计算机系统原理实验报告

姓名：朱雨珂

学号：3190103312

时间：2022.4.2

实验要求

选择原/补/移码中的一种，或自行设计一种合适的方式(一般由组里最帅的人自行设计，俗称：帅码)表示整数；给出其算术运算算法。要求有理论推导论证，并进行算法分析，编程实现。
比较补码、原码、移码及帅码制的表示方法与四则算术运算算法，分析各种码制的优缺点。同时分析字位扩展(8-16，16-32位)、运算溢出、大小比较等的方法。算法推导证明(如果手写则拍照上交)
计算机程序模拟，程序只可用无符号整数类型unsigned int，不可用int。(C语言：typedef unsigned int word;)，基本要求实现六个函数：

word atom(char*)：字符串转换成对应的二进制。*

char mtoa(word)：二进制转换成字符串。

word madd(word,word)：二进制所表示数的加法。

word msub(word,word)：减法。

word mmul(word,word)：乘法。

word mdiv(word,word)：除法。

word mmod(word,word)：取余。

位扩展(8-16、16-32位)方法，溢出判断，大小比较。

原码、补码、移码的比较

原码

原码是指一个二进制数左边加上符号位后所得到的码，且当二进制数大于0时，符号位为0；二进制数小于0时，符号位为1；二进制数等于0时，符号位可以为0或1(+0/-0)。

优势：

1. 对数的表示非常直观，符号位和值分开便于人阅读。
2. 比较容易判断运算的溢出。

缺点有：

1. 零有正零和负零之分，例如在8位条 件下0000 0000 [+0] 和 1000 0000 [-0] 同时表示0
2. 比较两数大小需要先比较符号位。
3. 用原码进行加减运算前， 必须先判断符号位，否则会出错。

补码

补码（英语：**2's complement**）是一种用二进制表示有符号数的方法，也是一种将数字的正负号变号的方式，常在计算机科学中使用。补码以有符号比特的二进制数定义。正数和0的补码就是该数字本身再补上最高比特0。负数的补码则是将其对应正数按位取反再加1。

优势：

1. 可以在加法或减法处理中，不需因为数字的正负而使用不同的计算方式。只要一种加法电路就可以处理各种有号数加法，而且减法可以用一个数加上另一个数的补码来表示，因此只要有加法电路及补码电路即可完成各种有号数加法及减法，在电路设计上相当方便。
2. 补码系统的0就只有一个表示方式，这和反码系统不同（在反码系统中，0有二种表示方式），因此在判断数字是否为0时，只要比较一次即可。

缺点：

1. 人们很难从形式上判断真值大小，与人们的习惯不符。

移码

移码（英语：**Offset binary**）是一种将全0码映射为最小负值、全1码映射为最大正值的编码方案。移码没有标准，但通常对于n位二进制数，偏移量$K = 2^{n-1}$——这使得真值0的编码的最高位为1、其余位均为0，相当于补码表示的最高位（符号位）取反；另外，移码在逻辑比较操作中可以得到和真值比较相同的结果，补码则当且仅当符号相同时逻辑比较操作的结果和真值比较相同，否则比较结果将颠倒（负值比正值大）。

优势：

1. 在数轴上，移码所表示的范围，恰好对应于真值在数轴上的范围向正方向移动2^n个单元，直观。
2. 移码的表示中与补码相差一个符号位，而且可以从移码看出真值的大小，转换方便。

关于大小比较和溢出的判断

原码和补码：先比较符号位，若符号位不同则可直接得出比较结果。若符号位相同且两数都为正数，则自高位起逐位比较，直到有不同01的位即可比较出结果。

移码：类似原码和补码，但因为没有符号位，所以不需要比较符号位，只需逐位比较即可得出结果。

本次实验使用的是16位移码，所以处理溢出只需要每次将运算结果和0xffff进行与运算，然后扔掉溢出位就行。

数据表达

本次试验选取的数据表达方式为16位移码，偏移量M为0x8000。

对于正整数Z，Z的移码为：Z+0x8000；对于-Z，移码为：-Z+0x8000。

对于移码X，如果X>0，则原数为：X-M，反之为M-X。

因此，将字符串转化为二进制、二进制移码转换成字符串的代码如下：

```
/*字符串转换成对应的二进制*/
word atom(string s) {
  word i = 0;
  word x = 0;
  bool flag = false;  //false: s>=0   true: s<0
  if (s[0] == '-') {
    flag = true;
    i++;
  }
  while (s[i]) { //convert string to number
    x = x * 10 + (s[i] - 48);
    i++;
  }
  if (flag) x = M - x;   //calculate frame shift
  else x = M + x;
  return x;
}

/*二进制移码转换成字符串*/
string mtoa(word x) {
  if (x & M) { //case x>=0
    x = x - M;
    return to_string(x);
  } else { //case x<0
    x = M - x;
    return "-" + to_string(x);
  }
}
```

代码介绍+原理分析推导

```
#define M 0x8000 //offset
#define F 0xffff //maximum 判断溢出
#define B 16 //bit
```

加法

加减法可以直接先取绝对值（和F与操作），再和M异或操作

加法规则为：f(x+y)=x+y+M=f(x)+f(y)-M=|f(x)+f(y)|^M

```
/*移码的加法操作*/
word madd(word x , word y) {
  return (x + y) & F ^ M;
}
```

减法

减法同加法，减法规则为：f(x-y)=x-y+M=f(x)-f(y)+M=|f(x)-f(y)|^M

```
/*移码的减法操作*/
word msub(word x , word y) {
  return (x - y) & F ^ M;
}
```

乘法

f(xy)=xy+M

首先需要得到xy符号——通过和M或操作得到符号，如果符号相同，异或结果为0，反之为1。

然后将x、y取绝对值，先转化为原码进行乘法运算，然后再把结果转化成移码。

```
/*移码的乘法操作*/
word mmul(word x , word y) {
  word product = 0;
  bool flag = (x & M) ^ (y & M); //flag:true xy<0 false:xy>=0
  if (x&M) x = x ^ M;
  else x = (M - x) & F; //abs
  if (y&M) y = y ^ M;
  else y = (M - y) & F; //abs
  while (y) {
    if (y & 1) product = (product + x) & F;
    x = x << 1;
    y = y >> 1;
  }
  if (flag) product = (M - product) & F; //convert to frame shift
  else product = (M + product) & F;
  return product;
}
```

除法

除法和乘法同理，先看符号是否相同，然后取绝对值运算，最后再转化为移码。

```
/*移码的除法操作*/
word mdiv(word x , word y) {
  word quotient = 0;
  word i = B; //maximum power
  bool flag = (x&M) ^ (y&M); //flag:true xy<0 false:xy>=0
  if (x&M) x = x ^ M;
  else x = (M - x) & F; //abs
  if (y&M) y = y ^ M;
  else y = (M - y) & F; //abs
  while (i) {
    if (x >= y << (i - 1)) {
      quotient = (quotient + (1 << (i - 1))) & F;
      x = (x - (y << (i - 1))) & F;
    }
    i--;
  }
  if (flag) quotient = (M - quotient) & F; //convert to frame shift
  else quotient = (M + quotient) & F;
  return quotient;
}
```

取余操作

f(x%y)=x%y+M

由于x=q*y+r，且在除法过程中每次被除数都被减去一定倍数的除数，因此余数即为除法操作中最后留下的被除数。

```
/*移码的取余操作*/
word mmod(word x , word y) {
  word quotient = 0;
  word remainder = 0;
  word i = B; //maximum power
  bool flag = x & M; //flag:true x>=0 false:x<0
  if (x&M) x = x ^ M;
  else x = (M - x) & F; //abs
  if (y&M) y = y ^ M;
  else y = (M - y) & F; //abs
  while (i) {
    if (x >= y << (i - 1)) {
      quotient = (quotient + (1 << (i - 1))) & F;
```

```
      x = (x - (y << (i - 1))) & F;
    }
    i--;
  }
  remainder = x;
  if (flag) remainder = (M + remainder) & F; //convert to frame shift
  else remainder = (M - remainder) & F;
  return reminder;
}
```

测试

```
int main(void) {
  string p , q;
  cin >> p;
  while (p.compare("end")) {
    cin >> q;
    test(p , q);
    cin >> p;
  }
}

void test(string &p, string &q) {
  word x = atom(p);
  word y = atom(q);
  word plus = madd(x , y);
  word minus = msub(x , y);
  word multi = mmul(x , y);
  word divide = mdiv(x , y);
  word mod = mmod(x , y);
  cout << "test:" << endl;
  cout << mtoa(x) << " + " << mtoa(y) << " = " << mtoa(plus) << "\t\t";
  cout << mtoa(x) << " - " << mtoa(y) << " = " << mtoa(minus) << "\t\t";
  cout << mtoa(x) << " * " << mtoa(y) << " = " << mtoa(multi) << "\t\t";
  cout << mtoa(x) << " / " << mtoa(y) << " = " << mtoa(divide) << "\t\t";
  cout << mtoa(x) << " % " << mtoa(y) << " = " << mtoa(mod) << endl;
}
```

结果展示:

```
zhuyukedeMacBook-Pro:desktop zyk$ g++ -o main main.cpp
zhuyukedeMacBook-Pro:desktop zyk$ ./main
12 12
test:
12 + 12 = 24            12 - 12 = 0             12 * 12 = 144           12 / 12 = 1             12 % 12 = 0
10 1
test:
10 + 1 = 11             10 - 1 = 9              10 * 1 = 10             10 / 1 = 10             10 % 1 = 0
111 222
test:
111 + 222 = 333         111 - 222 = -111                111 * 222 = 24642               111 / 222 = 0           111 % 222 = 111
5 6
test:
5 + 6 = 11              5 - 6 = -1              5 * 6 = 30              5 / 6 = 0               5 % 6 = 5
```

可以完成加、减、乘、除、取余运算，符合实验目标。