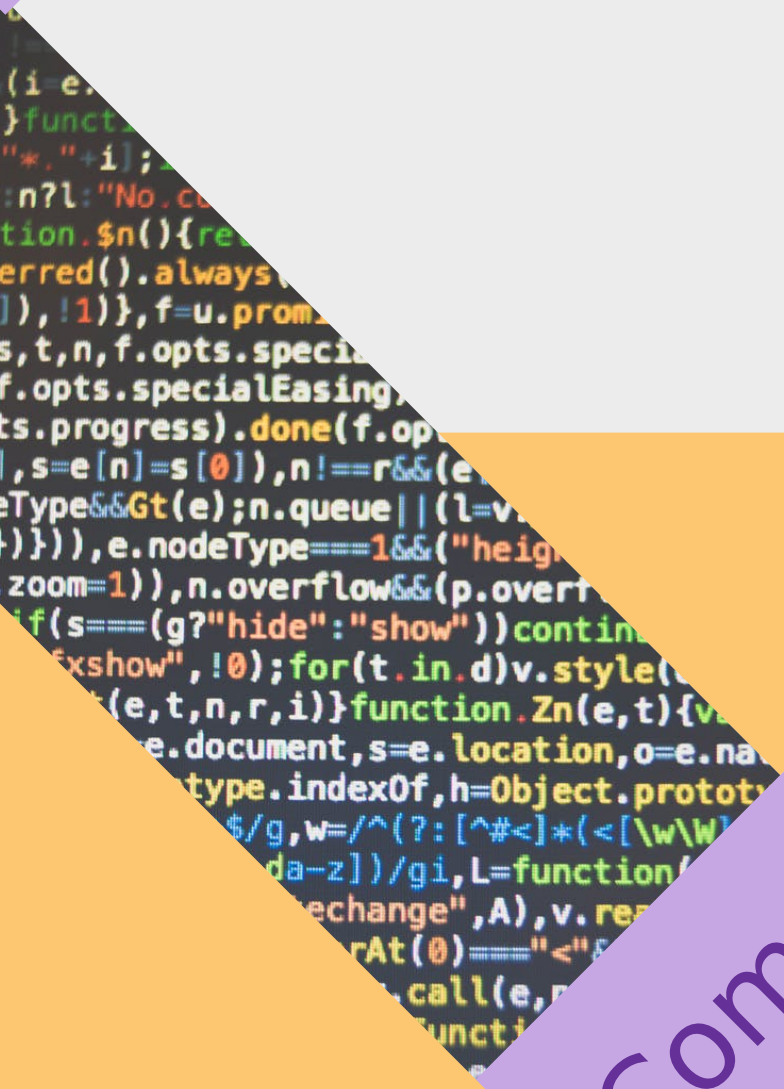
Curs 2020-2021



Computer Systems
Information Representation

IES Jaume II El Just
Tavernes de la Valldigna

Contents

1 Signals and Information **2**

2 Numeration Systems **2**
2.1 Decimal System . . . 2
2.2 Binary Code . . . 3
2.2.1 Binary to Decimal . . . 3
2.2.2 Decimal to Binary . . . 3
2.2.3 Binary arithmetic . . . 4
2.2.4 Boolean Algebra . . . 5
2.3 Octal . . . 6
2.4 Hexadecimal . . . 7
2.5 Representation of integers numbers . . . 8
2.5.1 Signed magnitud . . . 8
2.5.2 Ones' complement . . . 8
2.5.3 Two's complement . . . 8
2.5.4 Excess-Z . . . 9
2.6 Real numbers . . . 10
2.6.1 Fixed Point . . . 10
2.6.2 Floating point . . . 10

3 Unit of information and its system **11**

A computer is not able to work directly with information. First, it must be transformed in order to code it. The structure of the computer will be influenced by the way in which this transformation is done. This chapter will focus on the codification of this information representation.

1 Signals and Information

Information needs to be transformed in order to be manipulated by computers. After this transformation, electric signals are received so that electronic circuits are able to manipulate it in the proper way. So we have two kinds of signals:

- Analogic signals: the signal can get any value in an stablished interval of values.
- Digital signals: the signal gets only one value each time.

For instance, if we think about temperature. The temperature can reach any value in an analog thermometer. But with a digital thermometer, it can only read a limited number of values. Moreover, it's just a specific numeric value.

The smallest unit of information in digital signals (also called binary signals) is the **BIT**. The Bit can have just 2 values: 0, 1. Bits are often joined, 8 bits together are called **BYTE** (8 bits) and 16 bits together are called **WORD**.

2 Numeration Systems

2.1 Decimal System

In math we use the decimal system. The main reason scientists call it decimal system is because we can express 10 values (or values of 10). We can express it from 0 to 9 (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). And if we want to express a greater value, we use the next digit before the units. This digit is called tens. But the value for this digit is not 1. For instance:

$12 = 1 * 10 + 2 * 1 \Rightarrow$ 1 represent tens and it must be multiplied by 10. Number 2 represent units and it must be added and multiplied by 1.

Furthermore, we can express quantities until 99. The total numbers that we are able to represent with 2 digits is 100 (0 to 99):

$99 = 9 * 10 + 9 * 1$

Amount of numbers which can be represented:

B^d = total numbers $\Rightarrow$ B = base of system, d is number of digits

We can get the quantity: $10^2 = 100$ values with 2 digits. $\Rightarrow$ 10 is because we have 10 values (0-9) (decimal system) and 2 because we have 2 digits.

Going on with next digit. if we need to express values greater than 99 we need the next digit. We use the next digit before the tens place. This digit is called hundreds:

$132 = 1 * 100 + 3 * 10 + 2 * 1 \Rightarrow$ 1 represents hundreds and it must be multiplied by 100, 3 represents tens and it must be multiplied by 10. Number 2 represents units, and it must be added and multiplied by 1.

So, as a general rule, we can establish:

General rule in a numeric System

$N = a_n b^n + a_{n-1} b^{n-1} + ... + a_0 b^0 + a_{-1} b^{-1} + ... + a_{-p} b^{-p}$

N $\Rightarrow$ number

$a_n \Rightarrow$ a, digit, n position in the number

$b^n \Rightarrow$ b, base of the system, n position in the number

Example:

$2453,24 = 2 * 10^3 + 4 * 10^2 + 5 * 10^1 + 3 * 10^0 + 2 * 10^-1 + 4 * 10^-2$

2.2 Binary Code

As it has been said before, the smallest unit of information in the digital system is the bit. Bit can take 2 values, 0 and 1. Binary code is the numerical system in order to represent values with this basic information for computers. In order to distinguish numbers from their differents bases, we express numbers with a suscript which indicates the base. Example:
$10011_2 \Rightarrow$ Binary code number
$10011_{10} \Rightarrow$ Decimal system number

In Binary code, it is possible to represent just 2 numbers in 1 bit, 0 and 1. However, if we like to represent values greater than 1, we need 1 more digit before the smallest bit and so on:

Decimal	Binary code
0_{10}	0_2
1_{10}	1_2
2_{10}	10_2
3_{10}	11_2
4_{10}	100_2
5_{10}	101_2
6_{10}	110_2
7_{10}	111_2
8_{10}	1000_2
9_{10}	1001_2

Table 1: Decimal numebers and binary code

2.2.1 Binary to Decimal

According to the general rule described in 2.1, we can express any binary code as a sum of each digits depending on its position. Consequently, we will have the number in decimal code. Example:

$1011011,101_2 = 1*2^6+0*2^5+1*2^4+1*2^3+0*2^2+1*2^1+1*2^0+1*2^(-1)+0*2^(-2)+1*2^(-3) = 91,625_{10}$

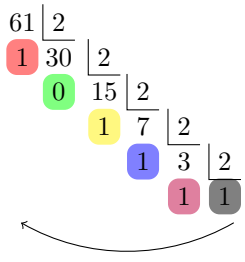
Try

Doing the same with the number in table 1. Essentially, it means convert binary code on the right to a decimal number on the left in order to check out values.

2.2.2 Decimal to Binary

In order to convert decimal numbers to binary code, we must divide decimal numbers between two as many times as needed until the quotient is lower than 2. Finally, binary code will be the result of the last quotient followed by the remainders of the previous divisions.

The following example shows us which has previously been mentioned. The most significant bit (leftmost) is the gray color bit in the image, followed by the one with the purple color, the blue bit, and yellow bit. The green bit and the least significant bit is represented by the red color. $61_{10} = 1\,1\,1\,1\,0\,1_2$



2.2.2.1 Real Numbers

We also need to convert Real Numbers in binary code. To do this, we are going to describe it with an example:

Convert to binary code: 61,625

1. We convert the integer number into the bynary code.$\Rightarrow 61_{10} = 111101_2$
2. We separate the real side. This part will be multiplied by the base (this case 2) as many times as we need to until we get 0. We can also get as many bits as it is said or if it is considered enough. Bits in each digit will be the real part of each product.
   $0.625 * 2 = 1.25 \Rightarrow 1$
   $0.25 * 2 = 0.5 \Rightarrow 0$
   $0.5 * 2 = 1 \Rightarrow 1$
3. $61.625_{10} = 111101,101_2$

Pay attetion to each product. The real part of each product is multiplied, just the real part

Convert to binary code 61,525

1. We convert into binary code the integer side.$\Rightarrow 61_{10} = 111101_2$
2. We separate the real side. This part will be multiplied by the base (this case 2) as many times as we need until we get 0 or we get as many bits as it is said or it is considered enough. Bits in each digit will be the real part of each product.
   $0.525 * 2 = 1.05 \Rightarrow 1$
   $0.05 * 2 = 0.1 \Rightarrow 0$
   $0.1 * 2 = 0.2 \Rightarrow 0$
   $0.2 * 2 = 0.4 \Rightarrow 0$
   $0.4 * 2 = 0.8 \Rightarrow 0$
   There are enough bits for the real part (five)
3. $61.525_{10} = 111101,10000_2$

2.2.3 Binary arithmetic

In the following points of this chapter, other different representation codes will be studied. It is interesting to know how adding, subtracting, multiplying and dividing in binary. Octal, hexadecimal (other represetation systems) are used as a compact representations of binary code.

2.2.3.1 Adding

In order to add 2 digits represented in binary code, we have the following odds:

- 0+0=0 $\Rightarrow$ Carrying=0
- 0+1=1 $\Rightarrow$ Carrying=0
- 1+0=1 $\Rightarrow$ Carrying=0
- 1+1=0 $\Rightarrow$ Carrying=1

And if it is implemented in a sum longer:

$$\begin{array}{r} 1\quad 1\ 11\quad 1 \\ 11010110011 \\ +10010111010 \\ \hline 101101101101 \end{array}$$

2.2.3.2 Subtraction

In order to subtract 2 digits represented in binary code, we have the following odds:

- 0-0=0 $\Rightarrow$ Carrying=0
- 0-1=1 $\Rightarrow$ Carrying=1

- 1-0=1 ⇒ Carrying=0
- 1-1=0 ⇒ Carrying=0

And if it is implemented in longer subtraction:

$$\begin{array}{r} 1\,11 \\ 1\,0001 \\ -1010 \\ \hline 0\,0111 \end{array}$$

2.2.3.3 Product

If we multiply 2 digits, we will get the following results:

- 0x0=0
- 0x1=0
- 1x0=0
- 1x1=1

The final results can be used in order to make a product with more digits. For example:

$$\begin{array}{r} 1011 \\ x\underline{1010} \\ 0000 \\ 1011 \\ 0000 \\ +1011 \\ \hline 1101110 \end{array}$$

2.2.3.4 Division

If we divide 2 digits, we will get the following results:

- 0/0= non defined
- 0/1=0
- 1/0=infinite
- 1/1=1

These results can be used in order to make a division with more digits. Example:

$$\begin{array}{rl} 1\,1\,0\,1100 & \big|\,\underline{101} \\ -\underline{1\,0\,1} & \quad 10101 \\ 0\,0\,1\,11 & \\ -\,\underline{1\,01} & \\ 0\,1000 & \\ -\,\underline{\;101} & \\ 0011 & \end{array}$$

Pay attention to the first subtraction, the solution is 001. And the next digit is downloaded, but it is impossible to do the division. So, the next digit is downloaded again and 0 is written down in quocient. Generally, it works the same.

2.2.4 Boolean Algebra

Binary code belongs group of values (0,1) in which it is possible to apply the Boolean Algebra. Bolean Algebra is a set of logical operations and each answer in this operation is carried in a table:

2.2.4.1 NOT

This operator is NOT logical or ¬.

IN	OUT
0	1
1	0

Table 2: NOT operation

2.2.4.2 AND

This operator is the product and the result is always 1 when both digits are 1. The operation can be represented by this signs: and, ∧, Y, *

IN	OUT
0 and 0	0
0 and 1	0
1 and 0	0
1 and 1	1

Table 3: AND

2.2.4.3 OR

In this operator, the result is always 1 when one of the digits is 1. The operation can be represented by these signs: or, ∨, O .

IN	OUT
0 or 0	0
0 or 1	1
1 or 0	1
1 or 1	1

Table 4: OR

2.2.4.4 XOR

This operatior can be represented by ⊕, xor.

IN	OUT
0 xor 0	0
0 xor 1	1
1 xor 0	1
1 xor 1	0

Table 5: XOR

2.3 Octal

To make things helpful, octal representation is used. In this representation, there are just 8 symbols (0,1,2,3,4,5,6,7). In binary code as well as in decimal, if it is required to represent some quantity greater than the maximum number of one digit (in octal 7), it will be needed more than 1 digit and so on:

Decimal	Binary code	Octal
0_{10}	0_2	0_8
1_{10}	1_2	1_8
2_{10}	10_2	2_8
3_{10}	11_2	3_8
4_{10}	100_2	4_8
5_{10}	101_2	5_8
6_{10}	110_2	6_8
7_{10}	111_2	7_8
8_{10}	1000_2	10_8
9_{10}	1001_2	11_8

Table 6: Decimal numebrs, binary code and octal

It should bear in mind that number 7 is represented with 3 bits, so $7_8 = 111_2$. Taking this into account, 3 bits will be needed to represent an octal digit. This way, it is possible to convert octal to binary code by just converting digit per digit, for instance:

$$347_8 = 011\ 100\ 111_2$$

Likewise, it is possible to convert binary code to octal code just by converting groups of three bits:

$$101\ 110\ 001_2 = 561_8$$

2.4 Hexadecimal

In this representation, there are just 16 simbols (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F). If it is required to represent some quantity greater than 16, it is necessary that there is more than 1 digit and so on:

Decimal	Binary code	Octal	Hexadecimal
0_{10}	0_2	0_8	0_{16}
1_{10}	1_2	1_8	1_{16}
2_{10}	10_2	2_8	2_{16}
3_{10}	11_2	3_8	3_{16}
4_{10}	100_2	4_8	4_{16}
5_{10}	101_2	5_8	5_{16}
6_{10}	110_2	6_8	6_{16}
7_{10}	111_2	7_8	7_{16}
8_{10}	1000_2	10_8	8_{16}
9_{10}	1001_2	11_8	9_{16}
10_{10}	1010_2	12_8	A_{16}
11_{10}	1011_2	13_8	B_{16}
12_{10}	1100_2	14_8	C_{16}
13_{10}	1101_2	15_8	D_{16}
14_{10}	1110_2	16_8	E_{16}
15_{10}	1111_2	17_8	F_{16}
16_{10}	10000_2	20_8	10_{16}

Table 7: Decimal numbers, binary code, octal and hexadecimal

At the same way, you should bear in mind that number 15 is represented by 4 bits, so $15_{16} = 1111_2$. Taking this into account, 4 bits will be needed to represent a hexadecimal digit. This way, it is possible to convert hexadecimal to binary code by converting digit per digit, for instance:

$$CA7_{16} = 1100\ 1010\ 0111_2$$

And in the same way, it is possible to convert binary code to hexadecimal code just converting groups of four bits:

$$1011\ 0110\ 1001_2 = B69_{16}$$

2.5 Representation of integers numbers

This section talks about how is possible to represent integer numbers (negative and positive numbers) in binary code. It is clear that we have just 2 numbers (0, 1) and we need a sign to represent them. Besides we have to take into account that data are kept in groups of binary digits called words. These words are limited in size as it is referred.

2.5.1 Signed magnitud

The first thing to bear in mind, is that we need to represent the sign, So, we will have one of the bits to represent it. For instance, in an 8 bits representation, the 8th bit will represent the sign, 0 in order to represent positive numbers and 1 to represent negative numbers.

$120_{10} \Rightarrow 0\,1111001_2 \Rightarrow$ Positive number so 8th bit is 0

$-120_{10} \Rightarrow 1\,1111001_2 \Rightarrow$ Negative number so 8th bit is 1

Notice!!

Notice how many ways of representing the number 0 there are:

$0\,0000000_2 \Rightarrow 0_{10}$

$1\,0000000_2 \Rightarrow 0_{10}$

Pay attention

So, we have 8 bits but one of them is dedicated to represent sign. So ¿What is the maximum and minimum number which it might be represented?

Don't forget that the 8th bit is the sign and it doesn't represent any value.

$0\,1111111_2 \Rightarrow$ Maximum number (positive number): _____

$1\,1111111_2 \Rightarrow$ Minimum number (negative number): _____

Try it

So, Could you say how many different values there are? Don't say all the values, just the quantity of them

_____

_____

_____

2.5.2 Ones' complement

In ones' complement representation, the most significant bit (leftmost) indicates the sign. So in positive numbers, the sign is indicated with 0, and in negative numbers, therefore, the sign is indicated with 1. Besides, negative numbers are expressed changing every bit from the number in the positive way. For example, for eight bits representation:

$120_{10} \Rightarrow 0\,1111001_2$

$-120_{10} \Rightarrow 1\,0000110_2$

Notice that every bit is complemented (changed).

2.5.3 Two's complement

Two's complememt is used because algebra is simplified in this kind of representation. In this representation, positive numbers are represented in binary code. On the other hand, negative numbers are represented in two's complement that is:

Two's complement 8 bits

- **Step 1**: Represent negative number in binary code as a positive number. Example: Represent -121_{10} in two's complement

  $121_{10} = 01111001_2$

- **Step 2**: Change every bit to it complement (0 to 1, 1 to 0)
  Example: $01111001_2 = 10000110_2$
- **Step 3**: Add 1
  Example: $-121_{10} = 10000110_2 + 1 = 10000111$
- **Step 4**: That's it:
  $-121_{10} = Ca_2(10000111)$

To convert Two's complement to decimal number, pay attention in the following steps:

Two's complement 8 bits

- **Step 1**: Notice whether the most signicant bit (leftmost) is 0 or 1 (positive or negative number)
  Example: Represent 10101001_2 in decimal number
  The most significant bit is 1 $\Rightarrow$ so negative number
  $Ca_2(10101001)$
- **Step 2**: Change every bit to it complement (0 to 1, 1 to 0)
  $10101001_2 \Rightarrow 01010110_2$
- **Step 3**: Add 1
  $01010110_2 + 1 = 01010111$
- **Step 4**: Convert the code binary number in decimal:
  $01010111_2 = 0 * 2^7 + 1 * 2^6 + 0 * 2^5 + 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0 = 87_{10}$
- **Step 5**: That's it:
  -87_{10}

As it was stated previously, the great benefit of Two's complement representation is the algebra. For example in order to sum two numbers with different signs, they will be added. No worries about sign or subtraction. Eventually, if the result is achieved with 1 or more, this bit will be rejected.

Add -75 and 18 in Two's complement

$-75_{10} = Ca_2(-75) = 10110101_2$
$18_{10} = 00010010_2$

$$\begin{array}{r} 10110101 \\ +00010010 \\ \hline 11000111 \end{array}$$

Try it

1. Convert in decimal form the result from last example and check it out.
   The solution will be $-75 + 18 = -57$

2.5.4 Excess-Z

In this system of representaction, a number a in binary is added to a value called Excess. The excess depending on the number of bits in the representation of the number. In fact, $Z = 2^{n-1}$ where n is the number of bits in the representation. This representation is specially interesting in order to compare numbers, although the sign bit takes 1 value for positive numbers and 0 for negatives.

Convert 9 to Excess Z representation with 6 bits

- **Step 1**: Z Excess:
  $Z = 2^{n-1} = 2^{6-1} = 32$
- **Step 2**: Number a + Excess:
  $a + Z = 9 + 32 = 41$
- **Step 3**: Convert the number in binary:
  $41_{10} = 101001_2$

Try it

1. Convert in decimal form the result from last example and check it out ($41_{10} = 101001_2$). Break up steps in the process

2.6 Real numbers

In this subsection is tried to explain rationals numbers. That is to say, numbers with whole part and real part. We have 2 kinds of representation: fixed point and floating point.

2.6.1 Fixed Point

In this representation, we have a number of bits for whole part and other quantity of bits in order to represent real part. The dot of the number will remain fixed. For example:if we represent in 8 bits, we can use 5 bits to represent whole part and 3 for real part:

$$b_7b_6b_5b_4b_3, b_2b_1b_0$$

If we have the following number in last representation: 01101001
In decimal form is:
$01101,001 = 0*2^4+1*2^3+1*2^2+0*2^1+1*2^0+0*2^(-1)+0*2^(-2)+1*2^(-3) = 4+4+1+0.125 = 13,125$
One of the benefits is that we can follow the same rules of binary algebra in order to add o sustract numbers with this representation:

$$\begin{array}{r} 10110{,}101 \\ +00010{,}010 \\ \hline 11000{,}111 \end{array}$$

Due to the fact that it is not possible to change the position of the dot, you need to know beforehand, the interval of numbers that it is going to represent. This makes this reprsentation less flexible.

2.6.2 Floating point

To represent a widely range of numbers, Floating point representation is used. As a general rule, any real number can be represented this way:

$$N = M * B^E$$

where:

- M: mantissa
- B: base
- E: exponent

We will try the representation. For example:

Represent in Floating point 125,75

$125,75_{10} = 1111101,11_2 \Rightarrow 0,111110111 * 2^7{}_2 \Rightarrow 0,111110111 * 2^{111}{}_2$

- M: mantissa $\Rightarrow 0,111110111$
- B: base $\Rightarrow 10$
- E: exponent $\Rightarrow 111$

To avoid confusion, dot is placed on the left of the most significant digit. In addition, exponent is represented with Excess-Z representation and mantissa is represented with sign and magnitud.

Represent in Floating point normalized

- M: mantissa $\Rightarrow 101,1100$
- B: base $\Rightarrow 10$
- E: exponent $\Rightarrow 100$
- Number: $\Rightarrow 101,1100 * 2^4$
- Mantissa $\Rightarrow 0,101111 * 2^3$
- Exponent $\Rightarrow 4 + 3 = 7 \Rightarrow Excess - Z(4bits)7 + 2^3 = 7 + 8 = 15 \Rightarrow 1111$

Mantissa: 0,101111 Exponent: 1111

IEEE 754 is one of the most normalised format used. There are 3 formats of this representation: double precision, simple presicion and half precision with 64, 32 and 16 bits. In this normalised representation:

- Excess-Z is used. The excess is $2^{n-1} - 1$
- mantissa is represented with implicit leading bit. Mantissa is normalized, as it has been said before. But the bit on the right of dot is not taken into account and not represented, but everybody knows that it is there.

Try it

1. Search information about IEEE 754 and represent the following numbers in IEE754 double precision:

   $-12,3215$

   $-8,25$

3 Unit of information and its system

It is well known, the basic unit of information is the bit, and its values can be 0 or 1. However, in computer science, it is used groups of bits. So, 8 bits make 1 byte.

To make things easier, they are used in bigger groups, but always multiples by power of 10. As it said by the International System. So, it is possible to find the following concepts: Kilobyte, Megabyte, terabyte, ... and the like. But in computer science, the power of 2 is always used. And it is possible to find another different scale:

You should bear in mind that, when capital B is expressed in abbreviations, it is talking about bytes, groups of 8 bits. But when we express with b (lowercase), it is talking about bit.

International System	International System	Binary System	Bynary System
Kilobyte (KB)	$10^3 = 1000bytes$	$2^{10} = 1024bytes$	Kibibyte (KiB)
Megabyte (KB)	$10^6 = 1000KB$	$2^{20} = 1024^2bytes$	Mebibyte (MiB)
Gigabyte (GB)	$10^9 = 1000MB$	$2^{30} = 1024^3bytes$	Gibibyte (GiB)
Terabyte (TB)	$10^{12} = 1000GB$	$2^{40} = 1024^4bytes$	Tebibyte (TiB)
Petabyte (PB)	$10^{15} = 1000TB$	$2^{50} = 1024^5bytes$	Pebibyte (PiB)
Exabyte (EB)	$10^{18} = 1000PB$	$2^{60} = 1024^6bytes$	Exbibyte (EiB)
Zetabyte (ZB)	$10^{21} = 1000EB$	$2^{70} = 1024^7bytes$	Zebibyte (ZiB)

Table 8: Unit System of information unit