CHAPTER 3

HOW INFORMATION IS STORED IN THE MEMORY OF A COMPUTER

Now, after having looked at a couple of computer programs, we know that it is possible to declare variables inside programs, and it is possible to store numerical values into the declared variables. When a program has been compiled and it is being run (executed) on a computer, the variables are in the memory of the computer, or we can say that memory space has been reserved to represent the variables of the program. Thus, the information that is stored in a variable is actually stored in the memory of the computer.

This chapter discusses how it is possible for the electronic circuitry of computers to store information, and how the memory of computers should be thought of by a programmer. A computer's memory can be considered a logical device which is built using electronic components. To understand how the memory in computers works, a programmer does not necessarily need to understand electronics or memory technology. It is enough to understand how the memory works in a logical sense.

In the following sections, we will study how different types of information (e.g. numbers, texts, and pictures) can be stored in the memory of a computer. The memory is such that it can hold information stored in a certain manner. The memory in a computer "remembers" what was written to it. The memory of a computers is, however, fundamentally different from human memory. Information stored in a computer's memory will be lost when some other information is written in the same place in the memory. Moreover, there is always a limit how much information a computer memory can hold.

3.1 Numerical information: numbering systems

The most typical way of showing numerical information is to use the ten familiar numerical symbols 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. With these symbols, it is possible to express any number. There are certain rules how to combine these symbols to form larger numbers. The numerical symbols are called digits when they appear in a larger number. For example, number 6378 has four digits, 6 is called the most significant digit, and 8 is the least significant digit.

Actually, it is quite amazing that with ten basic symbols it is possible to express a countless number of values. We could ask: why exactly ten different numerical symbols? Why not nine, or eleven, or three hundred, or one, or two? Probably the reason why we use ten different symbols for counting is that we have ten fingers and ten toes. Our counting system was invented by people who lived a long time ago, and they probably ended up in a ten-symbol system after counting with their fingers. In any case, it is possible to formulate a working numbering system for nearly any number of numerical symbols. We shall see that computers use only two numerical symbols to represent and store numerical information. To study different numbering systems, we must first take a closer look at our commonly used numbers, the Arabic numbers.

With the ten different symbols, we can basically express only ten different quantities, but our minds are accustomed to combine the ten numerical symbols in a sophisticated manner. When we want to express larger numbers, we start thinking in terms of times of ten. For example, the number 6378 is:

6 times 10 times 10 times 10 plus
3 times 10 times 10 plus
7 times 10 plus
8

By deciding that * means "times" (the multiplication operation), that + means "plus" (the addition operation), and that multiplication operations are carried out before additions in mathematics, we can express the number 6378 in a more mathematical way with the following expression

$$6 * 10 * 10 * 10 + 3 * 10 * 10 + 7 * 10 + 8$$

Furthermore, by deciding that 10^0 is 1, 10^1 is 10, 10^2 is 10 * 10, 10^3 is 10 * 10 * 10, etc., we can write the above expression even more elegantly, as follows:

$$6 * 10^3 + 3 * 10^2 + 7 * 10^1 + 8 * 10^0$$

In the mathematical expression above, it is important to note that ten to the power of zero is considered to be one. In mathematics, any number to the power of zero is one. It is possible to express any number in our commonly used numbering system in the same way as the number 6378 above. For example, the number 285024 can be expressed as

$$2 * 10^5 + 8 * 10^4 + 5 * 10^3 + 0 * 10^2 + 2 * 10^1 + 4 * 10^0$$

Because the Arabic numbering system has ten different symbols (0, 1, 2, 3, 4, 5, 6, 7, 8, and 9) to express numerical information, the number 10 is an important number in the mathematical expressions above. As we have ten different numerical symbols in use, our numbering system is said to be the base-10 numbering system.

The advantage of having such a numbering system as our base-10 system is that we can concisely write down numerical information. We can understand how much is 299

without thinking that it is actually $2 * 10^2 + 9 * 10^1 + 9 * 10^0$. We are so accustomed to using the base-10 system that in everyday life we do not need any other numbering systems. However, to understand how information is stored in the memory of a computer, and how computers process information, it is necessary to also know and understand other numbering systems.

It is possible to formulate new numbering systems from the base-10 system by removing the most significant numerical symbols from it. For example, we get a base-9 numbering system by leaving the symbol 9 out from the base-10 system. When we leave both the symbols 9 and 8 out from the base-10 system, we get a base-8 numbering system which is also called the octal numbering system. The most minimal numbering system is the base-2 system which contains only two numerical symbols, 0 and 1. That is called the binary numbering system. Another possibility to formulate new numbering systems is to add new numerical symbols to the symbols of the base-10 system. For example, we get a base-16 system by introducing six new numerical symbols A, B, C, D, E, and F to the symbols of the base-10 system. The base-16 system is called the hexadecimal numbering system.

Table 3-1 shows some numbers written down in different numbering systems. The essential difference between different numbering systems is the number of numerical symbols in use. By studying Table 3-1, you can see that whenever we count upwards and run out of symbols (i.e. reach symbol 9 in the base-10 system, reach symbol 7 in the base-8 system, reach symbol 1 in the base-2 system, or reach symbol F in the base-16 system) we start to use a new column of symbols, or we can say that we start to use a new digit which has more significance. In all numbers, in all numbering systems, the column to the left bears symbols of greater value, or more significant digits, and the rightmost digit of a number is the least significant. All the numbering systems are basically similar. The only fundamental difference is how many numerical symbols are in use.

Roman numbers

Our normal numbers are called Arabic numbers because they were introduced to Western Europe by Arabs who had learned them in India. The Arabic numbering system and the numbering systems that can be derived from it are not the only numbering systems in use these days. For example at the end of movies, the year when the movie was made is often expressed with a so-called Roman number. The letters MCMXCVI at the end of a movie mean that the movie was made in the year 1996. In the Roman numbering system, certain letters are used to denote certain numerical quantities, and there are rules how the numerical letters can be combined to form bigger numbers. The basic meanings of the numerical letters are the following:

I	one	V	five
X	ten	L	fifty
C	one hundred	D	five hundred
M	one thousand		

The letters I, X, C, and M can be combined so that two or three consecutive letters represent two or three times the value of that symbol (e.g. XX means twenty, XXX means thirty, C means one hundred, and CC means two hundred). When a lower-valued letter precedes a higher-valued letter, the numerical meaning of the higher-valued letter is reduced (e.g. XC means ninety, CM means nine hundred, IX means nine, and CD means four hundred). Full-valued letter combinations and letters with decreased values can be combined to express various quantities. In the Roman system, the numbers from one to twenty are I, II, III, IV, V, VI, VII, VIII, IX, X, XI, XII, XIII, XIV, XV, XVI, XVII, XVIII, XIX, and XX; 1234 is MCCXLVII; 333 is CCCXXXIII; and 444 is CDXLIV. It must be noted, that there is no quantity "zero" in the Roman numbering system. For that reason, it is not mathematically convenient and therefore it has been replaced by the Arabic system. When you learn more while reading this book, try to write a program that can convert from Roman numbers to Arabic ones and vice versa.

Table 3-1: Numbers expressed in different numbering systems.

Base-10 (decimal numbers)	Base-8 (octal numbers)	Base-2 (binary numbers)	Base-16 (hexadecimal numbers)
0	0	0	0
1	1	1	1
2	2	10	2
3	3	11	3
4	4	100	4
5	5	101	5
6	6	110	6
7	7	111	7
8	10	1000	8
9	11	1001	9
10	12	1010	A
11	13	1011	B
12	14	1100	C
13	15	1101	D
14	16	1110	E
15	17	1111	F
16	20	10000	10
17	21	10001	11
18	22	10010	12
19	23	10011	13
20	24	10100	14
21	25	10101	15
22	26	10110	16
23	27	10111	17
24	30	11000	18
25	31	11001	19
26	32	11010	1A
27	33	11011	1B
28	34	11100	1C
29	35	11101	1D
30	36	11110	1E
31	37	11111	1F
32	40	100000	20
33	41	100001	21
34	42	100010	22
35	43	100011	23
36	44	100100	24
37	45	100101	25
38	46	100110	26
39	47	100111	27
40	50	101000	28
41	51	101001	29
42	52	101010	2A
43	53	101011	2B
44	54	101100	2C
45	55	101101	2D

Table 3-2: Important numbers in computing.

2^n	Decimal	Hex	$16^{(n/4)}$	"slang"
2^0	1	1H	16^0	
2^1	2	2H		
2^2	4	4H		
2^3	8	8H		
2^4	16	10H	16^1	
2^5	32	20H		
2^6	64	40H		
2^7	128	80H		
2^8	256	100H	16^2	
2^9	512	200H		
2^{10}	1024	400H		1 k
2^{11}	2048	800H		2 k
2^{12}	4096	1000H	16^3	4 k
2^{13}	8192	2000H		8 k
2^{14}	16384	4000H		16 k
2^{15}	32768	8000H		32 k
2^{16}	65536	10000H	16^4	64 k
2^{17}	131072	20000H		128 k
2^{18}	262144	40000H		256 k
2^{19}	524288	80000H		512 k
2^{20}	1048576	100000H	16^5	1 M
2^{21}	2097152	200000H		2 M
2^{22}	4194304	400000H		4 M
2^{23}	8388608	800000H		8 M
2^{24}	16777216	1000000H	16^6	16 M

Numbers in different numbering systems can be expressed using mathematical expressions. For example, the base-8 number 40572 can be expressed as

$$4 * 8^4 + 0 * 8^3 + 5 * 8^2 + 7 * 8^1 + 2 * 8^0$$

The base-2 number 10110101B can be expressed as

$$1 * 2^7 + 0 * 2^6 + 1 * 2^5 + 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0$$

The base-16 number 85ADFH can be expressed as

$$8 * 16^4 + 5 * 16^3 + A * 16^2 + D * 16^1 + F * 16^0$$

The base number of the numbering system makes all the difference in the expressions above. Remember that * means a multiplication operation. Note also the convention of using the letter B at the end of a binary number to show that it is a binary number. Similarly, the letter H at the end of a hexadecimal number is used to denote that the number is of the hexadecimal numbering system. For base-8 octal numbers we do not have any special letters, because we do not use base-8 often in this book. By using letters at the end of numbers other than base-10 decimal numbers, we can write all numbers down without the possibility of misunderstanding. For example, numbers 16H, 10110B, and 22 mean the same.

The binary numbering system and the hexadecimal numbering system are important in the world of computers. It is often necessary to make conversions between these numbering systems and our base-10 decimal system. To make the conversions easily, you should buy a calculator which is capable of operating with binary and hexadecimal numbers. However, a computer specialist must be able to make the conversions by hand, if necessary. At this phase of your becoming a computer specialist, it is good practice to learn to make the conversions that will be explained below.

Conversions to the decimal system can be made by writing expressions like the ones above. For example, 101011B can be converted to a decimal number in the following way

$$\begin{aligned} 101011B &= 1 * 2^5 + 0 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0 \\ &= 32 + 0 + 8 + 0 + 2 + 1 \\ &= 43. \end{aligned}$$

Table 3-2 shows how much 2^0, 2^1, 2^2, 2^3, etc. is in the decimal system. That table is therefore useful when making conversions like the one above. Converting a binary number to the decimal system is basically a matter of taking the correct numbers from the second column of Table 3-2 and calculating the sum of these numbers.

Converting a hexadecimal number to the decimal system resembles the conversion of binary numbers. For example, 3AF2H can be converted to a decimal number in the following way

$$\begin{aligned} 3AF2H &= 3 * 16^3 + A * 16^2 + F * 16^1 + 2 * 16^0 \\ &= 3 * 4096 + 10 * 256 + 15 * 16 + 2 * 1 \\ &= 12288 + 2560 + 240 + 2 \\ &= 15090. \end{aligned}$$

In these kinds of conversions you can also exploit the important numbers shown in Table 3-2. And always remember that any number to the power of zero is one.

Conversions from the decimal system to the binary numbering system can be carried out by performing subtraction operations with the "magical numbers" in the second col-

umn of Table 3-2. Those numbers are somewhat magical because, provided that new rows are added to the table if necessary, any whole number (integer) can be expressed as the sum of a set of numbers from the second column of the table. For example, let's convert the number 2841 to the binary system. Beforehand, we know that the result of the conversion will be a series of 1s and 0s. We also know that the result begins with a 1 because leading zeroes in numbers are insignificant (e.g. 001101B is the same as 1101B). The decimal-to-binary conversion procedure is the following:

- Search for the largest number from the second column in Table 3-2 which can be subtracted from the number being converted. In the case of 2841, 2048 is the largest number that can be subtracted. Because 2048 is 2 to the power of 11, we can deduce that the binary number we are trying to construct has a 1 in the exponent position 11, and the binary number has 12 digits. The number thus looks like

  2^{11} 2^{10} 2^9 2^8 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0

  1 x x x x x x x x x x x

  Now the rest of our task is to find out whether there is 0 or 1 in place of each x.
- Subtract the number found in the second column from the number being converted. In our example case 2841 - 2048 makes 793.
- Moving upwards from the position found in Table 3-2, find the largest number that can be subtracted from what is left from the original decimal number. In our example case we go upwards from the exponent position 11 in the table, and search for a number in the second column that can be subtracted from 793. We can see that 1024 in the exponent position 10 cannot be subtracted, but 512 in the exponent position 9 can be subtracted from 793. This means that there is a 0 in the exponent position 10 and a 1 in the exponent position 9, and the binary number now looks like

  2^{11} 2^{10} 2^9 2^8 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0

  1 0 1 x x x x x x x x x

- Subtract the number found in the second column from what is left from the original number. By calculating 793 - 512 we get 281.
- Continue going upwards in the second column of Table 3-2, searching for the largest numbers that can be subtracted from what is currently left from the original decimal number. Mark a 1 in those exponent positions where subtraction is possible and carry out the subtraction. Mark a 0 to the exponent positions where no subtraction is possible. Stop the procedure when there is nothing left from the original number, and mark a 0 to any remaining unsolved exponent positions. In our example case we are searching for a number that can be subtracted from 281. The procedure goes as follows

256	can subtract	result is 25	binary digit is 1
128	cannot subtract		binary digit is 0
64	cannot subtract		binary digit is 0
32	cannot subtract		binary digit is 0
16	can subtract	result is 9	binary digit is 1
8	can subtract	result is 1	binary digit is 1
4	cannot subtract		binary digit is 0
2	cannot subtract		binary digit is 0
1	can subtract	result is 0	binary digit is 1

The last binary digit, the least significant bit, in our example was 1 because it was an odd number that was being converted. The final result of our conversion example is

2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	1	1	0	0	0	1	1	0	0	1

Making a decimal-to-binary conversion needs a little bit of work if you do it by hand (and brains), but converting between hexadecimal and binary numbers is an easier thing to do. One reason why hexadecimal numbers are loved so much by computing specialists is that it is very easy to convert a binary number to a hexadecimal number and vice versa. The relationship between binary numbers and hexadecimal numbers is such that four bits (binary digits) in a binary number correspond directly to a single digit in the same number in the hexadecimal form. Therefore, a binary number can be converted to a hexadecimal number in groups of four bits. Figure 3-1 shows how binary number 10101101101100B can be converted to a hexadecimal number. By applying the procedure described in Figure 3-1 in the reverse sense, it is possible to make hexadecimal-to-binary conversions. For example hexadecimal number 6F1DH is 0110111100011101B because 6H is 0110B, FH is 1111B, 1H is 0001B, and DH is 1101B.

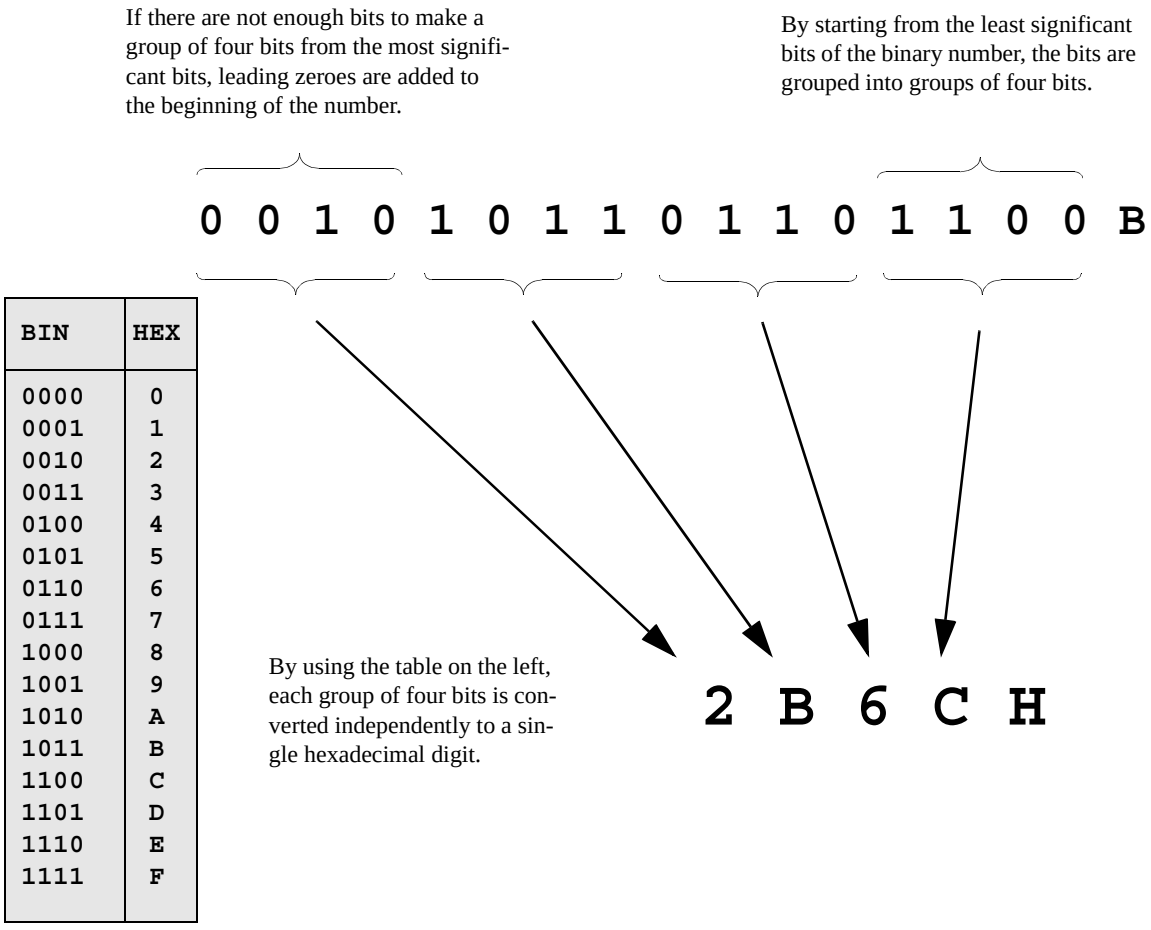

Figure 3-1. Converting 10101101101100B to a hexadecimal number.

The "slang" column of Table 3-2 may require some explanation. When speaking about things in the world of computers, we often need to speak about large numbers. It is a nice coincidence of the binary numbering system and the decimal numbering system that numbers 1024 and 1000, as well as numbers 1048576 and 1000000 are so close to each other. On the other hand, the word kilo (k) means 1000 times something and Mega (M) means 1000000 times something. Outside the world of computers it is usual to say 1 kilometer to mean 1000 meters, or 1 Megaton to mean 1000000 tons. Therefore, it has become customary to say, for example,

64 kilobits (kb) to mean 65 536 bits,
1 Megabit (Mb) to mean 1 048 576 bits, and
8 Megabytes (MB) to mean 8 388 608 bytes.

Conventions for writing hexadecimal numbers

Hexadecimal numbers are expressed by using the normal numerical symbols 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9, and six additional numerical symbols A, B, C, D, E, and F. To indicate that a number is a hexadecimal number, it is common to add the letter H at the end of the number. These are not, however, the only conventions. Another commonly used notation is to use the prefix 0x, a zero and letter x, to indicate that a number is a hexadecimal number. The additional hexadecimal symbols may also be written as lowercase letters a, b, c, d, e, and f. Thus we have the following four different possibilities to write hexadecimal numbers:

DECIMAL	HEXADECIMAL	HEXADECIMAL	HEXADECIMAL	HEXADECIMAL
22	16H	16H	0x16	0x16
31	1FH	1fH	0x1F	0x1f
254	FEH	feH	0xFE	0xfe
15090	3AF2H	3af2H	0x3AF2	0x3af2
55236	D7C4H	d7c4H	0xD7C4	0xd7c4

Exercises related to numbering systems

In the exercises below you should do the conversions manually as described in this section. You can verify your answers with a calculator which is capable of handling binary and hexadecimal numbers.

Exercise 3-1. Convert the binary numbers 10111010B and 101010110110B to decimal numbers and hexadecimal numbers.

Exercise 3-2. Convert the hexadecimal numbers AF21H and B29DH to binary numbers and decimal numbers.

Exercise 3-3. Convert the decimal numbers 1234 and 5678 first to binary numbers. Convert then the binary numbers to hexadecimal numbers.

3.2 Numerical information: the binary world of computers

Now you have learned that numerical information can be expressed in different ways. We are accustomed to using decimal numbers in our everyday life, but it is the binary numbering system that is THE numbering system in the world of computers. The binary numbers can be written with only two symbols, 0 and 1, but they are equally as adequate numbers as our common decimal numbers. Everything that can be written down as a decimal number can also be expressed as a binary number.

Binary numbers are convenient for computers because they need only two symbols. Binary numbers can be stored in the memory of computers in the form of electric phenomena. For example, a voltage present in a certain part of an electronic component can mean the binary 1. No voltage present can then mean a binary 0, and thereby we have all binary symbols expressed in electric form.

The most important electronic components inside modern computers are integrated circuits, the black components on greenish boards. Integrated circuits contain many transistors that are connected to each other in a special way. Transistors are the basic electronic elements inside integrated circuits. A single integrated circuit may contain thousands if not millions of transistors. The transistors inside integrated circuits are used to store information in binary form. By setting a voltage to a certain wire it is possible to store binary information into an integrated circuit, and by setting a voltage to another wire, it is possible to read the previously stored binary information.

Although computers are rather complex electronic constructions, a person who wants to write programs to be run on computers does not have to understand all the electronic details of computers. A programmer needs merely a logical view of a computer's electronics. A computer can be considered a device that contains very many logical memory cells which are able to store one bit of information. The memory cells are made of transistors. A bit (binary digit) is the smallest unit of information inside a computer. A memory cell which can hold a bit of information can contain a zero (0) or one (1). The key idea in electronic computing is that, although information is stored in small bits, it is possible to handle large amounts of information when there are very many of these single-bit memory cells.

Figure 3-2 shows a simple memory cell which is capable of storing one bit of information. The memory cell operates with a voltage of +5V and it has lines (wires) for writing and reading information. The information that can be stored is either 0 or 1. We can assume that zero Volts means 0 and +5 Volts means 1. The memory cell is capable of holding the voltage that has been stored in it, and it simply outputs a voltage of 0V or +5V depending on which of these two voltages has been stored in the memory cell. Information can be stored in the memory cell by switching a voltage of +5V to the WRITE MEMORY line and simultaneously setting the INPUT line to the voltage that represents the information which is being stored. Information can be read from the memory cell by switching a voltage of +5V to the READ MEMORY line. As long as the READ MEMORY line has an active voltage of +5V, the OUTPUT line has the voltage (0V or +5V) that has previously been stored in the memory cell. The lines WRITE MEMORY and READ MEMORY are control signals which are used to transfer information to or from the memory cell. The actual data transfer occurs via the INPUT and OUTPUT lines. The GROUND line is the basis for which all voltages are measured. The line that connects the memory cell to the operating voltage is also marked in Figure 3-2, though that line does not affect the logical operation of the cell.

Figure 3-3 shows a timing diagram that describes the operation of the single-bit memory cell. It is assumed that the memory cell is somewhere among the other electronic circuitry and the outside circuitry changes the voltages on the lines that are connected to the memory cell. Note that the line OUTPUT has a defined voltage only when the line READ MEMORY has a voltage with which the memory cell is ordered to deliver its contents to the outside world.

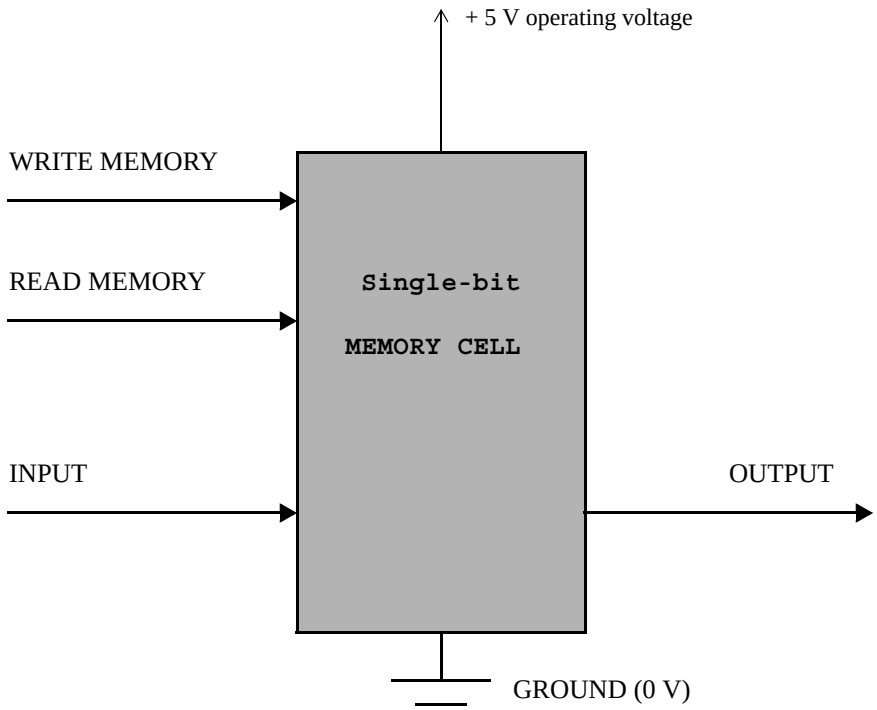

Figure 3-2. A logical model of a single-bit memory cell.

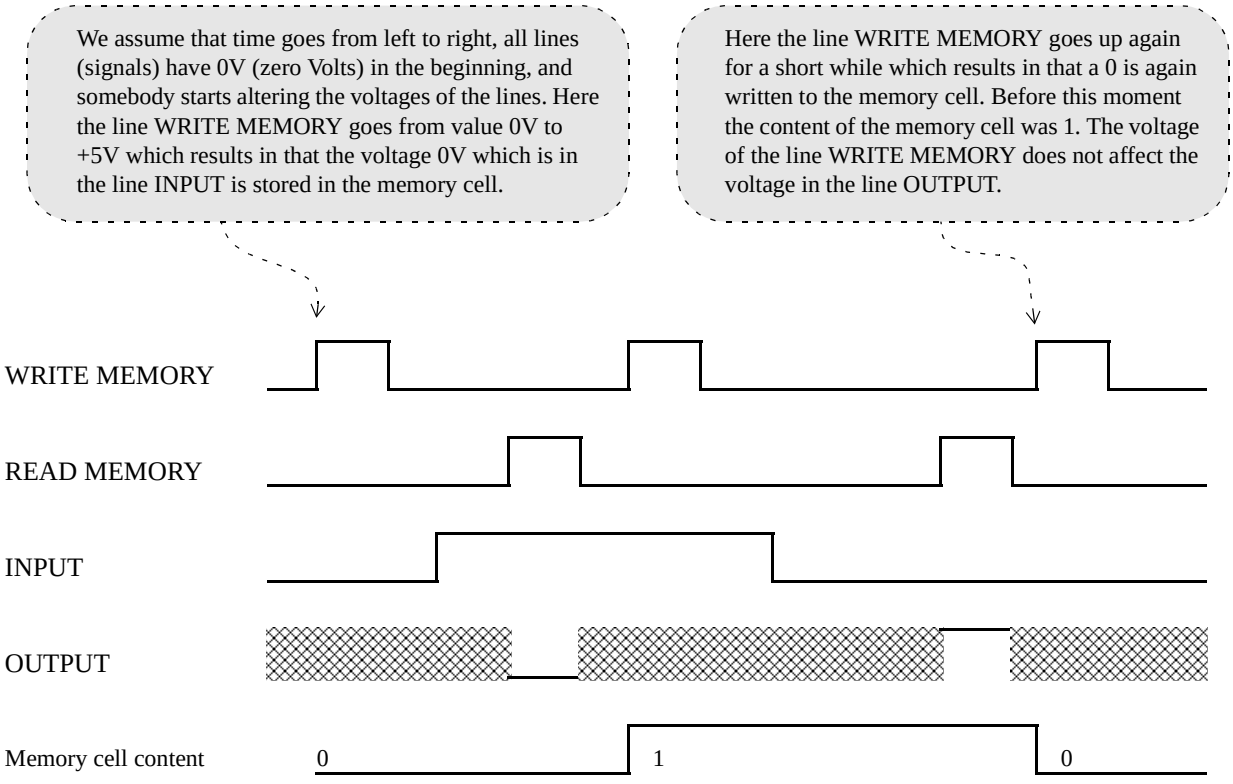

Figure 3-3. Writing and reading a single-bit memory cell.

Memory cells are called flip-flops in the literature of digital electronics. The memory cells that exist in the integrated circuits in computers are not necessarily exactly like the logical model in Figure 3-2 where 0V means the binary number 0 and +5V means the binary number 1. In practice, the electronic circuits can be constructed so that zero voltage means 1 and non-zero voltage means 0. Practical memory cells may also operate with different voltages and control signals may be different. However, it is most important to know how a memory cell of a computer operates in the logical sense, and that is shown in figures 3-2 and 3-3.

Although memory cells such as the one in Figure 3-2. cannot store more than one bit of information (0 or 1), these kinds of memory cells are useful in computer electronics when we connect many of these memory cells together. Present technology allows millions of these single-bit memory cells to be constructed on a single integrated circuit chip. Figure 3-4 shows the principle how single bit memory cells can be connected together. Figure 3-4 has eight input lines (INPUT7, INPUT6, ... , INPUT1, and INPUT0) which can be activated or controlled with a single WRITE MEMORY signal, and it has eight output lines (OUTPUT7, OUTPUT6, ... , OUTPUT1, and OUTPUT0) which are controlled with only one READ MEMORY signal. Each of the eight memory cells in Figure 3-4 works in the same way as the memory cell in Figure 3-2. We could, for example, connect the following voltages to the electric lines in Figure 3-4:

INPUT7	INPUT6	INPUT5	INPUT4	INPUT3	INPUT2	INPUT1	INPUT0	WRITE MEMORY
0V	+5V	+5V	0V	+5V	+5V	0V	+5V	+5V

After setting the input lines and line WRITE MEMORY to these voltages, the voltages in the input lines would be stored in the eight memory cells. Supposing that +5V means 1 and 0V means 0, we can say that by having stored the above voltages, we have actually stored the binary number 01101101B into the memory cells. The key idea here is that numeric information is stored in the form of electronic voltages, and the input lines correspond to bits in a binary number. The numbers of the input lines in Figure 3-4 are the exponents needed in the conversions of binary numbers. With the eight memory cells in Figure 3-4 it is possible to store decimal numbers from 0 to 255 by setting the input voltages in the following way:

INPUT 7	INPUT 6	INPUT 5	INPUT 4	INPUT 3	INPUT 2	INPUT 1	INPUT 0	BINARY NUMBER	DECIMAL NUMBER
0V	OV	0V	OV	0V	OV	0V	OV	00000000	0
0V	0V	0V	0V	0V	0V	0V	+5V	00000001	1
0V	0V	0V	0V	0V	0V	+5V	0V	00000010	2
0V	0V	0V	0V	0V	0V	+5V	+5V	00000011	3
0V	0V	0V	0V	0V	+5V	0V	0V	00000100	4
0V	0V	0V	0V	0V	+5V	0V	+5V	00000101	5
0V	0V	0V	0V	0V	+5V	+5V	0V	00000110	6
0V	0V	0V	0V	0V	+5V	+5V	+5V	00000111	7
	.....							...	..
+5V	+5V	+5V	+5V	+5V	0V	+5V	+5V	11111011	251
+5V	+5V	+5V	+5V	+5V	+5V	0V	0V	11111100	252
+5V	+5V	+5V	+5V	+5V	+5V	0V	+5V	11111101	253
+5V	+5V	+5V	+5V	+5V	+5V	+5V	0V	11111110	254
+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V	11111111	255

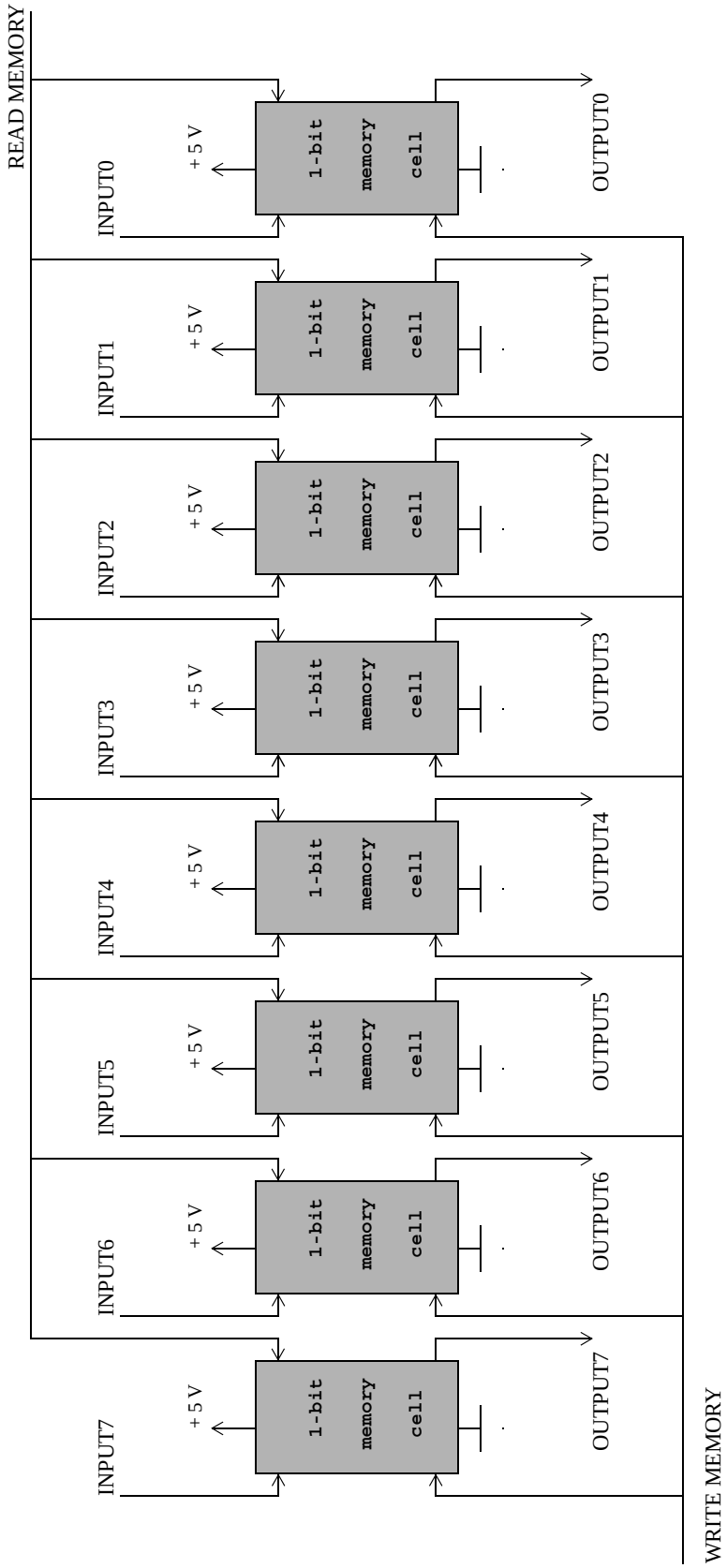

Figure 3-4. Eight single-bit memory cells connected in parallel.

The configuration in Figure 3-4 can be called an 8-bit memory cell which means that 8 bits of binary information can be stored in it. With 8 bits it is possible to express 256 (2 to the power of 8) different numbers (i.e. the numbers from 0 to 255). It is easy to imagine that even larger numbers than 255 could be stored in the memories of computers by connecting more single-bit memory cells in parallel. For example, when 16 single-bit memory cells are connected in parallel, we get a 16-bit memory cell which allows the numbers from 0 to 65535 to be stored. The decimal number 65535 is 1111111111111111B (sixteen ones). Into a 16-bit memory cell it is possible to store 16 bits of information, and 65536 different numbers can be expressed with 16 bits. The largest number that can be stored in a memory cell is 2 to the power of the width of the memory cell minus one. Therefore

- the largest number in a single-bit memory cell is $2^1 - 1 = 1$,
- the largest number in a 8-bit memory cell is $2^8 - 1 = 255$, and
- the largest number in a 16-bit memory cell is $2^{16} - 1 = 65535$.

In computing, eight bits of information is called a byte. The memory cell shown in Figure 3-4 is thus a one-byte memory cell. A byte is an important unit of information. Bytes are used to describe how much information can be stored in the memory of computers. For example, an old advertisement of computers might have said that "these computers contain 8 MB of main memory". Eight Megabytes (MB) means that 8 388 608 bytes of information can be stored in this type of memory. 8 Megabytes is 8 x 8 388 608 = 67 108 864 bits, which means that this kind of memory consists of 67 108 864 single-bit memory cells, such as the one in Figure 3-2.

When we write computer programs, we use one-byte memory cells like the one in Figure 3-4. Programming languages provide mechanisms for how memory can be reserved and used within computer programs. For example, the C# source program line

```
int  some_integer_variable ;
```

declares a variable named **some_integer_variable** and reserves four bytes (32 bits) of memory for the variable. The four bytes are treated like a single 32-bit memory cell which can hold an integer value, and the variable name **some_integer_variable** can be used to refer to the 32-bit memory cell. The C# program statement

```
some_integer_variable  =  88 ;
```

writes the integer value 88 to the 32-bit memory cell. We can use these kinds of statements in a program without knowing exactly where those four bytes, the 32-bit memory cell, are located in the computer's main memory. All we need to know is that those four bytes are really reserved, and they can hold a value that is written to them.

The memory cells, whose logical operation is described above, are used, for example, to construct the main memory of a computer. There are also other technologies to construct computer memory. These include magnetic memory like a hard disk and optical memory like that of CDs. Regardless of the technology on which a memory device is based, the mechanism for storing information is the same: information can be stored only as bits, zeroes and ones. The memory cell technology is, though, the most important in the operation of a computer. When you understand the logical operation of a memory cell, it will be easier to understand the logical operation of a computer in Chapter 4.

3.3 Textual information: character coding systems

Information is stored in the memory of computers as zeroes and ones. We learned this in the previous section, and, in fact, this is the most important thing to be learned about how computers store information. There is nothing else but zeroes and ones, represented by two voltage levels, in the memory of computers. Zeroes and ones are convenient when we want to store numerical information, but these two digits can also be used to store other types of information, such as various texts.

When we want to store a number, say 123, in a computer's memory, we can store it in binary form. 123 is 1111011B as a binary number. Since 1111011B consists of seven binary digits, a one-byte memory cell such as the one in Figure 3-4 is sufficient for storing this number. When larger numbers need to be stored, several bytes of memory can be used.

It is not difficult to store whole numbers in a computer memory since these numbers can be expressed clearly in binary form. But when we want to store text in a computer’s memory, the situation becomes somewhat more complicated. Because computer memories can only store zeroes and ones, there has to be a way to code textual information to zeroes and ones. A traditional coding system for textual information is the ASCII coding system. ASCII is an acronym of "American Standard Code for Information Interchange". The ASCII coding system is an agreement made by organizations working in the computing business. As its name implies, the coding system was developed in the United States where the commercial use of computers began.

There are ASCII character codes for all those textual symbols that can be found on the keyboard of a computer. The ASCII coding system is based on the idea that each letter or symbol must fit in one byte of memory. Because one byte is 8 bits, there can be 256 different character codes. 256 different codes is sufficient to represent all uppercase English letters (A, B, C, etc.), all lowercase English letters (a, b, c, etc.), all decimal numerical symbols (0, 1, 2, 3, 4, 5, 6, 7, 8, and 9), many special printable characters (!, ", #, $, %, &, ', (,), -, /, etc.), and many non-English letters and special symbols (Ä, Ö, å, etc.).

At the end of this book on page 600 you can find a table which lists the first 128 most commonly used ASCII character codes. You can see, for example, that the code for the letter A is 41H (0100 0001 binary, 65 decimal), and the code for the plus sign + is 2BH (0010 1011 binary, 43 decimal). By carefully studying the table you can find out the following facts about the ASCII coding system:

- The codes from 30H to 39H are reserved for numerical symbols from 0 to 9.
- The codes from 41H to 5AH are reserved for uppercase letters from A to Z.
- The codes from 61H to 7AH are reserved for lowercase letters from a to z.
- You get the code for a lowercase letter by adding 20H to the code of the corresponding uppercase letter. For example, because the character code of uppercase letter R is 52H, the character code of lowercase letter r is 52H + 20H = 72H.
- You get the character code of a numerical symbol by adding 30H to the number in question. For example, the character code of 8 is 30H + 8 = 38H
- The first codes from 0 to 1FH are not printable or visible characters. Instead, they represent special control characters. Some of the special control characters like BACKSPACE, TABULATOR, NEWLINE, and ESCAPE can be produced by computer keyboards. BELL is a special "character" that represent a sound that computers can produce. A special "character" is also NULL which has code 0. The NULL character has its own name so that it can be clearly distinguished from the character code of zero which is 30H.
- You cannot find character codes for non-English letters such as Ä, Ö, Å, Ñ, É, etc. in the table. The codes of these letters are in the range from 80H to FFH, and there exist different code tables for this code range.

Once you have a character code chart like the table on page 600 available, it is easy to convert textual information to character codes. For example, the word HELLO is coded in the following way:

```
48H  45H  4CH  4CH  4FH    ( hexadecimal codes )

72   69   76   76   79     ( decimal codes )

H    E    L    L    O
```

The sentence "Computing is fun." is coded with the following sequence of numbers:

```
43H 6FH 6DH 70H 75H 74H 69H 6EH 67H 20H 69H 73H 20H 66H 75H 6EH 2EH

67  111 109 112 117 116 105 110 103 32  105 115 32  102 117 110 46

'C' 'o' 'm' 'p' 'u' 't' 'i' 'n' 'g' ' ' 'i' 's' ' ' 'f' 'u' 'n' '.'
```

Note that spaces between the words in the sentence above have character codes (20H), and the full stop '.' at the end of the sentence has the code 2EH. Storing the text "Computing is fun." requires 17 bytes of memory: the letters require 14 bytes altogether, two spaces require two bytes, and the full stop needs one byte.

It is possible to express numbers with character codes. For example, number 123 could be coded as follows

```
31H   32H   33H    ( hexadecimal codes )

49    50    51     ( decimal codes )

'1'   '2'   '3'
```

Storing the number 123 coded with character codes would require 3 bytes of memory. If the number was stored as a binary number, it could be stored in a single byte of memory

ASCII coding is widespread in computing and telecommunications. For example, when you send an e-mail message on the Internet, the text is converted to character codes which are sent in electronic form. When you press some key on the keyboard of your computer, the program which is being run by the computer receives the character code of the key that was pressed. Even source programs are sequences of character codes in a file on a hard disk of a computer.

One problem with the ASCII coding system is that only the codes from 0 to 7FH (0 to 127) are the same for most computer operating systems. The codes from 80H to FFH (128 to 255) can have different meanings in different operating systems.

A further limitation of the ASCII coding system is that it defines only 256 different codes. Those codes represent letters and other symbols used in English and Western-European languages. There are many natural languages which use characters and symbols which cannot be expressed with the ASCII coding system. Because this kind of limitation causes problems, a new and more universal coding system for textual information has been developed. The name of the new system is Unicode, and it uses 16 bits to code each textual symbol. With 16 bits it is theoretically possible to code 65536 different symbols.

The Unicode system has codes for the characters of many natural languages. Each character set has been reserved a certain range of character codes. The following are examples of hexadecimal code ranges

0370 ... 03FF	Greek and Coptic characters
0400 ... 04FF	Cyrillic characters
0530 ... 058F	Armenian characters
0590 ... 05FF	Hebrew characters
0600 ... 06FF	Arabic characters

Because the Unicode system attempts to cover the characters of all natural languages, it is impossible to show all the character codes and the actual characters here. To see the character code tables and the actual characters, please visit the Internet address www.unicode.org.

The first 256 character codes in the Unicode system are the same as the codes of the standardized ASCII coding system. The following list compares Unicode character codes to ASCII character codes (Note that all codes are expressed as hexadecimal codes):

```
CHARACTER    Unicode    ASCII

 NULL        0000       00
 BACKSPACE   0008       08
 NEWLINE     000A       0A
 ESCAPE      001B       1B

 SPACE       0020       20
 !           0021       21
 "           0022       22
 #           0023       23
 $           0024       24
 *           002A       2A
 +           002B       2B

 0           0030       30
 1           0031       31
 2           0032       32
 3           0033       33

 A           0041       41
 B           0042       42
 C           0043       43
 D           0044       44

 a           0061       61
 b           0062       62
 c           0063       63
 d           0064       64
```

For example, the Unicode character code for the uppercase letter A is

```
0041H        0000 0000 0100 0001 B
```

and the corresponding ASCII code is

```
41H          0100 0001 B
```

The 8 most significant bits of the Unicode character code of A are zeroes. As leading zeroes in a number are not mathematically significant, the two codes above can be considered equal.

We shall see later that the C# programming language encodes textual information according to the Unicode system. Therefore, when a C# program stores a character, it reserves 16 bits (2 bytes) of memory to store the code of the character. Of those 16 bits, however, 8 most significant bits are zeroes when the character is one that belongs to the English alphabet.

Because the first 256 Unicode character codes are the same as the ASCII codes, you can use the table on page 600 to find out the Unicodes of characters. For example, when you want to find out the Unicode character code of the letter R, you can pick up the code 52H from the table, and add two leading zeroes to the hexadecimal code.

3.4 More information: pictures, sound, and moving pictures

The memory of a computer, whether it is silicon-based main memory, magnetic disk memory, or optical memory such as CD, contains just bits, zeroes and ones. All these types of memory store information in digital binary form. Numbers and texts are stored in binary form, and so it is in the case of such forms of information as pictures, sound, and even moving pictures.

We learned in the previous section that to store textual information in computer memory, there has to be a commonly accepted standard. The Unicode system is a standard according to which textual information can be stored and transferred in digital form. To store pictures, sound (music), and moving pictures in digital form, there are own special standards for each type of information. These standards can be called storage formats. One storage format can be more widely accepted than another storage format.

A picture can be put into digital form so that the picture area is divided into thousands of tiny points, and the color and brightness of each point is then described with a numerical value which is stored in binary form. The picture area of a digital picture can be, for example, 2048 points wide and 1536 points high. If the color and brightness of each point were stored in an 8-bit byte, storing this kind of a digital picture would require 2048 * 1536 = 3145728 bytes of storage capacity. This is the basic idea for how pictures can be stored in digital form. The existing standard formats for storing picture information are, however, more complex than the description above. Examples of digital picture formats are:

- GIF, Graphics Interchange Format,
- JPEG, the format of Joint Photographic Experts Group, and
- TIFF, Tagged Image File Format.

Sound can be converted to digital form so that extremely many samples are taken from the original sound, and each sample is then described with a numerical value which is stored in binary form. The music on audio CDs, for example, is made by taking every second thousands of samples from the music. The millions of resulting samples are stored on a CD in binary form. When the CD is played with a CD player, the player constructs the original sound from the binary sound samples. The audio CD standard is one way to store sound in digital form. Another example of a digital sound standard is called MP3.

Moving pictures are in digital form on video DVDs and on the tapes used in digital video cameras. Storing moving pictures in digital form requires a lot of storage capacity because moving pictures are made of many still pictures. Standards for storing moving pictures in digital form are set by Moving Pictures Experts Group (MPEG). The standard for storing digital video pictures on CD is called MPEG-1. The standard that is used on DVDs and digital TV is called MPEG-2. It is likely that companies that manufacture digital video cameras have their own standards for storing moving pictures on tape.

The purpose of this section was to give you an overview of the possibilities which exist to store digital information in computer memory. Storing and handling of digital information requires computer programs, and the purpose of this book is to teach you to write computer programs. While learning computer programming we will concentrate only on numerical and textual information in this book. The reason for this is that the handling of numerical and textual information is much easier for a beginner than the handling of pictures, sound, and moving pictures. After you have learned computer programming sufficiently well, you may one day write programs which do something smart with sound or moving pictures.