BÀI THỰC HÀNH TUẦN 7
KIẾN TRÚC MÁY TÍNH

Họ và tên: Đinh Huy Dương

MSSV: 20215020

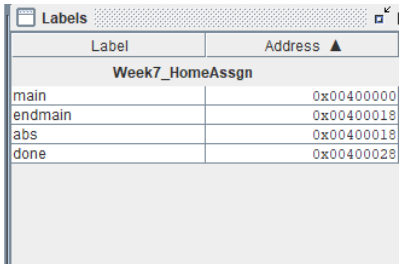
Bài 1:

```
.text
main:
        li      $a0, -100               #load input parameter
        jal     abs                     #jump and link to abs procedure
        nop
        add     $s0, $zero, $v0
        li      $v0, 10                 #terminate
        syscall
endmain:
abs:
        sub     $v0,$zero,$a0           #put -(a0) in v0; in case (a0)<0
        bltz    $a0,done                #if (a0)<0 then done
        nop
        add     $v0,$a0,$zero           #else put (a0) in v0
done:
        jr      $ra
```

Địa chỉ của các câu lệnh:

Bkpt	Address	Code	Basic	Source			
	0x00400000	0x2404ff9c	addiu $4,$0,0xffffff9c	4:	li	$a0, -100	#load input parameter
	0x00400004	0x0c100006	jal 0x00400018	5:	jal	abs	#jump and link to abs procedure
	0x00400008	0x00000000	nop	6:	nop		
	0x0040000c	0x00028020	add $16,$0,$2	7:	add	$s0, $zero, $v0	
	0x00400010	0x2402000a	addiu $2,$0,0x0000000a	8:	li	$v0, 10	#terminate
	0x00400014	0x0000000c	syscall	9:	syscall		
	0x00400018	0x00041022	sub $2,$0,$4	17:	sub	$v0,$zero,$a0	#put -(a0) in v0; in case (a0)<0
	0x0040001c	0x04800002	bltz $4,0x00000002	19:	bltz	$a0,done	#if (a0)<0 then done
	0x00400020	0x00000000	nop	20:	nop		
	0x00400024	0x00801020	add $2,$4,$0	21:	add	$v0,$a0,$zero	#else put (a0) in v0
	0x00400028	0x03e00008	jr $31	23:	jr	$ra	

Địa chỉ của các nhãn và thủ tục:


Labels

Label	Address ▲
Week7_HomeAssgn	
main	0x00400000
endmain	0x00400018
abs	0x00400018
done	0x00400028

Giá trị của thanh ghi $pc và $ra trước lệnh "jal": $pc vẫn mang địa chỉ của các câu lệnh tuần tự. Giá trị của $ra chưa được gán cho giá trị nào.

$ra	31	0x00000000
pc		0x00400004

Sau lệnh "jal": $pc mang địa chỉ thủ tục "abs" để nhảy sang nhãn này. bên cạnh đó giá trị của $ra bằng với giá trị của $pc trước +4 để có thể khôi phục lại trạng thái cũ và thực hiện câu lệnh tiếp theo của chương trình sau 1 hàm hoặc thủ tục.

$ra	31	0x00400008
pc		0x00400018

Khi kết thúc thủ tục "abs", tại lệnh `jr $ra`, thanh ghi $pc sẽ được gán với giá trị của $ra để quay trở về chương trình chính sau thủ tục:

$ra	31	0x00400008
pc		0x00400008

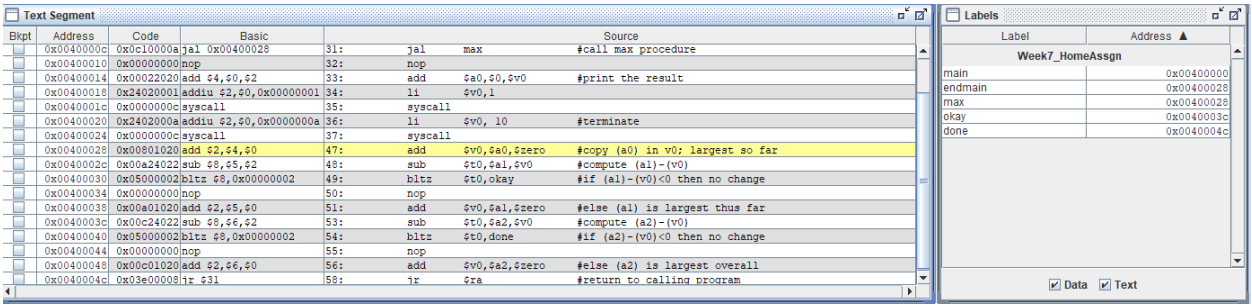
Chương trình được kết thúc, sau khi thực hiện hàm "abs", trả giá trị tuyệt đối của $a0 vào trong thanh ghi $s0. Trong chương trình, ta đặt $a0 = -100 = 0xffff ff9c, nên $s0 = 100 = 0x64

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0xffffff9c
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000064

Bài 2:

```
.text
main:
	li	$a0,4		#load test input
	li	$a1,100
	li	$a2,-3
	jal	max		#call max procedure
	nop
	add	$a0,$0,$v0	#print the result
	li	$v0,1
	syscall
	li	$v0, 10			#terminate
	syscall
endmain:
max:
	add	$v0,$a0,$zero		#copy (a0) in v0; largest so far
	sub	$t0,$a1,$v0		#compute (a1)-(v0)
	bltz	$t0,okay		#if (a1)-(v0)<0 then no change
	nop
	add	$v0,$a1,$zero		#else (a1) is largest thus far
okay:
	sub	$t0,$a2,$v0		#compute (a2)-(v0)
	bltz	$t0,done		#if (a2)-(v0)<0 then no change
	nop
	add	$v0,$a2,$zero		#else (a2) is largest overall
done:
	jr	$ra		#return to calling program
```

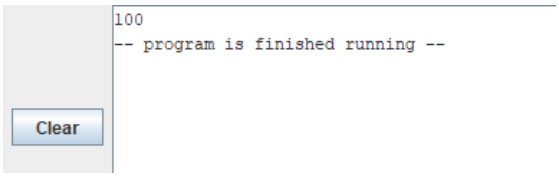
Địa chỉ câu lệnh và nhãn:



Tương tự với Bài 1, $ra sẽ mang địa chỉ của câu lệnh tiếp theo lệnh "jal" trong chương trình chính (main). Đây là giá trị của $pc và $ra sau lệnh "jal" (chương trình đã nhảy đến hàm "max"

$ra	31	0x00400010
pc		0x00400028

Kết thúc hàm ở lệnh `jr $ra` gán $pc = $ra để quay về địa chỉ câu lệnh tiếp sau gọi hàm trong main. Chương trình hoàn thiện và in ra kết quả số lớn nhất trong 3 số lưu tại $a0, $a1, $a2 (Lần lượt = 4, 100, -3):



Bài 3:

```
.text
push:  addi   $sp,$sp,-8   #adjust the stack pointer
       li     $s0, 4
       li     $s1 -3
       sw     $s0,4($sp)   #push $s0 to stack
       sw     $s1,0($sp)   #push $s1 to stack
work:  nop
       nop
       nop
pop:   lw     $s0,0($sp)   #pop from stack to $s0
       lw     $s1,4($sp)   #pop from stack to $s1
       addi   $sp,$sp,8    #adjust the stack pointer
```

Giá trị tại thanh ghi $sp: Địa chỉ của con trỏ ô nhớ tại vùng Stack:

Mặc định:

$sp	29	0x7fffeffc

Trừ 8 để cho thêm 2 phần tử:

$sp	29	0x7fffeff4

Ô nhớ tại vùng Stack sau lệnh "sw" vào trong địa chỉ của $sp và $sp +4 (push):

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x7fffefe0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0xfffffffd	0x00000004	0x00000000
0x7ffff000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Ô màu xanh là ô nhớ mang địa chỉ được lưu trong $sp. Cột (+1c) hàng đầu là địa chỉ mà $sp được mặc định đặt.

Sau "push" ta thực hiện "pop", phần tử nào đi vào Stack trước sẽ ra Stack cuối, nên đầu tiên ta sẽ "load" phần tử ở "đỉnh"- tại $sp qua lệnh `lw $s0,0($sp)` và tăng off set lên 4 để có thể pop được phần tử còn lại ra khỏi Stack vào trong thanh $s1. Cuối cùng, ta trả lại giá trị của $sp về với mặc định để chỉ rằng Stack không còn phần tử nào.

Tuy nhiên, lệnh "pop" không thực sự xóa dữ liệu ra khỏi ô nhớ, mà chỉ dịch chuyển linh hoạt giá trị $sp, để khi có phần tử mới được thêm vào Stack, ta có thể dịch $sp, và ghi đè lên dữ liệu cũ.

Một điều đáng lưu ý nữa, Stack được lưu trong bộ nhớ theo kiểu ngược, phần tử đầu tiên sẽ được đưa vào ô có địa chỉ lớn và phần tử nào đưa vào Stack sau sẽ được lưu vào ô có địa chỉ bé hơn, nên ta việc tăng $sp lên và lưu các giá trị sau $sp theo thứ tự tăng dần sẽ phá vỡ kiểu cấu trúc dữ liệu này.

Kết quả: $s0 = -3 = 0xffff fffd, $s1 = 4

Bài 4:

```
.data
	Message: .asciiz "Ket qua tinh giai thua la: "
	Ins: .asciiz "Nhap so N can tinh giai thua: "
.text
main:
	li	$v0,4
```

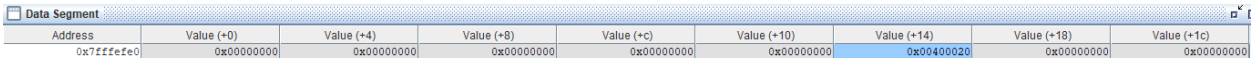
```
        la     $a0,Ins
        syscall
        li     $v0,5
        syscall
        add    $a0,$0,$v0          #load test input N
        jal    WARP
print:
        add    $a1, $v0, $zero     # $a0 = result from N!
        li     $v0, 4
        la     $a0, Message
        syscall
        li     $v0,1
        add    $a0,$0,$a1
        syscall
quit:
        li     $v0, 10                    #terminate
        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)
        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:
```

Chương trình sẽ sử dụng Stack để lưu các biến cục bộ và các địa chỉ để khôi phục vị trí cũ trong các thủ tục được gọi. Cụ thể, sử dụng $fp (frame pointer) để chỉ ra các địa chỉ của các khối thủ tục/ hàm; sử dụng $sp (stack pointer) để chỉ ra địa chỉ của phần đỉnh của Stack và thanh ghi $ra để khôi phục lại các thủ tục/hàm cũ.

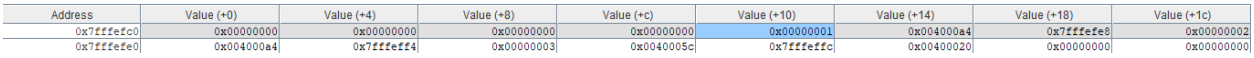
Bắt đầu thủ tục WARP, Stack lưu giá trị của $fp (=0) và giá trị $ra tại câu lệnh sau WARP trong main (tại 0x40020). Vị trí của $sp ban đầu ở 0x7fffeffc (ô cuối cùng bên phải), $fp được lưu trước đó 1 ô và $ra trước $sp 2 ô:


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	0x00400020	0x00000000	0x00000000

Ta lưu giá trị của cũ của $sp vào trong $fp (= 0x7fffeffc), và dịch $sp -8 (lên trước 2 ô). Cập nhật giá trị $ra mang địa chỉ lệnh sau lệnh "jal" đến FACT. Khối đầu tiên ở dưới đáy của Stack sẽ bao gồm các thông tin để chương trình có thể khôi phục lại trạng thái sau khi kết thúc WARP để quay trở lại main.

Ta tiếp tục gọi thủ tục FACT. Giá trị của thanh $fp được lưu tại ô trước ô $sp trỏ vào (trước ô chứa $ra cũ) để khôi phục lại hàm trước. Khi đó ta cập nhật $fp trỏ vào frame của hàm mới (là địa chỉ của ô chứa $ra cũ). Ta bắt đầu dịch chuyển $sp lên 3 ô (-12) để lưu thêm $a0 và $ra mới. Ta đẩy giá trị $a0 vào trong đỉnh của Stack. Chương trình sẽ đệ quy ở hàm "recursive" cho đến khi giá trị $a0 <2, mỗi lần lặp trừ 1 cho giá trị $a0 để lưu lại các giá trị của $a0 bé dần đến 1 vào trong Stack và trong 1 khối có chứa $fp của thủ tục gọi $a0 trước và giá trị $ra trước đó.



Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x7fffefc0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000001	0x004000a4	0x7fffefe8	0x00000002
0x7fffefe0	0x004000a4	0x7fffeff4	0x00000003	0x0040005c	0x7fffeffc	0x00400020	0x00000000	0x00000000

Ta có thể hình dung cấu trúc của Stack qua bảng sau (Trong trường hợp $a0 =3):

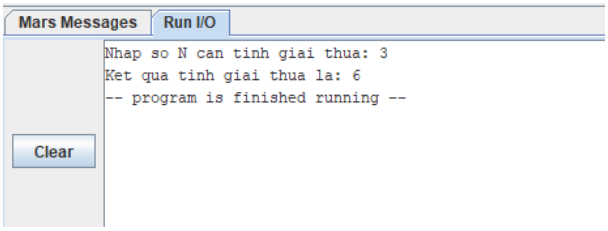
Giá trị		Địa chỉ
$a0 =1		0x7fff efd0
$ra		0x7fff efd4
$fp		0x7fff efd8
$a0 =2	←$fp (old)	0x7fff efdc
$ra		0x7fff efe0
$fp		0x7fff efe4
$a0 =3	←$fp (old)	0x7fff efe8
$ra		0x7fff efec
$fp		0x7fff eff0
$ra	←$fp (old)	0x7fff eff4
$fp (main)		0x7fff eff8
		0x7fff effc

Đối chiếu với bảng giá trị trong Stack ở trên ta có thể thấy được các giá trị địa chỉ $fp cũ được lưu vào đầu của một khối chương trình con mới để có thể khôi phục lại được $sp của thủ tục trước và $ra của chương trình con cũ để có thể nhảy lại về câu lệnh của chương trình con cũ. Trong hàm "recursive" có nhảy lại về FACT để

nhập các $a0 vào trong Stack sử dụng lệnh “jal”, nên $ra sẽ không thay đổi trong thủ tục nhập $a0 =2 và =3, địa chỉ chỉ $ra là lệnh sau lệnh “jal” tới FACT

Kết thúc nhập xong vào Stack, con trỏ $sp đang được trỏ vào $a0 ở đỉnh tại địa chỉ 0x 7fff efd0. So sánh được $a0 <2, ta nhảy sang nhãn “done” để thực hiện lại nhân các thành phần của Stack. Lúc này ta khôi phục $ra để có thể nhảy lại thủ tục con trước, khôi phục lại $a0, gán giá trị $sp = $fp để có thể trở về chương trình con trước, và cuối cùng là gán $fp lại với giá trị $fp để nhảy lần sau. Kết thúc sử dụng lệnh “jr” tới địa chỉ $ra để có thể thực hiện phép nhân $a0 và $v0 vào $v1 trong thủ tục “recursive”, và khi $ra được trả ngược về tới cuối Stack, quay ngược lại về main, và in ra kết quả.

Kết quả:



Bài 5:

```
.data
	max: .asciiz "\nLargest: "
	min: .asciiz "\nSmallest: "
	comma: .asciiz ", "
	temp: .asciiz " "
	A: .space 32
.text
MAIN:
	jal	init
	nop
	jal	STACK
	nop
	add	$s0,$v0,$0
	add	$s1,$v1,$0
	li	$v0,4
```

```
        la     $a0,max
        syscall
        li     $v0,1
        add    $a0,$s0,$0
        syscall
        li     $v0,4
        la     $a0,comma
        syscall
        li     $v0,1
        add    $a0,$t6,$0
        syscall

        li     $v0,4
        la     $a0,min
        syscall
        li     $v0,1
        add    $a0,$s1,$0
        syscall
        li     $v0,4
        la     $a0,comma
        syscall
        li     $v0,1
        add    $a0,$t7,$0
        syscall
quit:
        li     $v0,10
        syscall
end_main:
#-------------------------------------
init:
        li     $v0,0        #result max
```

```
        li      $v1,0           #result min
        li      $s0, 0
        li      $s1, 1
        li      $s2, 100
        li      $s3, -23
        li      $s4, 56
        li      $s5, 700
        li      $s6, -22
        li      $s7, 44
        la      $t0,A
in:
        sw      $s0,0($t0)
        sw      $s1,4($t0)
        sw      $s2,8($t0)
        sw      $s3,12($t0)
        sw      $s4,16($t0)
        sw      $s5,20($t0)
        sw      $s6,24($t0)
        sw      $s7,28($t0)
        li      $t1,0
        li      $t2,8
end_in:
        jr      $ra
end_init:
#-------------------------------------
STACK:
run:
        sll     $t3,$t1,2
        add     $t4,$t0,$t3
        lw      $t5,0($t4)
push:
```

```
        addi   $sp,$sp,-4
        sw     $t5,4($sp)
        nop
        addi   $t1,$t1,1
        bne    $t1,$t2,run
        nop
        li     $t1,0
compare:
        lw     $t8, 0($sp)
        bgt    $t8,$v0,change_max
        blt    $t8,$v1,change_min
continue:
        addi   $t1,$t1,1
        add    $sp,$sp,4
        bne    $t1,$t2,     compare
        jr     $ra
end_cpr:
end_stack:
change_max:
        add    $v0,$t8,$0
        add    $t7,$0,$t1          # index
        j      continue
change_min:
        add    $v1,$t8,$0
        add    $t6,$0,$t1
        j      continue
```

Kết quả: Giá trị $s0 tới $7 = 0,1,100,-23,56,700,-22,44