**CS210: (Lab 7)**

**Computer Architecture Lab**

**Flow Control, MIPS Functions, Floating-Point**

For each of the HL Code, test the equivalent MIPS Code with various initial values of registers

**Your code/output here**

li $t0, 5 # load the value 5 into register $t0 (representing i)

li $t1, 7 # load the value 7 into register $t1 (representing j)

li $t2, 3 # load the value 3 into register $t2 (representing g)

li $t3, 4 # load the value 4 into register $t3 (representing h)

li $t4, 0 # load the value 0 into register $t4 (representing f)

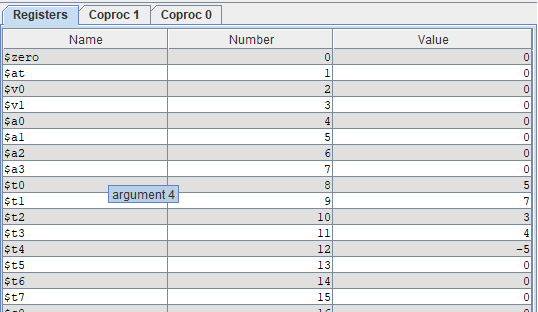
# if(i==j) f= g+h;

bne $t0, $t1, L1

# f=f-i;

add $t4, $t2, $t3 # add g and h and store the result in f

L1: sub $t4, $t4, $t0 # subtract i from f

****

**Your code/output here**

li $s1, 5 # load the value 5 into register $s1

li $s2, 3 # load the value 3 into register $s2

li $s3, 7 # load the value 7 into register $s3

li $s4, 6 # load the value 6 into register $s4

li $s0, 0 # load the value 0 into register $s0

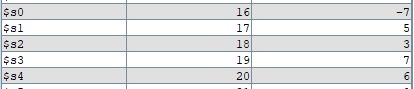
bne $s3,$s4,L1

add $s0,$s1,$s2

j done

L1: sub $s0,$s0,$s3

done:

****

**Your code/output here**

.data

message: .asciiz "The value of s1 is: "

newline: .asciiz "\n"

.text

addi $s0, $0, 1

add $s1, $0, $0

addi $t0, $0, 128

while: bge $s0, $t0, done

sll $s0, $s0, 1

addi $s1, $s1, 1

j while

done:

li $v0, 4

la $a0, message

syscall

li $v0, 4

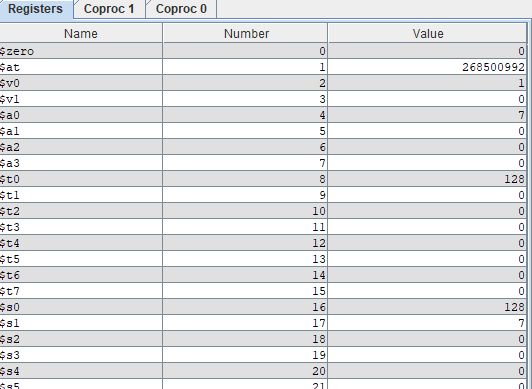
la $a0, newline

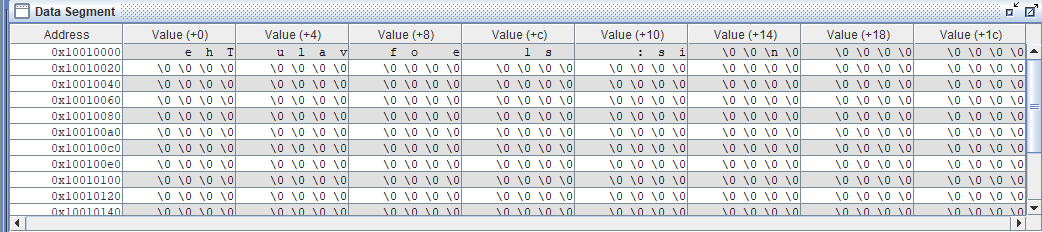
syscall

li $v0, 1

move $a0, $s1

syscall

****

****

**Your code/output here**

.text

# Initialize $s0 with value -7

li $s0, -7

# Check if $s0 is greater than zero

bgtz $s0, positive

# If $s0 is not greater than zero, print "Number is negative"

li $v0, 4 # system call for printing strings

la $a0, negative\_string # load address of "Number is negative" string into $a0

syscall

j end

positive:

# If $s0 is greater than zero, print "Number is positive"

li $v0, 4 # system call for printing strings

la $a0, positive\_string # load address of "Number is positive" string into $a0

syscall

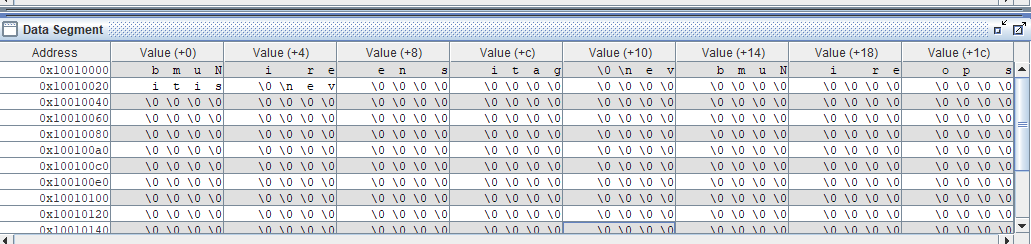
end:

.data

# Data section

negative\_string: .asciiz "Number is negative\n"

positive\_string: .asciiz "Number is positive\n"

****

Explaination:-

The ‘bgtz’ instruction will checks if the value stored in $s0 is greater than zero. If value stored in $s0 greater than 0 then, the program jumps to the label positive, otherwise it continues to the next instruction. The system calls li $v0, 4 and syscall are used to print strings, with the string to be printed being passed in the $a0 register. The strings "Number is negative" and "Number is positive" are stored in the data section and their addresses are loaded into $a0 as needed.

In this code I took value in $s0 = -7 i.e negative so Output “Number is negative” is displayed as shown in picture.

**Your code/output here**

.data

message: .ascii "The sum of 0 to 10 is: "

newline: .ascii "\n"

.text

.globl main

main:

addi $s1,$0,0 # initialize $s1 to 0

add $s0,$0,$0 # initialize $s0 to 0

addi $t0,$0,10 # initialize $t0 to 10

for:

add $s1,$s1,$s0 # add $s0 to $s1

addi $s0,$s0,1 # increment $s0

bne $s0,$t0,for # repeat until $s0 == $t0

# print the sum

li $v0,4 # code for printing string

la $a0,message # load address of message

syscall # print message

li $v0,1 # code for printing integer

move $a0,$s1 # move $s1 to $a0

syscall # print sum

li $v0,4 # code for printing string

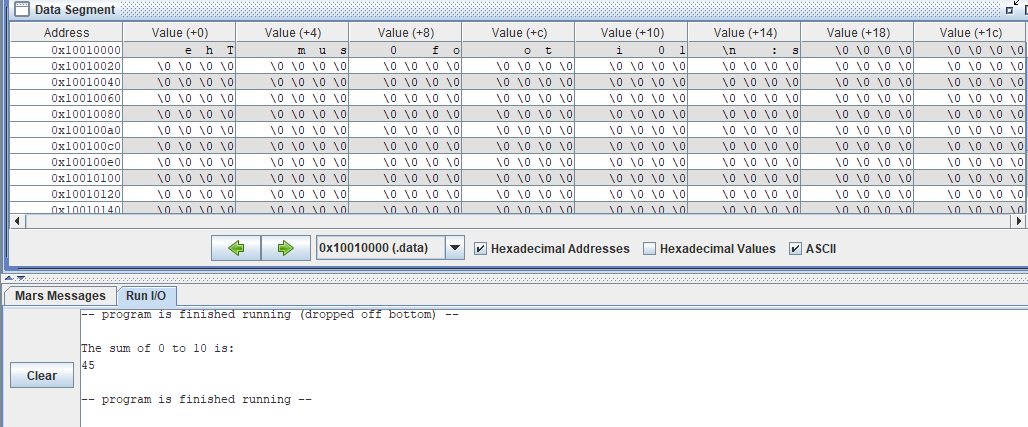
la $a0,newline # load address of newline

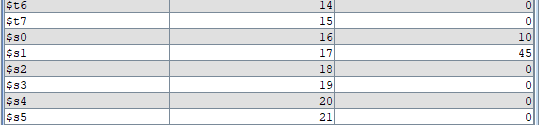
syscall # print newline

# exit program

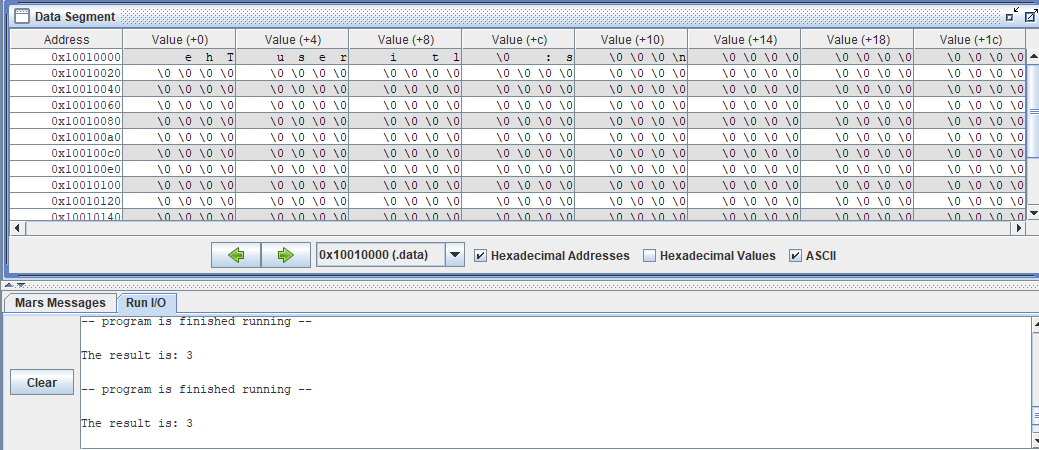
li $v0,10

syscall

****

****

**Your code/output here**

****

.data

msg: .asciiz "The result is: "

newline: .asciiz "\n"

.text

main:

# Call the simple function

jal simple

# Print the message "The result is: "

la $a0, msg

li $v0, 4

syscall

# Print the result stored in $s0

move $a0, $s0

li $v0, 1

syscall

# Print a newline character

la $a0, newline

li $v0, 4

syscall

# Exit the program

li $v0, 10

syscall

simple:

# Store the sum of 1 and 2 in $s0

addi $s0, $0, 1

addi $s1, $0, 2

add $s0, $s0, $s1

# Return to the main function

jr $ra

**Your code/output here**

Q: Consider MIPS assembly to compute the Nth Fibonacci number. Assume N is passedto your function in register. Sample Fibonacci recursive implementations are given. Inspect the code and analyze code and identify errors( if any). Analyze the computational instructions ( in terms of total instruction count, ALU, jump, branch, memory). Is it possible to optimize the

**(use MARS assembler: MARS-> tools-> instruction statistics, connect to MIPS before you run fib.asm**

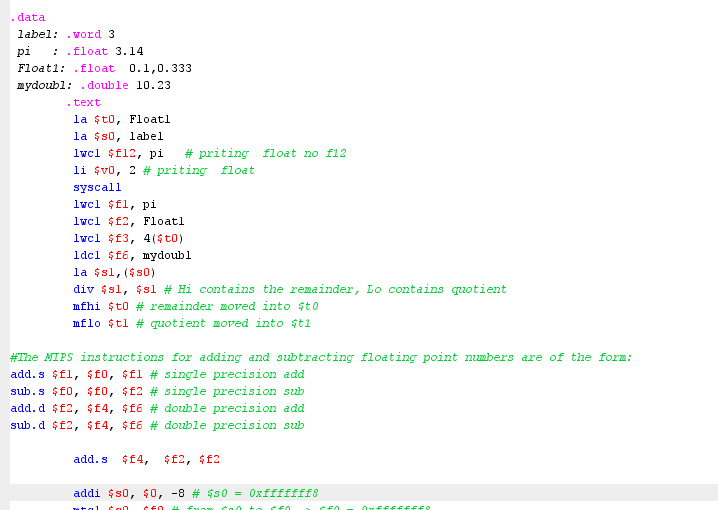
Recursive

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Fib no | **value** | ALU | Jump | branch | memory | Total |
| 1 |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |
| …. |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |

**Your Conclusions:**

**Q: Examine the float.asm, float\_cast and float\_cast1.asm. Study the int, float, and double representation of numbers and type casting. Create your own examples and examine register contents.**

**Use MARS->tools-Floating point representation to view the details of representation)**

****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **/Integer/ Float no** | **$1-$31** | **$f0-$f31** | **Double** | |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Submission:**

**Demonstrate to TAs**

Submit code and Report through

<https://u.pcloud.com/#page=puplink&code=SIPkZvCxgnOEMwC4xROrHDpjcKuyacHiV>