has developed a character coding scheme, called *Unicode*, to assign a unique value to every character symbol used in every language in the world. This allows texts from multiple languages to appear in

Dec	Bin	Char	Dec	Bin	Char	Dec	Bin	Char	Dec	Bin	Char
0	0000000	NUL	32	0100000	SPC	64	1000000	@	96	1100000	`
1	0000001	SOH	33	0100001	!	65	1000001	A	97	1100001	a
2	0000010	STX	34	0100010	"	66	1000010	B	98	1100010	b
3	0000011	ETX	35	0100011	#	67	1000011	C	99	1100011	c
4	0000100	EOT	36	0100100	$	68	1000100	D	100	1100100	d
5	0000101	ENQ	37	0100101	%	69	1000101	E	101	1100101	e
6	0000110	ACK	38	0100110	&	70	1000110	F	102	1100110	f
7	0000111	BEL	39	0100111	'	71	1000111	G	103	1100111	g
8	0001000	BS	40	0101000	(	72	1001000	H	104	1101000	h
9	0001001	TAB	41	0101001	)	73	1001001	I	105	1101001	i
10	0001010	LF	42	0101010	*	74	1001010	J	106	1101010	j
11	0001011	VT	43	0101011	+	75	1001011	K	107	1101011	k
12	0001100	FF	44	0101100	,	76	1001100	L	108	1101100	l
13	0001101	CR	45	0101101	-	77	1001101	M	109	1101101	m
14	0001110	SO	46	0101110	.	78	1001110	N	110	1101110	n
15	0001111	SI	47	0101111	/	79	1001111	O	111	1101111	o
16	0010000	DLE	48	0110000	0	80	1010000	P	112	1110000	p
17	0010001	DC1	49	0110001	1	81	1010001	Q	113	1110001	q
18	0010010	DC2	50	0110010	2	82	1010010	R	114	1110010	r
19	0010011	DC3	51	0110011	3	83	1010011	S	115	1110011	s
20	0010100	DC4	52	0110100	4	84	1010100	T	116	1110100	t
21	0010101	NAK	53	0110101	5	85	1010101	U	117	1110101	u
22	0010110	SYN	54	0110110	6	86	1010110	V	118	1110110	v
23	0010111	ETB	55	0110111	7	87	1010111	W	119	1110111	w
24	0011000	CAN	56	0111000	8	88	1011000	X	120	1111000	x
25	0011001	EM	57	0111001	9	89	1011001	Y	121	1111001	y
26	0011010	SUB	58	0111010	:	90	1011010	Z	122	1111010	z
27	0011011	ESC	59	0111011	;	91	1011011	[	123	1111011	{
28	0011100	FS	60	0111100	<	92	1011100	\	124	1111100	|
29	0011101	GS	61	0111101	=	93	1011101	]	125	1111101	}
30	0011110	RS	62	0111110	>	94	1011110	^	126	1111110	~
31	0011111	US	63	0111111	?	95	1011111	_	127	1111111	DEL

Figure 1.3: The ASCII table.

a single text file. A character symbol in Unicode uses 16 bits (2 bytes). Thus, Unicode can represent up to 65,536 (2^{16}) symbols. Different sections of Unicode are allocated to character symbols from different languages.

1.5 Operating Systems

Application software (also known as an *application*) is a set of programs that helps the user to carry out a specific task. For example, a *spreadsheet* is accounting software that helps the user to calculate numbers and organize information in a tabular form (e.g., columns and rows). In contrast, *system software* is a set of programs designed to operate the computer hardware and to provide a platform for running applications.

Especially, an *operating system* (also known as an *OS*) is system software that controls the operation of a computer and directs the processing of programs. It manages computer hardware resources and provides common services for application software. An application software ask a service to the operating system when necessary. The operating system assigns storage spaces in main memory, controls input and output functions, monitors the underlying computer system, etc. It is typical that a user cannot run an application on the computer without an operating system. Popular operating systems include Microsoft Windows, Mac OS, Linux, Unix, etc.

Utility software is system software designed to help the user manage and tune the computer hardware and software. It usually focuses on how the computer hardware and software operates. It is also referred to as *utility*, *tool*, and *service program*. Examples of utility software may include virus scanners, data compression utilities, disk partition utilities, archive utilities, system monitors, text editors, assemblers, etc.

1.6 Programming Languages

A *programming language* is a formal language in which computer programs are written. Most programmers write their programs in a *high-level programming language*, like Java, C, C++, FORTRAN, Scheme, ML, etc. The level of abstraction from the details of the underlying computer in a high-level programming language is higher than a *low-level programming language*. It is more close to natural languages and more understandable than a low-level language. Thus, a high-level programming language makes the process of developing programs simpler and easier.

An *assembly language* is a typical example of low-level programming languages. It represents machine instructions symbolically. The representation is defined by the hardware manufacturer and specific to a computer architecture. There exists an assembly instruction that corresponds to a machine instruction, but not *vice versa*. A program called an *assembler* is used to translate assembly language instructions into the target computer's machine code instructions.

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("hello, world\n'');
6 }
```

Figure 1.4: An example C program hello.

Figure 1.5: The compilation process.

The definition of a particular programming language consists of both *syntax* (how the various symbols of the language are combined) and *semantics* (the meaning of the language constructs). The syntax and semantics of a programming language are typically defined in its specification.

The *C* programming language was developed from 1969 to 1973 by Dennis Ritchie of Bell Laboratories. It was designed for a practical purpose to implement the UNIX operating system. As UNIX became popular, many software developers were exposed to C. Although it was originally designed as a systems programming language, it can be used for writing programs in a variety of different domains from business to engineering. It is one of the most widely used programming languages in these days.

In 1978, Brian Kernighan and Dennis Ritchie published the classical book on C, *The C Programming Language*[1]. This book was known to programmers as "K& R" and had served as an informal specification of C until 1989. In 1989, ANSI introduced C89 standard for the C programming language. The same standard was adopted by ISO (International Organization for Standardization) in 1990 and called C90. The standard was further revised, leading to ANSI/ISO C99 standard that was published in 1999. C99 is an internationally recognized C language specification, and almost all C compilers follow this standard. The standard promotes portability, reliability, maintainability, and efficient execution of C programs on a variety of machines.

Figure 1.4 shows an example C program hello. It is introduced in K&R, and prints hello, world on the screen when executed. The programmer creates the program with a *text editor* and saves it in a text file hello.c. The text file is stored in secondary storage. A *filename extension* c is used to indicate that the

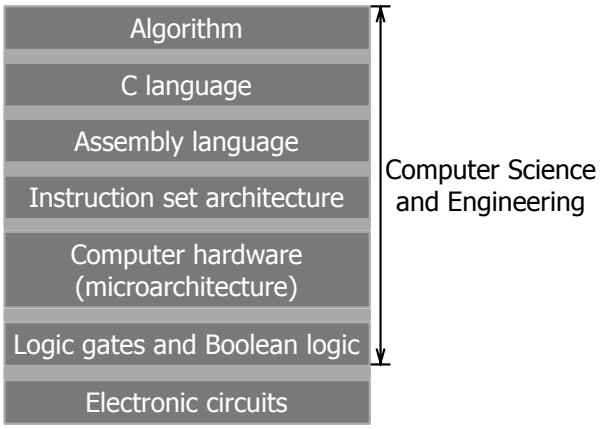

Figure 1.6: Abstractions in computer science and engineering.

text file contains a C program. A filename extension is a suffix to the filename (e.g., `c` that is separated from the base filename `hello` by a dot). It indicates the content type of the file. A text editor is a utility program to create and modify text files. Commonly used text editors include GNU emacs, UNIX vi, Microsoft word, etc.

To run the `hello` program on a computer, the C statements in the source program must be translated by a *compiler* into a sequence of machine instructions. A compiler is a program that automatically translates another program from some programming language to machine code. The resulting machine instructions are contained in a binary file called an *executable object file* (or simply *executable*). After compiling the source file `hello.c`, the resulting executable `hello` in Figure 1.5 is stored in secondary storage. It is ready to be loaded into the computer's main memory and executed.

1.7 Abstractions in Computer Science

Abstraction plays a key role in computer science. Computer science is fundamentally a science of abstraction. The concept of abstraction is pervasive in many arts and sciences. Abstraction is the process of considering the external properties of an object independently of its internal details. In other words, by abstracting an object, we are interested in "what the object does" without any interest in "how the object does it." To solve a complex problem, we devise an understandable model for it through abstraction. Then, we explore appropriate methods to solve it using the model. For example, the behavior of electronic circuits used to build computers can be modeled very well by logic gates and Boolean logic. Although this modeling is not exact, the model abstracts away many details, such as the behavior of transistors and electrical signals between

them.

We are able to design, analyze, and manage a complex computer system by applying abstractions. In this case, abstractions are built layer upon layer. As shown in Figure 1.6, the layer of abstraction starts from electronic circuits. Electronic circuits required to build a computer abstract away the requirements of the particular solid-state device technology (e.g., CMOS, NMOS, gallium arsenide, etc.) used to build the circuit. Then, as mentioned before, the behavior of the electronic circuits is modeled with logic gates and Boolean logic. At the next layer, each component of computer hardware (e.g., ALU, CU, and main memory) is implemented with logic gates and Boolean logic. There are various choices of hardware component implementations. They differ in performance, power consumption, and cost. However, at the computer hardware level, we do not worry about such implementation details.

An *instruction set architecture* (ISA) is the complete specification of the interface between the programmer and the underlying computer hardware. The ISA, not the computer hardware, is visible to the programmer when the programmer writes a program. An ISA is implemented by a *microarchitecture* that describes how the ISA should be implemented. Similar to the hardware component implementations with logic gates, microarchitectures have trade-offs between performance, power consumption, and cost. For example, the x86 ISA was originated by Intel and has been implemented by microprocessors from both Intel and AMD with radically different microarchitectures.

On top of the ISA, the assembly language provides one abstraction level. It implements a symbolic representation of machine instructions in the ISA to make the programming easier. An assembler is the interface between the assembly language and the ISA. The assembly language is also abstracted to the C language layer in a similar manner. The C compiler is the interface between the C language and the assembly language. A C program is translated into an assembly language program by the C compiler. High-level languages like C make writing a program simpler and easier than a low-level language, such as machine and assembly languages. Moreover, programs written in high-level languages can be transferred from one computer to another with little modification as long as the target computer has a compiler for the language.

An *algorithm* is a sequence of steps for carrying out a task or solving a problem. Each step is precisely and unambiguously specified and can be carried out by a computer. An algorithm is expressed formally as programs written in programming languages. As shown in Figure 1.6, an algorithm can be implemented with a C program.