Nguyễn Đình Dương – 20225966

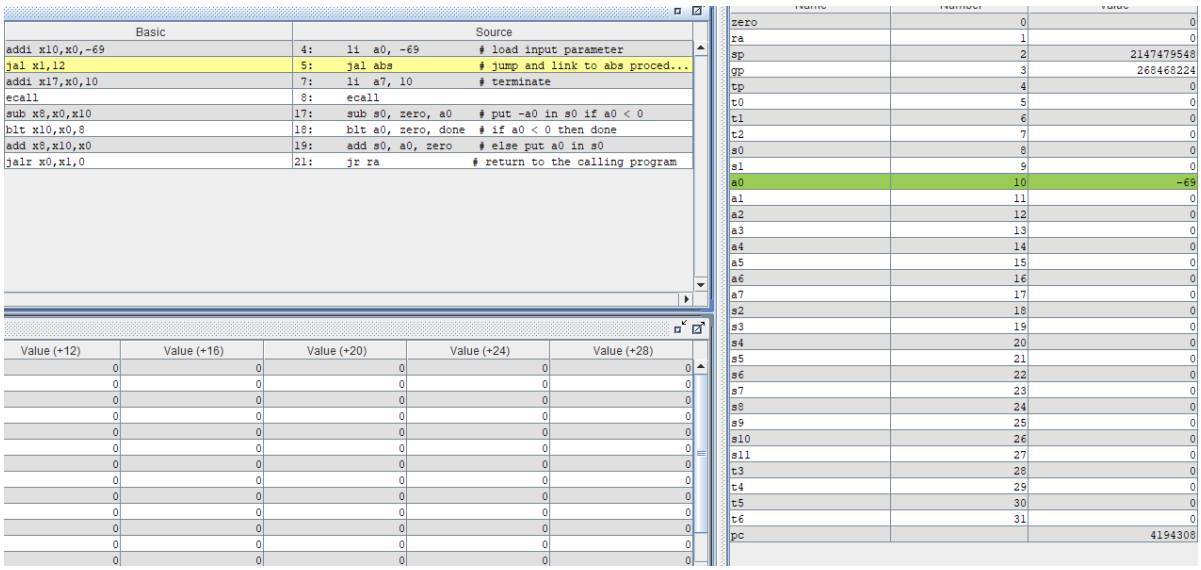
Assignment 1

Code:

```
# Laboratory Exercise 7, Home Assignment 1
.text
main:
    li  a0, -69         # load input parameter
    jal abs             # jump and link to abs procedure

    li  a7, 10          # terminate
    ecall
end_main:

# ---------------------------------------------------------------
# Procedure abs: Calculates the absolute value of a number
# param[in]    a0    the integer for which absolute value is needed
# return       s0    absolute value
# ---------------------------------------------------------------
abs:
    sub s0, zero, a0    # put -a0 in s0 if a0 < 0
    blt a0, zero, done  # if a0 < 0 then done
    add s0, a0, zero    # else put a0 in s0
done:
    jr ra               # return to the calling program
```

Output:

- Trước khi vào lệnh jal abs:



Basic		Source
addi x10,x0,-69	4:	li a0, -69 # load input parameter
jal x1,12	5:	jal abs # jump and link to abs proced...
addi x17,x0,10	7:	li a7, 10 # terminate
ecall	8:	ecall
sub x8,x0,x10	17:	sub s0, zero, a0 # put -a0 in s0 if a0 < 0
blt x10,x0,8	18:	blt a0, zero, done # if a0 < 0 then done
add x8,x10,x0	19:	add s0, a0, zero # else put a0 in s0
jalr x0,x1,0	21:	jr ra # return to the calling program

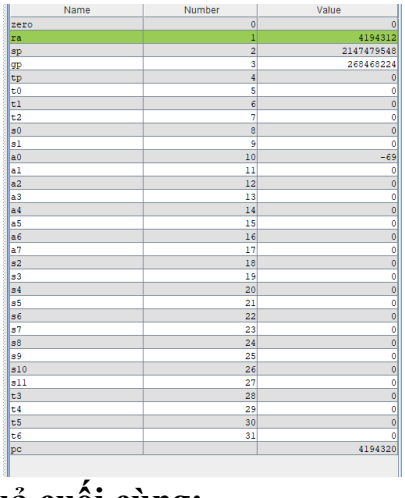
Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

zero	0	0
ra	1	0
sp	2	2147479548
gp	3	268468224
tp	4	0
t0	5	0
t1	6	0
t2	7	0
s0	8	0
s1	9	0
a0	10	-69
a1	11	0
a2	12	0
a3	13	0
a4	14	0
a5	15	0
a6	16	0
a7	17	0
s2	18	0
s3	19	0
s4	20	0
s5	21	0
s6	22	0
s7	23	0
s8	24	0
s9	25	0
s10	26	0
s11	27	0
t3	28	0
t4	29	0
t5	30	0
t6	31	0
pc		4194308

Ta thấy:

- **ra = 0**: Đây là giá trị ban đầu của thanh ghi **ra** (return address). Điều này có nghĩa là chưa có địa chỉ nào được lưu để quay về từ một thủ tục trước đó.
- **a0 = -69**: Thanh ghi **a0** được dùng để chứa tham số đầu vào cho hàm **abs**. Giá trị này là số nguyên **-69**, đây sẽ là tham số đầu vào cho hàm **abs**.
- **pc = 4194308**: **pc** là thanh ghi chứa địa chỉ của lệnh hiện tại, trong trường hợp này, nó chứa địa chỉ của lệnh **jal abs**. Địa chỉ này là **4194308** (hex: 0x00400004).

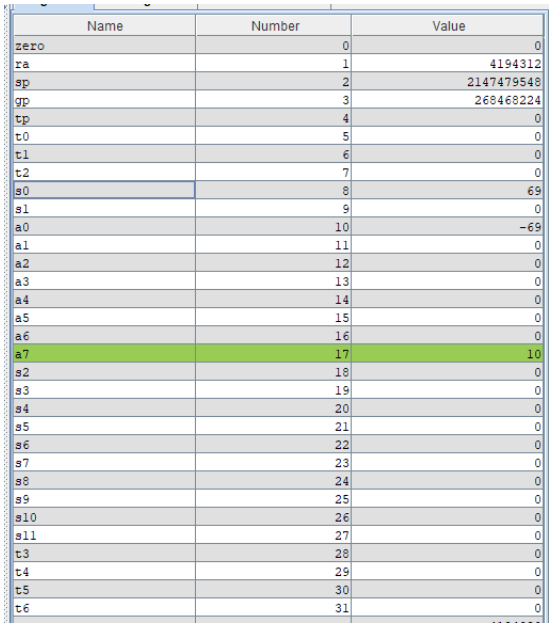
- Sau khi vào lệnh jal abs:



Name	Number	Value
zero	0	0
ra	1	4194312
sp	2	2147479548
gp	3	268468224
tp	4	0
t0	5	0
t1	6	0
t2	7	0
s0	8	0
s1	9	0
a0	10	-69
a1	11	0
a2	12	0
a3	13	0
a4	14	0
a5	15	0
a6	16	0
a7	17	0
s2	18	0
s3	19	0
s4	20	0
s5	21	0
s6	22	0
s7	23	0
s8	24	0
s9	25	0
s10	26	0
s11	27	0
t3	28	0
t4	29	0
t5	30	0
t6	31	0
pc		4194320

- **ra = 4194312**: Lệnh **jal abs** sẽ lưu địa chỉ của lệnh tiếp theo (pc + 4) vào thanh ghi **ra**. Trong trường hợp này, lệnh **jal abs** nằm tại địa chỉ **4194308**, nên **ra** sẽ được cập nhật với giá trị **4194312** (đây là địa chỉ của lệnh tiếp theo sau lệnh **jal abs**, tức là **li a7, 10**).
- **pc = 4194320**: Sau khi nhảy vào hàm **abs**, thanh ghi **pc** sẽ được cập nhật với địa chỉ của hàm **abs**.

Kết quả cuối cùng:



Name	Number	Value
zero	0	0
ra	1	4194312
sp	2	2147479548
gp	3	268468224
tp	4	0
t0	5	0
t1	6	0
t2	7	0
s0	8	69
s1	9	0
a0	10	-69
a1	11	0
a2	12	0
a3	13	0
a4	14	0
a5	15	0
a6	16	0
a7	17	10
s2	18	0
s3	19	0
s4	20	0
s5	21	0
s6	22	0
s7	23	0
s8	24	0
s9	25	0
s10	26	0
s11	27	0
t3	28	0
t4	29	0
t5	30	0
t6	31	0

- ra (return address) = 4194312: Đây là địa chỉ mà chương trình sẽ quay trở lại sau khi thực hiện xong hàm con abs. Nó tương ứng với lệnh tiếp theo sau khi gọi jal abs.
- a0 = -69:
- s0 = 69: Đây là kết quả tính toán của hàm abs.
- a7 = 10: Giá trị 10 trong thanh ghi a7 là mã hệ thống để thực hiện lệnh kết thúc chương trình (terminate) bằng cách gọi hệ thống qua ecall.

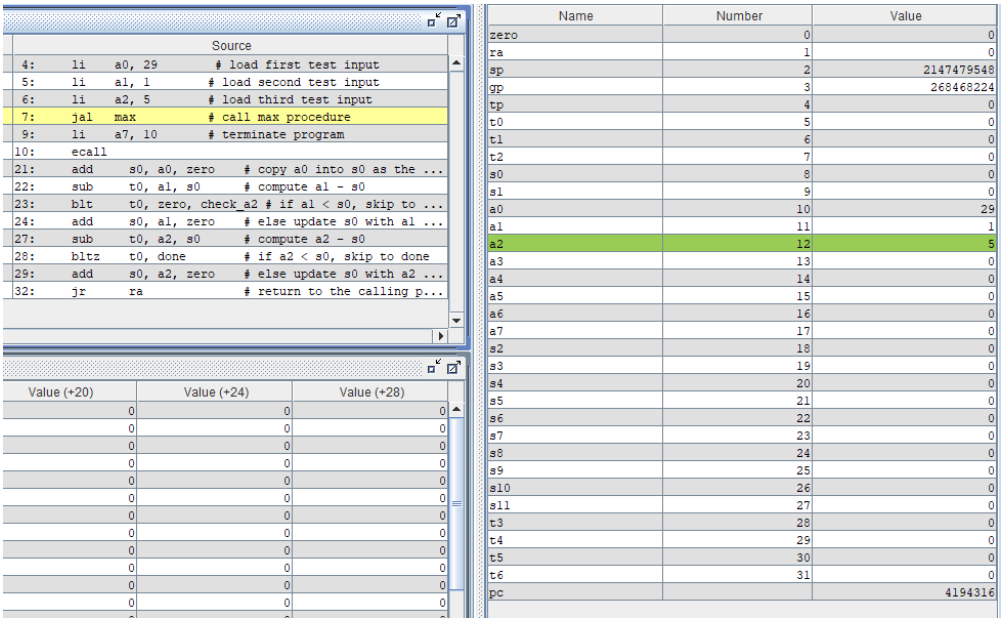
Như vậy, chương trình đã thực thi thành công việc tính giá trị tuyệt đối của a0 (là -69) và trả về kết quả 69 trong thanh ghi s0. Sau đó, chương trình kết thúc với lệnh ecall.

Assignment 2

```
# Laboratory Exercise 7, Home Assignment 2
.text
main:
    li    a0, 29       # load first test input
    li    a1, 1        # load second test input
    li    a2, 5        # load third test input
    jal   max          # call max procedure

    li    a7, 10       # terminate program
    ecall
end_main:

# ---------------------------------------------------------------
# Procedure max: Finds the largest of three integers
# param[in]  a0  integer 1
# param[in]  a1  integer 2
# param[in]  a2  integer 3
# return     s0  the largest value
# ---------------------------------------------------------------
max:
    add     s0, a0, zero    # copy a0 into s0 as the largest value so far
    sub     t0, a1, s0      # compute a1 - s0
    blt     t0, zero, check_a2 # if a1 < s0, skip to check_a2
    add     s0, a1, zero    # else update s0 with a1 as the largest so far

check_a2:
    sub     t0, a2, s0      # compute a2 - s0
    bltz    t0, done        # if a2 < s0, skip to done
    add     s0, a2, zero    # else update s0 with a2 as the largest overall

done:
    jr      ra              # return to the calling program
```

Output:

- Trước khi vào câu lệnh jal max:



Source

4:	li	a0, 29	# load first test input
5:	li	a1, 1	# load second test input
6:	li	a2, 5	# load third test input
7:	jal	max	# call max procedure
9:	li	a7, 10	# terminate program
10:	ecall		
21:	add	s0, a0, zero	# copy a0 into s0 as the ...
22:	sub	t0, a1, s0	# compute a1 - s0
23:	blt	t0, zero, check_a2	# if a1 < s0, skip to ...
24:	add	s0, a1, zero	# else update s0 with a1 ...
27:	sub	t0, a2, s0	# compute a2 - s0
28:	bltz	t0, done	# if a2 < s0, skip to done
29:	add	s0, a2, zero	# else update s0 with a2 ...
32:	jr	ra	# return to the calling p...

Value (+20)	Value (+24)	Value (+28)
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0

Name	Number	Value
zero	0	0
ra	1	0
sp	2	2147479548
gp	3	268468224
tp	4	0
t0	5	0
t1	6	0
t2	7	0
s0	8	0
s1	9	0
a0	10	29
a1	11	1
a2	12	5
a3	13	0
a4	14	0
a5	15	0
a6	16	0
a7	17	0
s2	18	0
s3	19	0
s4	20	0
s5	21	0
s6	22	0
s7	23	0
s8	24	0
s9	25	0
s10	26	0
s11	27	0
t3	28	0
t4	29	0
t5	30	0
t6	31	0
pc		4194316

- **a0 = 29**: Giá trị đầu tiên, được nạp vào để so sánh (thanh ghi đầu vào 1).
- **a1 = 1**: Giá trị thứ hai, được nạp vào để so sánh (thanh ghi đầu vào 2).
- **a2 = 5**: Giá trị thứ ba, được nạp vào để so sánh (thanh ghi đầu vào 3).
- **ra = 0**: Thanh ghi này chưa được cập nhật vì chưa có lệnh nhảy nào xảy ra.
- **s0 = 0**: Giá trị ban đầu của thanh ghi **s0** (dùng để lưu trữ kết quả là giá trị lớn nhất).

- Sau khi vào câu lệnh jal max:

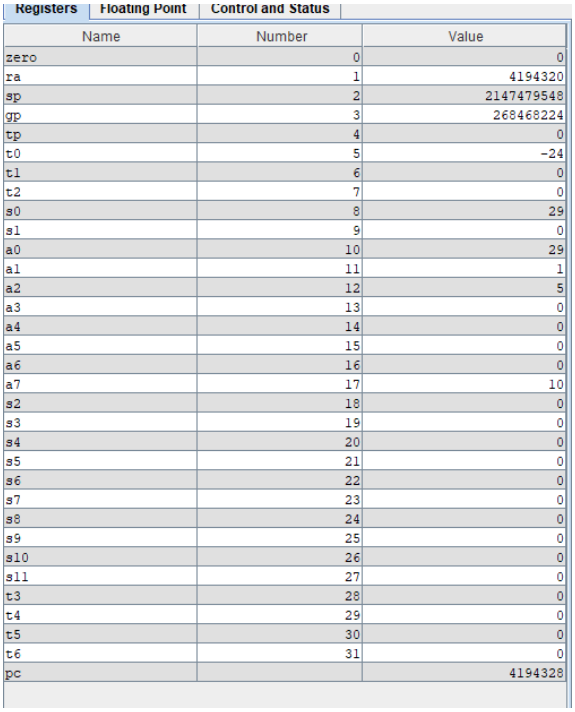
Name	Number	Value
zero	0	0
ra	1	4194320
sp	2	2147479548
gp	3	268468224
tp	4	0
t0	5	0
t1	6	0
t2	7	0
s0	8	0
s1	9	0
a0	10	29
a1	11	1
a2	12	5
a3	13	0
a4	14	0
a5	15	0
a6	16	0
a7	17	0
s2	18	0
s3	19	0
s4	20	0
s5	21	0
s6	22	0
s7	23	0
s8	24	0
s9	25	0
s10	26	0
s11	27	0
t3	28	0
t4	29	0
t5	30	0
t6	31	0
pc		4194328

- **ra = 4194316**: Sau khi thực hiện lệnh **jal max**, địa chỉ trả về của chương trình chính (địa chỉ của lệnh sau **jal max**) được lưu vào thanh ghi **ra**. Địa chỉ này là **4194316** (hex: 0x00400004), dùng để quay lại chương trình sau khi thực hiện xong hàm **max**.
- **pc** đã được nhảy đến địa chỉ của hàm **max**, nơi bắt đầu thực hiện các lệnh trong hàm.

- Sau khi chạy xong chương trình:

- **a0 = 29**: Đây là tham số đầu vào thứ nhất, giá trị ban đầu là **29**.
- **a1 = 1**: Đây là tham số đầu vào thứ hai, giá trị ban đầu là **1**.
- **a2 = 5**: Đây là tham số đầu vào thứ ba, giá trị ban đầu là **5**.

- **s0 = 29**: Giá trị này là kết quả cuối cùng của chương trình, đại diện cho số lớn nhất trong ba số nguyên đầu vào. Do số lớn nhất giữa **29**, **1**, và **5** là **29**, thanh ghi **s0** lưu giữ giá trị này.
- **t0 = -24**: Đây là kết quả của phép tính trung gian được thực hiện trong quá trình so sánh giữa các giá trị, nhưng không ảnh hưởng đến kết quả cuối cùng.
- **ra = 4194320**: Sau khi thực thi hàm **max**, địa chỉ này sẽ được sử dụng để quay trở lại chương trình chính sau khi kết thúc chương trình con.
- **pc = 4194328**: Đây là địa chỉ của lệnh tiếp theo sẽ được thực thi. Vì chương trình đã hoàn thành và đã đạt đến lệnh cuối cùng, **pc** đã cập nhật đến vị trí tiếp theo.



Registers | Floating Point | Control and Status

Name	Number	Value
zero	0	0
ra	1	4194320
sp	2	2147479548
gp	3	268468224
tp	4	0
t0	5	-24
t1	6	0
t2	7	0
s0	8	29
s1	9	0
a0	10	29
a1	11	1
a2	12	5
a3	13	0
a4	14	0
a5	15	0
a6	16	0
a7	17	10
s2	18	0
s3	19	0
s4	20	0
s5	21	0
s6	22	0
s7	23	0
s8	24	0
s9	25	0
s10	26	0
s11	27	0
t3	28	0
t4	29	0
t5	30	0
t6	31	0
pc		4194328

Assignment 3

Code:

```
.text
main:
    li      s0, 69          # Gán giá trị 10 cho thanh ghi s0
    li      s1, 96          # Gán giá trị 20 cho thanh ghi s1
    jal     swap            # Gọi thủ tục swap để hoán đổi giá trị s0 và s1
    # Kết thúc chương trình
    li      a7, 10          # Gọi dịch vụ hệ thống để kết thúc
    ecall
# ------------------------------------------------------------------
# Procedure swap: hoán đổi giá trị của hai thanh ghi s0 và s1
# ------------------------------------------------------------------
swap:
    addi    sp, sp, -8      # Điều chỉnh con trỏ ngăn xếp (giảm 8 byte)
    sw      s0, 4(sp)       # Lưu giá trị s0 vào ngăn xếp
    sw      s1, 0(sp)       # Lưu giá trị s1 vào ngăn xếp
    nop                     # Lệnh rỗng
    # Phục hồi giá trị từ ngăn xếp
    lw      s0, 0(sp)       # Lấy giá trị s1 từ ngăn xếp và gán cho s0
    lw      s1, 4(sp)       # Lấy giá trị s0 từ ngăn xếp và gán cho s1
    addi    sp, sp, 8       # Khôi phục con trỏ ngăn xếp (tăng 8 byte)

    jr      ra              # Quay lại chương trình chính
```

Output:

- Sau khi gán xong giá trị:
  S0 = 69, s1= 96

Name	Number	Value
zero	0	0
ra	1	0
sp	2	2147479548
gp	3	268468224
tp	4	0
t0	5	0
t1	6	0
t2	7	0
s0	8	69
s1	9	96
a0	10	0
a1	11	0
a2	12	0
a3	13	0
a4	14	0
a5	15	0
a6	16	0
a7	17	0
s2	18	0
s3	19	0
s4	20	0
s5	21	0
s6	22	0
s7	23	0
s8	24	0
s9	25	0
s10	26	0
s11	27	0
t3	28	0
t4	29	0
t5	30	0
t6	31	0
pc		4194312

- Ngay sau khi vào swap::

Name	Number	Value
zero	0	0
ra	1	4194316
sp	2	2147479548
gp	3	268468224
tp	4	0

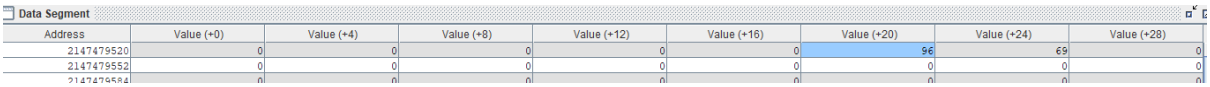
Giá trị ra đã thay đổi

- Ngay sau câu lệnh ***addi sp, sp, -8:***

Name	Number	Value
zero	0	0
ra	1	4194316
sp	2	2147479540
gp	3	268468224
tp	4	0

Giá trị sp đã giảm đi 8

- Sau 2 câu lệnh sw :



Giá trị 96 và 69 đã được lưu vào stack

- Sau 2 câu lệnh lw: giá trị s0 và s1 đã được đổi chỗ cho nhau:

t2	7	0
s0	8	96
s1	9	69
a0	10	0

Kết luận: Chức năng của stack trong chương trình:

1. **Lưu trữ địa chỉ quay lại:**
    - Khi chương trình gặp lệnh jal max, địa chỉ của lệnh tiếp theo (tức là địa chỉ sau lệnh jal) được lưu vào thanh ghi ra (return address). Lệnh jal tự động lưu địa chỉ này để sau khi chương trình con thực hiện xong, nó có thể quay lại vị trí ban đầu trong chương trình chính.
2. **Phục hồi địa chỉ quay lại:**
    - Sau khi thực hiện xong chương trình con max, lệnh jr ra được sử dụng để quay lại địa chỉ được lưu trong thanh ghi ra, đảm bảo rằng chương trình chính tiếp tục từ lệnh kế tiếp sau jal max.

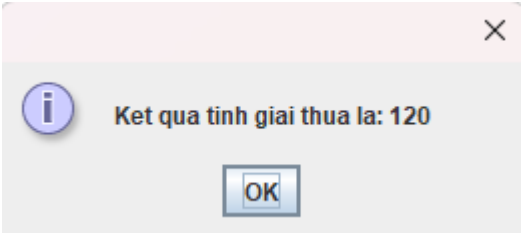
Assignment 4

Code thay đổi a0, a0=5 để tính giai thừa của 5:

```
.data
message: .asciz  "Ket qua tinh giai thua la: "  # Chuỗi thông báo kết quả

.text
main:
   jal    WARP              # Gọi thủ tục WARP để khởi tạo và tính giai thừa

print:
   add    a1, s0, zero      # Chuyển kết quả giai thừa vào a1 để in ra
   li     a7, 56            # Số dịch vụ hệ thống để in chuỗi
   la     a0, message       # Đưa địa chỉ chuỗi thông báo vào a0
   ecall                    # Gọi hệ thống để in thông báo

quit:
   li     a7, 10            # Số dịch vụ hệ thống để kết thúc chương trình
   ecall                    # Gọi hệ thống để kết thúc
end_main:

# ---------------------------------------------------------------------
# Procedure WARP: gán giá trị n và gọi thủ tục FACT
# ---------------------------------------------------------------------
WARP:
   addi   sp, sp, -4        # Điều chỉnh con trỏ ngăn xếp (stack pointer)
   sw     ra, 0(sp)         # Lưu địa chỉ trở về (return address) lên ngăn xếp

   li     a0, 5             # Gán giá trị n = 5 vào thanh ghi a0
   jal    FACT              # Gọi thủ tục FACT để tính giai thừa

   lw     ra, 0(sp)         # Phục hồi địa chỉ trở về từ ngăn xếp
   addi   sp, sp, 4         # Khôi phục con trỏ ngăn xếp
   jr     ra                # Quay trở lại chương trình chính
wrap_end:

# ---------------------------------------------------------------------
# Procedure FACT: tính giai thừa của n
# param[in]  a0  integer n (số nguyên n)
# return     s0  kết quả n! (giai thừa của n)
# ---------------------------------------------------------------------
FACT:
```

```
    addi    sp, sp, -8        # Cấp phát không gian trên ngăn xếp để lưu ra và a0
    sw      ra, 4(sp)         # Lưu thanh ghi ra lên ngăn xếp
    sw      a0, 0(sp)         # Lưu giá trị n (a0) lên ngăn xếp

    li      t0, 2             # Gán t0 = 2
    bge     a0, t0, recursive # Nếu n >= 2, gọi đệ quy
    li      s0, 1             # Nếu n < 2, trả về kết quả là 1 (0! = 1 và 1! = 1)
    j       done              # Kết thúc thủ tục FACT

recursive:
    addi    a0, a0, -1        # Giảm n đi 1 (n = n - 1)
    jal     FACT              # Gọi đệ quy để tính (n-1)!
    lw      s1, 0(sp)         # Phục hồi giá trị n từ ngăn xếp
    mul     s0, s0, s1        # Tính n! = n * (n-1)!

done:
    lw      ra, 4(sp)         # Phục hồi thanh ghi ra từ ngăn xếp
    lw      a0, 0(sp)         # Phục hồi thanh ghi a0 từ ngăn xếp
    addi    sp, sp, 8         # Khôi phục con trỏ ngăn xếp
    jr      ra                # Quay lại địa chỉ gọi ban đầu
fact_end:
```

Output:



Code với n=3:

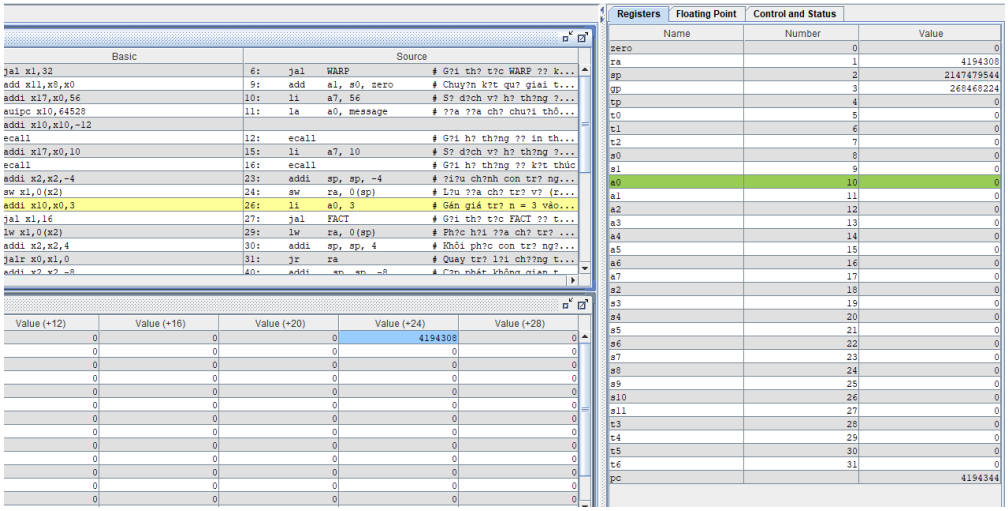
```
.data
message: .asciz  "Ket qua tinh giai thua la: "  # Chuỗi thông báo kết quả

.text
main:
    jal    WARP              # Gọi thủ tục WARP để khởi tạo và tính giai thừa

print:
    add    a1, s0, zero      # Chuyển kết quả giai thừa vào a1 để in ra
    li     a7, 56            # Số dịch vụ hệ thống để in chuỗi
    la     a0, message       # Đưa địa chỉ chuỗi thông báo vào a0
    ecall                    # Gọi hệ thống để in thông báo
```

```
quit:
   li     a7, 10           # Số dịch vụ hệ thống để kết thúc chương trình
   ecall                   # Gọi hệ thống để kết thúc
end_main:

# ------------------------------------------------------------------
# Procedure WARP: gán giá trị n và gọi thủ tục FACT
# ------------------------------------------------------------------
WARP:
   addi   sp, sp, -4       # Điều chỉnh con trỏ ngăn xếp (stack pointer)
   sw     ra, 0(sp)        # Lưu địa chỉ trở về (return address) lên ngăn xếp

   li     a0, 3            # Gán giá trị n = 3 vào thanh ghi a0
   jal    FACT             # Gọi thủ tục FACT để tính giai thừa

   lw     ra, 0(sp)        # Phục hồi địa chỉ trở về từ ngăn xếp
   addi   sp, sp, 4        # Khôi phục con trỏ ngăn xếp
   jr     ra               # Quay trở lại chương trình chính
wrap_end:

# ------------------------------------------------------------------
# Procedure FACT: tính giai thừa của n
# param[in]  a0  integer n (số nguyên n)
# return     s0  kết quả n! (giai thừa của n)
# ------------------------------------------------------------------
FACT:
   addi    sp, sp, -8      # Cấp phát không gian trên ngăn xếp để lưu ra và a0
   sw      ra, 4(sp)       # Lưu thanh ghi ra lên ngăn xếp
   sw      a0, 0(sp)       # Lưu giá trị n (a0) lên ngăn xếp

   li      t0, 2           # Gán t0 = 2
   bge     a0, t0, recursive # Nếu n >= 2, gọi đệ quy
   li      s0, 1           # Nếu n < 2, trả về kết quả là 1 (0! = 1 và 1! = 1)
   j       done            # Kết thúc thủ tục FACT

recursive:
   addi    a0, a0, -1      # Giảm n đi 1 (n = n - 1)
   jal     FACT            # Gọi đệ quy để tính (n-1)!
   lw      s1, 0(sp)       # Phục hồi giá trị n từ ngăn xếp
   mul     s0, s0, s1      # Tính n! = n * (n-1)!

done:
```

```
    lw      ra, 4(sp)        # Phục hồi thanh ghi ra từ ngăn xếp
    lw      a0, 0(sp)        # Phục hồi thanh ghi a0 từ ngăn xếp
    addi    sp, sp, 8        # Khôi phục con trỏ ngăn xếp
    jr      ra               # Quay lại địa chỉ gọi ban đầu
fact_end:
```

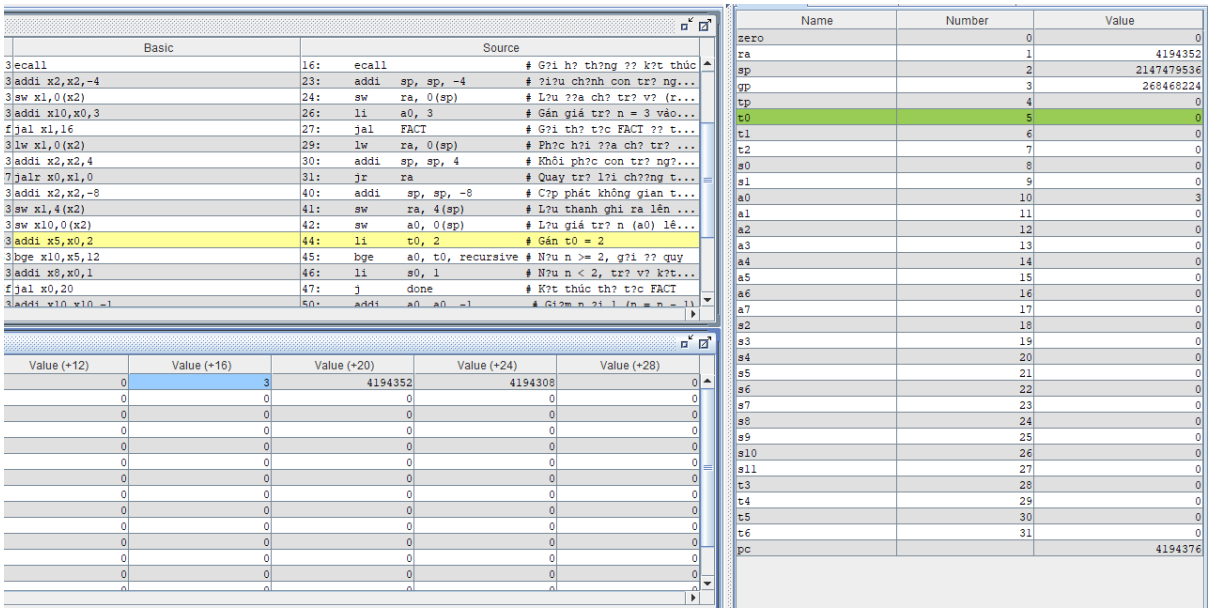
Output:

Kết quả sau khi chạy lệnh li a0, 3:



- **PC (4194344)**: Trỏ đến lệnh tiếp theo cần thực thi.
- **RA (4194308)**: Địa chỉ trả về sau khi hoàn thành chương trình con.
- **SP (2147479544)**: Địa chỉ hiện tại của đỉnh ngăn xếp.
- **A0 (3)**: Tham số đầu vào cho thủ tục tính giai thừa (n = 3).
- **S0 (0)**: Chưa được tính toán, giá trị giai thừa chưa có.
- **Value (+24) (4194308)**: Địa chỉ trả về của chương trình con đã được lưu trong ngăn xếp (ở vị trí +24 byte so với đỉnh ngăn xếp). Khi hoàn thành, chương trình sẽ sử dụng giá trị này để quay lại đúng vị trí trong chương trình gọi ban đầu..

Kết quả sau khi chương trình vào thủ tục FACT:



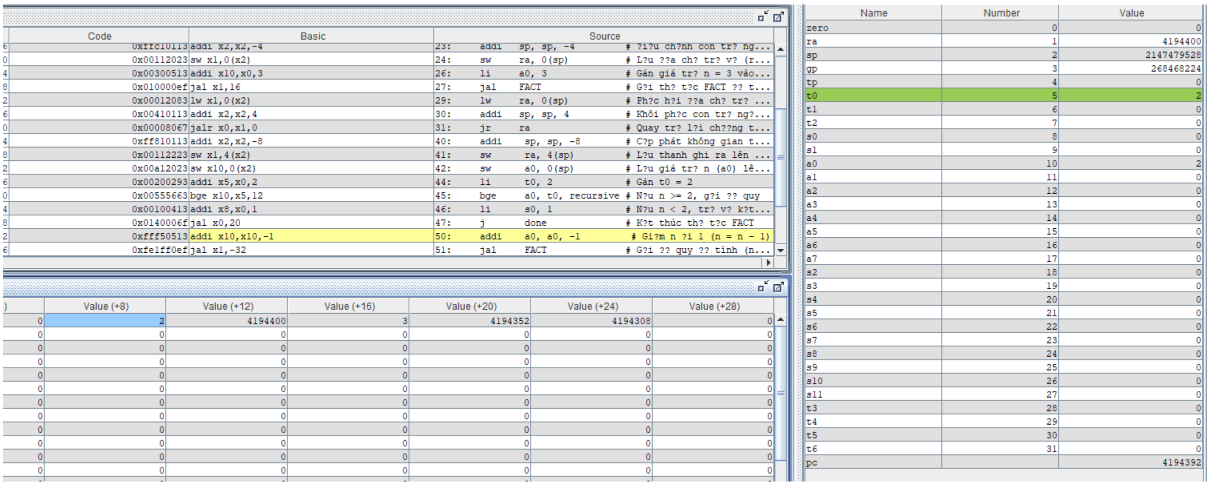
Basic	Source
3 ecall	16: ecall # G?i h? th?ng ?? k?t thúc
3 addi x2,x2,-4	23: addi sp, sp, -4 # ?i?u ch?nh con tr? ng...
3 sw x1,0(x2)	24: sw ra, 0(sp) # L?u ??a ch? tr? v? (r...
3 addi x10,x0,3	26: li a0, 3 # Gán giá tr? n = 3 vào...
f jal x1,16	27: jal FACT # G?i th? t?c FACT ?? t...
3 lw x1,0(x2)	29: lw ra, 0(sp) # Ph?c h?i ??a ch? tr? ...
3 addi x2,x2,4	30: addi sp, sp, 4 # Khôi ph?c con tr? ng?...
7 jalr x0,x1,0	31: jr ra # Quay tr? l?i ch??ng t...
3 addi x2,x2,-8	40: addi sp, sp, -8 # C?p phát không gian t...
3 sw x1,4(x2)	41: sw ra, 4(sp) # L?u thanh ghi ra lên ...
3 sw x10,0(x2)	42: sw a0, 0(sp) # L?u giá tr? n (a0) lê...
3 addi x5,x0,2	44: li t0, 2 # Gán t0 = 2
3 bge x10,x5,12	45: bge a0, t0, recursive # N?u n >= 2, g?i ?? quy
3 addi x8,x0,1	46: li s0, 1 # N?u n < 2, tr? v? k?t...
f jal x0,20	47: j done # K?t thúc th? t?c FACT
3 addi x10,x10,-1	50: addi a0, a0, -1 # Gi?m n ?i 1 (n = n - 1)

Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
0	3	4194352	4194308	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

Name	Number	Value
zero	0	0
ra	1	4194352
sp	2	2147479536
gp	3	268468224
tp	4	0
t0	5	0
t1	6	0
t2	7	0
s0	8	0
s1	9	0
a0	10	3
a1	11	0
a2	12	0
a3	13	0
a4	14	0
a5	15	0
a6	16	0
a7	17	0
s2	18	0
s3	19	0
s4	20	0
s5	21	0
s6	22	0
s7	23	0
s8	24	0
s9	25	0
s10	26	0
s11	27	0
t3	28	0
t4	29	0
t5	30	0
t6	31	0
pc		4194376

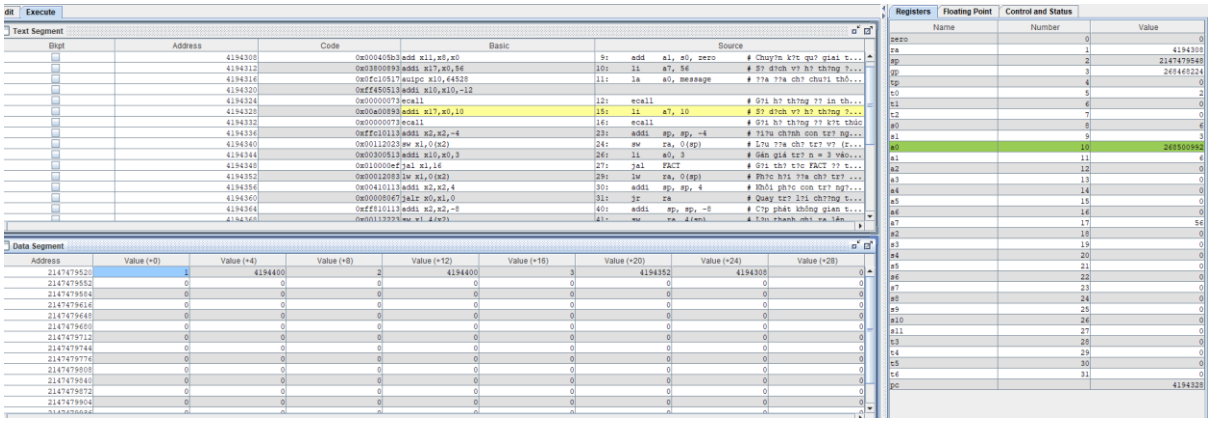
- **PC (4194376)**: Trỏ đến lệnh tiếp theo cần thực thi trong thủ tục FACT.
- **RA (4194352)**: Lưu địa chỉ trả về cho chương trình gọi, để quay lại sau khi hoàn thành thủ tục FACT.
- **SP (2147479536)**: Đã điều chỉnh để tạo không gian trên ngăn xếp, lưu trữ các giá trị tạm thời.
- **A0 (3)**: Giá trị n hiện tại đang được xử lý trong FACT, tương ứng với n = 3.
- **T0 (2)**: Gán giá trị 2 để kiểm tra điều kiện trong thủ tục FACT.
- **Value (+16) (3)**: Lưu trữ giá trị a0 (tức n = 3) vào ngăn xếp.
- **Value (+20) (4194352)**: Lưu giá trị thanh ghi ra (địa chỉ trả về) vào ngăn xếp.

Kết quả sau khi vào nhánh đệ quy (recursive) trong thủ tục Recursive:



- **PC (4194392)**: Trỏ đến lệnh tiếp theo cần thực thi, sau khi đã vào nhánh đệ quy trong thủ tục FACT.
- **RA (4194400)**: Địa chỉ trả về sau khi thực hiện cuộc gọi đệ quy, để quay lại sau khi tính xong giai thừa cho n-1.
- **SP (2147479528)**: Con trỏ ngăn xếp đã điều chỉnh để lưu trữ thêm dữ liệu cho cuộc gọi đệ quy mới.
- **T0 (2)**: Giá trị 2 được sử dụng để so sánh với a0 (n) nhằm kiểm tra điều kiện đệ quy.
- **A0 (2)**: Giá trị của n đã giảm xuống còn 2 do lệnh addi a0, a0, -1 (n = n - 1).
- **Value (+8) (2)**: Giá trị a0 = 2 đã được lưu trên ngăn xếp để tiếp tục cuộc gọi đệ quy.
- **Value (+12), (+16), (+20)**: Các giá trị của địa chỉ trả về trước đó (4194352) và các giá trị khác đã được lưu lại trên ngăn xếp trong các bước trước đó.

Kết quả sau khi chương trình kết thúc:



- **PC (4194328)**: Trỏ đến địa chỉ kết thúc chương trình, không còn lệnh nào cần thực thi.
- **RA (4194308)**: Địa chỉ trở về từ lời gọi chương trình con, đã không còn sử dụng sau khi hoàn thành chương trình.
- **SP (2147479548)**: Con trỏ ngăn xếp đã được phục hồi về vị trí ban đầu.
- **A0 (268500992)**: Kết quả của phép tính giai thừa đã được tính xong, nhưng có vẻ giá trị trong a0 không phải là giá trị đúng của giai thừa của 3 (nên kiểm tra lại quá trình thực hiện).
- **S0 (6)**: Giá trị kết quả đúng, là giai thừa của 3! = 6, đã được lưu trong thanh ghi s0.
- **A1 (6)**: Giá trị để in ra giai thừa 6 được chuyển vào thanh ghi a1 để in thông báo kết quả.
- **A7 (10)**: Giá trị 10 trong thanh ghi a7 biểu thị lệnh ecall để kết thúc chương trình.

Tổng kết:

1. Thanh ghi PC (Program Counter):

- **Chức năng**: PC giữ địa chỉ của lệnh tiếp theo cần thực thi. Mỗi khi một lệnh được thực hiện, PC được cập nhật để trỏ đến lệnh tiếp theo.
- **Sự thay đổi**:
    - Mỗi lần chương trình con được gọi (jal), PC được cập nhật để trỏ đến địa chỉ của chương trình con.
    - Sau khi kết thúc chương trình con, PC sẽ quay về địa chỉ tiếp theo của chương trình gọi, nhờ giá trị trong thanh ghi ra.

2. Thanh ghi RA (Return Address):

- **Chức năng**: Lưu địa chỉ mà chương trình con sẽ quay trở về sau khi hoàn thành.
- **Sự thay đổi**:
    - Khi chương trình gọi một chương trình con, ra được gán địa chỉ của lệnh tiếp theo trong chương trình chính.
    - Trong các cuộc gọi đệ quy, giá trị của ra liên tục được cập nhật và lưu trữ vào ngăn xếp, sau đó phục hồi lại khi quá trình đệ quy kết thúc.
    - Cuối cùng, khi chương trình con hoàn thành, nó sử dụng giá trị trong ra để quay lại đúng vị trí trong chương trình gọi.

3. Thanh ghi SP (Stack Pointer):

- **Chức năng**: Con trỏ ngăn xếp, chỉ vào đỉnh của ngăn xếp, nơi lưu trữ dữ liệu tạm thời như địa chỉ trở về và giá trị của các thanh ghi cần lưu.
- **Sự thay đổi**:
    - Mỗi lần một chương trình con được gọi, sp được điều chỉnh để cấp phát không gian trên ngăn xếp nhằm lưu trữ địa chỉ ra và các tham số/giá trị tạm thời khác.
    - Khi quá trình đệ quy xảy ra, con trỏ ngăn xếp liên tục được điều chỉnh để tạo không gian cho mỗi cuộc gọi mới.
    - Sau khi chương trình con kết thúc và các giá trị đã được phục hồi, sp quay trở về vị trí ban đầu để giải phóng không gian đã cấp phát.

4. Thanh ghi A0:

- **Chức năng**: Được sử dụng để truyền tham số đầu vào (ví dụ, giá trị n trong bài toán tính giai thừa) và lưu trữ kết quả trả về.
- **Sự thay đổi**:
    - a0 ban đầu chứa giá trị đầu vào (ví dụ: n = 3).
    - Trong quá trình đệ quy, giá trị của a0 liên tục được giảm xuống (n = n - 1) cho đến khi đạt điều kiện cơ sở (n = 1).

- Sau khi hoàn tất quá trình tính toán, giá trị trả về cuối cùng được lưu trong a0.

5. Thanh ghi S0:

- **Chức năng**: Lưu trữ kết quả tính toán trung gian và cuối cùng (ví dụ, kết quả của giai thừa).
- **Sự thay đổi**:
  - s0 ban đầu được khởi tạo là 0.
  - Khi cuộc gọi đệ quy tiến hành, giá trị của s0 được cập nhật với kết quả của từng bước đệ quy (giai thừa của n-1).
  - Cuối cùng, khi chương trình hoàn tất, s0 chứa kết quả cuối cùng của phép tính giai thừa (ví dụ, 3! = 6).

Assignment 5

Code:

```
.data
    msg1: .string "Largest: "
    msg2: .string ", "
    msg3: .string "\nSmallest: "
    newline: .string "\n"

.text
main:
    # Cấp phát bộ nhớ stack cho 8 số + 4 kết quả (max, maxpos, min, minpos)
    addi sp, sp, -48

    # Lưu các giá trị test vào stack
    li t0, 5
    sw t0, 0(sp)   # a0
    li t0, -2
    sw t0, 4(sp)   # a1
    li t0, 7
    sw t0, 8(sp)   # a2
    li t0, 9
    sw t0, 12(sp)  # a3
    li t0, 1
    sw t0, 16(sp)  # a4
```

```
li t0, 12
sw t0, 20(sp)  # a5
li t0, -3
sw t0, 24(sp)  # a6
li t0, -6
sw t0, 28(sp)  # a7

# Gọi hàm tìm max/min
jal ra, find_max_min

# In "Largest: "
la a0, msg1
li a7, 4
ecall

# In giá trị max
lw a0, 32(sp)
li a7, 1
ecall

# In ", "
la a0, msg2
li a7, 4
ecall

# In vị trí max
lw a0, 36(sp)
li a7, 1
ecall

# In "\nSmallest: "
la a0, msg3
li a7, 4
ecall

# In giá trị min
lw a0, 40(sp)
li a7, 1
ecall

# In ", "
la a0, msg2
li a7, 4
```

```
    ecall

    # In vị trí min
    lw a0, 44(sp)
    li a7, 1
    ecall

    # In newline
    la a0, newline
    li a7, 4
    ecall

    # Giải phóng stack và kết thúc
    addi sp, sp, 48
    li a7, 10
    ecall

find_max_min:
    # Khởi tạo giá trị max, min là phần tử đầu tiên
    lw t0, 0(sp)   # t0 = max value
    mv t1, t0      # t1 = min value
    li t2, 0       # t2 = max position
    li t3, 0       # t3 = min position
    li t4, 1       # t4 = counter (bắt đầu từ 1 vì đã lấy phần tử 0)
    li t5, 8       # t5 = size of array

loop:
    beq t4, t5, end_loop    # Nếu đã duyệt hết thì thoát

    # Load giá trị hiện tại
    slli t6, t4, 2          # t6 = t4 * 4 (offset)
    add t6, sp, t6          # t6 = địa chỉ phần tử hiện tại
    lw s1, 0(t6)            # s1 = giá trị hiện tại

    # So sánh với max
    bge t0, s1, check_min   # Nếu max >= current thì kiểm tra min
    mv t0, s1               # Update max value
    mv t2, t4               # Update max position

check_min:
    ble t1, s1, continue    # Nếu min <= current thì continue
    mv t1, s1               # Update min value
    mv t3, t4               # Update min position
```

```
continue:
    addi t4, t4, 1        # Tăng counter
    j loop

end_loop:
    # Lưu kết quả vào stack
    sw t0, 32(sp)         # max value
    sw t2, 36(sp)         # max position
    sw t1, 40(sp)         # min value
    sw t3, 44(sp)         # min position

    ret
```

Output:

```
-- program is finished running (0) --

Largest: 12, 5
Smallest: -6, 7

-- program is finished running (0) --
```

➔ Đúng với yêu cầu đề bài