**上机实验报告1 斐波那契数列**

919113100209 代领

**设计思想：**

overflow:输出

err2并退出

outerr:输出

err1并退出。

Hexout:

在课本提供的函数基础上拓展。

FIB:

检查N是否大于0如果小于0调用outerr。

初始化前两个数字。

循环（当前第i个数<=N）：当前数字等于前两个数字之和，判断是否溢出，溢出则跳到overflow，否则就赋值给数组，i++后继续循环。

output:

遍历fib数组：输出其十进制值，然后输出十六进制值（调用Hexout），最后调用nline，继续循环。最后退出。

先得出fib数组，然后输出。

main:

输入N

得出fib数组，调用FIB

输出结果，调用output

**源码：**

.data

N: .word

array: .space 2048

str1: .asciiz "input N: "

str2: .asciiz "output:\n"

nxtline: .asciiz "\n"

kk: .asciiz " "

err1: .asciiz "N is not greater than 0\n"

err2: .asciiz "OverFlow!\n"

buf: .space 20 #1个空格+2字符”0x”+8字符+1个表示字符串结束的空字符

.globl main

.text

main: #主函数

li $v0,4

la $a0,str1 # 提示输入

syscall

li $v0,5 #输入

syscall

la $a2,($v0) #把N放到a2

la $a1,array #把array放到a1

li $v0,4

la $a0,str2 #提示输出

syscall

la $a0,($a2) #N的值赋给a0

li $t1,2

jal FIB

jal output

FIB: #得出斐波那契数列

bltz $a0,outerr #检查N是否大于0

li $t7,1 #临时储存1

sw $t7,4($a1) #array[0]=1

sw $t7,12($a1) #array[1]=1

loop0: #循环

bge $t1,$a0,ret #t1没有超过a0就一直循环

move $t0,$t1 #t0作地址，t1作下标

mulu $t0,$t0,8

add $t0,$a1,$t0 #得出当前这个数的地址

lw $t4,-16($t0) #前两个数

lw $t5,-12($t0)

lw $t6,-8($t0) #前一个数

lw $t7,-4($t0)

addu $t3,$t5,$t7 #前两个数相加

sltu $t2,$t3,$t7

addu $t2,$t2,$t4

addu $t2,$t2,$t6

bgtu $t2,$t7,overflow#如果相加比前一个数小就是溢出

sw $t2,($t0) #把t7赋给t0所指的位置

sw $t3,4($t0)

addi $t1,$t1,1 #计数器加一

b loop0

output: #输出函数

li $t5,0 #t5计数

move $t6,$a0 #t6存上界

la $t4,array+4 #t4用来遍历数组

loop1:

bge $t5, $t6,exit #大于或等于上界就退出

move $a0,$t5

li $v0,1 #下角标

syscall

li $v0,4

la $a0,kk #空格

syscall

lw $t2, 0($t4) #赋值

move $a0, $t2

bgeu $t5,46,hex

li $v0, 1 #输出十进制数

syscall

hex:

move $a3,$t2

lw $a0,-4($t4)

addi $t4, $t4, 8 #下标后移

jal Hexout #输出十六进制数

la $a0,buf

li $v0,4

syscall

jal endl #换行

addi $t5, $t5, 1 #计数器加一

j loop1

endl: #换行函数

la $a0,nxtline

li $v0,4

syscall

jr $ra

ret:

jr $ra

Hexout: #输出十六进制数

la $a1, buf #缓冲区地址

li $a2, 10 #循环次数

addi $t1, $a1, 10 #从位置buf+10处开始存放16进制数

loop:

andi $t0, $a0, 0x0f #取a0的低4位

srl $a0, $a0, 4 #a0右移4位

bge $t0, 10, char #t0大于等于10跳转到为A-F处理

addi $t0, $t0, 0x30 #0的ASCII码为0x30，在原先基础上加0x30

b put

char:

addi $t0, $t0, 0x37 #A的ASCII码为65，在原先基础上加(65-10)

put:

sb $t0, ($t1) #放置字符

addi $t1, $t1, -1 #放置位置前移一个字符

addi $a2, $a2, -1 #将循环次数减1

bgtz $a2, loop #判断循环是否结束

out:

sb $0, 19($a1) #将0x00存储到最后一个位置（位置11）

li $t0, 0x78

sb $t0, 2($a1) #将0x78（字符x）存储到位置2

li $t0, 0x30

sb $t0, 1($a1) #将0x30（字符0）存储到位置1

li $t0, 0x20

sb $t0, ($a1) #将0x20（字符空格）存储到位置0

li $a2,8

addi $t1, $a1, 18 #从位置buf+18处开始存放16进制数

loop\_2:

andi $t0, $a3, 0x0f #取a3的低4位

srl $a3, $a3, 4 #a0右移4位

bge $t0, 10, char\_2 #t0大于等于10跳转到为A-F处理

addi $t0, $t0, 0x30 #0的ASCII码为0x30，在原先基础上加0x30

b put\_2

char\_2:

addi $t0, $t0, 0x37 #A的ASCII码为65，在原先基础上加(65-10)

put\_2:

sb $t0, ($t1) #放置字符

addi $t1, $t1, -1 #放置位置前移一个字符

addi $a2, $a2, -1 #将循环次数减1

bgtz $a2, loop\_2 #判断循环是否结束

jr $ra

overflow: #输出溢出的报错并退出

la $a0,err2

li $v0,4

syscall

j exit

outerr: #输出N不大于0的报错并退出

la $a0,err1

li $v0,4

syscall

j exit

exit: #退出

li $v0,10

syscall

**测试过程和结果：**

TEST1:

输入：

80

输出：

input N: 80

output:

0 1 0x0000000000000001

1 1 0x0000000000000001

2 2 0x0000000000000002

3 3 0x0000000000000003

4 5 0x0000000000000005

5 8 0x0000000000000008

6 13 0x000000000000000D

7 21 0x0000000000000015

8 34 0x0000000000000022

9 55 0x0000000000000037

10 89 0x0000000000000059

11 144 0x0000000000000090

12 233 0x00000000000000E9

13 377 0x0000000000000179

14 610 0x0000000000000262

15 987 0x00000000000003DB

16 1597 0x000000000000063D

17 2584 0x0000000000000A18

18 4181 0x0000000000001055

19 6765 0x0000000000001A6D

20 10946 0x0000000000002AC2

21 17711 0x000000000000452F

22 28657 0x0000000000006FF1

23 46368 0x000000000000B520

24 75025 0x0000000000012511

25 121393 0x000000000001DA31

26 196418 0x000000000002FF42

27 317811 0x000000000004D973

28 514229 0x000000000007D8B5

29 832040 0x00000000000CB228

30 1346269 0x0000000000148ADD

31 2178309 0x0000000000213D05

32 3524578 0x000000000035C7E2

33 5702887 0x00000000005704E7

34 9227465 0x00000000008CCCC9

35 14930352 0x0000000000E3D1B0

36 24157817 0x0000000001709E79

37 39088169 0x0000000002547029

38 63245986 0x0000000003C50EA2

39 102334155 0x0000000006197ECB

40 165580141 0x0000000009DE8D6D

41 267914296 0x000000000FF80C38

42 433494437 0x0000000019D699A5

43 701408733 0x0000000029CEA5DD

44 1134903170 0x0000000043A53F82

45 1836311903 0x000000006D73E55F

46 0x00000000B11924E1

47 0x000000011E8D0A40

48 0x00000001CFA62F21

49 0x00000002EE333961

50 0x00000004BDD96882

51 0x00000007AC0CA1E3

52 0x0000000C69E60A65

53 0x0000001415F2AC48

54 0x000000207FD8B6AD

55 0x0000003495CB62F5

56 0x0000005515A419A2

57 0x00000089AB6F7C97

58 0x000000DEC1139639

59 0x000001686C8312D0

60 0x000002472D96A909

61 0x000003AF9A19BBD9

62 0x000005F6C7B064E2

63 0x000009A661CA20BB

64 0x00000F9D297A859D

65 0x000019438B44A658

66 0x000028E0B4BF2BF5

67 0x000042244003D24D

68 0x00006B04F4C2FE42

69 0x0000AD2934C6D08F

70 0x0001182E2989CED1

71 0x0001C5575E509F60

72 0x0002DD8587DA6E31

73 0x0004A2DCE62B0D91

74 0x000780626E057BC2

75 0x000C233F54308953

76 0x0013A3A1C2360515

77 0x001FC6E116668E68

78 0x00336A82D89C937D

79 0x00533163EF0321E5

TEST2:

输入：

100

输出：

input N: 100

output:

OverFlow!

TEST3:

输入：

-1

输出：

input N: -1

output:

N is not greater than 0

**设计调试体会：**

总的感觉是汇编语言频繁的跳转，稍不注意就会写出bug，不写注释经常会忘记某个存储器的作用到底是存的地址还是存的值。还有就是时刻要搞清楚自己是要用寄存器给寄存器赋值，还是寄存器给内存赋值，还是内存给寄存器赋值，如果没有清晰的思路，容易写出地址没有对齐的错误。

最后聊一下体会：写了汇编才感觉到，所谓的高级语言真的是“伪码”。我使用的MARS编写，虽然MARS已经非常小巧便捷，但是我认为还有改进的地方：观察寄存器值的时候应该设置一下高亮，否则真的容易把眼睛看花；单步执行的时候不能直接在Edit里面观察，导致要记住Edit里面的行号，才可以在Execute里面打断点。改进一下这两点Debug的效率和体验会提高很多。这些任务我可能在未来实践。