# Digital Systems and Computer

John SUM

Institute of Technology Management

National Chung Hsing University

Taichung, ROC

October 28, 2019

# Contents

# 1 Logic Gates

A computer is a basically a digital system. Loosely speaking, digital system is a system that is able to perform logical and arithmetical operations[1]. It consists of many digital logic circuits. Some of these circuits are implemented by simple logic gates to perform those operations. Some of them are control circuits. They control the activations of the logic gates (or logic circuits) and the connections amongst them.

## 1.1 Logic Gate

Logic gate is the basic building block of a digital system. Common logic gates include AND gate, OR gate, NAND gate, NOR gate, XOR[2] gate and NOT gate. Except NOT gate which is a single-input-single-output logic gate, all others are two-input-single-output logic gates.

For electronic logic gates, the inputs to the gates are electrical signals (voltage). The logical signal could be defined in many different ways. For instance, 'high' voltage refers to 'TRUE' (or '1') and 'low' voltage refers to 'FALSE' (or '0'). Another setting is that 'positive' voltage refers to 'TRUE' (or '1') and 'negative' voltage refers to 'FALSE' (or '0'). As there are only two types of signal, this kind of signal is called binary (or digital). Output signal of a logic gate is also binary.

There are many specifications for high and low signals. One specification is that 5V for high signal and 0V for low signal. Another specification is that 5V for high and -5V for low. As both input and output signals are binary (i.e. digital), a logic gate performs simple logic function which implies the name logic gate. Therefore, the systems which are composed of these logic gates are digital systems.

Logic gate is itself an electronic circuit which consists of electronic components. In the early 20 century, vacuum tubes were used for such implementation. In 1940s, semiconductor were invented. Vacuum tubes were replaced by semiconductor transistors to implement the logic gates.

---

[1]Precisely, a digital system can simply perform logical operations. Arithmetical operations are implemented based upon these logical operations.

[2]It should be noted that the function of XOR is not logic. However, in computer science field, we still say that it is a logic gate for convenient.

## 1.2   Truth Tables

The logical operation of a logic gate is defined by its truth table. Conventionally, high voltage is represented by '1' and low voltage is represented by '0'. In other words, the basic operation of a logic gate is specified by its truth table.

```
AND GATE
X Y Z
0 0 0
0 1 0
1 0 0
1 1 1

OR GATE
X Y Z
0 0 0
0 1 1
1 0 1
1 1 1

NAND GATE
X Y Z
0 0 1
0 1 1
1 0 1
1 1 0

NOR GATE
X Y Z
0 0 1
0 1 0
1 0 0
1 1 0

XOR GATE
X Y Z
0 0 0
0 1 1
```

```
1 0 1
1 1 0
```

## 1.3  Digital logic design

These logic gates are the basic building blocks for any digital logic circuit that performs logic function. In the following examples, the notations are defined as follows :

| Notation | Meaning | Example |
|----------|---------|---------|
| $\bar{X}$ | Inversion | If $X = 1$, $\bar{X} = 0$. |
| $XY$ | AND operation | $XY = 1$ only if $X = 1$ and $Y = 1$. |
| $X + Y$ | OR operation | $XY = 1$ if either $X = 1$ or $Y = 1$. |

Besides, it should be noted that the following equalities hold for AND and OR operations.

$$ABC = A(BC) = (AB)C, \quad A + B + C = A + (B + C) = (A + B) + C.$$

Here are some illustrative examples that you can try to see if you can understand these basic operations.

**Example 1:** $Z = \overline{A\bar{B}C}$.

```
A B C Z
0 0 0 1
0 0 1 1
0 1 0 1
0 1 1 1
1 0 0 1
1 0 1 0
1 1 0 1
1 1 1 1
```

**Example 2:** $Z = \bar{A}BC + A\bar{B}\bar{C}$.

```
A B C Z
0 0 0 0
```

```
0 0 1 0
0 1 0 0
0 1 1 1
1 0 0 1
1 0 1 0
1 1 0 0
1 1 1 0
```

**Example 3:** $Z = \bar{A}\bar{B}\bar{C} + \bar{A}BC + A\bar{B}\bar{C} + \bar{A}B\bar{C}$.

```
A B C Z
0 0 0 1
0 0 1 0
0 1 0 1
0 1 1 1
1 0 0 1
1 0 1 0
1 1 0 0
1 1 1 0
```

For the above examples, it is clear that such logic functions can simply be implemented by NOT gate, AND gate and OR gate only. However, in computer science, the key problem in digital logic design is to find the simplest design, the one uses the minimum number of logic gates.

## 1.4   Exercises

1. Design a digital circuit which consists of 8 inputs $A, B, C, D, E, F, G and H$ and 1 output $Z$ with the following function.

$$Z = \begin{cases} 1 & \text{if } ABCD = EFGH \\ 0 & \text{otherwise.} \end{cases}$$

2. Design a digital circuit which consists of 8 inputs $A, B, C, D, E, F, G and H$ and 1 output $Z$ with the following function.

$$Z = \begin{cases} 1 & \text{if } A = \bar{E}, B = \bar{F}, C = \bar{G} \text{ and } D = \bar{H} \\ 0 & \text{otherwise.} \end{cases}$$

3. Design a digital circuit which can perform the logic function $Z = AB + CD$.

4. Design an AND gate using XOR gates only.

5. Design an AND gate using NAND gates only.

6. Design an OR gate using NAND gates only.

7. Design an NOR gate using NAND gates only.

8. Design an XOR gate using NAND gates only.

9. Using NAND gates only, design a digital circuit which can perform the logic function $Z = AB + CD$.

# 2 Arithmetical Operations

To start with, we need to know how numbers are represented in decimal and binary forms.

## 2.1 Decimal number

In decimal numeral system (base-ten numeral system), we have 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, and so forth to represent positive integers. Look at the numbers, what can you see? Actually, we use symbols '0', '1', '2', '3', '4', '5', '6', '7', '8' and '9' to form the number.

For value which is in between zero and nine, we use one digit. For value which is in between ten and ninety-nine, we use two digits. For value which is in between one hundred to nine hundred and ninety-nine, we use three digits.

Let us start with the first ten numbers,

```
0 1 2 3 4 5 6 7 8 9.
```

The last digit has reached its largest value 9. So, return to the next line starting with two digits.

```
10 11 12 13 14 15 16 17 18 19.
```

The last significant digit of the last number has reached its largest value 9. So, return to the next line starting and the first digit increments by one, we get that

```
20 21 22 23 24 25 26 27 28 29.
```

gain, the least significant digit has reached its largest value 9. So, return to the next line starting and the first digit increments by one. After repeat until the last number is 79, the next line reads

```
80 81 82 83 84 85 86 87 88 89.
```

Again, the least significant digit has reached its largest value 9. So, return to the next line starting and the first digit increments by one. Now, the first digit in the next line is 9, i.e.

```
90 91 92 93 94 95 96 97 98 99.
```

Again, the least significant digit has reached its largest value 9. So, return to the next line starting and the first digit increments by one. However, the first digit has already reached its largest value. Now, we need three digits. That is,

```
100 101 102 103 104 105 106 107 108 109.
```

See! That is the way we make up the positive integers. One should note three important numbers, '0', '10' and '100'. They are the first number on the line. If we unify the number of digits to be 3, the numbers are listed as follows.

```
000 001 002 003 004 005 006 007 008 009
010 011 012 013 014 015 016 017 018 019
020 021 022 023 024 025 026 027 028 029
 .  .  .
090 091 092 093 094 095 096 097 098 099
100 101 102 103 104 105 106 107 108 109
```

## 2.2 Binary number

Now, let us talk about the binary number. In binary numeral system, we use two symbols, '0' and '1'. In each line, we only allow '0' and '1'. '1' is the largest value. So, by the same principle as what we list the numbers in decimal numeral system, we can list all the numbers in binary form. The first line reads

```
0 1.
```

The rightmost digit has reached its largest value. So, we need to start a new line with two digits, i.e.

```
10 11.
```

The rightmost digit has reached its largest value. So, we need to start a new line with two digits. However, the first digit has reached its largest value. So, we need to start with a new line with three digits, i.e.

```
100 101.
```

The rightmost digit has reached its largest value. So, we need to start a new line with the second digit increments by one, i.e.

```
110 111.
```

Again, if we unify the number of digits to three, the above numbers will be listed as follows.

```
000 001
010 011
100 101
110 111
```

So, you see! The numbers '0', '1', '2', '3', '4', '5', '6', '7' in decimal form show quite different appearance in binary form.
**Question:** How do we know '101' in binary form is '5' in decimal form?
Answer: By counting.
**Question:** Would there be an easy way for us to identify its value in decimal form?
Answer: By using formulae.

Here are two examples.

$$101_2 = 1 \times 4 + 0 \times 2 + 1 \times 1 = 5_{10}.$$

$$100_2 = 1 \times 4 + 0 \times 2 + 0 \times 1 = 4_{10}.$$

4, 2, and 1 are the basis for three digit binary number. For a binary number with five digits, the basis will be 16, 8, 4, 2 and 1. The decimal value of '10110' is given by

$$10110_2 = 1 \times 16 + 0 \times 8 + 1 \times 4 + 1 \times 2 + 0 \times 1 = 22_{10}.$$

Now, it is your turn.

## 2.3 Number Conversion

First of all, we need to indicate a number if is belongs to a binary number or a decimal number. For a binary number, we use subscript '2'. EG. $101_2$. For a decimal number, we use subscript '10'. EG. $21_{10}$. In this regard, we can see that $101_2$ equals to $5_{10}$, and $21_{10}$ equals to $10101_2$.

### 2.3.1 Binary to Decimal

To convert a binary number to a decimal number, one needs the following table. Suppose that we want to convert a number $11011101_2$ to decimal.

| Weight | $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
|---|---|---|---|---|---|---|---|---|
| Digit | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |

Equivalently, the above table can be depicted as the following table.

| Weight | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|
| Digit | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |

Then, the decimal number of $11011101_2$ is done by the following formulae.

$$
\begin{aligned}
11011101_2 &= 1 \times 128_{10} + 1_{10} \times 64_{10} + 0 \times 32_{10} \\
&+ 1 \times 16_{10} + 1 \times 8_{10} + 1 \times 4_{10} \\
&+ 0 \times 2_{10} + 1 \\
&= 221_{10}.
\end{aligned}
$$

In summary, let an $n$-bit positive binary number, say $X_2$, is specified as $a_{n-1}\cdots a_1 a_0$, where $a_k \in \{0,1\}$ for $k = 0,\cdots,n-1$. Its decimal number $(X_{10})$ can be calculated as follows :

$$X_{10} = a_{n-1} \times 2^{n-1} + a_{n-2} \times 2^{n-2} + \cdots + a_1 \times 2^1 + a_0 \times 2^0.$$

As the largest number is the number with all ones, i.e. $a_k = 1$ for $k = 0,\cdots,n-1$, the largest number that can be represented by an $n$-bit is

$$2^{n-1} + 2^{n-2} + \cdots + 2^1 + 2^0,$$

which is equal to $2^n - 1$.

### 2.3.2 Decimal to Binary

To convert a decimal number to binary, the steps are different. We apply long division. Let us have two simple examples.

**Convert $5_{10}$ to binary number.**

| 2 | 5 | '1' |
|---|---|---|
| 2 | 2 | '0' |
| | '1' | |

So, $5_{10} = 101_2$.

**Convert $50_{10}$ to binary number.**

| 2 | 50 | '0' |
|---|---|---|
| 2 | 25 | '1' |
| 2 | 12 | '0' |
| 2 | 6 | '0' |
| 2 | 3 | '1' |
| | '1' | |

So, $50_{10} = 110010_2$.

## 2.4 Binary Addition

Addition of two binary numbers is similar to the addition of two decimal numbers. Let have three examples.

$$
\begin{array}{r}
0101 \\
+\quad 0100 \\
\hline
1001
\end{array}
\qquad
\begin{array}{r}
0011 \\
+\quad 0110 \\
\hline
1001
\end{array}
\qquad
\begin{array}{r}
0111 \\
+\quad 0011 \\
\hline
1010
\end{array}
$$

The procedure starts from the least significant bit (LSB) and then moves to left until the most significant bit (MSB) has reached.

$$
\begin{array}{rr}
\text{Carry} & 0000 \\
& 0101 \\
+ & 0100 \\
\hline
& 0001
\end{array}
\qquad
\begin{array}{rr}
\text{Carry} & 0000 \\
& 0101 \\
+ & 0100 \\
\hline
& 0001
\end{array}
\qquad
\begin{array}{rr}
\text{Carry} & 1000 \\
& 0101 \\
+ & 0100 \\
\hline
& 0001
\end{array}
\qquad
\begin{array}{rr}
\text{Carry} & 0000 \\
& 0101 \\
+ & 0100 \\
\hline
& 1001
\end{array}
$$

The bitwise operation is performed by a logic circuit called full adder. Its schematic diagram and truth table are shown in Figure 1. To add two 4-bit (positive) numbers, we would need to have to connect four full-adders in the way as shown in Figure 2. For the sake of explanation, you can imagine that the steps of operation is first done at the rightmost full-adder. Then, the steps move from right to left.

It should be noted that the circuit shown in Figure 2 consists of 8 inputs (4 inputs for each binary number) and 5 outputs. That means, this adder can allow the addition of 1111 and 1111. Moreover, one should note that the D-input of the rightmost adder is set to '0'.

One design of a full-adder is by combining two simpler adders called half-adders. Its schematic diagram and truth table are shown in Figure 3. From the truth table, it is clear that the outputs $C$ and $D$ are essentially implemented by an $XOR$ gate and an $AND$ gate respectively. How two half-adders and an $OR$ gate can implement a full-adder is left as an exercise for you.

## 2.5 Binary Subtraction

Before introducing the method of binary subtraction, we need to know how negative number is represented in binary. One simple approach is by using the leftmost bit as the sign bit. For a 4-bit format, the following table lists the numbers from $-7$ to 7.
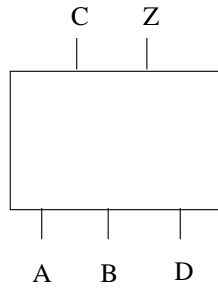
(a) Schematic diagram.

| A | B | D | C | Z |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

(b) Truth table.

Figure 1: Truth table of a full adder.



Direction of Computation
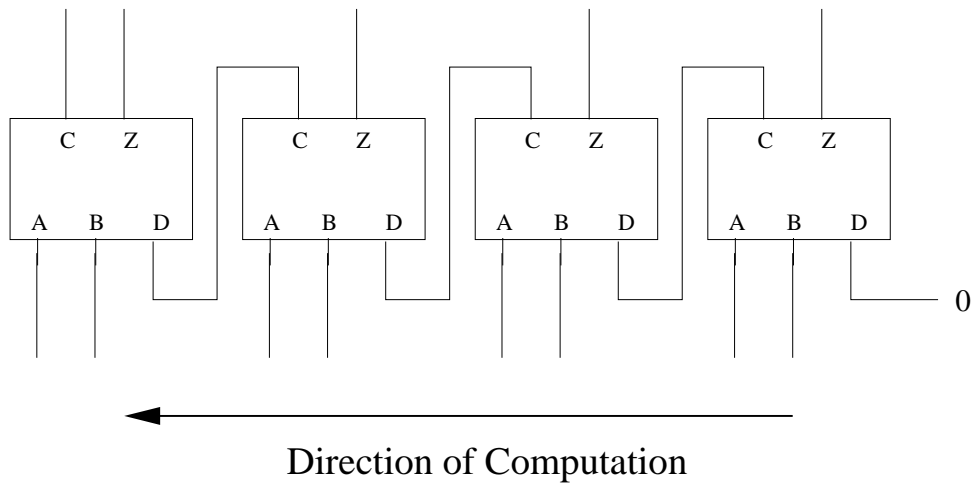
Figure 2: Schematic diagram of a 4-bit adder.
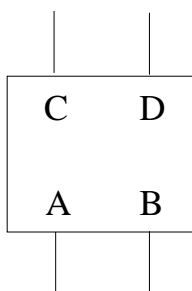
(a) Schematic diagram.

| A | B | C | D |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

(b) Truth table.

Figure 3: Truth table of an half-adder.

| Decimal | Binary | Decimal | Sign |
|---------|--------|---------|------|
| 0 | 0000 | – | – |
| 1 | 0001 | -1 | 1001 |
| 2 | 0010 | -2 | 1010 |
| 3 | 0011 | -3 | 1011 |
| 4 | 0100 | -4 | 1100 |
| 5 | 0101 | -5 | 1101 |
| 6 | 0110 | -6 | 1110 |
| 7 | 0111 | -7 | 1111 |

Sign magnitude format.

While this formate is easy to understand, it is not simplest enough for subtraction operation.

Another approach is based on the idea of 2's complement. Here are the representations of -1 to -7 in 2's complement. As a reference, the representations of 0 to 7 are given.

| Decimal | Binary | Decimal | 2's |
|---------|--------|---------|------|
| 0 | 0000 | – | – |
| 1 | 0001 | -1 | 1111 |
| 2 | 0010 | -2 | 1110 |
| 3 | 0011 | -3 | 1101 |
| 4 | 0100 | -4 | 1100 |
| 5 | 0101 | -5 | 1011 |
| 6 | 0110 | -6 | 1010 |
| 7 | 0111 | -7 | 1001 |

2's complement format.

A simple three steps procedure can help you how to convert a negative number from its decimal form to 2's complement. Let say the number is $-M_{10}$

**Step 1** Convert $M_{10}$ to binary.

**Step 2** Inversion all the bits.

**Step 3** Add the inversion by '1'.

**Example 1:** Convert $-6_{10}$ to its 2's complement, the following three steps give the result.

**Step 1** $6_{10} \rightarrow 0110_2$.

**Step 2** $0110_2 \rightarrow 1001_2$.

**Step 3** $1001_2 + 0001_2 = 1010_2$.

Note that there are 15 numbers (from $-7_{10}$ to $7_{10}$) can be represented by 4 bits. For $N$ bits, the range of numbers that can be represented is from $-(2^{N-1} - 1)$ to $(2^{N-1} - 1)$.

**Example 2:** Convert $-91_{10}$ to its 2's complement, we need 8 bits.

**Step 1** $91_{10} \rightarrow 01011011_2$.

**Step 2** $01011011_2 \rightarrow 10100100_2$.

**Step 3** $10100100_2 + 00000001_2 = 10100101_2$.

With the knowledge of 2's complement, it is time to learn the method of subtraction. Let start with an example.
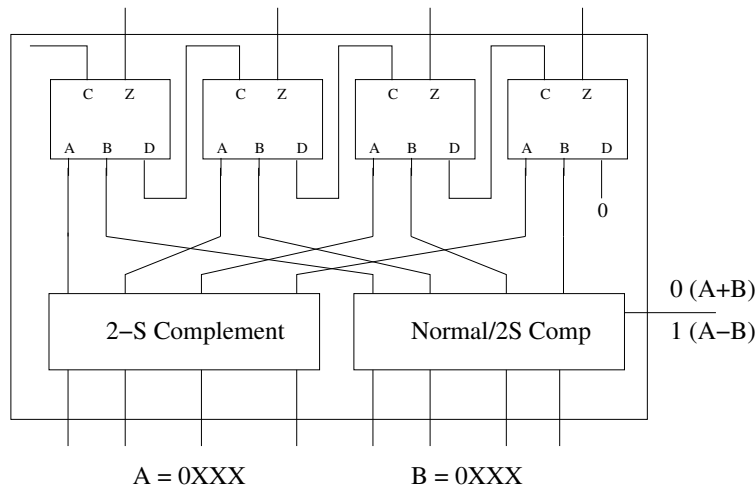
Figure 4: Schematic diagram of a circuit for doing addition and subtraction of two positive numbers.

**Example 3:** The way to calculate the value of $80_{10} - 26_{10}$ is by adding $80_{10}$ and $-26_{10}$, i.e.

$$
\begin{aligned}
80_{10} - 26_{10} &= 80_{10} + (-26_{10}) \\
&= 01010000_2 + 11100110_2 \\
&= 00110110_2.
\end{aligned}
$$

The last step is essentially the binary addition.

## 2.6 Circuit Design for Add/Sub

Figure 4 shows a schematic diagram of a circuit for doing both addition and subtraction of two positive numbers which are in the range of $[0, 7]$. The operation of the "Normal/2's Comp" block is defined in the table below. The pin on the right side of the circuit is to control if the circuit is doing $A + B$ or $A - B$. Here, it is defined that the circuit performs $A + B$ if the signal is '0'. Otherwise, it performs $A - B$.

| Number | $A + B$ | $A - B$ |
|--------|---------|---------|
| 0000 | 0000 | 0000 |
| 0001 | 0001 | 1111 |
| 0010 | 0010 | 1110 |
| 0011 | 0011 | 1101 |
| 0100 | 0100 | 1100 |
| 0101 | 0101 | 1011 |
| 0110 | 0110 | 1010 |
| 0111 | 0111 | 1001 |

Clearly, the circuit shown in Figure 4 is not the only design. There are many others.

## 2.7 Exercises

### Question 1

1. What are the bases for a binary number with eight digits?

2. What is the value of '10000000' in decimal form?

3. What is the value of '10101001' in decimal form?

4. What is the value of '11011011' in decimal form?

5. What is the value of '00101010' in decimal form?

6. What is the value of '11111111' in decimal form?

7. What are the bases for a binary number with sixteen digits?

8. What is the value of '1000000000000000' in decimal form?

9. What is the value of '1111111111111111' in decimal form?

10. What is the largest number (in decimal form) of a binary number with thirty-two digits?

11. What is the largest number (in decimal form) of a binary number with sixty-four digits?

(Hint: The largest value of a binary number with $N$ digits is $2^N - 1$. Why?)

**Question 2**

   (a) Using an $XOR$ and an $AND$ gates to implement a half-adder.

   (b) Using two half-adders and an $OR$ to implement a full-adder.

**Question 3**

In the question, the binary number is of 8-bit format.

   (a) Convert $20_{10}$ into binary number.

   (b) Represent $-12_{10}$ 2'S complement.

   (c) Show the steps of obtaining the value of $20_{10} - 12_{10}$.

**Question 4**

In the question, the binary number is of 16-bit format.

   (a) Convert $20_{10}$ into binary number.

   (b) Represent $-12_{10}$ 2'S complement.

   (c) Show the steps of obtaining the value of $20_{10} - 12_{10}$.

**Question 5**

In the question, the binary number is of 16-bit format.

   (a) Convert $58_{10}$ into binary number.

   (b) Represent $-78_{10}$ 2'S complement.

   (c) Show the steps of obtaining the value of $58_{10} - 78_{10}$.

   (d) Convert the answer in (c) in sign-magnitude form.

# 3   CPU

Central processing unit (CPU) is a digital system which is able to perform all logic and arithmetic operations. To understand how it works, one can refer to the circuit as shown in Figure 4.
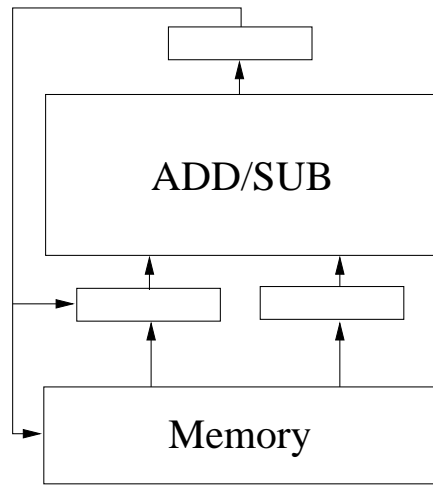
Figure 5: A system consists of arithmetic unit and memory. The control unit is not shown in the diagram.

## 3.1  Simple ADD/SUB system

As observed from Figure 4, the signal pin on the right side controls the operation of the circuit. In other words, it controls operation of the circuit on the inputs. Thus, one can imagine a very complex circuit which consists of logic circuits handling all the arithmetic operations and other circuits handling logic operations on the inputs. We could thus control the operation of this complex circuit by sending *control signals*. But this time, the number of signal pins will be many more than one.

   If moreover there are (electronic) components which can be built for storage (see the blank rectangular blocks in Figure 5), we can connect them to the inputs and the outputs. Let say the temporary storage for inputs are called **IA** and **IB** and the output is called **OUT**. Precisely, it is called **register**. By using the same components, it is for sure that memory can be built. Suppose we have a memory with 16 units. Each unit consists of 4 bits. Let say the memory units are **M1**, **M2**, **M3**, $\cdots$, **M16**.

## 3.2  Program

Now, suppose we have a problem to find out the subtotal of **M1**, **M2**, **M3** and then save the answer on **M4**. We use the term **ADD** as the "name" for

the "control signals" (binary signals) for controlling the circuit to perform addition. The following steps can help to do this task.

```
---------------------
A sample program

M4 = M1 + M2 + M3
---------------------
MOV IA M1
MOV IB M2
ADD IA IB
MOV IA OUT
MOV IB M3
ADD IA IB
MOV M4 OUT
---------------------
```

In the above example, the signal **MOV** is designed for signal-flow from the memory to register, and from register to register. The above steps for adding three numbers, while simple, are called instructions. These seven instructions constitute the program. *A program is a sequence of instructions.* **MOV** and **ADD** are called commands. Since this program describes what the ABB/SUB circuit and the memory should do, it is the so-called **low-level program**. The corresponding binary code is called **machine code**. The software translating the program to machine code is called **compiler**.

# 4 Artificial CPU (JS2019)

Below is a simple circuit. It consists of a **memory** with 16 memory spaces (from M1 to M16), an **ALU block**, 2 **input registers** (IA and IB) and one **output register** (OUT). M1 to M16, IA, IB and OUT are all 8 bits long. Numbers are represented in fixed-point *2's compliment* format.
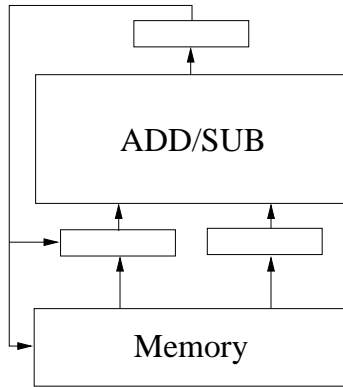
## 4.1 List of Commands

Eleven commands (MOV, ADD, SUB, MUL, DIV, CMP, SHL, SHR, DEF, MSK and IF) are provided for instructing the above circuit. The syntax and the descriptions of these commands are depicted in Table 1.

Table 1: Commands for using the CPU.

| Syntax | Description |
|---|---|
| MOV X Y | Copy the content of Y to X |
| ADD X Y | $OUT = X + Y$. |
| SUB X Y | $OUT = X - Y$. |
| MUL X Y | $OUT = X \times Y$. |
| DIV X Y | $OUT = X/Y$. |
| CMP X Y | $OUT = b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8$. |
| | $b_i = 0$ if $X_i = Y_i$. |
| | $b_i = 1$ if $X_i \neq Y_i$. |
| SHL X Y | $OUT$ is the content of X |
| | shifting left Y bits. |
| SHR X Y | $OUT$ is the content of X |
| | shifting right Y bits. |
| DEF X N | Define X as the number N. |
| MSK X M | Mask the value of X by M. |
| IF ELSE | Condition statement. |

A command is also called an instruction. The set of instruction available

21

for instructing a CPU is called instruction set. For a general purpose CPU, like Intel CORE i5, the total number of instructions is large. So, this type of processors is called complex instruction set processor. For some special purpose processors, like GPU and digital signal processors, they are used for specialized purposes, matrix multiplication for instance. Only a small set of instructions for arithmetic operations is enough. This type of processors is called reduced instruction set processor.

## 4.2  Command Descriptions

1. For the "CMP" command, if $X = 0110$ and $Y = 1101$, $OUT = 1011$.

2. For "SHL" and "SHR" commands, the content of $Y$ can only be one of the following.

   | $Y$ | Meaning |
   |---------|---------|
   | 10000000 | (Shift 7 bits) |
   | 01000000 | (Shift 6 bits) |
   | 00100000 | (Shift 5 bits) |
   | 00010000 | (Shift 4 bits) |
   | 00001000 | (Shift 3 bits) |
   | 00000100 | (Shift 2 bits) |
   | 00000010 | (Shift 1 bits) |
   | 00000001 | (No shift) |

   For example, if
   $$X = 00011000, Y = 00000100,$$
   the $OUT$ of "SHL X Y" is 01100000 and the $OUT$ of "SHR X Y" is 00000110.

3. For the "DEF" command, $N$ must be a number in *decimal* form. $X$ can only be a memory location. "DEF" command is not applicable for assigning values to a register. It is used to assign a value to a memory location. For example, "DEF M1 12" means that memory location $M1$ will be assigned with a value 12. Therefore, $M1 = 00001100$.

4. For the "MSK" command, it is used for masking a register (either IA or IB) by the mask $M$ (in binary). The mask must be 8 bits long.

Suppose that the content of $IA$ and $M$ are defined as follows :

$$IA = 01001001, M = 11110000.$$

Then, the output $OUT$ will be "01000000". The last four bits are masked. Here is an example.

```
-----------------------
DEF M1 45
MOV IA M1
MSK IA 00001111
MOV M2 OUT
-----------------------
```

Initially, $M1$ is assigned with value 45. In binary form, the content reads "00101101". Thus, the output $OUT$ is "00001101".

5. The "IF-ELSE" command is an advanced level command. It is for conditional statement. Once it is executed, the CPU will performs multiple steps in order to make it works. You do not need to know the detail how it works. In term of its usage, it is simple. Here is an example.

```
-----------------------
DEF M1 1
DEF M2 2
DEF M3 1

MOV IA M1
IF IA == 0
    MOV IA M2
    MOV IB M3
    ADD IA IB
    MOV M4 OUT
ELSE
    MOV IA M1
    MOV IB M2
    ADD IA IB
```

```
    MOV M4 OUT
ENDIF
-----------------------
```

Command "IF" checks if the content of IA is identical to "0". If it is, it will perform $M2 + M3$ and output the result to $M4$. Otherwise, it will perform $M1 + M2$ and output the result to $M4$.

```
-----------------------
DEF M1 1
DEF M2 2
DEF M3 1

MOV IA M1
IF IA == 0
    MOV IA M2
    MOV IB M3
    ADD IA IB
    MOV M4 OUT
ENDIF
-----------------------
```

In this example, the CPU performs $M2 + M3$ only if $IA$ is zero. Otherwise, it performs nothing.

6. For the "IF-ELSE" command, the following conditions are allowed for you to define. Here $NUM$ must be stated in decimal form but not in binary.

```
-----------------------
IA == NUM
IA > NUM
IA >= NUM
IA < NUM
IA <= NUM
-----------------------
```

## 4.3 Exercises

### Question 1

Refer to the artificial CPU and its commands, what will be the content of $M4$ if the following commands are executed?

```
DEF M1 0
DEF M2 2
DEF M3 5

MOV IA M1
IF IA == 0
    MOV IA M2
    MOV IB M3
    ADD IA IB
    MOV M4 OUT
ELSE
    MOV IA M1
    MOV IB M2
    ADD IA IB
    MOV M4 OUT
ENDIF
```

**Answer:**

  (a) 2.

  (b) 7.

  (c) 5.

  (d) 0.

### Question 2

What will be the content of $M4$ if the following program segment is executed?

```
DEF M1 16
DEF M2 22
DEF M3 10
```

```
MOV IA M1
MOV IB M2
CMP IA IB
MOV M4 OUT
MOV IA M2
MOV IB M3
CMP IA IB
MOV IA OUT
MOV IB M4
ADD IA IB
MOV M4 OUT
```

**Answer:**

(a) 28.

(b) 30.

(c) 32.

(d) 34.

## Question 3

Refer to the artificial CPU and its commands, what will be the content of $M4$ if the following commands are executed?

```
DEF M1 0
DEF M2 2
DEF M3 5

MOV IA M1
IF IA == 0
   MOV IA M2
   SHL IA 00000100
   MOV IA OUT
   MOV IB M2
   ADD IA IB
   MOV M4 OUT
ELSE
```

```
    MOV IA M3
    SHL IA 00000100
    MOV IA OUT
    MOV IB M3
    ADD IA IB
    MOV M4 OUT
ENDIF
```

**Answer:**

  (a) 4.

  (b) 6.

  (c) 8.

  (d) 10.

## Question 4

Three numbers have been stored in M1, M2 and M3. Which of the following program segments can correctly give the output of the following formulae?

$$M4 = M1 + M2 \times M3.$$

**Answer:**

  (a) ----------------
```
    MOV IA M1
    MOV IB M2
    MUL IA IB
    MOV IA OUT
    MOV IB M3
    ADD IA IB
    MOV M4 OUT
```
     ----------------

  (b) ----------------
```
    MOV IA M1
    MOV IB M2
    ADD IA IB
```

```
        MOV IA OUT
        MOV IB M3
        MUL IA IB
        MOV M4 OUT
        ----------------

  (c)   ----------------
        MOV IA M2
        MOV IB M3
        MUL IA IB
        MOV IA OUT
        MOV IB M1
        ADD IA IB
        MOV M4 OUT
        ----------------

  (d)   ----------------
        MOV IA M2
        MOV IB M3
        ADD IA IB
        MOV IA OUT
        MOV IB M1
        MUL IA IB
        MOV M4 OUT
        ----------------
```

## Question 5

Given that there are five memories M1, M2, M3, M4 and M5. Here is the program segment to instruct the circuit.

```
----------------
MOV IA M1
MOV IB M2
MUL IA IB
MOV M5 OUT
MOV IA M3
MOV IB M4
MUL IA IB
```

```
MOV IA OUT
MOV IB M5
ADD IA IB
MOV M5 OUT
----------------
```

Which of the following mathematical equation is identical to the operation of the following program segment?

**Answer:**

(a) $M5 = M1 + M2 \times M3 + M4$.

(b) $M5 = (M1 + M2) \times M3 + M4$

(c) $M5 = M1 \times (M2 + M3) \times M4$.

(d) $M5 = M1 \times M2 + M3 \times M4$.

## Question 6

Given that there are five memories M1, M2, M3, M4 and M5. Here is the program segment to instruct the circuit.

```
----------------
MOV IA M1
MOV IB M2
MUL IA IB
MOV IA OUT
MOV IB M3
MUL IA IB
MOV IA OUT
MOV IB M4
SUB IA IB
MOV M5 OUT
----------------
```

which of the following mathematical equation is identical to the operation of the following program segment?

**Answer:**

(a) $M5 = M4 - M1 \times M2 \times M3$.

(b) $M5 = M4 - (M1 + M2) \times M3$

(c) $M5 = M1 \times M2 \times M3 - M4$.

(d) $M5 = (M1 + M2) \times M3 - M4$.

## Question 7

If the content in $M1$ is either '1' or '0', which of the following mathematical equation is identical to the operation of the following program segment?

```
MOV IA M1
IF IA == 0
   MOV IA M2
   SHL IA 00000100
   MOV IA OUT
   MOV IB M2
   ADD IA IB
   MOV M4 OUT
ELSE
   MOV IA M3
   SHL IA 00000100
   MOV IA OUT
   MOV IB M3
   ADD IA IB
   MOV M4 OUT
ENDIF
```

**Answer:**

(a) $M1 \times (9 \times M2) + (1 - M1) \times (9 \times M3)$.

(b) $M1 \times (5 \times M2) + (1 - M1) \times (5 \times M3)$.

(c) $(1 - M1) \times (9 \times M2) + M1 \times (9 \times M3)$.

(d) $(1 - M1) \times (5 \times M2) + M1 \times (5 \times M3)$.

# 5　Computer Organization

Generally speaking, a computer is a huge logic circuit. For clarity, this huge logic circuit is partitioned into different sub-systems namely central processing unit, memory, input/output device, network communication device. To manage the usage of these hardware units, an operating system is needed.

## 5.1　Hardware

### 5.1.1　CPU & Registers

The central processing unit (CPU) is responsible for doing arithmetic and logic operations. (CPU) would not just have an ADD/SUB system in it. A CPU consists of at least a unit called **Arithmetic/Logic Unit** (ALU), an **Instruction Decode** unit and a **Control** unit. To improve the performance of a CPU, **registers** (normally tens of Bytes) and **cache memory** (normally a few hundred KBytes) are usually embedded inside the CPU.

### 5.1.2　Memory

Memory (primary storage) is used for temporarily storing the data or instruction that are needed for the CPU. Usually, this primary memory is called **RAM** (random access memory). It is workable only when power is off. Once the power is off, the content in the RAM will be gone. For storing the data permanently, we need secondary storages – the hard-disk, the CD or USB flash memory. In term of memory capacity, RAM usually ranges from a few hundred mega bytes (MBytes) to a few giga bytes (GBytes). The memory capacity of a USB flash memory can now be made to 128 GBytes. Hard-disk or the latest solid state drive (SSD) has the largest capacity up to tera bytes (TBytes).

One should note that the access speed of different types of memory are quite different. By access speed, it means the time for transferring data from the memory to the CPU. Since registers and cache are embedded in the CPU, it is clear that the access speed is the fastest. RAM is mounted on the mother board. While its location is still close to the CPU, transferring data between CPU and RAM is 1000 times slower than registers. For the secondary storages like hard-disk, the access speed is even slower. Normally, it is 1000 times slower than RAM.

### 5.1.3 I/O devices

Input devices are those devices that can let user to enter information to the computer. Keyboard, scanner and mouse are three common input devices. Output devices are those devices that can let user to know what is happening in the computer. Monitor and loudspeaker are two common output devices.

### 5.1.4 NetCom devices

Strictly speaking, network communication device is not belongs to be part of a computer. Recall that the usage of the first generation electronic computer did not require network communication. But nowadays, the usage of computer is attached to Internet access. So, network communication device turns out to be an important part in a computer. Without such, the usage of a computer will be so limited.

In a computer, the network communication device is referred to the Internet communication device. It is used for making communication via the Internet. In a smartphone, there are two communication devices. One is used for making Internet communication. The other is for making telecommunication.

## 5.2 Operating systems

Once you have understood that a computer is just a combination of various logic circuits and devices for input/output, what you need to know more is the operating system. Operating system is a complicated program. It consists of many lines of machine code (instruction)[3]. Operating system starts running when the computer is power on and then stops when the power is off. The main purposes of the operating system is to manage the file system, the memory, the input/output devices, see Figure 6. So that, computer user can interact with the operating system via either command prompt or GUI to control the computer.

Common operating systems are Window XP, Windows 7/8, Apple iOS for personal computers, Andorid, iOS, Window Phone 8 for smart phones. Unix and Linux are two common operating systems for network of computers.

---

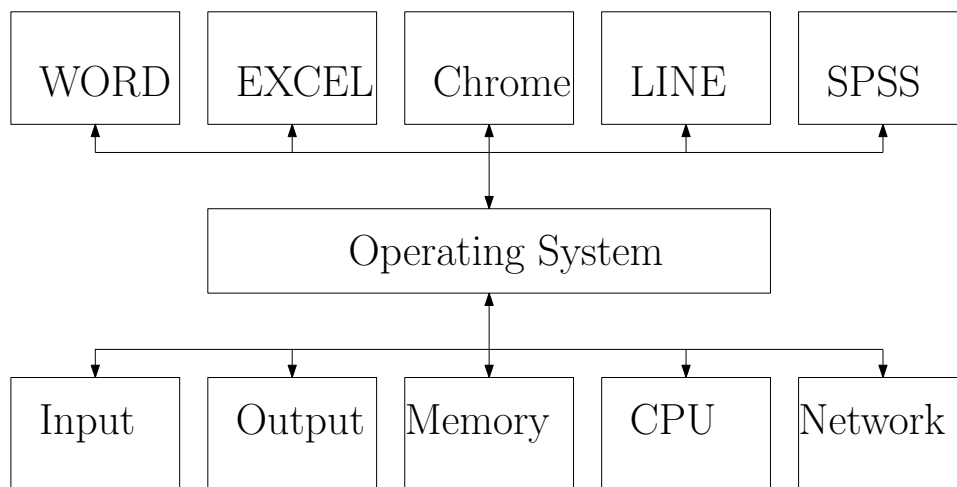[3]Recall that machine codes are sequence of electrical signals feeding to the CPU.

Figure 6: Operating system provides services to support the application systems.

# 6 More on Binary Number Representations

Previous section on computer arithmetic has introduced two formats for binary number, the unsigned integer and 2's-compliment. In fact, there are many other formats for number representation and they could be categorized as fixed-point format and floating point format. Unsigned integer and 2's-compliment are two special cases of fixed-point format.

## 6.1 Fixed-point

### 6.1.1 Sign-Magnitude

To represent a decimal number, say $-4.75$, in binary format, we need to have a binary point and a sign bit. One representation called sign-magnitude format is given below.

$$
\begin{aligned}
-4.75_{10} &= -\left(2^2 + 2^{-1} + 2^{-2}\right) \\
&= -\left(0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3}\right) \\
&= 10100.110_2.
\end{aligned}
$$

The general form of a $(m + q + 1)$-bit format is given by

$$
\text{Sign} \times \left(a_m \times 2^m + \cdots a_1 \times 2^1 + a_0 \times 2^0 + b_1 \times 2^{-1} \cdots b_q \times 2^{-q}\right),
$$

33

where $m$ is the number of integer bits and $q$ is the number of fractional bits. Remember that the leftmost bit is sign bit.

To convert a fractional number into binary, we apply the method of subtraction instead of long division.

$$0.75 = 2^{-1} + 0.25 = 2^{-1} + 2^{-2}.$$

Note that not all fractional number can be represented by finite number of fractional bits. Here gives you three examples.

$$
\begin{aligned}
0.236_{10} &= 0.00111100011010100111111011111\cdots \\
0.4852_{10} &= 0.01111100001101100001000100110\cdots \\
0.1_{10} &= 0.0001100110011001100110011\cdots
\end{aligned}
$$

Even for the decimal numbers 0.1 and 0.2, they cannot be represented by finite number of fractional bits. In this regard, these numbers have to be truncated (i.e. chopped) or rounded. To illustrate the ideas, let the number of fractional bits be 5.

**Truncation :** The idea of truncation is simply by chopping. Only the first five fractional bits are kept and ignore the others. Another name for this method is called round-down.

$$
\begin{aligned}
0.236_{10} &= 0.00111 \\
0.4852_{10} &= 0.01111 \\
0.1_{10} &= 0.00011.
\end{aligned}
$$

The maximum error, called precision error, committed due to truncation is $2^{-q}$. In the above examples, the precision error is $2^{-5}$.

**Rounding :** The method of rounding is the same as what we have learnt in elementary mathematics, round to nearest number. Here are two examples.

$$
\begin{aligned}
[0.236] &= 0.24 \ \text{(Round to two decimal points!)} \\
[0.4852] &= 0.485 \ \text{(Round to three decimal points!).}
\end{aligned}
$$

By the same principle, by rounding the above binary numbers to five fractional bits, we get that

$$
\begin{aligned}
0.236_{10} &= 0.01000 \\
0.4852_{10} &= 0.10000 \\
0.1_{10} &= 0.00011.
\end{aligned}
$$

As compared with the numbers obtained by truncation, the first two numbers are represented differently. The precision error, committed due to truncation is $2^{-(q+1)}$. In the above examples, the precision error is $2^{-6}$.

**Guard bits :** It is clear that precision error will be amplified if there are a large amount of numbers to be added. To overcome such problem, the number to be operated in the CPU is converted to a higher precision format, say from 16 bits to 20 bits or 32 bits. All arithmetic operations are thus conducted in this high precision level. Here is an example. Suppose the numbers are represented in 8-bit fixed point with 5 fractional bits, i.e. $F(8,5)$. In the CPU, the number of bits is 16.

$$
\begin{aligned}
0.4852_{10} \times 0.1_{10} &= 0.10000 \times 0.00011 \\
&= 000.10000 \times 000.00011 \\
&= 000.1000000000000 \times 000.0001100000000 \\
&= 000.0000100000000 + 000.0000010000000 \\
&= 000.0000110000000 \\
&= 000.000011
\end{aligned}
$$

The extra bits are called the guard bits.

### 6.1.2   2's-Compliment

To represent a fractional number in 2's-compliment, the idea and the conversion method are the same as for integer number. To make it easier to understand, we consider the number which can be represented by finite number of fractional bits.

$$3.71875_{10} = 11.10111_2.$$

Now, we would like to convert $-3.71875_{10}$ into binary. Here, we assume that the number of integer bits is 2 and the number of fractional bits is 5.

**Step 1** $3.71875_{10} \rightarrow 011.10111_2$.

**Step 2** $011.10111_2 \rightarrow 100.01000_2$.

**Step 3** $100.01000_2 + 000.00001_2 = 100.01001_2$.

So, we can check that the beauty of 2's-compliment format in subtraction preserves.

$$
\begin{aligned}
3.71875_{10} - 3.71875_{10} &= 3.71875_{10} + (-3.71875_{10}) \\
&= 011.10111_2 + 100.01001_2 \\
&= 000.00000_2 \\
&= 0_{10}.
\end{aligned}
$$

The addition in the second line is implemented by 8-bit full adder.

### 6.1.3 Range

For a fixed-point with $m$ integer bits and $q$ fractional bits, the maximum and minimum numbers it can be represented are given below.

$$
\mathrm{Max} = +\underbrace{111\cdots111}_{m \text{ bits}}.\underbrace{111\cdots111}_{q \text{ bits}}. \quad \mathrm{Min} = -\underbrace{111\cdots111}_{m \text{ bits}}.\underbrace{111\cdots111}_{q \text{ bits}}.
$$

The maximum number is given by

$$
\begin{aligned}
\mathrm{Max} &= \sum_{i=0}^{m} 2^i + \sum_{j=1}^{q} 2^{-j} \\
&= 2^{-q} \times \left( \sum_{i=0}^{m+q} 2^i \right) \\
&= 2^{-q} \left( 2^{m+q+1} - 1 \right) \\
&= 2^{m+1} - 2^{-q}.
\end{aligned}
$$

Therefore, the binary numbers that can be represented are in the range

$$
\left[ -\left( 2^{m+1} - 2^{-q} \right), \left( 2^{m+1} - 2^{-q} \right) \right].
$$

The interval between two consecutive numbers is $2^{-q}$.

## 6.2 Floating-point

As the range of fixed-point format is finite, it is not possible to represent the numbers outside the range. In this regard, we need another format to represent large numbers. It is the floating-point format.

Recall that the speed of light is 299792458 meters per second ($m/s$). Sometimes, we write it approximately as $3.0 \times 10^8 m/s$. It is already a base-10 floating-point format. Three important information in this representation, (i) the sign (i.e. '+'), (ii) the significant (i.e. 3.0) and (iii) the exponent (i.e. 8).

### 6.2.1  Representation

To illustrate how a fractional number can be represented in floating-point, let us consider the same number $3.71875_{10}$ (equivalently $11.10111_2$). To convert it to a floating-point, we carry the following steps. Here, we assume that there is one sign bit and the number of significant bits is 11. The number of exponent bits is 5 and the exponent bias is 15. It is exactly the half-precision floating-point format as defined in IEEE 754-2008 standard.

**Step 1**  $3.71875_{10} \rightarrow 011.10111_2$.

**Step 2**  $011.10111_2 \rightarrow 1.110111 \times 2^1$. Significant bits are 11011100000.

**Step 3**  Exponent $= 1 + 15 = 16$. Exponent bits are 10000.

**Step 4**  $3.71875_{10} \rightarrow 0100001101110000_{hf}$.

**Step 2** is the normalization step. The subscript $hf$ stands for half-float.

Let us consider another slightly different number $24.71875_{10}$. Clearly, the binary bits for the fractional part is the same. For the integer part, it is represented by $11000_2$. So, the conversion can be done by the following steps.

**Step 1**  $24.71875_{10} \rightarrow 11000.10111_2$.

**Step 2**  $11000.10111_2 \rightarrow 1.100010111 \times 2^4$.

**Step 3**  Exponent $= 4 + 15 = 19$. Exponent bits are 10011.

**Step 4**  $24.71875_{10} \rightarrow 0100111000101 1100_{hf}$.

To represent a negative number in floating format, simply turn the sign bit in '1'. For instance,

$$
\begin{aligned}
-3.71875_{10} &= 1100001101110 0000_{hf} \\
-24.71875_{10} &= 1100111000101 1100_{hf}
\end{aligned}
$$

Let us consider the number $2^{-16}$. Based on the normalization step, the number should be represented as follows :

$$2^{-16} = 1.00000000000 \times 2^{-16}.$$

However, the minimum power allowed is $-14$. In such case, there is no scale-down normalization. Instead, normalization is performed in a manner of scale-up, i.e.

$$2^{-16} = 0.01000000000 \times 2^{-14}.$$

Those numbers smaller than $2^{-14}$ are called subnormal numbers. So, the conversion of $2^{-16}$ can be done by the following steps.

**Step 1** $2_{10}^{-16} \to 0.0000000000000001_2$.

**Step 2** $0.0000000000000001_2 \to 0.01 \times 2^{-14}$.

**Step 3** As the number is smaller than $2^{-14}$, exponent bits are 00000.

**Step 4** $2_{10}^{-16} \to 00000001000000000_{hf}$.

**Step 3** is not the same as before. The exponent 00000 is to indicate that the number is subnormal, i.e. smaller than $2^{-14}$.

To clarify this point, let us consider the number $2^{-14} + 2^{-16}$. So, the conversion of $2^{-16}$ can be done by the following steps.

**Step 1** $(2^{-14} + 2^{-16})_{10} \to 0.0000000000000101_2$.

**Step 2** $0.0000000000000101_2 \to 1.01 \times 2^{-14}$.

**Step 3** Exponent $= -14 + 15 = 1$. Exponent bits are 00001.

**Step 4** $(2^{-14} + 2^{-16})_{10} \to 00000101000000000_{hf}$.

### 6.2.2 Interesting numbers

The smallest positive *subnormal* number that can be represented by the above half-precision floating-point is given by

$$\begin{aligned}
0\ 00000\ 00000000001 &= 0.00000000001_2 \times 2^{-14} \\
&= 2^{-11} \times 2^{-14} \\
&= 2^{-25}.
\end{aligned}$$

The largest positive *subnormal* number that can be represented by the above half-precision floating-point is given by

$$
\begin{aligned}
0\ 00000\ 11111111111 &= 0.11111111111_2 \times 2^{-14} \\
&= \left(1 - 2^{-11}\right) \times 2^{-14} \\
&= 2^{-14} - 2^{-25}.
\end{aligned}
$$

Here are a few other interesting numbers.

$$
0\ 00001\ 00000000000 = 2^{-14}.
$$

$$
\begin{aligned}
0\ 01111\ 00000000000 &= 1.00000000000_2 \times 2^{15-15} \\
&= 1.
\end{aligned}
$$

$$
\begin{aligned}
0\ 11110\ 11111111111 &= 1.11111111111_2 \times 2^{15} \\
&= \left(2 - 2^{-11}\right) \times 2^{15} \\
&= 65520_{10}.
\end{aligned}
$$

Note that the largest exponent is 11110, not 11111. The bit pattern 11111 is reserved for other use. Numbers larger than 65520 cannot be represented.

$$
\begin{aligned}
0\ 11111\ 00000000000 &= \infty \\
1\ 11111\ 00000000000 &= -\infty
\end{aligned}
$$

Here, only the 16-bit half-precision floating-point has been introduced. In many processors, the number of bits for a floating-point could be 32 bits, 64 bits and even 128 bits. Their principles of representing a number are almost the same. It consists of a sign bit, a number of bits for exponent and a number of bits for significant. For the exponent, it embraces with an exponent bias. If the exponent bits are all zeros, the number is subnormal.

### 6.2.3 Arithmetic

Floating-point arithmetic is more complicated than fixed-point arithmetic. For instance, addition of two numbers will need to normalize the numbers

with same exponent before doing the addition.

$$1.00010010011 \times 2^8 + 1.01011000000 \times 2^6$$
$$= 1.00010010011 \times 2^8 + 0.01010110000 \times 2^8$$
$$= 1.01101000011 \times 2^8.$$

For multiplication, the computation could be intensive (see the second step).

$$\left(1.00010010000 \times 2^8\right) \times \left(1.01000000000 \times 2^6\right)$$
$$= (1.00010010000 \times 1.01000000000) \times 2^{14}$$
$$= (1.00010010000 + 0.01000100100) \times 2^{14}$$
$$= 1.01010110100 \times 2^{14}.$$

Here, the number 1.01000000000 consists of one bit '1' in the fractional part. So, there is only one addition. If there are many bits '1', the number of addition could be large.

### 6.2.4   FLOPS

The number of steps in performing a single floating-point arithmetic, like addition or multiplication, is more than fixed-point arithmetic. Today, the number of floating-point operations in a second (equivalently, floating-point operations per second (FLOPS)) becomes the default measure for computer performance. Today, an Nvidia GPU can have processing speed up close to $10^{15}$ FLOPS.

## 6.3   Exercises

1. Convert $-32.125$ in 16-bit 2's compliment fixed-point format with 5 integer bits and 10 fractional bits.

2. Describe the steps how two fixed-point numbers are multiplied.

3. Convert $-32.125$ in IEEE 754 half-precision floating point format.

4. Convert the following half-precision floating numbers to decimal numbers.

   (a)  1000001010101010.

(b) 0011100010100000.

5. Describe the steps how two floating numbers are multiplied.