**Lab Report  
Lab 7: Subroutine Call and Passing Parameters Using the Stack**

|  |  |
| --- | --- |
| Student ID | 20235890 |
| Student Name | Nguyễn Đức Anh |

**Assignment 1:**

**Code:**

# Lab 7, Assignment 1

.text

main:

input:

    li a7, 5    # input a integer

    ecall       # system call

    jal abs     # jump and link to abs procedure

    li a7, 10   # terminate

    ecall       # system call

end\_main:

abs:

    sub s0, zero, a0    # put -a0 in s0; in case a0 < 0

    blt a0, zero, done  # if a0<0 then done

    add s0, a0, zero    # else put a0 in s0

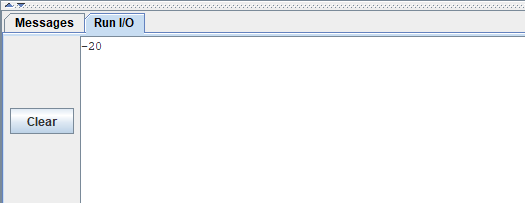
done:

    jr ra   # jump to return address

**Explaination:** Using system call to input a value to a0 and do abs procedure and save the value on s0.

**Result:**

Input:



Output:

A screenshot of a table

Description automatically generated

**Assignment 2:**

**Code:**

# Lab 7, Assignment 2

.text

main:

    li a0, 4    # load test input

    li a1, 12

    li a2, 9

    jal max     # call max procedure

    li a7, 10   # terminate

    ecall

end\_main:

max:

    add s0, a0, zero    # copy a0 in s0; largest so far

    sub t0, a1, s0      # compute a1 - s0

    blt t0, zero, okay  # if a1 - v0 < 0 then no change

    add s0, a1, zero    # else a1 is largest thus far

    okay:

    sub t0, a2, s0      # compute a2 - v0

    blt t0, zero, done  # if a2 - v0 <0 then no change

    add s0, a2, zero    # else a2 is largest overall

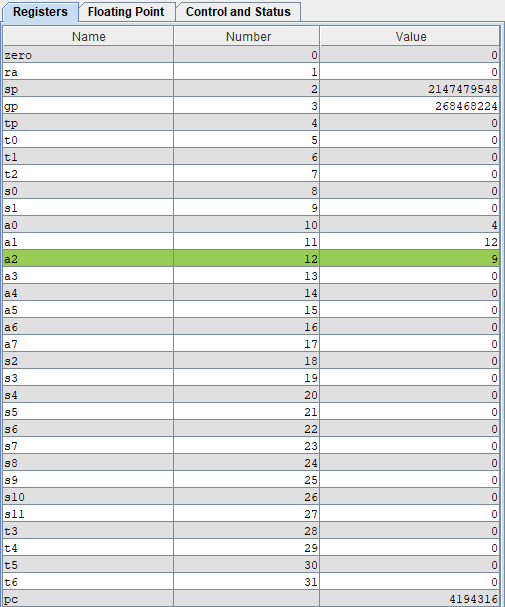
done:

    jr ra   # return to calling program

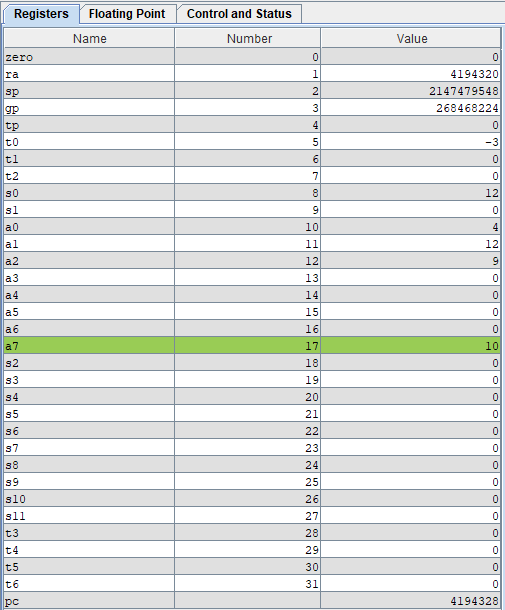
**Explaination:** Finding the maximum value on three parameters a0, a1, a2 and store on s0 by max procedure.

**Result:**

Input:



Output:



**Assignment 3:**

**Code:**

# Lab 7, Assignment 3

.text

    li s0, 0x00000005

    li s1, 0xfffffffe

push:

    addi sp, sp, -12 # adjust the stack pointer

    sw s0, 8(sp) # push s0 to stack

    sw s1, 4(sp) # push s1 to stack

work:

sum:

    add s2, s0, s1  # s2 = s0 + s1

push\_s2:

    sw s2, 0(sp)    # push s2 to stack

pop:

    lw s0, 0(sp) # pop from stack to s0

    lw s1, 4(sp) # pop from stack to s1

    lw s2, 8(sp) # pop from stack to s2

    addi sp, sp, 12 # adjust the stack pointer

**Explaination:** Using stack to push s0, s1, the sum s2 = s0 + s1 and pop it by swapping the value s0 = s2, s1 = s1 and s2 = s0.

**Result:**

Input:

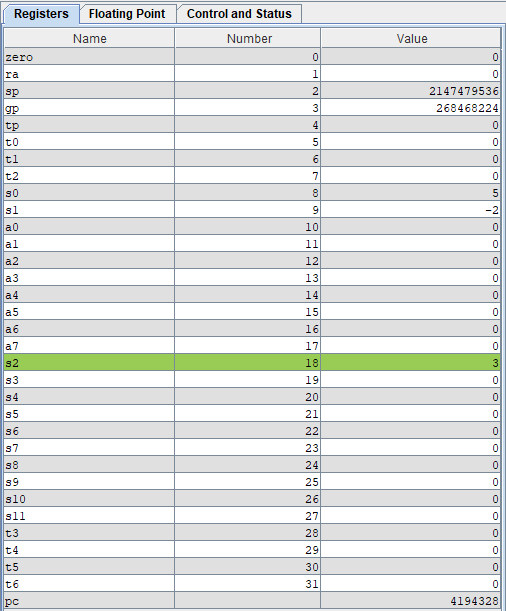
A screenshot of a table

Description automatically generated

A screenshot of a computer

Description automatically generated

Sum:



Output:

A screenshot of a table

Description automatically generated

**Assignment 4:**

**Code:**

# Lab 7, Assignment 4

.data

    message: .asciz "Ket qua tinh giai thua la: "

.text

main:

    jal WARP

print:

    add a1, s0, zero # a0 = result from N!

    li a7, 56

    la a0, message

    ecall

quit:

    li a7, 10 # terminate

    ecall

end\_main:

# ----------------------------------------------------------------------

# Procedure WARP: assign value and call FACT

# ----------------------------------------------------------------------

WARP:

    addi sp, sp, -4 # adjust stack pointer

    sw ra, 0(sp) # save return address

    li a0, 3 # load test input N

    jal FACT # call fact procedure

    lw ra, 0(sp) # restore return address

    addi sp, sp, 4 # return stack pointer

    jr ra

wrap\_end:

# ----------------------------------------------------------------------

# Procedure FACT: compute N!

# param[in] a0 integer N

# return s0 the largest value

# ----------------------------------------------------------------------

FACT:

    addi sp, sp, -8 # allocate space for ra, a0 in stack

    sw ra, 4(sp) # save ra register

    sw a0, 0(sp) # save a0 register

    li t0, 2

    bge a0, t0, recursive

    li s0, 1 # return the result N!=1

    j done

recursive:

    addi a0, a0, -1 # adjust input argument

    jal FACT # recursive call

    lw s1, 0(sp) # load a0

    mul s0, s0, s1

done:

    lw ra, 4(sp) # restore ra register

    lw a0, 0(sp) # restore a0 register

    addi sp, sp, 8 # restore stack pointer

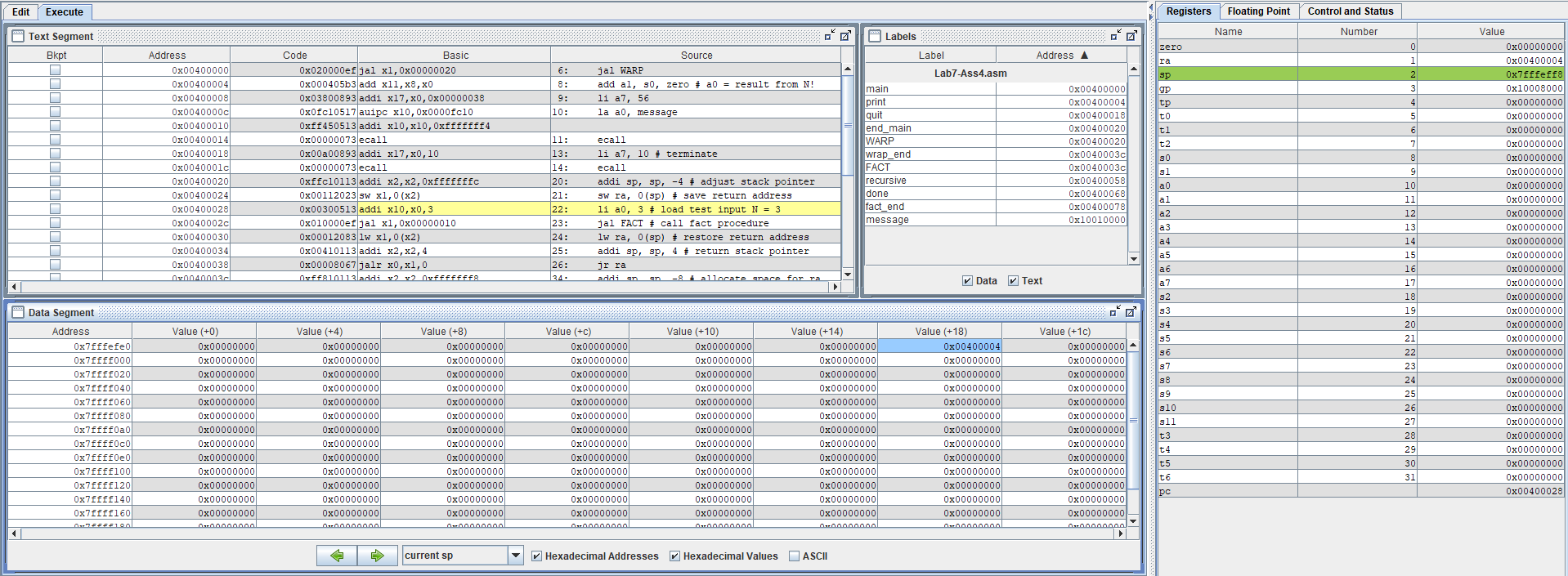
    jr ra # jump to caller

fact\_end:

**Explaination:** The program implement how to calculate the factorial of n by recursive algorithms and stack.

* The Warp procedure initials input for calculating factorial and the return address ra on the stack, after the Fact return, Warp restores the ra register and stack pointer before jumping back to main.
* The Fact procedure calculates the factorial recursively. It checks if N is greater than or equal to 2. If N is 1, it returns 1. If N is 3 (or any number >= 2), it decrements N (a0) by 1 and calls itself recursively. On each recursive return, Fact multiplies the current result (s0) by N, saved in s1. When the recursion completes, it restores a0, ra, and the stack pointer before returning to Warp.

**Result:**



A screenshot of a computer

Description automatically generated

A screenshot of a computer

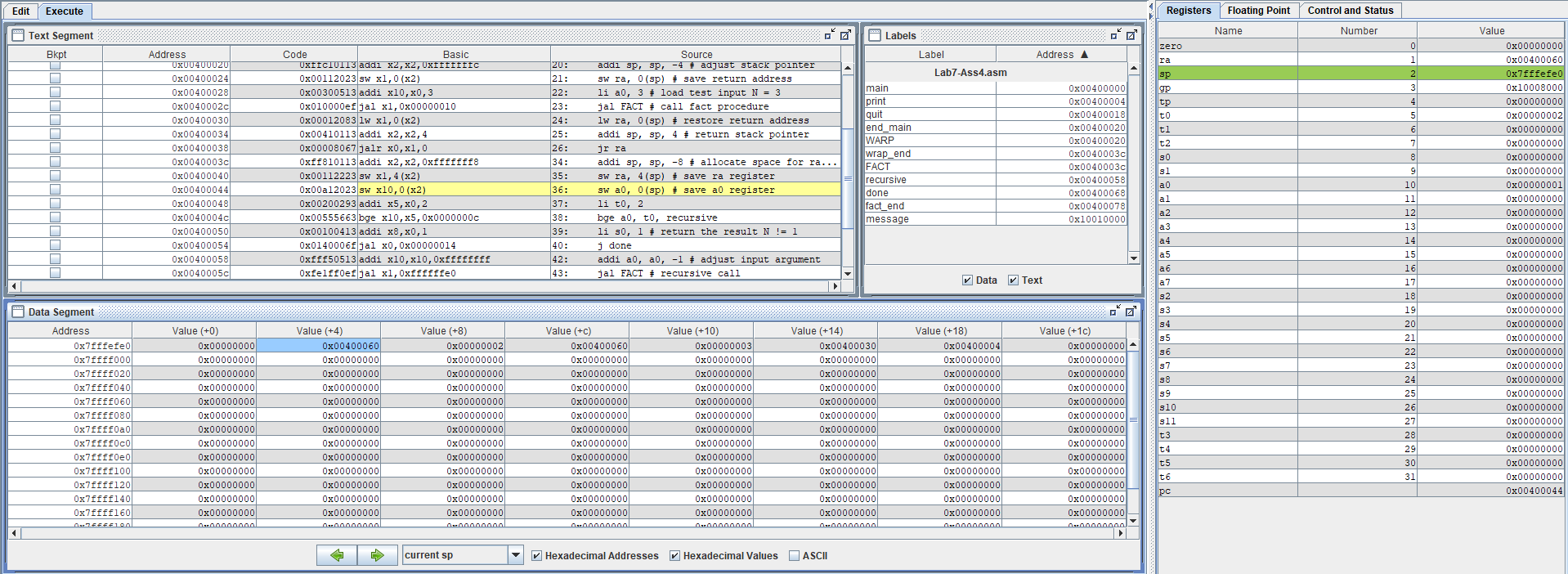
Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated



A screenshot of a computer

Description automatically generated

Output:

A screenshot of a computer error

Description automatically generated A screenshot of a computer

Description automatically generated

**Assignment 5:**

**Code:**

.data

    message\_largest: .asciz "Largest: "

    message\_smallest: .asciz "\nSmallest: "

    comma: .asciz ", "

.text

main:

    # Call Max\_Min subroutine

    jal Max\_Min

print:

    # Print largest value and its position

    la a0, message\_largest    # Load message for largest

    li a7, 4                  # System call for print string

    ecall

    addi a0, s0, 0            # Move largest value to a0 for print

    li a7, 1                  # System call for print integer

    ecall

    li a7, 4                  # Print comma and space

    la a0, comma

    ecall

    addi a0, s1, 0            # Move largest position to a0 for print

    li a7, 1                  # System call for print integer

    ecall

    # Print smallest value and its position

    la a0, message\_smallest   # Load message for smallest

    li a7, 4                  # System call for print string

    ecall

    addi a0, s2, 0            # Move smallest value to a0 for print

    li a7, 1                  # System call for print integer

    ecall

    li a7, 4                  # Print comma and space

    la a0, comma

    ecall

    addi a0, s3, 0            # Move smallest position to a0 for print

    li a7, 1                  # System call for print integer

    ecall

    li a7, 10                 # System call for program exit

    ecall

Max\_Min:

    # Initial example values in a0 to a7

    li a0, 2

    li a1, -1

    li a2, 4

    li a3, 9

    li a4, 5

    li a5, 3

    li a6, 21

    li a7, 7

    # Store initial values

    addi sp, sp, -16       # Adjust stack for storing registers

    sw ra, 12(sp)          # Store return address

    sw t0, 8(sp)           # Store temporary registers

    sw t1, 4(sp)

    sw t2, 0(sp)

    # Initialize largest and smallest values with a0 and positions with 0

    addi s0, a0, 0         # s0 = largest value

    addi s2, a0, 0         # s2 = smallest value

    li s1, 0               # s1 = position of largest value

    li s3, 0               # s3 = position of smallest value

    # Begin comparisons with registers a0 through a7

    li t0, 0               # Position counter for current register, start from 0

    # Manually check each register from a0 to a7

    addi t1, a0, 0         # Load a0

    jal compare\_and\_update

    addi t0, t0, 1         # Increment position

    addi t1, a1, 0         # Load a1

    jal compare\_and\_update

    addi t0, t0, 1         # Continue incrementing position

    addi t1, a2, 0         # Load a2

    jal compare\_and\_update

    addi t0, t0, 1

    addi t1, a3, 0         # Load a3

    jal compare\_and\_update

    addi t0, t0, 1

    addi t1, a4, 0         # Load a4

    jal compare\_and\_update

    addi t0, t0, 1

    addi t1, a5, 0         # Load a5

    jal compare\_and\_update

    addi t0, t0, 1

    addi t1, a6, 0         # Load a6

    jal compare\_and\_update

    addi t0, t0, 1

    addi t1, a7, 0         # Load a7

    jal compare\_and\_update

end\_loop:

    # Restore the stack and return

    lw ra, 12(sp)          # Restore return address

    lw t0, 8(sp)           # Restore temporary registers

    lw t1, 4(sp)

    lw t2, 0(sp)

    addi sp, sp, 16        # Restore stack pointer

    jr ra                  # Return to caller

compare\_and\_update:

    bgt t1, s0, update\_largest  # If current value > largest, update largest

    blt t1, s2, update\_smallest # If current value < smallest, update smallest

    jr ra                       # Return to caller

update\_largest:

    addi s0, t1, 0              # Update largest value

    addi s1, t0, 0              # Update largest position

    jr ra                       # Return to caller

update\_smallest:

    addi s2, t1, 0              # Update smallest value

    addi s3, t0, 0              # Update smallest position

    jr ra                       # Return to caller

**Explaination:**

* Procedure Max\_Min: Find the largest and smallest values and their positions from registers a0 to a7.

Returns:

* + s0 - largest value
  + s1 - position of largest value (0 to 7)
  + s2 - smallest value
  + s3 - position of smallest value (0 to 7)
* Procedure compare\_and\_update: Compares current value (t1) with current largest (s0) and smallest (s2), updates values and positions if necessary.

**Result:**

**A close-up of a computer screen

Description automatically generated**