

# BÁO CÁO THÚC HÀNH KIẾN TRÚC MÁY TÍNH TUẦN 7

Họ và tên: Phan Khánh Vũ

MSSV: 20235880

## 1. Assignment 1

- Nhập chương trình:

```
# Laboratory Exercise 7 Home Assignment 1
.text
main:
    li a0, -45        # Tải tham số đầu vào
    jal abs            # Chuyển điều khiển đến thủ tục abs
    li a7, 10          # Kết thúc chương trình
    ecall
end_main:
abs:
    sub s0, zero, a0    # Đặt -a0 vào s0; trong trường hợp a0 < 0
    blt a0, zero, done  # Nếu a0 < 0 thì nhảy đến done
    add s0, a0, zero    # Ngược lại, gán a0 vào s0
done:
    jr ra    # Trả về địa chỉ gọi
```

- Các bước thực hiện của chương trình:

- + Các câu lệnh trong thẻ main: để khai báo giá trị của số cần tính.
- + Câu lệnh jal abs để gọi chương trình con để tính abs.
- + Câu lệnh trong thẻ abs để thực hiện tính trị tuyệt đối của số: lấy số đối của số đó, sau đó so sánh với 0 nếu số ban đầu lớn hơn 0 thì nhảy đến done còn nếu nhỏ hơn 0 thì tiếp tục thực hiện gán giá trị của số đối cho số ban đầu.

- Chạy chương trình :

The screenshot shows a debugger interface with the following components:

- Assembly Editor:** Displays the assembly code for "Lab07\_1.asm". The code includes instructions like `li a0, -45` (Load a0 with -45), `jal abs` (Jump to absolute address), and `ecall` (End call).
- Registers:** A table showing register values. Key values include `a0` at 0xfffffd3, `sp` at 2, and `pc` at 0x00400010.
- Messages:** A log window showing the message "-- program is finished running (0) --" repeated twice.

- Quan sát kết quả của các thanh ghi và chương trình:

Code	Basic	Source	Registers
0xfd300513 addi x10,x0,0xfffffd3		4: li a0, -45 # Tải tham số đầu vào	sp 2 0x7ffffeffc
0x00c000e1 jal x1,0x0000000c		5: jal abs # Chuyển điều khiển đến t..	gp 3 0x10000000
0x00a00893 addi x17,x0,10		6: li a7, 10 # Kết thúc chương trình	tp 4 0x00000000
0x00000073 ecall		7: ecall	t0 5 0x00000000
0x40a00433 sub x8,x0,x10		11: sub s0, zero, a0 # Đặt -a0 vào s0; tro..	t1 6 0x00000000
0x00054463 blt x10,x0,0x00000008		12: blt a0, zero, done # Nếu a0 < 0 thì nhảy..	t2 7 0x00000000
0x00050433 add x8,x10,x0		13: add s0, a0, zero # Ngược lại, gán a0 v..	s0 8 0x00000000
0x00008067 jalr x0,x1,0		15: jr ra # Trả về địa chỉ gọi	s1 9 0x00000000
			a0 10 0xfffffd3
			a1 11 0x00000000
			a2 12 0x00000000

Code	Basic	Source	Registers
0xfd300513 addi x10,x0,0xfffffd3		4: li a0, -45 # Tải tham số đầu vào	sp 2 0x7ffffeffc
0x00c000e1 jal x1,0x0000000c		5: jal abs # Chuyển điều khiển đến t..	gp 3 0x10000000
0x00a00893 addi x17,x0,10		6: li a7, 10 # Kết thúc chương trình	tp 4 0x00000000
0x00000073 ecall		7: ecall	t0 5 0x00000000
0x40a00433 sub x8,x0,x10		11: sub s0, zero, a0 # Đặt -a0 vào s0; tro..	t1 6 0x00000000
0x00054463 blt x10,x0,0x00000008		12: blt a0, zero, done # Nếu a0 < 0 thì nhảy..	t2 7 0x00000000
0x00050433 add x8,x10,x0		13: add s0, a0, zero # Ngược lại, gán a0 v..	s0 8 0x00000000
0x00008067 jalr x0,x1,0		15: jr ra # Trả về địa chỉ gọi	s1 9 0x00000000
			a0 10 0xfffffd3
			a1 11 0x00000000
			a2 12 0x00000000

Code	Basic	Source	Registers
0xfd300513 addi x10,x0,0xfffffd3		4: li a0, -45 # Tải tham số đầu vào	sp 2 0x7ffffeffc
0x00c000e1 jal x1,0x0000000c		5: jal abs # Chuyển điều khiển đến t..	gp 3 0x10000000
0x00a00893 addi x17,x0,10		6: li a7, 10 # Kết thúc chương trình	tp 4 0x00000000
0x00000073 ecall		7: ecall	t0 5 0x00000000
0x40a00433 sub x8,x0,x10		11: sub s0, zero, a0 # Đặt -a0 vào s0; tro..	t1 6 0x00000000
0x00054463 blt x10,x0,0x00000008		12: blt a0, zero, done # Nếu a0 < 0 thì nhảy..	t2 7 0x00000000
0x00050433 add x8,x10,x0		13: add s0, a0, zero # Ngược lại, gán a0 v..	s0 8 0xfffffd3
0x00008067 jalr x0,x1,0		15: jr ra # Trả về địa chỉ gọi	s1 9 0x00000000
			a0 10 0xfffffd3
			a1 11 0x00000000
			a2 12 0x00000000

+Với giá trị của a0 được khởi tạo là a0 = -45 ta quan sát các thanh ghi:

⇒ Kết quả của thanh ghi  $a0 = -45$  và  $s0 = 45$  nên chương trình hoạt động bình thường.

No.	Vị trí	a0	s0	pc	ra	Chú thích
1	(3)	0x000000	0x000000	0x400000	0x000000	Bắt đầu chương trình
2	(5)	0xfffffd3	0x000000	0x400004	0x00008	Nhập số cần xử lí
3	(13)	0xfffffd3	0x0002d	0x0040001c	0x00008	Lấy trị tuyệt đối
4	(7)	0xfffffd3	0x0002d	0x00400010	0x00008	Kết thúc chương trình

- **Thay đổi các tham số chương trình (thanh ghi a0)**
  - + Với giá trị  $a0 = 36$  quan sát kết quả chương trình:

The screenshot shows a debugger interface with the following sections:

- Editor:** Displays the assembly code for Lab07\_1.asm. The code initializes  $a0$  to 36, calls `abs`, and then prints the result back to  $a0$ .
- Registers:** A table showing register values.  $a0$  is highlighted in green and has a value of 0xfffffd3. Other registers like zero, ra, sp, etc., have values starting from 0.
- Messages:** Shows the output of the program: "program is finished running (0)".

⇒ Kết quả của thanh ghi  $a0 = 36$  và  $s0 = 36$  nên chương trình hoạt động bình thường.

+ Với giá trị  $a0 = 0$  quan sát kết quả chương trình:

The screenshot shows a debugger window with the following sections:

- Registers:** A table showing register values from zero to 31.
- Floating Point:** Not applicable for this assembly code.
- Control and Status:** Not applicable for this assembly code.
- Messages:** Displays two lines: "-- program is finished running (0) --" repeated twice.

```

Edit Execute
Lab07_1.asm
1 # Laboratory Exercise 7 Home Assignment 1
2 .text
3 main:
4 li a0, 0      # Tải tham số đầu vào
5 jal abs       # Chuyển điều khiển đến thob tục abs
6 li a7, 10     # Kết thúc chương trình
7 ecall
8 end_main:
9
10 abs:
11 sub s0, zero, a0  # Bắt -a0 vào s0; trong trường hợp a0 < 0
12 bit a0, zero, done # Nếu a0 < 0 thì nhảy đến done
13 add s0, a0, zero # Ngược lại, gán a0 vào s0
14 done:
15 jr ra # Trở về địa chỉ gọi

Line: 4 Column: 14 ✓ Show Line Numbers
Messages Run I/O
-- program is finished running (0) --
Clear
-- program is finished running (0) --

```

⇒ Kết quả của thanh ghi  $a0 = 0$  và  $s0 = 0$  nên chương trình hoạt động bình thường.

## 2. Assignment 2

### - Nhập chương trình:

The screenshot shows an assembly editor with two tabs:

- Lab07\_1.asm:** Contains the assembly code for the first assignment.
- Lab07\_2.asm:** Contains the assembly code for the second assignment.

```

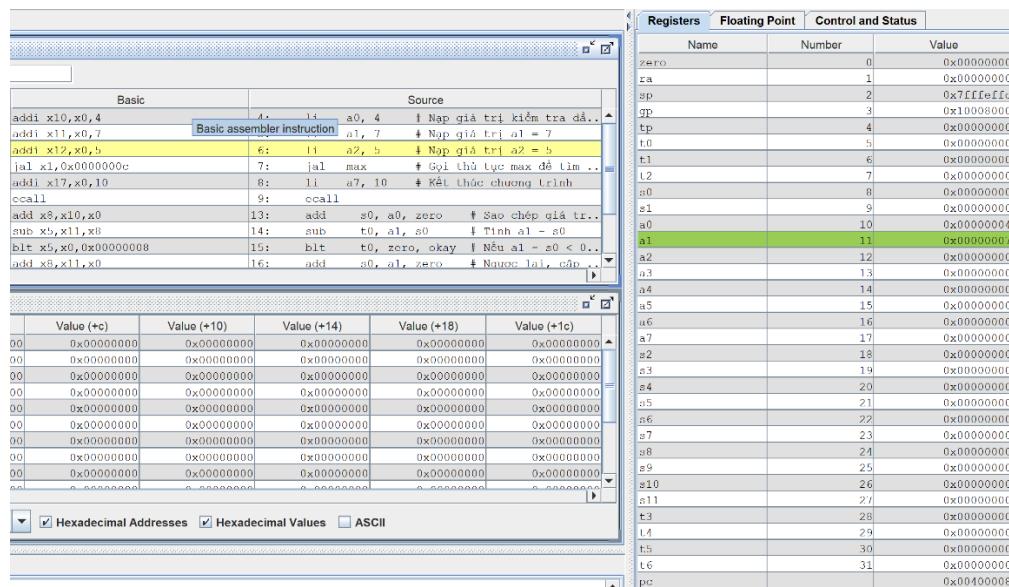
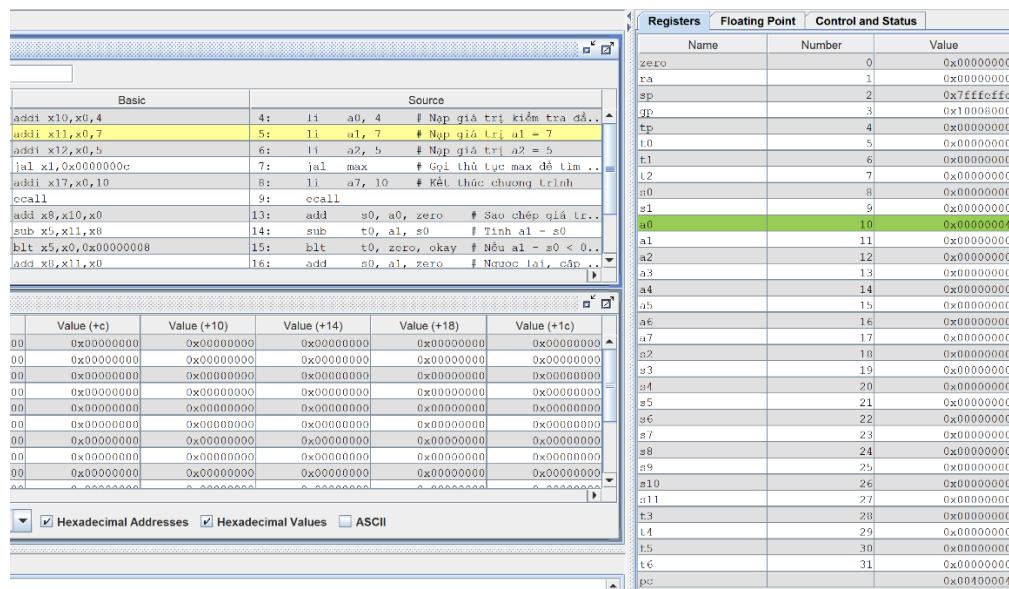
Lab07_1.asm Lab07_2.asm
1 # Laboratory Exercise 7, Home Assignment 2
2 .text
3 main:
4 li a0, 4      # Nạp giá trị kiểm tra đầu vào (a0 = 4)
5 li a1, 7      # Nạp giá trị a1 = 7
6 li a2, 5      # Nạp giá trị a2 = 5
7 jal max      # Gọi thủ tục max để tìm giá trị lớn nhất
8 li a7, 10     # Kết thúc chương trình
9 ecall
10 end_main:
11
12 max:
13 add s0, a0, zero # Sao chép giá trị a0 vào s0; giả sử là lớn nhất
14 sub t0, a1, s0   # Tính a1 - s0
15 blt t0, zero, okay # Nếu a1 - s0 < 0 thì giữ nguyên s0
16 add s0, a1, zero # Ngược lại, cập nhật s0 thành a1 (a1 lớn hơn)
17 okay:
18 sub t0, a2, s0   # Tính a2 - s0
19 blt t0, zero, done # Nếu a2 - s0 < 0 thì giữ nguyên s0
20 add s0, a2, zero # Ngược lại, cập nhật s0 thành a2 (a2 lớn nhất)
21
22 done:
23 jr ra          # Quay lại chương trình gọi
24

```

## - Các bước thực hiện chương trình:

- + Các câu lệnh sau thẻ main để khai báo các giá trị của chương trình:  
a0=4, a1 = 7, a2 = 5, sau đó gọi hàm con tìm max.
- + Các câu lệnh trong thẻ max để kiểm tra xem đâu là giá trị lớn nhất được lưu trong 3 thanh ghi a0, a1, a2: Bước đầu lưu giá trị của a0 vào s0 sau đó tính hiệu của a1 - a0 = t0, kiểm tra t0 nếu nhỏ hơn 0 thì tiếp tục so sánh s0( đang có giá trị là s0) với a2 bằng cách tương tự, nếu lớn hơn 0 thì thay đổi s0 = a1 và tiếp tục thực hiện chương trình.
- + Sau các câu lệnh trong thẻ max quay lại và kết thúc chương trình.

## - Quan sát giá trị các thanh ghi của chương trình:



The screenshot shows the QEMU debugger interface with several windows open:

- Registers**: Shows a table of registers with their names, numbers, and values.
- Floating Point**: Shows floating-point registers.
- Control and Status**: Shows control and status registers.
- Basic**: Shows assembly code with comments and addresses.
- Source**: Shows the source code corresponding to the assembly code.
- Memory**: Shows memory dump tables for address ranges 0x00000000-0x0000000f and 0x00000010-0x0000001f.
- Registers**: Shows the same register table as the top Registers window.
- Hexadecimal Addresses**: A checkbox for viewing memory dump in hex.
- Hexadecimal Values**: A checked checkbox for viewing memory dump in hex.
- ASCII**: An unchecked checkbox for viewing memory dump in ASCII.

Code	Basic	Source
0x00400513add1 x10,x0,4	4: li a0, 4	# Nap giá trị kiểm tra đếm
0x00700593add1 x11,x0,7	5: li a1, 7	# Nap giá trị a1 = 7
0x00500613add1 x12,x0,5	6: li a2, 5	# Nap giá trị a2 = 5
0x00c00efjal x1,0x000000c	7: jal max	# Gọi thê tuc max de tim v...
0x00a00893add1 x17,x0,10	8: li a7, 10	# Kêt thuc chuong trinh
0x00000673add1 x11	9: ecall	
0x00050433add1 x8,x10,x0	10: add a0, a0, zero	# Sao chép giá tr..
0x0056213add1 x5,x11,x8	11: sub a0, a1, a0	# Tinh a1 - a0
0x0002c463bit x8,x10,0x00000008	12: blt t0, zero, okay	# Nếu a1 - a0 < 0 ..
0x00058413add1 x8,x11,x0	13: add a0, a1, zero	# Ngược lại, crip ..

Code	Basic	Source
0x000400513	addi x10,x0,4	4: li a0, 4 # Nap giá trị kiêm tra đât..
0x00700592	addi x11,x0,7	5: li a1, 7 # Nap giá trị a1 - 7
0x00500613	addi x12,x0,5	6: li a2, 5 # Nap giá trị a2 = 5
0x0000000e	mul x1,x0,00000000	7: mul max # Gọi thử lops: max dà..
0x00000089	addi x17,x0,10	8: li a7, 10 # Kết thúc chương trình
0x00000073	ecall	9: ecall
0x00000433	add x9,x10,x0	10: add s0, a0, zero # Sao chép giá tr..
0x004502b3	sub x5,t0,x1,x6	11: sub t0, a1, a0 # Tính a1 - a0
0x00002462	bit x5,x0,00000008	12: bit t0, zero, okay # Nếu a1 - a0 < 0 ..
0x00005433	add x8,x11,x0	13: add s0, a1, zero # Ngược lại, cắp ..

Name	Number	Value
zero	0	0x00000000
t <sub>3</sub>	1	0x00000010
t <sub>2</sub>	2	0x7ffffeffe
t <sub>3</sub>	3	0x10000000
t <sub>4</sub>	4	0x00000009
t <sub>0</sub>	5	0x00000003
t <sub>1</sub>	6	0x00000000
t <sub>2</sub>	7	0x00000000
s <sub>0</sub>	8	0x0000000004
s <sub>1</sub>	9	0x00000000
a <sub>0</sub>	10	0x00000004
a <sub>1</sub>	11	0x00000007
a <sub>2</sub>	12	0x00000005
n <sub>3</sub>	13	0x00000000
a <sub>4</sub>	14	0x00000000
a <sub>5</sub>	15	0x00000000
a <sub>6</sub>	16	0x00000000
u <sub>7</sub>	17	0x00000000
s <sub>2</sub>	18	0x00000000
s <sub>3</sub>	19	0x00000000
s <sub>4</sub>	20	0x00000000
s <sub>5</sub>	21	0x00000000
s <sub>6</sub>	22	0x00000000
s <sub>7</sub>	23	0x00000000
n <sub>8</sub>	24	0x00000000
s <sub>9</sub>	25	0x00000000
s <sub>10</sub>	26	0x00000000
s <sub>11</sub>	27	0x00000000
t <sub>3</sub>	28	0x00000000
t <sub>4</sub>	29	0x00000000
t <sub>5</sub>	30	0x00000000
t <sub>6</sub>	31	0x00000000
pe		0x00000020

Code	Basic	Source
0x00700093	addi x11,x0,7	5: li al, 7 # Nhap gia tri al = 7
0x00506613	addi x12,x0,5	6: li a2, 5 # Nhap gia tri a2 = 5
0x00C00c0c	jal x1,0x0000000c	7: jal max # Gọi thuc tuc max do tim ..
0x00A00893	addi x17,x0,10	8: li a7, 10 # Ket thuc chuong trinh
0x00000073	call	9: call
0x00504333	add x8,x10,x0	10: add s0, a0, zero # Sap chep gia tri..
0x40B502b3	sub x1, x11, x8	11: sub t0, al, s0 # Tinh al - s0
0x002c463b	sub x8,x20,0x00000008	12: sub t0, zero, okay # Neu al - s0 < 0 ..
0x00504333	add x9,x11,x0	13: add s0, al, zero # Nuoc lai, cap ..
0x408602b3	sub x5, x12, x8	14: sub t0, a2, s0 # Tinh a2 - s0

	U1	0x0000000000000000
t0	1	0x0000000000000001
t2	2	0x7FFF000000000000
sp	3	0x1000000000000000
lr	4	0x0000000000000000
t0	5	0x0000000000000003
t1	6	0x0000000000000000
t2	7	0x0000000000000000
a0	8	0x0000000000000000
a1	9	0x0000000000000000
a0	10	0x0000000000000000
a1	11	0x0000000000000000
a2	12	0x0000000000000005
a3	13	0x0000000000000000
a0	14	0x0000000000000000
a5	15	0x0000000000000000
a6	16	0x0000000000000000
a7	17	0x0000000000000000
s2	18	0x0000000000000000
s3	19	0x0000000000000000
s4	20	0x0000000000000000
s5	21	0x0000000000000000
s6	22	0x0000000000000000
a7	23	0x0000000000000000
s8	24	0x0000000000000000
a9	25	0x0000000000000000
a10	26	0x0000000000000000
a11	27	0x0000000000000000
t3	28	0x0000000000000000
t4	29	0x0000000000000000
t5	30	0x0000000000000000
t6	31	0x0000000000000000
pc		0x0000000000000026

Code	Basic	Source
0x00000693addl x17,x0,10	8: addl a7, 10	# Kép giá trị
0x00000073 ecml	9: ecml	
0x00005043addl x8,x10,x0	13: addl a8, a10, zero	# Sao chép giá trị
0x048522b3sub x5,x11,x8	14: subl t0, a11, a8	# Tìm a1 - a8
0x0002c463b1ll x5,x0,0x00000008	15: blt t0, zero, okay	# Nếu a1 - a8 < 0 ..
0x00058A43addl x8,x11,x0	16: addl a8, a11, zero	# Ngược lại, cùp ..
0x000602b3sub x5,x12,x8	17: subl t0, a12, a8	# Ngược lại, cùp ..
0x0002c463b1ll x5,x0,0x00000008	18: blt t0, zero, done	# Nếu a2 - a0 < 0 ..
0x00060433add x8,x12,x0	19: addl a8, a12, zero	# Ngược lại, cùp ..
0x00000067lrl x0,x1,x0	20: lrl r0, a1, zero	
	21: lrl r0, a2, zero	
	22: lrl r0, a3, zero	
	23: lrl r0, a4, zero	# Quay lại chương ..

Registers	Floating Point	Control and Status
Name	Number	Value
zero	0	0x00000000
r <sub>0</sub>	1	0x00400010
r <sub>1</sub>	2	0xfffffffffc
r <sub>2</sub>	3	0x10000000
r <sub>3</sub>	4	0x00000000
r <sub>4</sub>	5	0xfffffffffc
r <sub>5</sub>	6	0x00000000
r <sub>6</sub>	7	0x00000000
r <sub>7</sub>	8	0x00000007
r <sub>8</sub>	9	0x00000000
r <sub>9</sub>	10	0x00000004
r <sub>10</sub>	11	0x00000007
r <sub>11</sub>	12	0x00000005
r <sub>12</sub>	13	0x00000000
r <sub>13</sub>	14	0x00000000
r <sub>14</sub>	15	0x00000000
r <sub>15</sub>	16	0x00000000
r <sub>16</sub>	17	0x00000000
r <sub>17</sub>	18	0x00000000
r <sub>18</sub>	19	0x00000000
r <sub>19</sub>	20	0x00000000
r <sub>20</sub>	21	0x00000000
r <sub>21</sub>	22	0x00000000
r <sub>22</sub>	23	0x00000000
r <sub>23</sub>	24	0x00000000
r <sub>24</sub>	25	0x00000000
r <sub>25</sub>	26	0x00000000
r <sub>26</sub>	27	0x00000000
r <sub>27</sub>	28	0x00000000
r <sub>28</sub>	29	0x00000000
r <sub>29</sub>	30	0x00000000
r <sub>30</sub>	31	0x00000000
pc		0x00000008

No	Vị trí	a0	a1	a2	s0	ra	t0	pc	Chú thích
1	(3)	0x00000	0x00000	0x00000	0x00000	0x00000	0x00000	0x400000	Bắt đầu chương trình
2	(7)	0x00004	0x00007	0x00005	0x00000	0x400010	0x00000	0x400018	Khởi tạo các giá trị
3	(17)	0x00004	0x00007	0x00005	0x00007	0x400010	0x00003	0x400028	So sánh a0 và a1
4	(23)	0x00004	0x00007	0x00005	0x00007	0x400010	0xfffffffffe	0x400010	So sánh max(a0,a1), a2
5	(10)	0x00004	0x00007	0x00005	0x00007	0x400010	0xfffffffffe	0x40002c	Kết thúc chương trình

⇒ Kết quả của thanh ghi  $a0 = 4$ ,  $a1 = 7$ ,  $a2 = 5$  và  $s0 = 7$  nên chương trình hoạt động bình thường.

#### - Thay giá trị tham số

+ Với  $a_0 = 1$ ,  $a_1 = 18$ ,  $a_2 = 36$  quan sát kết quả:

The screenshot shows the QEMU debugger interface with the following details:

**Registers**

Name	Number	Value
zero	0	0x00000000
ra	1	0x00400010
sp	2	0x7ffffeffc
gp	3	0x10000000
tp	4	0x00000000
t0	5	0x000000012
t1	6	0x00000000
t2	7	0x00000000
a0	8	0x000000024
s1	9	0x00000000
a0	10	0x000000001
a1	11	0x0000000012
a2	12	0x0000000024
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000a
a2	18	0x00000000
a3	19	0x00000000
s4	20	0x00000000
a5	21	0x00000000
a6	22	0x00000000
a7	23	0x00000000
a8	24	0x00000000
a9	25	0x00000000
a10	26	0x00000000
a11	27	0x00000000
b3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x00400018

**Floating Point**

**Control and Status**

**Line: 5 Column: 18  Show Line Numbers**

**Messages**

Reset: reset completed.

**Run IO**

-- program is finished running (0) --

⇒ Kết quả của thanh ghi  $a_0 = 1$ ,  $a_1 = 18$ ,  $a_2 = 36$  và  $s_0 = 36$  nên chương trình hoạt động bình thường.

### 3. Assignment 3

#### - Nhập chương trình:

Lab07\_1.asm Lab07\_2.asm riscv3.asm\*

```
1 # Laboratory Exercise 7, Home Assignment 3
2 .text
3 main:
4     li      s0, 21          # Gán giá trị 21 vào thanh ghi s0
5     li      s1, 5           # Gán giá trị 5 vào thanh ghi s1
6 push:
7     addi   sp, sp, -8       # Điều chỉnh con trỏ ngăn xếp (giảm 8 byte)
8     sw    s0, 4(sp)        # Đẩy giá trị s0 vào ngăn xếp
9     sw    s1, 0(sp)        # Đẩy giá trị s1 vào ngăn xếp
10 work:
11    nop                  # Không làm gì (lệnh chờ)
12    nop
13    nop
14 pop:
15    lw    s0, 0(sp)        # Lấy giá trị từ ngăn xếp về s0
16    sw    zero, 0(sp)      # Xóa dữ liệu khỏi ngăn xếp (gán 0)
17    addi   sp, sp, 4        # Điều chỉnh con trỏ ngăn xếp (tăng 4 byte)
18    lw    s1, 0(sp)        # Lấy giá trị từ ngăn xếp về s1
19    sw    zero, 0(sp)      # Xóa dữ liệu khỏi ngăn xếp (gán 0)
20    addi   sp, sp, 4        # Điều chỉnh con trỏ ngăn xếp (tăng 4 byte)
21 end:
```

- **Các bước hoạt động của chương trình:**
    - + Thẻ main để khởi tạo giá trị của s0, s1
    - + Câu lệnh trong thẻ push để thực hiện đưa giá trị vào stack: Đầu tiên khởi tạo vùng nhớ stack cho s0, s1 thông qua sp (stack pointer). Sau đó lần lượt đẩy giá trị của s0, s1 vào vùng nhớ stack
    - + Câu lệnh trong thẻ work để mô phỏng vùng hoạt động của chương trình
    - + Câu lệnh trong thẻ pop để lấy giá trị ra khỏi stack: Đầu tiên lấy giá trị của s1 ra khỏi vùng nhớ rồi lưu vào s0, sau đó xóa giá trị đã lấy và điều chỉnh lại giá trị của sp. Tương tự, lấy giá trị của s0 cũ đã lưu trong stack và lưu vào s1, sau đó xóa giá trị đã lưu cho stack và trả lại giá trị của sp

- Quan sát giá trị các thanh ghi trong chương trình:

Name	Number	Value
r24	9	0x00000000
ra	1	0x00000000
sp	2	0xffffffff
rp	3	0x00000000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
t3	8	0x00000000
t4	9	0x00000000
t5	10	0x00000000
t6	11	0x00000000
t7	12	0x00000000
t8	13	0x00000000
t9	14	0x00000000
t10	15	0x00000000
t11	16	0x00000000
t12	17	0x00000000
t13	18	0x00000000
t14	19	0x00000000
t15	20	0x00000000
t16	21	0x00000000
t17	22	0x00000000
t18	23	0x00000000
t19	24	0x00000000
t20	25	0x00000000
t21	26	0x00000000
t22	27	0x00000000
t23	28	0x00000000
t24	29	0x00000000
t25	30	0x00000000
t26	31	0x00000000
pc	32	0x04030000

Name	Number	Value
r24	0	0x00000000
ra	1	0x00000000
sp	2	0xffffffff
rp	3	0x10000000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
t3	8	0x00000000
t4	9	0x00000000
t5	10	0x00000000
t6	11	0x00000000
t7	12	0x00000000
t8	13	0x00000000
t9	14	0x00000000
t10	15	0x00000000
t11	16	0x00000000
t12	17	0x00000000
t13	18	0x00000000
t14	19	0x00000000
t15	20	0x00000000
t16	21	0x00000000
t17	22	0x00000000
t18	23	0x00000000
t19	24	0x00000000
t20	25	0x00000000
t21	26	0x00000000
t22	27	0x00000000
t23	28	0x00000000
t24	29	0x00000000
t25	30	0x00000000
t26	31	0x00000000
pc	32	0x04030000

Name	Number	Value
r24	0	0x00000000
ra	1	0x00000000
sp	2	0xffffffff
rp	3	0x10000000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
t3	8	0x00000000
t4	9	0x00000000
t5	10	0x00000000
t6	11	0x00000000
t7	12	0x00000000
t8	13	0x00000000
t9	14	0x00000000
t10	15	0x00000000
t11	16	0x00000000
t12	17	0x00000000
t13	18	0x00000000
t14	19	0x00000000
t15	20	0x00000000
t16	21	0x00000000
t17	22	0x00000000
t18	23	0x00000000
t19	24	0x00000000
t20	25	0x00000000
t21	26	0x00000000
t22	27	0x00000000
t23	28	0x00000000
t24	29	0x00000000
t25	30	0x00000000
t26	31	0x00000000
pc	32	0x04030000

The screenshot shows the WinDbg debugger interface with several windows open:

- Registers**: Shows CPU registers (eax, ebx, ecx, edx, etc.) and floating-point registers (st0, st1, st2) with their names, numbers, and values.
- Floating Point**: Shows the floating-point stack with values for st0 through st15.
- Control and Status**: Shows control and status flags like zero, sign, and overflow.
- Code**: Shows assembly code with columns for Address, Instruction, Registers, Stack, and Source. The instruction at address 0x00012023 is highlighted.
- Basic**: Shows the assembly code again, likely a simplified view of the Code window.
- Source**: Shows the source code corresponding to the assembly instructions.
- Registers**: Shows the CPU registers again.
- Stack**: Shows the stack contents.
- Call Stack**: Shows the call stack.
- Registers**: Shows the CPU registers again.
- Registers**: Shows the CPU registers again.

No	Vị trí	s1	s2	sp	pc	Chú thích
1	(3)	0x00000	0x00000	0x7ffffeffc	0x400000	Bắt đầu chương trình
2	(6)	0x00015	0x00005	0x7ffffeffc	0x400008	Khởi tạo giá trị ban đầu
3	(10)	0x00015	0x00005	0x7ffffeff4	0x400014	Đưa lần lượt s1, s2 vào stack
4	(17)	0x00005	0x00005	0x7ffffeff8	0x40002c	Lấy giá trị của s2 ra khỏi stack
5	(21)	0x00005	0x00015	0x7ffffeffc	0x0040003c	Kết thúc chương trình

- Quan sát kết quả được lưu ở của sổ DataSegment:

+ Giá trị ban đầu khi mới khởi tạo giá trị:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0xfffffe0	0	0	0	0	0	0	21	0
0xfffff000	0	0	0	0	0	0	0	0
0xfffff020	0	0	0	0	0	0	0	0
0xfffff040	0	0	0	0	0	0	0	0
0xfffff060	0	0	0	0	0	0	0	0
0xfffff080	0	0	0	0	0	0	0	0
0xfffff0a0	0	0	0	0	0	0	0	0
0xfffff0c0	0	0	0	0	0	0	0	0
0xfffff0e0	0	0	0	0	0	0	0	0
0xfffff100	0	0	0	0	0	0	0	0
0xfffff120	0	0	0	0	0	0	0	0
0xfffff140	0	0	0	0	0	0	0	0
0xfffff160	0	0	0	0	0	0	0	0
0xfffff180	0	0	0	0	0	0	0	0

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0xfffffe0	0	0	0	0	0	5	21	0
0xfffff000	0	0	0	0	0	0	0	0
0xfffff020	0	0	0	0	0	0	0	0
0xfffff040	0	0	0	0	0	0	0	0
0xfffff060	0	0	0	0	0	0	0	0
0xfffff080	0	0	0	0	0	0	0	0
0xfffff0a0	0	0	0	0	0	0	0	0
0xfffff0c0	0	0	0	0	0	0	0	0
0xfffff0e0	0	0	0	0	0	0	0	0
0xfffff100	0	0	0	0	0	0	0	0
0xfffff120	0	0	0	0	0	0	0	0
0xfffff140	0	0	0	0	0	0	0	0
0xfffff160	0	0	0	0	0	0	0	0
0xfffff180	0	0	0	0	0	0	0	0

+ Giá trị sau từng bước pop ra ngoài:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0xfffffe0	0	0	0	0	0	0	21	0
0xfffff000	0	0	0	0	0	0	0	0
0xfffff020	0	0	0	0	0	0	0	0
0xfffff040	0	0	0	0	0	0	0	0
0xfffff060	0	0	0	0	0	0	0	0
0xfffff080	0	0	0	0	0	0	0	0
0xfffff0a0	0	0	0	0	0	0	0	0
0xfffff0c0	0	0	0	0	0	0	0	0
0xfffff0e0	0	0	0	0	0	0	0	0
0xfffff100	0	0	0	0	0	0	0	0
0xfffff120	0	0	0	0	0	0	0	0
0xfffff140	0	0	0	0	0	0	0	0
0xfffff160	0	0	0	0	0	0	0	0
0xfffff180	0	0	0	0	0	0	0	0

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0xfffffe0	0	0	0	0	0	0	0	0
0xfffff000	0	0	0	0	0	0	0	0
0xfffff020	0	0	0	0	0	0	0	0
0xfffff040	0	0	0	0	0	0	0	0
0xfffff060	0	0	0	0	0	0	0	0
0xfffff080	0	0	0	0	0	0	0	0
0xfffff0a0	0	0	0	0	0	0	0	0
0xfffff0c0	0	0	0	0	0	0	0	0
0xfffff0e0	0	0	0	0	0	0	0	0
0xfffff100	0	0	0	0	0	0	0	0
0xfffff120	0	0	0	0	0	0	0	0
0xfffff140	0	0	0	0	0	0	0	0
0xfffff160	0	0	0	0	0	0	0	0
0xfffff180	0	0	0	0	0	0	0	0

- ⇒ Kết quả của chương trình hoạt động đúng với yêu cầu
- Thay đổi giá trị tham số s1 = 18, s2 = 36

The screenshot shows a debugger interface with several windows:

- Assembly Window:** Displays assembly code for Lab07\_3.asm. The code includes comments in Vietnamese explaining operations like copying values from memory to registers and vice versa.
- Registers Window:** Shows the state of various registers (r0 to r31, pc) with their corresponding memory addresses and values.
- Floating Point Window:** Shows floating-point register values.
- Control and Status Window:** Shows system status information.
- Messages Window:** Displays messages indicating the program has finished running.

⇒ Kết quả của thanh ghi  $a0 = 18, a1 = 36$  và đầu ra như kỳ vọng nên chương trình hoạt động bình thường.

#### 4. Assignment 4

##### - Nhập chương trình:

```
# Laboratory Exercise 7, Home Assignment 4
```

```
.data
```

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

```
.text
```

```
main:
```

```
jal WARP
```

```
print:
```

```
add a1, s0, zero # a1 = result từ giải thừa
```

```
li a7, 4
```

```
la a0, message
```

```
ecall
```

```
    li    a7, 1  
    mv    a0, s0  
    ecall
```

quit:

```
    li    a7, 10      # Thoát chương trình  
    ecall
```

end\_main:

WARP:

```
    addi   sp, sp, -4    # Điều chỉnh con trỏ stack  
    sw     ra, 0(sp)    # Lưu địa chỉ trả về vào stack
```

```
    li    a0, 3      # Gán giá trị test N = 3
```

```
    jal   FACT       # Gọi hàm FACT
```

```
    lw    ra, 0(sp)    # Phục hồi địa chỉ trả về  
    addi   sp, sp, 4    # Điều chỉnh lại con trỏ stack  
    jr    ra
```

wrap\_end:

FACT:

```
    addi   sp, sp, -8    # Cấp phát stack cho ra và a0  
    sw     ra, 4(sp)    # Lưu thanh ghi ra
```

```
sw    a0, 0(sp)    # Lưu giá trị N hiện tại

li    t0, 2

bge   a0, t0, recursive # Nếu N >= 2, tiếp tục đệ quy

li    s0, 1        # Nếu N < 2, trả về 1

j     done         # Nhảy đến kết thúc
```

recursive:

```
addi  a0, a0, -1    # Giảm N xuống (N-1)

jal   FACT          # Gọi đệ quy FACT(N-1)
```

```
lw    s1, 0(sp)    # Lấy lại giá trị N từ stack

mul  s0, s0, s1    # s0 = FACT(N-1) * N
```

done:

```
lw    ra, 4(sp)    # Phục hồi thanh ghi ra

lw    a0, 0(sp)    # Phục hồi giá trị a0

addi sp, sp, 8    # Giải phóng stack

jr    ra           # Trả về caller
```

fact\_end:

- **Chạy chương trình:**

The screenshot shows a debugger interface with several windows:

- Assembly Window:** Displays the assembly code for Lab07\_4.asm. The code implements a recursive factorial function. It includes sections for .data, .text, main, print, quit, and WARP. It uses instructions like add, li, la, ecall, jal, and sw.
- Registers Window:** Shows the state of various registers. Key values include ra (1), sp (2), a0 (3), t0 (2), and pc (0x0040002c).
- Floating Point Window:** Not visible in the screenshot.
- Control and Status Window:** Not visible in the screenshot.
- Messages Window:** Displays the output of the program: "Ket qua tinh giai thua la: 6".

- Với n=3 => kết quả đưa ra đúng với kì vọng
- **Các bước hoạt động của chương trình:**
  - + Câu lệnh trong cụm data để khởi tạo giá trị cho chuỗi để in ra màn hình
  - + Câu lệnh trong thẻ main để gọi đến hàm tính lũa thừa
  - + Câu lệnh trong thẻ print để sau khi chương trình đã được thực hiện thì in kết quả ra màn hình và kết thúc chương trình
  - + Câu lệnh trong thẻ quit để kết thúc chương trình
  - + Câu lệnh trong thẻ WARP chuẩn bị cho bước gọi hàm giai thừa: Trước tiên lưu giá trị của ra vào stack, sau đó khởi tạo giá trị của a0 = 3 là số giai thừa, tiếp tục gọi hàm giai thừa để thực hiện tính giai thừa, cuối cùng là lấy giá trị đã được lưu của ra trong stack để quay lại main tiếp tục chương trình.
  - + Câu lệnh trong thẻ FACT để tính đệ quy: Cấp phát thêm 8 byte bộ nhớ trong stack, lưu địa chỉ của ra vào stack, sau đó lưu địa chỉ a0 vào stack, tiếp tục khởi tạo giá trị t0 = 2. Kiểm tra xem nếu  $n \geq 2$  thì tiếp tục thực hiện đệ quy, nếu  $n < 2$  thì trả về giá trị 1 và nhảy đến thẻ done
  - + Câu lệnh trong thẻ recursive để gọi đệ quy: Giảm giá trị của a0 sau mỗi lần đệ quy để gọi đệ quy của  $n - 1$  sau đó lấy lại giá trị ban đầu, rồi tính tích của hàm đệ quy .
  - + Câu lệnh trong thẻ done để hoàn thành chương trình và quay trở lại main: Khôi phục giá trị của con trỏ sau đó quay lại caller
- **Quan sát giá trị các thanh ghi trong chương trình:**

Registers	Floating Point	Control and Status
Name	Number	Value
zero	0	0x00000000
r <u>u</u>	1	0x00000000
r <u>b</u>	2	0x7fffffe000
r <u>D</u>	3	0x1000000000
r <u>p</u>	4	0x00000000
r <u>0</u>	5	0x00000000
r <u>1</u>	6	0x00000000
r <u>2</u>	7	0x00000000
r <u>0</u>	8	0x00000000
r <u>1</u>	9	0x00000000
r <u>0</u>	10	0x00000000
r <u>1</u>	11	0x00000000
r <u>2</u>	12	0x00000000
r <u>3</u>	13	0x00000000
r <u>4</u>	14	0x00000000
r <u>5</u>	15	0x00000000
r <u>6</u>	16	0x00000000
r <u>7</u>	17	0x00000000
r <u>8</u>	18	0x00000000
r <u>9</u>	19	0x00000000
r <u>4</u>	20	0x00000000
r <u>5</u>	21	0x00000000
r <u>6</u>	22	0x00000000
r <u>7</u>	23	0x00000000
r <u>8</u>	24	0x00000000
r <u>9</u>	25	0x00000000
r <u>10</u>	26	0x00000000
r <u>11</u>	27	0x00000000
r <u>3</u>	28	0x00000000
r <u>4</u>	29	0x00000000
r <u>5</u>	30	0x00000000
r <u>6</u>	31	0x00000000
pc		0x00000000

Registers	Floating Point	Control and Status
Name	Number	Value
zero	0	0x0000000000000000
ru	1	0x0000000000000001
sp	2	0x1f11fe1fe1fe1fe1f
tp	3	0x1000000000000000
fp	4	0x0000000000000000
t0	5	0x0000000000000000
t1	6	0x0000000000000000
t2	7	0x0000000000000000
u0	8	0x0000000000000000
s1	9	0x0000000000000000
a0	10	0x0000000000000000
a1	11	0x0000000000000000
a2	12	0x0000000000000000
a3	13	0x0000000000000000
a4	14	0x0000000000000000
a5	15	0x0000000000000000
a6	16	0x0000000000000000
a7	17	0x0000000000000000
s2	18	0x0000000000000000
n3	19	0x0000000000000000
n4	20	0x0000000000000000
e5	21	0x0000000000000000
s6	22	0x0000000000000000
u7	23	0x0000000000000000
s0	24	0x0000000000000000
s4	25	0x0000000000000000
a10	26	0x0000000000000000
s11	27	0x0000000000000000
t3	28	0x0000000000000000
t4	29	0x0000000000000000
t5	30	0x0000000000000000
t6	31	0x0000000000000000
pc		0x000000000000002c

Registers	Floating Point	Control and Status
Name	Number	Value
zero	0	0x00000000
z1	1	0x00000004
z2	2	0x7fffffe#F
z3	3	0x10000000
tzp	4	0x00000009
t1	5	0x00000000
t3	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000
s0	10	0x00000003
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s0	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s0	24	0x00000000
s4	25	0x00000000
s0	26	0x00000000
s1	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
ps2		0x00000000

**Registers** **Floating Point** **Control and Status**

Name	Number	Value
zero	0	0x00000000
ra	1	0x0040003c
sp	2	0x7ffffeff8
gp	3	0x10000000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
a1	9	0x00000000
a0	10	0x00000003
u1	11	0x00000000
a2	12	0x00000000
u3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
a3	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x00400048

**Code** **Basic** **Source**

124	0x00a00093 addi x17,x0,10	18: li a7, 10 # Thoát chương trình
128	0x00000003 ocall	19: ocall
129	0xffff1013 addi x2,x2,0xffffffff	23: addi sp, sp, -4 # Điều chỉnh con trỏ
130	0x00110203 sw x2,0(x2)	24: sw ra, 0(sp) ! Lưu địa chỉ trả về
134	0x00300513 addi x10,x0,3	26: li a0, 3 # Gán giá trị trả về
138	0x010000ef jal x1,0x00000010	27: jal FACT # Gọi hàm FACT
139	0x00012083 lw x1,0(x2)	29: lw ra, 0(sp) # Phục hồi địa chỉ
140	0x00410113 addi x2,x2,4	30: addi sp, sp, 4 # Điều chỉnh lại con trỏ
141	0x00000067 jalr x0,x1,0	31: jr ra
148	0xffff0113 addi x2,x2,0xffffffff	35: addi sp, sp, -8 # Cấp phát stack

**Registers** **Floating Point** **Control and Status**

Name	Number	Value
zero	0	0x00000000
ra	1	0x0040003c
sp	2	0x7ffffeff8
gp	3	0x10000000
tp	4	0x00000000
t0	5	0x00000002
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
a1	9	0x00000003
a0	10	0x00000000
a2	11	0x00000000
a3	12	0x00000000
a4	13	0x00000000
a5	14	0x00000000
a6	15	0x00000000
a7	16	0x00000000
s2	17	0x00000000
s3	18	0x00000000
a3	19	0x00000000
a4	20	0x00000000
a5	21	0x00000000
a6	22	0x00000000
a7	23	0x00000000
a8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x00400048

**Code** **Basic** **Source**

18	0x00300513 addi x10,x0,3	26: li a0, 3 # Gán giá trị trả về
18	0x010000ef jal x1,0x00000010	27: jal FACT # Gọi hàm FACT
19	0x00012083 lw x1,0(x2)	29: lw ra, 0(sp) # Phục hồi địa chỉ
19	0x00410113 addi x2,x2,4	30: addi sp, sp, 4 # Điều chỉnh lại con trỏ
19	0x00000067 jalr x0,x1,0	31: jr ra
18	0xffff0113 addi x2,x2,0xffffffff	35: addi sp, sp, -8 # Cấp phát stack
19	0x00112223 sw x1,4(x2)	36: sw ra, 4(sp) ! Lưu thành ghi ra...
19	0x00a12023 sw x1,0(x2)	37: sw a0, 0(sp) ! Lưu giá trị N...
19	0x00200293 addi x1,x0,2	39: li t0, 2
19	0x0055663 bge x10,x5,0x0000000c	40: bge a0, t0, recursive # Nếu N >= 2, ..
19	0x00100113 addi x1,x0,1	41: li a0, 1 # Nếu N < 2, trả về
19	0x014000ef jal x1,0x00000014	42: jal done # Nhận đến kết quả
19	0xffff0513 addi x10,x10,0xffffffff	45: addi a0, a0, -1 # Giảm N xuống (N-1)
19	0xffff00ef jal x1,0xffffffff	46: jal FACT # Gọi đệ quy FACT.

**Registers** **Floating Point** **Control and Status**

Name	Number	Value
zero	0	0x00000000
ra	1	0x0040003c
sp	2	0x7ffffeff0
gp	3	0x10000000
tp	4	0x00000000
t0	5	0x00000002
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
a1	9	0x00000003
a0	10	0x00000000
a2	11	0x00000000
a3	12	0x00000000
a4	13	0x00000000
a5	14	0x00000000
a6	15	0x00000000
a7	16	0x00000000
s2	17	0x00000000
s3	18	0x00000000
a3	19	0x00000000
a4	20	0x00000000
a5	21	0x00000000
a6	22	0x00000000
a7	23	0x00000000
a8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x00400048

**Code** **Basic** **Source**

19	0x00000067 jalr x0,x1,0	31: jr ra
19	0xffff1013 addi x2,x2,0xffffffff	35: addi sp, sp, -8 # Cấp phát stack
19	0x00112223 sw x1,4(x2)	36: sw ra, 4(sp) ! Lưu thành ghi ra...
19	0x00a12023 sw x1,0(x2)	37: sw a0, 0(sp) ! Lưu giá trị N...
19	0x00200293 addi x1,x0,2	39: li t0, 2
19	0x0055663 bge x10,x5,0x0000000c	40: bge a0, t0, recursive # Nếu N >= 2, ..
19	0x00100113 addi x1,x0,1	41: li a0, 1 # Nếu N < 2, trả về
19	0x014000ef jal x1,0x00000014	42: jal done # Nhận đến kết quả
19	0xffff0513 addi x10,x10,0xffffffff	45: addi a0, a0, -1 # Giảm N xuống (N-1)
19	0xffff00ef jal x1,0xffffffff	46: jal FACT # Gọi đệ quy FACT.

**Registers** **Floating Point** **Control and Status**

Name	Number	Value
zero	0	0x00000000
ra	1	0x0040003c
sp	2	0x7ffffeff0
gp	3	0x10000000
tp	4	0x00000000
t0	5	0x00000002
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
a1	9	0x00000003
a0	10	0x00000000
a2	11	0x00000000
a3	12	0x00000000
a4	13	0x00000000
a5	14	0x00000000
a6	15	0x00000000
a7	16	0x00000000
s2	17	0x00000000
s3	18	0x00000000
a3	19	0x00000000
a4	20	0x00000000
a5	21	0x00000000
a6	22	0x00000000
a7	23	0x00000000
a8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x00400048

**Code** **Basic** **Source**

19	0x00000067 jalr x0,x1,0	31: jr ra
19	0xffff1013 addi x2,x2,0xffffffff	35: addi sp, sp, -8 # Cấp phát stack
19	0x00112223 sw x1,4(x2)	36: sw ra, 4(sp) ! Lưu thành ghi ra...
19	0x00a12023 sw x1,0(x2)	37: sw a0, 0(sp) ! Lưu giá trị N...
19	0x00200293 addi x1,x0,2	39: li t0, 2
19	0x0055663 bge x10,x5,0x0000000c	40: bge a0, t0, recursive # Nếu N >= 2, ..
19	0x00100113 addi x1,x0,1	41: li a0, 1 # Nếu N < 2, trả về
19	0x014000ef jal x1,0x00000014	42: jal done # Nhận đến kết quả
19	0xffff0513 addi x10,x10,0xffffffff	45: addi a0, a0, -1 # Giảm N xuống (N-1)
19	0xffff00ef jal x1,0xffffffff	46: jal FACT # Gọi đệ quy FACT.

**Registers** **Floating Point** **Control and Status**

Name	Number	Value
zero	0	0x00000000
ra	1	0x0040003c
sp	2	0x7ffffeff0
gp	3	0x10000000
tp	4	0x00000000
t0	5	0x00000002
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
a1	9	0x00000003
a0	10	0x00000000
a2	11	0x00000000
a3	12	0x00000000
a4	13	0x00000000
a5	14	0x00000000
a6	15	0x00000000
a7	16	0x00000000
s2	17	0x00000000
s3	18	0x00000000
a3	19	0x00000000
a4	20	0x00000000
a5	21	0x00000000
a6	22	0x00000000
a7	23	0x00000000
a8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x00400048

Registers   Floating Point   Control and Status

Name	Number	Value
zero	0	0x00000000
ra	1	0x0400006c
sp	2	0xffffffff
gp	3	0x10000000
tp	4	0x00000000
t0	5	0x00000002
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000
a0	10	0x00000002
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x04000048

Name	Number	Value
zero	0	0x00000000
ra	1	0x0400006c
sp	2	0xffffffff
gp	3	0x10000000
tp	4	0x00000000
t0	5	0x00000002
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000001
s1	9	0x00000000
a0	10	0x00000001
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x0400007c

Name	Number	Value
zero	0	0x00000000
ra	1	0x0400006c
sp	2	0xffffffff
gp	3	0x10000000
tp	4	0x00000000
t0	5	0x00000002
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000006
s1	9	0x00000003
a0	10	0x00000003
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x04000040

Name	Number	Value
zero	0	0x00000000
ra	1	0x04000004
sp	2	0xfffffefffc
gp	3	0x10080000
tp	4	0x00000000
t0	5	0x00000002
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000006
s1	9	0x00000003
a0	10	0x00000006
a1	11	0x00000006
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x0000000a
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x0040002c

- Chú ý:

### Cách hoạt động:

Nếu  $N = 3$ , chương trình sẽ thực hiện:

- +  $\text{FACT}(3) = 3 * \text{FACT}(2)$
- +  $\text{FACT}(2) = 2 * \text{FACT}(1)$
- +  $\text{FACT}(1) = 1$  (**Điều kiện dừng**)

Sau đó, kết quả sẽ được tính ngược lại:

- +  $\text{FACT}(2) = 2 * 1 = 2$
- +  $\text{FACT}(3) = 3 * 2 = 6$

Vòng lặp này tiếp tục đến khi giá trị cuối cùng được tính xong.

⇒ Kết quả của thanh ghi  $s0 = 6$  với  $n = 3$  ( $1 \times 2 \times 3 = 6$ ) đầu ra như kỳ vọng nên chương trình hoạt động bình thường.

- Thay giá trị tham số  $a0 = 5$

The screenshot shows a debugger window with the following sections:

- Editor:** Displays assembly code for Lab07\_4.asm. The code includes instructions for printing results, quitting, and calculating factorials.
- Registers:** A table showing register values. Key values include zero (0), ra (1), sp (2), gp (3), tp (4), t0 (5), t1 (6), t2 (7), s0 (8), s1 (9), a0 (10), a1 (11), a2 (12), a3 (13), a4 (14), a5 (15), a6 (16), a7 (17), s2 (18), s3 (19), s4 (20), s5 (21), s6 (22), s7 (23), s8 (24), s9 (25), s10 (26), s11 (27), t3 (28), t4 (29), t5 (30), and t6 (31). The value for zero is 0x00000000.
- Messages:** A log of program output. It shows the program running and exiting with status 0, and then running again with a different input, exiting with status 120.

⇒ Kết quả của thanh ghi s0 = 120 với n = 5 ( $1 \times 2 \times 3 \times 4 \times 5 = 120$ ) đầu ra như kỳ vọng nên chương trình hoạt động bình thường.

## 5. Assignment 5

### Nhập chương trình:

```
# Laboratory Exercise 7, Home Assignment 5
```

```
.data
```

```
min_msg: .string "Smallest: "
```

```
at_msg: .string ", "
```

```
max_msg: .string "Largest: "
```

```
newline: .string "\n"
```

```
.text
```

```
main:
```

```
# Nạp 8 số nguyên vào các thanh ghi a0 - a7
```

```
li a0, 0
```

```
li a1, 2
```

```
li a2, 4
```

```
li a3, 6
```

```
li a4, 8
```

```
li a5, 1
```

```
li a6, 3
```

```
li a7, 5
```

```
# Đây các giá trị lén ngăn xếp
```

```
addi sp, sp, -32 # Dành chỗ trên stack
```

```
sw a0, 28(sp)
```

```
sw a1, 24(sp)
```

```
sw a2, 20(sp)
```

```
sw a3, 16(sp)
```

```
sw a4, 12(sp)
```

```
sw a5, 8(sp)
```

```
sw a6, 4(sp)
```

```
sw a7, 0(sp)
```

```
# Gọi hàm tìm min/max
```

```
jal fnd_minmax
```

# Hiển thị kết quả

la a0, min\_msg

li a7, 4

ecall

mv a0, s0

li a7, 1

ecall

la a0, at\_msg

li a7, 4

ecall

mv a0, s2

li a7, 1

ecall

la a0, newline

li a7, 4

ecall

la a0, max\_msg

li a7, 4

ecall

mv a0, s1

li a7, 1

ecall

la a0, at\_msg

```
li a7, 4
```

```
ecall
```

```
mv a0, s3
```

```
li a7, 1
```

```
ecall
```

```
# Kết thúc chương trình
```

```
li a7, 10
```

```
ecall
```

```
find_minmax:
```

```
# Khởi tạo giá trị ban đầu
```

```
lw s0, 28(sp) # Giá trị nhỏ nhất
```

```
lw s1, 28(sp) # Giá trị lớn nhất
```

```
li s2, 0      # Vị trí giá trị nhỏ nhất
```

```
li s3, 0      # Vị trí giá trị lớn nhất
```

```
li t0, 1      # Chỉ số hiện tại
```

```
# Vòng lặp kiểm tra từng phần tử
```

```
li t1, 24 # Offset bắt đầu
```

```
loop:
```

```
lw t2, (sp) # Lấy giá trị từ stack
```

```
bge t2, s0, not_min # Nếu giá trị >= min thì bỏ qua
```

```
mv s0, t2      # Cập nhật giá trị nhỏ nhất
```

```
mv s2, t0      # Cập nhật vị trí nhỏ nhất
```

not\_min:

```
ble t2, s1, not_max # Nếu giá trị <= max thì bỏ qua
```

```
mv s1, t2      # Cập nhật giá trị lớn nhất
```

```
mv s3, t0      # Cập nhật vị trí lớn nhất
```

not\_max:

```
addi sp, sp, 4 # Di chuyển đến phần tử tiếp theo
```

```
addi t0, t0, 1 # Tăng chỉ số
```

```
addi t1, t1, -4 # Giảm số phần tử cần duyệt
```

```
bnez t1, loop # Nếu còn phần tử, tiếp tục lặp
```

```
# Khôi phục stack pointer
```

```
addi sp, sp, -32
```

```
jr ra
```

- Chạy chương trình:

The screenshot shows a debugger interface with several windows:

- Registers:** A table showing register values from zero to 31. Most registers contain 0x00000000 except for:
  - t0: 1
  - t2: 2
  - tE: 3
  - tP: 4
  - t0: 5
  - t1: 6
  - t2: 7
  - s0: 8
  - s1: 9
  - a0: 10
  - a1: 11
  - a2: 12
  - a3: 13
  - a4: 14
  - a5: 15
  - a6: 16
  - a7: 17
  - s2: 18
  - s3: 19
  - s4: 20
  - s5: 21
  - s6: 22
  - s7: 23
  - s8: 24
  - s9: 25
  - s10: 26
  - s11: 27
  - t3: 28
  - t4: 29
  - t5: 30
  - t6: 31
- Floating Point:** An empty table.
- Control and Status:** An empty table.
- Messages:** Displays the output of the program:

```
-- program 18 finished running (0) --
Smallest: 0, 0
Largest: 8, 4
-- program 19 finished running (0) --
```

- Các bước thực hiện chương trình:

### - **Khởi tạo danh sách số nguyên**

- + Gán các giá trị 0, 2, 4, 6, 8, 1, 3, 5 vào các thanh ghi từ a0 đến a7.
- + Lưu các giá trị này vào ngăn xếp để bảo toàn dữ liệu.

### - **Thiết lập các biến min, max, index**

- + Khởi tạo s0 = a0 làm giá trị nhỏ nhất (min).
- + Khởi tạo s1 = a0 làm giá trị lớn nhất (max).
- + Đặt s2 = 0 làm chỉ số của giá trị nhỏ nhất.
- + Đặt s3 = 0 làm chỉ số của giá trị lớn nhất.
- + Đặt s4 = 1 làm chỉ số để duyệt qua các phần tử tiếp theo.

### - **Duyệt qua danh sách để tìm min/max**

- + So sánh từng giá trị a1 → a7 với s0 (min) và s1 (max).
- + Nếu giá trị hiện tại nhỏ hơn s0, cập nhật s0 và s2.
- + Nếu giá trị hiện tại lớn hơn s1, cập nhật s1 và s3.
- + Tăng biến đếm s4 sau mỗi lần kiểm tra.

### - **Điều chỉnh chỉ số về giá trị thực tế**

- + Vì chỉ số bắt đầu từ 0, ta cộng thêm 1 cho s2 và s3 để hiển thị chính xác.

### - **Xuất kết quả**

- + In chuỗi "min = ", sau đó in giá trị s0 và vị trí s2.
- + In chuỗi "max = ", sau đó in giá trị s1 và vị trí s3.

### - **Kết thúc chương trình**

- + Khôi phục thanh ghi ra.
- + Gọi syscall 10 để thoát chương trình.

## **Bài tập không yêu cầu sự tham đổi của các thanh ghi**