**Lab 7. Procedure calls, stack and parameters**

**Họ và tên: Phạm Long Khánh**

**MSSV: 20225640**

**Assignment 1:**

.text

main: li $a1, -10 # load input parameter

jal abs # jump and link to abs procedure

nop

# print absolute value of a1

add $a0, $v0, $0

li $v0, 1

syscall

li $v0, 10

syscall # terminate

end\_main:

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

abs: addi $v0, $a1, 0 # set v0 = a1

slt $t0, $a1, $0 # check if a1 < 0

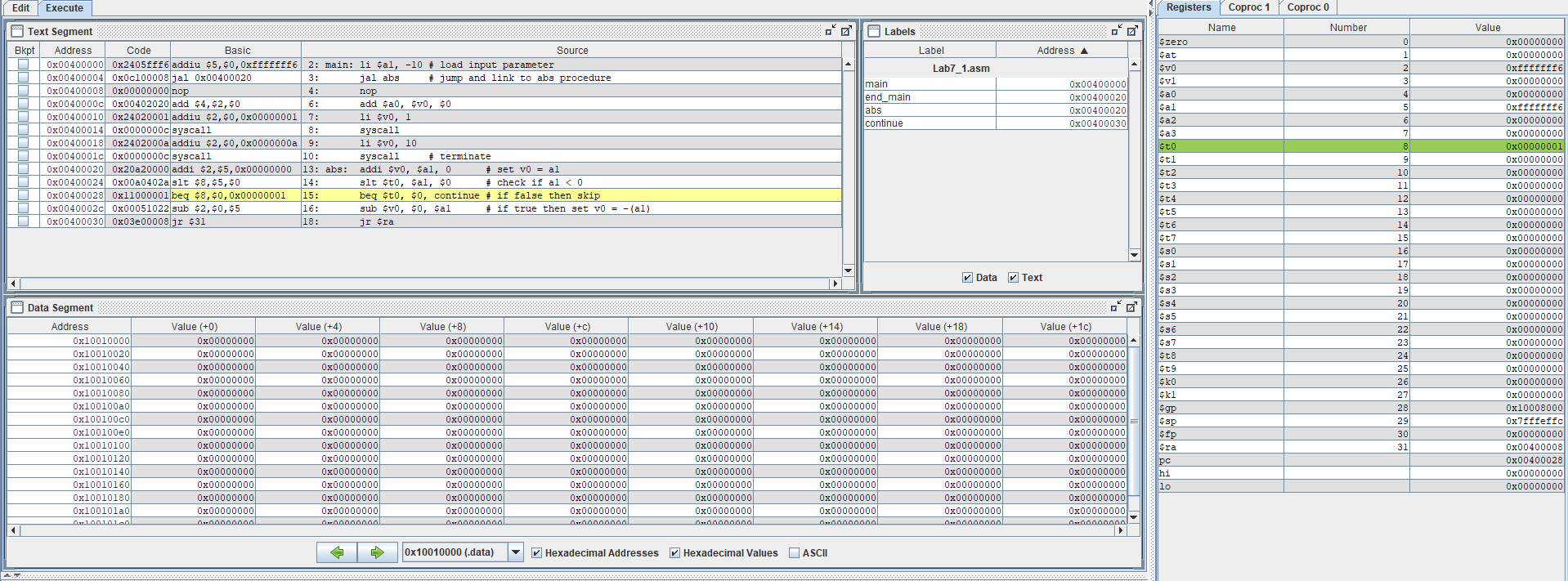
beq $t0, $0, continue # if false then skip

sub $v0, $0, $a1 # if true then set v0 = -(a1)

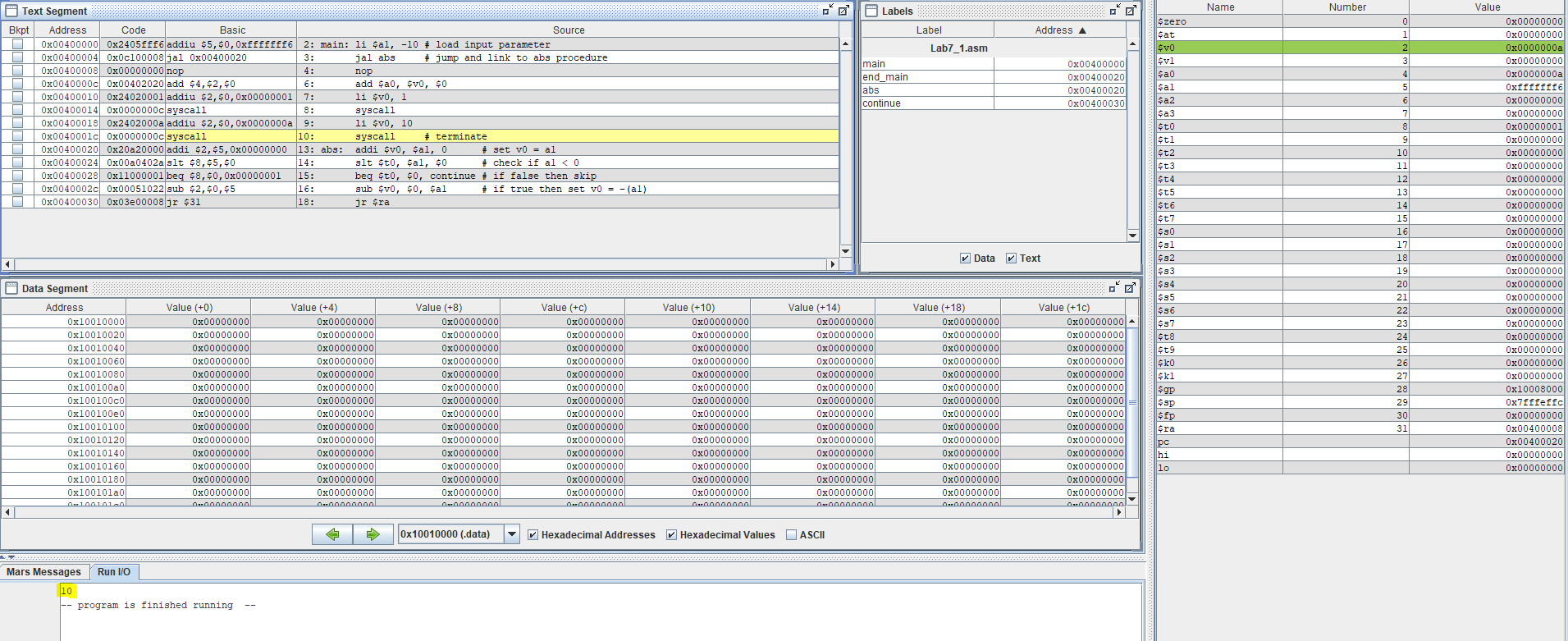
continue:

jr $ra

* Quá trình thực hiện chương trình:



* Kết quả của chương trình:



* Ta cho đầu vào là giá trị -10. Khi chương trình kết thúc, giá trị xuất hiện ỏe cửa sổ Run I/O là 10 – giá trị tuyệt đối của (-10) -> Đúng (có thể thử với các đầu vào khác).
* Chương trình chạy đúng và cho kết quả chính xác

**Assignment 2:**

.data

mes1: .asciiz "The largest element: "

.text

main: li $a0, 1 # load test input

li $a1, -4

li $a2, 3

jal max

nop

addi $t0, $v0, 0 # set t0 = max

print: li $v0, 56

la $a0, mes1

add $a1, $t0, $0

syscall

li $v0, 10

syscall # terminate

end\_main:

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

max: add $v0, $a0, $0 # set max = a0

slt $t0, $v0, $a1 # check if max < a1

beq $t0, $0, continue # if false then skip

add $v0, $a1, $0 # if true then set max = a1

continue:

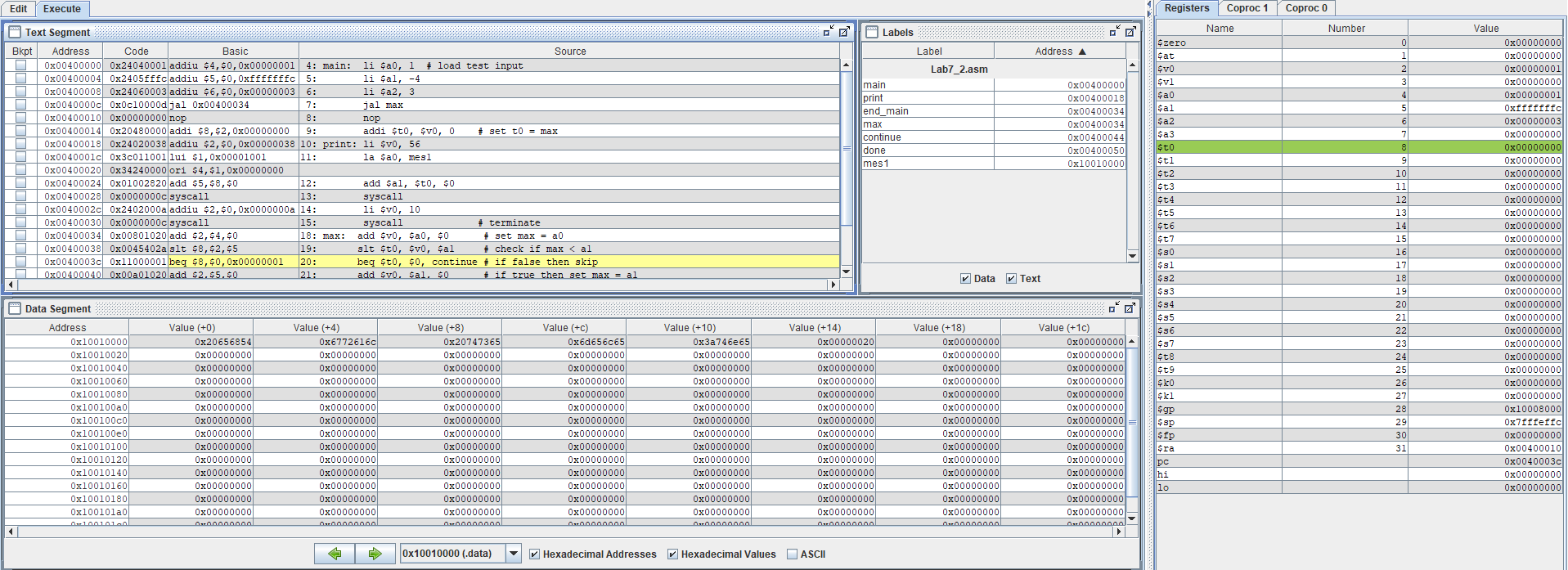
slt $t0, $v0, $a2 # check if max < a2

beq $t0, $0, done # if false then skip

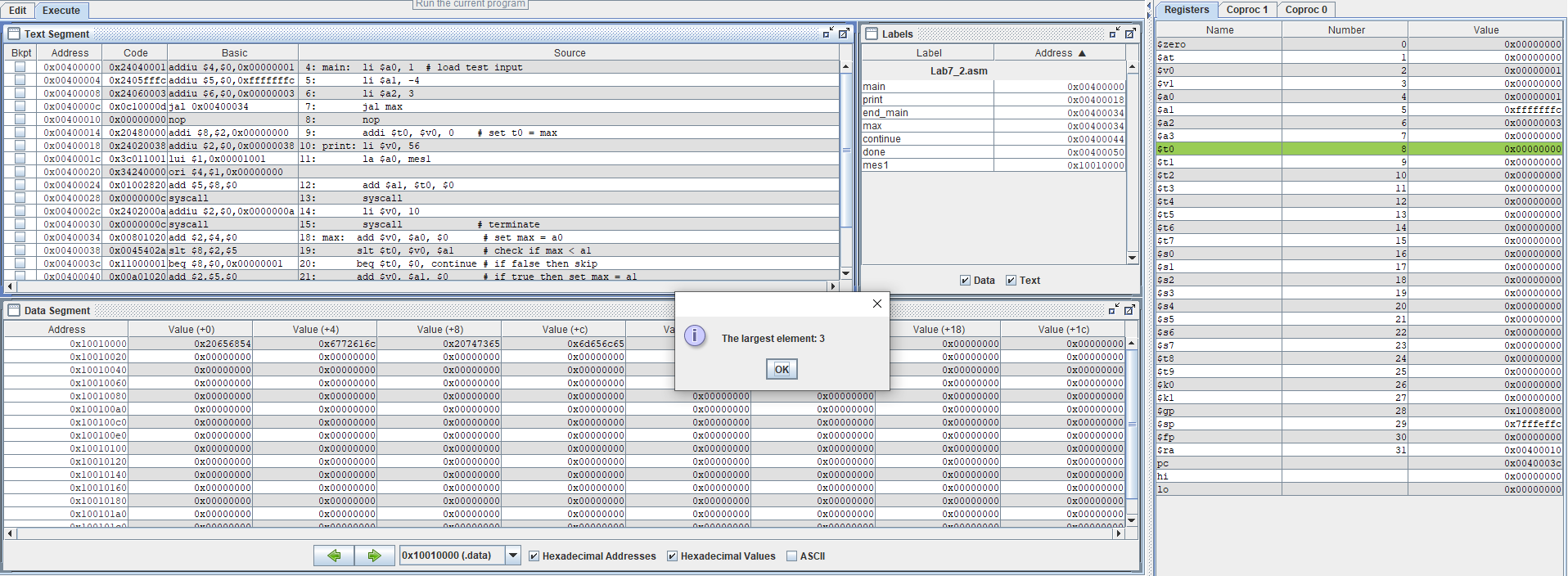
add $v0, $a2, $0 # if true then set max = a2

done: jr $ra # return to calling program

* Quá trình thực hiện chương trình:



* Kết quả của chương trình:



* Ta cho đầu vào là 3 số có giá trị lần lượt là 1, -4, 3. Khi kết thúc chương trình hộp thoại hiện ra giá trị lớn nhất là 3 -> đúng. (có thể thử với các đầu vào khác)
* Chương trình chạy đúng và chính xác

**Assignment 3:**

.text

input: addi $s0, $0, 2 # set s0 = 2

addi $s1, $0, 5 # set s1 = 5

push: addi $sp, $sp, -8 # adjust stack pointer

sw $s0, 4($sp) # push s0 in stack

sw $s1, 0($sp) # push s1 in stack

work: nop

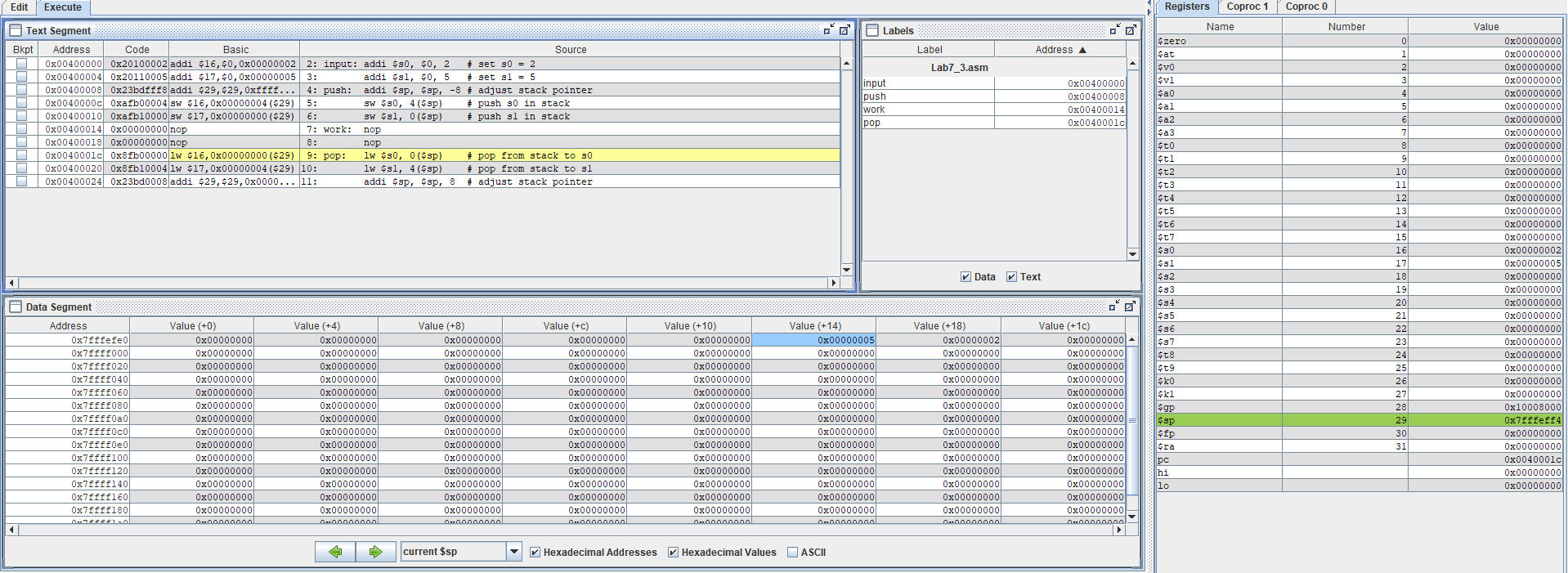
nop

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

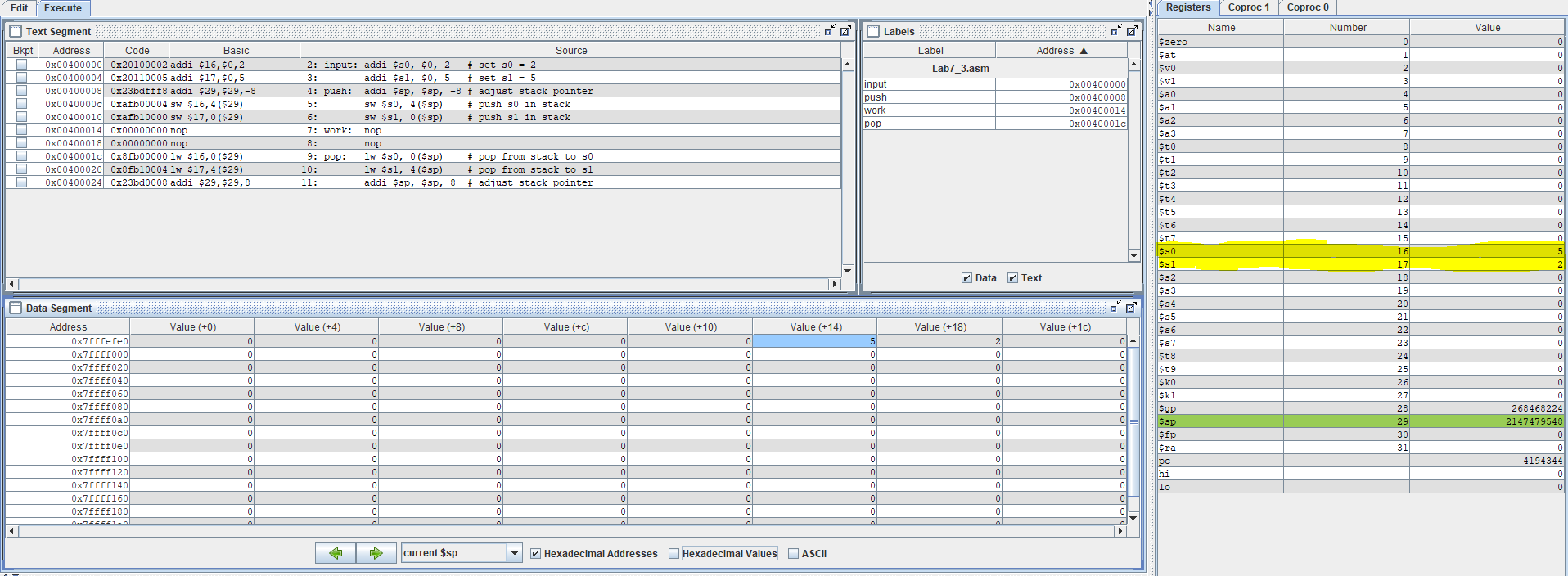
lw $s1, 4($sp) # pop from stack to s1

addi $sp, $sp, 8 # adjust stack pointer

* Quá trình thực hiện chương trình:



* Kết quả của chương trình:



* Lúc đầu ta nhập giá trị 2 vào thanh ghi $s0 và 5 vào thanh ghi $s1. Sau quá trình thực hiện push – pop trong chương trình ta có được giá trị lưu trong thah ghi $s0 = 5 và giá trị lưu tỏng thanh ghi $s1 = 2 -> Thành công hoán đổi giá trị của 2 thanh ghi (có thể thử với các đầu vào khác).
* Chương trình chạy đúng và chính xác

**Assignment 4:**

.data

mes1: .asciiz "Ket qua tinh giai thua la: "

.text

main:

li $a0, 10 # n = 10

jal factorial

add $t0, $v0, $0 # set t0 = v0 = n!

print: addi $v0, $0, 56

la $a0, mes1

add $a1, $t0, $0 # a0 = n!

syscall

end\_main: li $v0, 10

syscall # terminate

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

addi $t0, $0, 1 # t0 = 1

factorial:

sw $fp, -4($sp) # store frame pointer

add $fp, $sp, $0 # fp = sp

addi $sp, $sp, -12 # adjust the stack pointer (4 space)

sw $ra, -8($fp) # store return address

sw $a0, -12($fp) # store a0

blt $t0, $a0, L1 # if (n > 1) then branch to L1

addi $v0, $0, 1 # else v0 = 1

addi $sp, $fp, 0 # restore sp

lw $fp, -4($sp) # restore fp

jr $ra # return

L1: addi $a0, $a0, -1 # n = n - 1

jal factorial # call recursion

do: lw $a0, -12($fp) # restore n

lw $ra, -8($fp) # restore return address (address of next command)

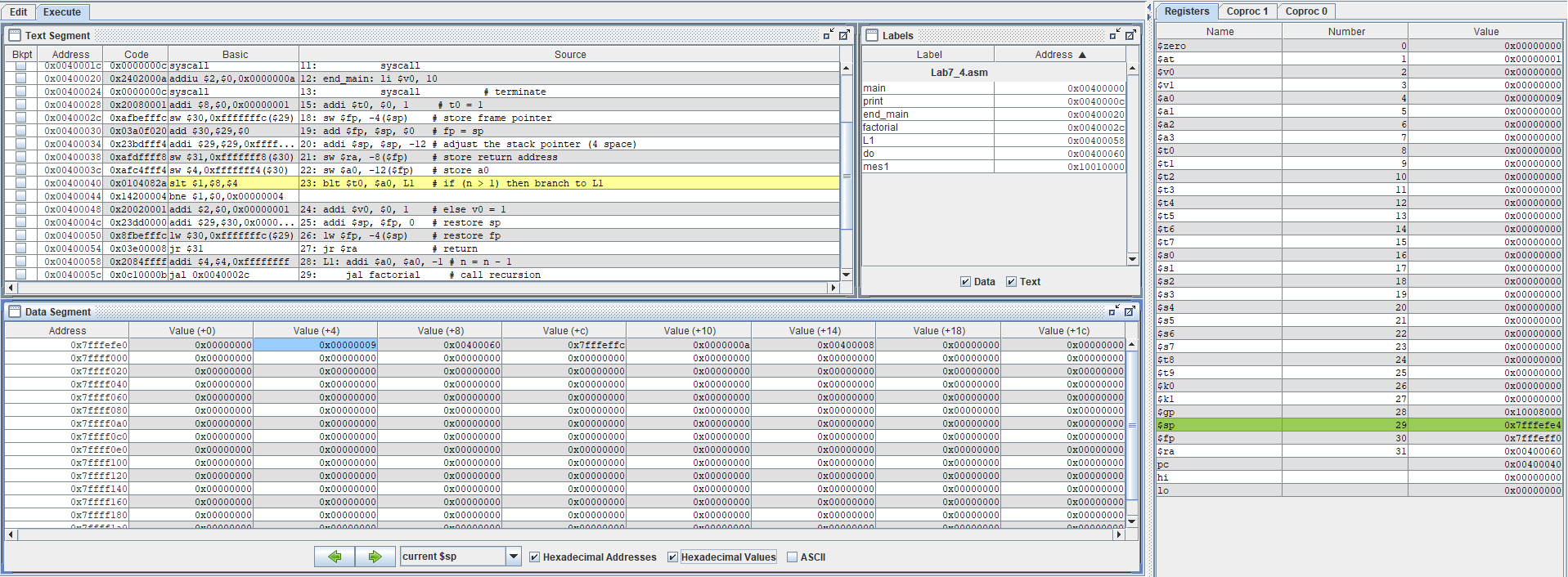
addi $sp, $fp, 0 # restore sp

lw $fp, -4($sp) # restore fp

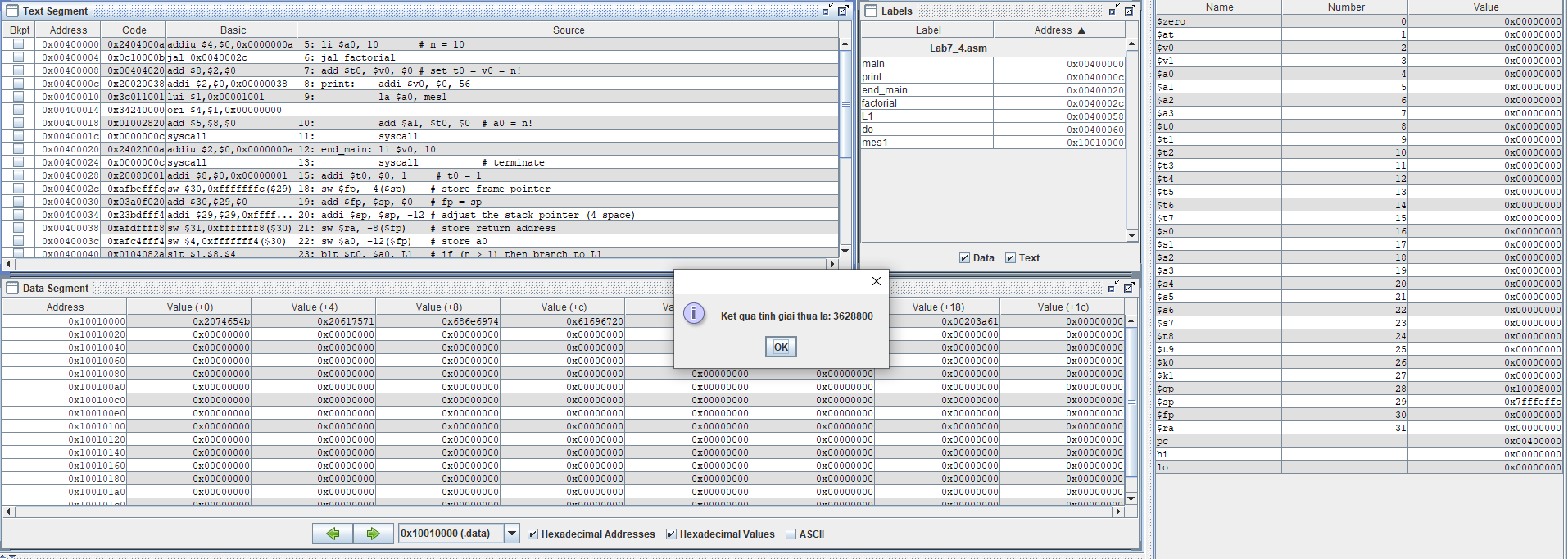
mul $v0, $v0, $a0 # v0 = v0 \* n

jr $ra

* Quá trình thực hiện chương trình:



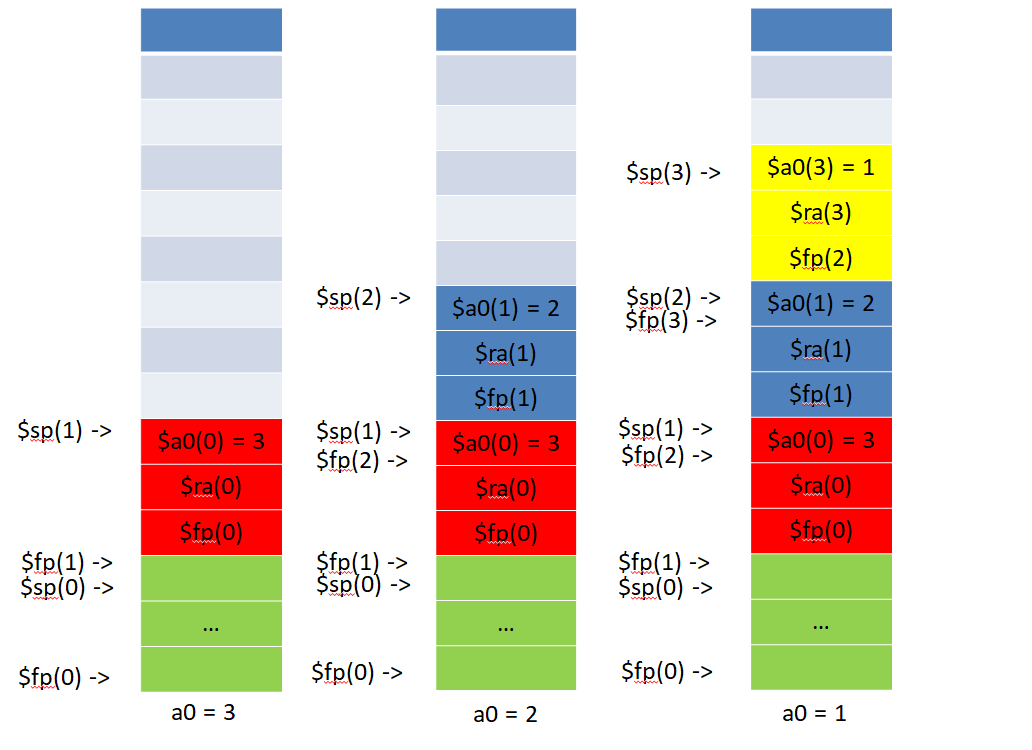
* Kết quả của chương trình:



Khi đầu vào là n = 10 thì kết quả có được là 3628800 = 10! -> đúng (Có thể thử với các đầu vào khác)

* Chương trình chạy đúng và cho kết quả chính xác

Hình vẽ thể hiện Stack trong quá trình chạy hàm đệ quy với đầu vào là n = 3 ( Thực hiện tính 3!).



**Assignment 5:**

.data

mes1: .asciiz "Largest: "

mes2: .asciiz " Smallest: "

.text

main:

addi $s0, $0, 2 # s0 = 2

addi $s1, $0, 1 # s1 = 1

addi $s2, $0, -3 # s2 = -3

addi $s3, $0, 4 # s3 = 4

addi $s4, $0, 0 # s4 - 0

addi $s5, $0, 7 # s5 = 7

addi $s6, $0, 9 # s6 = 9

addi $s7, $0, 4 # s7 = 4

jal proc\_max

addi $t0, $v0, 0 # t0 = largest

addi $t1, $v1, 0 # t1 = largest position

print\_max: # Print the largest element and the positon of this element in 8 registers

addi $v0, $0, 4

la $a0, mes1

syscall

addi $v0, $0, 1

addi $a0, $t0, 0

syscall

addi $v0, $0, 11

addi $a0, $0, 44

syscall

addi $v0, $0, 1

addi $a0, $t1, 0

syscall

jal proc\_min # Print the smallest element and the positon of this element in 8 registers

addi $t2, $v0, 0 # t2 = smallest

addi $t3, $v1, 0 # t3 = smallest position

print\_min:

addi $v0, $0, 4

la $a0, mes2

syscall

addi $v0, $0, 1

addi $a0, $t2, 0

syscall

addi $v0, $0, 11

addi $a0, $0, 44

syscall

addi $v0, $0, 1

addi $a0, $t3, 0

syscall

exit: addi $v0, $0, 10

syscall

end\_main:

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

proc\_max: # The procedure to find the largest element and the position of this element

sw $fp, -4($sp) # store frame pointer

addi $fp, $sp, 0 # fp = sp

addi $sp, $sp, -40 # allocate space in stack

sw $ra, 32($sp) # store return address

sw $s7, 28($sp) # sotore s7

sw $s6, 24($sp) # store s6

sw $s5, 20($sp) # store s5

sw $s4, 16($sp) # store s4

sw $s3, 12($sp) # store s3

sw $s2, 8($sp) # store s2

sw $s1, 4($sp) # store s1

sw $s0, 0($sp) # store s0

add $v0, $s0, $0 # set max = s0

addi $v1, $0, 0 # set max\_pos = 0

addi $t0, $0, 1 # i = 0

addi $t1, $0, 8 # t1 = n = 8 (8 registers)

LOOP1: slt $t2, $t0, $t1 # check if i < 8

beq $t2, $0, end\_LOOP1 # if not then end loop

sll $t2, $t0, 2 # t2 = 4 \* i

add $t2, $sp, $t2 # t2 = sp[i]

lw $t2, 0($t2) # t2 = si (i in [0, 7])

slt $t3, $v0, $t2 # check if si > max

beq $t3, $0, continue1 # if false then skip

add $v0, $t2, $0 # if true then set max = si

add $v1, $t0, $0 # and set max\_pos = i (which mean largest element is stored in $si)

continue1:

addi $t0, $t0, 1 # i = i + 1

j LOOP1

end\_LOOP1:

lw $s0, 0($sp) # restore s0

lw $s1, 4($sp) # restore s1

lw $s2, 8($sp) # restore s2

lw $s3, 12($sp) # restore s3

lw $s4, 16($sp) # restore s4

lw $s5, 20($sp) # restore s5

lw $s6, 24($sp) # restore s6

lw $s7, 28($sp) # restore 27

lw $ra, 32($sp) # restore return address

addi $sp, $fp, 0 # restore sp

lw $fp, -4($sp) # restore fp

jr $ra # return

proc\_min: # The procedure to find the smallest element and the position of this element

sw $fp, -4($sp) # store frame pointer

addi $fp, $sp, 0 # fp = sp

addi $sp, $sp, -40 # allocate space in stack

sw $ra, 32($sp) # store return address

sw $s7, 28($sp) # sotore s7

sw $s6, 24($sp) # store s6

sw $s5, 20($sp) # store s5

sw $s4, 16($sp) # store s4

sw $s3, 12($sp) # store s3

sw $s2, 8($sp) # store s2

sw $s1, 4($sp) # store s1

sw $s0, 0($sp) # store s0

add $v0, $s0, $0 # set min = s0

addi $v1, $0, 0 # set min\_pos = 0

addi $t0, $0, 1 # i = 0

addi $t1, $0, 8 # t1 = n = 8 (8 registers)

LOOP2: slt $t2, $t0, $t1 # check if i < 8

beq $t2, $0, end\_LOOP2 # if not then end loop

sll $t2, $t0, 2 # t2 = 4 \* i

add $t2, $sp, $t2 # t2 = sp[i]

lw $t2, 0($t2) # t2 = si (i in [0, 7])

slt $t3, $t2, $v0 # check if si < min

beq $t3, $0, continue2 # if false then skip

add $v0, $t2, $0 # if true then set min = si

add $v1, $t0, $0 # and set min\_pos = i (which mean smallest element is stored in $si)

continue2:

addi $t0, $t0, 1 # i = i + 1

j LOOP2

end\_LOOP2:

lw $s0, 0($sp) # restore s0

lw $s1, 4($sp) # restore s1

lw $s2, 8($sp) # restore s2

lw $s3, 12($sp) # restore s3

lw $s4, 16($sp) # restore s4

lw $s5, 20($sp) # restore s5

lw $s6, 24($sp) # restore s6

lw $s7, 28($sp) # restore 27

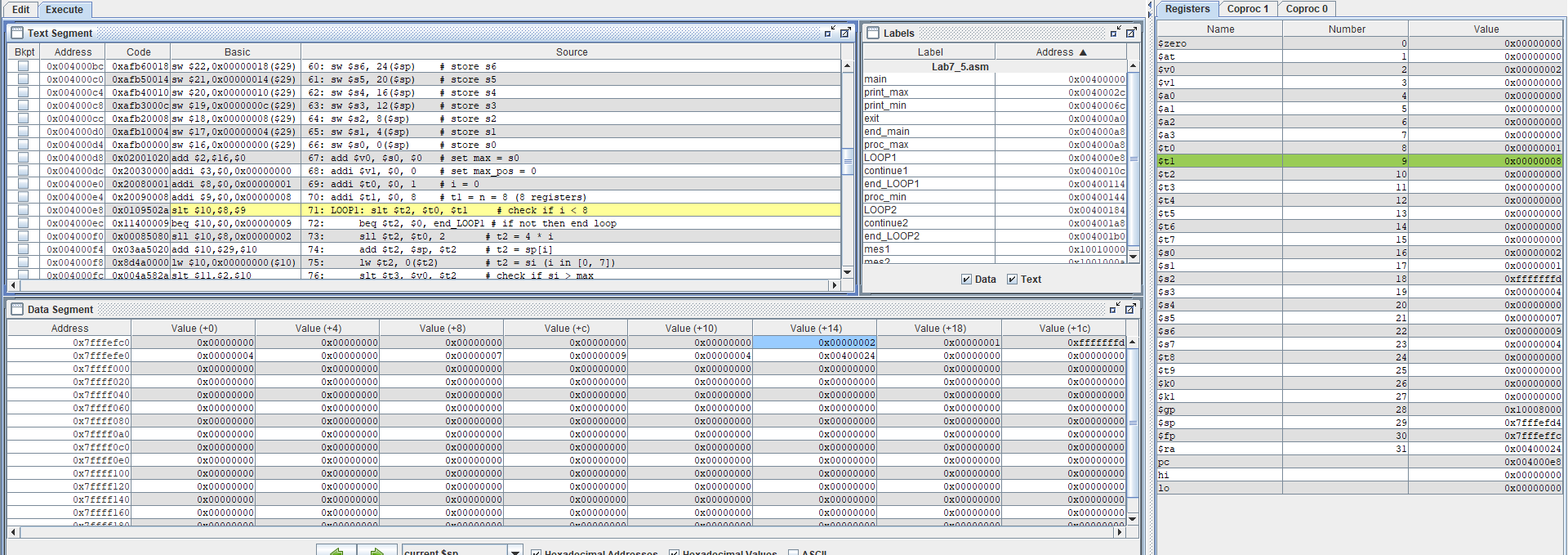
lw $ra, 32($sp) # restore return address

addi $sp, $fp, 0 # restore sp

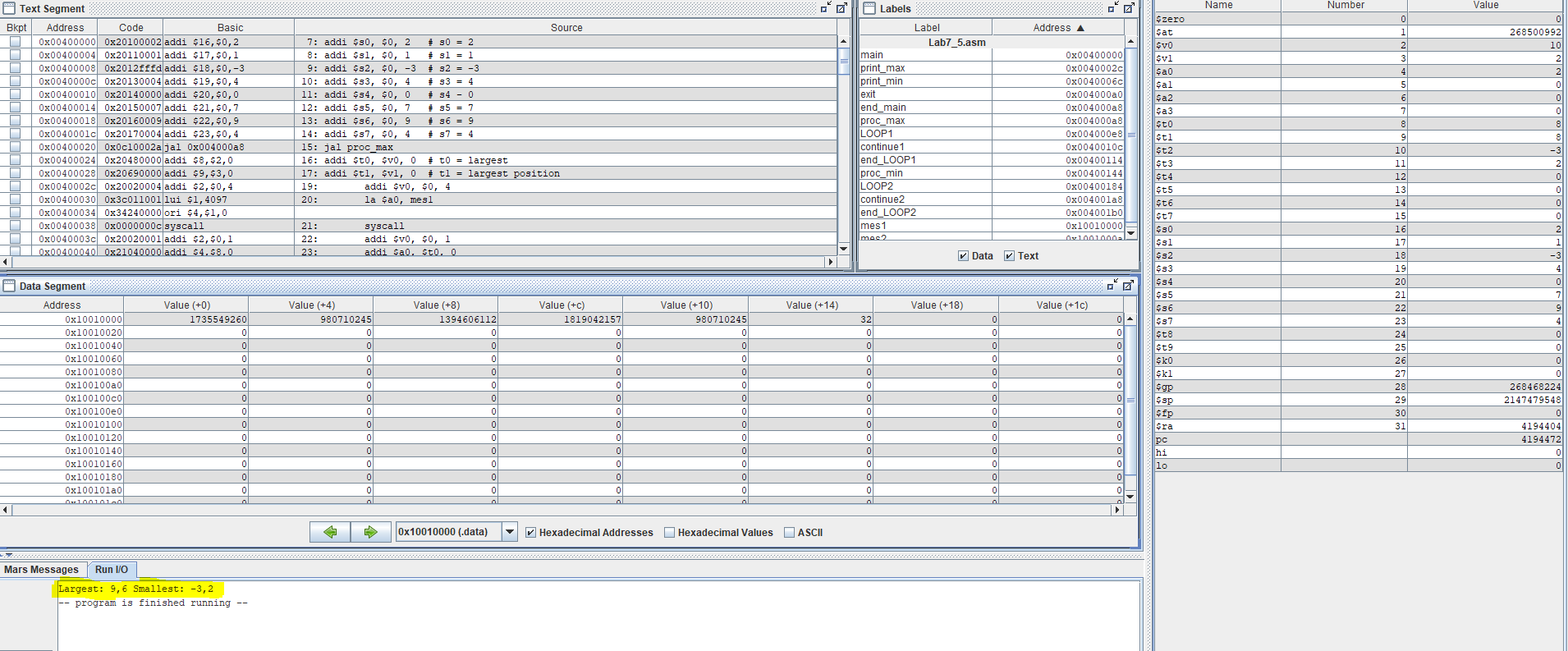
lw $fp, -4($sp) # restore fp

jr $ra # return

* Ta chọn đầu vào:
* Giá trị lưu trong thanh ghi $s0 = 2
* Giá trị lưu trong thanh ghi $s1 = 1
* Giá trị lưu trong thanh ghi $s2 = -3
* Giá trị lưu trong thanh ghi $s3 = 4
* Giá trị lưu trong thanh ghi $s4 = 0
* Giá trị lưu trong thanh ghi $s5 = 7
* Giá trị lưu trong thanh ghi $s6 = 9
* Giá trị lưu trong thanh ghi $s7 = 4
* Quá trình thực hiện chương trình:



* Kết quả của chương trình:



* Kết quả được đưa ra ở cửa sổ Run I/O:
* Largest: 9, 6 – Giá trị lớn nhất là 9 nằm ở thanh ghi $s6.
* Smallest: -3, 2 – Giá trị nhỏ nhất là -3 nằm ở thanh ghi $s2.

(Có thể thử với các đầu vào khác)

* Chương trình chạy đúng và chính xác