计算机系统原理第 6 次作业报告

原码计算的实现

1. 四则运算与取余的算法理论推导

a) 加法：
   i. 先进行最高位符号判断
   ii. 若都为正则直接先去掉符号后进行 unsigned int 的相加
   iii. 若都为负则
   iv. 若不同则进入减法并把符号位都变为正

b) 减法：
   i. 进行最高位判断
   ii. 若为正则 unsigned int 相减，若为都为负则第二个数的绝对值减第一个数的绝对值，另外两种情况返回加法的结果即可。

c) 乘法：
   由于计算机中，所有数值都是用 2 的 N 次方来表示的:2^n0+2^n1+2^n2+2^n3+2^n4.....
   因此 x*y,(x)*(2^n0+2^n1+2^n2+2^n3+2^n4)=(x*2^n0)+(x*2^n1)+(x*2^n2)+(x*2^n3)+(x*2^n4)+.....即(x 左移 n0)+(x 左移 n1)+(x 左移 n2)+(x 左移 n3)+(x 左移 n4)+.....
   用 15(x)*13(y)来举例，15*13 为 1111*1101
   a.首先 y 的最低位为 1(2^0)，x 左移 0 位得到 1111
   b.然后 y 的最低第二位为 0，没有 2^1 存在，因此本次无运算(结果可以看作为 0)
   c.然后 y 的最低第三位为 1(2^2)，x 左移 2 位得到 111100
   d.然后 y 的最低第四位为 1(2^3)，x 左移 3 位得到 1111000
   e.把 a、b、c、d 的结果相加 1111+0+111100+1111000=11000011(195),该结果就是乘法的结果

d) 除法：
   设 ri 表示第 i 次运算后所得的余数,则：
   若 ri>0，则商 1，余数和商左移 1 位，再减去除数，即 ri+1=2ri-y
   若 ri<0，则商 0，余数和商左移 1 位，再加上除数，即 ri+1=2ri+y

   用 85/6 来举例,85/6=1010101/110
   a.101(0101)左移 1 位到第 3 位都小于 110，因此商=000
   b.1010(101)左移四位是 1010,比 110 大，商=0001,余数=1010-110=100(101)
   c.余数 100(101)左移一位是 1001,比 110 大，商=00011,余数=1001-110=11(01)
   d.余数 11(01)左移一位是 110,等于 110，商=000111,余数=0(1)
   e.余数 0(1)左移一位是 01,小于 110,商=0001110,余数=01

e) 取余：
   综合运用乘法减法以及除法即可得到结果。

2. 代码实现

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
typedef unsigned int word;

//判断是否溢出,溢出返回 0
int judge(word oldcode,word currentcode)
{
    if ((oldcode > 0x8000 && currentcode < 0x18000) || (oldcode < 0x8000 && cur
rentcode < 0x8000))return 1;
    else return 0;
}
//字符串转数
word atom(char* str)
{
    int i = 0;
    word temp = 0;
    if (str[0] == '-')
    {
        temp = temp | 0x8000;
        i = 1;
    }
    for (; str[i]; i++)
    {
        if(judge(temp,temp*10+(str[i]-48)))temp = temp * 10 + (str[i] - 48);
        else
        {
            printf("该字符串溢出！\n");
            return 0;
        }
    }
    return temp;
}
//数转字符串
char* mtoa(word code)
{
    char *temp1;
    int temp2;
    temp1 = (char *)malloc(sizeof(char) * 20);
    char* str = (char*)malloc(sizeof(char) * 20);
    if (code & 0x8000)
    {
```

```
        str[0] = '-';
        str[1] = 0;
    }
    else str[0] = 0;
    code = code & 0x7FFF;
    temp2 = code;
    itoa(temp2, temp1, 2);
    strcat(str, temp1);
    return str;
}
//加法
word madd(word code1, word code2)
{
    if (code1 & 0x8000 || code2 & 0x8000)
    {
        if (code1 & 0x8000 && code2 & 0x8000)
        {
            code1 = code1 & 0x7FFF;
            code2 = code2 & 0x7FFF;
            return madd(code1, code2) | 0x8000;
        }
        else if (code1 & 0x8000)return msub(code2, code1 & 0x7FFF);
        else return msub(code1, code2 & 0x7FFF);
    }
    else
    {
        if (code1 + code2 < 0x8000)
        {
            return (code1 + code2);
        }
        else printf("溢出！\n");
    }
    return 0;
}
//减法
word msub(word code1 , word code2)
{
    if (code1 & 0x8000 || code2 & 0x8000)
    {
        if (code1 & 0x8000 && code2 & 0x8000)
            return msub(code2 & 0x7FFF, code1 & 0x7FFF);
        else return madd(code1, code2 & 0x7FFF);
    }
    else
```

```
86.     {
87.         if (code2 > code1)return (code2 - code1) | 0x8000;
88.         else return code1 - code2;
89.     }
90. }
91. //乘法
92. word mmul(word code1 , word code2)
93. {
94.     word ans = 0;
95.     while (code2)
96.     {
97.         if (!judge(ans, ans += code1))
98.         {
99.             printf("溢出! \n");
100.            return 0;
101.        }
102.        if (code2 & 1 == 1)
103.            ans += code1;
104.        code2 = code2 >> 1;
105.        code1 = code1 << 1;
106.    }
107.    return ans;
108.}
109.//除法
110.word mdiv(word code1, word code2)
111.{
112.    int flag = 1;
113.    word ans = 0;
114.    if (code1 & 0x8000)
115.    {
116.        flag = !flag;
117.        code1 = code1 & 0x7FFF;
118.    }
119.    if (code2 & 0x8000)
120.    {
121.        flag = !flag;
122.        code2 = code2 & 0x7FFF;
123.    }
124.
125.    for (int i = 15; i >=0; i--)
126.    {
127.        if (code1 >> i >= code2)
128.        {
129.            ans += 1 << i;
```

```
            code1 = code1 - (code2 << 1);
        }
    }
    if (flag)
        return ans;
    else
        return ans | 0x8000;
}
//取余
word mmod(word code1 , word code2)
{
    if ((code1 & 0x8000 && code2 & 0x8000) || !(code1 & 0x8000 || code2 & 0x8000))
    {
        code1 = code1 & 0x7FFF;
        code2 = code2 & 0x7FFF;
        return msub(code1, mmul(mdiv(code1, code2), code2));
    }
    else if (code1 & 0x8000)
        return (mmod((code1 & 0x7FFF) , code2)) | 0x8000;
    else
        return mmod(code1 , (code2 & 0x7FFF));
}
//比大小
int compare(word code1, word code2)
{
    if (code1 == code2)return 0;
    else if (code1 & 0x8000 && code2 & 0x8000)return compare(code2 & 0x7FFF, code1 & 0x7FFF);
    else if (code1 & 0x8000)return -1;
    else if (code2 & 0x8000)return 1;
    else
    {
        if (code1 > code2)return 1;
        else return -1;
    }
}
```

3. 比较优缺点

1、 原码：是机器数的一种简单的表示法。其符号位用 0 表示正号，用 1 表示负号，数值一般用二进制形式表示。

优点：最简单直观。

缺点：不能直接参加运算，可能会出错。

原码来历：在机器中，只能识别二进制数字，所以所有的数字都用原码来表示。

2、 反码：可由原码得到。如果机器数是正数，则该机器数的反码与原码一样；如果机器数是负数，则该机器数的反码是对它的原码（符号位除外）各位取反而得到的。

优点：解决负数加法运算问题，将减法运算转换为加法运算，从而简化运算规则。

反码来历：为了解决“正负相加等于 0”的问题，在“原码”的基础上，人们发明了“反码”。

3、 补码：可由原码得到。如果机器数是正数，则该机器数的补码与原码一样；如果机器数是负数，则该机器数的补码是对它的原码（除符号位外）各位取反，并在末位加 1 而得到的

优点：可以把负数直接拿来算加法。

缺点：计算稍微复杂，容易忘记公式，计算错误。

补码来历：计算机里面，只有加法器，没有减法器，所有的减法运算，都必须用加法进行，用补数代替原数，可把减法转变为加法。