HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

School of information and communication technology

----- ☙ 🕮 ❧ -----

FINAL PROJECT REPORT

IT3280E – ASSEMBLY LANGUAGE AND COMPUTER ARCHITECTURE LAB

Lecturer: **MsC. Lê Bá Vui**

Student Group 09:

	Name	ID
1	Trần Tùng Dương	20215190
2	Nguyễn Xuân Kiên	20215217

Ha Noi, January 2024

TABLE OF CONTENTS

TABLE OF CONTENTS ....

PART I. PROJECT 1 ....

I.1. Method..

I.2. Algorithm ....

I.3. Source Code..

I.4. Result..

PART II. PROJECT 6..

II.1. Method ....

II.2. Algorithm..

II.3. Source Code ....

II.4. Result..

I. PROBLEM 1

I.1. Method

- This program will read the control code from the Digital Lab Sim and the command from MMIO to execute the MarsBot.

- The codes and commands are as follow :

Control code	Meaning
1b4	Start moving
c68	Stop moving
444	Turn left 90 degrees with the current direction
666	Turn right 90 degrees with the current direction
dad	Start to leave a trace
cbc	Stop to leave the trace
999	Follow the reverse route without leaving a trace and accept the control code until the end of the route.

Command	Meaning
Enter	Complete receiving the control code, Marsbot takes the action.
Delete	Clear the receiving control code.
Space	Repeat the last taken control code.

I.2. Algorithm

a) Main function

- Initialize the current direction is 90 degree

-Execute funcion « read input »

b) Read Input

- Read the command from MMIO.

+ Case 1 : If the command is Enter, the program will then read the current control code from the key matrix. If the control code is invalid, print error message, else it will be printed on console and the MarsBot will move according to the control code.

+ Case 2 : If the command is Delete, the program will automatically reset the input

+ Case 3 : If the command is Space, execute the previous control code

c) MarBots moving function

- There will be total of seven directions, each direction match with control code of it.

d) Print console function

- This will either print the message notify invalid control code or print the control code to the console.

e) String related function

- This program will recreate the strcmp, strcpy funtion in order to satisfy above functions.

I.3. Source Code

```
# eqv for Digital Lab Sim
.eqv key0 0x11
.eqv key1 0x21
.eqv key2 0x41
.eqv key3 0x81
.eqv key4 0x12
.eqv key5 0x22
.eqv key6 0x42
.eqv key7 0x82
.eqv key8 0x14
.eqv key9 0x24
.eqv keya 0x44
.eqv keyb 0x84
```

```
.eqv keyc 0x18

.eqv keyd 0x28

.eqv keye 0x48

.eqv keyf 0x88


# eqv for Keyboard

.eqv IN_ADRESS_HEXA_KEYBOARD 0xFFFF0012

.eqv OUT_ADRESS_HEXA_KEYBOARD 0xFFFF0014

.eqv KEY_CODE 0xFFFF0004

.eqv KEY_READY 0xFFFF0000



# eqv for Mars bot

.eqv HEADING 0xffff8010

.eqv MOVING 0xffff8050

.eqv LEAVETRACK 0xffff8020

.eqv WHEREX 0xffff8030

.eqv WHEREY 0xffff8040


.data


        string1: .asciiz "1b4"

        string2: .asciiz "c68"

        string3: .asciiz "444"

        string4: .asciiz "666"

        string5: .asciiz "dad"
```

```
string6: .asciiz "cbc"
string7: .asciiz "999"

error:  .asciiz "Invalid command : "


# HISTORY
# save history before changing direction

x_his: .word 0 : 16
y_his: .word 0 : 16

# For rotation
a_his: .word 0 : 16
l_his: .word 4

a_now: .word 0

is_going: .word 0
is_tracking: .word 0

# Array and variables
control_code: .space 8
code_length: .word 0

prev_code: .space 8
```

```
.text

main:
	li $k0, KEY_CODE
	li $k1, KEY_READY

	li $t1, IN_ADRESS_HEXA_KEYBOARD # enable the interrupt of Digital Lab Sim
	li $t3, 0x80   # bit 7 = 1 to enable
	sb $t3, 0($t1)

# run at start of program
init:
	# increase length history by 4
	# (as saving current state: x = 0; y = 0; a = 90)

	lw $t7, l_his # l_history += 4
	addi $t7, $zero, 4
	sw $t7, l_his

	li $t7, 90
	sw $t7, a_now # a_current = 90 -> head to the right
	jal ROTATE
	nop
```

```
        sw $t7, a_his # a_history[0] = 90

        j waitForKey

# Function to print to console
printError:
        li $v0, 4
        la $a0, error
        syscall

printCode:
        li $v0, 4
        la $a0, control_code
        syscall
        j resetInput

repeatCode:


        jal strcpyPrevToCur
        j checkCode

resetInput:
        jal strClear
        nop

#input
```

```
waitForKey:
        lw $t5, 0($k1)   # $t5 = [$k1] = KEY_READY
        beq $t5, $zero, waitForKey  # if $t5 == 0 -> Polling
        nop
        beq $t5, $zero, waitForKey

readKey:
        lw $t6, 0($k0)   # $t6 = [$k0] = KEY_CODE
        # if $t6 == 'DEL' -> reset input
        beq $t6, 0x8, resetInput

        # if $t6 == 'SPACE' -> reset copy from previous input and
        # go to checkCode label
        beq $t6, 0x20, repeatCode

        # if $t6 != 'ENTER' -> Polling
        bne $t6, 0x0a, waitForKey
        nop
        bne $t6, 0x0a, waitForKey

checkCode:
        lw $s2, code_length   # code_length != 3 -> invalid code
        bne $s2, 3, printError

        la $s3, string1
        jal strcmp
```

```
beq $t0, 1, go

la $s3, string2
jal strcmp
beq $t0, 1, stop

la $s3, string3
jal strcmp
beq $t0, 1, turnLeft

la $s3, string4
jal strcmp
beq $t0, 1, turnRight

la $s3, string5
jal strcmp
beq $t0, 1, track

la $s3, string6
jal strcmp
beq $t0, 1, untrack

la $s3, string7
jal strcmp
beq $t0, 1, goBackward
nop
```

```
	j printError



# Perform function MarsBot
go:
	jal strCpy2
	jal GO
	j printCode

stop:
	jal strCpy2
	jal STOP
	j printCode

track:
	jal strCpy2
	jal TRACK
	j printCode

untrack:
	jal strCpy2
	jal UNTRACK
	j printCode
```

```
turnRight:
        jal strCpy2
        lw $t7, is_going
        lw $s0, is_tracking

        jal STOP
        nop
        jal UNTRACK
        nop

        la $s5, a_now
        lw $s6, 0($s5)
        addi $s6, $s6, 90
        sw $s6, 0($s5)

        jal saveHistory
        jal ROTATE

        beqz $s0, noTrack1
        nop
        jal TRACK
        noTrack1: nop

        beqz $t7, noGo1
        nop
        jal GO
```

```
noGo1:
        nop
        j printCode



turnLeft:
        jal strCpy2
        lw $t7, is_going
        lw $s0, is_tracking

        jal STOP
        nop
        jal UNTRACK
        nop

        la $s5, a_now
        lw $s6, 0($s5)  # $s6 is heading at now
        addi $s6, $s6, -90 # decrease alpha by 90*
        sw $s6, 0($s5)  # update a_current

        jal saveHistory
        jal ROTATE

        beqz $s0, noTrack2
        nop
        jal TRACK
```

```
	noTrack2: nop


	beqz $t7, noGo2
	nop
	jal GO
noGo2:
	nop
	j printCode



goBackward:
	jal strCpy2
	li $t7, IN_ADRESS_HEXA_KEYBOARD # Disable interrupts when going backward
	sb $zero, 0($t7)

	lw $s5, l_his  # $s5 = code_length
	jal UNTRACK
	jal GO

goBackward_turn:
	addi $s5, $s5, -4   # code_length--
	lw $s6, a_his($s5)  # $s6 = a_history[code_length]
	addi $s6, $s6, 180  # $s6 = the reverse direction of alpha
	sw $s6, a_now
	jal ROTATE
```

```
        nop

goBackward_toTurningPoint:
        lw $t9, x_his($s5)  # $t9 = x_history[i]
        lw $t7, y_his($s5)  # $t9 = y_history[i]

get_x:
        li $t8, WHEREX   # $t8 = x_current
        lw $t8, 0($t8)

        bne $t8, $t9, get_x  # x_current == x_history[i]
        nop
        bne $t8, $t9, get_x
get_Y:
        li $t8, WHEREY   # $t8 = y_current
        lw $t8, 0($t8)
        bne $t8, $t7, get_Y  # y_current == y_history[i]
        nop
        bne $t8, $t7, get_Y  # y_current == y_history[i]
        beq $s5, 0, goBackward_end  # l_history == 0
        nop    # -> end
        j goBackward_turn   # else -> turn

goBackward_end:
        jal STOP
        sw $zero, a_now  # update heading
```

```
	jal ROTATE
	addi $s5, $zero, 4
	sw $s5, l_his  # reset l_history = 0
	j printCode


#-------------------------------------------------------
# saveHistory()
#-------------------------------------------------------


saveHistory:
	addi $sp, $sp, 4   # backup
	sw $t1, 0($sp)
	addi $sp, $sp, 4
	sw $t2, 0($sp)
	addi $sp, $sp, 4
	sw $t3, 0($sp)
	addi $sp, $sp, 4
	sw $t4, 0($sp)
	addi $sp, $sp, 4
	sw $s1, 0($sp)
	addi $sp, $sp, 4
	sw $s2, 0($sp)
	addi $sp, $sp, 4
	sw $s3, 0($sp)
	addi $sp, $sp, 4
	sw $s4, 0($sp)
```

```
lw $s1, WHEREX   # s1 = x
lw $s2, WHEREY   # s2 = y
lw $s4, a_now  # s4 = a_current

lw $t3, l_his  # $t3 = l_history
sw $s1, x_his($t3)  # store: x, y, alpha
sw $s2, y_his($t3)
sw $s4, a_his($t3)

addi $t3, $t3, 4   # update lengthPath
sw $t3, l_his

lw $s4, 0($sp)   # restore backup
addi $sp, $sp, -4
lw $s3, 0($sp)
addi $sp, $sp, -4
lw $s2, 0($sp)
addi $sp, $sp, -4
lw $s1, 0($sp)
addi $sp, $sp, -4
lw $t4, 0($sp)
addi $sp, $sp, -4
lw $t3, 0($sp)
addi $sp, $sp, -4
lw $t2, 0($sp)
```

```
	addi $sp, $sp, -4
	lw $t1, 0($sp)
	addi $sp, $sp, -4

saveHistory_end:
	jr $ra

#======================================================================
======================
# Procedure for Mars bot
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~
# GO()
#------------------------------------------------------------
GO:
	addi $sp, $sp, 4   # backup
	sw $at, 0($sp)
	addi $sp, $sp, 4
	sw $k0, 0($sp)

	li $at, MOVING   # change MOVING port
	addi $k0, $zero, 1  # to logic 1,
	sb $k0, 0($at)   # to start running

	li $t7, 1   # is_going = 0
	sw $t7, is_going
```

```
        lw $k0, 0($sp)   # restore back up
        addi $sp, $sp, -4
        lw $at, 0($sp)
        addi $sp, $sp, -4

GO_end:
        jr $ra

#--------------------------------------------------------
# STOP()
#--------------------------------------------------------
STOP:
        addi $sp, $sp, 4   # backup
        sw $at, 0($sp)

        li $at, MOVING   # change MOVING port to 0
        sb $zero, 0($at)  # to stop

        sw $zero, is_going  # is_going = 0

        lw $at, 0($sp)   # restore back up
        addi $sp, $sp, -4

STOP_end:
        jr $ra
```

```
#-------------------------------------------------------

# TRACK()

#-------------------------------------------------------

TRACK:

        addi $sp, $sp, 4   # backup

        sw $at, 0($sp)

        addi $sp, $sp, 4

        sw $k0, 0($sp)


        li $at, LEAVETRACK  # change LEAVETRACK port

        addi $k0, $zero,1  # to logic 1,

        sb $k0, 0($at)   # to start tracking


        addi $s0, $zero, 1

        sw $s0, is_tracking


        lw $k0, 0($sp)   # restore back up

        addi $sp, $sp, -4

        lw $at, 0($sp)

        addi $sp, $sp, -4


TRACK_end:

        jr $ra


#-------------------------------------------------------
```

```
# UNTRACK()
#-------------------------------------------------------
UNTRACK:
        addi $sp, $sp, 4  # backup
        sw $at, 0($sp)

        li $at, LEAVETRACK # change LEAVETRACK port to 0
        sb $zero, 0($at) # to stop drawing tail

        sw $zero, is_tracking

        lw $at, 0($sp)  # restore back up
        addi $sp, $sp, -4

UNTRACK_end:
        jr $ra

#-------------------------------------------------------
# ROTATE()
#-------------------------------------------------------
ROTATE:
        addi $sp, $sp, 4  # backup
        sw $t1, 0($sp)
        addi $sp, $sp, 4
        sw $t2, 0($sp)
        addi $sp, $sp, 4
        sw $t3, 0($sp)

        li $t1, HEADING # change HEADING port
        la $t2, a_now
        lw $t3, 0($t2)  # $t3 is heading at now
        sw $t3, 0($t1)  # to rotate robot

        lw $t3, 0($sp)  # restore back up
        addi $sp, $sp, -4
        lw $t2, 0($sp)
        addi $sp, $sp, -4
        lw $t1, 0($sp)
        addi $sp, $sp, -4

ROTATE_end:
        jr $ra

#================================================================
====================
# Procedure for string
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~
# strcmp()
# - input: $s3 = string to compare with control_code
# - output: $t0 = 0 if not equal, 1 if equal
#---------------------------------------------------------
```

```
strcmp:
        addi $sp, $sp, 4   # back up
        sw $t1, 0($sp)
        addi $sp, $sp, 4
        sw $s1, 0($sp)
        addi $sp,$sp,4
        sw $t2, 0($sp)
        addi $sp, $sp, 4
        sw $t3, 0($sp)

        xor $t0, $zero, $zero  # $t1 = return value = 0
        xor $t1, $zero, $zero  # $t1 = i = 0

strcmp_loop:
        beq $t1, 3, strcmp_equal  # if i = 3 -> end loop -> equal
        nop

        lb $t2, control_code($t1)  # $t2 = control_code[i]

        add $t3, $s3, $t1  # $t3 = s + i
        lb $t3, 0($t3)   # $t3 = s[i]

        beq $t2, $t3, strcmp_next  # if $t2 == $t3 -> continue the loop
        nop

        j strcmp_end
```

```
strcmp_next:
	addi $t1, $t1, 1
	j strcmp_loop

strcmp_equal:
	add $t0, $zero, 1  # i++

strcmp_end:
	lw $t3, 0($sp)   # restore the backup
	addi $sp, $sp, -4
	lw $t2, 0($sp)
	addi $sp, $sp, -4
	lw $s1, 0($sp)
	addi $sp, $sp, -4
	lw $t1, 0($sp)
	addi $sp, $sp, -4

	jr $ra

#---------------------------------------------------------
# strClear()
#---------------------------------------------------------
strClear:
	addi $sp, $sp, 4   # backup
	sw $t1, 0($sp)
```

```
        addi $sp, $sp, 4
        sw $t2, 0($sp)
        addi $sp, $sp, 4
        sw $s1, 0($sp)
        addi $sp, $sp, 4
        sw $t3, 0($sp)
        addi $sp, $sp, 4
        sw $s2, 0($sp)

        lw $t3, code_length   # $t3 = code_length
        addi $t1, $zero, -1  # $t1 = -1 = i

strClear_loop:
        addi $t1, $t1, 1   # i++
        sb $zero, control_code  # control_code[i] = '\0'

        bne $t1, $t3, strClear_loop # if $t1 <=3 resetInput loop
        nop

        sw $zero, code_length  # reset code_length = 0

strClear_end:
        lw $s2, 0($sp)   # restore backup
        addi $sp, $sp, -4
        lw $t3, 0($sp)
        addi $sp, $sp, -4
        lw $s1, 0($sp)
        addi $sp, $sp, -4
        lw $t2, 0($sp)
        addi $sp, $sp, -4
        lw $t1, 0($sp)
        addi $sp, $sp, -4

        jr $ra

#---------------------------------------------------------
# strcpyPrevToCur(): copy value from prev to current code
#---------------------------------------------------------
strcpyPrevToCur:
        addi $sp, $sp, 4   # backup
        sw $t1, 0($sp)
        addi $sp, $sp, 4
        sw $t2, 0($sp)
        addi $sp, $sp, 4
        sw $s1, 0($sp)
        addi $sp, $sp, 4
        sw $t3, 0($sp)
        addi $sp, $sp, 4
        sw $s2, 0($sp)

        li $t2, 0
        # load address of control_code
```

```
    la $s1, control_code

    # load address of prev_control_code
    la $s2, prev_code

strCpy1_loop:
    beq $t2, 3, strCpy1_end

    # $t1 as control_code[i]
    lb $t1, 0($s2)
    sb $t1, 0($s1)

    addi $s1, $s1, 1
    addi $s2, $s2, 1
    addi $t2, $t2, 1

    j strCpy1_loop

strCpy1_end:
    # reset code length
    li $t3, 3
    sw $t3, code_length

    lw $s2, 0($sp)   # restore backup
    addi $sp, $sp, -4
    lw $t3, 0($sp)
```

```
        addi $sp, $sp, -4
        lw $s1, 0($sp)
        addi $sp, $sp, -4
        lw $t2, 0($sp)
        addi $sp, $sp, -4
        lw $t1, 0($sp)
        addi $sp, $sp, -4

        jr $ra


#----------------------------------------------------------
# strcpyCurToPrev(): copy value from current code to prev code
#----------------------------------------------------------
strCpy2:
        addi $sp, $sp, 4   # backup
        sw $t1, 0($sp)
        addi $sp, $sp, 4
        sw $t2, 0($sp)
        addi $sp, $sp, 4
        sw $s1, 0($sp)
        addi $sp, $sp, 4
        sw $t3, 0($sp)
        addi $sp, $sp, 4
        sw $s2, 0($sp)
        li $t2, 0
        # load address of prev_control_code
        la $s1, prev_code

        # load address of control_code
        la $s2, control_code

strCpy2_loop:
        beq $t2, 3, strCpy2_end

        # $t1 as control_code[i]
        lb $t1, 0($s2)
        sb $t1, 0($s1)

        addi $s1, $s1, 1
        addi $s2, $s2, 1
        addi $t2, $t2, 1

        j strCpy2_loop

strCpy2_end:
        lw $s2, 0($sp)   # restore backup
        addi $sp, $sp, -4
        lw $t3, 0($sp)
        addi $sp, $sp, -4
        lw $s1, 0($sp)
        addi $sp, $sp, -4
        lw $t2, 0($sp)
        addi $sp, $sp, -4
        lw $t1, 0($sp)
        addi $sp, $sp, -4

        jr $ra

#==================================================================
=====================
# GENERAL INTERRUPT SERVED ROUTINE for all interrupts
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~
.ktext 0x80000180
#------------------------------------------------------------
# SAVE the current REG FILE to stack
#------------------------------------------------------------
backup:
        addi $sp, $sp, 4
        sw $ra, 0($sp)

        addi $sp, $sp, 4
        sw $t1, 0($sp)

        addi $sp, $sp, 4
        sw $t2, 0($sp)
```

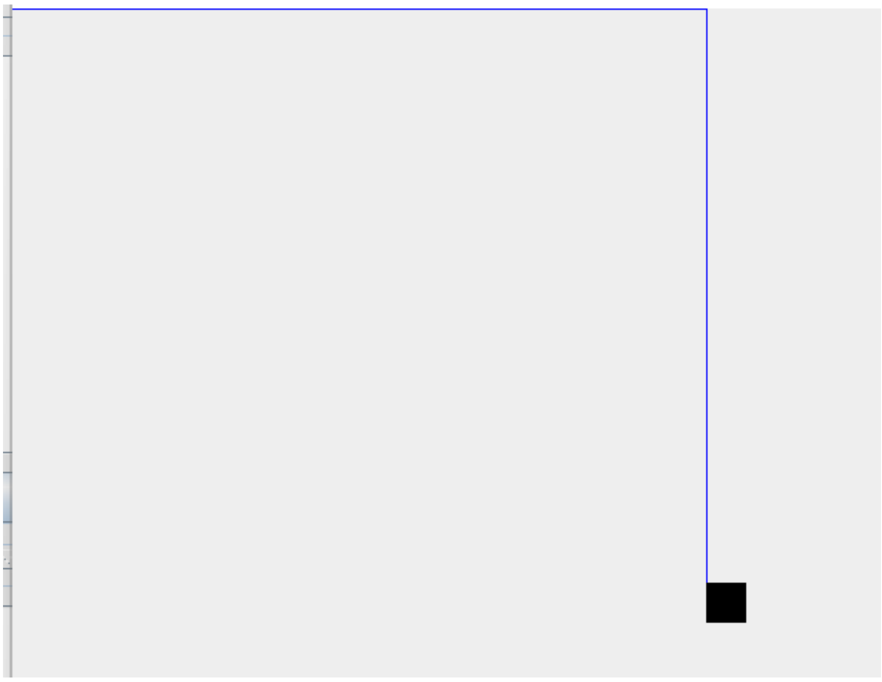
```
        addi $sp, $sp, 4
        sw $t3, 0($sp)

        addi $sp, $sp, 4
        sw $a0, 0($sp)

        addi $sp, $sp, 4
        sw $at, 0($sp)
        addi $sp, $sp, 4
        sw $s0, 0($sp)
        addi $sp, $sp, 4
        sw $s1, 0($sp)
        addi $sp, $sp, 4
        sw $s2, 0($sp)
        addi $sp, $sp, 4
        sw $t4, 0($sp)
        addi $sp, $sp, 4
        sw $s3, 0($sp)
        #-------------------------------------------------------
        # Processing
        #-------------------------------------------------------
getCode:
        li $t1, IN_ADRESS_HEXA_KEYBOARD
        li $t2, OUT_ADRESS_HEXA_KEYBOARD
```

```
	# scan row 1
	li $t3, 0x81
	sb $t3, 0($t1)
	lbu $a0, 0($t2)
	bnez $a0, getCodeInChar

	# scan row 2
	li $t3, 0x82
	sb $t3, 0($t1)
	lbu $a0, 0($t2)
	bnez $a0, getCodeInChar

	# scan row 3
	li $t3, 0x84
	sb $t3, 0($t1)
	lbu $a0, 0($t2)
	bnez $a0, getCodeInChar

	# scan row 4
	li $t3, 0x88
	sb $t3, 0($t1)
	lbu $a0, 0($t2)
	bnez $a0, getCodeInChar

getCodeInChar:
	beq $a0, key0, case_0
```

```
        beq $a0, key1, case_1

        beq $a0, key2, case_2

        beq $a0, key3, case_3

        beq $a0, key4, case_4

        beq $a0, key5, case_5

        beq $a0, key6, case_6

        beq $a0, key7, case_7

        beq $a0, key8, case_8

        beq $a0, key9, case_9

        beq $a0, keya, case_a

        beq $a0, keyb, case_b

        beq $a0, keyc, case_c

        beq $a0, keyd, case_d

        beq $a0, keye, case_e

        beq $a0, keyf, case_f


case_0:

        li $s0, '0'  # $s0 store code in char type

        j storeCode

case_1:

        li $s0, '1'

        j storeCode

case_2:

        li $s0, '2'

        j storeCode

case_3:
        li $s0, '3'
        j storeCode
case_4:
        li $s0, '4'
        j storeCode
case_5:
        li $s0, '5'
        j storeCode
case_6:
        li $s0, '6'
        j storeCode
case_7:
        li $s0, '7'
        j storeCode
case_8:
        li $s0, '8'
        j storeCode
case_9:
        li $s0, '9'
        j storeCode
case_a:
        li $s0, 'a'
        j storeCode
case_b:
        li $s0, 'b'
        j storeCode
```

```
case_c:
        li $s0, 'c'
        j storeCode
case_d:
        li $s0, 'd'
        j storeCode
case_e:
        li $s0, 'e'
        j storeCode
case_f:
        li $s0, 'f'
        j storeCode

storeCode:
        la  $s1, control_code
        la $s2, code_length
        lw $s3, 0($s2)   # $s3 = strlen(control_code)
        addi $t4, $t4, -1   # $t4 = i

storeCodeLoop:
        addi $t4, $t4, 1
        bne $t4, $s3, storeCodeLoop
        add $s1, $s1, $t4  # $s1 = control_code + i
        sb $s0, 0($s1)   # control_code[i] = $s0

        addi $s0, $zero, '\n'  # add '\n' character to end of string
        addi $s1, $s1, 1
        sb $s0, 0($s1)


        addi $s3, $s3, 1
        sw $s3, 0($s2)   # update code_length


#-----------------------------------------------------
# Evaluate the return address of main routine
# epc <= epc + 4
#-----------------------------------------------------
next_pc:
        mfc0 $at, $14  # $at <= Coproc0.$14 = Coproc0.epc
        addi $at, $at, 4  # $at = $at + 4 (next instruction)
        mtc0 $at, $14  # Coproc0.$14 = Coproc0.epc <= $at
#-----------------------------------------------------
# RESTORE the REG FILE from STACK
#-----------------------------------------------------
restore:
        lw $s3, 0($sp)
        addi $sp, $sp, -4
        lw $t4, 0($sp)
        addi $sp, $sp, -4
        lw $s2, 0($sp)
        addi $sp, $sp, -4
        lw $s1, 0($sp)
        addi $sp, $sp, -4
        lw $s0, 0($sp)
        addi $sp, $sp, -4
        lw $at, 0($sp)
        addi $sp, $sp, -4
        lw $a0, 0($sp)
        addi $sp, $sp, -4
        lw $t3, 0($sp)
        addi $sp, $sp, -4
        lw $t2, 0($sp)
        addi $sp, $sp, -4
        lw $t1, 0($sp)
        addi $sp, $sp, -4
        lw $ra, 0($sp)
        addi $sp, $sp, -4
return: eret # Return from exception
```

I.4. Result



II. PROBLEM 6

II.1. Method

- The program displays a menu which allows user to chooses an operation to perform or exit the program.

- Each operation will ask user for input values and display the corresponding results.

II.2. Algorithm

1) The word memory allocation has an error since the rule that the word address must be divisible by 4 is not guaranteed. Fix this error.

- The malloc function first checks if the size of the elements to be allocated is equal to 4, if not (1-byte) allocate memory.

- If the elements' size is 4, the function check if the first free address is divisible by 4 by calculating the remainder when the current address is divided by 4 (using AND operator). If yes, memory is allocated. If not, the address is incremented by 4 and the remainder is then subtracted from the address to move to the next address divisible by 4.

2) Write a function to get the value of the pointer.

- The function getValue first loads the value stored at the memory address pointed to by the pointer **$a0** into register **$v1.**

- Then it checks the size of the element.

+ If elements' size is 1, branches to loadByte which uses lb to load the value

+ If elements's size is 4, branches to loadWord, which uses lw to load the value

3) Write a function to get the address of the pointer.

- Use lw to load the address of the pointer

4) Write a function to copy two character pointers.

- Load the character from the source string at the position indicated by the source pointer .Store the loaded character into the target string at the destination pointer.

- Increment both the source and target pointers by 1.

- Loop continues until the null terminator **'\0'** is reached.

5) Write a function to free the memory allocated to pointers.

- Store the address of the first allocated memory at the memory location of the top of free memory. This marks the entire allocated memory space as free.

6) Write a function to calculate the amount of allocated memory.

- Subtract the address of the first allocated memory from the top of free memory to

7) Write a function malloc2 to allocate a 2-dimensional array of type .word with parameters including:

a. The starting address of the array

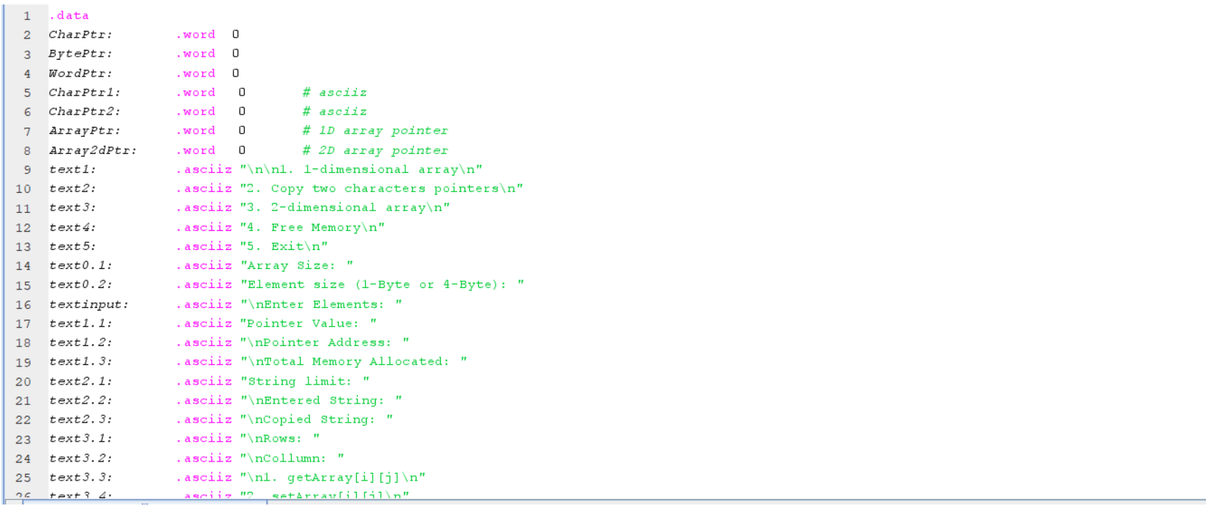
b. Number of rows

c. Number of columns

8) Bases on question 7, write two functions getArray[i][j] and setArray[i][j] to get/set the value for the element in row i column j of the array

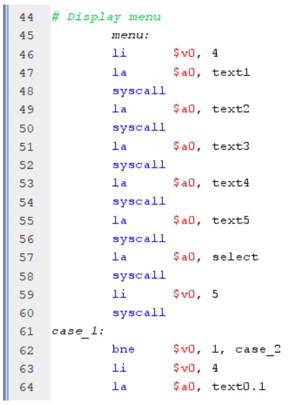
II.3. Source Code

a. Data initilization



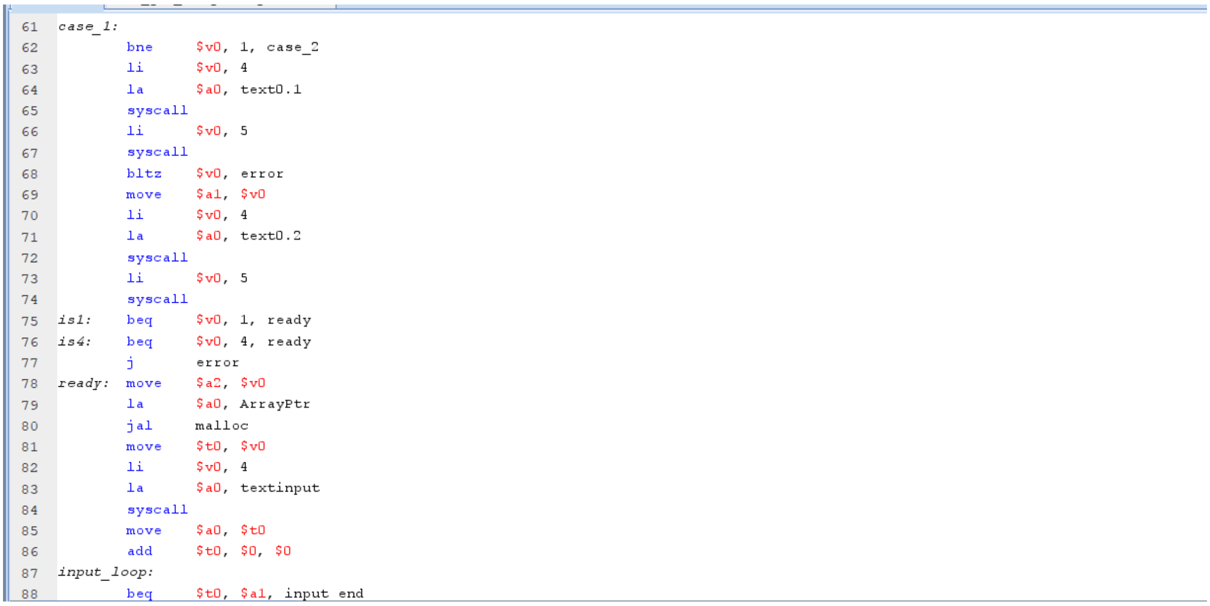
```
.data
CharPtr:        .word  0
BytePtr:        .word  0
WordPtr:        .word  0
CharPtr1:       .word   0       # asciiz
CharPtr2:       .word   0       # asciiz
ArrayPtr:       .word   0       # 1D array pointer
Array2dPtr:     .word   0       # 2D array pointer
text1:          .asciiz "\n\n1. 1-dimensional array\n"
text2:          .asciiz "2. Copy two characters pointers\n"
text3:          .asciiz "3. 2-dimensional array\n"
text4:          .asciiz "4. Free Memory\n"
text5:          .asciiz "5. Exit\n"
text0.1:        .asciiz "Array Size: "
text0.2:        .asciiz "Element size (1-Byte or 4-Byte): "
textinput:      .asciiz "\nEnter Elements: "
text1.1:        .asciiz "Pointer Value: "
text1.2:        .asciiz "\nPointer Address: "
text1.3:        .asciiz "\nTotal Memory Allocated: "
text2.1:        .asciiz "String limit: "
text2.2:        .asciiz "\nEntered String: "
text2.3:        .asciiz "\nCopied String: "
text3.1:        .asciiz "\nRows: "
text3.2:        .asciiz "\nCollumn: "
text3.3:        .asciiz "\n1. getArray[i][j]\n"
text3.4:        .asciiz "2. setArray[i][j]\n"
```

b. Menu Display

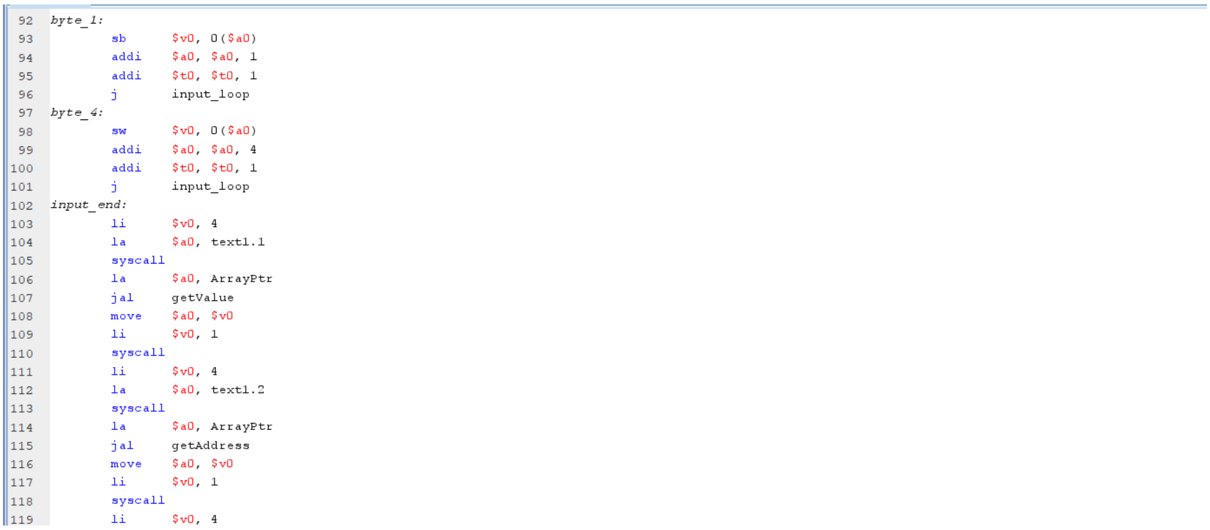


```
# Display menu
        menu:
        li      $v0, 4
        la      $a0, text1
        syscall
        la      $a0, text2
        syscall
        la      $a0, text3
        syscall
        la      $a0, text4
        syscall
        la      $a0, text5
        syscall
        la      $a0, select
        syscall
        li      $v0, 5
        syscall
case_1:
        bne     $v0, 1, case_2
        li      $v0, 4
        la      $a0, text0.1
```

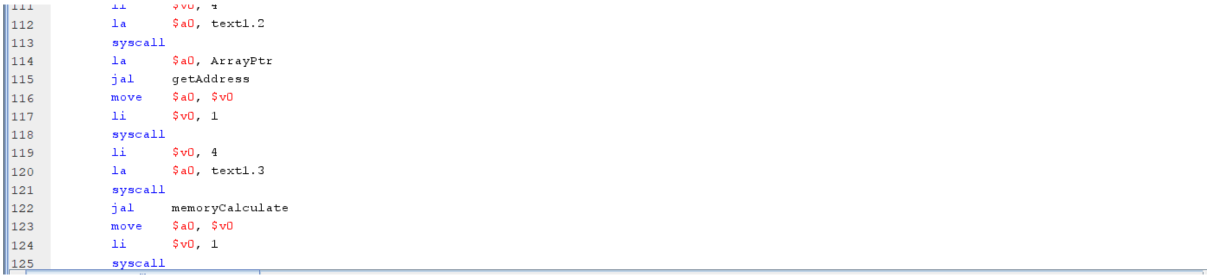
b.1. Option 1 values input and result display (1-byte or 4-byte 1 dimensional array)



```
case_1:
        bne     $v0, 1, case_2
        li      $v0, 4
        la      $a0, text0.1
        syscall
        li      $v0, 5
        syscall
        bltz    $v0, error
        move    $a1, $v0
        li      $v0, 4
        la      $a0, text0.2
        syscall
        li      $v0, 5
        syscall
is1:    beq     $v0, 1, ready
is4:    beq     $v0, 4, ready
        j       error
ready:  move    $a2, $v0
        la      $a0, ArrayPtr
        jal     malloc
        move    $t0, $v0
        li      $v0, 4
        la      $a0, textinput
        syscall
        move    $a0, $t0
        add     $t0, $0, $0
input_loop:
        beq     $t0, $a1, input_end
```

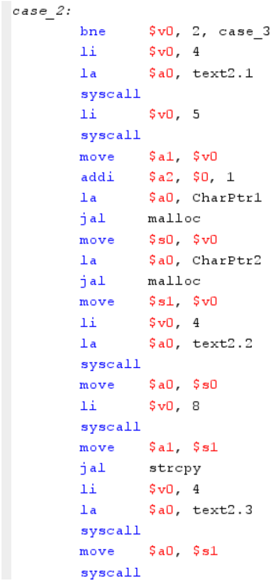


```
byte_1:
        sb      $v0, 0($a0)
        addi    $a0, $a0, 1
        addi    $t0, $t0, 1
        j       input_loop
byte_4:
        sw      $v0, 0($a0)
        addi    $a0, $a0, 4
        addi    $t0, $t0, 1
        j       input_loop
input_end:
        li      $v0, 4
        la      $a0, text1.1
        syscall
        la      $a0, ArrayPtr
        jal     getValue
        move    $a0, $v0
        li      $v0, 1
        syscall
        li      $v0, 4
        la      $a0, text1.2
        syscall
        la      $a0, ArrayPtr
        jal     getAddress
        move    $a0, $v0
        li      $v0, 1
        syscall
        li      $v0, 4
```



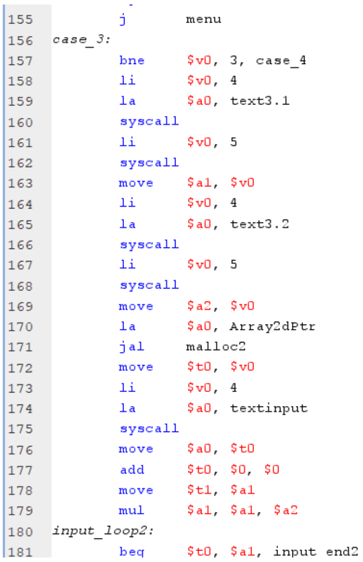
```
        li      $v0, 4
        la      $a0, text1.2
        syscall
        la      $a0, ArrayPtr
        jal     getAddress
        move    $a0, $v0
        li      $v0, 1
        syscall
        li      $v0, 4
        la      $a0, text1.3
        syscall
        jal     memoryCalculate
        move    $a0, $v0
        li      $v0, 1
        syscall
```

b.2. Option 2 : String character copy



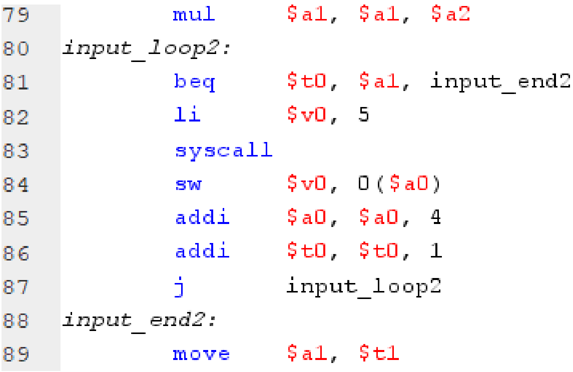
```
case_2:
        bne     $v0, 2, case_3
        li      $v0, 4
        la      $a0, text2.1
        syscall
        li      $v0, 5
        syscall
        move    $a1, $v0
        addi    $a2, $0, 1
        la      $a0, CharPtr1
        jal     malloc
        move    $s0, $v0
        la      $a0, CharPtr2
        jal     malloc
        move    $s1, $v0
        li      $v0, 4
        la      $a0, text2.2
        syscall
        move    $a0, $s0
        li      $v0, 8
        syscall
        move    $a1, $s1
        jal     strcpy
        li      $v0, 4
        la      $a0, text2.3
        syscall
        move    $a0, $s1
        syscall
```

b.3. Option 3 : 2D array



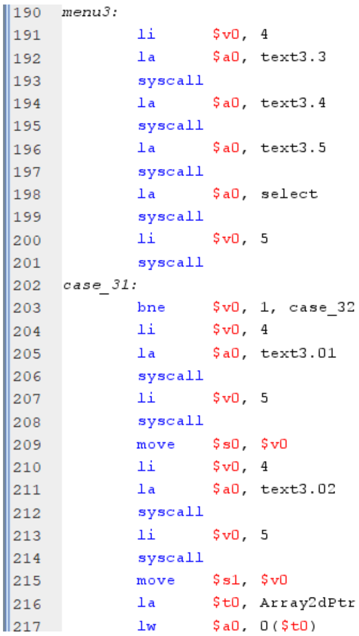
```
        j       menu
case_3:
        bne     $v0, 3, case_4
        li      $v0, 4
        la      $a0, text3.1
        syscall
        li      $v0, 5
        syscall
        move    $a1, $v0
        li      $v0, 4
        la      $a0, text3.2
        syscall
        li      $v0, 5
        syscall
        move    $a2, $v0
        la      $a0, Array2dPtr
        jal     malloc2
        move    $t0, $v0
        li      $v0, 4
        la      $a0, textinput
        syscall
        move    $a0, $t0
        add     $t0, $0, $0
        move    $t1, $a1
        mul     $a1, $a1, $a2
input_loop2:
        beq     $t0, $a1, input_end2
```

- 2D array input :

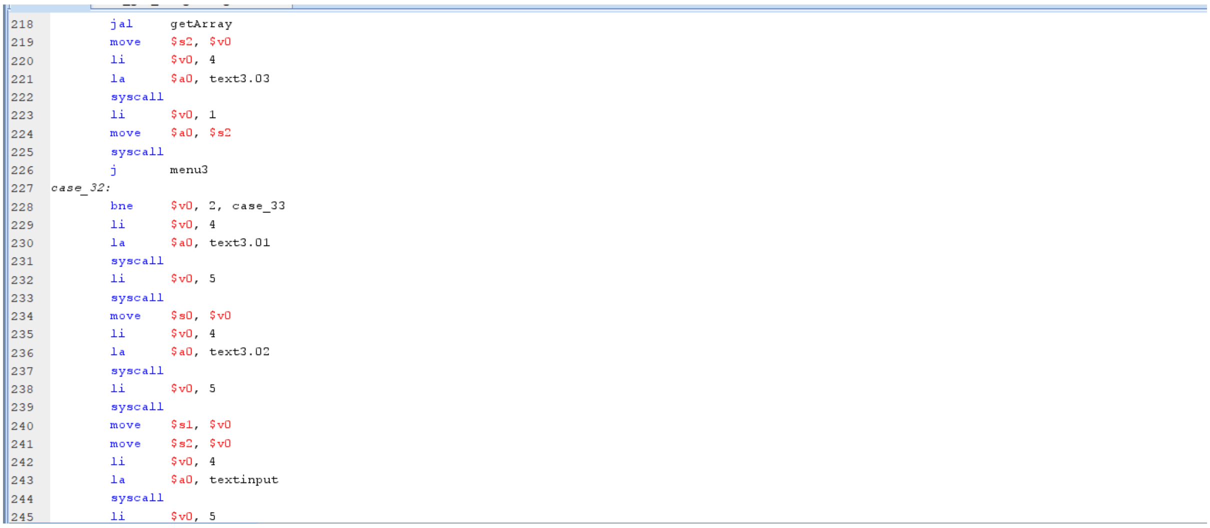


```
        mul     $a1, $a1, $a2
input_loop2:
        beq     $t0, $a1, input_end2
        li      $v0, 5
        syscall
        sw      $v0, 0($a0)
        addi    $a0, $a0, 4
        addi    $t0, $t0, 1
        j       input_loop2
input_end2:
        move    $a1, $t1
```

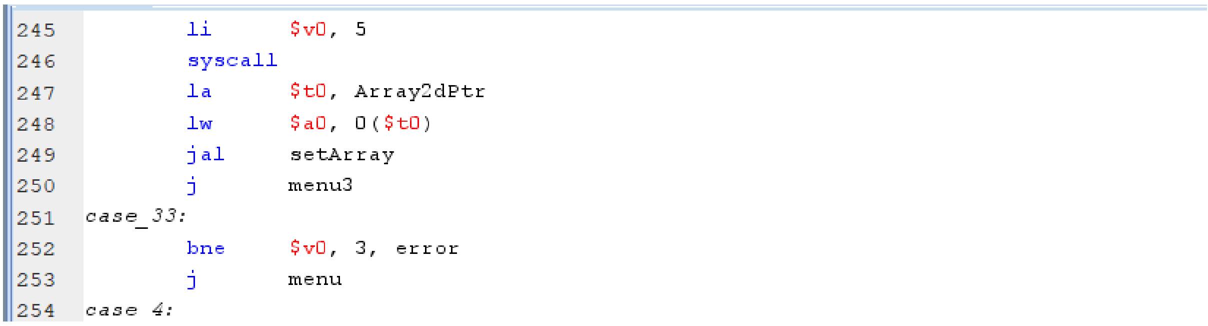
- Display sub-menu for option 3 (getArray, setArray or return)



```
menu3:
        li      $v0, 4
        la      $a0, text3.3
        syscall
        la      $a0, text3.4
        syscall
        la      $a0, text3.5
        syscall
        la      $a0, select
        syscall
        li      $v0, 5
        syscall
case_31:
        bne     $v0, 1, case_32
        li      $v0, 4
        la      $a0, text3.01
        syscall
        li      $v0, 5
        syscall
        move    $s0, $v0
        li      $v0, 4
        la      $a0, text3.02
        syscall
        li      $v0, 5
        syscall
        move    $s1, $v0
        la      $t0, Array2dPtr
        lw      $a0, 0($t0)
```

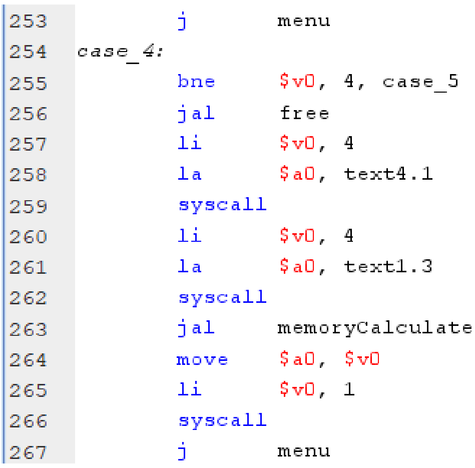


```
        jal     getArray
        move    $s2, $v0
        li      $v0, 4
        la      $a0, text3.03
        syscall
        li      $v0, 1
        move    $a0, $s2
        syscall
        j       menu3
case_32:
        bne     $v0, 2, case_33
        li      $v0, 4
        la      $a0, text3.01
        syscall
        li      $v0, 5
        syscall
        move    $s0, $v0
        li      $v0, 4
        la      $a0, text3.02
        syscall
        li      $v0, 5
        syscall
        move    $s1, $v0
        move    $s2, $v0
        li      $v0, 4
        la      $a0, textinput
        syscall
        li      $v0, 5
```



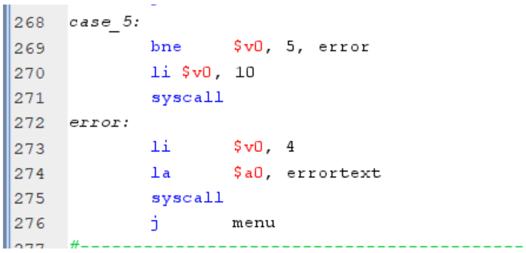
```
        li      $v0, 5
        syscall
        la      $t0, Array2dPtr
        lw      $a0, 0($t0)
        jal     setArray
        j       menu3
case_33:
        bne     $v0, 3, error
        j       menu
case 4:
```

b.4. Option 4 : Free memory



```
        j       menu
case_4:
        bne     $v0, 4, case_5
        jal     free
        li      $v0, 4
        la      $a0, text4.1
        syscall
        li      $v0, 4
        la      $a0, text1.3
        syscall
        jal     memoryCalculate
        move    $a0, $v0
        li      $v0, 1
        syscall
        j       menu
```

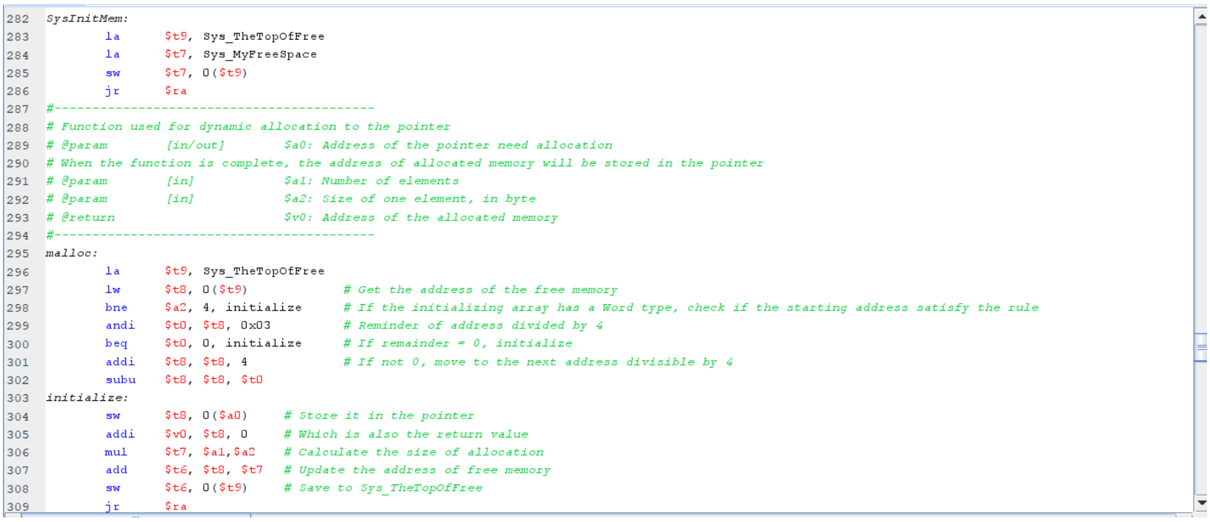
b.5. Option 5 : Exit



```
case_5:
        bne     $v0, 5, error
        li $v0, 10
        syscall
error:
        li      $v0, 4
        la      $a0, errortext
        syscall
        j       menu
#-----------------------------------------
```

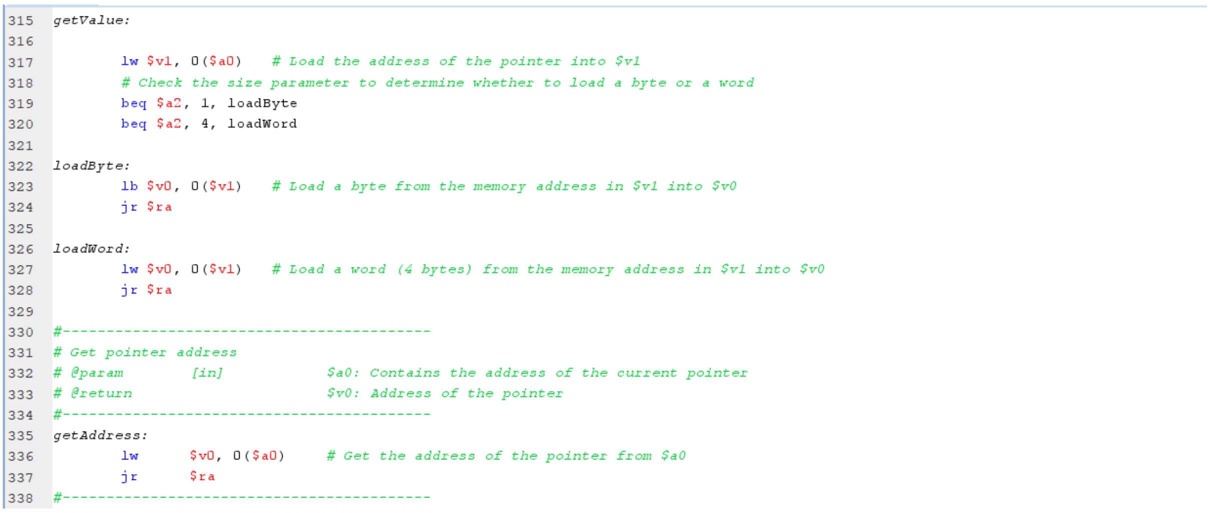
- error text is display when an invalid option is entered

c. Malloc function:



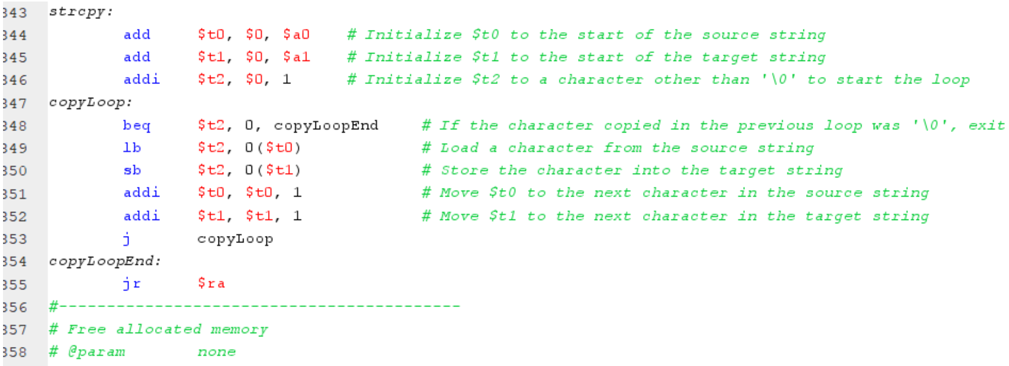
```
SysInitMem:
        la      $t9, Sys_TheTopOfFree
        la      $t7, Sys_MyFreeSpace
        sw      $t7, 0($t9)
        jr      $ra
#-------------------------------------------
# Function used for dynamic allocation to the pointer
# @param        [in/out]        $a0: Address of the pointer need allocation
# When the function is complete, the address of allocated memory will be stored in the pointer
# @param        [in]            $a1: Number of elements
# @param        [in]            $a2: Size of one element, in byte
# @return                       $v0: Address of the allocated memory
#-------------------------------------------
malloc:
        la      $t9, Sys_TheTopOfFree
        lw      $t8, 0($t9)             # Get the address of the free memory
        bne     $a2, 4, initialize      # If the initializing array has a Word type, check if the starting address satisfy the rule
        andi    $t0, $t8, 0x03          # Reminder of address divided by 4
        beq     $t0, 0, initialize      # If remainder = 0, initialize
        addi    $t8, $t8, 4             # If not 0, move to the next address divisible by 4
        subu    $t8, $t8, $t0
initialize:
        sw      $t8, 0($a0)     # Store it in the pointer
        addi    $v0, $t8, 0     # Which is also the return value
        mul     $t7, $a1,$a2    # Calculate the size of allocation
        add     $t6, $t8, $t7   # Update the address of free memory
        sw      $t6, 0($t9)     # Save to Sys_TheTopOfFree
        jr      $ra
```

d. Get value and get address function:



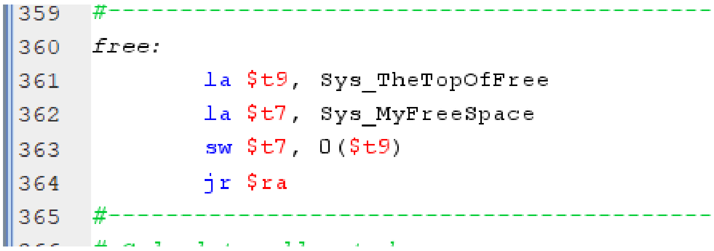
```
getValue:

        lw $v1, 0($a0)    # Load the address of the pointer into $v1
        # Check the size parameter to determine whether to load a byte or a word
        beq $a2, 1, loadByte
        beq $a2, 4, loadWord

loadByte:
        lb $v0, 0($v1)    # Load a byte from the memory address in $v1 into $v0
        jr $ra

loadWord:
        lw $v0, 0($v1)    # Load a word (4 bytes) from the memory address in $v1 into $v0
        jr $ra

#-------------------------------------------
# Get pointer address
# @param        [in]            $a0: Contains the address of the current pointer
# @return                       $v0: Address of the pointer
#-------------------------------------------
getAddress:
        lw      $v0, 0($a0)     # Get the address of the pointer from $a0
        jr      $ra
#-------------------------------------------
```

e. String copy function:



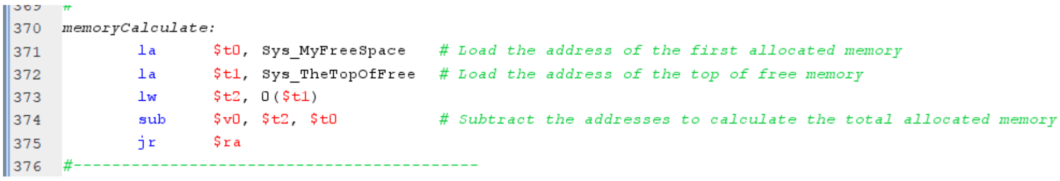
```
strcpy:
        add     $t0, $0, $a0    # Initialize $t0 to the start of the source string
        add     $t1, $0, $a1    # Initialize $t1 to the start of the target string
        addi    $t2, $0, 1      # Initialize $t2 to a character other than '\0' to start the loop
copyLoop:
        beq     $t2, 0, copyLoopEnd     # If the character copied in the previous loop was '\0', exit
        lb      $t2, 0($t0)             # Load a character from the source string
        sb      $t2, 0($t1)             # Store the character into the target string
        addi    $t0, $t0, 1             # Move $t0 to the next character in the source string
        addi    $t1, $t1, 1             # Move $t1 to the next character in the target string
        j       copyLoop
copyLoopEnd:
        jr      $ra
#-------------------------------------------
# Free allocated memory
# @param        none
```

f. Free memory function:



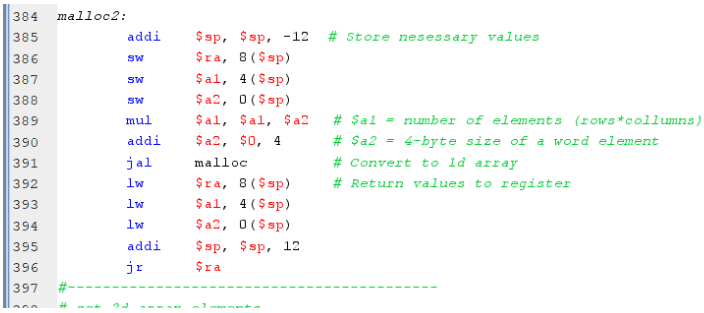
```
#-------------------------------------------
free:
        la $t9, Sys_TheTopOfFree
        la $t7, Sys_MyFreeSpace
        sw $t7, 0($t9)
        jr $ra
#-------------------------------------------
```

g. Calculate allocated memory function:



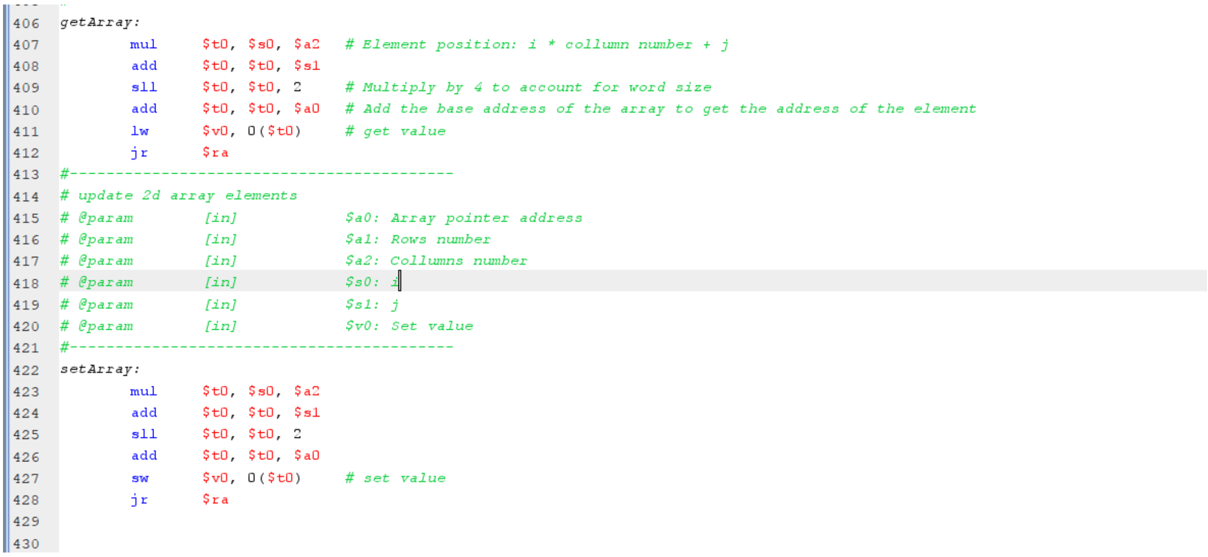
```
memoryCalculate:
        la      $t0, Sys_MyFreeSpace    # Load the address of the first allocated memory
        la      $t1, Sys_TheTopOfFree   # Load the address of the top of free memory
        lw      $t2, 0($t1)
        sub     $v0, $t2, $t0           # Subtract the addresses to calculate the total allocated memory
        jr      $ra
#-------------------------------------------
```

h. Malloc 2 function:



```
malloc2:
        addi    $sp, $sp, -12  # Store nesessary values
        sw      $ra, 8($sp)
        sw      $a1, 4($sp)
        sw      $a2, 0($sp)
        mul     $a1, $a1, $a2   # $a1 = number of elements (rows*collumns)
        addi    $a2, $0, 4      # $a2 = 4-byte size of a word element
        jal     malloc          # Convert to 1d array
        lw      $ra, 8($sp)     # Return values to register
        lw      $a1, 4($sp)
        lw      $a2, 0($sp)
        addi    $sp, $sp, 12
        jr      $ra
#-------------------------------------------
```

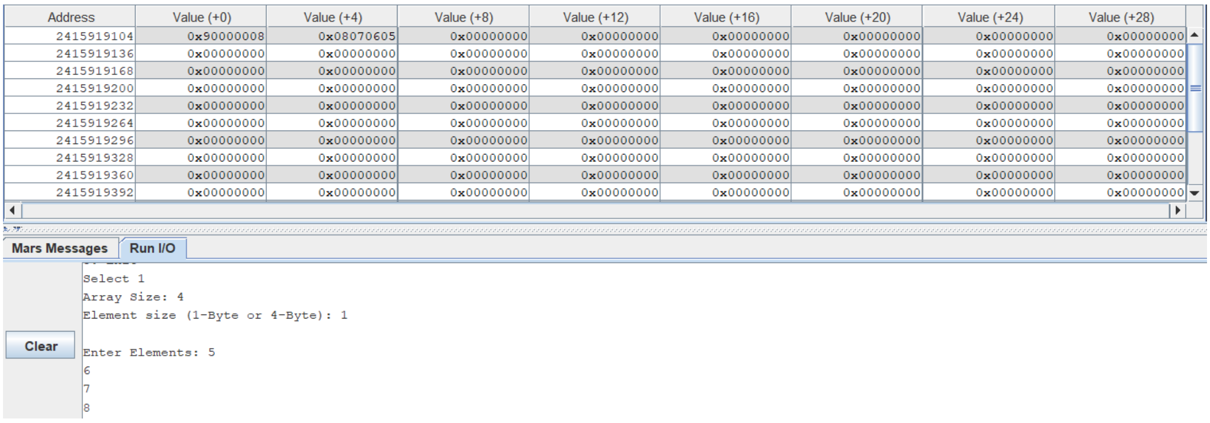
i. Get array and set array function:



```
getArray:
        mul     $t0, $s0, $a2   # Element position: i * collumn number + j
        add     $t0, $t0, $s1
        sll     $t0, $t0, 2     # Multiply by 4 to account for word size
        add     $t0, $t0, $a0   # Add the base address of the array to get the address of the element
        lw      $v0, 0($t0)     # get value
        jr      $ra
#-------------------------------------------
# update 2d array elements
# @param        [in]            $a0: Array pointer address
# @param        [in]            $a1: Rows number
# @param        [in]            $a2: Collumns number
# @param        [in]            $s0: i
# @param        [in]            $s1: j
# @param        [in]            $v0: Set value
#-------------------------------------------
setArray:
        mul     $t0, $s0, $a2
        add     $t0, $t0, $s1
        sll     $t0, $t0, 2
        add     $t0, $t0, $a0
        sw      $v0, 0($t0)     # set value
        jr      $ra
```

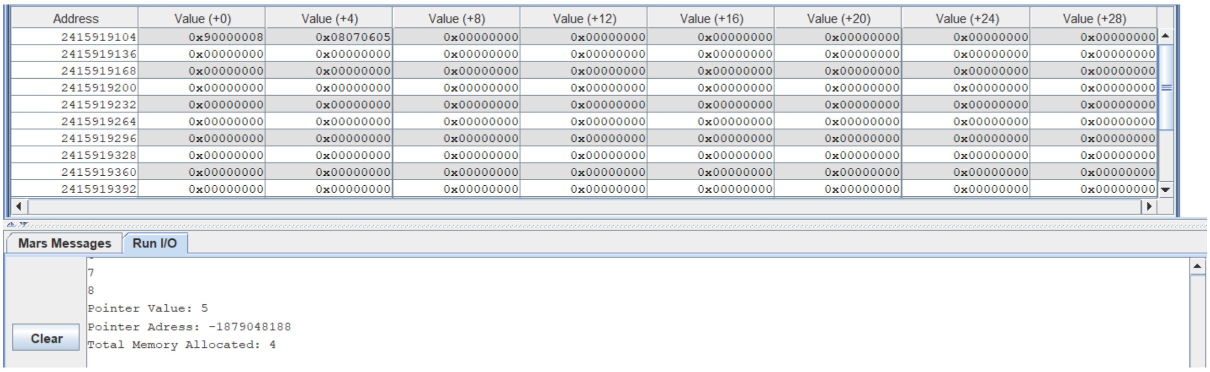
II.4. Result

For 1-byte array with 4 elements 5,6,7,8 :



Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
2415919104	0x90000008	0x08070605	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
2415919136	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
2415919168	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
2415919200	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
2415919232	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
2415919264	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
2415919296	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
2415919328	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
2415919360	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
2415919392	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

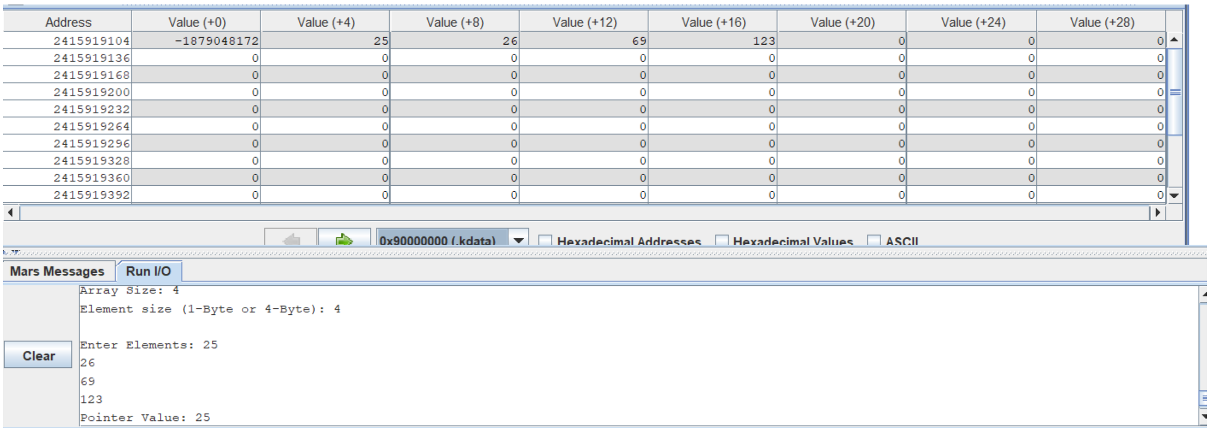
```
Select 1
Array Size: 4
Element size (1-Byte or 4-Byte): 1

Enter Elements: 5
6
7
8
```



Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
2415919104	0x90000008	0x08070605	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
2415919136	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
2415919168	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
2415919200	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
2415919232	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
2415919264	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
2415919296	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
2415919328	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
2415919360	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
2415919392	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

```
7
8
Pointer Value: 5
Pointer Adress: -1879048188
Total Memory Allocated: 4
```

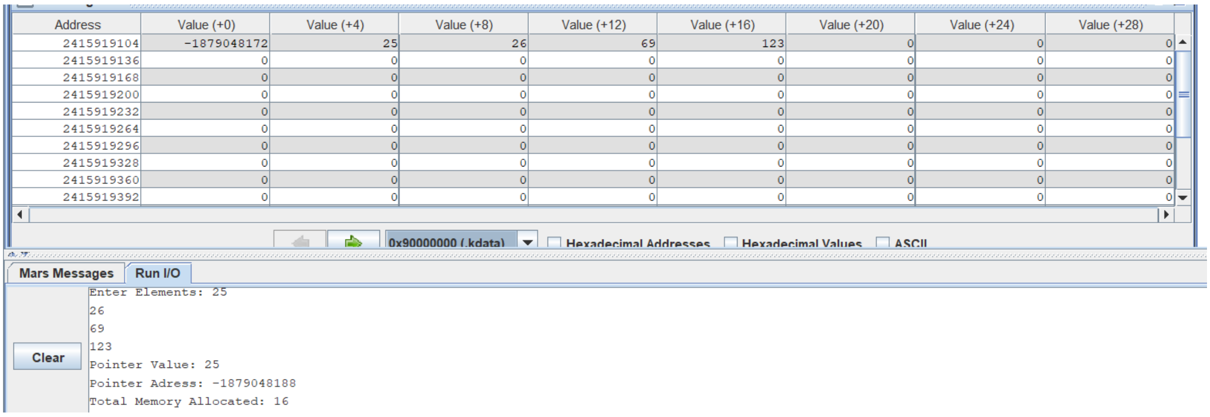
For 4-byte word array with 4 elements 25,26,69,123 :



Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
2415919104	-1879048172	25	26	69	123	0	0	0
2415919136	0	0	0	0	0	0	0	0
2415919168	0	0	0	0	0	0	0	0
2415919200	0	0	0	0	0	0	0	0
2415919232	0	0	0	0	0	0	0	0
2415919264	0	0	0	0	0	0	0	0
2415919296	0	0	0	0	0	0	0	0
2415919328	0	0	0	0	0	0	0	0
2415919360	0	0	0	0	0	0	0	0
2415919392	0	0	0	0	0	0	0	0

0x90000000 (.kdata) □ Hexadecimal Addresses □ Hexadecimal Values □ ASCII

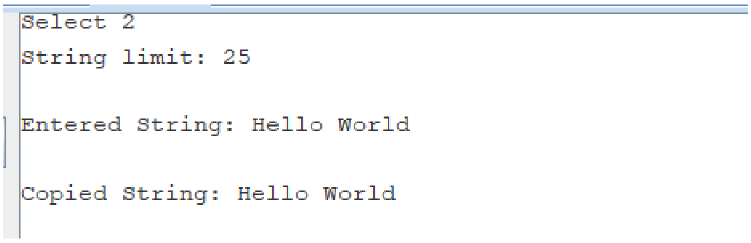
```
Array Size: 4
Element size (1-Byte or 4-Byte): 4

Enter Elements: 25
26
69
123
Pointer Value: 25
```



Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
2415919104	-1879048172	25	26	69	123	0	0	0
2415919136	0	0	0	0	0	0	0	0
2415919168	0	0	0	0	0	0	0	0
2415919200	0	0	0	0	0	0	0	0
2415919232	0	0	0	0	0	0	0	0
2415919264	0	0	0	0	0	0	0	0
2415919296	0	0	0	0	0	0	0	0
2415919328	0	0	0	0	0	0	0	0
2415919360	0	0	0	0	0	0	0	0
2415919392	0	0	0	0	0	0	0	0

```
Enter Elements: 25
26
69
123
Pointer Value: 25
Pointer Adress: -1879048188
Total Memory Allocated: 16
```

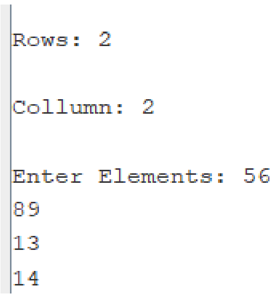
Copy two character pointers



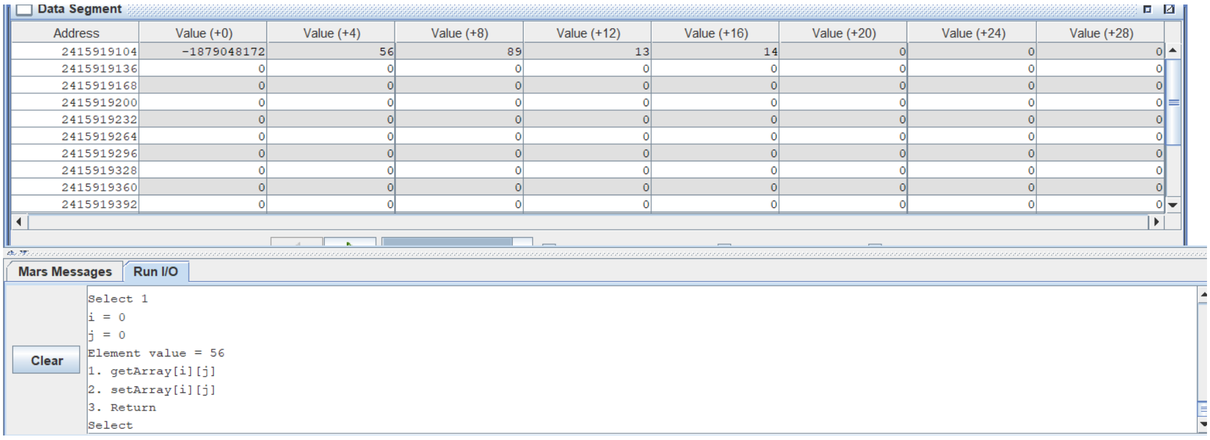
```
Select 2
String limit: 25

Entered String: Hello World

Copied String: Hello World
```

2-dimensional array

- Get element



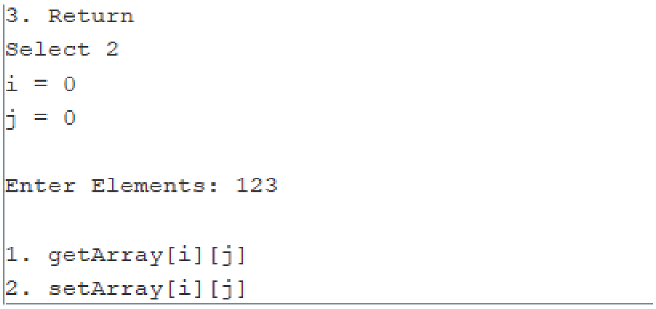
```
Rows: 2

Collumn: 2

Enter Elements: 56
89
13
14
```



Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
2415919104	-1879048172	56	89	13	14	0	0	0
2415919136	0	0	0	0	0	0	0	0
2415919168	0	0	0	0	0	0	0	0
2415919200	0	0	0	0	0	0	0	0
2415919232	0	0	0	0	0	0	0	0
2415919264	0	0	0	0	0	0	0	0
2415919296	0	0	0	0	0	0	0	0
2415919328	0	0	0	0	0	0	0	0
2415919360	0	0	0	0	0	0	0	0
2415919392	0	0	0	0	0	0	0	0

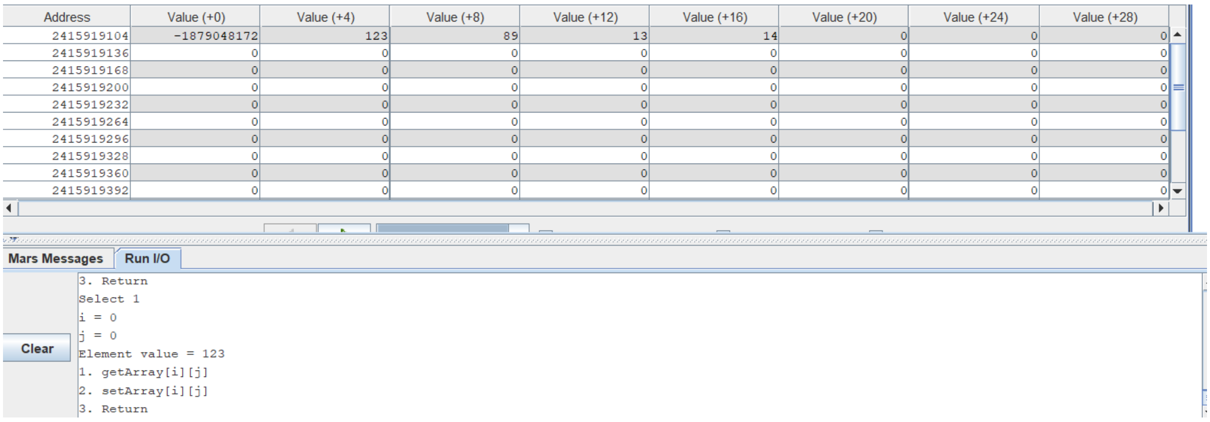
```
Select 1
i = 0
j = 0
Element value = 56
1. getArray[i][j]
2. setArray[i][j]
3. Return
Select
```

- Set element



```
3. Return
Select 2
i = 0
j = 0

Enter Elements: 123

1. getArray[i][j]
2. setArray[i][j]
```

- After the element is updated



Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
2415919104	-1879048172	123	89	13	14	0	0	0
2415919136	0	0	0	0	0	0	0	0
2415919168	0	0	0	0	0	0	0	0
2415919200	0	0	0	0	0	0	0	0
2415919232	0	0	0	0	0	0	0	0
2415919264	0	0	0	0	0	0	0	0
2415919296	0	0	0	0	0	0	0	0
2415919328	0	0	0	0	0	0	0	0
2415919360	0	0	0	0	0	0	0	0
2415919392	0	0	0	0	0	0	0	0

```
3. Return
Select 1
i = 0
j = 0
Element value = 123
1. getArray[i][j]
2. setArray[i][j]
3. Return
```

TÀI LIỆU THAM KHẢO

[1] Tài liệu

[2] Tài liệu

[3] Tài liệu