

Week 7

Assignment 1

Source code and explanation:

#Laboratory Week 7, Assignment 1

```
.text

main:
    li $a0, -45 #init input
    jal abs      #jump and link to procedure (assign $ra to current
program counter value)
    nop

abs:
    sub $t0, $zero, $a0    #inverse $a0 value
    bltz $a0, done         #if $a0 < 0 then go to 'done'
    nop
    add $t0, $a0, $zero    #else set $t0 to $a0

done:
    jr $ra              #jump back to after jal opcode
```

Run results:

Case 1:

Input: -45 (0xfffffd3)

Output: 45 (0x0000002d)

Case 2:

Input: 45 (0x0000002d)

Output: 45 (0x0000002d)

The screenshot displays a MIPS assembly editor on the left and a register window on the right.

Assembly Editor:

```
1  #Laboratory Week 7, Assignment 1
2
3  .text
4
5  main:
6      li $a0, -45    #init input
7      jal abs        #jump and link to procedure (assign $ra to current p
8      nop
9
10 abs:
11     sub $t0, $zero, $a0    #inverse $a0 value
12     bltz $a0, done         #if $a0 < 0 then go to 'done'
13     nop
14     add $t0, $a0, $zero    #else set $t0 to $a0
15
16 done:
17     jr $ra              #jump back to after jal opcode
18
```

Register Window:

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0xfffffd3
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x0000002d
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10000000
\$sp	29	0x7ffffc
\$fp	30	0x00000000
\$ra	31	0x00400008
pc		0x0040001c
hi		0x00000000

The screenshot shows the MARS MIPS simulator interface. On the left, the assembly code for 'asm1.asm' is displayed, starting with a comment '#Laboratory Week 7, Assignment 1'. The code includes a 'main' procedure that initializes \$a0 to 45, jumps to 'abs', and then a 'done' label. The 'abs' procedure calculates the absolute value of \$a0 using a loop. On the right, the 'Registers' window shows the state of MIPS registers. Register \$t0 is highlighted with a green background and contains the value 8. Other registers like \$zero, \$at, \$v0, \$v1, \$a0, \$a1, \$a2, \$a3, \$s0, \$s1, \$s2, \$s3, \$s4, \$s5, \$s6, \$s7, \$t1, \$t2, \$t3, \$t4, \$t5, \$t6, \$t7, \$gp, \$sp, \$fp, \$ra, and \$h1 are also visible with their respective values.

Assignment 2

Source code and explanation

#Laboratory Week 7, Assignment 2

```
.text
```

```
main:
```

```
    li $s0, 5    #Init test value
    li $s1, 4
    li $s2, 1
    jal max
    nop
```

```
endmain:
```

```
max:
```

```
    add $t0, $s0, $zero    #$t0 stores current largest integer
    sub $t1, $s1, $t0
    bltz $t1, continue    #if ($s1 - $t0) < 0 then continue
    nop
    add $t0, $s1, $zero    #else update current largest integer
```

```
continue:
```

```
    sub $t1, $s2, $t0
    bltz $t1, endmax    #if ($s2 - $t0) < 0 then end procedure
    nop
    add $t0, $s2, $zero    #else update current largest integer
```

```
endmax:
```

```
    jr $ra    #return back to main
```

Run results:

The screenshot shows the MARS MIPS simulator. The assembly code in the main window is as follows:

```

1  #Laboratory Week 7, Assignment 2
2  |
3  .text
4  |
5  main:
6      li $s0, 5      #Init test value
7      li $s1, 4
8      li $s2, 1
9      jal max
10     nop
11 endmain:
12 |
13 max:
14     add $t0, $s0, $zero    #$t0 stores current largest integer
15     sub $t1, $s1, $t0
16     bltz $t1, continue    #if ($s1 - $t0) < 0 then continue
17     nop
18     add $t0, $s1, $zero    #else update current largest integer
19 |
20 continue:
21     sub $t1, $s2, $t0
22     bltz $t1, endmax      #if ($s2 - $t0) < 0 then end procedure
23     nop
24     add $t0, $s2, $zero    #else update current largest integer

```

The register window on the right shows the state of the MIPS registers. The \$t0 register is highlighted in green, indicating it contains the result of the execution, which is 8.

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000008
\$t1	9	0xffffffff
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000005
\$s1	17	0x00000004
\$s2	18	0x00000001
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10000000
\$sp	29	0xffffffff
\$fp	30	0x00000000
\$ra	31	0x00400010
pc		
hi		0x00000000

Result is stored in \$t0 register.

Assignment 3

Source code and explanation:

#Laboratory Week 7, Assignment 3

```

.text
    li $s0, 3    #Init test value
    li $s1, 6

push:
    addi $sp, $sp, -8 #adjust stack pointer
    sw $s0, 4($sp)    #push $s0 to stack
    sw $s1, 0($sp)    #push $s1 to stack

work:
    nop

pop:
    lw $t0, 0($sp)    #pop to $t0
    lw $t1, 4($sp)    #pop to $t1
    addi $sp, $sp, 8 #adjust stack pointer

```

Run results: Stack's values are popped to \$t0 and \$t1 register and follow the 'First in last out' rule

The screenshot shows the MARS MIPS simulator interface. On the left, the assembly code for 'asm3.asm' is displayed, titled '#Laboratory Week 7, Assignment 3'. The code includes instructions for initializing registers, pushing data onto the stack, and popping it back. A tooltip for the 'nop' instruction at line 13 states: 'nop Null operation : machine code is all zeroes'. On the right, the 'Registers' window shows the state of MIPS registers. The 'Stack Pointer' (\$sp) register at index 29 is highlighted with a green background and contains the value 0x7ffffc.

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$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	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10000000
\$sp	29	0x7ffffc
\$fp	30	0x00000000
\$ra	31	0x00400024
pc		0x00400024
hi		0x00000000

Assignment 4

Source code and explanation:

#Laboratory Week 7, Assignment 4

```
.data
msg: .asciiz "Factorial calculation result: "

.text

main:
    jal    warp

print:
    la $a0, msg
    add $a1, $t1, $zero
    li $v0, 56
    syscall

exit:
    li $v0, 10
    syscall

endmain:

warp:
    sw $fp, -4($sp)      #save frame pointer to stack
    addi $fp, $sp, 0     #save stack's top address to frame pointer
    addi $sp, $sp, -8    #adjust stack pointer to make space
    sw $ra, 0($sp)       #save return address to stack

    li $s0, 5           #Init test input N
    jal fact             #jump to procedure factorial
    nop
    lw $ra, 0($sp)       #retrieve return address
```

```

    addi $sp, $fp, 0 #return stack pointer
    lw $fp, -4($sp)      #return frame pointer
    jr $ra

warpent:

fact:
    sw $fp, -4($sp)      #save frame pointer to stack
    addi $fp, $sp, 0 #save stack's top address to frame pointer

top:
    addi $sp, $sp, -12    #adjust stack pointer to make space for $fp,
    $ra, $s0

stack:
    sw $ra, 4($sp)        #save return address to stack
    sw $s0, 0($sp)        #save N to stack

    slti $t0, $s0, 2 #if N < 2 false
    beqz $t0, recursive  #then continue recursive
    nop
    li $t1, 1            #else resultN! = 1
    j done
    nop

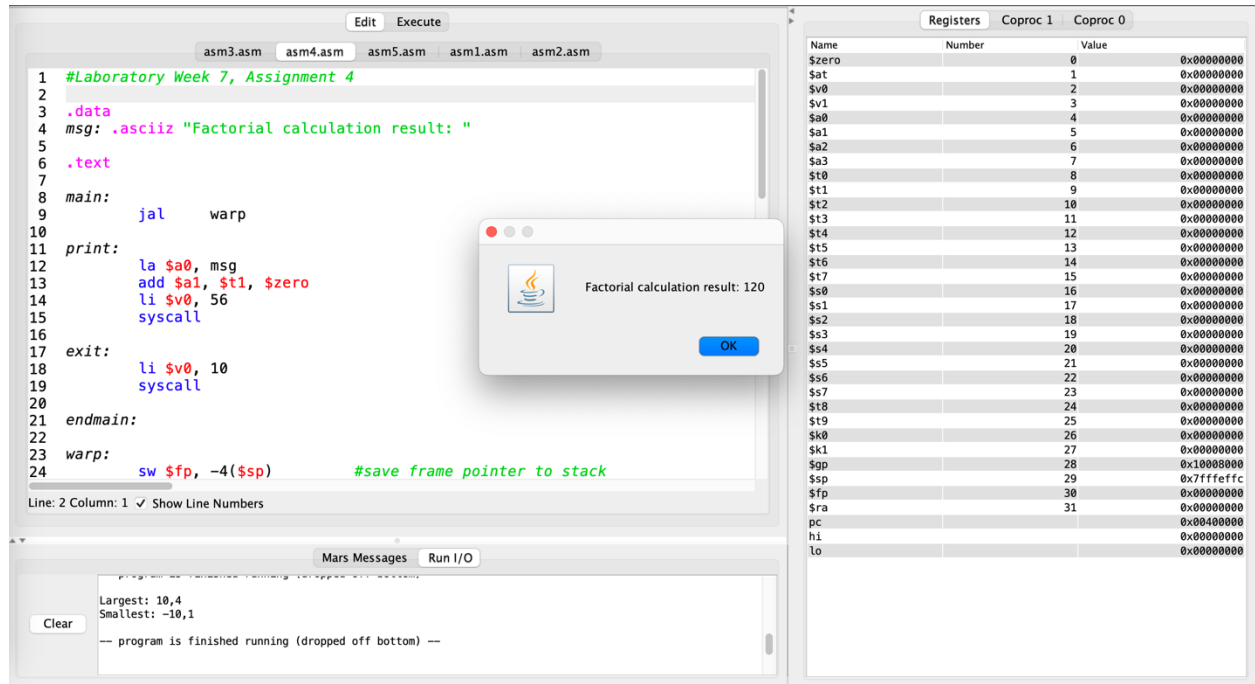
recursive:
    addi $s0, $s0, -1 #N Decrement
    jal fact           #recursive call
    nop
    lw $t2, 0($sp)      #Load current N
    mult $t2, $t1        #Perform multiplication
    mflo $t1

done:
    lw $ra, 4($sp)      #retrieve return address
    lw $s0, 0($sp)      #retrieve initial N
    addi $sp, $fp, 0 #retrieve stack pointer
    lw $fp, -4($sp)      #retrieve frame pointer
    jr $ra              #jump back

factend:

```

Run result: With test value N = 5



Assignment 5

Source code and explanation:

#Laboratory Week 7, Assignment 5

```
.data
max_message: .asciiz "Largest: "
min_message: .asciiz "Smallest: "
comma: .asciiz ","
newline: .asciiz "\n"
```

```
.text
main:
    li $s0, -2
    li $s1, -10
    li $s2, 5
    li $s3, 6
    li $s4, 10
    li $s5, 2
    li $s6, -3
    li $s7, 9
    addi $sp, $sp, -32    #Allocate space in stack
```

```
push:
    sw $s7, 0($sp)        #Push to stack
    addi $sp, $sp, 4
    sw $s6, 0($sp)
    addi $sp, $sp, 4
    sw $s5, 0($sp)
    addi $sp, $sp, 4
```

```

        sw $s4, 0($sp)
        addi $sp, $sp, 4
        sw $s3, 0($sp)
        addi $sp, $sp, 4
        sw $s2, 0($sp)
        addi $sp, $sp, 4
        sw $s1, 0($sp)
        addi $sp, $sp, 4
        sw $s0, 0($sp)
        addi $sp, $sp, 4

find_min_max:
        li $t3, 0    # $t3 = 0 (i)
        li $t4, 8    # Loop condition
        lw $k0, -4($sp) # $k0 = $s0 => initial value
        lw $k1, -4($sp) # $k1 = $s0
                        # $k0 = max, $k1 = min, $t1 = index of max, $t0 = index of
min, $a0 = value, $a1 = value position

load:
        addi $t3, $t3, 1 # $t3 = $t3 + 1 = i + 1
        add $t5, $t3, $t3 # $t5 = 2*$t3 = 2*i
        add $t5, $t5, $t5 # $t5 = 4*$t3 = 4*i
        sub $t5, $zero, $t5 # $t1 = -4*i
        add $t2, $t5, $sp # $t2 = $sp - 4*$t1 = &A[i]
        lw $a0, 0($t2) # $a0 = *$t2 = A[i]
        addi $a1, $t3, -1 # $a1 = $t0 - 1 = position of A[i]
        beq $a1, $t4, print # if $a1 == $t4, jump to print
        nop

check:
        slt $a2, $k0, $a0 # if $k0 < $a0 => $a1 = 1 ($a0 <= $k0)
        beqz $a2, min # if $a1 == 0 ($a0 <= $k0 => no change), jump
to min
        nop
        addi $k0, $a0, 0 # $k0 = $a0
        addi $t0, $a1, 0 # $t0 = $a1 (new max position)

min:
        slt $a2, $a0, $k1 # if $k1 <= $a0 => jump, if $a0 < $k1 => $a1 = 1
        beqz $a2, load # if $a1 == 0 ($k1 <= $a0 => no change), jump to
load (new loop)
        nop
        addi $k1, $a0, 0 # $k1 = $a0
        addi $t1, $a1, 0 # $t1 = $a1 (new min position)
        j load

print:
        li $v0, 4 # Print max result
        la $a0, max_message
        syscall
        li $v0, 1
        addi $a0, $k0, 0
        syscall

```

```

li $v0, 4
la $a0, comma
syscall
li $v0, 1
addi $a0, $t0, 0
syscall
li $v0, 4
la $a0, newline
syscall

li $v0, 4          #Print min result
la $a0, min_message
syscall
li $v0, 1
addi $a0, $k1, 0
syscall
li $v0, 4
la $a0, comma
syscall
li $v0, 1
addi $a0, $t1, 0
syscall
li $v0, 4
la $a0, newline
syscall

```

Run Results:

The screenshot displays the MARS MIPS simulator interface. The main window shows assembly code for a program that finds the largest and smallest numbers in an array. The code includes data declarations for messages, stack allocation, and a loop to compare elements. The output window shows the results: "Largest: 10,4" and "Smallest: -10,1". The registers window on the right shows the state of the MIPS registers, with \$t0 and \$t1 containing the values 10 and 4 respectively.

Assembly Code:

```

1 #Laboratory Week 7, Assignment 5
2
3 .data
4 max_message: .asciiz "Largest: "
5 min_message: .asciiz "Smallest: "
6 comma: .asciiz ","
7 newline: .asciiz "\n"
8
9 .text
10 main:
11     li $s0, -2
12     li $s1, -10
13     li $s2, 5
14     li $s3, 6
15     li $s4, 10
16     li $s5, 2
17     li $s6, -3
18     li $s7, 9
19     addi $sp, $sp, -32    #Allocate space in stack
20
21 push:
22     sw $s7, 0($sp)      #Push to stack
23     addi $sp, $sp, 4
24     sw $s6, 0($sp)

```

Registers:

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x10010000
\$v0	2	0x00000004
\$v1	3	0x00000000
\$a0	4	0x10010017
\$a1	5	0x00000007
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000004
\$t1	9	0x00000001
\$t2	10	0x7fffffffdc
\$t3	11	0x00000000
\$t4	12	0x00000007
\$t5	13	0xffffffffe0
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0xfffffffffe
\$s1	17	0xfffffffff6
\$s2	18	0x00000005
\$s3	19	0x00000006
\$s4	20	0x0000000a
\$s5	21	0x00000002
\$s6	22	0xffffffffd
\$s7	23	0x00000009
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x0000000a
\$k1	27	0xfffffffff6
\$gp	28	0x10000000
\$sp	29	0x7ffffeffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400154
hi		0x00000000
lo		0x00000000

Output:

```

Largest: 10,4
Smallest: -10,1

```

program is finished running (dropped off bottom) --