计算机系统原理实验报告

课程名称： 计算机系统原理 **实验类型：** 上机 **实验项目名称：** 数据表达与运算

学生姓名： 应承峻 **专业：** 软件工程 **学号：** 3170103456 **实验日期：** 2019年4月4日

实验描述

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。基本要求实现六个函数：

```
typedef unsigned int word;
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。

数据表达方式

本实验中，数据使用16位移码来进行表达，从原码到移码的映射关系记作f, 对应的映射关系（偏移量$M =$ 0x8000）记作：$f(x) = x + 0x8000 = x + M$. 由移码的定义可得出以下推论：

推论： 不发生溢出时，对于正整数P，P 的移码为$M + P$， $-P$ 的移码为$M - P$.

由推论可将字符串转换成对应的二进制：

```
/*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;
}
```

推论：若移码X代表非负数，则其代表的原数的绝对值Y为$X-M$，否则为$M-X$.

证明：当X代表非负时其最高位为1，即$X\geq M$，且$Y+M=X$，可得$Y=|X-M|=X-M$

当X代表负数时，其最高位为0，即$X<M$，故$Y=|X-M|=M-X$

由推论可将二进制移码转换成字符串

```
/*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);
    }
}
```

移码算术运算算法推导证明

结论1：不发生溢出时，两16位无符号整数移码相加其二进制结果最高位必为1

证明： 记两无符号整数分别为x,y，则：$f(x)\geq 0,f(y)\geq 0$

$f(x+y)=x+y+M=(x+M)+(y+M)-M=f(x)+f(y)-M\geq 0$ 得 $f(x)+f(y)\geq M$

故最高位必为1，原命题得证.

结论2：16位无符号整数x满足$f(x)-M=f(x)$ ^ $M=f(x)+M$

证明：设$f(x)$的每一位比特值为$x[i]$，由于：0 ^ 0 = 0， 1 ^ 0 = 1，故 $x[i]$ ^ 0 = $x[i]$， 同理$x[i]$ ^ 1 = ~$x[i]$

① 当最高位为1时，$f(x)$^ M 等价于将$f(x)$的最高位取反，其余各位保持不变，即将1变成0，也就是减去 了M，因而满足$f(x)-M=f(x)$ ^ M . 此时$f(x)+M$ 发生溢出且溢出位被舍弃，故有

$f(x)+M=[f(x)-M+2M]\ mod\ 2^{16}$ = $f(x)-M=f(x)$ ^ M

② 当最高位为0时，$f(x)$^ M 的最高位从0变成了1，等价于$f(x)+M$. 又因为$f(x)-M<0$，此时发生 溢出且需要向假想高位借位，故有$f(x)-M=[f(x)-M+2^{16}]\ mod\ 2^{16}$ = $f(x)+M$ = $f(x)$ ^ M.

综上所述，原命题得证.

由上述结论可以得出以下推论：

$f(x+y) = x + y + M = (x+M) + (y+M) - M = f(x) + f(y) - M = [f(x) + f(y)] \wedge M$

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

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

$f(x-y) = x - y + M = (x-M) - (y-M) + M = f(x) - f(y) + M = [f(x) - f(y)] \wedge M$

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

$f(xy) = xy + M$

移码的乘法需要考虑两数符号，通过$f(x)\&M$ 和$f(y)\&M$分别得到x和y的符号，当符号相同时异或值为0，符号不同时异或值为1，因此当$(f(x)\&M) \oplus (f(y)\&M)$为1时，两数异号，乘积为负，否则乘积为正。

基本的运算思路是：先将x和y取绝对值转换成原码进行乘法，再将计算好的结果转换成移码。

对于任意非负值x，其绝对值的原码为$f(x) - M$，由引理知$f(x) - M$等价于$f(x) \oplus M$。

对于负值x而言，由于$0 - x = (0+M) - (x+M) = M - f(x)$ 其绝对值的原码即为$M - f(x)$。

乘法的过程为：每次通过$y\&1$取乘数y的最后一位，若其为0，则部分积结果不变，否则将部分积加上当前被乘数的错位积。每次执行后，都需要将乘数右移一位，将被乘数左移一位直到乘数为0。

乘法完成后根据其符号来进行相应的移码处理，若乘积为负，则其移码为$M - P$，否则其移码为$M + P$。

```
/*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;
}
```

$f(\frac{x}{y}) = \frac{x}{y} + M$

除法与乘法的出发点一致，也是通过$f(x)\&M$ 和$f(y)\&M$分别得到x和y的符号，当符号相同时异或值为0，符号不同时异或值为1，因此当$(f(x)\&M)\oplus(f(y)\&M)$为1时，两数异号，乘积为负，否则乘积为正。

基本的运算思路是：先将x和y取绝对值转换成原码进行除法，再将计算好的结果转换成移码。

做除法时，在16位无符号整数下，商的绝对值的最大的可能性是2^{15}即$|-2^{15}/1|$，因而我们从2^{15}开始试探，如果被除数减去除数的2^i倍后仍然非负，那么就将2^i加到商上并且在被除数中减去它。

减去2^i倍还有剩余等价于将y左移i位后仍不大于x，其表达式可以描述为：`x >= y << i`。

将2^i加到商上即将商加上$(1<<i)$的值。

```
/*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;
}
```

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

由于$x=q\times y+r$，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x。定义取模运算的规则：模的符号与被除数的符号相同。

```
/*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 remainder;
}
```

测试驱动程序

```
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 << 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;
    cout << "-------------------------------------------------------------------------------------------------------------" << endl;
}
```

测试样例与测试结果

```
12 8
12 + 8 = 20            12 - 8 = 4             12 * 8 = 96            12 / 8 = 1             12 % 8 = 4
------------------------------------------------------------------------------------------------------------
-12 8
-12 + 8 = -4           -12 - 8 = -20          -12 * 8 = -96          -12 / 8 = -1           -12 % 8 = -4
------------------------------------------------------------------------------------------------------------
-12 -8
-12 + -8 = -20         -12 - -8 = -4          -12 * -8 = 96          -12 / -8 = 1           -12 % -8 = -4
------------------------------------------------------------------------------------------------------------
12 -8
12 + -8 = 4            12 - -8 = 20           12 * -8 = -96          12 / -8 = -1           12 % -8 = 4
------------------------------------------------------------------------------------------------------------
2 -6
2 + -6 = -4            2 - -6 = 8             2 * -6 = -12           2 / -6 = 0             2 % -6 = 2
------------------------------------------------------------------------------------------------------------
0 5
0 + 5 = 5              0 - 5 = -5             0 * 5 = 0              0 / 5 = 0              0 % 5 = 0
------------------------------------------------------------------------------------------------------------
-36 -9
-36 + -9 = -45         -36 - -9 = -27         -36 * -9 = 324         -36 / -9 = 4           -36 % -9 = 0
------------------------------------------------------------------------------------------------------------
36 -9
36 + -9 = 27           36 - -9 = 45           36 * -9 = -324         36 / -9 = -4           36 % -9 = 0
------------------------------------------------------------------------------------------------------------
```

原码、移码、补码的比较

原码：原码就是符号位加上真值的绝对值，即用第一位表示符号，其余位表示值。其优点有：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。但其缺点有：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。例如1+(-1) 用原码计算为0000 0001 + 1000 0001 = 1000 0010，结果在十进制下为-2，这显然是错误的。

移码：移码是在原码的基础上加上一定的偏移量，N位整数其偏移量通常取2^{N-1}，其目的是将被编码数X转换成一个非负数。其优点有①相对于原码，可以方便的比较两数的大小和进行两数的减法运算 ②解决了零有正零和负零的矛盾，N位整数其移码的取值范围为$[0, 2^N - 1]$，对应的原码为$[-2^{N-1}, 2^{N-1} - 1]$。其缺点为：①还是没有解决原码加减运算不能合并成同一种操作的问题 ②乘除法实现较为麻烦。

补码：补码的优点有：①补码使得加法操作能够和减法操作合并为同一种操作，减去一个数等价于加上这个数的补数 ②补码的使用解决了原码中相反数相加不为0的问题 ③补码在字位扩展中具有较好的性能。 补码的缺点对于初学者而言较难理解且不适合阅读，直观上难以判断两数大小

字位扩展

带符号扩展：有符号整数按照符号扩展的形式扩展高位，即最高位是1则高位全部补1，最高位是0全部补0。

无符号扩展：无符号整数按照零扩展的方式扩展高位，即高位全部直接补零。

大小比较

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，直到某一位$x_i \oplus y_i = 1$时即可得出结果。

移码：类似原码和补码，但不需要比较符号位，只需逐位比较即可得出结果。

溢出判断

由于采用16位无符号整数进行运算，因而只要在每次运算后与`0xffff`进行与运算舍弃掉溢出位即可。