

ASSIGNMENT LAB 7

-Họ tên: Hồ Tuấn Huy

-MSSV: 20225856

Assignment 1:

Assignment 1

Create a new project to implement the program in Home Assignment 1. Compile and upload to simulator. Change input parameters and observe the memory when run the program step by step. Pay attention to register pc, \$ra to clarify invoking procedure process (Refer to figure 7).

```
1  #Laboratory Exercise 7 Home Assignment 1
2  .text
3  main: li $a0,-39 #load input parameter
4  jal abs #jump and link to abs procedure
5  nop
6  add $s0, $zero, $v0
7  li $v0,20 #terminate
8  syscall
9  endmain:
10
11 abs:
12 sub $v0,$zero,$a0 #put -(a0) in v0; in case (a0)<0
13 bltz $a0,done #if (a0)<0 then done
14 nop
15 add $v0,$a0,$zero #else put (a0) in v0
16 done:
17 jr $ra
```

- \$a0 là tham số đầu vào, \$v0 là giá trị trả ra của chương trình con.
- Lệnh jal abs=>jump and link đến chương trình con abs, đồng thời gán cho \$ra địa chỉ của câu lệnh tiếp theo câu lệnh jal là nop.

Chương trình con abs:

+ Gán giá trị của -\$a0 vào \$v0.

+ So sánh nếu \$a0<0 thì chuyển đến hàm done.

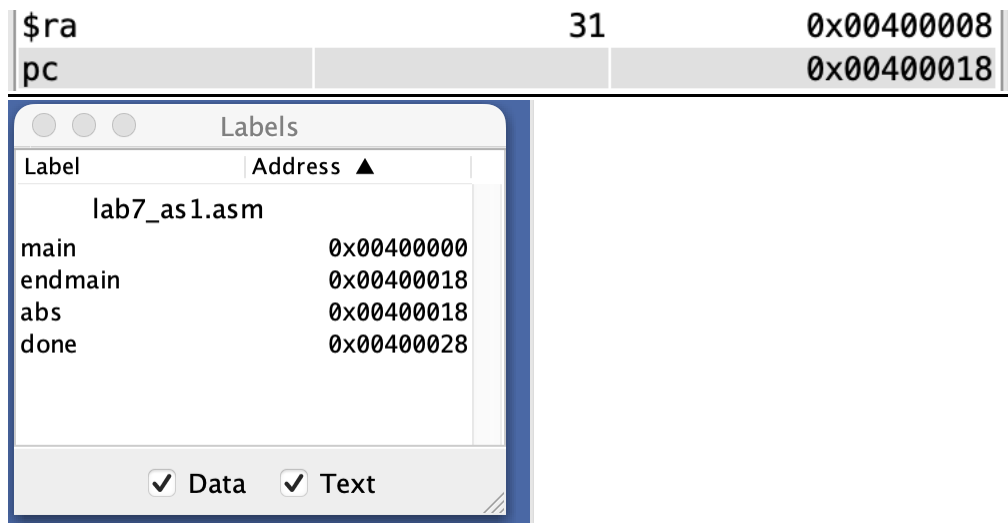
Hàm done: Lệnh jr \$ra để quay trở về địa chỉ lưu trong thanh ghi \$ra.

⇒ **Kết quả:**

\$s0	16	0x00000027
------	----	------------

27 là biểu diễn của 39 ở hệ 16 ($7 \cdot 16^0 + 2 \cdot 16^1 = 39$).

Khi chạy lệnh jal abs (địa chỉ lệnh 0x00400004) thì địa chỉ của \$ra được gán bằng địa chỉ của lệnh tiếp theo (0x00400008) và địa chỉ của \$pc được gán bằng địa chỉ tại nhãn \$abs:



Assignment 2:

Assignment 2

Create a new project to implement the program in Home Assignment 2. Compile and upload to simulator. Change input parameters (register a0, a1, a2) and observe the memory when run the program step by step. Pay attention to register pc, \$ra to clarify invoking procedure process (Refer to figure 7).

```

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

```

- Trước hết gán \$a0, \$a1, \$a2 lần lượt là 1, 2, 3.
- Lệnh jal max=>jump and link đến chương trình con max, đồng thời gán cho \$ra địa chỉ của câu lệnh tiếp theo câu lệnh jal là nop. Thanh ghi \$pc được

gán bằng địa chỉ của nhãn max để câu lệnh tiếp tục được thực hiện bắt đầu từ nhãn max. Sau khi chạy đến jr \$ra thì \$pc được gán bằng địa chỉ trong \$ra (là địa chỉ của nop).

Chương trình con max:

+ Gán \$a0 vào \$v0 (max).

+ Tính giá trị \$a1-\$v0 rồi gán vào \$t0, nếu \$t0<0 (\$a1<max) thì nhảy đến okay, còn nếu ngược lại (\$t0>0) thì gán max là \$a1, sau đó thực hiện tiếp hàm okay.

Hàm okay: Tính hiệu \$a2-\$v0 rồi gán vào \$t0, so sánh và thực hiện tương tự như với trường hợp chương trình con max.

Hàm done: Lệnh jr \$ra để quay trở về địa chỉ lưu trong thanh ghi \$ra.

⇒ Kết quả:

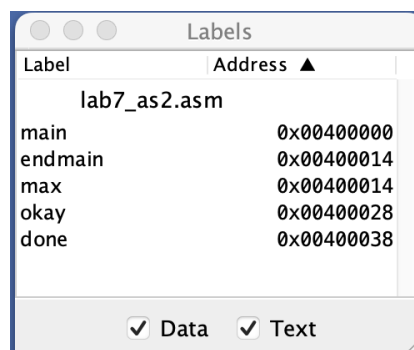
- Thu được max là 3 (\$a2) được lưu vào \$v0:

\$a0	4	0x00000001
\$a1	5	0x00000002
\$a2	6	0x00000003

\$v0	2	0x00000003
------	---	------------

- Khi chạy lệnh jal abs (địa chỉ lệnh 0x00400012) thì \$ra được gán địa chỉ của câu lệnh tiếp theo (0x00400016), ở đây đuôi 10 ở hệ 16 đổi ra hệ thập phân là $0 \cdot 16^0 + 1 \cdot 16^1 = 16$ (thoả mãn). Còn thanh ghi \$pc được gán bằng địa chỉ của nhãn max để câu lệnh tiếp tục được thực hiện bắt đầu từ nhãn max. Khi chạy đến jr \$ra thì \$pc được gán địa chỉ của \$ra.

\$ra	31	0x00400010
pc		0x00400038



Label	Address ▲
lab7_as2.asm	
main	0x00400000
endmain	0x00400014
max	0x00400014
okay	0x00400028
done	0x00400038

☒ Data ☒ Text

Assignment 3:

Assignment 3

Create a new project to implement the program in Home Assignment 3. Compile and upload to simulator. Pass test value to registers \$s0 and \$s1, observe run process, pay attention to stack pointer. Goto memory space that pointed by sp register to view push and pop operations in detail.

```
1  #Laboratory Exercise 7, Home Assignment 3
2  .text
3  li $s0, 1
4  li $s1, 2
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
10 pop: lw $s0,0($sp) #pop from stack to $s0
11     lw $s1,4($sp) #pop from stack to $s1
12     addi $sp,$sp,8 #adjust the stack pointer
13 end:
```

- Đầu tiên gán giá trị cho \$s0 và \$s1 lần lượt là 1 và 2.
- Hàm push:
 - + Mở rộng bộ nhớ của stack ra 8 bytes (tương ứng 2 ngăn nhớ). Vì stack hoạt động ngược lại so với các trường hợp khác, mà mỗi lần gán tương ứng 4 bytes nên phải trừ đi 8 bytes.
 - + Đẩy giá trị \$s0 và \$s1 vào stack.
- Hàm pop: Lấy ra lần lượt các giá trị từ đầu stack lưu lại vào \$s0 và \$s1.

⇒ **Kết quả:**

- Push:

\$s0	16	0x00000001
\$s1	17	0x00000002

- Pop:

\$s0	16	0x00000002
\$s1	17	0x00000001

Assignment 4:

Assignment 4

Create a new project to implement the program in Home Assignment 4. Compile and upload to simulator. Pass test input through register a0, run this program and test result in register v0. Run this program in step by step mode, observe the changing of register pc, ra, sp and fp. Draw the stack through this recursive program in case of n=3 (compute 3!).

```

1  #Laboratory Exercise 7, Home Assignment 4
2  .data
3  Message: .asciiz "Ket qua tinh giai thua la: "
4  .text
5  main: jal WARP
6  print: add $a1, $v0, $zero # $a0 = result from N!
7  li $v0, 56
8  la $a0, Message
9  syscall
10 quit: li $v0, 10 #terminate
11 syscall
12 endmain:
13
14 WARP: sw $fp,-4($sp) #save frame pointer (1)
15 addi $fp,$sp,0 #new frame pointer point to the top (2)
16 addi $sp,$sp,-8 #adjust stack pointer (3)
17 sw $ra,0($sp) #save return address (4)
18 li $a0,3 #load test input N
19 jal FACT #call fact procedure
20 nop
21 lw $ra,0($sp) #restore return address (5)
22 addi $sp,$fp,0 #return stack pointer (6)
23 lw $fp,-4($sp) #return frame pointer (7)
24 jr $ra
25 wrap_end:

```

```

26
27 FACT: sw $fp,-4($sp) #save frame pointer
28 addi $fp,$sp,0 #new frame pointer point to stack's
29 top:
30 addi $sp,$sp,-12 #allocate space for $fp,$ra,$a0 in
31 stack:
32 sw $ra,4($sp) #save return address
33 sw $a0,0($sp) #save $a0 register
34 slti $t0,$a0,2 #if input argument N < 2
35 beq $t0,$zero,recursive#if it is false ((a0 = N) >=2)
36 nop
37 li $v0,1 #return the result N!=1
38 j done
39 nop
40 recursive:
41 addi $a0,$a0,-1 #adjust input argument
42 jal FACT #recursive call
43 nop

```

```

43 nop
44 lw $v1,0($sp) #load a0
45 mult $v1,$v0 #compute the result
46 mflo $v0
47 done: lw $ra,4($sp) #restore return address
48 lw $a0,0($sp) #restore a0
49 addi $sp,$fp,0 #restore stack pointer
50 lw $fp,-4($sp) #restore frame pointer
51 jr $ra #jump to calling
52 fact_end:

```

- \$a0 là thanh ghi được sử dụng để truyền tham số đầu vào cho các hàm con. Trong trường hợp này, nó truyền tham số đầu vào là N (số cần tính giai thừa) vào hàm FACT.
- Trong chương trình con WARP: Mở rộng stack ra 8 bytes để lưu giá trị \$fp và \$ra=>Đảm bảo sau khi kết thúc chương trình con thì chương trình sẽ quay lại được vị trí gọi chương trình con từ chương trình chính main.
- Trong chương trình con FACT:
 - + Mỗi lần đệ quy sẽ mở rộng stack thêm 12 bytes để lưu \$a0, \$fp, \$ra=>Đảm bảo các giá trị quan trọng được lưu trữ và không bị ghi đè trong quá trình đệ quy.
 - + Giảm \$a0 đi 1 đơn vị cho đến khi \$a0<2 để thực hiện hàm done=>Đảm bảo đệ quy dừng khi đạt được điều kiện cơ bản là N<2.
 - + Sau mỗi lần thoát đệ quy, thực hiện nhân giá trị trên đỉnh stack với \$v0 (\$v0 chứa kết quả hàm con, nhân \$v0 để tính giá trị giai thừa cuối cùng).

⇒ **Kết quả** (tính 3!):



- **Minh hoạ:**

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

Assignment 5:

Assignment 5

Write a procedure to find the largest, the smallest and these positions in a list of 8 elements that are stored in registers \$s0 through s7. For example:

Largest: 9,3 -> The largest element is stored in \$s3, largest value is 9

Smallest: -3,6 -> The smallest element is stored in \$s6, smallest value is -3

Tips: using stack to pass arguments and return results.

```
1  .data
2  message1: .asciiz "largest: "
3  message2: .asciiz "smallest: "
4  comma: .asciiz ","
5  newline: .asciiz "\n"
6  .text
7  main: jal warp
8  print:
9  add $a1, $v0, $zero
10 add $a2, $v1, $zero
11 li $v0, 4
12 la $a0, message1
13 syscall
14 li $v0, 1
15 addi $a0, $a1, 0
16 syscall
17 li $v0, 4
18 la $a0, comma
19 syscall
20 li $v0, 1
21 addi $a0, $t0, 0
22 syscall
23 li $v0, 4
24 la $a0, newline
25 syscall
26 li $v0, 4
27 la $a0, message2
28 syscall
29 li $v0, 1
30 addi $a0, $a2, 0
31 syscall
32 li $v0, 4
33 la $a0, comma
34 syscall
35 li $v0, 1
36 addi $a0, $t1, 0
37 syscall
38 quit: li $v0, 10
39 syscall
40 endmain:
41 warp:
42 addi $fp, $sp, 0
43 addi $sp, $sp, -32
44 addi $s0, $zero, 3
45 sw $a0, 28($sp)
46 addi $s1, $zero, 6
47 sw $s1, 24($sp)
48 addi $s2, $zero, -1
49 sw $s2, 20($sp)
50
51 sw $s2, 20($sp)
52 addi $s3, $zero, -10
53 sw $s3, 16($sp)
54 addi $s4, $zero, 15
55 sw $s4, 12($sp)
56 addi $s5, $zero, 22
57 sw $s5, 8($sp)
58 addi $s6, $zero, 10
59 sw $s6, 4($sp)
60 addi $s7, $zero, 1
61 sw $s7, 0($sp)
62 addi $v0, $zero, 0x80000000 #value of max element
63 addi $v1, $zero, 0x7fffffff #value of min element
64 addi $t0, $zero, 7 #index of max element
65 addi $t1, $zero, 7 #index of min element
66 addi $t7, $zero, 7 #index
67 loop:
68 lw $t2, 0($sp)
69 checkMax:
70 slt $t3, $v0, $t2 #check max<current
71 beq $t3, $zero, checkMin #max>current->checkMin
72 addi $v0, $t2, 0 #max<current then update max=current
73 addi $t0, $t7, 0 #update index
74
75 addi $t0, $t7, 0 #update index
76 checkMin:
77 slt $t3, $t2, $v1 #check current<min
78 beq $t3, $zero, continue #if current>min->continue
79 addi $v1, $t2, 0 #if current<min->update min=current
80 addi $t1, $t7, 0 #update index
81 continue:
82 addi $sp, $sp, 4
83 addi $t7, $t7, -1
84 bne $sp, $fp, loop
85 li $fp, 0
86 jr $ra
87
```

- Hàm warp:
 - + Mở rộng stack 32 bytes để tạo 8 ngăn nhớ.
 - + Khai báo lần lượt các giá trị trong danh sách 8 phần tử vào các thanh ghi \$s0->\$s7 và push vào stack.

- + \$v0, \$v1 là giá trị max và min; \$t0, \$t1 là index của max và min.
- + \$t7 là biến chạy, sau mỗi lần lặp bị giảm đi 1.
- Hàm loop: Trong mỗi vòng lặp lấy ra giá trị ở đầu stack lưu vào \$t2 rồi so sánh \$t2 với giá trị max và min bằng các hàm checkMax và checkMin:
 - + checkMax kiểm tra nếu giá trị hiện tại > max thì cập nhật max và vị trí tương ứng \$t0.
 - + checkMin kiểm tra nếu giá trị hiện tại < min thì cập nhật min và vị trí tương ứng \$t1.
 - + Sau mỗi lần lặp thì di chuyển con trỏ stack \$sp đến phần tử tiếp theo và giảm biến đếm \$t7.
 - + Khi con trỏ stack \$sp đạt đến vị trí bằng \$fp thì kết thúc vòng lặp và quay lại hàm chính.

⇒ **Kết quả:**

largest: 22,5		
smallest: -10,3		
\$s0	16	0x00000003
\$s1	17	0x00000006
\$s2	18	0xffffffff
\$s3	19	0xffffffff6
\$s4	20	0x0000000f
\$s5	21	0x00000016
\$s6	22	0x0000000a
\$s7	23	0x00000001