Atul Pande

2101CS88

Q1)

1.

Code->

addi $s0,$0,5 #f

addi $s1,$0,10 #g

addi $s2,$0,7 #h

addi $s3,$0,10 #i

addi $s4,$0,20 #j

bne $s3,$s4, label

add $s0,$s1,$s2 #f=g+h

label:

sub $s0,$s0,$s3 #f=f-i

li $v0 ,1

move $a0,$s0

syscall

output->

-5

2.

Code->

addi $s0,$0,7 #f

addi $s1,$0,19 #g

addi $s2,$0,8 #h

addi $s3,$0,20 #i

addi $s4,$0,20 #j

bne $s3,$s4, label

add $s0,$s1,$s2 #f=g+h

j done

label:

sub $s0,$s0,$s3 #f=f-i

done:

li $v0 ,1

move $a0,$s0

syscall

output-> 27

3.

Code->

addi $s0,$0,1 #pow

addi $s1,$0,0 #x

addi $t0,$0,128

while:

beq $s0,$t0,done

add $s1,$s1,1

sll $s0,$s0,1

j while

done:

li $v0 ,1

move $a0,$s1

syscall

output->7

4.

Code->

.data

str1: .asciiz "Number is positive"

str2: .asciiz "Number is negative"

.text

addi $s0,$0,7

bgt $s0,$0, label

li $v0,4

la $a0,str2

syscall

j done

label:

li $v0,4

la $a0,str1

syscall

done:

output-> Number is positive

5.

Code-> addi $s0,$0,0 #sum

addi $s1,$0,0 #i

addi $t0,$0,10

for:

beq $s1,$t0,done

add $s0,$s0,$s1

add $s1,$s1,1

j for

done:

li $v0 ,1

move $a0,$s0

syscall

output-> 45

6.

Code->

.data

endl: .asciiz "\n"

.text

# $s0 = array base address, $s1 = i

# initialization code

lui $s0, 0x1000 # $s0 = 0x23B80000

ori $s0, $s0, 0x1000 # $s0 = 0x23B8F000

addi $s1, $0, 0 # i = 0

addi $t2, $0, 10 # $t2 = 1000

loop:

slt $t0, $s1, $t2 # i < 1000?

beq $t0, $0, done # if not then done

sll $t0, $s1, 2 # $t0 = i \* 4 (byte offset)

add $t0, $t0, $s0 # address of array[i]

lw $t1, 0($t0) # $t1 = array[i]

add $t1, $t1, $s1

sll $t1,$t1,3 # $t1 = array[i] \* 8

sw $t1, 0($t0) # array[i] = array[i] \* 8

addi $s1, $s1, 1 # i = i + 1

li $v0, 1

move $a0, $t1

syscall

li $v0, 4 # endlline

la $a0, endl

syscall

j loop # repeat

done:

Output->

0

8

16

24

32

40

48

56

64

72

7.

Code->

addi $s0,$0,0 #a

addi $s1,$0,3 #b

addi $s2,$0,6 #c

main:

jal simple

add $s0,$s1,$s2

li $v0 ,1

move $a0,$s0

syscall

li $v0,10

syscall

simple:

jr $ra

output-> 9

8.

Code->

addi $s0,$0,2

addi $s1,$0,3

addi $s2,$0,4

addi $s3,$0,5

addi $s4,$0,0

main:

jal diffofsum

add $s4,$v0,$0

li $v0 ,1

move $a0,$s4

syscall

li $v0,10

syscall

diffofsum:

add $t1,$s0,$s1

add $t2,$s2,$s3

sub $t0,$t1,$t2

add $v0,$t0,$0

jr $ra

output-> -4

Q2)

No, there are no errors

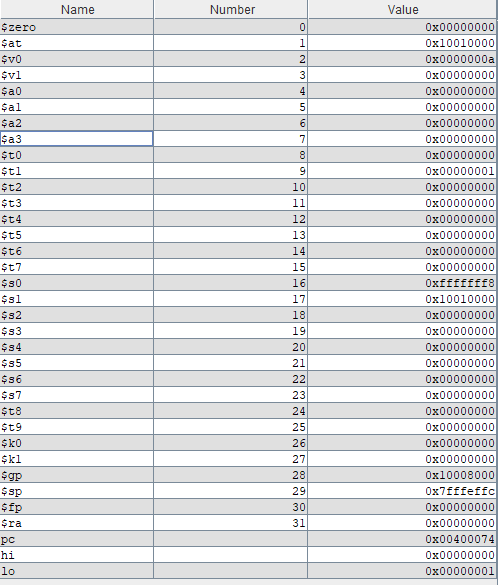
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Fib No.** | **Value** | **ALU** | **Jump** | **Branch** | **Memory** | **Total** |
| 1 | 0 | 19 | 2 | 2 | 2 | 30 |
| 5 | 3 | 226 | 38 | 38 | 80 | 403 |
| 9 | 21 | 1748 | 334 | 309 | 808 | 3236 |
| 13 | 144 | 12275 | 24210 | 2190 | 5986 | 22914 |
| 17 | 987 | 84522 | 16686 | 15106 | 41664 | 158047 |
| 21 | 6765 | 579804 | 114582 | 103657 | 286392 | 1084520 |
| 25 | 46368 | 3974611 | 785618 | 710618 | 1963970 | 7434918 |
| 29 | 317811 | 27243058 | 5385014 | 4870814 | 13462448 | 7434918 |

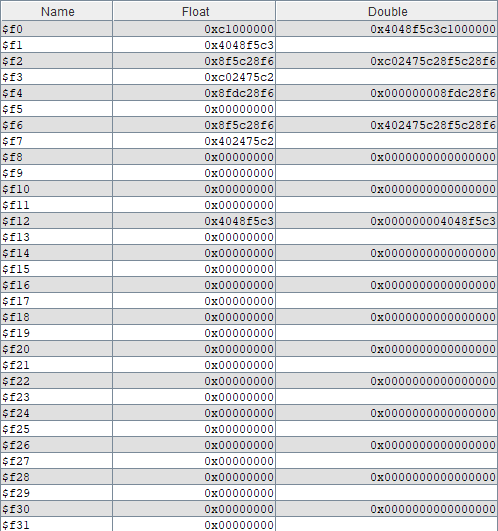
Yes, there is an optimal way. We can write it iteratively as it will take lesser total instructions and therefore is faster than recursion. Also, recursion takes a lot of memory as compared to iterative way.

After n=29 it takes a lot of time to compute

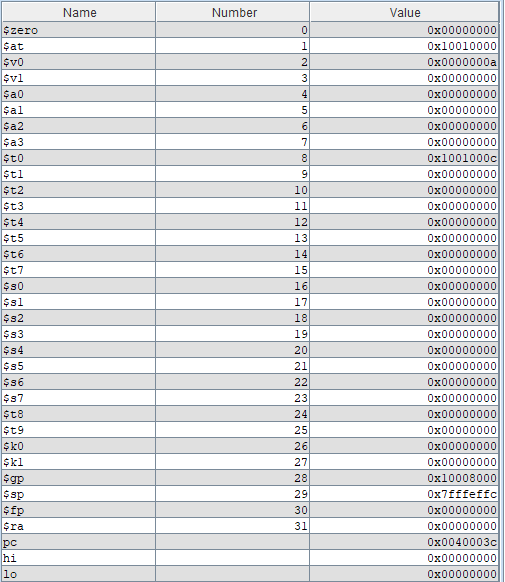
Q3)

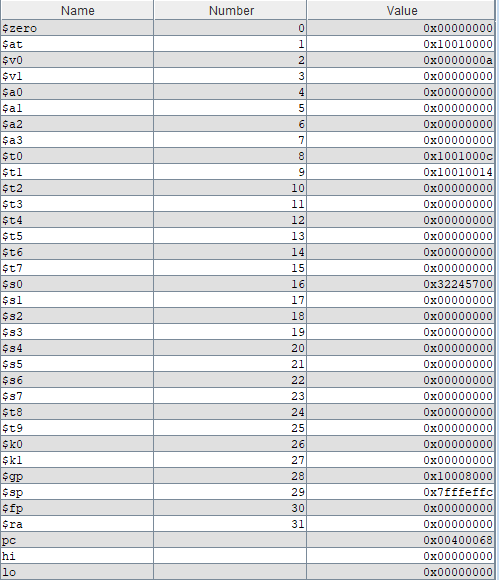
1. Float.asm



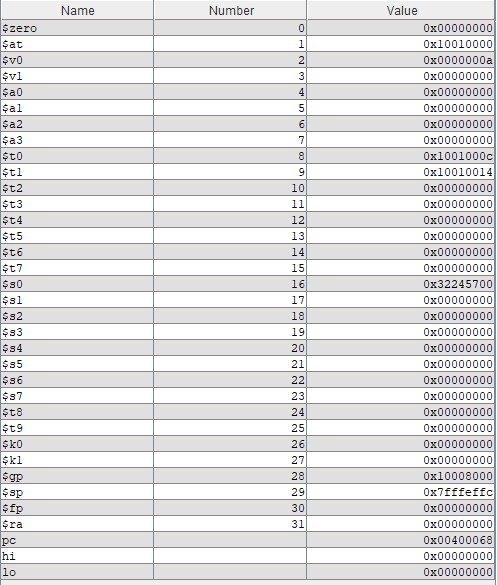


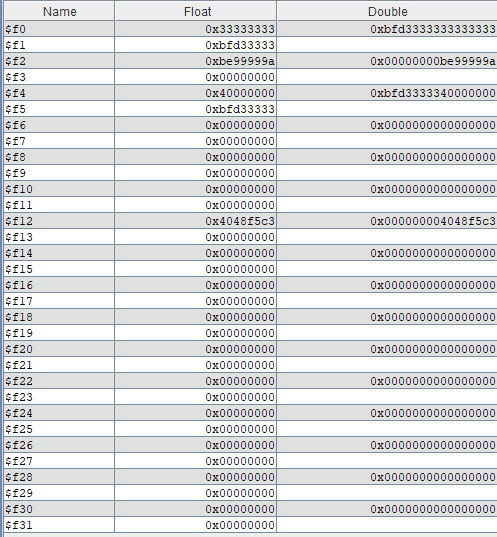
1. Float\_cast.asm





1. Float\_cast1\_1.asm





|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Integer/ Float no** | **$1-$31** | **$f0-$f31** | **Double** | |
| x=(932213431)10 | 0x379072b7 | 0x379072c0 | N/A | N/A |
| x=(532325811)10 | 0x1fbaa5b3 | 0x1fbaa5b3 | 0xb3000000 | 0x41bfaa5 |
| f=(6.63233411E28)10 | 0x6f564d62 | 0x6f564d62 | 0x40000000 | 0x45eac9ac |
| d=(-1.9)10 | 0xBFFE666666666666 | 0xbffe666660000000 | 0x66666666 | 0xbffe6666 |