**南京理工大学计算机科学与工程学院**

**程序设计基础（Ⅱ） 实验报告**

**学生姓名 xx**

**学 号 xxxxxxxxxxxx**

**教 师 xxxxxx**

目录

[一、设计思路 3](#_Toc103195408)

[二、伪代码 4](#_Toc103195409)

[三、源代码 6](#_Toc103195410)

[四、寄存器使用对照表 11](#_Toc103195411)

[五、程序运行结果 13](#_Toc103195412)

[六、心得体会 14](#_Toc103195413)

# 一、设计思路

1. 输入输出：从输入输出来看，为了实现有出错处理的输入，以字符串的形式接收用户输入，当用户输入非正整数或不合法的整数时，程序能识别出错误类别（如：空字符串、负整数、零、非法整数）并且提示相应的错误信息，还可以自行对前导零、前导空格、后导空格进行纠错，具体为int Decin(int)函数。
2. 通过迭代方式实现打印前n个斐波那契数列：主要难点在于对循环的处理，与数列打印，具体为void print\_Fibonacci\_iteration(int)。
3. 通过递归方式实现计算第n个斐波那契数列数：主要难点在于递归函数的参数传递，使用栈存储函数的参数、返回值和$ra。具体为int Fibonacci\_recursive(int)
4. 通过迭代方式实现判断某个数是否属于斐波那契数列：类似第二种设计思想,若找到返回位于数列下标，若找不到返回-1，具体为int find\_Fibonacci\_iteration(int)

总结：以上函数均使用栈作为参数传递

# 二、伪代码

伪代码使用c++进行描述，如下：

1. #include<cstdio>
2. #include<iostream>
3. #define ull unsigned long long
4. **using** **namespace** std;
5. **char** buffer[10];
6. **const** **char**\*err1="\nIt's a empty string!\n";
7. **const** **char**\*err2="\nIt's a error string!\n";
8. **const** **char**\*err3="\nA nonpositive is not allowed!\n";
9. **const** **char**\*err4="\nIt's a number that is not allowed!\n";
10. **const** **char**\*start\_screen="-----------------Fibonacci-----------------\n-------------------menu--------------------\n\n1.Print first N numbers of Fiboaccai\n2.Print the N number of Fibonacci\n3.Find a number in Fibonacci\n4.exit\n\n";
11. **const** **char**\*exit\_screen="\n-----------------byebye-----------------\n\n";
13. // 包括出错处理的输入函数
14. **int** Decin(**int** type){
15. **while**(1){
16. **int** state=0;//状态
17. **if** (type==0)
18. {
19. cout<<"Please input a dec value(<1000000000):";
20. }**else**{
21. cout<<"Please input your choice:";
22. }
23. cin.getline(buffer,10);
24. **char** sp=' ';
25. **char** enter='\n';
26. **char** fh='-';
27. **char** \*fp=buffer;
28. **char** temp;
29. //do\_space1
30. **do**{
31. temp=\*fp;
32. fp++;
33. **if**(temp==0 || temp==enter) {cout<<err1;state=1;**break**;}//检测出空串
34. **if**(temp==fh) {cout<<err3;state=3;**break**;}//检测出负号
35. }**while**(temp==sp);
36. **if**(state!=0) **continue**;
37. //do\_zero
38. **while**(temp=='0'){
39. temp=\*fp++;
40. **if**(temp==0 || temp==enter) {cout<<err3;state=3;**break**;}//检测出0
41. }
42. **if**(state!=0) **continue**;
43. **int** m=0;
44. **int** basis=10;
45. //loop1
46. **do**{
47. **if**(temp==sp){
48. //do\_space2
49. **do**{
50. temp=\*fp;
51. fp++;
52. }**while**(temp==sp);
53. **if**(temp!=0 && temp!=enter){state=2;cout<<err2;**break**;}//检测出错误数字
54. }
55. **if**(temp==0 ||temp==enter) **break**;
56. **if**(temp>'9' || temp<'0'){state=2;cout<<err2;**break**;}//检测出错误数字
57. //计算数字
58. m\*=basis;
59. m+=temp-'0';
60. temp=\*fp;
61. fp++;
62. }**while**(1);
63. **if**(state!=0) **continue**;
64. **return** m;
65. }
66. }
67. // 使用迭代实现的斐波那契数列
68. **void** print\_Fibonacci\_iteration(**int** n){
69. ull pre=0;//前一个数
70. ull cur=1;//当前数
71. cout<<"1 ";
72. **while**(--n>0){
73. ull temp=cur;
74. cur+=pre;
75. pre=temp;
76. cout<<cur<<" ";
77. }
78. cout<<endl;
79. }
80. // 使用迭代实现的斐波那契数列 找到返回下标(从1开始)，找不到返回-1
81. **int** find\_Fibonacci\_iteration(**int** n){
82. ull pre=0;//前一个数
83. ull cur=1;//当前数
84. **int** index=1;
85. **while**(1){
86. **if**(cur==n) **return** index;
87. **else** **if**(cur>n){
88. **return** -1;
89. }
90. index++;
91. ull temp=cur;
92. cur+=pre;
93. pre=temp;
94. }
95. }
96. // 使用递归实现的斐波那契数列
97. ull Fibonacci\_recursive(**int** n){
98. **if**(n<=2){
99. **return** 1;
100. }
101. **return** Fibonacci\_recursive(n-1)+Fibonacci\_recursive(n-2);
102. }
103. **int** main(){
104. cout<<start\_screen;//输出开始屏幕
105. **while**(1){
106. **int** type = Decin(1);
107. **if**(type==4){
108. cout<<exit\_screen;//输出结束屏幕
109. **return** 0;
110. }
111. **int** m = Decin(0);
112. **if**(type==2)
113. cout<<Fibonacci\_recursive(m)<<endl;
114. **else** **if**(type==1)
115. print\_Fibonacci\_iteration(m);
116. **else** **if**(type==3){
117. cout<<find\_Fibonacci\_iteration(m)<<endl;
118. }
119. **else**{
120. cout<<err4;
121. }
123. }
124. }

# 三、源代码

GitHub链接：<https://github.com/ky-sfms/MY_NJUST_HOMEWORK/blob/main/%E6%B1%87%E7%BC%96%E4%BD%9C%E4%B8%9A/Fibinacci.asm>

|  |
| --- |
| .data  start\_screen: .asciiz "-----------------Fibonacci-----------------\n-------------------menu--------------------\n\n1.Print first N numbers of Fiboaccai\n2.Print the N number of Fibonacci\n3.Find a number in Fibonacci\n4.exit\n"  exit\_screen: .asciiz "\n-----------------byebye-----------------\n"  tip1: .asciiz "\nPlease input a dec value(<1000000000):"  tip2: .asciiz "\nPlease input your choise:"  buffer: .space 10  endl: .asciiz "\n"  sp: .asciiz " "  err1: .asciiz "\nIt's a empty string!\n"  err2: .asciiz "\nIt's a error string!\n"  err3: .asciiz "\nA nonpositive is not allowed!\n"  err4: .asciiz "\nIt's a number that is not allowed!\n"  .globl main  .text  main:  # 打印开始界面  li $v0 4  la $a0 start\_screen  syscall  # end  loop:  # 输入用户的选择  choose:  addi $sp,$sp,-4 # 用于储存局部变量  addi $sp,$sp,-8  li $8,1 # 对应Decin函数参数type  sw $8,($sp) # type参数压栈  jal Decin # 跳转到Decin函数  lw $8,4($sp) # 加载Decin函数返回值，对应局部变量type  addi $sp,$sp,8  li $9,4  bgt $8,$9,out\_5 # 用户选择值未在选择范围  beq $8,$9,end\_all # 终止程序，返回系统  addi $8,$8,-2 # type=type-2  sw $8,($sp) # 存储 type 到栈  # # 输入数字  addi $sp,$sp,-8  li $8,0 # 对应Decin函数参数type  sw $8,($sp ) # type参数压栈  jal Decin # 跳转到Decin函数  lw $a0,4($sp ) # 加载Decin函数返回值，对应局部变量m  addi $sp,$sp,8  # # end  lw $8,($sp ) # 加载局部变量type  beqz $8,recursive # 若type==0,跳转到递归计算部分  bgtz $8,find\_iteration # 若type>0,跳转到迭代查找部分  b print\_iteration # 否则,跳转到迭代打印部分  out\_5:  li $v0,4 # 打印err4  la $a0,err4  syscall # end  b choose # 继续输入  # end  print\_iteration:  # 调用迭代打印前n个数列  addi $sp,$sp,-4  sw $a0,($sp) # 将print\_Fibonacci\_iteration函数的参数n压栈  jal print\_Fibonacci\_iteration  addi $sp,$sp,4  b loop  # end  recursive:  # 调用递归计算第n个数字  addi $sp,$sp,-16  sw $a0,($sp) # 将Fibonacci\_recursive函数的参数n压栈  jal Fibonacci\_recursive  lw $a0,12($sp) # 加载并打印Fibonacci\_recursive函数返回值  li $v0,1  syscall  la $a0,endl  li $v0,4  syscall # end  addi $sp,$sp,16  b loop  # end  find\_iteration:  # 调用迭代查找某个数字是否属于斐波那契数列  addi $sp,$sp,-8  sw $a0,($sp) # 将find\_Fibonacci\_iteration函数的参数n压栈  jal find\_Fibonacci\_iteration  lw $a0,4($sp) # 加载并打印find\_Fibonacci\_iteration函数返回值  li $v0 1  syscall  li $v0,4  la $a0,endl  syscall # end  addi $sp,$sp,8  b loop  # end  end\_all: # 程序结束  addi $sp,$sp,4  # 打印结束界面  li $v0 4  la $a0 exit\_screen  syscall  # end  li $v0,10  syscall  Decin: # 读取十进制函数 Decin(m,type)  Decin\_begin:  lw $a0,($sp) # 读取参数type  li $v0,4 # 打印提示信息  beqz $a0,tip\_1  la $a0,tip2  b endif  tip\_1:  la $a0,tip1  endif:  syscall # end  la $a0,buffer # 输入字符串，对应局部变量fp  li $a1,10  li $v0,8  syscall # end  li $8,0x20 # 对应局部变量sp ' '  li $9,0x0a # 对应局部变量enter '\n'  li $10,0x2d # 对应局部变量fh '-'  do\_space1: # 检测前空格  lb $17, ($a0) # 对应局部变量temp  addi $a0, $a0, 1 # fp++  beqz $17, out\_1 # 检测出空串  beq $17, $9, out\_1 # 检测出空串  beq $17, $8, do\_space1 # 若检测出空格，则继续检测  beq $17, $10, out\_3 # 检测出负号  li $12,0x30 # '0'  li $13,0x39 # '9'  move $v0,$0 # 对应局部变量m  li $14,10 # 权,对应局部变量basis  do\_zero:  bne $17, $12, loop1 # 判断是否为‘0’  lb $17, ($a0)  beqz $17, out\_3 # 检测0，出错  beq $17, $9, out\_3 # 检测0，出错  addi $a0, $a0, 1 # fp++  b do\_zero  loop1:  beq $17, $8, do\_space2  beqz $17, out\_4 # 结束  beq $17, $9, out\_4 # 结束  blt $17, $12, out\_2 # 小于'0'，出错  bgt $17, $13, out\_2 # 大于'9'，出错  mulo $v0, $v0, $14  addi $17, $17, -48 # '?' - '0'  add $v0, $v0, $17  lb $17, ($a0) # 更新temp  addi $a0, $a0, 1 # fp++  b loop1  do\_space2: # 检测后空格  lb $17, ($a0) # 更新temp  addi $a0, $a0, 1 # fp++  beqz $17, out\_4 # 结束  beq $17, $9, out\_4 # 结束  beq $17, $8, do\_space2  b out\_2  out\_1:  li $v0,4 # 打印err1  la $a0,err1  syscall # end  b Decin\_begin  out\_2:  li $v0,4 # 打印err2  la $a0,err2  syscall # end  b Decin\_begin  out\_3:  li $v0,4 # 打印err3  la $a0,err3  syscall # end  b Decin\_begin  out\_4:  sw $v0,4($sp) # 函数返回值压栈  jr $ra # 返回  # Decin end  print\_Fibonacci\_iteration: # 斐波那契数列迭代算法 print\_Fibonacci\_iteration(n)  lw $8,($sp) # 加载print\_Fibonacci\_iteration函数参数n  move $9,$0 # 对应局部变量pre  li $17,1 # 对应局部变量cur  li $v0,1 # 打印斐波那契数列第一个数  li $a0,1  syscall  li $v0,4  la $a0,sp  syscall # end  loop2:  addi $8,$8,-1 # n = n - 1  blez $8,end2 # n <= 0则跳出循环  move $10,$17 # 对应局部变量temp = pre  add $17,$17,$9 # cur = cur + pre  move $9,$10 # pre = temp  li $v0,1 # 打印cur  move $a0,$17  syscall  li $v0,4  la $a0,sp  syscall # end  b loop2  end2:  li $v0,4 # 打印换行符  la $a0,endl  syscall # end  jr $ra  # print\_Fibonacci\_iteration end  find\_Fibonacci\_iteration: # 斐波那契数列迭代算法 find\_Fibonacci\_iteration(index,n)  lw $8,($sp) # 加载find\_Fibonacci\_iteration函数参数n  move $9,$0 # 对应局部变量pre  li $17,1 # 对应局部变量cur  li $v0,1 # 对应局部变量index  loop3:  beq $8,$17,end3 # n == cur,找到跳出循环  blt $17,$8,content3 # n < cur,继续查找  li $v0,-1 # n>cur,没找到，跳出循环  b end3  content3:  addi $v0,$v0,1 # index = index + 1  move $10,$17 # 对应局部变量temp = pre  add $17,$17,$9 # cur = cur + pre  move $9,$10 # pre = cur  b loop3  end3:  sw $v0,4($sp) # 函数返回值压栈  jr $ra # 返回  # find\_Fibonacci\_iteration end  Fibonacci\_recursive : # 斐波那契数列递归算法 Fr(n,Fr(n-1),Fr(n-2),re)  lw $8,($sp) # 加载Fibonacci\_recursive函数参数n  li $9,1 # 对应返回值re  li $10,2  ble $8,$10,re # n<=2,则直接返回  addi $8,$8,-1 # n-1  addi $sp,$sp,-20  sw $8,($sp) # n-1入栈  sw $ra,16($sp) # $ra入栈  jal Fibonacci\_recursive # Fr(n-1, , ,r')  lw $9,12($sp) # 加载r'  lw $ra,16($sp) # 加载$ra  addi $sp,$sp,20  sw $9,4($sp) # r'入栈，对应Fr(n-1)  lw $8,($sp) # 加载n  addi $8,$8,-2 # n-2  addi $sp,$sp,-20  sw $8,($sp) # n-2入栈  sw $ra,16($sp) # $ra入栈  jal Fibonacci\_recursive # Fr(n-2, , ,r'')  lw $9,12($sp ) # 加载r''  lw $ra,16($sp ) # 加载$ra  addi $sp,$sp,20  sw $9,8($sp) # r''入栈，对应Fr(n-2)  lw $8,4($sp) # 加载Fr(n-1)  add $9,$8,$9 # re = Fr(n-1) + Fr(n-1)  re:  sw $9,12($sp) # 函数返回值入栈  jr $ra # 返回  # Fibonacci\_recursive end |

# 四、寄存器使用对照表

|  |  |
| --- | --- |
| **Decin** | |
| 输入参数type | $a0、($sp) |
| fp | $a0 |
| temp | $17 |
| sp | $8 |
| enter | $9 |
| fh | $10 |
| basis | $14 |
| 返回值 | $v0、4($sp) |

|  |  |
| --- | --- |
| **print\_Fibonacci\_iteration** | |
| 输入参数n | $8、($sp) |
| pre | $9 |
| cur | $17 |
| temp | $10 |
| 返回值 | 无 |

|  |  |
| --- | --- |
| **Fibonacci\_recursive** | |
| 输入参数n | $8、($sp) |
| Fr(n-1) | $9、$8、4($sp) |
| Fr(n-2) | $9、8($sp) |
| 返回值re | $9、12($sp) |

|  |  |
| --- | --- |
| **find\_Fibonacci\_iteration** | |
| 输入参数n | $8、($sp) |
| pre | $9 |
| cur | $17 |
| temp | $10 |
| 返回值index | $v0、4($sp) |

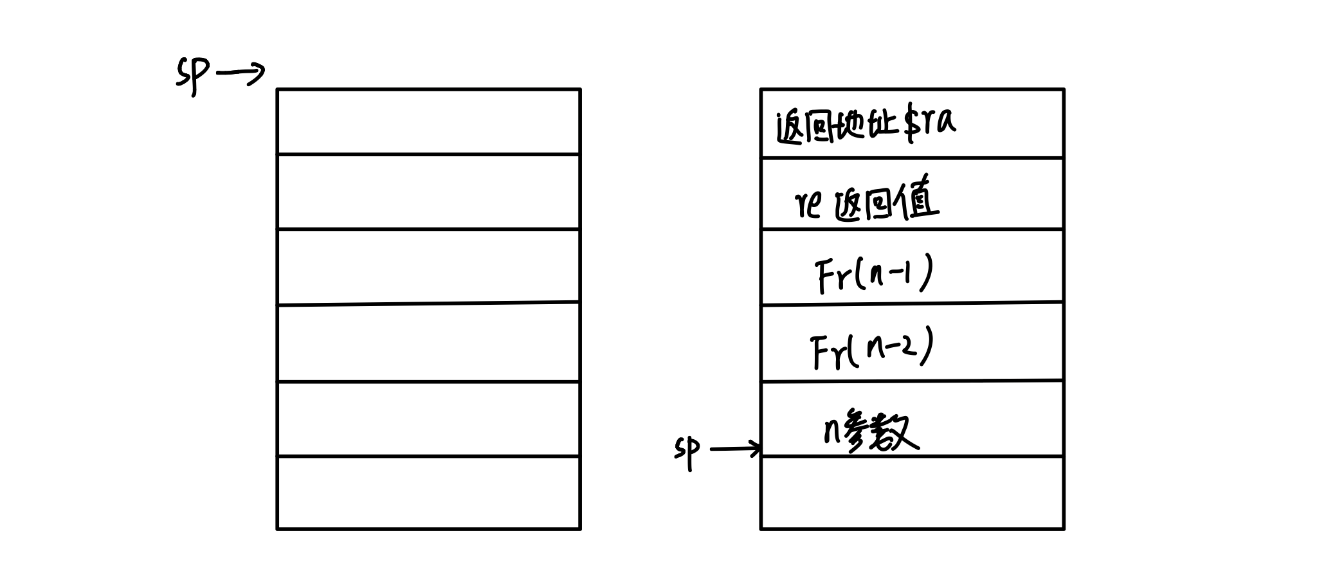
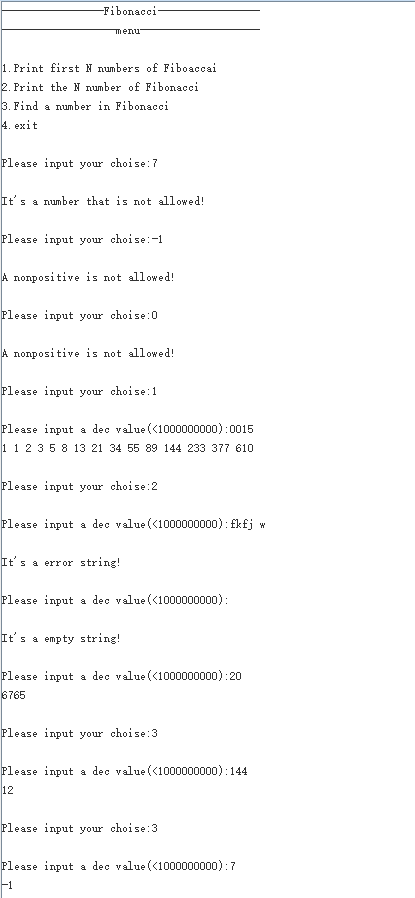
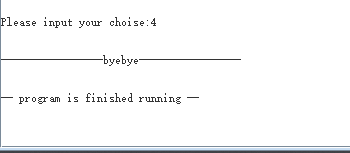


Figure 1 Fibonacci\_recursive函数的栈表

# 五、程序运行结果





# 六、心得体会

这个程序本身并不难,用C++进行伪代码描述的时候很快就实现了，但是用汇编实现的时候就显得烦琐多了。

首先是对Decin输入函数进行处理，因为要进行相关检错和纠错，用系统自己的整数输入是不可行的，所以使用了输入字符串进行处理，对空字符串、负整数、零、非法整数进行检错，对含有前导零或前导空格或后导空格进行剔除。对于这个函数，为了保证检错和纠错的正确率，进行了大量的调试与测试。

其次是迭代相关的函数find\_Fibonacci\_iteration、print\_Fibonacci\_iteration，伪代码中使用了循环结构，汇编中则熟练使用条件跳转语句，在有伪代码进行对照的前提下，实现较为简单。

再者是递归相关的函数Fibonacci\_recursiv，这个函数在伪代码中十分简单，但是在汇编中因为设计到栈的使用，需要另外绘制函数的栈表（figure 1），此函数也是可重入函数。

最后，为了汇编代码好看点，我还用python写了个程序，把注释规范了一下。

这个程序并不是一天之内完成的，中间因为某些原因隔了大概一礼拜，这时候，注释就显得尤为重要。我清晰地记得有一行给寄存器赋值的代码，我之前没写注释，再次看的时候又没搞懂它的用处，就随手把它删了，结果后面测试时出了bug，找了好久才发现是之前那行代码的原因，气人。还有一次我的循环没处理好，结果运行程序的时候，卡死了，不得不把MARS4\_5.jar关掉重开。

总而言之，代码这种东西都是熟能生巧，只要写的多，那些汇编语句就不怕记不住，不过就像老师说的，我们现在学的只是皮毛，老师之前上学的时候布置的作业都是实现那种图形界面的，那可比我们这个复杂多了。