计算机系统原理第6次作业报告

原码计算的实现

## 四则运算与取余的算法理论推导

1. 加法：
2. 先进行最高位符号判断
3. 若都为正则直接先去掉符号后进行unsigned int的相加
4. 若都为负则
5. 若不同则进入减法并把符号位都变为正
6. 减法：
7. 进行最高位判断
8. 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。
9. 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

1. 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

1. 取余：

综合运用乘法减法以及除法即可得到结果。

## 代码实现

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<string.h>
4. **typedef** unsigned **int** word;
6. //判断是否溢出,溢出返回0
7. **int** judge(word oldcode,word currentcode)
8. {
9. **if** ((oldcode > 0x8000 && currentcode < 0x18000) || (oldcode < 0x8000 && currentcode < 0x8000))**return** 1;
10. **else** **return** 0;
11. }
12. //字符串转数
13. word atom(**char**\* str)
14. {
15. **int** i = 0;
16. word temp = 0;
17. **if** (str[0] == '-')
18. {
19. temp = temp | 0x8000;
20. i = 1;
21. }
22. **for** (; str[i]; i++)
23. {
24. **if**(judge(temp,temp\*10+(str[i]-48)))temp = temp \* 10 + (str[i] - 48);
25. **else**
26. {
27. printf("该字符串溢出！\n");
28. **return** 0;
29. }
30. }
31. **return** temp;
32. }
33. //数转字符串
34. **char**\* mtoa(word code)
35. {
36. **char** \*temp1;
37. **int** temp2;
38. temp1 = (**char** \*)malloc(**sizeof**(**char**) \* 20);
39. **char**\* str = (**char**\*)malloc(**sizeof**(**char**) \* 20);
40. **if** (code & 0x8000)
41. {
42. str[0] = '-';
43. str[1] = 0;
44. }
45. **else** str[0] = 0;
46. code = code & 0x7FFF;
47. temp2 = code;
48. itoa(temp2, temp1, 2);
49. strcat(str, temp1);
50. **return** str;
51. }
52. //加法
53. word madd(word code1, word code2)
54. {
55. **if** (code1 & 0x8000 || code2 & 0x8000)
56. {
57. **if** (code1 & 0x8000 && code2 & 0x8000)
58. {
59. code1 = code1 & 0x7FFF;
60. code2 = code2 & 0x7FFF;
61. **return** madd(code1, code2) | 0x8000;
62. }
63. **else** **if** (code1 & 0x8000)**return** msub(code2, code1 & 0x7FFF);
64. **else** **return** msub(code1, code2 & 0x7FFF);
65. }
66. **else**
67. {
68. **if** (code1 + code2 < 0x8000)
69. {
70. **return** (code1 + code2);
71. }
72. **else** printf("溢出！\n");
73. }
74. **return** 0;
75. }
76. //减法
77. word msub(word code1 , word code2)
78. {
79. **if** (code1 & 0x8000 || code2 & 0x8000)
80. {
81. **if** (code1 & 0x8000 && code2 & 0x8000)
82. **return** msub(code2 & 0x7FFF, code1 & 0x7FFF);
83. **else** **return** madd(code1, code2 & 0x7FFF);
84. }
85. **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. }
125. **for** (**int** i = 15; i >=0; i--)
126. {
127. **if** (code1 >> i >= code2)
128. {
129. ans += 1 << i;
130. code1 = code1 - (code2 << 1);
131. }
132. }
133. **if** (flag)
134. **return** ans;
135. **else**
136. **return** ans | 0x8000;
137. }
138. //取余
139. word mmod(word code1 , word code2)
140. {
141. **if** ((code1 & 0x8000 && code2 & 0x8000) || !(code1 & 0x8000 || code2 & 0x8000))
142. {
143. code1 = code1 & 0x7FFF;
144. code2 = code2 & 0x7FFF;
145. **return** msub(code1, mmul(mdiv(code1, code2), code2));
146. }
147. **else** **if** (code1 & 0x8000)
148. **return** (mmod((code1 & 0x7FFF) , code2)) | 0x8000;
149. **else**
150. **return** mmod(code1 , (code2 & 0x7FFF));
151. }
152. //比大小
153. **int** compare(word code1, word code2)
154. {
155. **if** (code1 == code2)**return** 0;
156. **else** **if** (code1 & 0x8000 && code2 & 0x8000)**return** compare(code2 & 0x7FFF, code1 & 0x7FFF);
157. **else** **if** (code1 & 0x8000)**return** -1;
158. **else** **if** (code2 & 0x8000)**return** 1;
159. **else**
160. {
161. **if** (code1 > code2)**return** 1;
162. **else** **return** -1;
163. }
164. }

## 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，并在未位加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。