CN-SCC131

Science of Computing and Communication

Digital System

About me

- Jie Liu, office 303 SD or SX706
- jieliu@bjtu.edu.cn 刘杰
- My research is on Medical image processing,

  Computer vision

Overview of content

Hardware: Number systems/logic gate/ computer architecture (term 1)

Assembler(汇编语言) programming (term 1 + 2)

Operating System concept (term 2)

Course Aims

Understanding hardware of digital system (This semester)

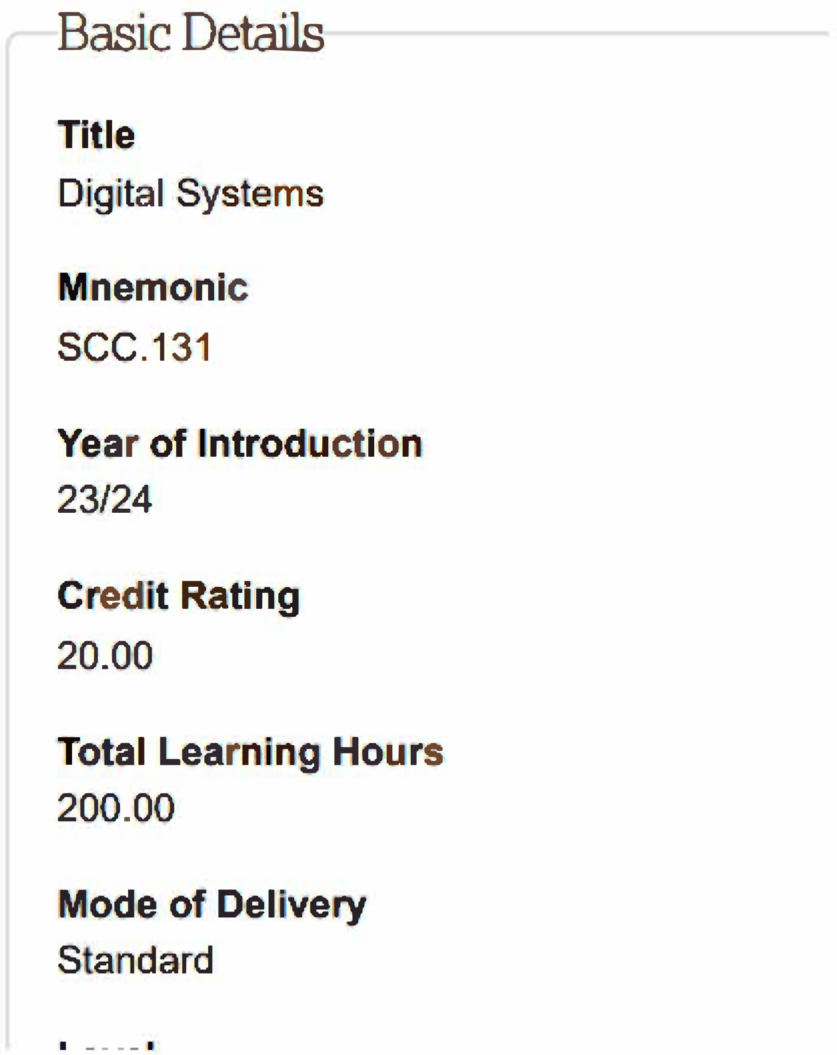
From fundamental concepts and components to whole computer systems

Understanding how to program at low levels that are "close to the machine":

-Assembly language (汇编语言)

Understanding how OS control hardware and software

-C programming(next semester)



Basic Details

Title
Digital Systems

Mnemonic
SCC.131

Year of Introduction
23/24

Credit Rating
20.00

Total Learning Hours
200.00

Mode of Delivery
Standard

Indicative Syllabus for Prospective Students (between 100 to 250 words)

The creation of the microprocessor revolutionised global innovation and creativity. Without such hardware we would have no laptops, no smartphones, no tablets. Life changing technologies from MRI scanners to the Internet would simply not exist. This module provides an introduction to the field of Digital Systems – the engineering principles upon which all contemporary computer systems are based.

We study the elements that work together to form the architecture of digital computers, including computer processors, memory, data storage and input/output. We also unearth the ways in which these are enabled by digital logic – where George Boole's theory of a binary based algebra meets electronics. Building on SCC.111 we also discover how the software programs we write translate to, and interact with, such hardware. Finally, we also study the effects of multi-process operating systems, and how these interplay with the capabilities and architecture of modern computers to optimise performance and robustness.

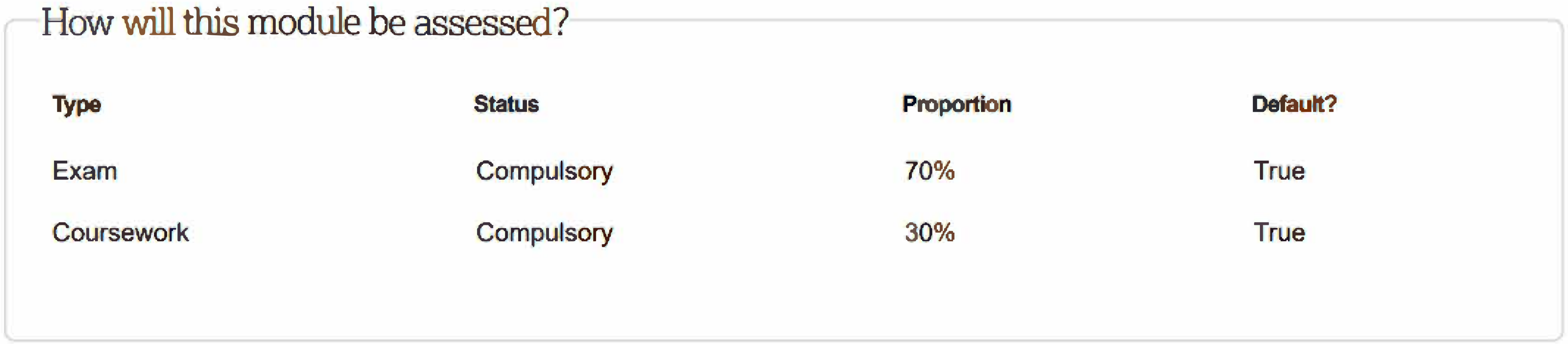
Educational Aims: Subject Specific

This primary aim of this module is to demystify the operation of contemporary computers, such that students can begin to make reasoned judgements about the behaviour, capabilities and real-world limitations of computer systems. Furthermore, it aims to instil a basic understanding of computer architecture, data representation, operating systems concepts and how these relate to the underlying theoretical concepts of digital logic.

[Learning Outcomes] Learning Outcomes: Subject Specific

On successful completion of this module students will be able to

- Describe the role and operation of the primary hardware components of modern computer systems, and how they are built upon the principles of digital logic - including processors, memory and input/output.
- Demonstrate how the principles of high-level imperative programming languages are translated into low-level machine instructions, data structures and binary representations in a computer's memory.
- Relate the components that form the structure of an operating system, and its associated system software.
- Recognize the benefits of multi-process environements, and discuss the need for the resource management provided by operarting systems schedulers and basic mutual exclusion mechanisms.
- Apply software development concepts to low-level programming languages, such as C and assembler.



How will this module be assessed?

Type	Status	Proportion	Default?
Exam	Compulsory	70%	True
Coursework	Compulsory	30%	True

High level Language C

```
void main(){
if (i==j) {
f=g+h;
}
else f=g-h;
}
```

Compiler

Assembler language

```
      beq $s3, $s4, Then
      sub $s0, $s1, $s2
      j Exit
Then:    add $s0, $s1, $s2
Exit:
```

Mnemonic symbol (助记符)

Assembly

Assembler

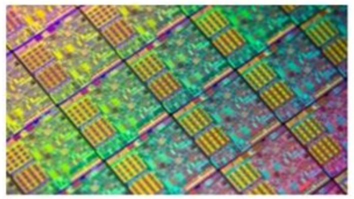
Machine language

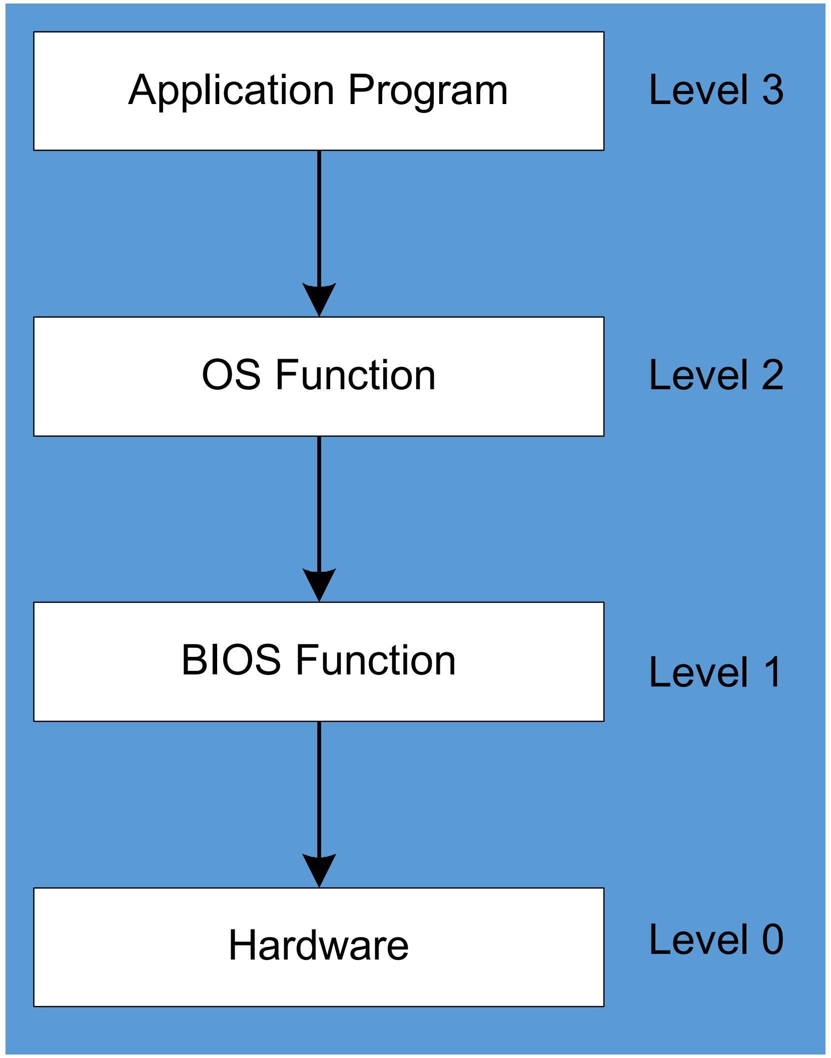
```
10001101001010000000010010110000
00000010010010000100000000100000
10101101001010000000010010110000
```

Machine code

Operating system

Hardware




Application Program
Level 3
OS Function
Level 2
BIOS Function
Level 1
Hardware
Level 0

Hardware

(week1-week7)

- Basic component and principle of computer hardware
- Computer organization and architecture(计算机组成与体系结构)
  - Data encoding(编码)
  - Boolean logic gates(布尔逻辑门)
  - ALU unit: Adder 加法器
  - Memory unit: Flip-flop(触发器), counter(计数器)
  - Control unit
  - I/O unit (Input/Output):Interrupt, polling
  - Microcontroller 单片机

Assembly Programming

(week9-week10)

- Brief introduction to assembly
- Instruction set of X86
- Register
- Assembly programing
- ………

Operating system concept

- OS structure
- Process
- Threads & Concurrency
- CPU Scheduling
- Process synchronization
- Memory management
- Storage management
- I/O Systems
- File System
- Security and protection

Reference book

Computer Organization and Architecture 10th - William Stallings

Assembly.Language.For_.x86.Processors.Kip_.R..Irvine..6ed.

Operating System Concepts 10th Edition

Topic 1: Data Encoding

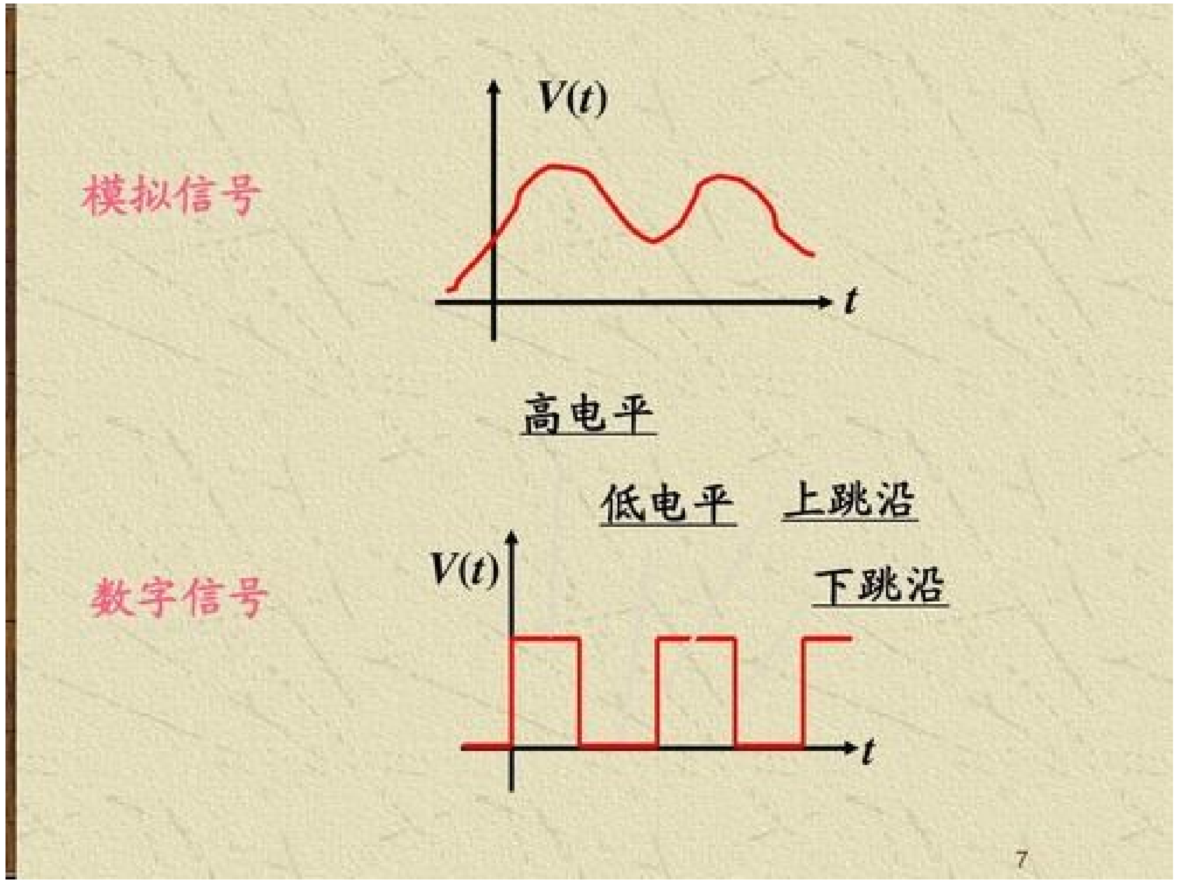
Digital system

Digital System is a system in which signals are discrete in time and magnitude
In this course, it refer to computer system

digital signal

The most fundamental of these systems is a binary system, which simply represent various information in a series of binary number, traditionally ones and zeros, or "on" and "off" values.

Analog and digital signal



Analog signal :Continuous

Digital signal:Discrete

1. What is data encoding?

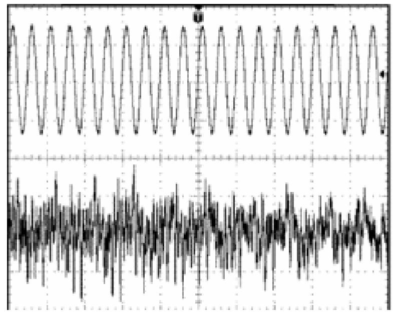
- **Definition**
- Encoding is the process of converting data into a format required for a number of information processing needs.
  e.g. a letter -> binary number , an analog sound ->digital file

- Why need to encode in computer system?
  For computer processing
- How to encode in computer system?

Use binary numbers to represent data in computer

Data type form

- Integer, 1,2,3,4
- Negative,-1,-2
- Real number: float, double, 1.2, 2.3
- Char, character and symbol, A,B,C,…@,%,~….
- Image, sound…..



Binary number

Number system: binary, decimal(10), hexadecimal (16), octal(8)

Why binary number in computer?

Binary circuit is easy to implement in hardware

Data encoding

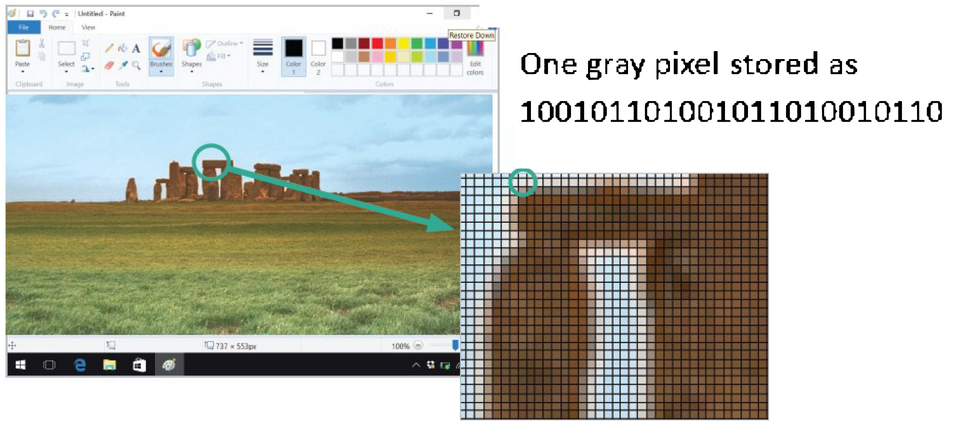
- Integer: positive

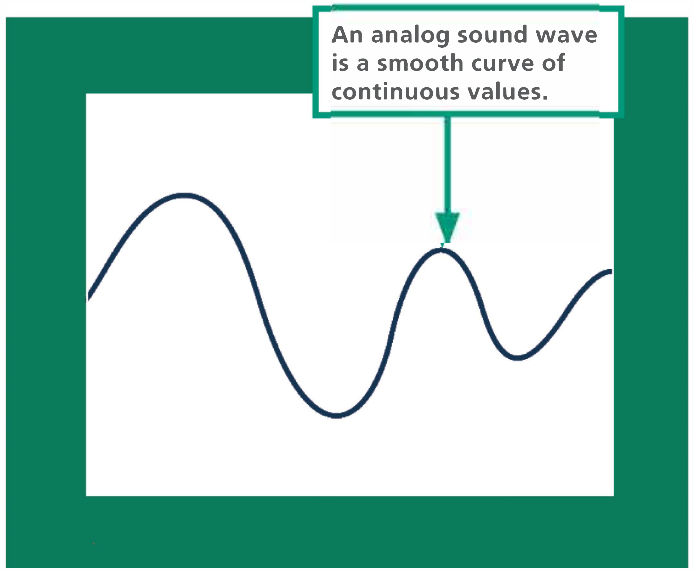
  direct binary conversion
- Negative:

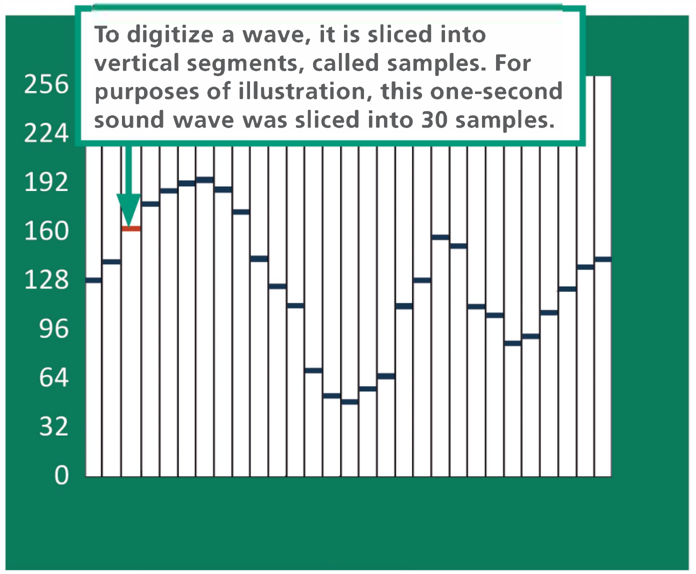
  2’s complement
- Real number:

  scientific notion
- Character and symbol:

  ASCII, Unicode
- Digital image, sound,




An analog sound wave is a smooth curve of continuous values.



SAMPLE	SAMPLE HEIGHT (DECIMAL)	SAMPLE HEIGHT (BINARY)
1	130	10000010
2	140	1000110
3	160	10100000
4	175	10101111

The height of each sample is converted into a binary number and stored. The height of sample 3 is 160 (decimal), so it is stored as its binary equivalent—10100000.

Binary number

base of radix is 2, the digit is either 0 or 1

binary -> decimal

-Digit Dn: 1, 0

$D_n \ldots D_3 D_2 D_1 D_0$

-Weight: 2k

Calculate the sum of product of each bit multiply with its weight

The value of a decimal number $= D_n 2^n + \ldots + D_3 2^3 + D_2\, 2^2 + D_1\, 2^1 + D_0\, 2^0$

Hexadecimal (Base 16, base 10 + ABCDE)

$4A9.2B_{hex}$

$4\times16^{2}+10\times16^{1}+9\times16^{0}+2\times16^{-1}+11\times16^{-2}=1193.1679$

Decimal Number (base 10)

$$\begin{tikzcd}
6 \arrow[d] & 2 \arrow[d] & 3 \arrow[d] \\
6 \text{ x } 10^{2} \arrow[d] & 2 \text{ x } 10^{1} \arrow[d] & 3 \text{ x } 10^{0} \arrow[d] \\
600 \quad + & 20 \quad + & 3 \quad = 623_{10}
\end{tikzcd}$$

10^{n} : Place value or weight

Subscript(下标10)

Number System

- Decimal: 10

- Binary:

  1010_b

- Hexadecimal:

  0XA

- Octal:

  12_8 （0， 1， 2， 3， 4， 5， 6， 7）

  -One octal correspond 3 bits

Why hex?

$0xFFFF=(1111111111111111)_b$

one hex digit corresponds to a group of 4 bits

Conversion of number

- Non decimal to decimal
- Calculate the sum of product of each bit multiply with its place value

- Decimal to non decimal (integer)
- Repeatedly dividing the decimal number by the base till the decimal number become zero
- Read remainder from last to first

- For fraction, multiplied by base until fraction part zero
- Read the integer part from first to last

Example

$101001_{binary} = 41_{decimal}$

$101001.0101_{binary} = 41.3125_{decimal}$

$255(\text{decimal}) = (11111111)_b$

$1567_{decimal} = 61F_{hex}$

- $0.75\ (\text{decimal}) = (0.11)_2$
- $0.25\ (\text{decimal}) = (0.01)_2$

3. Negative number

- Sign and magnitude(符号和幅度)
- Excess n (余数 n)
- One’s complement(反码)
- Two’s complement(2 补码)

Sign and magnitude

- Left most bit → sign bit, 0:positive, 1:negative
- Given 4 bits
- 5
- 0101(原码, true code)

- -5
- 1101

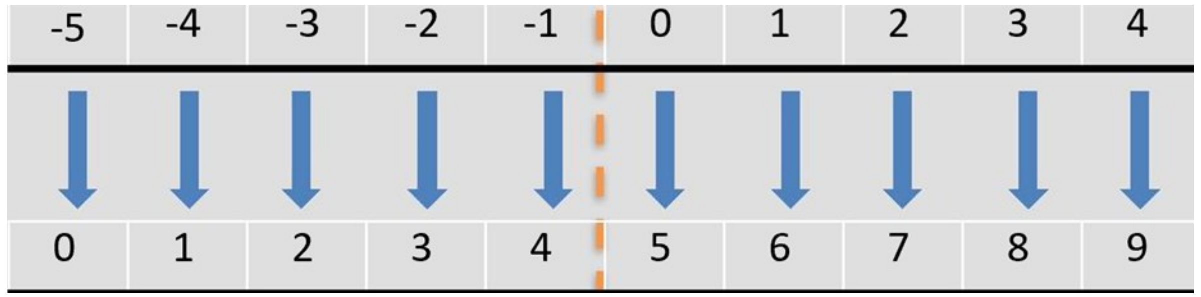
Problems

- 2 zeros, +0, -0
- $2+(-1) \neq 1$, wrong   0010+1001=1011=-3

Excess N(余数N)

Excess: an amount of something that is more than necessary, permitted, or desirable.
N: Offset value or bias

We code (i.e. store) a number by adding the excess to it

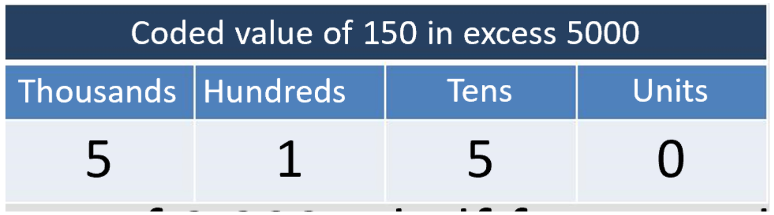


Example: code values in the range -5..+5 using excess 5…

We decode a representation by subtracting the excess from it

Overall range： -5000, …, -1, 0, …, 4999

Excess N=5000, 150



Coded value of 150 in excess 5000			
Thousands	Hundreds	Tens	Units
5	1	5	0

Problems

- -4, -3, -2, -1, 0, 1, 2, 3 (Excess 4)
- 0, 1, 2, 3, 4, 5, 6, 7
- 000, 001,010, 011, 100, 101, 110, 111

- 0+(-1) excess 4
- 100+011
- =111
- =3

- 0-1=3? Wrong

2’s complement, e.g,-5

- Represent the equivalent of positive binary number
- Flip each bit
- Adding 1 to the number

反码 ones-complement code

-
  - -5: 0101→1010→1011
  - Why 2’s complement
  - 1-1=0 (1+FF=0); 3-2=3+(-2)=1

(0011+1110=1)

How to calculate true code for 2's complements code

- -1
- Negate each bit

- Or

- Negate each bit
- Add 1
- 1011→0100→0101=5
- -5

4 bits

Positive:0-15
unsigned

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

2’s complement
-8→7

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
-8	1000
-7	1001
-6	1010
-5	1011
-4	1100
-3	1101
-2	1110
-1	1111

2’s complement
in order

-8	1000
-7	1001
-6	1010
-5	1011
-4	1100
-3	1101
-2	1110
-1	1111
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

- C data type
  unsigned char(8 bits): 0-255 range: 2^8-1

- char: (2’s complement)

10000000(-128),..., FF,0,… ,01111111(127)   range: $-2^7 \rightarrow 2^7-1$

- $-2^{N-1} \rightarrow -2^{N-1}-1$

- int: $-2^{31} \rightarrow 2^{31}-1$

4. How to represent character

(ASCII) is a widely-used code for characters

American Standard Code for Information Interchange

It defines 128 (i.e. 2^7) symbols

A..Z: 65..90, nul : end of character

Latin-1: another character-code standard

8-bit code that extends ASCII from 128 to 256 values (0-255)

Unicode: a further extension(统一码)

- 16-bit code that extends Latin-1 from 256 to 65536 values (0-65535)

- public class Excrise_3 {
- public static void main(String[] args) {
- char a='啊';
- int v=(int)a;
- System.out.println(v);}
- output： 21834

5. Real Number(实数)

- 1.25
- float, double
- Fixed point number 定点数
- Float point number 浮点数

Fixed point(定点小数)

The position of point is fixed

We simply reserve some columns for the fractional part

e.g., 1798.059

Thousands	Hundreds	Tens	Units	Tenths	Hundredths	Thousandths
n x 10^3	n x 10^2	n x 10^1	n x 10^0	n x 10^{-1}	n x 10^{-2}	n x 10^{-3}
n x 1000	n x 100	n x 10	n x 1	n x .1	n x .01	n x .001

But there’s a problem with fixed point

We rapidly run out of columns!

Let’s say we use 8 columns for numbers, and reserve half of these for fractional parts:

Scientific notion-float number

- Real number is represent by exponent(指数) and mantissa(尾数)
- represent 0.00015 as 1.5 x 10^{-4}
- represent 150000 as 1.5 x 10^{5}
- represent 30..0 as 3.0 x 10^{20} , 0.000…3, 3.0 x 10^{-20}
- Scientific notion: 科学表示法，float : point is not fixed
- M x 10^{E}

.

why float point

- Only represents exponent and mantissa
- Efficient, save space
- Point position is not fixed but floating; Position is decided by exponent

```
float x=12.33;
```

IEEE 754 Floating Point

- This is the standard floating point representation that is used by almost all modern computers

- The mantissa is coded using sign and magnitude

- The exponent is coded in excess n(余数 n)
  - For an exponent held in b bits, $n = (2^{b-1}) - 1$
    $[\ (2^{8-1}) - 1 = 2^7 - 1 = 127]$

So, use excess 127 for an 8-bit exponent...

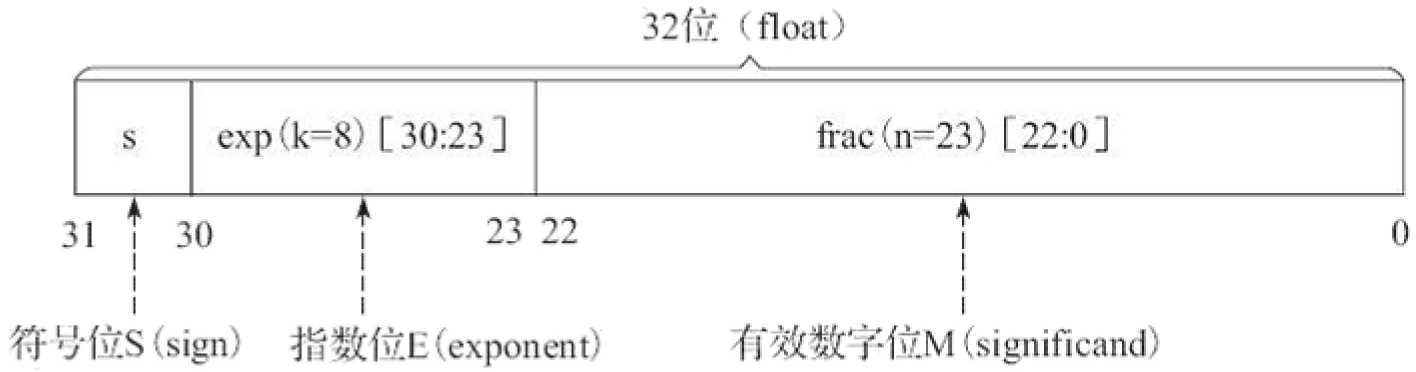
Normalizing(规范化) the mantissa: a space saving optimization (saves 1 bit)

- normalizes the mantissa to xxx... and just stores the fractional part(not 1xx.xxx form)
  - Leaves the “1.” part as implicit: there’s actually no need to store it explicitly! By default(默认) Float: Real position of point is dependent on exponent (It is always possible to normalize such that the most significant [first] bit is a 1)

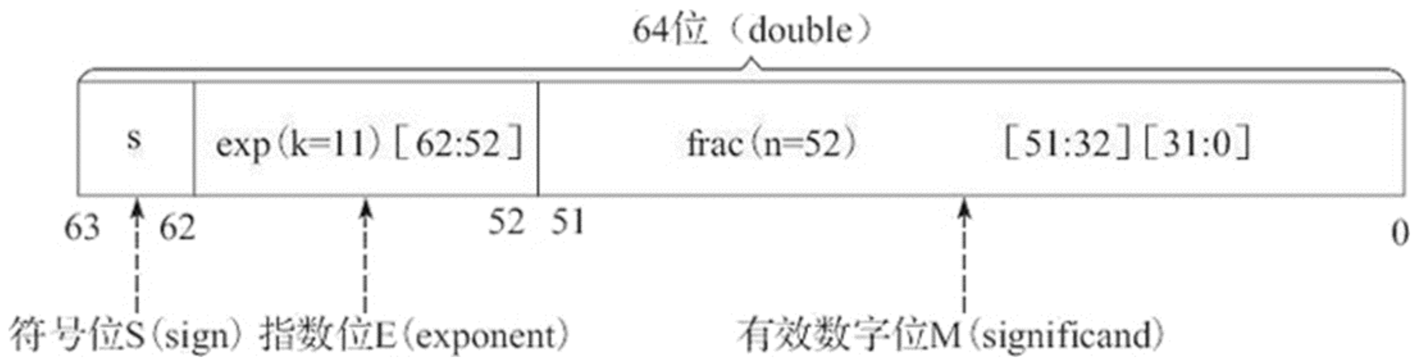
$= (-1)^{sign} \times 2^{(exponent - 127)} \times (1 + mantissa)$ explicitly represented

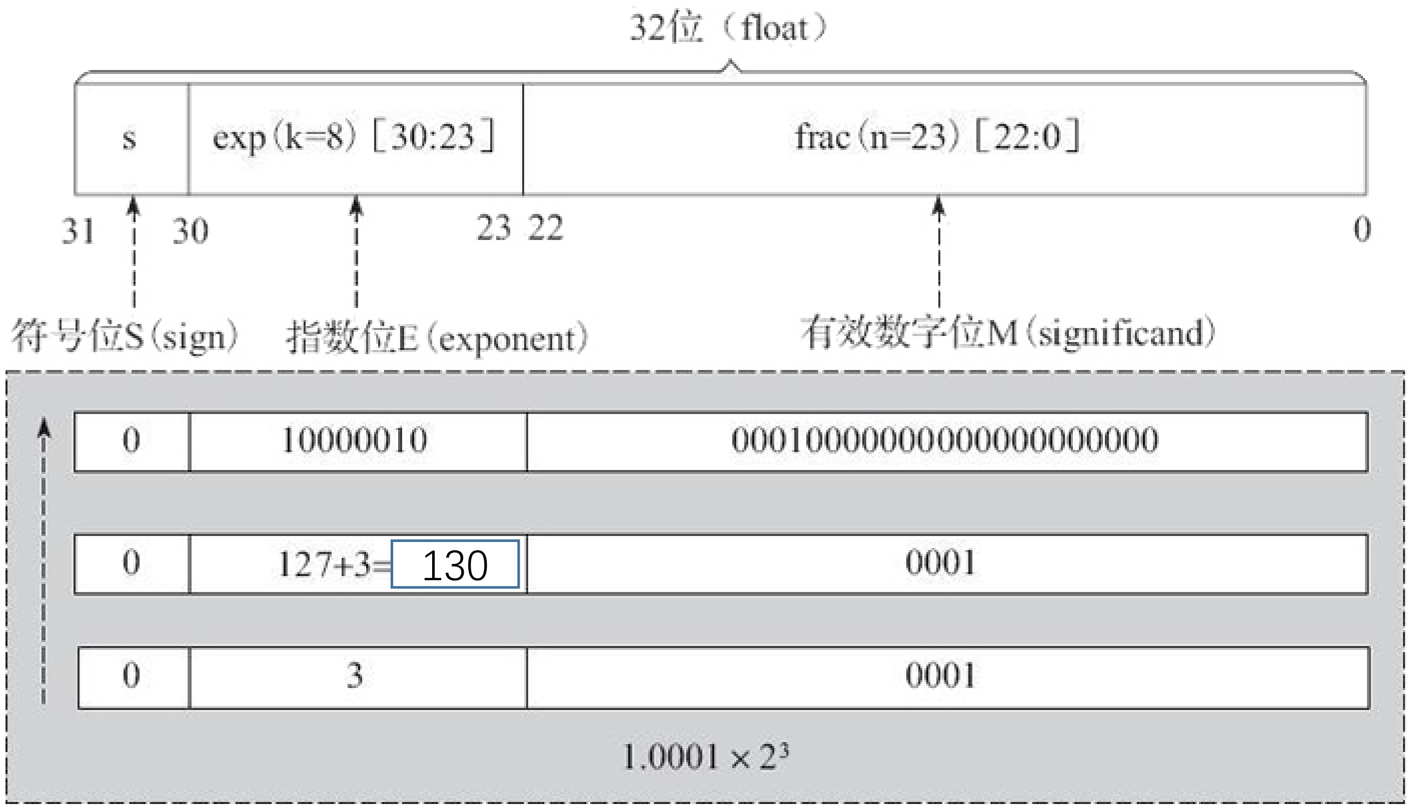
Float number

Sign bit
1: -
0: +

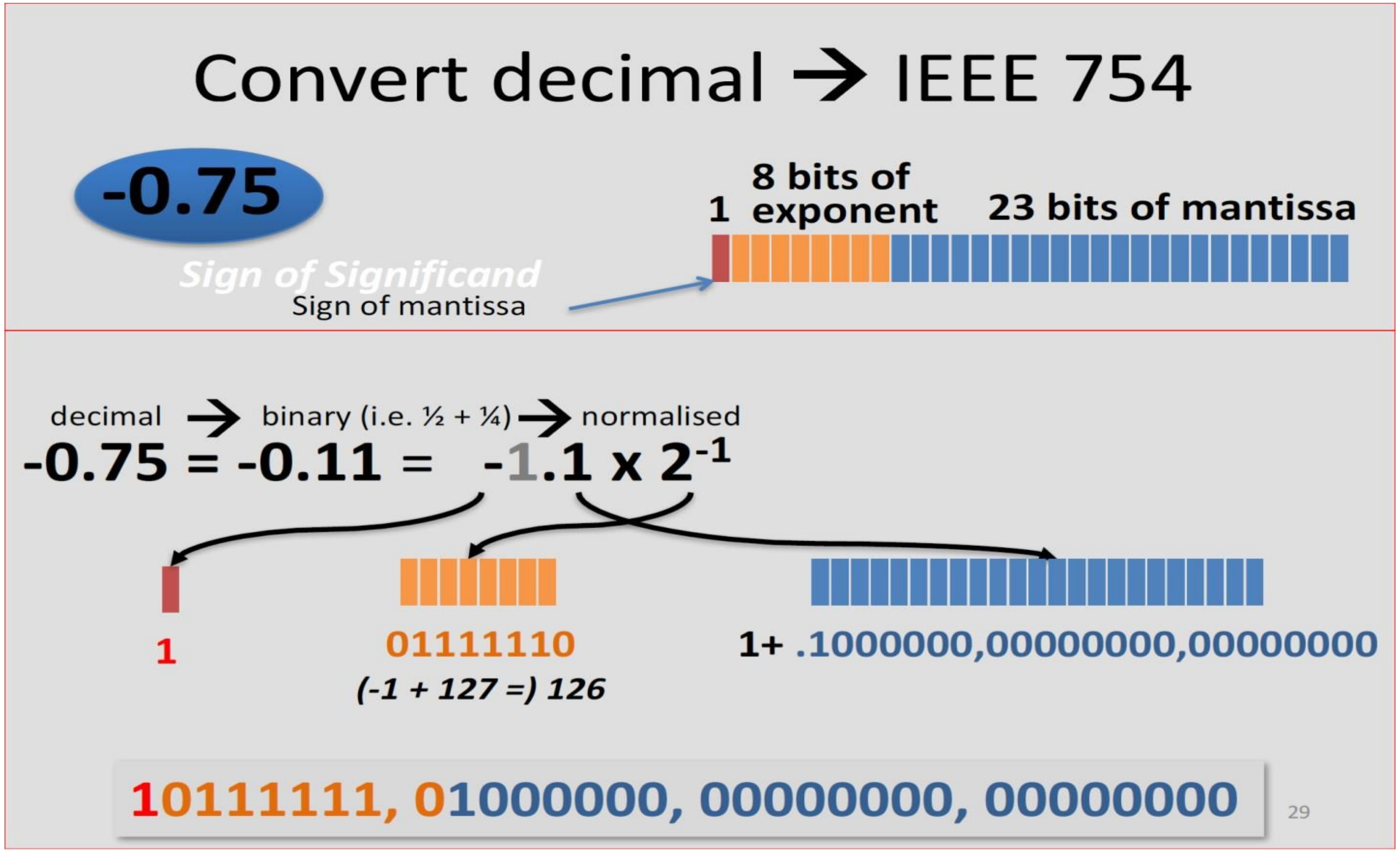


Double number




32位（float）
s
exp (k=8) [30:23]
frac (n=23) [22:0]
31
30
23
22
0
符号位S (sign)
指数位E (exponent)
有效数字位M (significand)
0
10000010
00010000000000000000000
0
127+3=
130
0001
0
3
0001
1.0001 × 2³

-0.75, 7.375



Convert decimal → IEEE 754

-0.75

decimal → binary (i.e. ½ + ¼) → normalised

$-0.75 = -0.11 = -1.1 \times 2^{-1}$

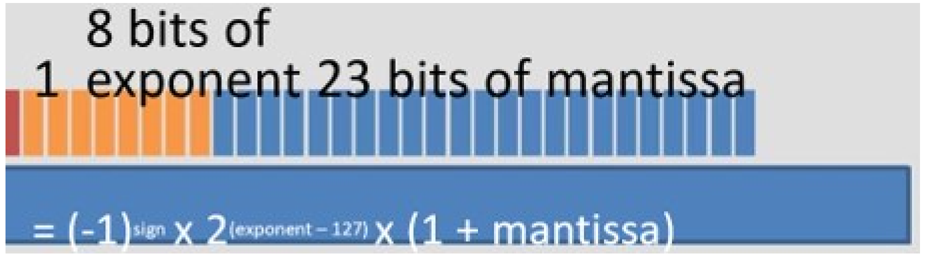
10111111, 01000000, 00000000, 00000000

-0.75

Convert IEEE 754 decimal

Decoding

01000010, 10101010, 00000000, 00000000



$= (-1)^{\text{sign}} \times 2^{(\text{exponent} - 127)} \times (1 + \text{mantissa})$

Convert IEEE 754 decimal

01000010, 10101010, 00000000, 00000000

Assume 1.xxx

+ve 10000101 1 + .0101010, 00000000, 00000000

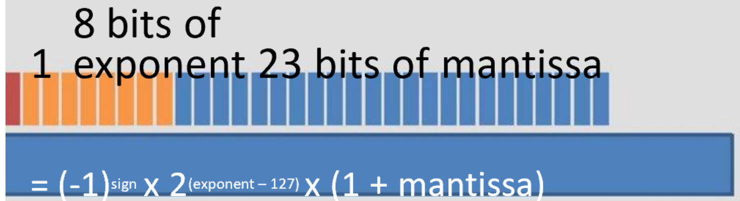
128 + 4 + 1 = 133

133 – 127 = 6

1+¼+ 1⁄16 +1⁄64

= 1 + 0.25 + 0.0625 + 0.015625

1.328125



$= (-1)^{\text{sign}} \times 2^{(\text{exponent} - 127)} \times (1 + \text{mantissa})$

1.328125×2^{6}

$= +1.328125 \times 64$

= 85decima l

```
float x=-0.75;

System.out.println(Float.floatToIntBits(x));//get decimal


System.out.println(Integer.toBinaryString(Float.floatToIntBits(x)));//get binary


1061158912

111111010000000000000000000000

-0.75?
```

- float data range

- 2^{-126}-2^{127} (-127 and128 are used for special number)

  (-127 means zero or near zero)

- Maximum: positive
- $+(1.11111111111111111111111)\times 2^{127}\approx 3.402823\times 10^{+38}$

- Minimum positive
- $+(1.0)\times 2^{-126}\approx 1.175494\times 10^{-38}$

- Ngative maximum： S=1
- -(1.0)$\times 2^{-126} \approx -1.175494 \times 10^{-38}$

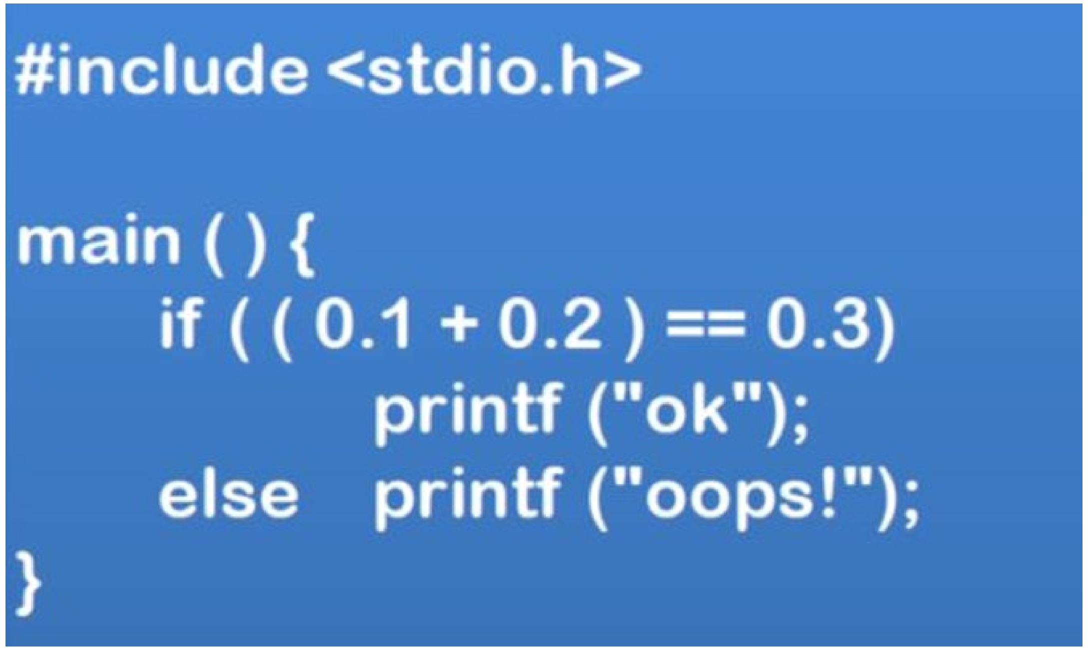
- Negative minimum
  S=1，127，
- -(1.11111111111111111111111)$\times 2^{127} \approx$-3.402823$\times 10^{+38}$

Special” values in IEEE 754

Meaning	Exponent	Mantissa
Zero	0	0
Infinity (∞)	2^8–1 (all 8 bits set)	0
Not a Number (NaN)	2^8–1 (all 8 bits set)	Non zero

- E = 255 M=0， ±∞
- +∞： 0 11111111 000 0000 0000 0000 0000
- -∞： 1 11111111 000 0000 0000 0000 0000

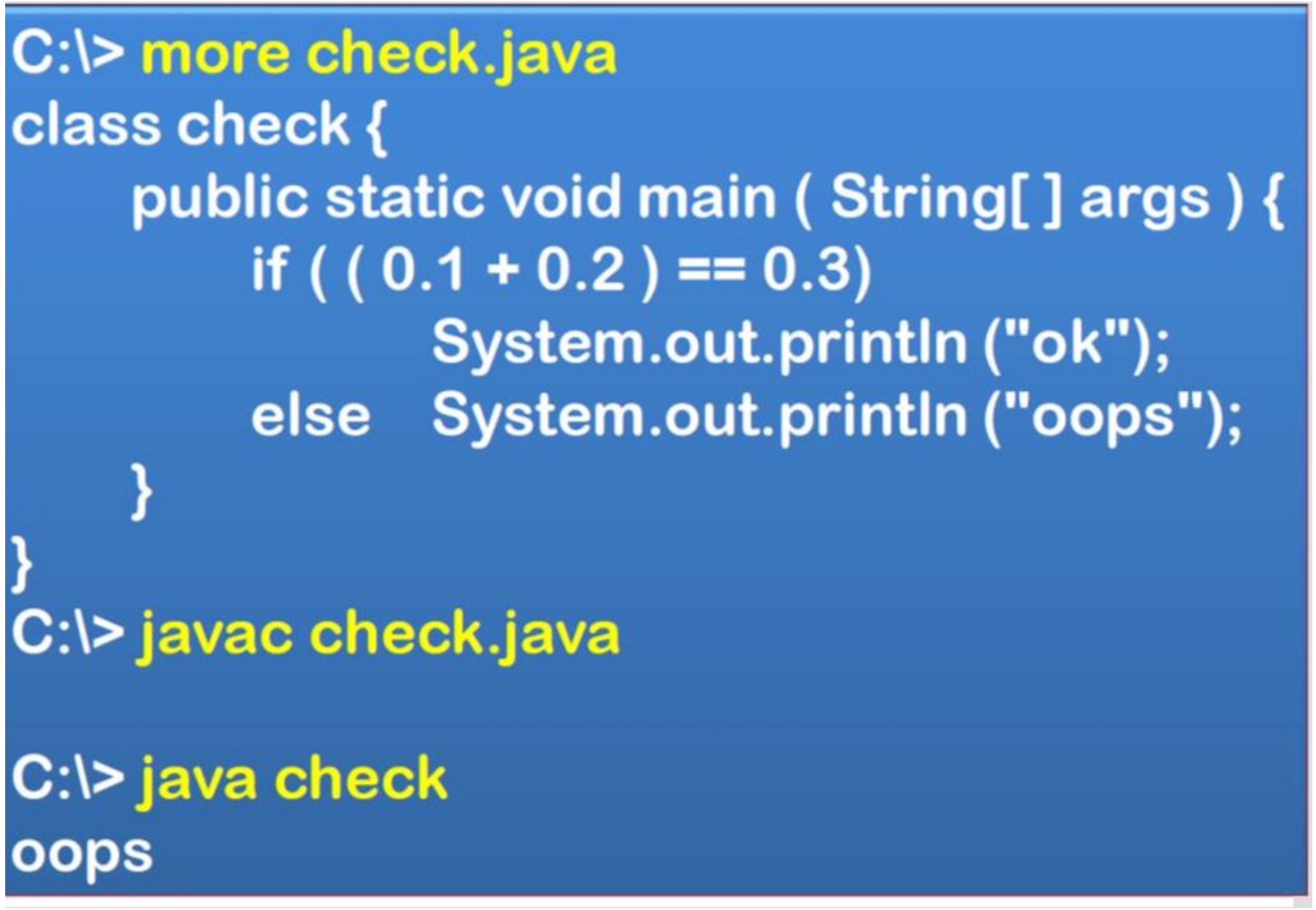
- 0.1 + 0.2 = 0.3 ?



```
#include <stdio.h>

main ( ) {
        if ( ( 0.1 + 0.2 ) == 0.3)
                printf ("ok");
        else    printf ("oops!");
}
```

Output: oops!



```
C:\> more check.java
class check {
      public static void main ( String[ ] args ) {
            if ( ( 0.1 + 0.2 ) == 0.3)
                  System.out.println ("ok");
            else  System.out.println ("oops");
      }
}
C:\> javac check.java

C:\> java check
oops
```

0.1+0.2=0.30000000000000004
false

Equality testing with floating point numbers

– A possible rounding error resulting from lack of precision means that we can never be confident that two numbers that “should be” equal are actually coded as equal…

- Instead, we should take a cautious approach:

```
if (abs(x–y) < error_tolerance) {…}
```

- 0.125+0.25=0.375?
- 0.5+0.25=0.75?
- 1+0.5=1.5?

6. 4-bit BCD

- Binary Coded Decimal, 4 bit binaries correspond to a decimal number

Decimal digits	BCD code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Simple way to convert decimal number to binary number

5327： 0101 0011 0010 0111

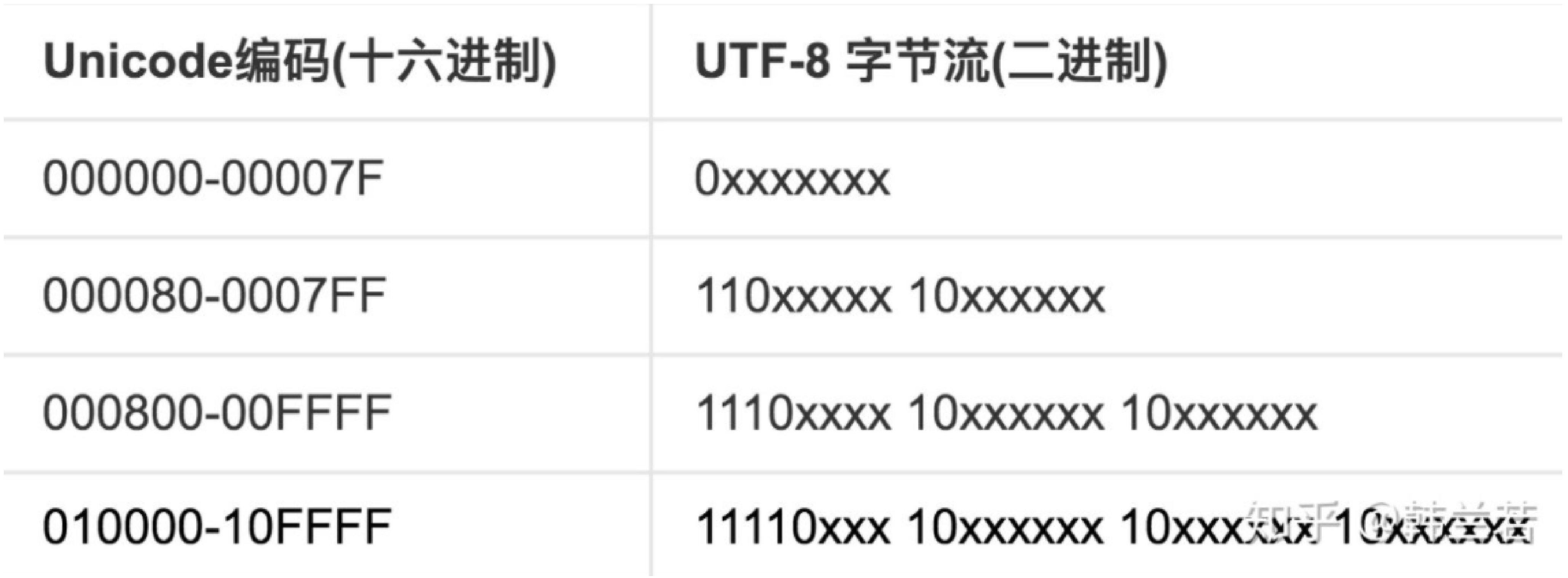
One decimal→4 bits for display

UTF-8

- UTF-8
- Basic unit: 8 bits
- Universal Character Set/Unicode Transformation Format)
- Use 1-4 bytes to convey, variable-length mode

Unicode

- Use 16 bits to express more character



Unicode编码(十六进制)	UTF-8 字节流(二进制)
000000-00007F	0xxxxxxx
000080-0007FF	110xxxxx 10xxxxxx
000800-00FFFF	1110xxxx 10xxxxxx 10xxxxxx
010000-10FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

Added range

『汉』0110 1100 0100 1001

UTF-8 used in internet

Summary

- Number system(2, 10, 16)
- Data encoding: Integer, negative, real number, text
- Conversion between them(2, 10, 16)

Exercise I

1: Convert decimal number into binary number
e.g. 18, 114, -59
2: Convert binary number into decimal number
e.g. 101010, 100101
3: Convert the real number into binary number
e.g. 0.5, 3.75, -1.25
4: Convert real number into IEEE 754 floating point form(optional)
e.g. 1.5,-0.5, 0.2

5： Write a program(Java or c) to complete
Print character ‘A’ : ASCII value; print ‘北’: Unicode value
Print 0.1+0.2 and test 0.1+0.2=0.3?
Calculate -4 binary number in 2’s complement