

## LABORATORY 7

Họ tên: Vũ Quốc Bảo  
MSSV: 20225694

### Assignment 1:

```
.text
main:
li $a0,-45 # 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:

The screenshot shows the MARS MIPS simulator interface. The 'Text Segment' window displays the assembly code with labels and addresses. The 'Registers' window shows the state of registers, with \$v0 containing 10. The 'Data Segment' window shows memory addresses and their values.

Label	Address
main	0x00400000
endmain	0x00400010
abs	0x00400015
done	0x00400020

Register	Value
\$zero	0
\$at	1
\$v0	10
\$v1	0
\$a0	-45
\$a1	0
\$a2	0
\$a3	0
\$a4	0
\$a5	0
\$a6	0
\$a7	0
\$s0	0
\$s1	0
\$s2	0
\$s3	0
\$s4	0
\$s5	0
\$s6	0
\$s7	0
\$t0	0
\$t1	0
\$t2	0
\$t3	0
\$t4	0
\$t5	0
\$t6	0
\$t7	0
\$f0	0
\$f1	0
\$f2	0
\$f3	0
\$f4	0
\$f5	0
\$f6	0
\$f7	0
\$f8	0
\$f9	0
\$f10	0
\$f11	0
\$f12	0
\$f13	0
\$f14	0
\$f15	0
\$f16	0
\$f17	0
\$f18	0
\$f19	0
\$f20	0
\$f21	0
\$f22	0
\$f23	0
\$f24	0
\$f25	0
\$f26	0
\$f27	0
\$f28	0
\$f29	0
\$f30	0
\$f31	0

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0	0	0	0	0	0	0	0
0x10010004	0	0	0	0	0	0	0	0
0x10010008	0	0	0	0	0	0	0	0
0x1001000c	0	0	0	0	0	0	0	0
0x10010010	0	0	0	0	0	0	0	0
0x10010014	0	0	0	0	0	0	0	0
0x10010018	0	0	0	0	0	0	0	0
0x1001001c	0	0	0	0	0	0	0	0
0x10010020	0	0	0	0	0	0	0	0
0x10010024	0	0	0	0	0	0	0	0
0x10010028	0	0	0	0	0	0	0	0
0x1001002c	0	0	0	0	0	0	0	0
0x10010030	0	0	0	0	0	0	0	0
0x10010034	0	0	0	0	0	0	0	0
0x10010038	0	0	0	0	0	0	0	0
0x1001003c	0	0	0	0	0	0	0	0
0x10010040	0	0	0	0	0	0	0	0
0x10010044	0	0	0	0	0	0	0	0
0x10010048	0	0	0	0	0	0	0	0
0x1001004c	0	0	0	0	0	0	0	0
0x10010050	0	0	0	0	0	0	0	0
0x10010054	0	0	0	0	0	0	0	0
0x10010058	0	0	0	0	0	0	0	0
0x1001005c	0	0	0	0	0	0	0	0
0x10010060	0	0	0	0	0	0	0	0
0x10010064	0	0	0	0	0	0	0	0
0x10010068	0	0	0	0	0	0	0	0
0x1001006c	0	0	0	0	0	0	0	0
0x10010070	0	0	0	0	0	0	0	0
0x10010074	0	0	0	0	0	0	0	0
0x10010078	0	0	0	0	0	0	0	0
0x1001007c	0	0	0	0	0	0	0	0
0x10010080	0	0	0	0	0	0	0	0
0x10010084	0	0	0	0	0	0	0	0
0x10010088	0	0	0	0	0	0	0	0
0x1001008c	0	0	0	0	0	0	0	0
0x10010090	0	0	0	0	0	0	0	0
0x10010094	0	0	0	0	0	0	0	0
0x10010098	0	0	0	0	0	0	0	0
0x1001009c	0	0	0	0	0	0	0	0
0x100100a0	0	0	0	0	0	0	0	0
0x100100a4	0	0	0	0	0	0	0	0
0x100100a8	0	0	0	0	0	0	0	0
0x100100ac	0	0	0	0	0	0	0	0
0x100100b0	0	0	0	0	0	0	0	0
0x100100b4	0	0	0	0	0	0	0	0
0x100100b8	0	0	0	0	0	0	0	0
0x100100bc	0	0	0	0	0	0	0	0
0x100100c0	0	0	0	0	0	0	0	0
0x100100c4	0	0	0	0	0	0	0	0
0x100100c8	0	0	0	0	0	0	0	0
0x100100cc	0	0	0	0	0	0	0	0
0x100100d0	0	0	0	0	0	0	0	0
0x100100d4	0	0	0	0	0	0	0	0
0x100100d8	0	0	0	0	0	0	0	0
0x100100dc	0	0	0	0	0	0	0	0
0x100100e0	0	0	0	0	0	0	0	0
0x100100e4	0	0	0	0	0	0	0	0
0x100100e8	0	0	0	0	0	0	0	0
0x100100ec	0	0	0	0	0	0	0	0
0x100100f0	0	0	0	0	0	0	0	0
0x100100f4	0	0	0	0	0	0	0	0
0x100100f8	0	0	0	0	0	0	0	0
0x100100fc	0	0	0	0	0	0	0	0
0x10010100	0	0	0	0	0	0	0	0
0x10010104	0	0	0	0	0	0	0	0
0x10010108	0	0	0	0	0	0	0	0
0x1001010c	0	0	0	0	0	0	0	0
0x10010110	0	0	0	0	0	0	0	0
0x10010114	0	0	0	0	0	0	0	0
0x10010118	0	0	0	0	0	0	0	0
0x1001011c	0	0	0	0	0	0	0	0
0x10010120	0	0	0	0	0	0	0	0
0x10010124	0	0	0	0	0	0	0	0
0x10010128	0	0	0	0	0	0	0	0
0x1001012c	0	0	0	0	0	0	0	0
0x10010130	0	0	0	0	0	0	0	0
0x10010134	0	0	0	0	0	0	0	0
0x10010138	0	0	0	0	0	0	0	0
0x1001013c	0	0	0	0	0	0	0	0
0x10010140	0	0	0	0	0	0	0	0
0x10010144	0	0	0	0	0	0	0	0
0x10010148	0	0	0	0	0	0	0	0
0x1001014c	0	0	0	0	0	0	0	0
0x10010150	0	0	0	0	0	0	0	0
0x10010154	0	0	0	0	0	0	0	0
0x10010158	0	0	0	0	0	0	0	0
0x1001015c	0	0	0	0	0	0	0	0
0x10010160	0	0	0	0	0	0	0	0
0x10010164	0	0	0	0	0	0	0	0
0x10010168	0	0	0	0	0	0	0	0
0x1001016c	0	0	0	0	0	0	0	0
0x10010170	0	0	0	0	0	0	0	0
0x10010174	0	0	0	0	0	0	0	0
0x10010178	0	0	0	0	0	0	0	0
0x1001017c	0	0	0	0	0	0	0	0
0x10010180	0	0	0	0	0	0	0	0
0x10010184	0	0	0	0	0	0	0	0
0x10010188	0	0	0	0	0	0	0	0
0x1001018c	0	0	0	0	0	0	0	0
0x10010190	0	0	0	0	0	0	0	0
0x10010194	0	0	0	0	0	0	0	0
0x10010198	0	0	0	0	0	0	0	0
0x1001019c	0	0	0	0	0	0	0	0
0x100101a0	0	0	0	0	0	0	0	0
0x100101a4	0	0	0	0	0	0	0	0
0x100101a8	0	0	0	0	0	0	0	0
0x100101ac	0	0	0	0	0	0	0	0
0x100101b0	0	0	0	0	0	0	0	0
0x100101b4	0	0	0	0	0	0	0	0
0x100101b8	0	0	0	0	0	0	0	0
0x100101bc	0	0	0	0	0	0	0	0
0x100101c0	0	0	0	0	0	0	0	0
0x100101c4	0	0	0	0	0	0	0	0
0x100101c8	0	0	0	0	0	0	0	0
0x100101cc	0	0	0	0	0	0	0	0
0x100101d0	0	0	0	0	0	0	0	0
0x100101d4	0	0	0	0	0	0	0	0
0x100101d8	0	0	0	0	0	0	0	0
0x100101dc	0	0	0	0	0	0	0	0
0x100101e0	0	0	0	0	0	0	0	0
0x100101e4	0	0	0	0	0	0	0	0
0x100101e8	0	0	0	0	0	0	0	0
0x100101ec	0	0	0	0	0	0	0	0
0x100101f0	0	0	0	0	0	0	0	0
0x100101f4	0	0	0	0	0	0	0	0
0x100101f8	0	0	0	0	0	0	0	0
0x100101fc	0	0	0	0	0	0	0	0
0x10010200	0	0	0	0	0	0	0	0
0x10010204	0	0	0	0	0	0	0	0
0x10010208	0	0	0	0	0	0	0	0
0x1001020c	0	0	0	0	0	0	0	0
0x10010210	0	0	0	0	0	0	0	0
0x10010214	0	0	0	0	0	0	0	0
0x10010218	0	0	0	0	0	0	0	0
0x1001021c	0	0	0	0	0	0	0	0
0x10010220	0	0	0	0	0	0	0	0
0x10010224	0	0	0	0	0	0	0	0
0x10010228	0	0	0	0	0	0	0	0
0x1001022c	0	0	0	0	0	0	0	0
0x10010230	0	0	0	0	0	0	0	0
0x10010234	0	0	0	0	0	0	0	0
0x10010238	0	0	0	0	0	0	0	0
0x1001023c	0	0	0	0	0	0	0	0
0x10010240	0	0	0	0	0	0	0	0
0x10010244	0	0	0	0	0	0	0	0
0x10010248	0	0	0	0	0	0	0	0
0x1001024c	0	0	0	0	0	0	0	0
0x10010250	0	0	0	0	0	0	0	0
0x10010254	0	0	0	0	0	0	0	0
0x10010258	0	0	0	0	0	0	0	0
0x1001025c	0	0	0	0	0	0	0	0
0x10010260	0	0	0	0	0	0	0	0
0x10010264	0	0	0	0	0	0	0	0
0x10010268	0	0	0	0	0	0	0	0
0x1001026c	0	0	0	0	0	0	0	0
0x10010270	0	0	0	0	0	0	0	0
0x10010274	0	0	0	0	0			

- Giải thích:
- + Trước khi nhảy đến chương trình con **abs**:
  - Chương trình khởi tạo giá trị cho thanh ghi \$a0.
  - \$ra = 0.
  - pc ghi địa chỉ của câu lệnh đang trở tới. (pc = 4194304)
- + Sau khi nhảy đến chương trình con **abs**:
  - lệnh **jal** lưu địa chỉ trở về \$ra. (\$ra = 4194312)
  - pc ghi địa chỉ của câu lệnh đang trở tới. (pc = 4194328)
  - giá trị tuyệt đối của \$a0 sẽ được lưu vào thanh ghi \$v0.
  - điều kiện **bltz \$a1, done** kiểm tra xem giá trị đầu vào có âm hay không. Trong trường hợp này, vì giá trị là âm, nó nhảy tới **done** và kết thúc. Còn nếu không âm, giá trị đó sẽ được sao chép vào \$v0.
  - sau khi hoàn thành chương trình con **abs**, chương trình quay trở lại lệnh sau **jal** là lệnh **nop** trong **main**. Khi này thì giá trị tuyệt đối của \$a0 đã được lưu vào trong \$v0.
- + Nhảy ra khỏi **abs**:
  - \$ra = 41943112.
  - pc = 4194328.
  - lưu giá trị từ \$v0 chuyển sang \$s0. Gán \$v0 = 10 rồi sử dụng lệnh **syscall** để kết thúc chương trình.

### Assignment 2:

```
.text
main: li $a0,4 #load test input
li $a1,16
li $a2,-9
jal max #call max procedure
nop
add $s0, $zero, $v0
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 (a1)-(v0)
bltz $t0,okay #if (a1)-(v0)<0 then no change
nop
```

- **Kết quả:**

- Giải thích:

$$+ \$ra = 4194320.$$

+ Trong hàm *max*:

- + Ở đây, giá trị lớn nhất trong 3 thanh ghi đã được lưu vào \$v0.

+ Sau đó, gán  $v0 = 10$  rồi sử dụng lệnh *syscall* để kết thúc chương trình.

.data

```
y: .word 15
```

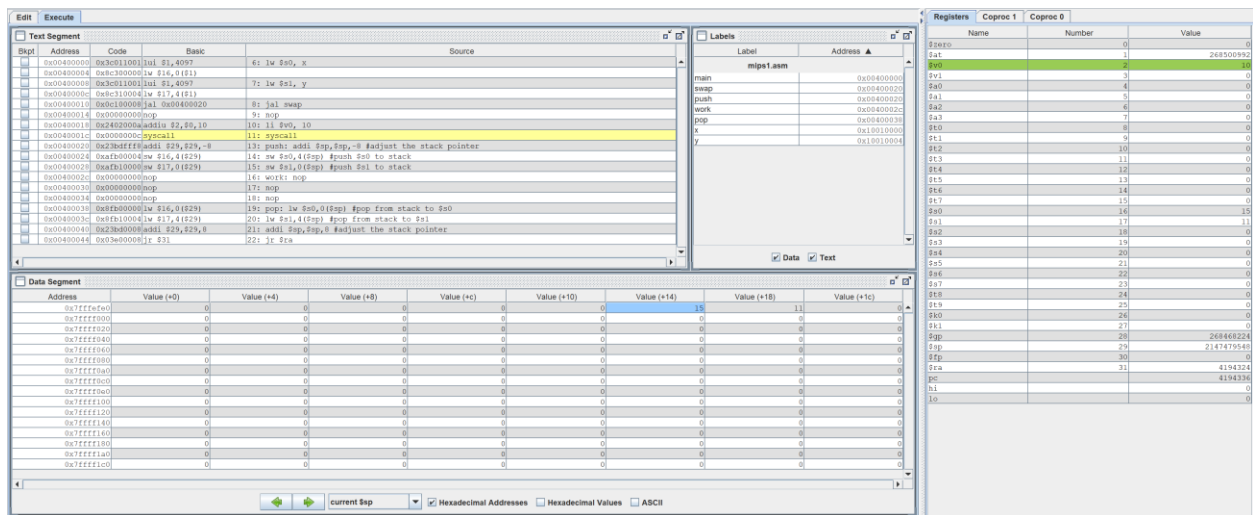
main:

```

lw $s0, x
lw $s1, y
jal swap
nop
li $v0, 10
syscall
swap:
push: addi $sp,$sp,-8 #adjust the stack pointer
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
jr $ra

```

- Kết quả:



- Giải thích:

- + Giá trị của biến x được nạp vào thanh ghi \$s0 bằng lệnh **lw \$s0, x**.
- + Giá trị của biến y được nạp vào thanh ghi \$s1 bằng lệnh **lw \$s1, y**.
- + Chương trình gọi hàm **swap** bằng lệnh **jal swap**. Điều này làm chương trình nhảy đến địa chỉ của hàm **swap**.
- + Trong hàm **swap**:
  - Đầu tiên, một không gian trên ngăn xếp được cấp phát bằng cách giảm con trỏ ngăn xếp \$sp đi 8 byte bằng lệnh **addi \$sp, \$sp, -8**.
  - Các giá trị của \$s0 và \$s1 được đẩy vào ngăn xếp. \$s0 được đẩy trước và lưu tại địa chỉ 4(\$sp), sau đó \$s1 được đẩy và lưu tại địa chỉ 0(\$sp).

- Sau khi giá trị đã được đẩy vào ngăn xếp, các lệnh ***nop*** được sử dụng làm lùi bước để có thời gian cho việc thực hiện đẩy và lấy giá trị trên ngăn xếp.
- Các giá trị đó sẽ được lấy từ ngăn xếp và sao chép vào các thanh ghi \$s0 và \$s1.
- Cuối cùng, không gian trên ngăn xếp được giải phóng bằng cách tăng con trỏ ngăn xếp \$sp lên 8 byte bằng lệnh ***addi \$sp, \$sp, 8***, và hàm kết thúc bằng lệnh ***jr \$ra*** để quay lại lệnh sau lệnh gọi hàm.

+ Chương trình tiếp tục thực hiện lệnh ***nop*** sau lệnh ***jal***.

+ Sau đó, gán \$v0 = 10 rồi sử dụng lệnh ***syscall*** để kết thúc chương trình.

→ Vì vậy, trong quá trình này, giá trị của biến x và y (hay giá trị của 2 thanh ghi \$s0 và \$s1) đã được hoán đổi bằng cách sử dụng một hàm ***swap***.

#### **Assignment 4:**

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 \$a0,3 #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:

#-----

#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) >=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:

The screenshot shows a MIPS simulator interface with three main panels:

- Test Segment:** Displays assembly code with addresses, codes, and comments. The code implements a factorial function using recursion.
- Registers:** Shows the state of MIPS registers. The \$v0 register contains the value 6, which is the result of the factorial calculation.
- Data Segment:** Shows memory addresses and their corresponding values. A dialog box is overlaid on this panel, displaying the message "Kết quả tính giai thừa là: 6" (The result of the factorial calculation is: 6).

At the begin of WRAP, \$sp = 0x7fffeffc

\$ra (4)	7fffeff4 ← new \$sp (3) addi \$sp,\$sp,-8
\$fp	7fffeff8 (1) sw \$fp,-4(\$sp)
	7fffeffc ← new \$fp (2) addi \$fp,\$sp,0

At the end of WRAP, \$sp = 0x7fffeff4

\$a0	1
\$ra	0x00400080
\$fp	0x7fffe8
\$a0	2
\$ra	0x00400080
\$fp	0x7fffeff4
\$a0	3
\$ra	0x00400038
\$fp	0x7fffeffc
\$ra (4)	7fffeff4 → restore \$ra (5)
\$fp	7fffeff8 → restore \$fp (7)
	7fffeffc → restore \$sq (6)

### Assignment 5:

.data

Mess1: .asciiz "Lon nhât: "

Mess2: .asciiz "\nNho nhât: "

Comma: .asciiz ", "

.text

main: # khởi tạo giá trị cho thanh ghi \$s0-\$s7

li \$s0, 15

li \$s1, 5

li \$s2, 9

li \$s3, 30

li \$s4, 169

li \$s5, 99

li \$s6, -17

li \$s7, -2

jal init

```

nop
li $v0, 4
la $a0, Mess1 # print Mess1
syscall
li $v0, 1
add $a0, $t0, $zero # print max value
syscall
li $v0, 4
la $a0, Comma # print ","
syscall
li $v0, 1
add $a0, $t5, $zero # print max value's position
syscall
li $v0, 4
la $a0, Mess2 # print Mess2
syscall
li $v0, 1
add $a0, $t1, $zero # print min value
syscall
li $v0, 4
la $a0, Comma # print ","
syscall
li $v0, 1
add $a0, $t6, $zero # print min value's position
syscall
li $v0, 10 # exit
syscall
endmain:
swapMax:
add $t0, $t3, $zero # set Max = $t3
add $t5, $t2, $zero # set i of max = $t2
jr $ra
swapMin:
add $t1, $t3, $zero # set Min = $t3
add $t6, $t2, $zero # set i of min = $t2
jr $ra
init:
add $fp, $sp, $zero
addi $sp, $sp, -36
sw $s0, 0($sp)
sw $s1, 4($sp)
sw $s2, 8($sp)
sw $s3, 12($sp)
sw $s4, 16($sp)
sw $s5, 20($sp)
sw $s6, 24($sp)

```



```

sw $s7, 28($sp)
sw $ra, 32($sp)
add $t0, $s0, $zero # set Max = $s0
# $t0 lưu giá trị max
add $t1, $s0, $zero # set Min = $s0
# $t1 lưu giá trị min
li $t5, 0
li $t6, 0
li $t2, 0 # i = 0
max_min:
addi $sp, $sp, 4
lw $t3, -4($sp)
sub $t4, $sp, $fp # check if meet $ra
beq $t4, $zero, done
addi $t2, $t2, 1 # i++
sub $t4, $t0, $t3
bltzal $t4, swapMax
sub $t4, $t3, $t1
bltzal $t4, swapMin
j max_min # repeat
done:
lw $ra, -4($sp)
jr $ra

```

The screenshot displays a MIPS2 simulator interface with four main panels:

- Text Segment:** Shows assembly code with columns for Bkpt, Address, Code, Basic, and Source. The code includes instructions like `li $v0, 1`, `addi $a0, $t6, $zero`, `syscall`, `addi $v0, 10`, `syscall`, `add $t0, $t3, $zero`, `add $t1, $t3, $zero`, `jr $ra`, `addi $sp, $sp, -36`, `sw $s0, 0($sp)`, `lw $t3, 4($sp)`, `sw $s1, 8($sp)`, `lw $t3, 12($sp)`, `sw $s2, 16($sp)`, and `lw $t3, 20($sp)`.
- Labels:** Lists labels such as `main`, `endmain`, `swapMax`, `swapMin`, `init`, `done`, `Msg1`, `Msg2`, and `Comma` with their corresponding addresses.
- Registers:** A table showing register names, numbers, and values. Key registers include `$zero` (0), `$at` (268500992), `$v0` (1), `$a0` (0), `$a1` (0), `$a2` (0), `$a3` (0), `$t0` (0), `$t1` (0), `$t2` (0), `$t3` (0), `$t4` (0), `$t5` (0), `$t6` (0), `$t7` (0), `$s0` (0), `$s1` (0), `$s2` (0), `$s3` (0), `$s4` (0), `$s5` (0), `$s6` (0), `$s7` (0), `$s8` (0), `$s9` (0), `$k0` (0), `$k1` (0), `$k2` (0), `$k3` (0), `$k4` (0), `$k5` (0), `$k6` (0), `$k7` (0), `$ra` (0), `$pc` (0), `$hi` (0), and `$lo` (0).
- Data Segment:** A table showing memory addresses and their corresponding values in hexadecimal, decimal, and ASCII formats. The values are mostly 0, with some non-zero values like 15, 26, 27, 28, 29, 30, 31, 419410, and 419411.

Mars Messages	Run I/O
	<p>program is finished running --</p> <p>Lon nhat: 169,5 Nho nhat: -17,7 -- program is finished running --</p> <p>Lon nhat: 169,5 Nho nhat: -17,7 -- program is finished running --</p> <p>7 -- program is finished running --</p>

Clear