BÁO CÁO THỰC HÀNH KIẾN TRÚC MÁY TÍNH TUẦN 7

Họ và tên: Hoàng Văn Thắng

MSSV: 20235828

Assignment 1

Tạo project để thực hiện Home Assignment 1. Dịch và chạy mô phỏng. Thay đổi các tham số chương trình (thanh ghi **a0**) và quan sát kết quả thực hiện. Chạy chương trình ở chế độ từng dòng lệnh và chú ý sự thay đổi của các thanh ghi, đặc biệt là thanh ghi **pc** và **ra**.

Nhập chương trình

```
1  # Laboratory Exercise 7 Home Assignment 1
2  .text
3  main:
4          li      a0, -21         # load input parameter
5          jal     abs             # jump and link to abs procedure
6
7          li      a7, 10          # terminate
8          ecall
9  end_main:
10 # ---------------------------------------------------------------
11 # function abs
12 # param[in]    a0     the interger need to be gained the absolute value
13 # return       s0     absolute value
14 # ---------------------------------------------------------------
15 abs:
16         sub     s0, zero, a0    # put -a0 in s0, in case a0 < 0
17         blt     a0, zero, done  # if a0 < 0 then done
18         add     s0, a0, zero    # else put a0 in s0
19 done:
20         jr      ra
```

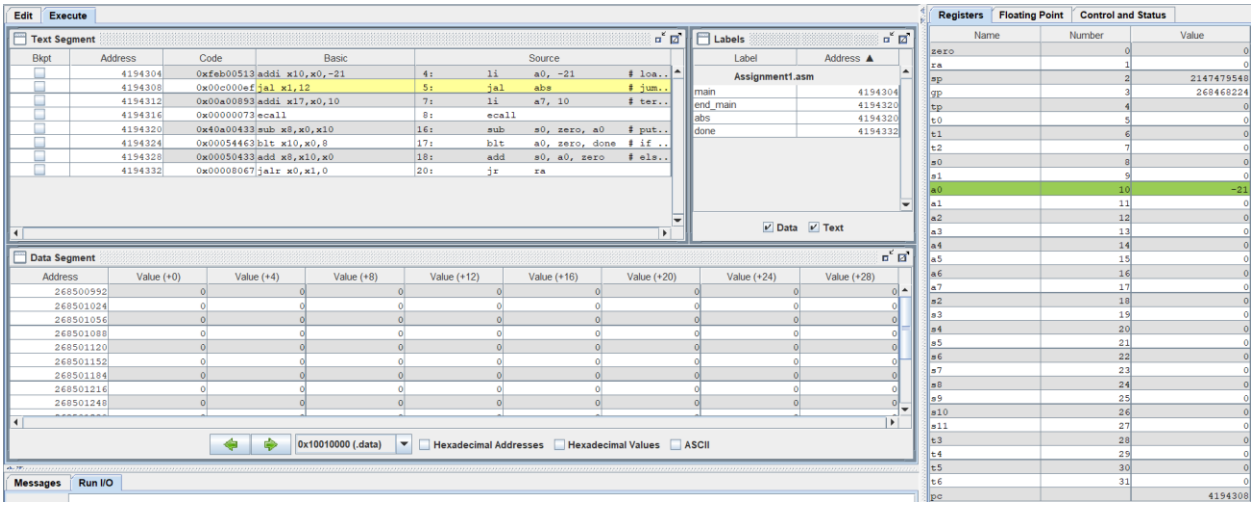
```
# Laboratory Exercise 7 Home Assignment 1
.text
main:
        li      a0, -21             # load input parameter
        jal     abs             # jump and link to abs procedure

        li      a7, 10          # terminate
        ecall
end_main:
```

```
# ---------------------------------------------------------------
# function abs
# param[in]    a0     the interger need to be gained the absolute value
# return       s0     absolute value
# ---------------------------------------------------------------
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
```

Kết quả chạy

- Trước khi vào lệnh jal abs:



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 = -21**: Thanh ghi **a0** được dùng để chứa tham số đầu vào cho hàm **abs**. Giá trị này là số nguyên **-21** (hex: 0xffffffeb), 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	-21
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 của chương trình:

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	21
s1	9	0
a0	10	-21
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
pc		4194320

- **ra** (return address) = **4194312**: Đây là địa chỉ mà chương trình sẽ quay trở lại sau khi thực hiện xong chương trình con **abs**. Nó tương ứng với lệnh tiếp theo sau khi gọi *jal abs*.
- **a0 = - 21**
- **s0 = 21**: 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 bằng cách gọi hệ thống qua ecall.

Như vậy, chương trình đã được thực thi thành công việc tính giá trị tuyệt đối của **a0** (-21) và trả về kết quả 21 trong thanh ghi **s0**. Sau đó, chương trình kết quả với lệnh ecall.

Assignment 2

Tạo project để thực hiện Home Assignment 2. Dịch và chạy mô phỏng. Thay đổi các tham số chương trình (thanh ghi **a0**, **a1**, **a2**) và quan sát kết quả thực hiện. Chạy chương trình ở chế độ từng dòng lệnh và chú ý sự thay đổi của các thanh ghi, đặc biệt là thanh ghi **pc** và **ra**.

Nhập chương trình

```
1  # Laboratory Exercise 7, Home Assignment 2
2  .text
3  main:
4          li      a0, 21          # load first test input
5          li      a1, 2           # load second test input
6          li      a2, 5           # load third test input
7          jal     max             # call max procedure
8
9          li      a7, 10          # terminate
10         ecall
11 end_main:
12
13 # ------------------------------------------------------------------------
14 # Procedure max: find the largest of three integers
15 # param[in]  a0  integers
16 # param[in]  a1  integers
17 # param[in]  a2  integers
18 # return     s0   the largest value
19 # ------------------------------------------------------------------------
20 max:
21         add     s0, a0, zero    # copy a0 in s0; largest so far
22         sub     t0, a1, s0      # compute a1 - s0
23         blt     t0, zero, okay  # if a1 - s0 < 0 then no change
24         add     s0, a1, zero    # else a1 is largest thus far
25 okay:
26         sub     t0, a2, s0      # compute a2 - s0
27         bltz    t0, zero, done  # if a2 - s0 < 0 then no change
28         add     s0, a2, zero    # else a2 is largest overall
29 done:
30         jr      ra              # return to calling program
```

```
# Laboratory Exercise 7, Home Assignment 2
.text
main:
        li      a0, 21          # load first test input
```

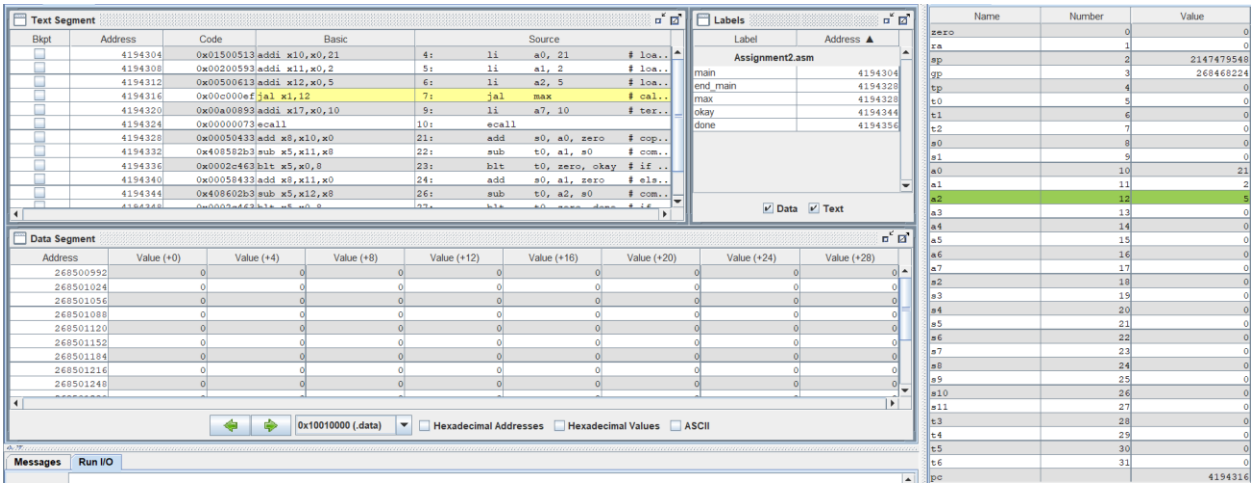
```
        li      a1, 2           # load second test input
        li      a2, 5           # load third test input
        jal     max             # call max procedure

        li      a7, 10          # terminate
        ecall
end_main:

# ------------------------------------------------------------------
# Procedure max: find the largest of three integers
# param[in]  a0  integers
# param[in]  a1  integers
# param[in]  a2  integers
# return     s0   the largest value
# ------------------------------------------------------------------
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 - s0 < 0 then no change
        add     s0, a1, zero  # else a1 is largest thus far
okay:
        sub     t0, a2, s0    # compute a2 - s0
        bltz    t0, zero, done        # if a2 - s0 < 0 then no change
        add     s0, a2, zero  # else a2 is largest overall
done:
        jr      ra              # return to calling program
```

Kết quả chạy:

- Trước khi vào câu lệnh *jal max*:



- **a0 = 21**: 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	21
a1	11	2
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 = 4194320**: Sau khi thực hiện *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 **4194320** (0x00400010), 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:

Name	Number	Value
zero	0	0
ra	1	4194320
sp	2	2147479548
gp	3	268468224
tp	4	0
t0	5	-16
t1	6	0
t2	7	0
s0	8	21
s1	9	0
a0	10	21
a1	11	2
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

- **a0 = 21**: Đây là tham số đầu vào thứ nhất, giá trị ban đầu là **21**.
- **a1 = 2**: Đây là tham số đầu vào thứ hai, giá trị ban đầu là 2.
- **a2 = 5**: Đây là tham số đầu vào thứ ba, giá trị ban đầu là 5
- **s0 = 21**: 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 **21**, **2** và **5** là **21**, 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.

Assignment 3

Tạo project để thực hiện Home Assignment 3. Dịch và chạy mô phỏng. Thay đổi tham số chương trình (thanh ghi **s0**, **s1**), quan sát quá trình và kết quả thực hiện. Chú ý sự thay đổi giá trị của thanh ghi **sp**. Quan sát vùng nhớ được trỏ bởi thanh ghi **sp** trong cửa sổ Data Segment.

Nhập chương trình

```
# Laboratory Exercise 7, Home Assignment 3
.text
main:
        li      s0, 69          # Gán giá trị 69 cho thanh ghi s0
        li      s1, 96          # Gán giá trị 96 cho thanh ghi s1
        jal     stack
        # Kết thúc chương trình
        li      a7, 10
        ecall

stack:
        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
        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
```


```
# Laboratory Exercise 7, Home Assignment 3
.text
main:
        li      s0, 69          # Gán giá trị 69 cho thanh ghi s0
        li      s1, 96          # Gán giá trị 96 cho thanh ghi s1
        jal     stack
        # Kết thúc chương trình
        li      a7, 10
        ecall

stack:
        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
        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
```


Kết quả chạy

- 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 chương trình con stack:

Name	Number	Value
zero	0	0
ra	1	4194316

Giá trị thanh ghi **ra** thay đổi

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

Name	Number	Value
zero	0	0
ra	1	4194316
sp	2	2147479540

Giá trị của thanh ghi sp đã giảm đi 8

- Sau 2 câu lệnh sw:

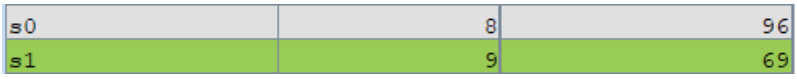
Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
2147479520	0	0	0	0	0	96	69	0
2147479552	0	0	0	0	0	0	0	0
2147479584	0	0	0	0	0	0	0	0
2147479616	0	0	0	0	0	0	0	0
2147479648	0	0	0	0	0	0	0	0
2147479680	0	0	0	0	0	0	0	0
2147479712	0	0	0	0	0	0	0	0
2147479744	0	0	0	0	0	0	0	0
2147479776	0	0	0	0	0	0	0	0

current sp Hexadecimal Addresses Hexadecimal Values ASCII

Giá trị 96 và 69 đã được lưu vào stack

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



s0	8	96
s1	9	69

Kết luận: Chức năng của stack trong chương trình:

- 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 ví trí ban đầu trong chương trình chính.
- Phục hồi địa chỉ quay lại: Sau khi thực hiện xong chương trình con, 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

Tạo project để thực hiện Home Assignment 4. Dịch và chạy mô phỏng. Thay đổi tham số ở thanh ghi **a0** và kiểm tra kết quả ở thanh ghi **s0**. Chạy chương trình ở chế độ từng dòng lệnh và quan sát sự thay đổi giá trị của các thanh ghi **pc**, **ra**, **sp**, **a0**, **s0**. Liệt kê các giá trị trong vùng nhớ ngăn xếp khi thực hiện chương trình với n = 3.

Nhập chương trình

Với **a0 = 5**

```
# Laboratory Exercise 7, Home 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
        ecall
```

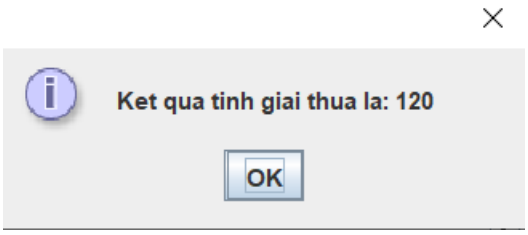
```
end_main:

# ---------------------------------------------------------------------
# Procedure WARP: assign value and call FACT
# ---------------------------------------------------------------------
WARP:
        addi   sp, sp, -4      # adjust stack pointer
        sw     ra, 0(sp)       # save return address
        li     a0, 5           # 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
warp_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:
```

Kết quả chạy



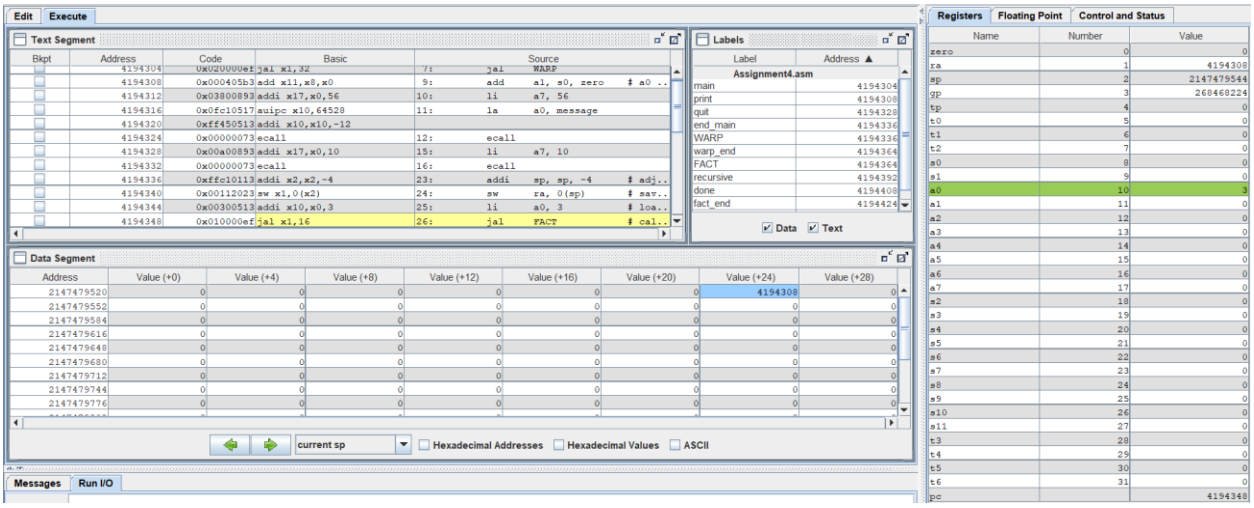
Name	Number	Value
zero	0	0
ra	1	4194308
sp	2	2147479548
gp	3	268468224
tp	4	0
t0	5	2
t1	6	0
t2	7	0
s0	8	120
s1	9	5
a0	10	268500992
a1	11	120
a2	12	0
a3	13	0
a4	14	0
a5	15	0
a6	16	0
a7	17	10
s2	18	0
s3	19	0
s4	20	0
s5	21	0
s6	22	0
s7	23	0
s8	24	0
s9	25	0
s10	26	0
s11	27	0
t3	28	0
t4	29	0
t5	30	0
t6	31	0
pc		4194336

Đúng với lý thuyết 5! = 120

Với **n = 3**:

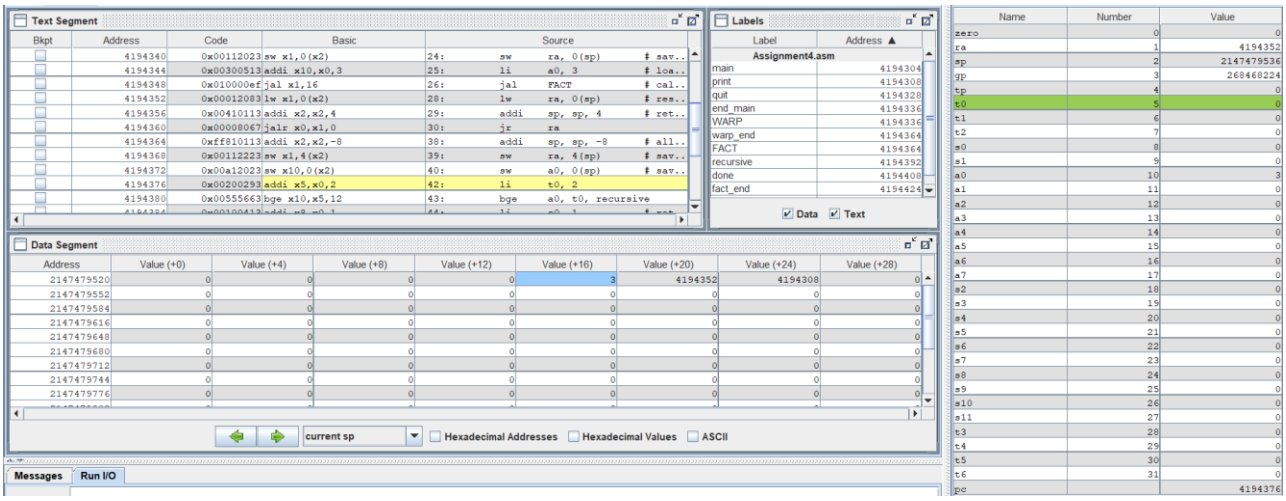
Kết quả chạy

- Sau khi chạy lệnh *li a0, 3*:



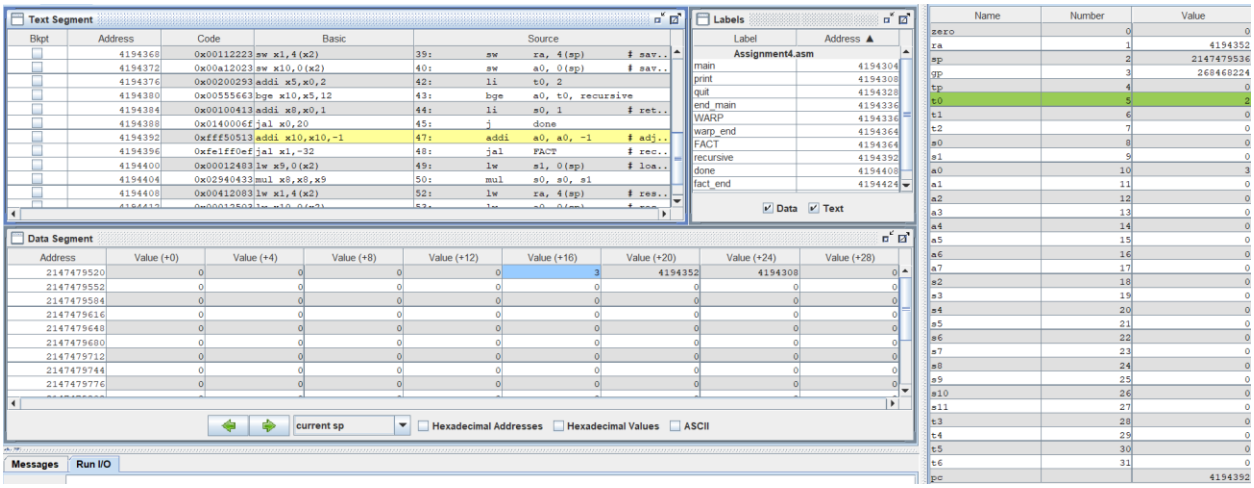
- **PC** (4194344): Trỏ đến lệnh tiếp theo cần thực thi.
- **RA** (4194308): Địa chỉ trả về sau khi hoàn thành chương trình con
- **SP** (2147479544): Địa chỉ hiện tại của đỉnh ngăn xếp
- **A0** (3): Tham số đầu vào cho thủ tục tính giai thừa (n = 3).
- **S0** (0): Chưa được tính toán, giá trị giai thừa 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:



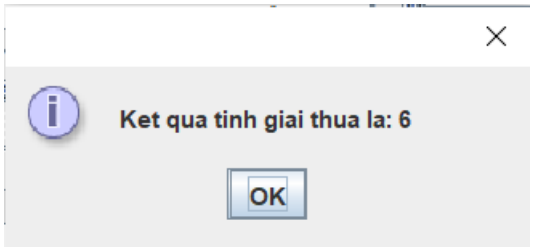
- **PC** (4194376): Trỏ đến lệnh tiếp theo cần thực thi trong thủ tục FACT.
- **RA** (4194352): Lưu địa chỉ trả về cho chương trình gọi, để quay lại sau khi hoàn thành thủ tục FACT.
- **SP** (2147479536): Đã điều chỉnh để tạo không gian trên ngăn xếp, lưu trữ các giá trị tạm thời.
- **A0** (3): Giá trị 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 giá trị a0 (tức n = 3) vào ngăn xếp
- **Value** (+20) (4194352): Lưu giá trị trang ghi ra (địa chỉ trả về) vào ngăn xếp.

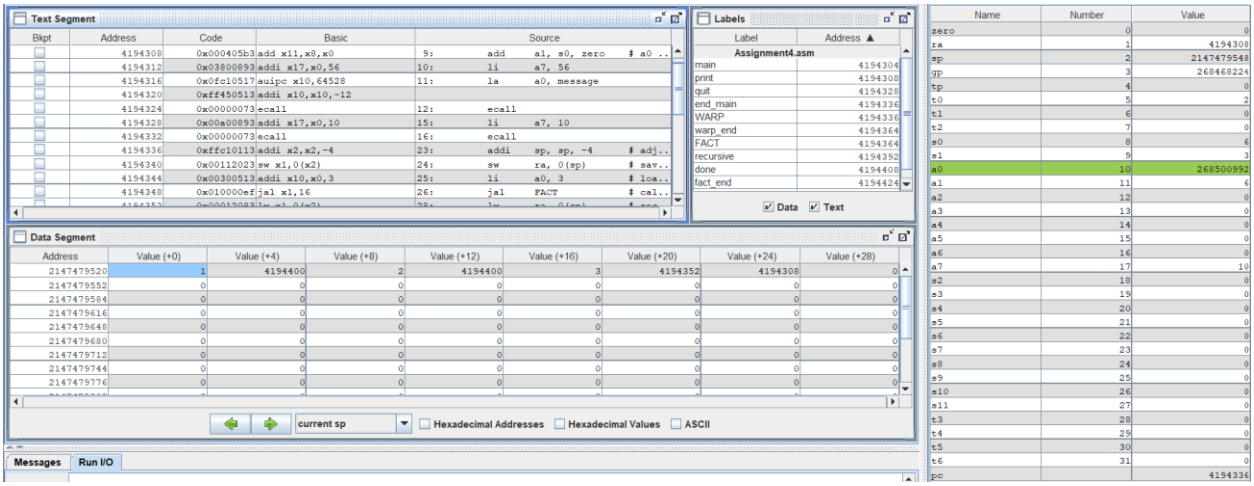
Kết quả sau khi vào nhánh đệ quy (recursive) trong thủ tục Recursive:



- **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 sanh 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.

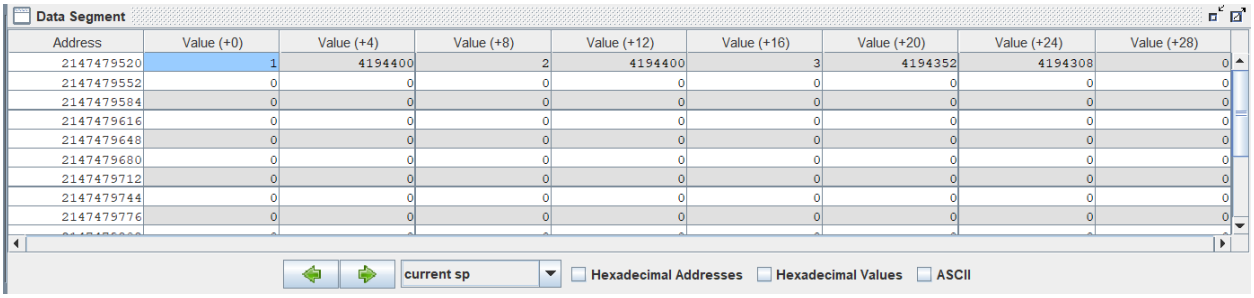
- Kết quả sau khi chương trình kết thúc:





- **PC** (4194336): 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.

Các giá trị trong vùng nhớ ngăn xếp khi thực hiện chương trình với n = 3



Tổng kết:

- 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, 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.
- 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 trong 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.
- Thanh ghi SP (Stach 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.
- 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. Trong quá trình đệ quy, giá trị của a0 liên tục được giảm xuống cho đến khi đạt đến điều kiện cơ sở. Sau khi hoàn tát tính toán, giá trị trả về cuối cùng được lưu trong a0.
- Thanh ghi s0:
    - Chức năng: Lưu trữ kết quả tính toán trung gian và cuối cùng
    - 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 ((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

Viết chương trình con tìm giá trị lớn nhất, nhỏ nhất và vị trí tương ứng trong gồm 8 số nguyên được lưu trữ trong các thanh ghi từ a0 đến a7. Ví dụ:

Largest: 9, 3 => Giá trị lớn nhất là 9 được lưu trữ trong a3

Smallest: -3, 6 => Giá trị nhỏ nhất là -3 được lưu trữ trong a6

Nhập chương trình

```
.data
        msg1: .string "Largest: "
        msg2: .string ", "
        msg3: .string "\Smallest: "
        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 check
        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

    jal     ra, find_max_min

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

    # In gía 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: "
    li      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
        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)
        mv      t1, t0
        li      t2, 0
        li      t3, 0
        li      t4, 1
        li      t5, 8

loop:
        beq     t4, t5, end_loop

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

        # 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
```

Kết quả chạy

```
Largest: 12, 5
Smallest: -6, 7

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

Đúng với yêu cầu đề bài

Giải thích chương trình: Chương trình đang dùng thuật toán brute force để tìm max, min