Bài thực hành số 7

Lớp: 139365 – Học phần: Thực Hành Kiến Trúc Máy Tính

Đào Minh Nhật – 20215107

Assignment 1:

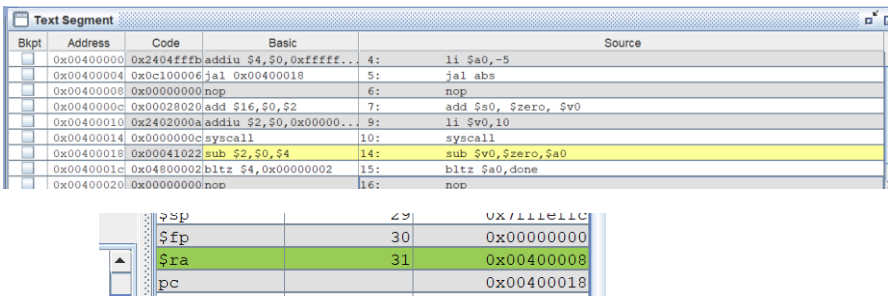
Code:

```
1  #Lab7, Assignment 1
2  .text
3  main:
4          li $a0,-5
5          jal abs
6          nop
7          add $s0, $zero, $v0
8
9          li $v0,10
10         syscall
11 endmain:
12
13 abs:
14         sub $v0,$zero,$a0
15         bltz $a0,done
16         nop
17         add $v0,$a0,$zero
18 done:
19         jr $ra
```

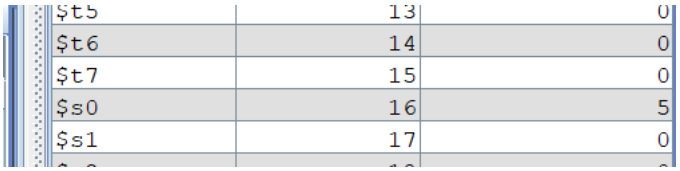
Giải thích:

Dòng 4: a0 là giá trị đầu vào

Dòng 5 : Thực hiện gọi tới hàm abs và đồng thời lưu trữ địa chỉ của lệnh tiếp theo (dòng số 6) vào thanh $ra.



Dòng 7 : Gán giá trị của thanh ghi $v0 lúc này chứa kết quả cho thanh $s0. Lúc này thanh ghi $s0 là thanh chứa kết quả của hàm abs.



$t5	13	0
$t6	14	0
$t7	15	0
$s0	16	5
$s1	17	0

Dòng 9 10 : Kết thúc chương trình.

Dòng 14 : v0 = -a0

Dòng 15 : Nếu mà a0 < 0 thì jum tới done, lúc này thanh ghi $v0 chứa kết quả.

Dòng 17 : Ngược lại trường hợp a0 > 0 thì gán luôn giá trị của a0 vào thanh ghi $v0.

Dòng 19 : Thực hiện jum tới địa chỉ đã được lưu trong thanh $ra là dòng 6.

Assignment 2

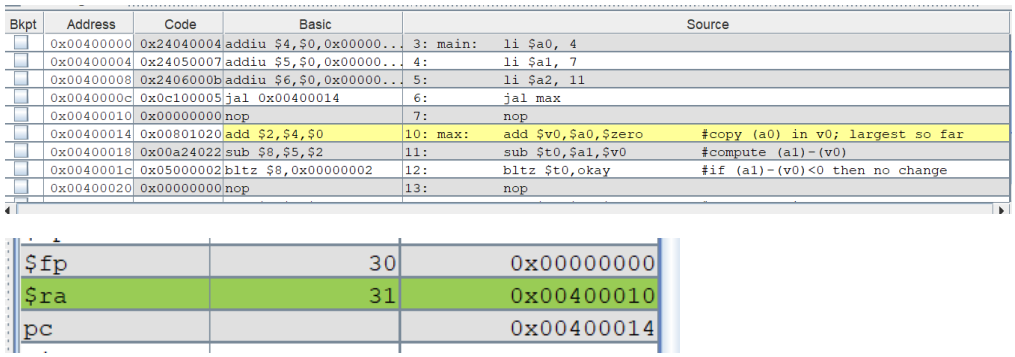
Code :

```
1  #Lab 7, Assignment 2
2  .text
3  main:   li $a0, 4
4          li $a1, 7
5          li $a2, 11
6          jal max
7          nop
8  endmain:
9
10 max:    add $v0,$a0,$zero      #copy (a0) in v0; largest so far
11         sub $t0,$a1,$v0        #compute (a1)-(v0)
12         bltz $t0,okay          #if (a1)-(v0)<0 then no change
13         nop
14         add $v0,$a1,$zero      #else (a1) is largest thus far
15
16 okay:   sub $t0,$a2,$v0        #compute (a2)-(v0)
17         bltz $t0,done          #if (a2)-(v0)<0 then no change
18         nop
19         add $v0,$a2,$zero      #else (a2) is largest overall
20
21 done: jr $ra                   #return to calling program
```

Giải thích :

Dòng 3-5 : Nhập 3 số nguyên

Dòng 6: Thực hiện gọi tới hàm max và đồng thời lưu trữ địa chỉ của lệnh tiếp theo (dòng số 7) vào thanh $ra



Bkpt	Address	Code	Basic	Source	
	0x00400000	0x24040004	addiu $4,$0,0x00000...	3: main: li $a0, 4	
	0x00400004	0x24050007	addiu $5,$0,0x00000...	4: li $a1, 7	
	0x00400008	0x2406000b	addiu $6,$0,0x00000...	5: li $a2, 11	
	0x0040000c	0x0c100005	jal 0x00400014	6: jal max	
	0x00400010	0x00000000	nop	7: nop	
	0x00400014	0x00801020	add $2,$4,$0	10: max: add $v0,$a0,$zero	#copy (a0) in v0; largest so far
	0x00400018	0x00a24022	sub $8,$5,$2	11: sub $t0,$a1,$v0	#compute (a1)-(v0)
	0x0040001c	0x05000002	bltz $8,0x00000002	12: bltz $t0,okay	#if (a1)-(v0)<0 then no change
	0x00400020	0x00000000	nop	13: nop	

$fp	30	0x00000000
$ra	31	0x00400010
pc		0x00400014

Dòng 10: v0 = a0 (v0 là thanh ghi lưu trữ số lớn nhất. được khởi tạo bằng số đầu tiên a0).

Dòng 11: t0 = a1 – v0

Dòng 12: Nếu t0 < 0 tức là a1 < v0 thì gọi tới hàm okay ngược lại thì thực hiện tiếp dòng 13

Dòng 14: v0 = a1 => cập nhật lại giá trị lớn nhất.

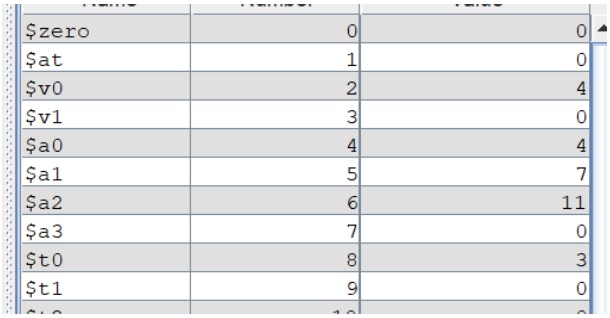
Dòng 16: t0 = a2 – v0

Dòng 17: Nếu t0 < 0 tức là a2 < v0 thì thực hiện hàm done (đã tìm được max)

Dòng 19: Cập nhật lại giá trị max v0 = a2

Dòng 20: Thực hiện lệnh mà địa chỉ đã được lưu ở trong thanh ghi $ra.

Kết quả:



$zero	0	0
$at	1	0
$v0	2	4
$v1	3	0
$a0	4	4
$a1	5	7
$a2	6	11
$a3	7	0
$t0	8	3
$t1	9	0

Assignment 3

Code:

```
1   #Lab 7, Assignment 3
2   .text
3           li $s0, 4
4           li $s1, 7
5   push:   addi $sp,$sp,-8 #adjust the stack pointer
6           sw $s0,4($sp) #push $s0 to stack
7           sw $s1,0($sp) #push $s1 to stack
8   work:   nop
9           nop
10          nop
11  pop:    lw $s0,0($sp) #pop from stack to $s0
12          lw $s1,4($sp) #pop from stack to $s1
13          addi $sp,$sp,8 #adjust the stack pointer
14
```

Giải thích:

Dòng 3-4: Khởi tạo s0, s1

$s0	16	0x00000004
$s1	17	0x00000007

Dòng 5: Giảm địa chỉ của ngăn xếp $sp đi 8byte để mở rộng ngăn xếp để có thể lưu trữ s0 và s1.

Trước:

$gp	28	0x10008000
$sp	29	0x7fffeffc

Sau:

$gp	28	0x10008000
$sp	29	0x7fffeff4
$fp	30	0x00000000

Dòng 6-7: Lần lượt đẩy s0 và s1 vào trong ngăn xếp.

+ Giá trị $s0 được lưu trữ tại địa chỉ 4($sp)

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x7fffefe0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000004	0x00000000
0x7ffff000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff0a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

+ Giá trị $s1 được lưu trữ tại địa chỉ 0($sp)

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x7fffefe0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000007	0x00000004	0x00000000
0x7ffff000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff0a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Dòng 11-12: Lần lượt lấy giá trị ở trong ngăn xếp rồi gán vào thanh ghi $s0, $s1.

+ Giá trị $s0 được lấy từ địa chỉ 0($sp) và được lưu trữ lại trong $s0.

$s0	16	0x00000007
$s1	17	0x00000007

+ Giá trị $s1 được lấy từ địa chỉ 4($sp) và được lưu trữ lại trong $s1.

$t7	15	0x00000000
$s0	16	0x00000007
$s1	17	0x00000004

Dòng 13: Con trỏ ngăn xếp $sp được điều chỉnh trở lại vị trí ban đầu bằng cách giảm địa chỉ ngăn xếp đi 8 byte và trả lại kích thước ban đầu cho ngăn xếp.

Trước:

$gp	28	0x10008000
$sp	29	0x7fffeff4
$fp	30	0x00000000

Sau:

$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000

Assignment 4

Code:

```
#Lab 7, Assignment 4
.data
        Message: .asciiz "Ket qua tinh giai thua la: "
.text
main:   jal WARP
print:  add $a1, $v0, $zero # $a0 = result from N!
        li $v0, 56
        la $a0, Message
        syscall
quit:   li $v0, 10
        syscall
endmain:

WARP:   sw $fp,-4($sp) #save frame pointer (1)
        addi $fp,$sp,0 #new frame pointer point to the top (2)
        addi $sp,$sp,-8 #adjust stack pointer (3)
        sw $ra,0($sp) #save return address (4)
        li $a0,3 #load test input N
        jal FACT #call fact procedure
        nop

        lw $ra,0($sp) #restore return address (5)
        addi $sp,$fp,0 #return stack pointer (6)
        lw $fp,-4($sp) #return frame pointer (7)
        jr $ra
wrap_end:
```

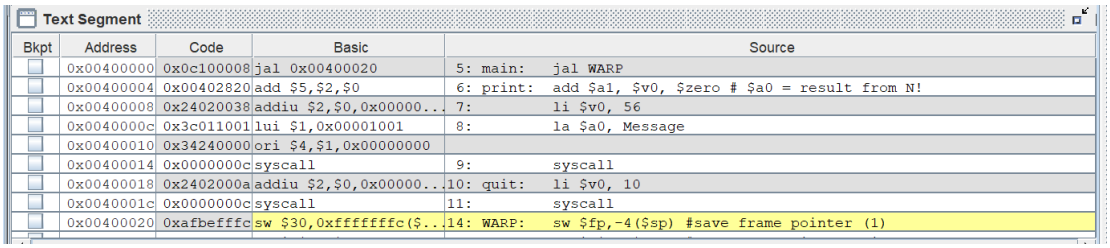
```

FACT:   sw $fp,-4($sp) #save frame pointer
        addi $fp,$sp,0 #new frame pointer point to stack's top
        addi $sp,$sp,-12 #allocate space for $fp,$ra,$a0 in stack
        sw $ra,4($sp) #save return address
        sw $a0,0($sp) #save $a0 register

        slti $t0,$a0,2 #if input argument N < 2
        beq $t0,$zero,recursive#if it is false ((a0 = N) >=2)
        nop
        li $v0,1 #return the result N!=1
        j done
        nop
recursive:
        addi $a0,$a0,-1 #adjust input argument
        jal FACT #recursive call
        nop
        lw $v1,0($sp) #load a0
        mult $v1,$v0 #compute the result
        mflo $v0
done:   lw $ra,4($sp) #restore return address
        lw $a0,0($sp) #restore a0
        addi $sp,$fp,0 #restore stack pointer
        lw $fp,-4($sp) #restore frame pointer
        jr $ra #jump to calling
fact_end:
```

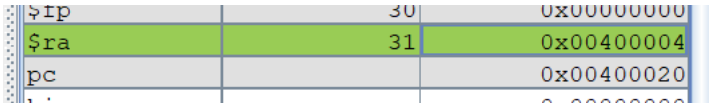
Giải thích:

Dòng 5: Thực hiện hàm WARP (Dòng 14) đồng thời lưu trữ địa chỉ của lệnh tiếp theo (Dòng 6) vào thanh ghi $ra = 0x00400004



Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x0c100008	jal 0x00400020	5: main: jal WARP
	0x00400004	0x00402820	add $5,$2,$0	6: print: add $a1, $v0, $zero # $a0 = result from N!
	0x00400008	0x24020038	addiu $2,$0,0x00000..	7: li $v0, 56
	0x0040000c	0x3c011001	lui $1,0x00001001	8: la $a0, Message
	0x00400010	0x34240000	ori $4,$1,0x00000000	
	0x00400014	0x0000000c	syscall	9: syscall
	0x00400018	0x2402000a	addiu $2,$0,0x00000..	10: quit: li $v0, 10
	0x0040001c	0x0000000c	syscall	11: syscall
	0x00400020	0xafbefffc	sw $30,0xfffffffc($..	14: WARP: sw $fp,-4($sp) #save frame pointer (1)



$ra	31	0x00400004
pc		0x00400020

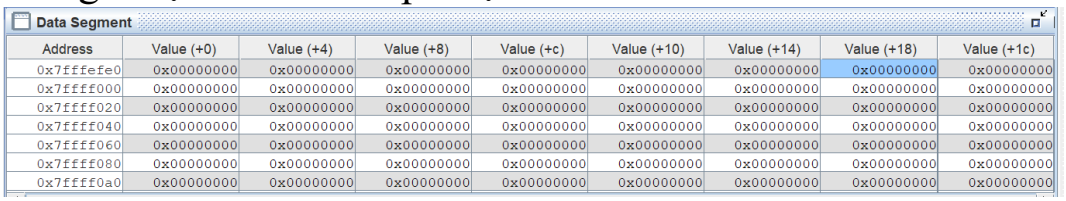
ĐỊA CHỈ STACK $sp = 0x7fffeffc (1)

Dòng 14: Giá trị $fp (frame pointer) được lưu trữ tại địa chỉ -4($sp)

Thanh ghi $sp:

$sp	29	0x7fffeffc

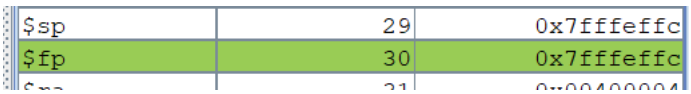
Nơi giá trị của thanh $fp được lưu:



Data Segment

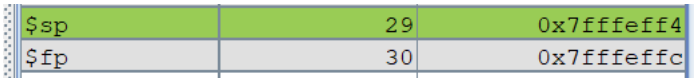
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x7fffefe0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff0a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Dòng 15: Tạo frame pointer mới trỏ tới đỉnh của stack



$sp	29	0x7fffeffc
$fp	30	0x7fffeffc

Dòng 16: Thay đổi con trỏ trỏ tới stack:



$sp	29	0x7fffeff4
$fp	30	0x7fffeffc

ĐỊA CHỈ STACK $sp = 0x7fffeff4 (2)

Dòng 17: Lưu trữ địa chỉ trả về (địa chỉ của lệnh printf) tại 0($sp)

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x7fffefe0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00400004	0x00000000	0x00000000
0x7ffff000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff0a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Dòng 18: Khởi tạo giá trị cần tính tại $a0. (a0 = 3)

Dòng 19: Gọi tới hàm FACT (Dòng 28) để cập nhật các giá trị vào stack. Đồng thời lưu trữ lệnh tiếp theo (Dòng 20) vào $ra

$sp	29	0x7fffeff4
$fp	30	0x7fffeffc
$ra	31	0x00400038

Text Segment

Bkpt	Address	Code	Basic		Source
	0x00400030	0x24040003	addiu $4,$0,0x00000..	18:	li $a0,3 #load test input N
	0x00400034	0x0c100013	jal 0x0040004c	19:	jal FACT #call fact procedure
	0x00400038	0x00000000	nop	20:	nop
	0x0040003c	0x8fbf0000	lw $31,0x00000000($..	22:	lw $ra,0($sp) #restore return address (5)
	0x00400040	0x23dd0000	addi $29,$30,0x0000..	23:	addi $sp,$fp,0 #return stack pointer (6)
	0x00400044	0x8fbefffc	lw $30,0xfffffffc($..	24:	lw $fp,-4($sp) #return frame pointer (7)
	0x00400048	0x03e00008	jr $31	25:	jr $ra
	0x0040004c	0xafbefffc	sw $30,0xfffffffc($..	28: FACT:	sw $fp,-4($sp) #save frame pointer
	0x00400050	0x23be0000	addi $30,$29,0x0000..	29:	addi $fp,$sp,0 #new frame pointer point to stack's top

Dòng 28: Lưu trữ $fp (frame pointer) vào địa chỉ -4($sp)

$fp = 0x7fffeffc

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x7fffefe0	0x00000000	0x00000000	0x00000000	0x00000000	0x7fffeffc	0x00400004	0x00000000	0x00000000
0x7ffff000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff0a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Dòng 29: Tạo frame pointer mới trỏ tới đỉnh của stack

$fp = $sp = 0x7fffeff4

$sp	29	0x7fffeff4
$fp	30	0x7fffeff4
$ra	31	0x00400038

Dòng 30: Cấp cho stack 12byte dùng để lưu trữ $fp,$ra,$a0

$sp	29	0x7fffefe8
$fp	30	0x7fffeff4
$ra	31	0x00400038

ĐỊA CHỈ STACK $sp = 0x7fffefe8 (3)

Dòng 31: Lưu trữ địa chỉ trả về dòng 20 tại 4($sp)

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x7fffefe0	0x00000000	0x00000000	0x00000000	0x00400038	0x7fffeffc	0x00400004	0x00000000	0x00000000
0x7ffff000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff0a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Dòng 32: Lưu trữ giá trị a0 = 3 tại 0($sp).

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x7fffefe0	0x00000000	0x00000000	0x00000003	0x00400038	0x7fffeffc	0x00400004	0x00000000	0x00000000
0x7ffff000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff0a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Dòng 34-35: Kiểm tra nếu a0 > = 2 thì thực hiện hàm recursive.

a0 = 3 => thực hiện hàm recursive.

Dòng 41: Giảm a0. a0 = a0 – 1 = 2.

Dòng 42: Gọi lại hàm FACT(dòng 28). Đồng thời lưu trữ địa chỉ trả về dòng 43 tại $ra

$sp	29	0x7fffefe8
$fp	30	0x7fffeff4
$ra	31	0x00400080

```
0x0040004c 0xafbefffc sw $30,0xfffffffc($...28: FACT:   sw $fp,-4($sp) #save frame pointer
0x00400050 0x23be0000 addi $30,$29,0x0000...29:         addi $fp,$sp,0 #new frame pointer point to stack's top
0x00400054 0x23bdfff4 addi $29,$29,0xffff...30:         addi $sp,$sp,-12 #allocate space for $fp,$ra,$a0 in stack
0x00400058 0xafbf0004 sw $31,0x00000004($...31:         sw $ra,4($sp) #save return address
0x0040005c 0xafa40000 sw $4,0x00000000($29) 32:         sw $a0,0($sp) #save $a0 register
0x00400060 0x28880002 slti $8,$4,0x00000002 34:         slti $t0,$a0,2 #if input argument N < 2
0x00400064 0x11000004 beq $8,$0,0x00000004  35:         beq $t0,$zero,recursive#if it is false ((a0 = N) >=2)
0x00400068 0x00000000 nop                   36:         nop
0x0040006c 0x24020001 addiu $2,$0,0x00000...37:         li $v0,1 #return the result N!=1
0x00400070 0x08100024 j 0x00400090          38:         j done
0x00400074 0x00000000 nop                   39:         nop
0x00400078 0x2084ffff addi $4,$4,0xffffffff 41:         addi $a0,$a0,-1 #adjust input argument
0x0040007c 0x0c100013 jal 0x0040004c        42:         jal FACT #recursive call
0x00400080 0x00000000 nop                   43:         nop
```

Dòng 28: lưu trữ $fb = 0x7fffeff4 vào địa chỉ -4($sp).

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x7fffefe0	0x00000000	0x7fffeff4	0x00000003	0x00400038	0x7ffffffc	0x00400004	0x00000000	0x00000000
0x7ffff000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff0a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Dòng 29: Tạo frame pointer mới trỏ tới đỉnh của stack

$fp = $sp = 0x7fffefe8

$sp	29	0x7fffefe8
$fp	30	0x7fffefe8
$ra	31	0x00400080

Dòng 30: Cấp cho stack 12byte dùng để lưu trữ $fp,$ra,$a0.

$sp	29	0x7fffefdc
$fp	30	0x7fffefe8
$ra	31	0x00400080

ĐỊA CHỈ STACK $sp = 0x7fffefdc (4)

Dòng 31-32: Lưu trữ địa chỉ trả về dòng 43 tại 4($sp) và lưu trữ giá trị a0 = 2 tại 0($sp).

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x7fffefc0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000002
0x7fffefe0	0x00400080	0x7fffeff4	0x00000003	0x00400038	0x7ffffffc	0x00400004	0x00000000	0x00000000
0x7ffff000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Dòng 34-35: Kiểm tra nếu a0 > = 2 thì thực hiện hàm recursive.

a0 = 2 => thực hiện hàm recursive.

Dòng 41: Giảm a0. a0 = a0 – 1 = 1.

Dòng 42: Gọi lại hàm FACT(dòng 28). Đồng thời lưu trữ địa chỉ trả về dòng 43 tại $ra (tưởng tự trên).

Dòng 28: lưu trữ $fb = 0x7fffefe8 vào địa chỉ -4($sp).

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x7fffefc0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x7fffefe8	0x00000002
0x7fffefe0	0x00400080	0x7fffeff4	0x00000003	0x00400038	0x7ffffffc	0x00400004	0x00000000	0x00000000
0x7ffff000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Dòng 29: Tạo frame pointer mới trỏ tới đỉnh của stack

$fp = $sp = 0x7fffefdc

$sp	29	0x7fffefdc
$fp	30	0x7fffefdc
$ra	31	0x00400080

Dòng 30: Cấp cho stack 12byte dùng để lưu trữ $fp,$ra,$a0.

$sp	29	0x7fffefd0
$fp	30	0x7fffefdc
$ra	31	0x00400080

ĐỊA CHỈ STACK $sp = 0x7fffefd0 (5)

Dòng 31-32: Lưu trữ địa chỉ trả về dòng 43 tại 4($sp) và lưu trữ giá trị a0 = 1 tại 0($sp)

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x7fffefc0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000001	0x00400080	0x7fffefe8	0x00000002
0x7fffefe0	0x00400080	0x7fffeff4	0x00000003	0x00400038	0x7fffeffc	0x00400004	0x00000000	0x00000000
0x7ffff000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Dòng 34-35: Kiểm tra nếu a0 > = 2 thì thực hiện hàm recursive.

a0 = 1=> thực hiện dòng 36.

Dòng 37: Trả lại kết quả với a0 = 1 => v0 = 1! = 1

Dòng 38: Thực hiện hàm done.

Dòng 47-48: Lấy ra địa chỉ trả về dòng 43 $ra = 0x00400080 từ 4($sp) và giá trị a0 = 1 từ 0($sp).

Dòng 49: Khôi phục lại con trỏ stack trước đó là giá trị của $fp hiện tại.

$sp	29	0x7fffefdc
$fp	30	0x7fffefdc
$ra	31	0x00400080

Dòng 50: Khôi phục lại frame pointer được lưu tại -4($sp).

$sp	29	0x7fffefdc
$fp	30	0x7fffefe8
$ra	31	0x00400080

Dòng 51: Thực hiện jum tới địa chỉ đang được lưu trong thanh $ra. Hiện tại là địa chỉ của dòng 43.

Dòng 44: Load giá trị a0 = 2 đang lưu tại 0($sp) vào thanh $v1.

v1 = 2

Dòng 45-46: Nhân v1=2 với v0=1 rồi kết quả lưu vào thanh v0

$v0	2	0x00000002
$v1	3	0x00000002

Dòng 47-50: Khôi phục lại $ra = 0x00400080, $a0 = 2, $sp = 0x7fffefe8, $fp = 0x7fffeff4 được lấy ra từ stack.

$sp	29	0x7fffefe8
$fp	30	0x7fffeff4
$ra	31	0x00400080

Dòng 51: Thực hiện jum tới địa chỉ đang được lưu trong thanh $ra. Hiện tại là địa chỉ của dòng 43.

Dòng 44: Load giá trị a0 = 3 đang lưu tại 0($sp) vào thanh $v1.

v1 = 3

Dòng 45-46: Nhân v1=3 với v0=2 rồi kết quả lưu vào thanh v0

$v0	2	0x00000006
$v1	3	0x00000003

Dòng 47-50: Khôi phục lại $ra = 0x00400038, $a0 = 3, $sp = 0x7fffeff4, $fp = 0x7fffeffc được lấy ra từ stack.

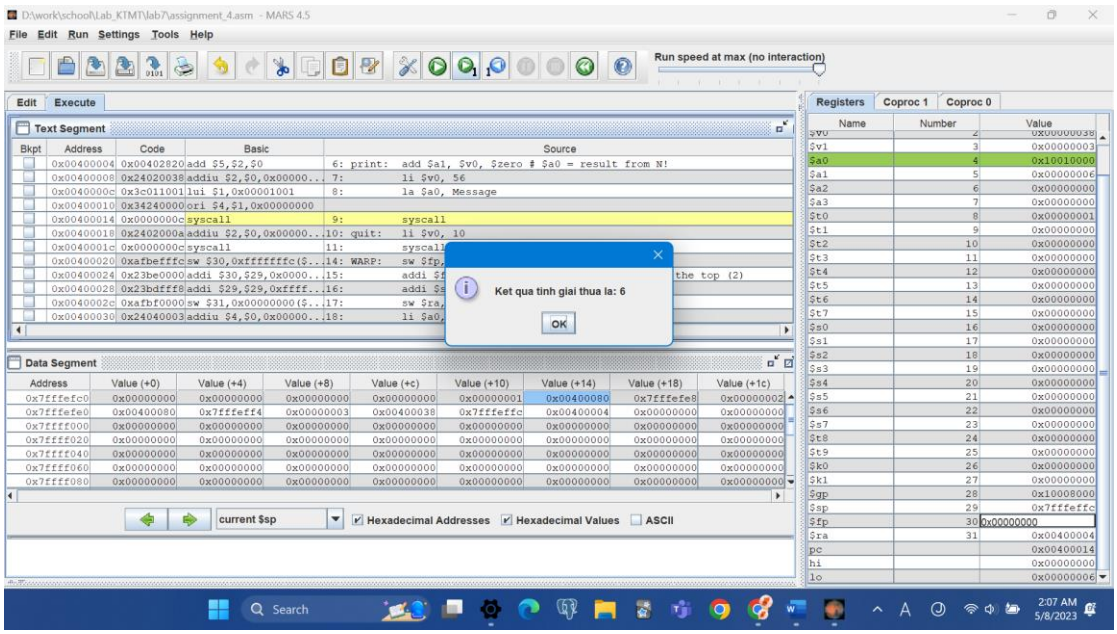
$sp	29	0x7fffeff4
$fp	30	0x7fffeffc
$ra	31	0x00400038

Dòng 51: Thực hiện jum tới địa chỉ đang được lưu trong thanh $ra. Hiện tại là địa chỉ của dòng 20.

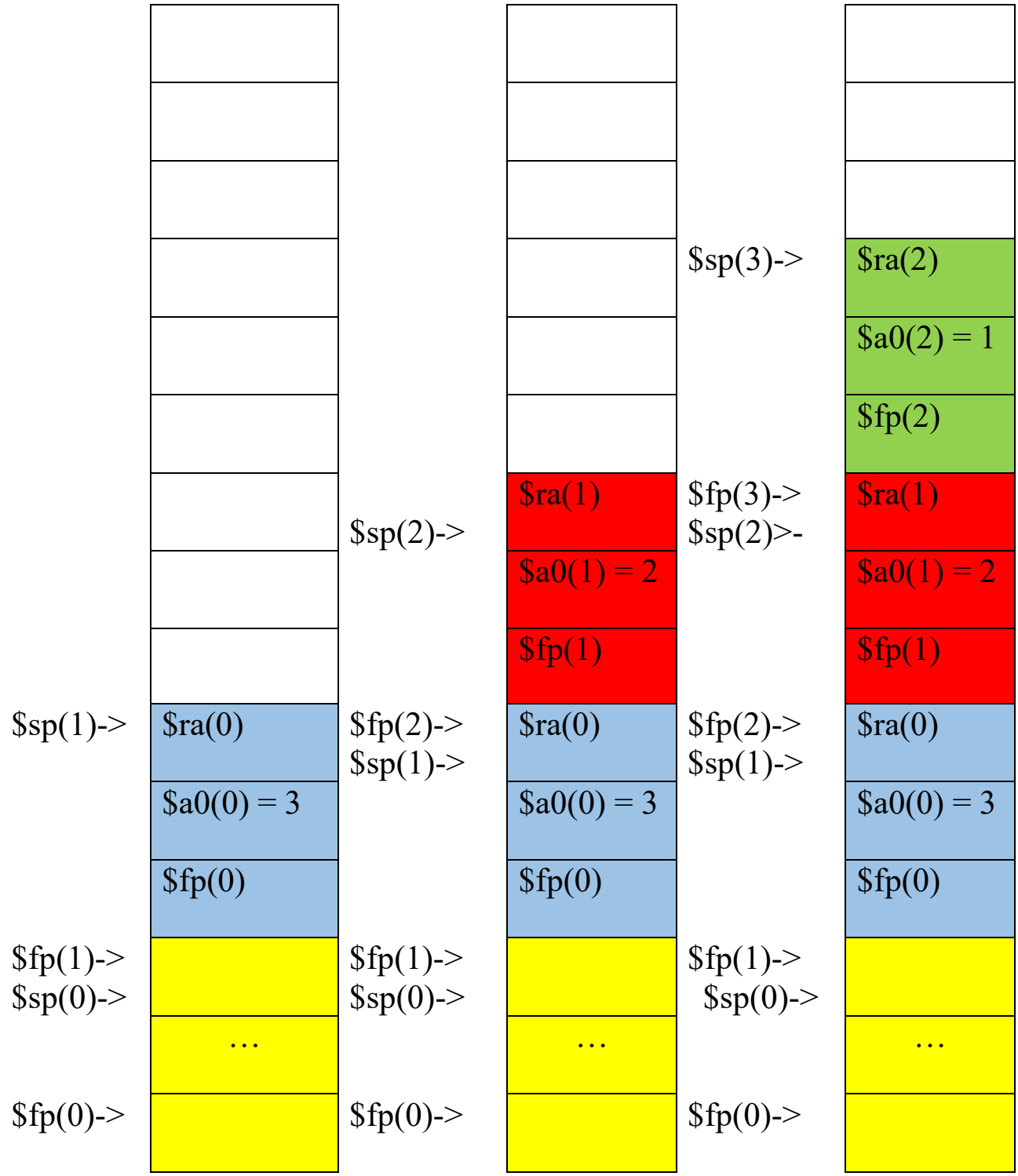
Dòng 22-24: Khôi phục lại $ra = 0x00400004, $sp = 0x7fffeffc, $fp = 0x00000000

Dòng 25: Thực hiện jum tới địa chỉ đang được lưu trong thanh $ra. Hiện tại là địa chỉ của dòng 6.

Dòng 6-9: Thực hiện in ra màn hình.



Dòng 10-11: Kết thúc chương trình.


$sp(1)->
$fp(1)->
$sp(0)->
$fp(0)->
$ra(0)
$a0(0) = 3
$fp(0)
…
$sp(2)->
$fp(2)->
$sp(1)->
$fp(1)->
$sp(0)->
$fp(0)->
$ra(1)
$a0(1) = 2
$fp(1)
$ra(0)
$a0(0) = 3
$fp(0)
…
$sp(3)->
$fp(3)->
$sp(2)>-
$fp(2)->
$sp(1)->
$fp(1)->
$sp(0)->
$fp(0)->
$ra(2)
$a0(2) = 1
$fp(2)
$ra(1)
$a0(1) = 2
$fp(1)
$ra(0)
$a0(0) = 3
$fp(0)
…

Asignment 5

Code:

```
#Lab 7, Assignment 5
.data
        Message1: .asciiz "Lon nhat: "
        Message2: .asciiz "Nho nhat: "
        space: .asciiz ", "
        newline: .asciiz "\n"
.text
main:
        li $s0, 1
        li $s1, 5
        li $s2, -3
        li $s3, -7
        li $s4, 7
        li $s5, -6
        li $s6, 9
        li $s7, 1
        jal MINMAX

print:
        addi $t0, $a0, 0
        addi $t1, $a1, 0
        addi $t2, $a2, 0
        addi $t3, $a3, 0
        li $v0, 4
        la $a0, Message1
        syscall
```

```
        li $v0, 1
        addi $a0, $t2, 0
        syscall
        li $v0, 4
        la $a0, space
        syscall
        li $v0, 1
        addi $a0, $t3, 0
        syscall
        li $v0, 4
        la $a0, newline
        syscall
        la $a0, Message2
        syscall
        li $v0, 1
        addi $a0, $t0, 0
        syscall
        li $v0, 4
        la $a0, space
        syscall
        li $v0, 1
        addi $a0, $t1, 0
        syscall
quit:   li $v0, 10
        syscall
```

```
52
53  MINMAX:
54          addi $sp, $sp, -8
55          sw $fp, 4($sp)
56          sw $ra, 0($sp)
57  push_to_stack:
58          addi $a0, $zero, 1000 #min
59          addi $a1, $zero, -1 #min position
60          addi $a2, $zero, -1000 #max
61          addi $a3, $zero, -1 #max position
62          addi $sp, $sp, -48
63          sw $s7, 44($sp)
64          sw $s6, 40($sp)
65          sw $s5, 36($sp)
66          sw $s4, 32($sp)
67          sw $s3, 28($sp)
68          sw $s2, 24($sp)
69          sw $s1, 20($sp)
70          sw $s0, 16($sp)
71          sw $a0, 12($sp)
72          sw $a1, 8($sp)
73          sw $a2, 4($sp)
74          sw $a3, 0($sp)
75
76          addi $t0, $zero, -1
77  loop:
```

```
77  loop:
78          addi $t0, $t0, 1 #t0 is loop counter
79          beq $t0, 8, MINMAX_done
80          jal compare
81          nop
82  j loop
83          nop
84  compare:
85          lw $a3, 0($sp)
86          lw $a2, 4($sp)
87          lw $a1, 8($sp)
88          lw $a0, 12($sp)
89          lw $t1, 16($sp) #t1 is the current value to compare
90          addi $sp, $sp, 20
91  compare_min:
92          sub $t2, $t1, $a0 #compare with min
93          slti $t2, $t2, 0
94          beqz $t2, compare_max
95          addi $a0, $t1, 0
96          addi $a1, $t0, 0
97  compare_max:
98          sub $t2, $t1, $a2 #compare with max
99          sgt $t2, $t2, 0
100         beqz $t2, compare_done
101         addi $a2, $t1, 0
102         addi $a3, $t0, 0
```

```
103 compare_done:
104         addi $sp, $sp, -16 #push results back to stack
105         sw $a0, 12($sp)
106         sw $a1, 8($sp)
107         sw $a2, 4($sp)
108         sw $a3, 0($sp)
109         jr $ra
110 MINMAX_done:
111         lw $a3, 0($sp)
112         lw $a2, 4($sp)
113         lw $a1, 8($sp)
114         lw $a0, 12($sp)
115         lw $ra, 16($sp)
116         lw $fp, 20($sp)
117         addi $sp, $sp, 24
118         jr $ra
119
```

Giải thích:

Dòng 9-16: Khai báo 8 phần tử

Dòng 17: Thực hiện hàm MINMAX (dòng 53), lưu địa chỉ lệnh tiếp theo vào thanh $ra.

Dòng 20-51: Thực hiện in ra kết quả và kết thúc chương trình.

Dòng 53: Bắt đầu hàm MINMAX

Dòng 54-56: Mở rộng ngăn xếp để lưu frame pointer ($fp) và địa chỉ trở về ($ra) đã được lưu trước đó.

Dòng 58-61: Khởi tạo giá trị và vị trí của số min, max.

Dòng 62: Mở rộng ngăn xếp để lưu trữ các phần tử.

Dòng 63-74: Lưu trữ các phần tử và các giá trị vửa khởi tạo của min, max vào ngăn xếp.

Dòng 76: Khởi tạo giá trị t0 để bắt đầu vòng lặp.

Dòng 77-82: Vòng lặp duyệt từ 0 đến 7 để tìm min, max.

Dòng 85-90: Lấy ra các giá trị, vị trí min max.

Dòng 91-96: Hàm tìm min. a0 là giá trị min tìm đc và a1 là vị trí

Dòng 97-102: Hàm tìm max. a2 là giá trị max tìm đc và a3 là vị trí.

Dòng 103-109: Kết thúc hàm so sánh compare. Đẩy kết quả vừa tìm được vào stack. Và trở lại thực hiện địa chỉ đã đc lưu trong stack để thực hiện lệnh tiếp theo.

Dòng 110-118: Kết thúc hàm MINMAX. Lấy ra các giá trị, vị trí vừa tìm đc vào trở về thực hiện lệnh tiếp theo để in kết quả ra màn hình.

Kết quả:

```
li $s0, 1
li $s1, 5
li $s2, -3
li $s3, -7
li $s4, 7
li $s5, -6
li $s6, 9
li $s7, 1
```

Với các giá trị như trên, ta thu được kết quả dưới đây sau khi thực hiện chương trình: