

# Laboratory Exercise 7 – Report:

## Procedure calls, stack and parameters

Lê Văn Duẩn - 20194508

### 1. Assignment 1

- Mã nguồn:

#Laboratory Exercise 7 Home Assignment 1

.text

main: li \$a0, 20194508 #load input parameter

jal abs #jump and link to abs procedure

nop

add \$s0, \$zero, \$v0

li \$v0,10 #terminate

syscall

endmain:

#-----

# function abs

# param[in] \$a1 the interger need to be gained the absolute value

# return \$v0 absolute value

#-----

abs:

sub \$v0,\$zero,\$a0 #put -(a0) in v0; in case (a0)<0

bltz \$a0,done #if (a0)<0 then done

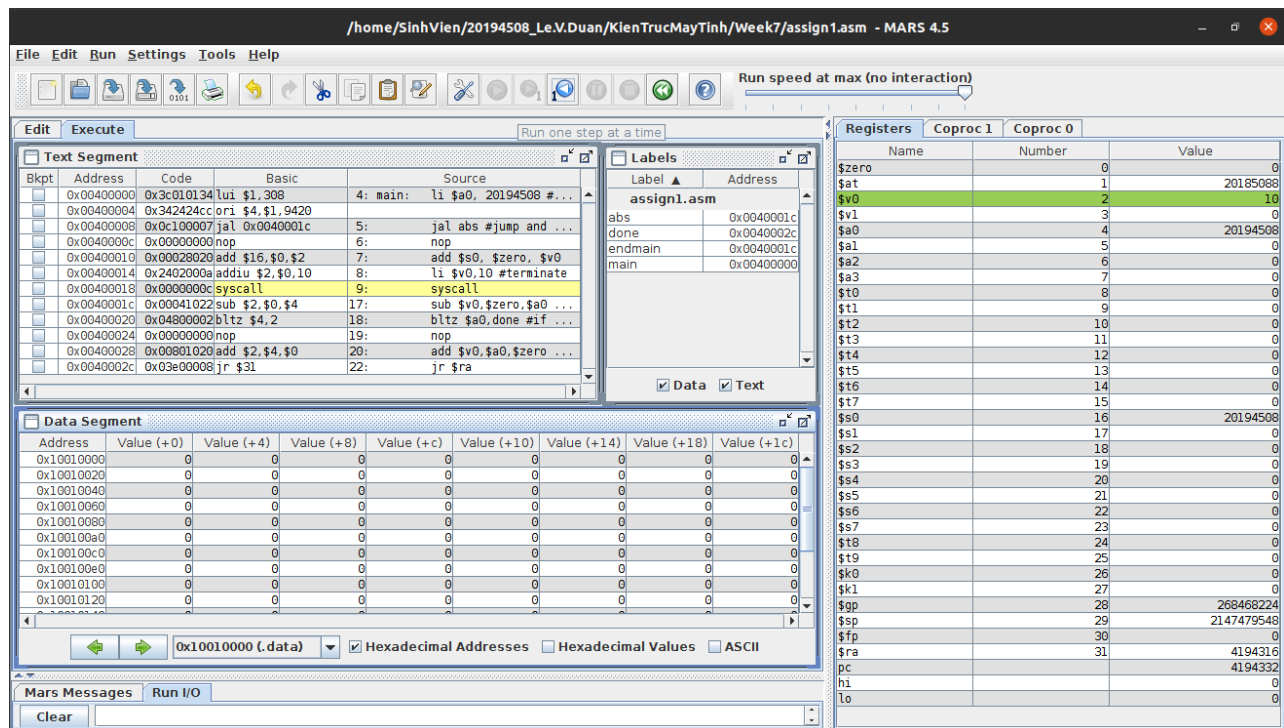
nop

add \$v0,\$a0,\$zero #else put (a0) in v0

done:

jr \$ra

- Kết quả chạy mô phỏng:



- Giải thích:

+ Chương trình nhận tham số 20194508 vào thanh ghi \$a0 và kết quả trị tuyệt đối của nó được lưu vào thanh ghi \$v0

+ lệnh `jal abs` -> nhảy đến thủ tục có nhãn `abs` và lưu thanh ghi `$ra = pc = 0x004000c` (địa chỉ lệnh tiếp theo chính là lệnh `nop` trong chương trình chính), còn `pc` thay đổi từ `0x004008` (địa chỉ lệnh `jal` hiện tại) thành địa chỉ đầu tiên của thủ tục `abs 0x0040001c`

+ sau khi thực hiện xong thủ tục vào lưu kết quả vào `$v0` trong ví dụ này là 20194508 thì thoát thủ tục bằng lệnh `jr $ra` -> lưu lại thanh ghi `pc = $ra` để quay

lại chương trình chính. Nên pc thay đổi từ 0x0040002c(địa chỉ của lệnh jr hiện tại) thành 0x004000c(\$ra)

## 2. Assignment 2

- Mã nguồn:

#Laboratory Exercise 7 Home Assignment 2

.text

main: li \$a0,5 #load test input (20194)508

li \$a1,0

li \$a2,8

jal max #call max procedure

nop

li \$v0,10 #terminate

syscall

endmain:

#-----

#Procedure max: find the largest of three integers

#param[in] \$a0 integers

#param[in] \$a1 integers

#param[in] \$a2 integers

#return \$v0 the largest value

#-----

max: add \$v0,\$a0,\$zero #copy (a0) in v0; largest so far

sub \$t0,\$a1,\$v0 #compute t0 = (a1)-(v0) = (a1) - (a0)

bltz \$t0,okay #if (a1)-(a0)<0 then no change

nop

add \$v0,\$a1,\$zero #else (a1) is largest thus far

okay: sub \$t0,\$a2,\$v0 #compute (a2)-(v0)

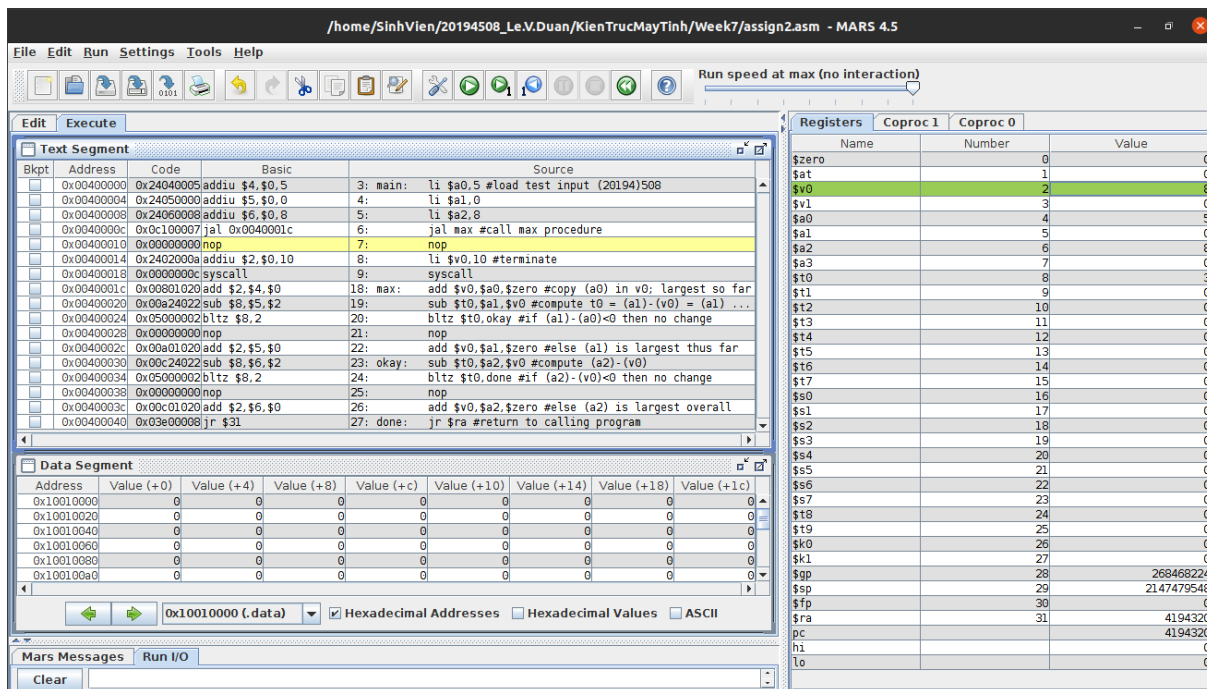
bltz \$t0,done #if (a2)-(v0)<0 then no change

nop

add \$v0,\$a2,\$zero #else (a2) is largest overall

done: jr \$ra #return to calling program

- Kết quả chạy:



- Giải thích:

+ Chương trình nhận 3 tham số đầu vào lưu lần lượt trong 3 thanh ghi \$a0, \$a1, \$a2 tìm max trong 3 số vào lưu vào thanh ghi \$v0

+ lệnh jal max -> nhảy đến thủ tục với nhãn max, thanh ghi \$ra = pc nên \$ra = 0x0040000c là giá trị hiện tại của pc và pc thay đổi từ 0x0040000c thành 0x004001c là địa chỉ của nhãn max

+ Thủ tục max:

add \$v0,\$a0,\$zero -> gán giá trị max tạm thời vào thanh ghi \$v0 = \$a0 = 5

sub \$t0,\$a1,\$v0 -> tính hiệu \$a1 - \$a0 và lưu vào \$t0 nên giá trị là 0 - 5 = -5

bltz \$t0,okay -> nếu \$t0 < 0 -> \$a0 > \$a1 -> \$v0 = \$a0 = 5

nop

add \$v0,\$a1,\$zero -> nếu \$t0 > 0 -> \$a1 > \$a0 -> cập nhật max = \$a0 -> \$v0 = \$a1  
nhưng trong ví dụ này ko xảy ra nên \$v0 không thay đổi

okay: sub \$t0,\$a2,\$v0 -> tương tự xét hiệu của max hiện tại với \$a2

bltz \$t0,done -> nếu \$t < 0 thì max hiện tại chính là max trong 3 số -> done

nop

add \$v0,\$a2,\$zero -> ngược lại \$t0 > 0 -> \$a2 > max hiện tại trong \$v0 đúng trong  
ví dụ là 8 > 5 nên thanh ghi \$v0 được cập nhật là giá trị của \$a2 là 8

done: jr \$ra -> pc = \$ra để trở lại chương trình chính nên nó thay đổi từ 0x40040  
(địa chỉ lệnh jr hiện tại) thành 0x400010 (địa chỉ tiếp theo hay chính là lệnh nop  
trên chương trình chính)

+ kết quả: trước khi gán lại \$v0 = 10 để kết thúc chương trình thì \$v0 = 8 đúng với  
kết quả mong muốn

### 3. Assignment 3

-Mã nguồn:

#Laboratory Exercise 7, Home Assignment 3

.text

push: li \$s0, 2019

li \$s1, 4508

addi \$sp,\$sp,-8 #adjust the stack pointer

sw \$s0,0(\$sp) #push \$s0 to stack

sw \$s1,4(\$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

- Kết quả chạy mô phỏng:

/home/SinhVien/20194508\_Le.V.Duan/KienTrucMayTinh/Week7/assign3.asm - MARS 4.5

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x241007e3	addiu \$t6,\$0,2019	3: push: li \$s0, 2019
	0x00400004	0x2411119c	addiu \$t7,\$0,4508	4: li \$s1, 4508
	0x00400008	0x23b0ffff	addi \$sp,\$sp,-8	5: addi \$sp,\$sp,-8 #adjust the stack pointer
	0x0040000c	0xafb00000	sw \$t6,0(\$sp)	6: sw \$s0,0(\$sp) #push \$s0 to stack
	0x00400010	0xafb10004	sw \$t7,4(\$sp)	7: sw \$s1,4(\$sp) #push \$s1 to stack
	0x00400014	0x00000000	nop	8: work: nop
	0x00400018	0x00000000	nop	9: nop
	0x0040001c	0x00000000	nop	10: nop
	0x00400020	0x8fb00000	lw \$t6,0(\$sp)	11: pop: lw \$s0,0(\$sp) #pop from stack to \$s0
	0x00400024	0x8fb10004	lw \$t7,4(\$sp)	12: lw \$s1,4(\$sp) #pop from stack to \$s1
	0x00400028	0x23b00008	addi \$sp,\$sp,8	13: addi \$sp,\$sp,8 #adjust the stack pointer

Data Segment

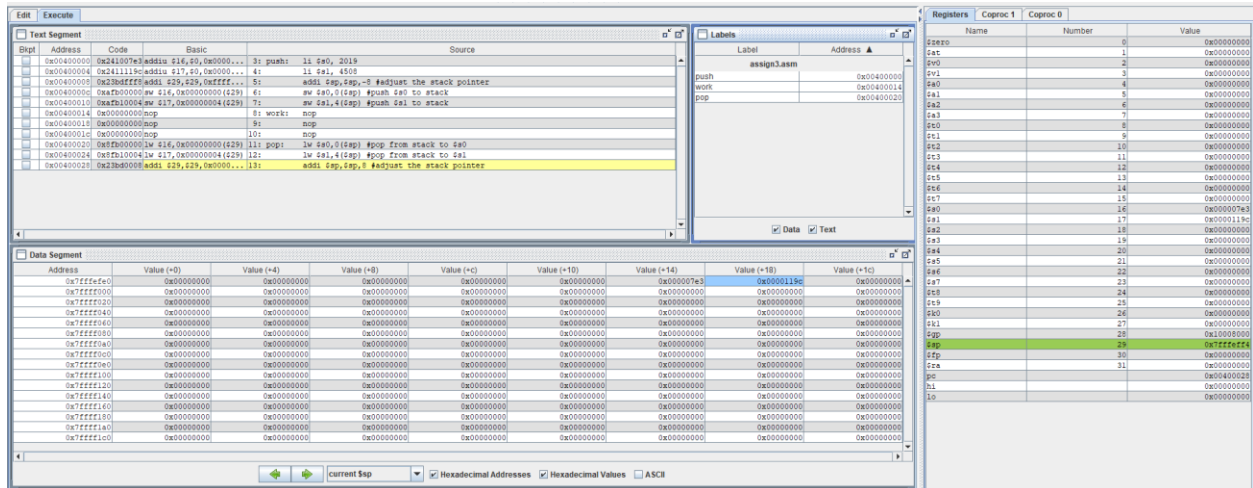
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x7ffffef0	0	0	0	0	0	2019	4508	0
0x7fffff00	0	0	0	0	0	0	0	0
0x7ffff020	0	0	0	0	0	0	0	0
0x7ffff040	0	0	0	0	0	0	0	0
0x7ffff060	0	0	0	0	0	0	0	0
0x7ffff080	0	0	0	0	0	0	0	0

Registers

Name	Number	Value
\$zero	0	0
\$at	1	0
\$v0	2	0
\$v1	3	0
\$a0	4	0
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	0
\$t1	9	0
\$t2	10	0
\$t3	11	0
\$t4	12	0
\$t5	13	0
\$t6	14	0
\$t7	15	0
\$s0	16	2019
\$s1	17	4508
\$s2	18	0
\$s3	19	0
\$s4	20	0
\$s5	21	0
\$s6	22	0
\$s7	23	0
\$t8	24	0
\$t9	25	0
\$k0	26	0
\$k1	27	0
\$gp	28	268468224
\$sp	29	2147479548
\$fp	30	0
\$ra	31	0
pc		4194348
hi		0
lo		0

Mars Messages Run I/O

Clear



- Giải thích:

+ Tạo ngăn xếp để push và pop 2 tham số vào 2019, 4508 được lưu vào các thanh ghi \$s0, \$s1

+ addi \$sp,\$sp,-8 -> thanh ghi \$sp lùi 8 byte để đủ chỗ cho 2 phần tử của stack và vì stack có đáy ở dưới và chiều địa chỉ từ trên xuống nên phải là -8. \$sp cũng đóng vai trò chứa địa chỉ của các phần tử trên cùng của ngăn xếp.

-> giá trị \$sp thay đổi từ giá trị ban đầu 0x7ffefffc thành 0x7ffefff4

+ 2 lệnh sw để push lần lượt vào thanh ghi nên giá trị sẽ được lưu và hiển thị trong data segment ở địa chỉ tương ứng như trong hình.

+ pop phần tử ra khỏi stack bằng cách lw phần tử và điều chỉnh con trỏ \$sp như trong ví dụ chúng ta pop cả 2 phần tử nên phải +8 (vì kích thước phần tử là 4byte)

-> \$sp = 0x7ffefffc chính là địa chỉ ban đầu -> stack rỗng

## 4. Assignment 4

- Mã nguồn:

#Laboratory Exercise 7 Home 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 #terminate
      syscall

endmain:

#-----
#Procedure WARP: assign value and call FACT
#-----

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 $a0,8 #load test input N = 8
      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:

#-----

```



#Procedure FACT: compute N!

#param[in] \$a0 integer N

#return \$v0 the largest value

#-----

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 ( $a0 = N \geq 2$ )

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:

- Kết quả chạy:

Test Segment

Dispt	Address	Code	Basic	Source
0x00400000	0x00400000	0x00400000	0x00400000	6: main: jai WARP
0x00400001	0x00400001	0x00400001	0x00400001	7: print: addi \$a1, \$v0, \$zero # \$a0 = result from \$!
0x00400002	0x00400002	0x00400002	0x00400002	8: j1 \$v0, \$4
0x00400003	0x00400003	0x00400003	0x00400003	9: j1 \$a0, \$message
0x00400004	0x00400004	0x00400004	0x00400004	10: syscall
0x00400005	0x00400005	0x00400005	0x00400005	11: quit: j1 \$v0, 10 #terminate
0x00400006	0x00400006	0x00400006	0x00400006	12: syscall
0x00400007	0x00400007	0x00400007	0x00400007	13: WARP: sw \$fp,-4(\$sp) #save frame pointer (\$)
0x00400008	0x00400008	0x00400008	0x00400008	14: addi \$fp,\$sp,0 #new frame pointer point to the top (\$)
0x00400009	0x00400009	0x00400009	0x00400009	15: sw \$ra,\$(\$sp) #save return address (\$)
0x0040000A	0x0040000A	0x0040000A	0x0040000A	16: sw \$ra,\$(\$sp) #save return address (\$)
0x0040000B	0x0040000B	0x0040000B	0x0040000B	17: li \$a0,0 #load test input N = 0
0x0040000C	0x0040000C	0x0040000C	0x0040000C	18: jai FACT #call fact procedure
0x0040000D	0x0040000D	0x0040000D	0x0040000D	19: nop
0x0040000E	0x0040000E	0x0040000E	0x0040000E	20: lw \$ra,\$(\$sp) #restore return address (\$)
0x0040000F	0x0040000F	0x0040000F	0x0040000F	21: addi \$fp,\$fp,0 #return stack pointer (\$)
0x00400010	0x00400010	0x00400010	0x00400010	22: jw \$fp,-4(\$sp) #return frame pointer (\$)
0x00400011	0x00400011	0x00400011	0x00400011	23: sw \$ra,-4(\$sp) #save frame pointer
0x00400012	0x00400012	0x00400012	0x00400012	24: addi \$fp,\$sp,0 #new frame pointer point to stack(\$)
0x00400013	0x00400013	0x00400013	0x00400013	25: sw \$ra,\$(\$sp) #save return address
0x00400014	0x00400014	0x00400014	0x00400014	26: sw \$a0,\$(\$sp) #save \$a0 register
0x00400015	0x00400015	0x00400015	0x00400015	27: li \$t0,\$(\$sp),2 #if input argument N < 2
0x00400016	0x00400016	0x00400016	0x00400016	28: beq \$t0,\$zero,#recursive if it is false (\$a0 = 0) =>2
0x00400017	0x00400017	0x00400017	0x00400017	29: nop
0x00400018	0x00400018	0x00400018	0x00400018	30: li \$v0,1 #return the result N!=1
0x00400019	0x00400019	0x00400019	0x00400019	31: j done
0x0040001A	0x0040001A	0x0040001A	0x0040001A	32: nop
0x0040001B	0x0040001B	0x0040001B	0x0040001B	33: addi \$a1,\$a0,-1 #adjust input argument
0x0040001C	0x0040001C	0x0040001C	0x0040001C	34: jai FACT #recursive call
0x0040001D	0x0040001D	0x0040001D	0x0040001D	35: nop
0x0040001E	0x0040001E	0x0040001E	0x0040001E	36: lw \$v1,\$(\$sp) #load \$0
0x0040001F	0x0040001F	0x0040001F	0x0040001F	37: mult \$v1,\$v1 #compute the result
0x00400020	0x00400020	0x00400020	0x00400020	38: mflo \$v0
0x00400021	0x00400021	0x00400021	0x00400021	39: done: jw \$ra,\$(\$sp) #restore return address
0x00400022	0x00400022	0x00400022	0x00400022	40: lw \$a0,\$(\$sp) #restore stack pointer
0x00400023	0x00400023	0x00400023	0x00400023	41: addi \$fp,\$fp,0 #restore frame pointer
0x00400024	0x00400024	0x00400024	0x00400024	42: jw \$fp,-4(\$sp) #return frame pointer
0x00400025	0x00400025	0x00400025	0x00400025	43: jr \$ra #jump to calling

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+C)	Value (+10)	Value (+14)	Value (+18)	Value (+1C)
0x00100000	0x00744950	0x00617954	0x00617954	0x00617954	0x00617954	0x00617954	0x00617954	0x00617954
0x00100004	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00100008	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x0010000C	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00100010	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00100014	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00100018	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x0010001C	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00100020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Registers

Coproc 1

Coproc 0

Name	Number	Value
\$R0	0	0x00000000
\$a1	1	0x00000000
\$a2	2	0x00000000
\$a3	3	0x00000000
\$a4	4	0x00000000
\$a5	5	0x00000000
\$a6	6	0x00000000
\$a7	7	0x00000000
\$a8	8	0x00000000
\$a9	9	0x00000000
\$a10	10	0x00000000
\$a11	11	0x00000000
\$a12	12	0x00000000
\$a13	13	0x00000000
\$a14	14	0x00000000
\$a15	15	0x00000000
\$a16	16	0x00000000
\$a17	17	0x00000000
\$a18	18	0x00000000
\$a19	19	0x00000000
\$a20	20	0x00000000
\$a21	21	0x00000000
\$a22	22	0x00000000
\$a23	23	0x00000000
\$a24	24	0x00000000
\$a25	25	0x00000000
\$a26	26	0x00000000
\$a27	27	0x00000000
\$a28	28	0x00000000
\$a29	29	0x00000000
\$a30	30	0x00000000
\$a31	31	0x00000000
\$v0	32	0x00000000
\$v1	33	0x00000000
\$t0	34	0x00000000
\$t1	35	0x00000000
\$t2	36	0x00000000
\$t3	37	0x00000000
\$t4	38	0x00000000
\$t5	39	0x00000000
\$t6	40	0x00000000
\$t7	41	0x00000000
\$t8	42	0x00000000
\$t9	43	0x00000000
\$s0	44	0x00000000
\$s1	45	0x00000000
\$s2	46	0x00000000
\$s3	47	0x00000000
\$s4	48	0x00000000
\$s5	49	0x00000000
\$s6	50	0x00000000
\$s7	51	0x00000000
\$s8	52	0x00000000
\$s9	53	0x00000000
\$s10	54	0x00000000
\$s11	55	0x00000000
\$s12	56	0x00000000
\$s13	57	0x00000000
\$s14	58	0x00000000
\$s15	59	0x00000000
\$s16	60	0x00000000
\$s17	61	0x00000000
\$s18	62	0x00000000
\$s19	63	0x00000000
\$s20	64	0x00000000
\$s21	65	0x00000000
\$s22	66	0x00000000
\$s23	67	0x00000000
\$s24	68	0x00000000
\$s25	69	0x00000000
\$s26	70	0x00000000
\$s27	71	0x00000000
\$s28	72	0x00000000
\$s29	73	0x00000000
\$s30	74	0x00000000
\$s31	75	0x00000000
\$k0	76	0x00000000
\$k1	77	0x00000000
\$k2	78	0x00000000
\$k3	79	0x00000000
\$k4	80	0x00000000
\$k5	81	0x00000000
\$k6	82	0x00000000
\$k7	83	0x00000000
\$k8	84	0x00000000
\$k9	85	0x00000000
\$ka	86	0x00000000
\$kb	87	0x00000000
\$kc	88	0x00000000
\$kd	89	0x00000000
\$ke	90	0x00000000
\$kf	91	0x00000000
\$kg	92	0x00000000
\$kh	93	0x00000000
\$ki	94	0x00000000
\$kj	95	0x00000000
\$kk	96	0x00000000
\$kl	97	0x00000000
\$km	98	0x00000000
\$kn	99	0x00000000
\$ko	100	0x00000000
\$kp	101	0x00000000
\$kq	102	0x00000000
\$kr	103	0x00000000
\$ks	104	0x00000000
\$kt	105	0x00000000
\$ku	106	0x00000000
\$kv	107	0x00000000
\$kw	108	0x00000000
\$kx	109	0x00000000
\$ky	110	0x00000000
\$kz	111	0x00000000
\$t0	112	0x00000000
\$t1	113	0x00000000
\$t2	114	0x00000000
\$t3	115	0x00000000
\$t4	116	0x00000000
\$t5	117	0x00000000
\$t6	118	0x00000000
\$t7	119	0x00000000
\$t8	120	0x00000000
\$t9	121	0x00000000
\$t10	122	0x00000000
\$t11	123	0x00000000
\$t12	124	0x00000000
\$t13	125	0x00000000
\$t14	126	0x00000000
\$t15	127	0x00000000

0x10010000 (Data)

☒ Hexadecimal Addresses

☒ Hexadecimal Values

☐ ASCII

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

FileEditExecute

Text Segment

DisptAddressCodeBasicSource

0x004000000x00400000jal 0x0040000206: main: jal WARP

0x004000040x00402020addi \$t2,\$0,207: print: addi \$a1,\$v0,\$zero # \$a0 = result from \$!

0x004000100x00402030addiu \$t2,\$t2,\$454: j1 \$v0,\$4

0x0040000c0x00110101jui \$t1,40979: j1 \$a0,\$message

0x004000140x00404000ori \$t4,\$0,1010: syscall

0x004000180x00000000rrwrtall

0x004000110x00402000addiu \$t2,\$0,1011: quit: j1 \$v0,10 #terminate

0x004000160x00000000rrwrtall

0x004000200x0040fffffw \$t0,-4(\$t2)13: WARP: sw \$fp,-4(\$sp) #save frame pointer (\$)

0x004000240x003be00000addi \$t0,\$t0,\$0,16: addi \$fp,\$sp,0 #new frame pointer point to the top (\$)

0x004000280x003be0fffffw \$t0,\$t0,-815: sw \$ra,\$(\$sp) #save return address (\$)

0x004000320x0040f00000sw \$t1,\$t1,\$t220: sw \$ra,\$(\$sp) #save return address (\$)

0x004000360x00404000addiu \$t4,\$0,\$0,21: li \$a0,0 #load test input N = 0

0x0040003a0x00110101jal 0x0040400422: jai FACT #call fact procedure

0x0040003c0x00000000nop23: nop

0x0040003e0x0040f00000lw \$t1,\$t1,\$t224: lw \$ra,\$(\$sp) #restore return address (\$)

0x004000400x003be00000addiu \$t2,\$t2,\$0,25: addi \$fp,\$sp,0 #return stack pointer (\$)

0x004000440x0040fffffw \$t0,-4(\$t2)26: lw \$fp,-4(\$sp) #return frame pointer (\$)

0x004000480x003be0fffffw \$t0,\$t0,-827: sw \$ra,\$(\$sp) #save frame pointer

0x0040004c0x0040fffffw \$t0,\$t0,-828: addi \$fp,\$sp,0 #new frame pointer point to stack(\$)

0x004000500x003be0fffffw \$t0,\$t0,-1229: sw \$ra,\$(\$sp) #save return address

0x004000540x0040f00000sw \$t1,\$t1,\$t230: sw \$ra,\$(\$sp) #save return address

0x004000580x0040f00000sw \$t1,\$t1,\$t231: sw \$a0,\$(\$sp) #save \$a0 register

0x0040005c0x0040f00000sw \$t1,\$t1,\$t232: li \$t0,\$(\$sp),2 #if input argument N < 2

0x004000600x01000004beq \$t0,\$0,440: beq \$t0,\$zero,#recursive if it is false (\$a0 = 0) =>2

0x004000640x00000000nop41: nop

0x004000680x00402001addiu \$t2,\$0,142: li \$v0,1 #return the result N!=1

0x004000700x00400024j 0x0040009043: j done

0x004000740x00000000nop44: nop

0x004000780x002040fffffw \$t4,\$4,-146: addi \$a1,\$a0,-1 #adjust input argument

0x0040007c0x00110101jal 0x0040400447: jai FACT #recursive call

0x004000800x00000000nop48: nop

0x004000840x00403000lw \$t1,\$t1,\$t249: lw \$v1,\$(\$sp) #load \$0

0x004000880x0040f00000mul \$t2,\$t2,\$t250: mult \$v1,\$v1 #compute the result

0x0040008c0x00000101mflo \$t251: mflo \$v0

0x004000900x0040f00000lw \$t1,\$t1,\$t252: done: lw \$ra,\$(\$sp) #restore return address

0x004000940x0040f00000lw \$t1,\$t1,\$t253: lw \$a0,\$(\$sp) #restore stack pointer

0x004000980x003be00000addi \$t2,\$t2,\$0,54: addi \$fp,\$sp,0 #restore frame pointer

0x0040009c0x0040fffffw \$t0,-4(\$t2)55: jr \$ra #jump to calling

Data Segment

AddressValue (+0)Value (+4)Value (+8)Value (+C)Value (+10)Value (+14)Value (+18)Value (+1C)

0x7ffffefc21474795040419433221474795160419433221474795200

0x7ffffef8419433221474795040041943302147479540419433000

0x7ffffef4000000000000000000000000000000000000000000

0x7ffffef0000000000000000000000000000000000000000000

0x7ffffee4000000000000000000000000000000000000000000

0x7ffffee0000000000000000000000000000000000000000000

0x7ffffed4000000000000000000000000000000000000000000

0x7ffffed0000000000000000000000000000000000000000000

0x7ffffec4000000000000000000000000000000000000000000

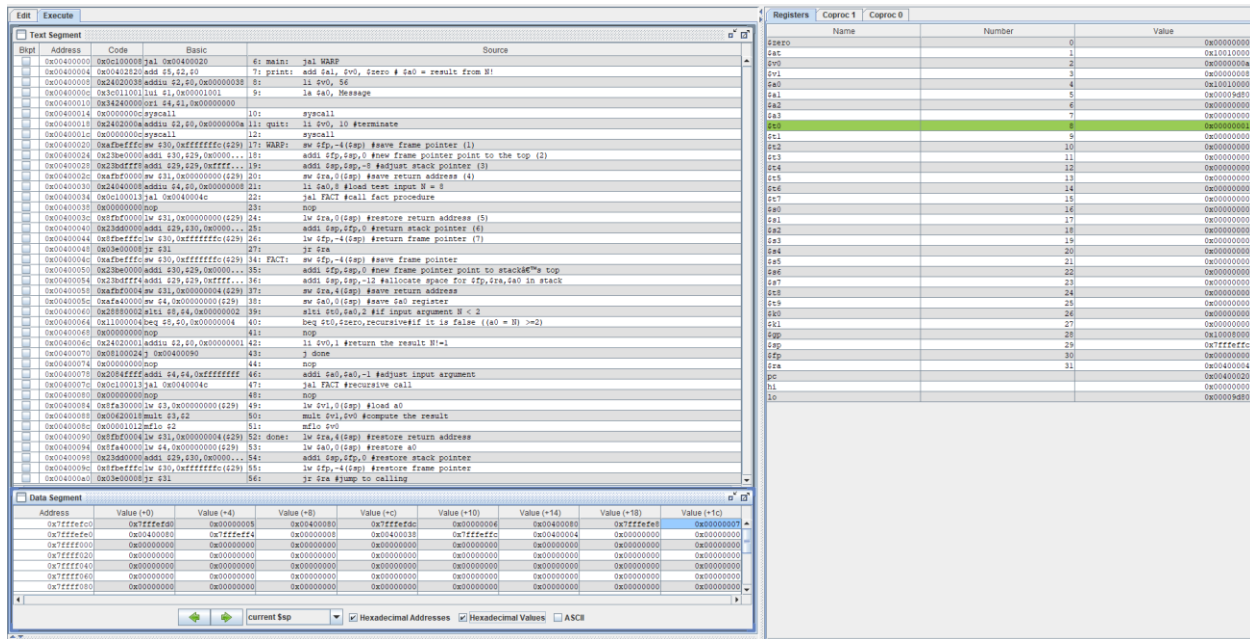
0x7ffffec0000000000000000000000000000000000000000000

Current \$sp

Hexadecimal Addresses

Hexadecimal Values

ASCII



- Giải thích:

+ thanh ghi \$fp lưu con trỏ đến khung trang frame pointer, \$sp – stack pointer

+ thanh ghi \$a0 lưu kết quả của chương trình

+ thủ tục WRAP:

- Khai báo 2 vị trí trong stack cho khung trang và địa chỉ return -> \$fp, \$ra

-> vì thế nên địa chỉ thanh và địa chỉ thanh return nên giá trị thanh ghi \$fp thanh đổi sang 0x7ffefffc, vì có thủ tục lồng nhau nên ta dùng lưu địa chỉ \$ra ra bộ nhớ ở địa chỉ 0(\$sp) bằng lệnh sw -> hiển thị trong data segment

- Lưu giá trị N vào thanh ghi \$a0 nên \$a0 = 0x00000008

- gọi thủ tục FACT

- Restore giá trị từ 0(\$sp) vào lại \$ra để gọi lệnh jr \$ra trở về chính xác địa chỉ lệnh tiếp theo trong chương trình chính

+ Thủ tục FACT:

- Khai báo 3 vị trí trong stack trong cho khung trang và các giá trị trả về \$fp, \$ra, \$a0 -> \$sp lùi 12 giá trị -> \$sp = 0x7ffeff4 -> 0x7ffefe8, cũng như WRAP lưu giá trị lần lượt vào stack

- Xét số đưa vào <2 -> kết quả = 1

- nếu  $\geq 2$  đi đến thủ tục RECURSIVE

- Nhảy đến nhãn done để restore lại các giá trị return vào thanh ghi \$ra để lệnh jr \$ra được thực thi chính xác

+ Giá trị thay đổi của các thanh ghi:

Step	\$pc	Giá trị thanh ghi thay đổi
1	0x00400020	\$ra = 0x00400004
2	0x00400024	\$ra = 0x00400004
3	0x00400028	\$fp = 0x7ffeffc
4	0x0040002c	\$sp = 0x7ffeff4
5	0x00400030	\$sp = 0x7ffeff4
6	0x00400034	\$a0 = 0x00000000
7	0x0040004c	\$ra = 0x00000038
8	0x00400050	\$ra = 0x00000038
9	0x00400054	\$fp = 0x7ffeff4
10	0x00400058	\$sp = 0x7ffeff8
11	0x00400064	\$t0 = 0x00000001
12	0x00400070	\$v0 = 0x00000001
13	0x00400094	\$ra = 0x00000038
14	0x00400098	\$a0 = 0x00000000
15	0x0040009c	\$fp = 0x7ffeff4
16	0x004000a0	\$fp = 0x7ffeffc
Recurse ...	...	...

## 5. Assignment 5

- Mã nguồn:

#Laboratory Exercise 7 Home Assignment 5

.data

message\_max: .asciiz "Largest: "

message\_index: .asciiz ", "

message\_min: .asciiz "Smallest: "

enter: .asciiz "\n"

.text

load\_data: li \$s0, 2 #MSSV: 20194508

li \$s1, 0

li \$s2, 1

li \$s3, -9

li \$s4, 4

li \$s5, -5

li \$s6, 0

li \$s7, 8

li \$t3, 8 # bien dem nguoc lai vi push vao stack thu tu bi dao nguoc

li \$t4, 8 # bien dem nguoc lai vi push vao stack thu tu bi dao nguoc

main: jal push

nop

jal max

nop

jal min

nop

print\_max: li \$v0, 4 # code for print to console

la \$a0, message\_max

syscall

li \$v0, 1

add \$a0, \$zero, \$a1 # \$a1 = max

syscall

```

        li $v0, 4
        la $a0, message_index
        syscall

        li $v0, 1
        add $a0, $zero, $t1 # $t1 -> register containing the max value
        syscall

        li $v0, 4
        la $a0, enter
        syscall

print_min: li $v0, 4
        la $a0, message_min
        syscall

        li $v0, 1
        add $a0, $zero, $a2 # $t3 = min
        syscall

        li $v0, 4
        la $a0, message_index
        syscall

        li $v0, 1
        add $a0, $0, $t2 # $t4 -> register containing the min value
        syscall

exit:   li $v0, 10
        syscall

end_main:

```

#-----

push: addi \$sp, \$sp, -32 # stack chua 8 ptu

sw \$s0, 28(\$sp)

sw \$s1, 24(\$sp)

sw \$s2, 20(\$sp)

sw \$s3, 16(\$sp)

sw \$s4, 12(\$sp)

sw \$s5, 8(\$sp)

sw \$s6, 4(\$sp)

sw \$s7, 0(\$sp)

push\_end: jr \$ra

#-----

max: lw \$v1, 0(\$sp) # pop phan tu dau tien vao \$v1

addi \$sp, \$sp, 4

addi \$t3, \$t3, -1 # index -1, giam sau moi lan lap

beq \$t3, 0, max\_end #kiem tra xem da den phan tu cuoi hay chua

slt \$t0, \$v1, \$a1 # \$a1 luu gia tri max hien tai

bne \$t0, \$zero, max

add \$t1, \$zero, \$t3 #luu index cua max

add \$a1, \$zero, \$v1 #luu max vao tu \$v1 sang \$t5

j max

max\_end: add \$sp, \$sp, -32 #adjust stack pointer to the top of stack

jr \$ra

#-----

min: lw \$v1, 0(\$sp) #pop tu stack

addi \$sp, \$sp, 4

addi \$t4, \$t4, -1 # \$t4 -> index = 1, tang sau moi lan lap

beq \$t4, 0, min\_end # khi stack rong -> min\_end

slt \$t0, \$a2, \$v1 # so sanh \$1 dang chua min hien tai voi \$t0 la ptu hien tai

bne \$t0, \$zero, min

add \$t2, \$zero, \$t4 # luu index cua min

add \$a2, \$zero, \$v1 # luu gia tri min

j min

min\_end: add \$sp, \$sp, -32 #adjust stack pointer to the top of stack

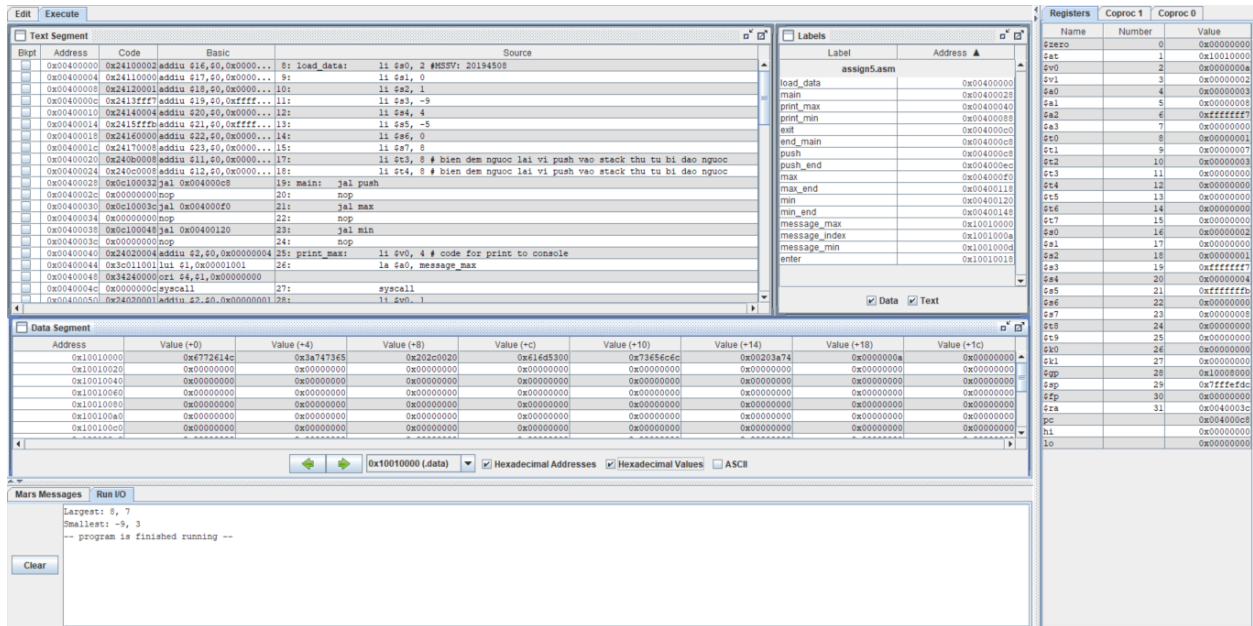
jr \$ra

- Kết quả chạy mô phỏng:

The screenshot displays a MIPS simulator interface with several panels:

- Text Segment:** A table of assembly instructions with columns for Btpt, Address, Code, Basic, and Source. The code includes instructions for loading data, adjusting the stack pointer, comparing values, and jumping to min\_end.
- Labels:** A table mapping labels to addresses, including 'assign5.asm', 'load\_data', 'main', 'print\_max', 'print\_min', 'end', 'end\_main', 'push', 'push\_end', 'max', 'max\_end', 'min', 'min\_end', 'message\_max', 'message\_min', and 'enter'.
- Registers:** A table showing the state of registers, including \$zero, \$at, \$v0, \$v1, \$a0, \$a1, \$a2, \$a3, \$a4, \$a5, \$a6, \$a7, \$a8, \$a9, \$d0, \$d1, \$d2, \$d3, \$d4, \$d5, \$d6, \$d7, \$d8, \$d9, \$t0, \$t1, \$t2, \$t3, \$t4, \$t5, \$t6, \$t7, \$t8, \$t9, \$s0, \$s1, \$s2, \$s3, \$s4, \$s5, \$s6, \$s7, \$s8, \$s9, \$k0, \$k1, \$k2, \$k3, \$k4, \$k5, \$k6, \$k7, \$k8, \$k9, \$ra, \$pc, and \$sp.
- Data Segment:** A table showing the state of data memory, including addresses and values for various data words.
- Messages:** A panel at the bottom showing the output of the program, including the message "program is finished running --".





- Giải thích:

+ Chương trình tìm max, min và vị trí thanh ghi lưu trữ nó trong dãy số được lưu lần lượt vào lần lượt thanh ghi \$s0 -> \$s7 vì thế giá trị các thanh ghi thay đổi từ 0x00000000 sang giá trị tương ứng. Thanh ghi \$a1, \$a2 lưu max và min, thanh ghi \$t1, \$t2 lưu index max và min tương ứng.

+ Thanh ghi \$t3, \$t4 lưu giá trị biến index để đếm sau các vòng lặp để tìm min, max. Vì dùng stack nên khi push vào thứ tự bị đảo ngược nên index cũng đếm lùi và vì có 8 phần tử nên gán giá trị ban đầu là 8 nên giá trị 2 thanh ghi đều là 0x00000008

+ Thủ tục main: - nhảy đến thủ tục push để lưu các giá trị dãy số vào lần lượt

- nhảy đến thủ tục max để tìm max

- nhảy đến thủ tục min để tìm min

+ Thủ tục max: - pop từ stack vào lưu vào \$v1 nên giá trị sẽ thay đổi trở thành giá trị của phần tử pop ra, và giá trị của \$sp tăng lên 4 để trở đến phần tử top stack

- giảm biến đếm bằng thanh ghi \$t3

- kiểm tra kết thúc bằng \$t3 khi nó đếm về 0 -> hết dãy

- so sánh giá trị \$v1 (giá trị phần tử trong dãy vừa pop ra) và \$a0 (giá trị max hiện tại) để xem có cập nhật max là \$a0 hay tiếp tục lặp

- sau khi kết thúc hàm thì đưa con trỏ stack về lại với dãy ban đầu nên -32 giá trị -> thay đổi từ 0x7ffefdc tăng dần 4 đơn vị sau mỗi lần lặp và khi kết thúc -32 thì nó trở lại địa chỉ ban đầu trước khi pop là 0x7ffefdc

+ thủ tục min: hoạt động tương tự max như thay vào đó là sự tham gia của thanh ghi \$a2 lưu min, \$t4 lưu index

-> vì các thủ tục không lồng nhau nên thanh ghi \$ra ko phải lưu lại ra bộ nhớ để restore nên nó thay đổi theo địa chỉ con trỏ pc.

Cụ thể:

Lệnh	Thanh ghi \$ra	Giá trị sau khi thực thi
jal push	0x00000000	0x0040002c = pc địa chỉ lệnh đầu thủ tục push
jal max	0x0040002c	0x00400034 = pc địa chỉ lệnh đầu thủ tục max
jal min	0x00400034	0x0040003c = pc địa chỉ lệnh đầu thủ tục min