ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CNTT & TT

***

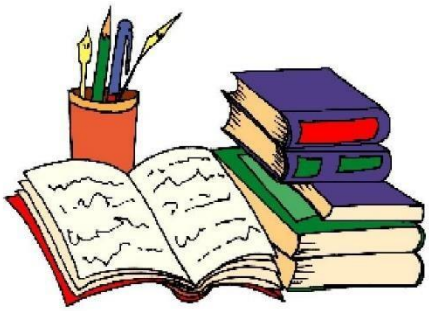
BÁO CÁO BÀI TẬP LỚN

Học phần : THỰC HÀNH KIẾN TRÚC MÁY TÍNH
Mã lớp học : 147789
Nhóm : 11
Giảng viên hướng dẫn : Lê Bá Vui
Trợ giảng : Đỗ Gia Huy

Nhóm sinh viên thực hiện:

Phùng Duy Nghĩa – 20225896 – Bài 1

Trần Doãn Huy – 20225856 – Bài 2



Nội dung

Bài 1. Điều khiển Curiosity Marsbot

1. Yêu cầu bài toán

- Điều khiển Curiosity Marsbot bằng các mã điều khiển từ một bàn phím ma trận

Mã điều khiển	**Ý nghĩa**
1b4	Marsbot bắt đầu chuyển động
c68	Marsbot đứng im
444	Rẽ trái 90 độ so với phương chuyển động gần nhất
666	Rẽ phải 90 độ so với phương chuyển động gần nhất
dad	Bắt đầu để lại vết trên đường
cbc	Chấm dứt để lại vết trên đường
999	Tự động đi theo lộ trình ngược lại. Không vẽ vết, không nhận mã khác cho tới khi kết thúc lộ trình ngược. Mô tả: Marsbot được lập trình để nhớ lại toàn bộ lịch sử các mã điều khiển và khoảng thời gian giữa các lần đổi mã. Vì vậy, nó có thể đảo ngược lại lộ trình để quay về điểm xuất phát.

- Sau khi nhận mã điều khiển, Curiosity Marsbot sẽ không xử lý ngay, mà phải đợi lệnh kích hoạt mã từ bàn phím Keyboard và Display MMIO Simulator. Có 3 lệnh như vậy:

Kích hoạt mã	**Ý nghĩa**
Phím Enter	Kết thúc nhập mã và yêu cầu Marsbot thực thi
Phím Delete	Xóa toàn bộ mã điều khiển đang nhập
Phím Space	Lặp lại lệnh đã thực hiện trước đó

- Tính năng bổ sung: mỗi khi gửi một mã điều khiển cho Marsbot, hiển thị mã đó lên màn hình console để người xem có thể giám sát lộ trình của xe

2. Hướng giải quyết

2.1.Các bước thực hiện

- Mỗi lần bấm các kí tự trên Digital Lab Sim để tạo mã điều khiển cần lưu vào một chuỗi để kiểm tra

- Sau đó bấm các phím kích hoạt để chạy
  - → Khi bấm phím Enter: kiểm tra mã điều khiển có hợp lệ: độ dài mã có bằng 3 ? Có phải là 1 trong các mã điều khiển đề bài yêu cầu không ?
  - → Khi bấm phím Delete: Xóa toàn bộ mã điều khiển đang nhập (tức xóa toàn bộ các kí tự trong lần nhập đó ở chuỗi)
  - → Khi bấm phím Space: lấy giá trị của mã đã nhập trước đó
- Chạy mã điều khiển đã nhập nếu mã điều khiển phù hợp
- Nếu mã điều khiển nhập vào là “444” (tức rẽ trái) hoặc “666” (tức rẽ phải) thì cần lưu lại tọa độ x,y, góc hay hướng di chuyển vào 3 bảng : x_history, y_history, a_history
- Sau đó in mã điều khiển ra màn hình console, và lặp lại quy trình nếu có sự thay đổi khi điều khiển Marsbot

2.2.Các chương trình con (các nhãn, các hàm thực hiện những công việc nhỏ cụ thể)

2.2.1.Xử lý chuỗi

- strcmp: để so sánh mã nhập vào có trùng khớp với 1 trong các mã điều khiển theo yêu cầu đề bài không
- strClear: xóa toàn bộ mã nhập hiện tại
- strCpyToPrev: nội dung của chuỗi hiện tại được sao chép vào chuỗi trước đó
- strCpyPrevToCur: nội dung của chuỗi trước đó được sao chép vào chuỗi hiện tại

2.2.2.Các hàm điều khiển Marsbot

- GO, STOP: điều khiển Marsbot di chuyển hay dừng lại, lưu trạng thái vào isGoing
- ROTATE: điều khiển Marsbot xoay theo góc được lưu trữ ở a_current
- TRACK, UN_TRACK: điều khiển Marsbot có để lại vết đường đi hay không, lưu trạng thái vào isTracking
- saveHistory: lưu tọa đọ x, y và góc hiện tại trước kho xoay hướng khác

2.2.3.Trong phần main-điều khiển chính chương trình

- setStartHeading: thiết lập góc ban đầu cho Marsbot
- printErrorMsq: in ra lỗi nhập
- printCmd: in ra mã điều khiển trên console

- resetInput: Xóa mã điều khiển đang nhận hiện tại
- repeatInput: Lặp lại mã điều khiển đó
- readKey: đọc ký tự được nhấn từ bàn phím
- checkCmd: kiểm tra mã điều khiển có hợp lệ
- go, stop, turnLeft, turnRight, track, untrack, goBackward: thực thi các mã khiển khiển khi chúng hợp lệ

3. Mã nguồn

```
#khoi tao cac phim bam
.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 IN_ADDRESS_HEXA_KEYBOARD 0xFFFF0012
.eqv OUT_ADDRESS_HEXA_KEYBOARD 0xFFFF0014
.eqv KEY_CODE 0xFFFF0004
.eqv KEY_READY 0xFFFF0000

#khoi tao chuc nang Marsbot
.eqv HEADING 0xffff8010
.eqv MOVING 0xffff8050
.eqv LEAVETRACK 0xffff8020
.eqv WHEREX 0xffff8030
.eqv WHEREY 0xffff8040

.data
        x_history:      .word 0:16
        y_history:      .word 0:16
        a_history:      .word 0:16
        l_history:      .word 4
        a_current:      .word 0         #alpha hien tai
```

```
        isGoing:        .word 0
        isTracking:     .word 0

        cmdCode:    .space 8        #ma dau vao
        cmdLen:             .word 0                 #do dai ma dau vao
        prev_cmdCode:       .space 8        #doan ma truoc do

#chuc nang cua marsbot
        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"

#thong bao
        invalidCmd_msg:.asciiz "ma nhap khong hop le\n"

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

        #ngat
        li $t1, IN_ADDRESS_HEXA_KEYBOARD
        li $t3, 0x80
        sb $t3, 0($t1)

#thiet lap goc ban dau cho marsbot
setStartHeading:
        lw $t7, l_history
        addi $t7, $zero, 4
        sw $t7, l_history

        li $t7, 90
        sw $t7, a_current
        jal ROTATE
        nop

        sw $t7, a_history # a_history[1] = 90

        j waitForKey

printErrorMsg:
        li $v0, 4
        la $a0, invalidCmd_msg
        syscall
```

```
printCmd:
        li $v0, 4
        la $a0, cmdCode
        syscall
        j resetInput

repeatCmd:    #sao chep prev_cmdCode
        jal strCpyPrevToCur
        j checkCmd

resetInput:
        jal strClear
        nop

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

readKey:
        lw $t6, 0($k0)  #$t6 = [$k0] = KEY_CODE
        beq $t6, 0x7f, resetInput       #neu $t6 =='DEL' (ma asciiz = 127) ->reset input
        beq $t6, 0x20, repeatCmd      #neu $t6 =='Space' (ma asciiz = 32) ->rêpat input

        bne $t6, 0x0a, waitForKey     #neu $t6 != '\n' ->polling : do chua an dung phim Enter
        nop
        bne $t6, 0x0a, waitForKey

checkCmd:
        lw $s2, cmdLen                  # cmdLen != 3 -> invalid cmd
        bne $s2, 3, printErrorMsg

        la $s3, MOVE_CODE
        jal strcmp
        beq $t0, 1, go

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

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

        la $s3, TURN_RIGHT_CODE
        jal strcmp
```

```
        beq $t0, 1, turnRight

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

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

        la $s3, GOBACKWARD_CODE
        jal strcmp
        beq $t0, 1, goBackward

        nop
        j printErrorMsg

#cac chuc nang
go:
        jal strCpyCurToPrev
        jal GO
        j printCmd

stop:
        jal strCpyCurToPrev
        jal STOP
        j printCmd

track:
        jal strCpyCurToPrev
        jal TRACK
        j printCmd

untrack:
        jal strCpyCurToPrev
        jal UNTRACK
        j printCmd

turnRight:
        jal strCpyCurToPrev
        lw $t7, isGoing
        lw $s0, isTracking

        jal STOP
        nop
        jal UNTRACK
        nop
```

```
        la $s5, a_current
        lw $s6, 0($s5) # $s6 huong hien tai
        addi $s6, $s6, 90 # tang aplpha 90
        sw $s6, 0($s5) # cap nhat a_current

        jal saveHistory
        jal ROTATE

        beqz $s0, noTrack1
        nop
        jal TRACK

noTrack1:
        nop
        beqz $t7, noGo1
        nop
        jal GO

noGo1:
        nop
        j printCmd

turnLeft:
        jal strCpyCurToPrev
        lw $t7, isGoing
        lw $s0, isTracking

        jal STOP
        nop
        jal UNTRACK
        nop

        la $s5, a_current
        lw $s6, 0($s5) #$s6 la huong hien tai
        addi $s6, $s6, -90      # giam alpha 90 do
        sw $s6, 0($s5)# cap nhat a_current

        jal saveHistory
        jal ROTATE

        beqz $s0, noTrack2 # neu khong theo doi -> bo qua
        nop
        jal TRACK

noTrack2:
        nop
        beqz $t7, noGo2 # neu khong di -> bo qua
        nop
```

```
        jal GO

noGo2:
        nop
        j printCmd

goBackward:
        jal strCpyCurToPrev
        li $t7, IN_ADDRESS_HEXA_KEYBOARD # vo hieu hoa ngat khi quay lui
        sb $zero, 0($t7)

        lw $s5, l_history # $s5 = cmdLen
        jal UNTRACK
        jal GO

goBackward_turn:
        addi $s5, $s5, -4 # cmdLen--
        lw $s6, a_history($s5) # $s6 = a_history[cmdLen]
        addi $s6, $s6, 180 # $s6 = the reverse direction of alpha
        sw $s6, a_current
        jal ROTATE
        nop

goBackward_toTurningPoint:
        lw $t9, x_history($s5) # $t9 = x_history[i]
        lw $t7, y_history($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_current # update heading
        jal ROTATE

        addi $s5, $zero, 4
        sw $s5, l_history # reset l_history = 0
        j printCmd

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_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

#chuc nang marsbot
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 # isGoing = 0
        sw $t7, isGoing
        lw $k0, 0($sp) # restore back up
        addi $sp, $sp, -4
        lw $at, 0($sp)
        addi $sp, $sp, -4
        jr $ra

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, isGoing      # isGoing = 0

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

        jr $ra

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

        jr $ra

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, isTracking

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

        jr      $ra

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_current
        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

        jr      $ra

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      $t2, cmdCode($t1)           # $t2 = cmdCode[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:       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, cmdLen                     # $t3 = cmdLen
        addi    $t1, $zero, -1          # $t1 = -1 = i

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

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

        sw      $zero, cmdLen                   # reset cmdLen = 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:
        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 cmdCode
        la $s1, cmdCode

        # load address of prev_cmdCode
        la $s2, prev_cmdCode

strCpyPrevToCur_loop:
        beq $t2, 3, strCpyPrevToCur_end
```

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

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

        j strCpyPrevToCur_loop

strCpyPrevToCur_end:
        # reset code length
        li $t3, 3
        sw $t3, cmdLen

        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:
        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_cmdCode
        la $s1, prev_cmdCode

        # load address of cmdCode
        la $s2, cmdCode

strCpyCurToPrev_loop:
```

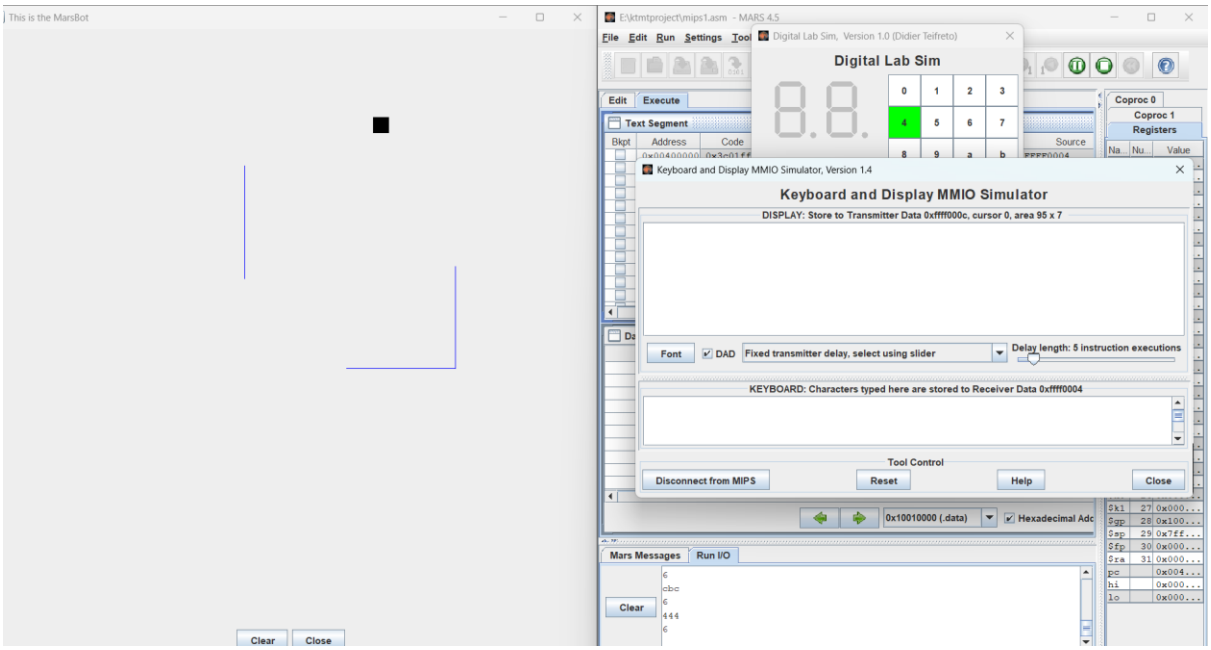
```
        beq $t2, 3, strCpyCurToPrev_end

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

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

        j strCpyCurToPrev_loop

strCpyCurToPrev_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

.ktext  0x80000180
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)

#xu ly chay
get_cod:        li      $t1, IN_ADDRESS_HEXA_KEYBOARD
        li      $t2, OUT_ADDRESS_HEXA_KEYBOARD

scan_row1:      li      $t3, 0x81 # row in digital lab sim
        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, cmdCode
            la      $s2, cmdLen
            lw      $s3, 0($s2)                 # $s3 = strlen(cmdCode)
            addi    $t4, $t4, -1            # $t4 = i

store_code_loop:
            addi    $t4, $t4, 1
            bne     $t4, $s3, store_code_loop
            add     $s1, $s1, $t4           # $s1 = cmdCode + i
            sb      $s0, 0($s1)                 # cmdCode[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 cmdLen

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:        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
```

4. Kết quả



Bài 2. Vẽ hình trên màn hình Bitmap

Viết chương trình vẽ một quả bóng hình tròn 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.
- 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), tăng tốc độ (Z), giảm tốc độ (X) trong bộ giả lập Keyboard and Display MMIO Simulator).
- Vị trí bóng ban đầu ở giữa màn hình.

1. Phân tích cách làm

- Sử dụng Keyboard and Display MMIO Simulator để phát hiện phím được nhấn.
- Chuyển động của quả bóng có thể biểu diễn bằng cách xóa vòng tròn ở vị trí cũ và vẽ vòng tròn ở vị trí mới

2. Thuật toán :

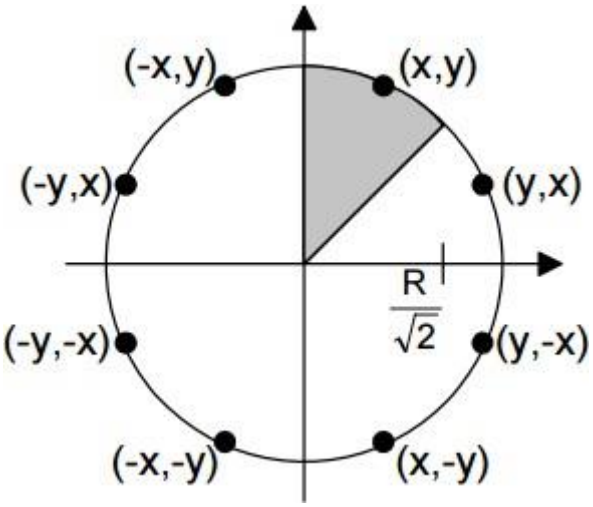
- Vẽ đường tròn: Thiết lập giá trị ban đầu cho tâm đường tròn (x, y), bán kính (R), di chuyển khoảng cách của vòng tròn và thời gian ngủ của mỗi chuyển động.

+ Tạo một mảng để lưu dữ liệu về tất cả các điểm trong đường tròn.

+ Lặp lại giá trị tọa độ x từ 0 đến R, tính giá trị tọa độ y bằng công thức :

$$y = \sqrt{R^2 - x^2}$$

Vì x và y dương nên ta cũng tạo điểm (-x, y);(x, -y); (-x, -y) sau đó hoán đổi giá trị của x và y nên ta có 8 điểm để vẽ một vòng tròn. Lưu các điểm này vào mảng.



- Kiểm tra xem bóng có chạm vào mép màn hình không. Nếu tọa độ tâm cộng với R lớn hơn giá trị giới hạn của màn hình, quả bóng sẽ di chuyển theo hướng ngược lại.
- Vẽ hình tròn mới: Xóa hình tròn cũ bằng cách đổi màu hình tròn gần đây thành đen. Cập nhật giá trị mới cho điểm trong vòng tròn và vẽ lại.

3. Mã nguồn :

mips1.asm

```
.eqv SCREEN, 0x10010000
.eqv YELLOW, 0x00FFFF66
.eqv BACKGROUND, 0x00000000

# Thiết lập ký tự
.eqv KEY_A, 0x00000061          # Di chuyển sang trái
.eqv KEY_D, 0x00000064          # Di chuyển sang phải
.eqv KEY_S, 0x00000073          # Di chuyển xuống dưới
.eqv KEY_W, 0x00000077          # Di chuyển lên trên
.eqv KEY_Z, 0x0000007A          # Tăng tốc độ di chuyển
.eqv KEY_X, 0x00000078          # Giảm tốc độ di chuyển
.eqv KEY_ENTER, 0x0000000A      # Chương trình dừng lại

# Thiết lập khoảng cách giữa hai đường tròn
.eqv KHOANG_CACH, 10
.eqv KEY_CODE, 0xFFFF0004
.eqv KEY_READY, 0xFFFF0000
```

```
#===============================================================================================
.data
    Array: .space 512          # Cấp bộ nhớ lưu tọa độ các điểm của đường tròn
.text
    li   $s0, 256              # x = 256 khởi tạo tọa độ x ban đầu của tâm đường tròn
    li   $s1, 256              # y = 256 khởi tạo tọa độ y ban đầu của tâm đường tròn
    li   $s2, 20               # R = 20, R là bán kính của đường tròn
    li   $s3, 512              # SCREEN_WIDTH = 512, chiều rộng màn hình
    li   $s4, 512              # SCREEN_HEIGHT = 512, chiều dài màn hình
    li   $s5, YELLOW           # Đường tròn có màu vàng
    li   $t6, KHOANG_CACH      # Khoảng cách giữa các hình tròn
    li   $s7, 0                # dx = 0, tọa độ x hiện tại của tâm đường tròn
    li   $t8, 0                # dy = 0, tọa độ y hiện tại của tâm đường tròn
    li   $t9, 70               # Thanh ghi lưu trữ thời gian delay (tốc độ di chuyển của hình tròn)
```

```
#=============================================================================================
# HÀM KHỞI TẠO TỌA ĐỘ ĐƯỜNG TRÒN
#=============================================================================================
khoi_tao:
    li  $t0, 0              # Khởi tạo i = 0
    la  $t5, Array          # Lưu địa chỉ của mảng vào thanh ghi $t5
loop:
    slt $v0, $t0, $s2       # v0=1 nếu i < R
    beq $v0, $zero, ket_thuc # v0=0 <=> i >= R thì nhảy đến kết_thuc
    mul $s6, $s2, $s2       # s6 = R * R = R^2
    mul $t3, $t0, $t0       # t3 = i * i = i^2
    sub $t3, $s6, $t3       # t3 = R^2 - i^2
    move $v1, $t3           # v1 = t3
    jal sqrt                # Nhảy đến hàm tính căn của t3

    sw  $a0, 0($t5)         # Lấy giá trị của thanh ghi a0 = sqrt(R^2 - i^2) lưu vào mảng dữ liệu
    addi $t0, $t0, 1        # i = i + 1
    add $t5, $t5, 4         # Đi đến vị trí tiếp theo của mảng dữ liệu
    j   loop
ket_thuc:
```

```
#=============================================================================================
# HÀM LÀM CHO CHƯƠNG TRÌNH DỪNG CHẠY TRONG MỘT KHOẢNG THỜI GIAN
#=============================================================================================
.macro delay(%r)
    addi $a0, %r, 0
    li   $v0, 32
    syscall
.end_macro

# Tạo hàm để đặt lại màu và vẽ thêm đường tròn ở vị trí mới
# Địa chỉ của màu lưu ở thanh ghi %color khi gọi hàm
.macro datmauveduongtron(%color)
    li   $s5, %color
    jal  ham_ve_duong_tron
.end_macro

#=============================================================================================
# HÀM NHẬP DỮ LIỆU TỪ BÀN PHÍM
#=============================================================================================
Start:
doc ky tu:
```

```
    lw  $k1, KEY_READY       # Kiểm tra đã nhập ký tự nào chưa?
    beqz $k1, check_vi_tri   # Nếu k1 != 0 => đã nhập ký tự thì nhảy đến hàm kiểm tra vị trí
    lw  $k0, KEY_CODE        # Thanh ghi k0 lưu giá trị ký tự nhập vào
    beq $k0, KEY_A, case_a   # Di chuyển qua trái
    beq $k0, KEY_D, case_d   # Di chuyển qua phải
    beq $k0, KEY_S, case_s   # Di chuyển xuống dưới
    beq $k0, KEY_W, case_w   # Di chuyển lên trên
    beq $k0, KEY_Z, case_z   # Tăng tốc độ
    beq $k0, KEY_X, case_x   # Giảm tốc độ
    beq $k0, KEY_ENTER, case_enter # Dừng chương trình
    j   check_vi_tri
    nop

case_a:
    jal di_sang_trai
    j   check_vi_tri

case_d:
    jal di_sang_phai
    j   check_vi_tri
```

```
case_s:
    jal di_chuyen_xuong
    j   check_vi_tri

case_w:
    jal di_chuyen_len
    j   check_vi_tri

case_x:
    addi $t9, $t9, 30
    j    check_vi_tri

case_z:
    addi $t9, $t9, -30
    j    check_vi_tri

case_enter:
    j   endProgram
```

```
endProgram:
    li  $v0, 10
    syscall

#=============================================================================================
# CÁC HÀM DI CHUYỂN
#=============================================================================================
di_sang_trai:
    sub $s7, $zero, $t6     # Tọa độ x hiện tại của đường tròn = - khoảng cách giữa 2 đường tròn
    li  $t8, 0
    jr  $ra

di_sang_phai:
    add $s7, $zero, $t6     # Tọa độ x hiện tại của đường tròn = + khoảng cách giữa 2 đường tròn
    li  $t8, 0
    jr  $ra
```

```
di_chuyen_len:
    li  $s7, 0
    sub $t8, $zero, $t6     # Tọa độ y hiện tại của đường tròn = - khoảng cách giữa 2 đường tròn
    jr  $ra

di_chuyen_xuong:
    li  $s7, 0
    add $t8, $zero, $t6     # Tọa độ y hiện tại của đường tròn = + khoảng cách giữa 2 đường tròn
    jr  $ra

#=============================================================================================
# HÀM KIỂM TRA VỊ TRÍ
#=============================================================================================
```

```
check_vi_tri:
phia_ben_phai:
    add $v0, $s0, $s2        # v0 = x0 + R, tọa độ tâm hiện tại + bán kính
    add $v0, $v0, $s7        # Nếu x0 + R + khoảng cách > 512 thì nhảy đến hàm di_sang_trai
    slt $v1, $v0, $s3        # v1 = 1 nếu v0 < 512
    bne $v1, $zero, phia_ben_trai
    jal di_sang_trai
    nop

phia_ben_trai:
    sub $v0, $s0, $s2        # v0 = x0 - R
    add $v0, $v0, $s7        # Nếu x0 - R + khoảng cách < 0 thì nhảy đến hàm di_sang_phai
    slt $v1, $v0, $zero      # v1 = 1 nếu v0 < 0
    beq $v1, $zero, phia_tren
    jal di_sang_phai
    nop

phia_tren:
    sub $v0, $s1, $s2        # v0 = y0 - R
    add $v0, $v0, $t8        # Nếu y0 - R + khoảng cách < 0 thì nhảy đến hàm di_chuyen_len
    slt $v1, $v0, $zero      # v1 = 1 nếu v0 < 0
    beq $v1, $zero, phia_duoi
    jal di_chuyen_xuong
    nop

phia_duoi:
    add $v0, $s1, $s2        # v0 = y0 + R
    add $v0, $v0, $t8        # Nếu y0 + R + khoảng cách > 512 thì nhảy đến hàm di_chuyen_xuong
    slt $v1, $v0, $s4        # v1 = 1 nếu v0 < 512
    bne $v1, $zero, draw
    jal di_chuyen_len
    nop

#=============================================================================================
# HÀM VẼ ĐƯỜNG TRÒN
#=============================================================================================
draw:
    datmauveduongtron(BACKGROUND)  # Vẽ đường tròn trung màu nền
    add $s0, $s0, $s7              # Cập nhật tọa độ x của đường tròn
    add $s1, $s1, $t8              # Cập nhật tọa độ y của đường tròn

    datmauveduongtron(YELLOW)      # Vẽ đường tròn mới màu vàng
    delay($t9)                     # Dừng 1 khoảng thời gian rồi vẽ đường tròn mới
    j Start

ham_ve_duong_tron:
    add  $sp, $sp, -4
    sw   $ra, 0($sp)
    li   $t0, 0                  # Khởi tạo biến i = 0

loop_ve_duong_tron:
    slt  $v0, $t0, $s2           # v0 = 1 nếu i < R
    beq  $v0, $zero, ket_thuc_ve # Nếu v0 = 0 <=> i >= R => ket_thuc_ve
    sll  $t5, $t0, 2             # Dịch trái thanh ghi t0 2 bit
    lw   $t3, Array($t5)         # Nạp sqrt(R^2-i^2) lưu ở Array vào thanh ghi $t3(y)
    move $a0, $t0                # i = $t0 = $a0
    move $a1, $t3                # j = $t3 = $a1
    jal  ve_diem                 # Vẽ 2 điểm (x0 + i, y0 + j), (x0 + j, y0 + i) trên phần tử thứ I
    sub  $a1, $zero, $t3
    jal  ve_diem                 # Vẽ 2 điểm (x0 + i, y0 - j), (x0 + j, y0 - i) trên phần tử thứ II
    sub  $a0, $zero, $t0
    jal  ve_diem                 # Vẽ 2 điểm (x0 - i, y0 - j), (x0 - j, y0 - i) trên phần tử thứ III
    add  $a1, $zero, $t3
    jal  ve_diem                 # Vẽ 2 điểm (x0 - i, y0 + j), (x0 - j, y0 + i) trên phần tử thứ IV
    addi $t0, $t0, 1
    j    loop_ve_duong_tron
```

```
ket_thuc_ve:
    lw   $ra, 0($sp)
    add  $sp, $sp, 0
    jr   $ra

# Ham vẽ điểm trên đường tròn
ve_diem:
    add  $t1, $s0, $a0          # xi = x0 + i
    add  $t4, $s1, $a1          # yi = y0 + j
    mul  $t2, $t4, $s3          # yi * SCREEN_WIDTH
    add  $t1, $t1, $t2          # yi * SCREEN_WIDTH + xi (Tọa độ 1 chiều của điểm ảnh)
    sll  $t1, $t1, 2            # Địa chỉ tương đối của điểm ảnh
    sw   $s5, SCREEN($t1)       # Vẽ ảnh

    add  $t1, $s0, $a1          # xi = x0 + j
    add  $t4, $s1, $a0          # yi = y0 + i
    mul  $t2, $t4, $s3          # yi * SCREEN_WIDTH
    add  $t1, $t1, $t2          # yi * SCREEN_WIDTH + xi (Tọa độ 1 chiều của điểm ảnh)
    sll  $t1, $t1, 2            # Địa chỉ tương đối của điểm ảnh
    sw   $s5, SCREEN($t1)       # Vẽ ảnh
```

```
# Ham tính căn của t3
sqrt:
    mtc1 $v1, $f1               # Đưa giá trị trong thanh ghi v1 vào thanh ghi f1
    cvt.s.w $f1, $f1            # Chuyển giá trị của f1 tương đương với giá trị số nguyên 32 bit
    sqrt.s $f1, $f1             # Tính căn bậc hai của giá trị thanh ghi f1
    cvt.w.s $f1, $f1            # Chuyển f1 về dạng 32-bit
    mfc1 $a0, $f1               # Đặt giá trị thanh ghi a0 = f1
    jr   $ra

# End of project
```

4. Kết quả chạy chương trình: