ĐẠI HỌC BÁCH KHOA HÀ NỘI
Trường Công Nghệ Thông Tin & Truyền Thông

--O  O--

BÁO CÁO PROJECT CUỐI KỲ

Môn: Thực Hành Kiến Trúc Máy Tính

Lớp: 147796

Giảng viên hướng dẫn: **Đỗ Công Thuần**
Nhóm sinh viên thực hiện: **Dương Đức Hiếu 20225624 (chủ đề 1)**
Nguyễn Đức Long 20225876 (chủ đề 2)

Nhóm: 16

Hà Nội, ngày 24 tháng 05 năm 2024

Chủ đề 1: Curiosity Marsbot
Dương Đức Hiếu - 20225624

1. Phân tích bài toán:

- Xe tự hành Curiosity Marsbot chạy trên sao Hỏa, được vận hành từ xa bởi các lập trình viên trên Trái Đất bằng cách gửi các mã điều khiển.
- Các mã điều khiển được nhập từ Digital Lab Sim => cần lưu trữ các mã quét được từ Digital Lab Sim.
- Sau khi nhận mã điều khiển cần nhập lệnh kích hoạt từ Keyboard & Display MMIO Simulator:

    + Enter: Kết thúc nhập mã và yêu cầu Marsbot thực thi.
    ⇨ Trước khi thực hiện cần kiểm tra xem mã có trong kịch bản không?

    + Delete: Xóa toàn bộ mã điều khiển đang nhập.

- Các hành động như di chuyển, dừng, rẽ trái,... thì chỉ cần ra lệnh trực tiếp cho marsbot thực hiện.
- Đặc biệt có hành động quay về theo lộ trình ngược lại thì cần phải lưu trữ lịch sử di chuyển cho marsbot.

2. Cách thực hiện :

- Bước 1: Khi người dùng nhập 1 ký tự từ Digital Lab Sim sẽ tạo ra interrupt để lưu ký tự đó vào bộ nhớ, cứ như vậy cho tới khi người dùng nhập lệnh kích hoạt
    => có được mã điều khiển
- Bước 2: Người dùng nhập lệnh kích hoạt thông qua Keyboard & Display MMIO Simulator suy ra cần kiểm tra liên tục xem ký tự Enter, Delete có được nhập hay không ?
+ Nếu Enter được nhập chuyển sang Bước 3;
+ Nếu Delete được nhập chuyển sang Bước 4;
+ Nếu không thì tiếp tục Bước 2.
- Bước 3:
+ Hiển thị mã điều khiển ra console
+ Kiểm tra mã điều khiển có trong kịch bản không?
    ⇨ nếu có: thực hiện hành động tương ứng
    ⇨ nếu không: in mã không hợp lệ ra console.

- Bước 4: Xóa lưu trữ mã điều khiển trong bộ nhớ.
- Bước 5: Lặp lại các lệnh vừa thực hiện

3. Các hàm thực hiện:

- Hàm main

Các nhãn và công việc tương ứng của từng nhãn trong hàm main như sau:
+ setStartHeading: set góc đầu tiên của Marsbot là góc 0 độ
+ print_error: in ra thông báo lỗi
+ print_current_code: in ra mã điều khiển vừa nhập vào
+ resetInput: xóa mã điều khiển đã nhập để chuẩn bị cho mã tiếp theo
+ waitForKey: chờ phím được nhấn từ Digital Lab Sim
+ readKey: đọc ký tự được nhập vào từ Keyboard & Display MMIO Simulator
+ check_code: kiểm tra mã điều khiển có hợp lệ về độ dài và khớp với một trong các mã đã được quy ước

+ go, stop, turnLeft, turnRight, track, untrack, goBackward: thực thi mã điều khiển

- Các hàm cho Marsbot

Các hàm và chức năng tương ứng của từng hàm như sau:
+ GO, STOP: điều khiển Marsbot bắt đầu chuyển động (GO) hoặc dừng lại (STOP); lưu trạng thái đang chuyển động hay không vào isGoing
+ ROTATE: điều khiển Marsbot quay theo góc lưu ở a_current
+ TRACK, UNTRACK: điều khiển Marsbot bắt đầu để lại vết (TRACK) hoặc dừng để lại vết (UNTRACK); lưu trạng thái đang ghi vết hay không vào isTracking
+ saveHistory: lưu tọa độ x, y và góc hiện tại trước khi Marsbot thực hiện lệnh ROTATE

- Các hàm để xử lý xâu

Các hàm và chức năng tương ứng của từng hàm như sau:
+ strcmp: so sánh xâu ở $s3 với mã điều khiển vừa nhập (current_code), trả về giá trị boolean ở $t0
+ strClear: xóa mã điều khiển vừa nhập (current_code)

4. Mã nguồn:

```
#DuongDucHieu 20225624
#Curiosity Marsbot
#-----------------------------------------------------
#          col 0x1    col 0x2    col 0x4     col 0x8
#
#  row 0x1      0          1          2          3
#             0x11       0x21       0x41       0x81
#
#  row 0x2      4          5          6          7
#             0x12       0x22       0x42       0x82
#
#  row 0x4      8          9          a          b
#             0x14       0x24       0x44       0x84
#
#  row 0x8      c          d          e          f
#             0x18       0x28       0x48       0x88
#
# key value tuong ung tu 0 -> f trong Digital Lab Sim
.eqv KEY_0 0x11
.eqv KEY_1 0x21
.eqv KEY_2 0x41
.eqv KEY_3 0x81
.eqv KEY_4 0x12
.eqv KEY_5 0x22
.eqv KEY_6 0x42
.eqv KEY_7 0x82
.eqv KEY_8 0x14
.eqv KEY_9 0x24
.eqv KEY_a 0x44
.eqv KEY_b 0x84
.eqv KEY_c 0x18
```

```
.eqv KEY_d 0x28
.eqv KEY_e 0x48
.eqv KEY_f 0x88
# eqv for Keyboard
.eqv IN_ADRESS_HEXA_KEYBOARD 0xFFFF0012
.eqv OUT_ADRESS_HEXA_KEYBOARD 0xFFFF0014
.eqv KEY_CODE 0xFFFF0004 # ASCII code from keyboard, 1 byte
.eqv KEY_READY 0xFFFF0000 # = 1 if has a new keycode ?

# Auto clear after lw

# eqv for Mars bot
.eqv HEADING 0xffff8010
.eqv MOVING 0xffff8050
.eqv LEAVETRACK 0xffff8020
.eqv WHEREX 0xffff8030
.eqv WHEREY 0xffff8040
#-------------------------------------------------------------------------
#set up các biến lưu các trạng thái của Mars Bot
.data
x_history: .word 0 : 16 #vị trí x đầu tiên của Marsbot. đặt = 16 để gỡ lỗi dễ hơn
y_history: .word 0 : 16 # Vị trí y đầu tiên của MarsBot
a_history: .word 0 : 16 #góc alpha cũ
l_history: .word 4 # history length (độ dài dấu vết)
a_current: .word 0 # current alpha
isGoing: .word 0
isTracking: .word 0
current_code: .space 8 # input command code

length: .word 0 # input command length
MOVE_CODE: .asciiz "1b4"
STOP_CODE: .asciiz "c68"
TURN_LEFT_CODE: .asciiz "444"
TURN_RIGHT_CODE: .asciiz "666"
TRACK_CODE: .asciiz "dad"
UNTRACK_CODE: .asciiz "cbc"
GOBACKWARD_CODE: .asciiz "999"
INVALID_CODE: .asciiz "Ma khong hop le!\n"
#-------------------------------------------------------------------------
#Khởi tạo và cbi để xử lý các sự kiện từ bàn phím
.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)


setStartHeading: # đánh dấu điểm bắt đầu của MarsBot là góc 0 độ
```

```
lw $t7, l_history # l_history += 4 tang vung
addi $t7, $zero, 4 # lưu các gtri ban đầu x = 0; y = 0; a = 0
sw $t7, l_history
li $t7, 0
sw $t7, a_current # a_current = 0 -> heading up
jal ROTATE
nop
sw $t7, a_history + 4 # lịch sử góc quay a_history[1] = 0
j waitForKey #nhảy đến nhãn dán waitForKey


print_error: # In ra thông báo lỗi
li $v0, 4 #mã hệ thống để in ra 1 chuỗi kí tự
la $a0, INVALID_CODE # Địa chỉ lỗi thông báo cần in ra
syscall


print_current_code: #in ra mã điều khiển vừa nhập vào
li $v0, 4
la $a0, current_code
syscall


resetInput: #xoá mã đk đã nhập để chuẩn bị cho mã tiếp theo
jal strClear
nop


waitForKey: # chờ phím được ấn từ Digital Lab Sim
lw $t5, 0($k1) # $t5 = [$k1] = KEY_READY

beq $t5, $zero, waitForKey # if $t5 == 0 -> quay lại để tiếp tục chờ
nop
beq $t5, $zero, waitForKey


readKey: # Đọc ký tự được nhập vào từ Keyboard & Display MMIO Simulator
lw $t6, 0($k0) # $t6 = [$k0] = KEY_CODE
beq $t6, 0x7f , resetInput # $t6 == 'DEL' -> reset input
beq $t6, 32, replay # $t6 == " " -> Replay, so sánh với ký tự space
nop
beq $t6, 0x0a, check_code # $t6 =='\n'
nop

check_code: #kiểm tra mã điều khiển có hợp lệ về độ dài và khớp với một trong các mã đã được quy ước
lw $s2, length # length != 3 -> invalid cmd
bne $s2, 3, print_error
la $s3, MOVE_CODE #lưu địa chỉ các chuỗi mã điều khiển vào $s3
jal strcmp # sau khi ss tra ve gia tri t0
beq $t0, 1, case_go
la $s3, STOP_CODE
```

```
jal strcmp
beq $t0, 1, case_stop
la $s3, TURN_LEFT_CODE
jal strcmp
beq $t0, 1, case_turnLeft
la $s3, TURN_RIGHT_CODE
jal strcmp
beq $t0, 1, case_turnRight
la $s3, TRACK_CODE
jal strcmp
beq $t0, 1, case_track
la $s3, UNTRACK_CODE
jal strcmp
beq $t0, 1, case_untrack
la $s3, GOBACKWARD_CODE
jal strcmp
beq $t0, 1, goBackward
nop
j print_error


switch:
case_go: j go
case_stop: j stop
case_turnLeft: j turnLeft
case_turnRight: j turnRight
case_track: j track
case_untrack: j untrack
case_goBackWard: j goBackward
default:
#-------------------------------------------------------
go: jal GO
j print_current_code
#-------------------------------------------------------
stop: jal STOP
j print_current_code
#-------------------------------------------------------
track: jal TRACK
j print_current_code
#-------------------------------------------------------
untrack: jal UNTRACK
j print_current_code


#-------------------------------------------------------
turnRight:
lw $t7, isGoing
lw $s0, isTracking
jal STOP
nop
jal UNTRACK
```

```
nop
la $s5, a_current
lw $s6, 0($s5) # $s6 is heading at now
addi $s6, $s6, 90 # increase alpha by 90*
sw $s6, 0($s5) # update a_current
jal saveHistory
jal ROTATE #thực hiện quay phải 90*
beqz $s0, noTrack1
nop
jal TRACK
noTrack1: nop
beqz $t7, noGo1
nop
jal GO
noGo1: nop
j print_current_code


#---------------------------------------------------------
turnLeft:
lw $t7, isGoing
lw $s0, isTracking
jal STOP
nop
jal UNTRACK
nop
la $s5, a_current
lw $s6, 0($s5) # $s6 is heading at now
addi $s6, $s6, -90 # decrease alpha by 90* (quay trái 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 print_current_code


#---------------------------------------------------------
goBackward:
li $t7, IN_ADRESS_HEXA_KEYBOARD # Disable interrupts
sb $zero, 0($t7) #vô hiệu hoá các ngắt = ghi 0 vào bộ nhập liệu
lw $s5, l_history # $s5 = length
jal UNTRACK
jal GO
```

```
goBackward_turn:
addi $s5, $s5, -4 # length--
lw $s6, a_history($s5) # $s6 = a_history[length]
addi $s6, $s6, 180 # $s6 = góc ngược lại với góc alpha (tăng thêm 180*)
sw $s6, a_current
jal ROTATE
nop

goBackward_toTurningPoint:
lw $t9, x_history($s5) # $t9 = x_history[i]
get_x: li $t8, WHEREX # $t8 = x_current
lw $t8, 0($t8)
bne $t8, $t9, get_x # if x_current != x_history[i] lặp lại lấy x_current
nop # -> get y
lw $t9, y_history($s5) # $t9 = y_history[i]
get_Y: li $t8, WHEREY # $t8 = y_current
lw $t8, 0($t8)
bne $t8, $t9, get_Y # if y_current != y_history[i] lặp lại lấy y_current
nop # -> turn or end
beq $s5, 0, goBackward_end # l_history == 0 đã di chuyển về điểm xuất phát
nop # -> end
j goBackward_turn # else -> turn

goBackward_end:
jal STOP
sw $zero, a_current # update heading
jal ROTATE
addi $s5, $zero, 4
sw $s5, l_history # reset l_history = 0
j print_current_code


#-----------------------------------------------------------
# saveHistory()
#-----------------------------------------------------------
saveHistory:  #Sử dụng stack để lưu các giá trị
addi $sp, $sp, 4 # backup: de khong bát thay doi gia tri khi lay ra thuc hien
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_current # s4 = a_current
lw $t3, l_history # $t3 = l_history
sw $s1, x_history($t3) # store: x, y, alpha
sw $s2, y_history($t3)
sw $s4, a_history($t3)
addi $t3, $t3, 4 # update lengthPath
sw $t3, l_history
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) # lưu gtri thanh ghi $at vào vtri 0(đchi) của năng epes
addi $sp, $sp, 4
sw $k0, 0($sp)
li $at, MOVING # MOVING == 1 -> DI CHUYEN
addi $k0, $zero, 1
sb $k0, 0($at) #đối tượng đang di chuyển
li $t7, 1 # thay doi isGoing = 0 -> 1
sw $t7, isGoing
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 # MOVING = 0 -> stop
sb $zero, 0($at)
sw $zero, isGoing # isGoing = 1 -> 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, isTracking
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 tracking
sw $zero, isTracking
lw $at, 0($sp) # restore back up
addi $sp, $sp, -4
UNTRACK_end: jr $ra
#------------------------------------------------------------
# ROTATE()
#------------------------------------------------------------
ROTATE:
addi $sp, $sp, 4 # backup luu quang duong vua di
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_current
lw $t3, 0($t2) # $t3 is heading at now
sw $t3, 0($t1) # Cập nhật hướng đi của Marsbot
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 current_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)
addi $t0, $zero, 0 # $t1 = return value = 0
addi $t1, $zero, 0 # $t1 = i = 0
strcmp_loop:
beq $t1, 3, strcmp_equal # if i = 3 -> end loop -> equal
nop
lb $t2, current_code($t1) # $t2 = current_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, length # $t3 = length
addi $t1, $zero, -1 # $t1 = -1 = i
strClear_loop: addi $t1, $t1, 1 # i++
sb $zero, current_code # current_code[i] = '\0'
bne $t1, $t3, strClear_loop # if $t1 <=3 resetInput loop
nop
sw $zero, length # reset 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
#-------------------------------------------------------
# Replay()
#-------------------------------------------------------
replay:
li $t7, IN_ADRESS_HEXA_KEYBOARD # Disable interrupts
sb $zero, 0($t7)
lw $s5, l_history # $s5 = length
jal UNTRACK
jal GO
replay_turn: addi $s5, $s5, -4 # length--
lw $s6, a_history($s5) # $s6 = a_history[length]
addi $s6, $s6, 180 # $s6 = the reverse direction of alpha
sw $s6, a_current
jal ROTATE

nop
replay_toTurningPoint:
lw $t9, x_history($s5) # $t9 = x_history[i]
get_x1: li $t8, WHEREX # $t8 = x_current
lw $t8, 0($t8)
```

```
bne $t8, $t9, get_x # if x_current == x_history[i]
nop # -> get y
lw $t9, y_history($s5) # $t9 = y_history[i]
get_Y1: li $t8, WHEREY # $t8 = y_current
lw $t8, 0($t8)
bne $t8, $t9, get_Y # if y_current == y_history[i]
nop # -> turn or end
beq $s5, 0, replay_end # l_history == 0
nop # -> end
j replay_turn # else -> turn
replay_end:
jal STOP
sw $zero, a_current # update heading
jal ROTATE
addi $s5, $zero, 4
sw $s5, l_history # reset l_history = 0
j print_current_code
#==============================================================================
# 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
#---------------------------------------------------
```

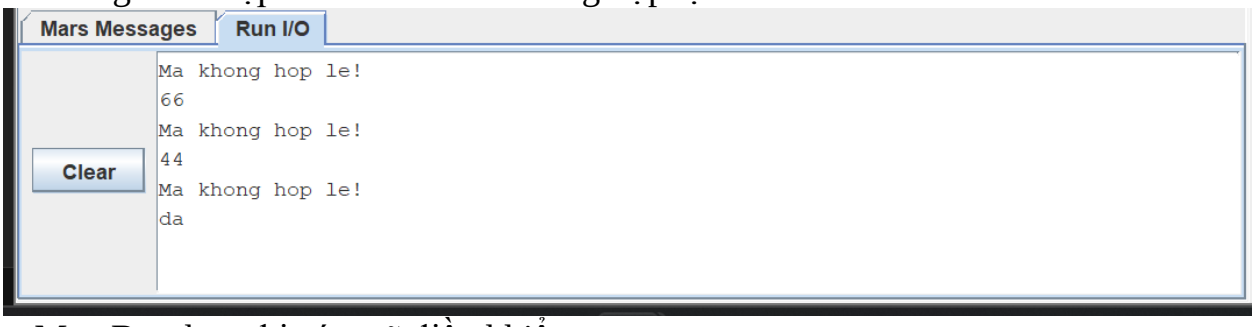
```
get_cod: li $t1, IN_ADRESS_HEXA_KEYBOARD
li $t2, OUT_ADRESS_HEXA_KEYBOARD
scan_row1: li $t3, 0x81
sb $t3, 0($t1)
lbu $a0, 0($t2)
bnez $a0, get_code_in_char
scan_row2: li $t3, 0x82
sb $t3, 0($t1)
lbu $a0, 0($t2)
bnez $a0, get_code_in_char
scan_row3: li $t3, 0x84
sb $t3, 0($t1)
lbu $a0, 0($t2)
bnez $a0, get_code_in_char
scan_row4: li $t3, 0x88
sb $t3, 0($t1)
lbu $a0, 0($t2)
bnez $a0, get_code_in_char
get_code_in_char:
beq $a0, KEY_0, case_0
beq $a0, KEY_1, case_1
beq $a0, KEY_2, case_2
beq $a0, KEY_3, case_3
beq $a0, KEY_4, case_4
beq $a0, KEY_5, case_5
beq $a0, KEY_6, case_6
beq $a0, KEY_7, case_7
beq $a0, KEY_8, case_8
beq $a0, KEY_9, case_9
beq $a0, KEY_a, case_a
beq $a0, KEY_b, case_b
beq $a0, KEY_c, case_c
beq $a0, KEY_d, case_d
beq $a0, KEY_e, case_e
beq $a0, KEY_f, case_f
case_0: li $s0, '0' # $s0 store code in char type
j store_code
case_1: li $s0, '1'
j store_code
case_2: li $s0, '2'

j store_code
case_3: li $s0, '3'
j store_code
case_4: li $s0, '4'
j store_code
case_5: li $s0, '5'
j store_code
case_6: li $s0, '6'
j store_code
case_7: li $s0, '7'
```

```
j store_code
case_8: li $s0, '8'
j store_code
case_9: li $s0, '9'
j store_code
case_a: li $s0, 'a'
j store_code
case_b: li $s0, 'b'
j store_code
case_c: li $s0, 'c'
j store_code
case_d: li $s0, 'd'
j store_code
case_e: li $s0, 'e'
j store_code
case_f: li $s0, 'f'
j store_code
store_code: la $s1, current_code
la $s2, length
lw $s3, 0($s2) # $s3 = strlen(current_code)
addi $t4, $t4, -1 # $t4 = i
store_code_loop: addi $t4, $t4, 1
bne $t4, $s3, store_code_loop
add $s1, $s1, $t4 # $s1 = current_code + i
sb $s0, 0($s1) # current_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 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)
```

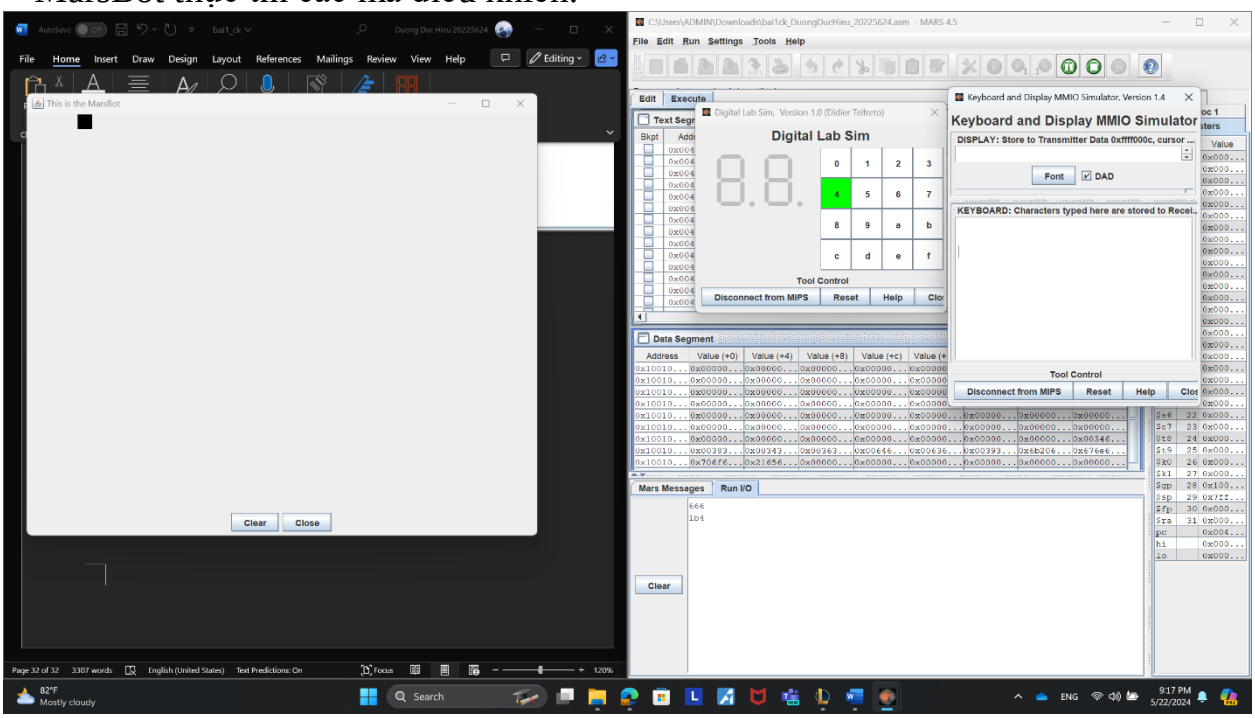
```
addi $sp, $sp, -4
lw $at, 0($sp)
addi $sp, $sp, -4
lw $a0, 0($sp)
addi $sp, $sp, -4
lw $t3, 0($sp)
addi $sp, $sp, -4
lw $t2, 0($sp)
addi $sp, $sp, -4
lw $t1, 0($sp)
addi $sp, $sp, -4
lw $ra, 0($sp)
addi $sp, $sp, -4
return: eret # Return from exception
```

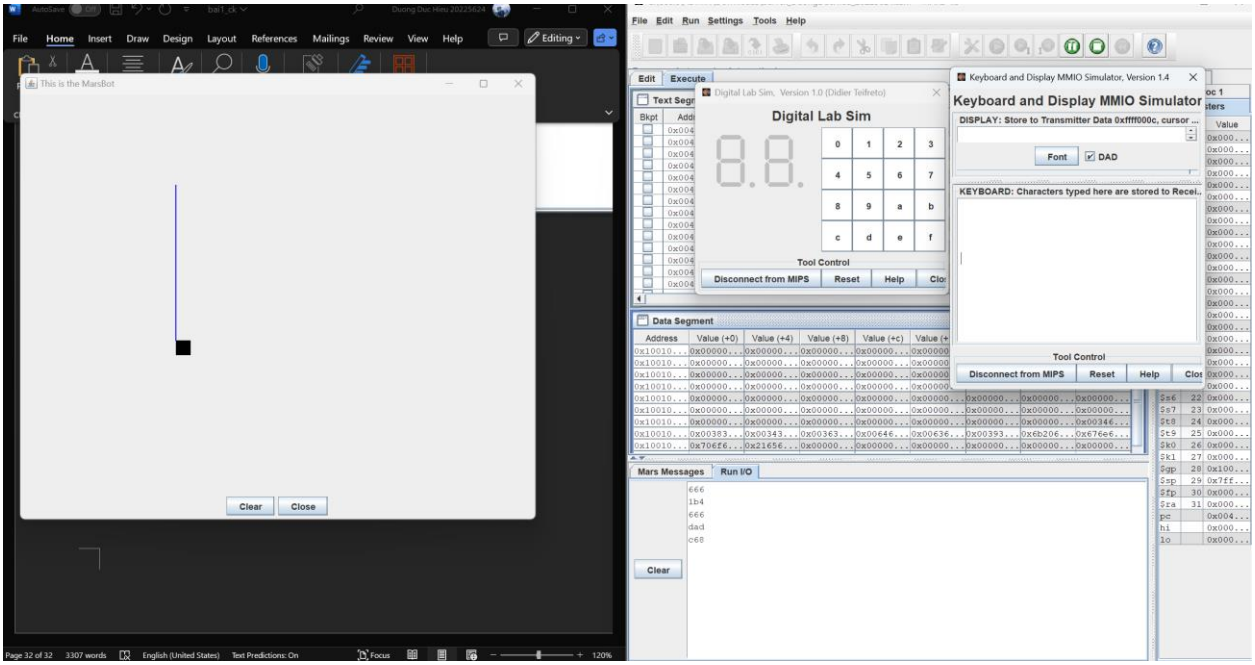
5. Mô phỏng chương trình:

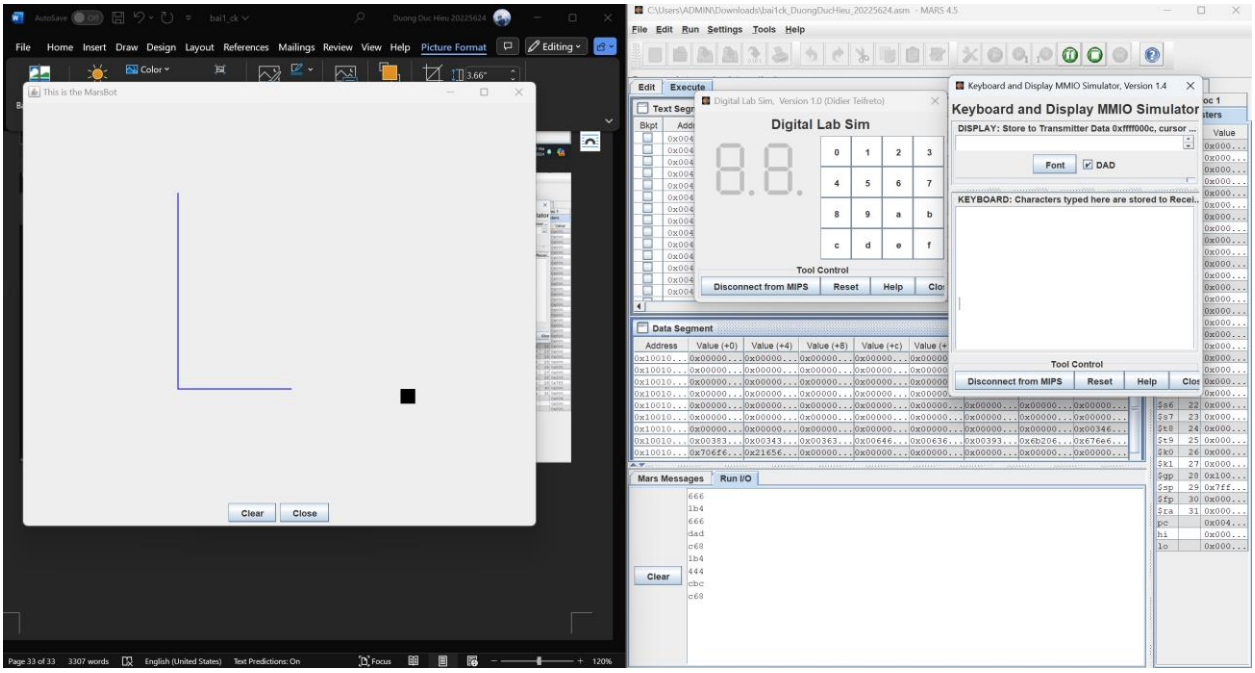
+ Thông báo nhập mã điều khiển không hợp lệ:

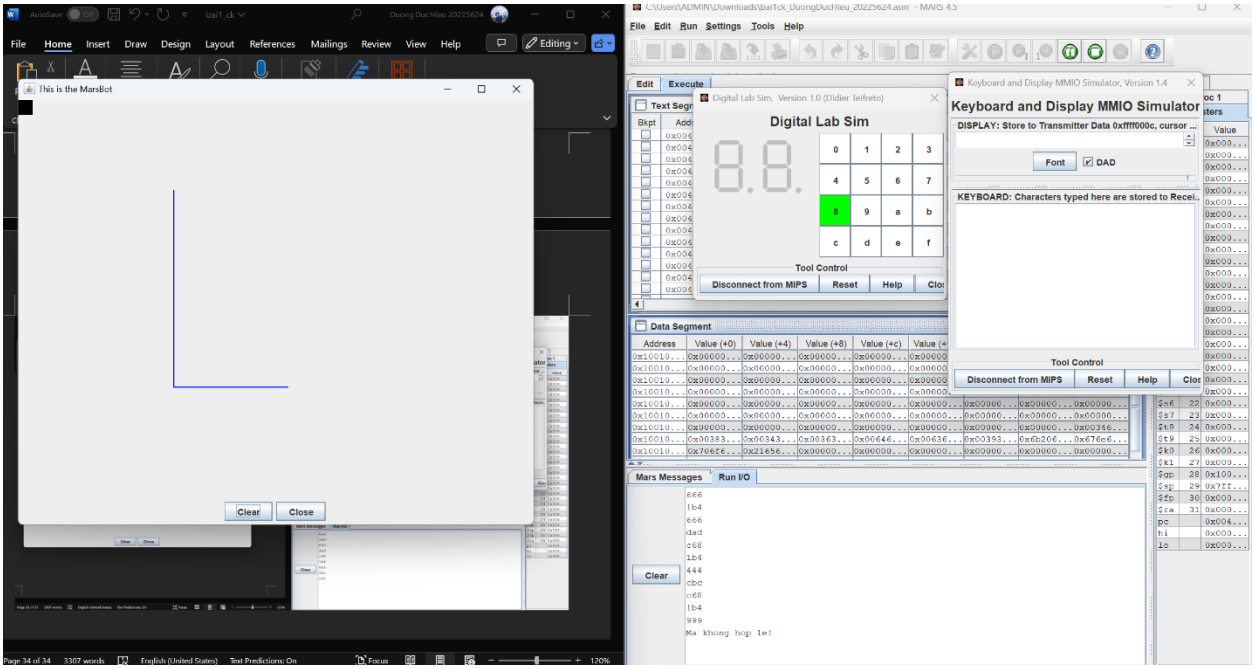


+ MarsBot thực thi các mã điều khiển:




This is the MarsBot
Clear
Close
Digital Lab Sim
Keyboard and Display MMIO Simulator
Tool Control
Disconnect from MIPS
Reset
Help
Data Segment
Mars Messages
Run I/O
Clear


This is the MarsBot
Clear
Close
Digital Lab Sim
Keyboard and Display MMIO Simulator
Tool Control
Disconnect from MIPS
Reset
Help
Data Segment
Mars Messages
Run I/O
Clear


This is the MarsBot
Clear
Close
Digital Lab Sim
Keyboard and Display MMIO Simulator
Tool Control
Disconnect from MIPS
Reset
Help
Data Segment
Mars Messages
Run I/O
Clear
Ma khong hop le!

Chủ đề 2: Vẽ hình trên Bitmap

NGUYỄN ĐỨC LONG – 20225876

I) Yêu cầu bài toán

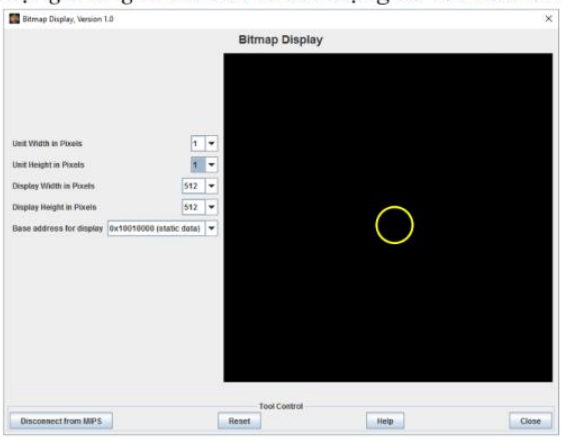
Viết một chương trình sử dụng MIPS để vẽ một quả bóng di chuyển trên màn hình mô phỏng Bitmap của Mars). Nếu đối tượng đập vào cạnh của màn hình thì sẽ di chuyển theo chiều ngược lại.

Yêu cầu:

- Thiết lập màn hình ở kích thước 512x512. Kích thước pixel 1x1.
- Quả bóng là một đường tròn.

Chiều di chuyển phụ thuộc vào phím người dùng bấm, gồm có (di chuyển lên (W), di chuyển xuống (S), Sang trái (A), Sang phải (D) trong bộ giả lập Keyboard and Display MMIO Simulator). Vị trí bóng ban đầu ở giữa màn hình. Tốc độ bóng di chuyển là có thay đổi không đổi. Khi người dùng giữ một phím nào đó (W, S, A, D) thì quả bóng sẽ tăng tốc theo hướng đó với gia tốc tuỳ chọn.

Gợi ý: *Để làm một đối tượng di chuyển thì chúng ta sẽ xóa đối tượng ở vị trí cũ và vẽ đối tượng ở vị trí mới. Để xóa đối tượng chúng ta chỉ cần vẽ đối tượng đó với màu là màu nền.*



II) Tổng quát quá trình

B1: Set up các giá trị cơ bản: tâm đường tròn, bán kính, khoảng di chuyển dx dy (1px) tương ứng hướng di chuyển, $a0 chứa thời gian sleep.

B2: Tính toán ra các điểm để tạo đường tròn (tính toán vị trí điểm với hệ quy chiếu là so với tâm hình tròn), rồi lưu tất cả các điểm này vào 1 mảng.

B3: Đọc từ input và kiểm tra xem có đổi hướng /hay tăng tốc độ không. Nếu không có thì tức là vẫn di chuyển như cũ, tiến hành việc kiểm tra chạm cạnh.

B4: Kiểm tra vòng tròn đã chạm cạnh màn hình chưa. Nếu chạm cạnh màn hình thì phải thay đổi hướng di chuyển(và vẽ hình tròn ở vị trí mới). Nếu không thì vẫn là di chuyển theo hướng cũ(và vẽ hình tròn ở vị trí mới)

B5: Xóa hình tròn cũ (Dùng màu đen, tô lên các data cũ(hiện tại) của hình tròn), sau đó cập nhật data mới của hình tròn, dùng màu vàng, tô lên các vị trí mới này.

B6: Lặp quá trình từ bước 3.

III) Chi tiết các quá trình

Phần 1: Set up

```
setup:
    li $s0, 255        # x = 255
    li $s1, 255        # y = 255
    li $s2, 0# dx = 0
    li $s3, 0# dy = 0
    li $s4, 20         # r = 20
    li $a0, 40         # t = 40ms
    jal      get_circle_data
```

- Màn hình có kích thước 512x512 nên ta cho tâm hình tròn (x,y) ở vị trí (255,255) lưu ở thanh ghi $s0 và $s1.
- $s2 và $s3 lưu hướng di chuyển dx dy.
  +) dx = 1 là sang phải, dx = -1 là sang trái
  +) dy = 1 là xuống, dy = -1 là lên
- $s4 lưu bán kính r
- $a0 lưu thời gian sleep (ngủ ít -> chạy nhanh)

Phần 2: Tính toán vị trí tương đối các điểm để tạo nên hình tròn

-Tạo cặp (px, py) thỏa mãn thuộc đường tròn bằng cách:

+ Chọn px nguyên bắt đầu từ 0. Tính (py)^2=r^2- (px)^2
+ Chọn số nguyên z có bình phương gần (py)^2 nhất -> Chọn z là py
+ Được cặp giá trị (px, z) chính là (px, py)

-Vì px, py dương (góc phần tư thứ nhất) nên ta sẽ tiến hành đảo để tạo các cặp (-px, py), (-px, -py), (px, -py) để tạo ra các điểm đối xứng

-Swap px cho py dể đối xứng trong góc phần 8 (Vì khi nãy thuật toán tính y không có sự chính xác cao nên không có sự cân đối giữa x và y). Vậy 1 lần tìm ra cặp (px, py) sẽ có 8 điểm khác sinh ra.

-Chạy giá trị x từ 0 -> r = 20 để tìm tiếp các điểm. Sau khi xong thì lưu hết dữ liệu vào mảng Circle_points

Phần 3: Kiểm tra di chuyển, tăng tốc độ:

- Mặc định của tín hiệu là giữ ở giữa màn hình (dx = dy = 0).
- Khi tiếp nhận phím điều hướng di chuyển/ tốc độ, giá trị của dx và dy (hoặc thời gian ngủ) sẽ được điều chỉnh để sẵn sàng cho việc di chuyển từ trạng thái cũ sang mới.

+) w: lên: dx = 0 , dy = -1
+) s: xuống: dx = 0, dy = 1
+) a: trái: dx = -1, dy = 0
+) d: phải: dx = 1, dy = 0.

Tăng tốc: ví dụ khi đang di chuyển lên, mà ta nhấn 'w' thì sẽ tăng tốc quả bóng bằng cách làm giảm thời gian sleep

Phần 4: Kiểm tra chạm cạnh

Kiểm tra bằng cách tính tọa độ tâm (x, y) cộng thêm bán kính đã chạm tới rìa màn hình. Nếu chạm đến thì tiến hành đổi ngược lại dx, dy tùy theo hướng để điều chỉnh lại trạng thái di chuyển. Ví dụ, trong 4 hàm check trái phải trên dưới, ta kiểm tra thấy đang có hướng di chuyển sang phải(dx = 1, dy = 0) thì ta nhảy đến hàm check right. Lấy vị trí tâm (x,y) cộng với bán kính R xem có chạm cạnh chưa (y + r = 511 ?) không chạm thì tiếp tục quá trình, có chạm thì cài lại hướng dx dy, rồi tiếp tục quá trình.

```
check_right:
    add     $t0, $s0, $s4     # Set $t0 to the right point of the circle
    beq     $t0, 511, reverse_direction       # Reverse direction if point hits edge
    j       move_circle     # Return otherwise

check_left:
    sub     $t0, $s0, $s4     # Set $t0 to the left point of the circle
    beq     $t0, 0, reverse_direction# Reverse direction if point hits edge
    j       move_circle     # Return otherwise
check_down:
    add     $t0, $s1, $s4     # Set $t0 to the below point of the circle
    beq     $t0, 511, reverse_direction       # Reverse direction if point hits edge
    j       move_circle     # Return otherwise

check_up:
    sub     $t0, $s1, $s4     # Set $t0 to the upper point of the circle
    beq     $t0, 0, reverse_direction# Reverse direction if point hits edge
    j       move_circle     # Return otherwise

reverse_direction:
    sub     $s2, $0, $s2      # dx = -dx
    sub     $s3, $0, $s3      # dy = -dy
    j       move_circle
```

Phần 5: Xóa hình cũ, vẽ hình tròn mới

-Xóa hình tròn từ lần cũ bằng cách đổi màu sang màu đen, vẽ hình tròn với dữ liệu hiện có (dữ liệu cũ) , đổi sang màu vàng, cập nhật dữ liệu rồi vẽ thêm 1 lần

- Cách vẽ hình tròn mới:

+ Tính toán giá trị x, y sau khi cộng thêm (dx, dy) theo hướng phù hợp.
+ Cộng thêm (px, py) từ mảng hình tròn để được những điểm tạo nên hình tròn xung quanh tâm (x, y)
+Xác định vị trí trên bit maps và tiến hành vẽ.
+Quá trình vẽ dừng lại khi mảng circle_points chạm đến phần tử cuối(tức là đã vẽ hết tất cả các điểm)

```
draw_circle:
    addi    $sp, $sp, -4     # Save $ra
    sw      $ra, 0($sp)
    la      $s6, array_end
    lw      $s7, 0($s6)      # $s7 becomes the end address of the "circle" array
    la      $s6, circle_points      # $s6 becomes the pointer to the "circle" array

draw_loop:
    beq     $s6, $s7, draw_end      # Stop when reach to the end of array
    lw      $a1, 0($s6)             # Get px
    lw      $a2, 4($s6)             # Get py
    jal     point_draw
    addi    $s6, $s6, 8             # Get to the next point
    j       draw_loop

draw_end:
    lw      $ra, 0($sp)
    addi    $sp, $sp, 4
    jr      $ra

point_draw:
    li      $t0, SCREEN_MONITOR
    add     $t1, $s0, $a1           # x_point = x + px
    add     $t2, $s1, $a2           # y_point = y + py
    sll     $t2, $t2, 9             # $t2 = y_point * 512
    add     $t2, $t2, $t1           # $t2 += x_point
    sll     $t2, $t2, 2             # $t2 *= 4
    add     $t0, $t0, $t2           #point (pixel_position) on screen monitor
    sw      $s5, 0($t0)             #draw yellow in this pixel
    jr      $ra
```

MÃ NGUỒN

```
.eqv KEY_CODE 0xFFFF0004
.eqv KEY_READY 0xFFFF0000
.eqv SCREEN_MONITOR 0x10010000
.data
array_end: .word   1          # The end of the "circle_points" array
circle_points:                .word              # Array saves all points position of circle
.text
setup:
    li $s0, 255       # x = 255
    li $s1, 255       # y = 255
    li $s2, 1         # dx = 1
    li $s3, 0         # dy = 0
    li $s4, 20        # r = 20
    li $a0, 40        # t = 40ms/frame
    jal       get_circle_data

input:
    li        $k0, KEY_READY    # Check whether there is input data
    lw        $t0, 0($k0)
    bne       $t0, 1, edge_check
    jal       direction_change

edge_check:
right:
    bne       $s2, 1, left
    j         check_right

left:
    bne       $s2, -1, down
    j         check_left

down:
    bne       $s3, 1, up
    j         check_down

up:
    bne       $s3, -1, move_circle
    j         check_up

move_circle:
    add       $s5, $0, $0       # Set color to black
    jal       draw_circle       # Erase the old circle

    add       $s0, $s0, $s2     # Set x and y to the coordinates of the center of the new circle
    add       $s1, $s1, $s3
    li        $s5, 0x00FFFF00         # Set color to yellow
    jal       draw_circle       # Draw the new circle
```

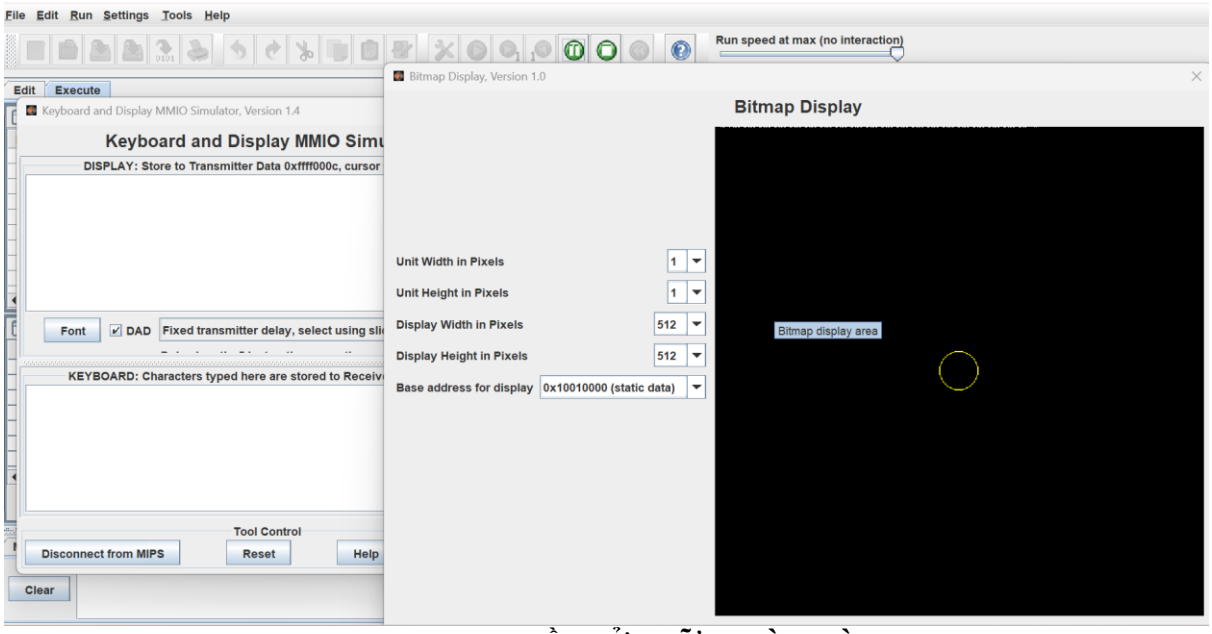
```
    loop:
        li $v0, 32              # Syscall value for sleep
        syscall
        j       input           # Renew the cycle

    # Procedure below

    get_circle_data:
        addi    $sp, $sp, -4    # Save $ra
        sw      $ra, 0($sp)
        la      $s5, circle_points      # $s5 becomes the pointer of the "circle" array
        mul     $a3, $s4, $s4   # $a3 = r^2
        add     $s7, $0, $0     # pixel x (px) = 0

    point_of_circle:
        bgt     $s7, $s4, data_end
        mul     $t0, $s7, $s7   # $t0 = px^2
        sub     $a2, $a3, $t0   # $a2 = r^2 - px^2 = py^2
        jal     square_root             # $a2 = py
        add     $a1, $0, $s7    # $a1 = px
        add     $s6, $0, $0     # After saving (px, py), (-px, py), (-px, -py), (px, -py), we swap px and py,
then save (-py, px), (py, px), (py, -px), (-py, -px)

    doiXung:
        beq     $s6, 2, finish
        jal     point_save      # px >= 0 , py >= 0
        sub     $a1, $0, $a1
        jal     point_save      # px <= 0, py >= 0
        sub     $a2, $0, $a2
        jal     point_save      # px <= 0, py <= 0
        sub     $a1, $0, $a1
        jal     point_save      # px >= 0, py <= 0

        add     $t0, $0, $a1    # Swap px and -py
        add     $a1, $0, $a2
        add     $a2, $0, $t0

        addi    $s6, $s6, 1
        j       doiXung

    finish:
        addi    $s7, $s7, 1
        j       point_of_circle

    data_end:
        la      $t0, array_end
        sw      $s5, 0($t0)     # Save the end address of the "circle_points" array
        lw      $ra, 0($sp)
        addi    $sp, $sp, 4
```

```
    jr      $ra

square_root:        # Find the square root of $a2
    add $t9, $a2, $0        #$t9 = $a2
    mtc1.d $a2, $f2             # move $a2 to $f2
    cvt.d.w $f2, $f2        #convert $f2 from word to double
    sqrt.d $f2, $f2         #sqrt $f2
    cvt.w.d $f2, $f2        #convert $f2 from double to word
    mfc1.d $t2, $f2             #move to $t2
    add $a2, $t9, $0        #$a2 = $t9
    mul $t5, $t2, $t2       #$t2 = $t2 ^ 2
    addi $t3, $t2, 1        #$t3 = $t2 + 1
    mul $t6, $t3, $t3       #$t3 = $t3 ^ 2
    sub $t8, $a2, $t5       #$t8 = py^2 - $t2
    sub $t9, $t6, $a2       #$t9 = $t3 - py^2
compare:
    blt     $t8, $t9, set_closest   # if $t8 < $t9, $t2 is nearer to square root of $a2
    add     $a2, $0, $t3            # Else $t3 is the nearear number to square root of $a2
    jr      $ra
set_closest:
    add $a2, $0, $t2
    jr $ra

point_save:
    sw      $a1, 0($s5)     # Store px in the "circle_points" array
    sw      $a2, 4($s5)     # Store py in the "circle_points" array
    addi    $s5, $s5, 8     # Move the pointer to next block
    jr      $ra

direction_change:
    li      $k0, KEY_CODE
    lw      $t0, 0($k0)

char_D:
    bne     $t0, 'd', char_A
    bgtz    $s2, speed_up
    li $s2, 1       # dx = 1
    li $s3, 0       # dy = 0
    li      $a0, 50
    jr      $ra

char_A:
    bne     $t0, 'a', char_S
    bltz    $s2, speed_up
    li $s2, -1      # dx = -1
    li $s3, 0       # dy = 0
    li      $a0, 50
    jr      $ra
```

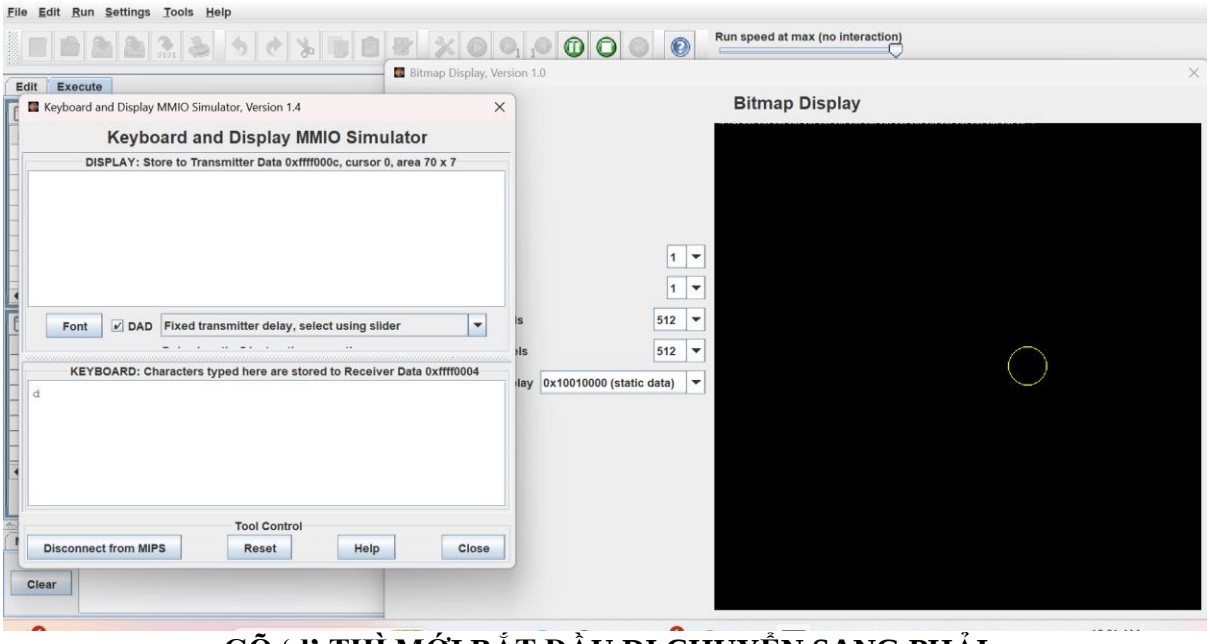
```
char_S:
    bne     $t0, 's', char_W
    bgtz $s3, speed_up
    li $s2, 0          # dx = 0
    li $s3, 1          # dy = 1
    li      $a0, 50
    jr      $ra

char_W:
    bne     $t0, 'w', default
    bltz $s3, speed_up
    li $s2, 0          # dx = 0
    li $s3, -1         # dy = -1
    li      $a0, 50
    jr      $ra

speed_up:
    addi    $a0, $a0, -5
    jr      $ra

default:
    jr      $ra

check_right:
    add     $t0, $s0, $s4   # Set $t0 to the right point of the circle
    beq     $t0, 511, reverse_direction     # Reverse direction if point hits edge
    j       move_circle    # Return otherwise

check_left:
    sub     $t0, $s0, $s4   # Set $t0 to the left point of the circle
    beq     $t0, 0, reverse_direction       # Reverse direction if point hits edge
    j       move_circle    # Return otherwise
check_down:
    add     $t0, $s1, $s4   # Set $t0 to the below point of the circle
    beq     $t0, 511, reverse_direction     # Reverse direction if point hits edge
    j       move_circle    # Return otherwise

check_up:
    sub     $t0, $s1, $s4   # Set $t0 to the upper point of the circle
    beq     $t0, 0, reverse_direction       # Reverse direction if point hits edge
    j       move_circle    # Return otherwise

reverse_direction:
    sub     $s2, $0, $s2    # dx = -dx
    sub     $s3, $0, $s3    # dy = -dy
    j       move_circle

draw_circle:
    addi    $sp, $sp, -4    # Save $ra
```

```
    sw      $ra, 0($sp)
    la      $s6, array_end
    lw      $s7, 0($s6)     # $s7 becomes the end address of the "circle" array
    la      $s6, circle_points      # $s6 becomes the pointer to the "circle" array

draw_loop:
    beq     $s6, $s7, draw_end      # Stop when reach to the end of array
    lw      $a1, 0($s6)             # Get px
    lw      $a2, 4($s6)             # Get py
    jal     point_draw
    addi    $s6, $s6, 8             # Get to the next point
    j       draw_loop

draw_end:
    lw      $ra, 0($sp)
    addi    $sp, $sp, 4
    jr      $ra

point_draw:
    li      $t0, SCREEN_MONITOR
    add     $t1, $s0, $a1           # x_point = x + px
    add     $t2, $s1, $a2           # y_point = y + py
    sll     $t2, $t2, 9             # $t2 = y_point * 512
    add     $t2, $t2, $t1           # $t2 += x_point
    sll     $t2, $t2, 2             # $t2 *= 4
    add     $t0, $t0, $t2           #point (pixel_position) on screen monitor
    sw      $s5, 0($t0)             #draw yellow in this pixel
    jr      $ra
```

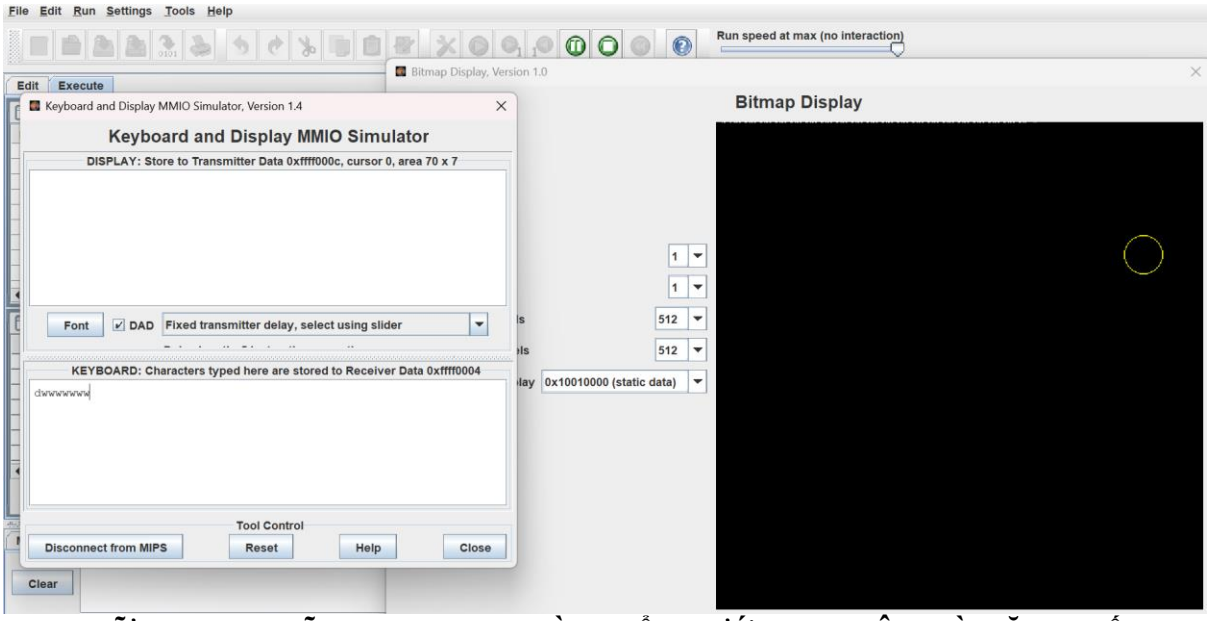
IV) CHẠY DEMO CHƯƠNG TRÌNH



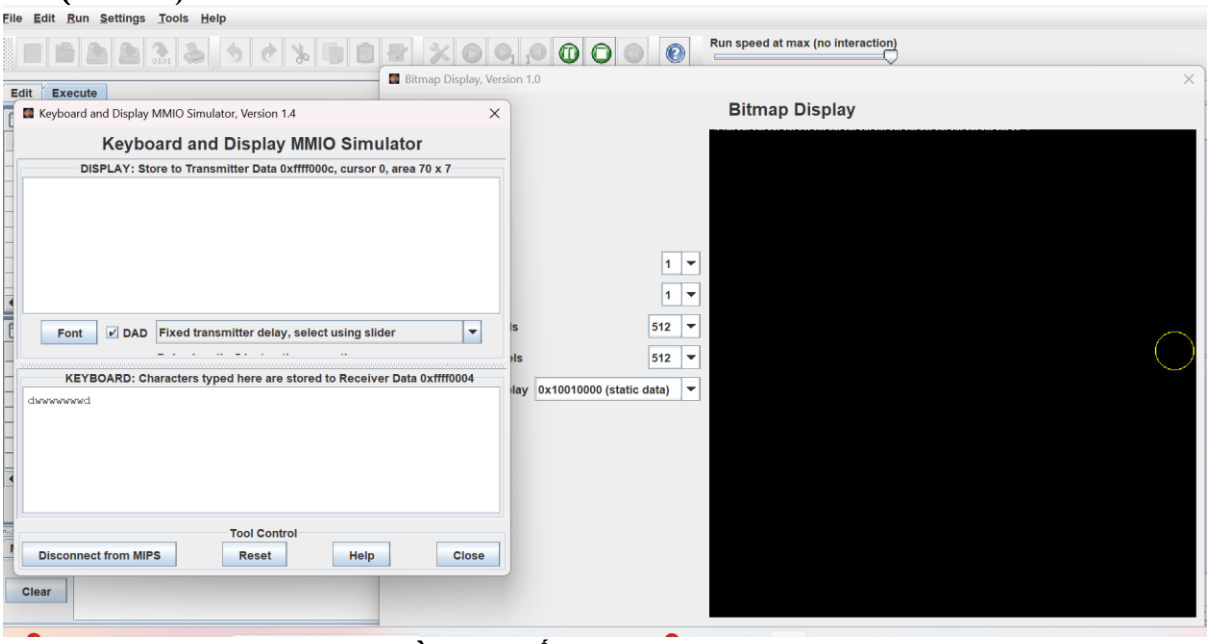
BAN ĐẦU, Ở GIỮA MÀN HÌNH



GÕ ‘d’ THÌ MỚI BẮT ĐẦU DI CHUYỂN SANG PHẢI



(GIỮ W) HAY GÕ ‘WWWWW’ LÀM ĐỔI HƯỚNG ĐI LÊN VÀ TĂNG TỐC



TÌNH HUỐNG CHẠM CẠNH