hw6 整数运算器

表示方式

补码

算法分析

原理：

两数之和的补码等于补码之和的补码。

$$x_{com} = (2^N + x) mod\ 2^N$$
$$y_{com} = (2^N + y) mod\ 2^N$$
$$x_{com} + y_{com} = (2^{N+1} + x + y) mod (2^N) = (x + y) mod (2^N) = (x + y)_{com}$$

两数之差等于减数与被减数的相反数之和，也满足加法定律。

两数乘积为若干次加法之和，也满足加法定律。

优缺点

原码表示数字较为直接，但运算不便

补码的优点是能使符号位与有效值部分一起参与运算，从而简化了运算规则，但不易直接看出数字大小。

移码是在补码的基础上把首位取反得到的，这样使得移码非常适合于阶码的运算。

补码的溢出

补码运算的溢出判别方式为双高位判别法，利用CS表示符号位是否进位，利用CP表示最高数值位是否进位。如果CS ∧ CP的结果为真，则代表发生了溢出（例如正数+正数得到负数），否则没有溢出。

大小比较

先比较符号位，即最高位，1为负数0为正数（负数肯定小于正数），然后由高位向低位进行 字典序依次比较，如果这个数是正数即符号位为0，则字符串比较（字典序）结果大的数，其 值大；如果为负数，则字符串比较（字典序）结果小的数，其值反而大

符号扩展

如果为负数，则在高位上补1，正数则补0。

代码及测试

```
#include<iostream>
#include<string>
using namespace std;
typedef unsigned long word;
//补码表示
word atom(char *s) {//字符串转数
    bool isneg = false;
    word i, result;
    i = result = 0;
    if (s[0] == '-') {
        isneg = true;
        i++;
    }
    while (s[i] != '\0') {
        result = result * 10 + s[i++] - '0';
    }
    if (isneg) {
        return 0xFFFFFFFF - result + 1;
    }
    else
        return result;
}

word madd(word a, word b) {
    word sum = a;
    word carry = b;
    word tmp;
    while (carry) {
        tmp = sum;
        sum = sum ^ carry;
        carry = (tmp & carry) << 1;
    }
    return sum;
}

word msub(word a, word b) {
    return madd(a, madd(~b, 1));//a + b补码
}

word mmut(word x, word y) {// booth算法
    bool isneg = (x >> 31) ^ (y >> 31);
    word result = 0;
    x = (x >> 31) ? madd(~x, 1) : x;
```

```
    y = (y >> 31) ? madd(~y, 1) : y;
    while(y){
        if(y & 1){
            result += x;
            y >>= 1;
            x <<= 1;
        }
    }
    if(isneg)
        return madd(~result, 1);
    else
        return result;
}

word mdiv(word x, word y) {
    bool isneg = (x >> 31) ^ (y >> 31);
    word a, b;
    word result = 0;
    a = (x >> 31) ? madd(~x, 1) : x;
    b = (y >> 31) ? madd(~y, 1) : y;
    for (int i = 31; i >= 0; i--) {
        if ((a >> i) >= b) {

            result += (1 << i);
            a -= (b << i);
        }
    }
    if (isneg)
        return madd(~result, 1);
    else
        return result;
}

word mmod(word x, word y) {
    word a, b, result = 0;
    a = (x >> 31) ? madd(~x, 1) : x;
    b = (y >> 31) ? madd(~y, 1) : y;
    result = msub(x, mmut(mdiv(x, y), y));
    if (result >> 31)
        return madd(~result, 1);
    else
        return result;
}

char* mtoa(word n) {//数转字符串
    char str[40];
    word i = 0;
    while (i < 32) {
        if (n >> (31 - i) & 1) {
```

```
            str[i] = '1';
        }
        else
            str[i] = '0';
        i++;
    }
    str[i] = '\0';
    return str;
}

int main() {
    word n = -127;
    cout << mtoa(n) << endl;
    char nn[33] = "127";
    cout << atom(nn) << endl;
    word a = 125;
    word b = 127;
    printf("%d - %d = %X\n", a, b, msub(a, b));
    printf("%d x %d = %d\n", a, b, mmut(a, b));
    a = 256;
    b = -16;
    printf("%d / %d = %d\n", a, b, mdiv(a, b));
}

```

```
11111111111111111111111110000001
127
125 - 127 = FFFFFFFE
125 x 127 = 15875
256 / -16 = -16

--------------------------------
Process exited after 0.2007 seconds with return value 0
请按任意键继续. . .
```