计算机系统原理实验报告

> 数据表达与运算
>
> 王子腾 3180102173 软件工程
>
> 2020.04.06

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(一般可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

数据转换

```
//宏定义
#define M 0x10000   // 2^16
#define N 0x8000    // 符号位
#define F 0xffff    // 满位
```

本实验采用16位补码表示数据，对于原码x，对应的补码为f(x)，则满足：

> f(x) = (x + 2^16) % 2^16

由补码的定义，得出推论，对于正整数 x， x 补码为 x， -x 补码为 (2^16 - x) % 2^16 。 因此可写出函数 atom

```
//字符串转换成对应的二进制。
word atom(string s)
{
    word i = 0;
    word num = 0;
    bool flag = false;  //true:负数 flase:正数
    if(s[0]=='-')
    {
        flag = true;
        i++;
    }
    while(s[i])
    {
        num = num*10 + (s[i]-'0');
        i++;
    }
    if(flag)
        num = M - num;
    return num;
}
```

反之，若补码为 X，则根据符号位判断正负后,由 X = (x + 2^16) % 2^16 可知：

> 若 x 为正整数，则 x = X
>
> 若 x 为负整数，则 x = 2^16 - X

因此写出函数mtoa：

```
//二进制转换成字符串。
string mtoa(word n)
{
    bool flag = n&N;
    word num = (n&N)? (M-n) : n;    //真数绝对值
    stringstream ss;
    ss << num;
```

```
    if(!flag)           //正整数
        return ss.str();
    n = (M-n);          //负整数
    return '-'+ss.str();
}
```

算术运算

加法

> 定理：对于两整数x，y，则：f(x + y) = (x + y + 2^16) % 2^16

由此写出函数madd：

```
//二进制所表示数的加法。
word madd(word a, word b)
{
    return (((a+b)&F)+M)%M;
}
```

减法

由于可将减法看作加法的特殊运算，即 x - y 与 x + (-y) 同理, 因此由推理

> 推理：对于两整数x，y，则：f(x - y) = (x - y + 2^16) % 2^16

由此写出函数msub：

```
//二进制所表示数的减法。
word msub(word a, word b)
{
    return (((a-b)&F)+M)%M;
}
```

乘法

> 推理：对于两整数x，y，则：f(x * y) = (x * y + 2^16) % 2^16

补码的乘法需要考虑两数符号，通过与N与运算得到两数的符号，当符号相同时异或值 flag 为0，符号不同时异或值为1，因此当 flag 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 a 和 b 取绝对值转换成原码进行乘法，再将计算好的结果转换成补码。

对于任意非负补码 a ，其绝对值的原码为 a ；对于负值 a 而言，其绝对值的原码即为 M - a。

乘法的过程为：每次通过取乘数 b 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补码处理，若乘积 P 为负，则其补码为 (-1*P+M)%M ，否则其补码为 (product+M)%M 。

```
//二进制所表示数的乘法。
word mmul(word a, word b)
{
    word product = 0;
    bool flag = (a&N)^(b&N);
    word num1 = (a&N)? (M-a) : a;    //真数绝对值
    word num2 = (b&N)? (M-b) : b;

    while(num2)
    {
        if(num2 & 1)    product = (product + num1)&F;
        num1 = num1<< 1;
        num2 = num2>> 1;
    }
    if(flag)
        return ((((F^product)+1)&F)+M)%M;
    else    return (product+M)%M;
}
```

除法

> 推理：对于两整数x，y，则：f(x / y) = (x / y + 2^16) % 2^16

除法与乘法的出发点一致，也是通过与N与运算分别得到两数的符号，当符号相同时异或值 flag 为0，符号不 同时异或值 flag 为1，因此当 flag 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 a 和 b 取绝对值转换成原码进行除法，再将计算好的结果转换成补码。

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 0x8000 ，因而我们从 2^15 开始试探，如果被除数 a 减去除数 b 的 2^off 倍后仍然非负，那么就将 2^off 加到商上并且在被除数 a 中减去它。

最终当 a 的剩余值小于 b 时，除法结束。

```
//二进制所表示数的除法。
word mdiv(word a, word b)
{
    word quotient = 0;
    bool flag = (a&N)^(b&N);
    word num1 = (a&N)? (M-a) : a;    //真数绝对值
    word num2 = (b&N)? (M-b) : b;
    word off = 16;

    while(off)
    {
        if(num1 > num2<<(off-1))
        {
            quotient+=1<<(off-1)&F;
            num1 -= num2<<(off-1)&F;
        }
        off--;
    }
    if(flag)
        return ((((F^quotient)+1)&F)+M)%M;
    else    return (quotient+M)%M;
}
```

取余

> 推理：对于两整数x，y，则：f(x % y) = (x % y + 2^16) % 2^16
>
> 推理：r = a - b * q 其中q = (a/b)为a，b相除的整数商

对于取余可知，将数 a 减去 b 与 a，b的商 的乘积，即为余数

```
//二进制所表示数的除法。
word mmod(word a,word b)
{
    word remainder = 0;
    remainder = msub(a, mmul(b,mdiv(a,b))); //a-b*quotient
    return remainder;
}
```

测试程序

```
//测试驱动
void test(string x, string y)
{
    word a = atom(x);
    word b = atom(y);

    word radd = madd(a, b);
    word rsub = msub(a, b);
    word rmul = mmul(a, b);
    word rdiv = mdiv(a, b);
    word rmod = mmod(a, b);

    string s1 = x+" + "+y+" = "+mtoa(radd);
    string s2 = x+" - "+y+" = "+mtoa(rsub);
    string s3 = x+" * "+y+" = "+mtoa(rmul);
    string s4 = x+" / "+y+" = "+mtoa(rdiv);
    string s5 = x+" % "+y+" = "+mtoa(rmod);

    cout << s1 << endl;
    cout << s2 << endl;
    cout << s3 << endl;
    cout << s4 << endl;
    cout << s5 << endl;
    cout << '\n' << endl;
}

int main()
{
    string x, y;
    cin>>x;
```

```
    while(x!="end")
    {
        cin>>y;
        test(x,y);
        cin>>x;
    }
    return 0;
}
```

测试样例

```
11 7
11 + 7 = 18
11 - 7 = 4
11 * 7 = 77
11 / 7 = 1
11 % 7 = 4


-11 14
-11 + 14 = 3
-11 - 14 = -25
-11 * 14 = -154
-11 / 14 = 0
-11 % 14 = -11


11 -5
11 + -5 = 6
11 - -5 = 16
11 * -5 = -55
11 / -5 = -2
11 % -5 = 1


-13 -21
-13 + -21 = -34
-13 - -21 = 8
-13 * -21 = 273
-13 / -21 = 0
-13 % -21 = -13


end
```

原、补、移码的比较

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。

补码

补码是在原码的基础上对除符号位外的每一位取反后加一得到的，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

移码

移码是在补码的基础上对符号位取反得到的，其目的是将被编码数转换成一个非负数。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, 2^N - 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。

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。

本题中由于使用补码运算，采用带符号扩展。

运算溢出

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。

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 ^ y = 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。