数据的表示与运算

考纲要求：

1. 数制与编码：
    1. 进位计数制机器数据之间的相互转换
    2. 定点数的编码表示
2. 运算方法和运算电路
    1. 基本运算部件：加法器，算数逻辑部件ALU
    2. 加减法运算：补码加/减运算器，标志位的生成
3. 整数表示和运算
    1. 无符号整数的表示和运算
    2. 带符号整数的表示和运算
4. 浮点数的表示和运算
    1. 浮点数的表示：IEEE754标准
    2. 浮点数的加减运算

进位计数制

二进制B、八进制Q（或O）、十进制D、十六进制H（或0x）

r进制数D转为十进制数N：
`N = d(n) × r^(n-1) + d(n-1) × r^(n-2) + ... + d(1) × r^0 + d(-1) × r^(-1) + ... + d(-m) × r^(-m)`,
d(n)表示D对应位数的数值

二进制转八进制、十六进制：
每3位二进制，相当于1位八进制；
每4位二进制，相当于1位十六进制；
三十二进制、六十四进制依此类推

十进制转二进制（通用方法）：
整数部分÷2取余，余数逆序（从下往上）位数递减；
小数部分×2取整，整数顺序（从上往下）位数递减

十进制转二进制（拼凑法）：
根据`2^n`依次减去对应的二进制数的十进制数，余下的差依此类推，各个n则为1所在的二进制位数

真值和机器数

真值：±和某进制数绝对值的形式成为真值；
机器数：符号数码化的数为机器数，例如0/1表示+/-

计算机使用的数据可分为符号数据（ASCII码、汉字、图形等）、数值数据（定点、浮点）。

数据格式

定点数:
约定机器中所有数据的小数点位置固定不变，由于约定在固定位置，小数点不再用“.”表示。
定点表示数的范围受字长限制，且精度有限

1. 定点纯小数：符号+整数部分+小数部分
2. 定点纯整数：符号+数值

浮点数:
小数点的位置随阶码的不同而浮动，表示方式：`N=R^E·M`，例如233*10^3，`(1.75)10 = 1.11*2^0 = 0.111*2^1 = 0.0111*2^2`
IEEE754规定了单精度float（32）和双精度double（64），由
`【32】数符s(1位)+阶码E（8位,移码）+尾数M(23位,原码)`或
`【64】数符s(1位)+阶码E（11位,移码）+尾数M(52位,原码)`
构成， 它们由二进制转为十进制的公式为：
`【32】D = (-1)^s × (1.M) × 2^(E-127)`
`【64】D = (-1)^s × (1.M) × 2^(E-1023)`
例如`(41360000)16`求十进制的过程为：

1. 转为二进制数：`0 1000 0010 011 0110 0000 0000 0000 0000`;
2. e = 阶码-127 = 100000010 - 01111111 = 00000011 = (3)10;
3. 包含隐藏位1的尾数：`1.M = 1.01101100000000000000000 = 1.011011`;
4. 由2、3得：(-1)^s × 1.M × 2^e = +(1.011011)×2^3 = +1.011.011 = (11.375)10

数的机器码表示

源码表示法:
用第一位表示±，其余部分为二进制表示的数值本身;
简单易表示，乘除运算简单，加减运算麻烦。
定点小数`x0.x1x2x3...xn`，例如：
x = +0.11011，x源码 = 0.11011
y = -0.11001，y源码 = 1.11001
定点整数`x0x1x2x3...xn`，例如：
x = +11011，x源码 = 011011
y = -11001，y源码 = 111001

补码表示法:
正数的补码和源码相同，负数的补码是正数的反码每一位取反 并在末位加1;
将加法运算转换为加法运算：`[x]补 - [y]补 = [x]补 + [-y]补`;
没有正零和负零之分。
定点小数`x0.x1x2x3...xn`以`2`为模
定点整数`x0x1x2x3...xn`以`2^(n+2)`为模
【注：计算时最高1位若超过字长，需丢掉】
变形补码:
双符号补码，为了防止溢出

反码表示法:
正数的反码和源码相同，负数的反码是对应正数的源码每一位取反;
电路容易实现，触发器的输出有正负之分。

移码表示法:
通常用在阶码种，定点整数定义为`[x]移 = 2^n + x, 2^n > x ≥ -2^n`;
只能用于整数。

字符和字符串（非数值）的表示方式

编码的各个字符通常以`00H`（也就是`\0`）作为结尾

ASCII码： 用一个字节表示，低七位用于编码(128)，最高位为校验位。
字符串占用主存中连续多个字节，每个字节存储一个字符。

- 0：48，30H
- A：65，41H
- a：97，61H

汉字编码:
GB2312-80：汉字+各种符号共7445个。

C语言中的整数类型转换

有符号数和无符号数的转换:
强制类型转换的结果保持位置不变，仅改变解释这些位的方式

```
int mian(){

   /*有符号转无符号*/
   short x = -4321;
   unsigned short y = (unsigned short)x;
   printf("x=%d, y=%u\n", x, y);   //x=-4321, y=61215

   /*无符号转有符号*/
   unsigned short x = 65535;
   short y = (short)x;
   printf("x=%u, y=%d\n", x, y);    //x=65535, y=1

}
```

不同字长的整数转换:

```
int main(){

   /*长转换为短*/
   int x = 165537, u = -34991;
   short y = (short)x, v = (short)u;
   printf("x=%d, y=%d\n", x, y);
   printf("u=%d, v=%d\n", u, v);
   //x=165537, y=-31071
   //u=-34991, v=30545
```

```
    /*短转换为长*/
    short x=-4321;
    int y=x;
    unsigned short u = (unsigned short)x;
    unsigned int v = u;
    printf("x=%d, y=%d\n", x, y);
    printf("u=%u, v=%u\n", u, v);
    //x=-4321(0xed1f), y=-4321(0xffffef1f)
    //u=61215(0xef1f), v=61215(0x0000ef1f)

}
```

32位系统下的各数据类型长度

数据类型	占据内存大小(byte)
short	2
int	4
long	4
float	4
double	8
char	1
	从大字长向小字长强制类型转换时，会把多余的高位字长截断，低位直接赋值。
	短字长整数到长字长整数转换，不仅要使相应的位值相等，高位部分还会扩展为原数字的符号位
	char类型转为8位ASCII码整数int时，在高位部分补0即可

短数据转为长数据:
正整数整数在高位扩展，小数在低位扩展；

负数的短数据转为长数据:

	负整数	小数
源码	原最高位挪到转换后的最高位，其余高位补0	低位补0
反码	高位补1	低位补1
补码	高位补1	低位补0

数据的存储和排列

小端方式 和 大端方式 存储:
较低的有效字节存放在较低的存储器地址（地址偏移越多的越低），较高的字节存放在较高的存储器地址。

大端存储反之。
例如存储(0x12345678)到int，OP0表示32位数据最高字节MSB，OP3表示32位数据最低字节LSB：

地址偏移	大端模式	小端模式
0x00	12(OP0)	78(OP3)
0x01	34(OP1)	56(OP2)
0x02	56(OP2)	34(OP1)
0x03	78(OP3)	12(OP0)

边界对齐 存储:
通常32位的计算机，可按照字节、半字和字寻址。
数据以边界对齐的方式存储，半字地址一定是2的整数倍，字地址一定是4的整数倍。
这样数据可以一次访存取出。
对于不满足的数据则填充空白字节。

定点加减法运算

先判断两数符号，同号相加、异号相减，绝对值大的减去小的，最后确定符号位

补码加减法

补码加法:
[x+y]补 = [x]补 + [y]补 mod 2 或 mod 2^(n+1)。
补码减法:
[x-y]补 = [x]补 - [y]补 = [x]补 + [-y]补。

从[y]补求[-y]补:
[-y]补 = ¬[y]补 + 2^(-n)，
¬表示对[y]补作包括符号位内的求反操作，
2^(-n)表示最末位的1。
负数求补码口诀:
从右向左，第一个1和第一个0保持不变，其它按位取反。

溢出:
定点小数中，数的范围是|x| < 1，如果大于1则为“溢出”，在定点机种通常不允许。
例如：俩正数相加变负数、俩负数相加变正数。
检测方法:
两个符号位看作数码参加运算；两数以4 或 2^(n+2)为模的加法，最高位符号位产生的进位需要舍弃；采用变形补码后，两数相加其结果符号位出现01或10组合是，表示溢出。
电路加法器中，当Cn = C(n-1)，运算无溢出；当Cn ≠ C(n-1)，运算溢出。

进位标志的生成：条件码:
标志寄存器由16位的存放条件标志、控制标志寄存器，用于反映处理器状态和ALU运算结果的某些特征以及控制指令的执行。
条件码:

- 溢出标志 OverflowFlag OF：==1时溢出

- 符号标志 SignFlag SF：==0表示正数，==1表示负数
- 零标志 ZeroFlag ZF：==1表示结果为0
- 进位/错位标志 CarryFlag CF：==1表示无符号数的加减法发生了错位/进位，即将发生溢出

浮点运算

浮点数加减

存在两个浮点数`x=2^(Ex)Mx`和`y=2^(Ey)My`，Ex、Ey是阶码，Mx、My是尾数，则它们相加减的规则是：`x±y = (Mx·2^(Ex-Ey)±My)·2^(Ey)`（设Ex≤Ey）

运算步骤

1. 对阶：求阶差的绝对值`▲E = |Ex - Ey|`，当其不等于0时，小阶向大阶看齐——阶码较小的尾数右移▲E位，其阶码值+▲E（每右移一位都要+1）。对原码尾数，符号位不参加唯一，尾数高位补0；对补码尾数，尾数高位补符号位：
    - 逻辑左/右移：直接移动，低位/高位补0，高位/低位丢弃。
    - 算数左/右移：高位不变，其余位同逻辑移动。
2. 尾数加减：完成对阶后，对尾数求和/差；
3. 规格化：对原码，最高数值位为1；对补码尾数，必须是`00.1xxx...`或`11.0xxx...`。补码规格化规则：
    - 若尾数符号位不同（尾数计算结果溢出），应当使结果右移一位，并使阶码值+1（向右规格化，简称右归）。
    - 若尾数双符号位相同（不溢出），且最高位数值位与符号位相同，表示不满足规格化规则，此时应重复使尾数左移、阶-1，直到出现最高位数值与符号位数值不同为止（向左规格化，简称左归）。
4. 舍入：要求有舍有入，尽量使舍和入的机会均等（防止误差的积累）。常用方法是"0舍1入"法：
    - 就近摄入：类似四舍五入，丢弃最高位为1，进1。
    - 朝0舍入：截尾。
    - 朝+∞舍入：正数多余位不全为0，进1；负数截尾。
    - 朝-∞舍入：负数多余位不全为0，进1；正数截尾。
5. 判溢出：浮点数的溢出以**阶码溢出**来表现出来的：
    - 若阶码正常，加/减法运算正常结束。
    - 若阶码下溢，要置运算结果为浮点形式的机器0。
    - 若阶码上溢，则置溢出标志。

补充

综合题01

按字节编址的计算器存储器，用小端方式存储。编译器规定int为32位，short为16位，数据按边界对齐存储，有一个从语言段如下：

```
struct{
   int a;
   char b;
   short c;
}record;
record.a=273;
```

若record变量的首地址位0xC008，则其中的内容的地址为0x11，record.c的地址为0xC00E

a	a	a	a
b	-	c	c

补码计算

字长8位计算机，`x补 = 1 111 0100`，`y补 = 1 011 0000`，求`z=2x+y/2`，则直接对补码进行计算：`2x = 1101000，y/2=11011000`，相加即可得到机器码`1100000`