

**ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG**



**BÁO CÁO BÀI TẬP
THỰC HÀNH KIẾN TRÚC MÁY TÍNH
ĐỀ TÀI SỐ 1, 4**

Giảng viên hướng dẫn: ThS. Lê Bá Vui

Sinh viên :

Đào Ngọc Bản - 20194482

Nguyễn Anh Vương- 20194723

Nhóm: 3 - Lớp: Việt Nhật K64

Mục Lục

A. Chủ đề 1

- I. Đề bài
- II. Hướng giải quyết
- III. Ý nghĩa các hàm trong mã nguồn
 - 1. Khởi tạo
 - 2. checkControl
 - 3. saveHistory
 - 4. back
 - 5. do_one_more
 - 6. righ và left
 - 7. NAVIGATE
 - 8. TRACK và UNTRACK
 - 9. go và stop
 - 10. ConsologError
 - 11. checkCase
 - 11. remove
 - 12. Interrupt
 - 13. backup và restore
- IV. Mã Nguồn
- V. Demo

B. Chủ đề 4

- I. Đề bài
- II. Hướng giải quyết

III. Ý nghĩa các hàm trong mã nguồn

1. GET_INPUT
2. INIT
3. PSCRIPT_TRAVERSAL
4. SLEEP
5. END

IV. Mã nguồn

V. Demo

A. BÀI TẬP 1.

I. ĐỀ BÀI

Curiosity Marsbot

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 đi các mã điều khiển từ một bàn phím ma trận, lập trình viên điều khiển quá trình di chuyển của Marbot như sau:

Mã điều khiển	Ý nghĩa
1b4	Marbot bắt đầu chuyển động
c68	Marbot đứng im
444	Rẽ trái 90* so với phương chuyển động gần đây và giữ hướng mới
666	Rẽ phải 90* so với phương chuyển động gần đây và giữ hướng mới
dad	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 & Display MMIO Simulator. Có lệnh như vậy:

Kích hoạt mã	Ý nghĩa
Phím Enter	Kết thúc nhập mã và yêu cầu Marbot thực thi
Phím Delete	Xoá toàn bộ mã điều khiển đang nhập dở dang

Phím Space

Chạy lại lệnh trước đó

Hãy lập trình để Marsbot có thể hoạt động như đã mô tả.

Đồng thời bổ sung thêm tính năng: 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.

II. Hướng giải quyết

- B1: Mỗi khi người dùng nhập 1 kí tự từ Digital Lab Sim sẽ tạo ra interrupt để lưu kí tự được nhập vào bộ nhớ, tạo nên đoạn code điều khiển.
- B2: Kiểm tra liên tục xem kí tự Enter có được nhập ở Keyboard & Display MMIO Simulator hay không. Khi kí tự Enter được nhập, sẽ kiểm tra xem đoạn code điều khiển có hợp lệ không (gồm 3 kí tự), nếu không sẽ thông báo code lỗi và sang bước 4. Nếu có thì chuyển sang bước 3.
- B3: Lần lượt kiểm tra xem code điều khiển được nhập vào có trùng với các đoạn code điều khiển đã quy định sẵn. Nếu không thì thông báo đoạn code bị lỗi. Ngược lại thực hiện thao tác theo quy định sẵn.
- B4: In ra console code điều khiển đã nhập và xóa lưu trữ trong bộ nhớ.

III. Ý nghĩa các hàm trong mã nguồn

1. Khởi tạo

Key value của bàn phím:

```
# Key
.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
```

Marsbot

```
# Marsbot
.eqv STATE 0xfffff8010
.eqv MOVING 0xfffff8050
.eqv LEAVETRACK 0xfffff8020
.eqv AXISX 0xfffff8030
.eqv AXISY 0xfffff8040
```

Control Code

```

moveCode: .asciiz "1b4"
stopCode: .asciiz "c68"
leftCode: .asciiz "444"
rightCode: .asciiz "666"
trackCode: .asciiz "dad"
UNtrackCode: .asciiz "cbc"
backCode: .asciiz "999"
...

```

Ý nghĩa: Câu lệnh điều khiển Marsbot

Biến lưu input

```

inputCode: .space 50
lengthInputCode: .word 0
nowState: .word 0

```

Ý nghĩa: các biến lưu lại địa chỉ, độ dài, state hiện tại của input code.

Biến lưu đường đi và thời gian hoạt động của Marsbot

```

timeHistory:word 0:100
historyControl:word 0:100 # luu cac buoc da su dung
history: .space 600      # luu toa do cua con bot
length: .word 12          #bytes

```

Ý nghĩa: đường đi của Marsbot được lưu trữ trong mảng path.Mỗi một cạnh được lưu trữ dưới dạng structure:{x,y,z} .Trong đó x,y là toạ độ còn z là hướng đi.Structure mặc định sẽ là {0,0,0}.Độ dài đường đi bắt đầu là 12bytes (3 x 4 byte).

Chờ và đọc ký tự từ Keyboard

```

WAIT:   lw $t5, 0($k1)      #$t5 = [$k1] = HEX_READY
        beq $t5, $zero, WAIT    #if $t5 == 0 then Polling
        nop
        beq $t5, $zero, WAIT
READ:   lw $t6, 0($k0)      #$t6 = [$k0] = HEX_CODE
        beq $t6, 127, continue  #if $t6 == delete key then remove input , 127 is delete key in ascii
        beq $t6, 32, do_one_more #if $t6 == space key then run previous command, 32 is space key in ascii
        bne $t6, '\n', loop      #if $t6 != '\n' then Polling
        nop
        bne $t6, '\n' , loop

```

Ý nghĩa: nếu tại keyboard người dùng ấn **delete**(ascii có mã 127) thì sẽ nhận dạng code và thực hiện thao tác xoá các code đang có trong **inputCode**.Nếu người dùng ấn **space**(ascii 32) thì sẽ thực hiện lại các câu lệnh đã thực hiện bằng cách gọi hàm **do_one_more**.Còn nếu người dùng **enter** thì có nghĩa là ký tự “\n” trong ascii nên sẽ tiếp tục lặp lại thao tác nhập code .

2. checkControl

```

checkControl:
    la $s2, lengthInputCode # kiem tra xem ma dieu khien co do dai = 3 hay khong
    lw $s2, 0($s2)
    bne $s2, 3, consologError # in loi
    la $s3, moveCode
    jal checkCase
    beq $t0, 1, go

    la $s3, stopCode
    jal checkCase
    beq $t0, 1, stop

    la $s3, leftCode
    jal checkCase
    beq $t0, 1, left

    la $s3, rightCode
    jal checkCase
    beq $t0, 1, right

    la $s3, trackCode
    jal checkCase
    beq $t0, 1, track

    la $s3, UNtrackCode
    jal checkCase
    beq $t0, 1, untrack

    la $s3, backCode
    jal checkCase
    beq $t0, 1, back

    beq $t0, 0, consologError # neu khong khop duoc lenh nao thi se bao loi

```

Ý nghĩa: Hàm điều khiển Marsbot . Đầu tiên sẽ nạp độ dài của *inputCode* vào thanh ghi \$s2 . Nếu giá trị bằng 3 chứng tỏ có lỗi . Tiếp theo Marsbot sẽ được điều khiển thông qua các case bên dưới. Đầu tiên sẽ gán từng lệnh vào thanh ghi \$s3 rồi so sánh với input code. Nếu string giống nhau thì thanh ghi \$t0 sẽ = 1 và thực hiện nhảy tới nhãn lệnh được chỉ định , ngược lại \$t0 = 0 và chạy các câu lệnh tiếp theo.

3. saveHistory

```

saveHistory:
    #luu du lieu khi dung jal
    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 $t4, 0($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)
    #
    li $t1, AXISX
    lw $s1, 0($t1)      #s1 = x
    li $t2, AXISY
    lw $s2, 0($t2)      #s2 = y
    la $s4, nowState
    lw $s4, 0($s4)       #s4 = now state
    la $t3, length
    lw $s3, 0($t3)       #$s3 = length (dv: byte)
    la $t4, history
    add $t4, $t4, $s3    #position to save
    sw $s1, 0($t4)       #save x
    sw $s2, 4($t4)       #save y
    sw $s4, 8($t4)       #save state
    addi $s3, $s3, 12     #update length 12 = 3 (word) x 4 (bytes)
    sw $s3, 0($t3)
    #
    #tra lai du lieu
    lw $s4, 0($sp)
    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
    jr $ra
    nop
    jr $ra

```

Ý nghĩa: Lưu lại thông tin về đường đi của Marsbot vào mảng **history**.

Đầu vào: biến **nowState**, **length**.

Đầu tiên dùng **stack** với các thanh ghi cần sử dụng. Lưu lại tọa độ x, y từ các thanh ghi \$t1,\$t2 vào 2 thanh ghi \$s1,\$s2. Load dữ liệu của nowState vào \$s4 , \$t4 sẽ load dữ liệu của history . Lưu lại giá trị x,y vào **history** qua store byte rồi cập nhập giá trị **nowState** mới và độ dài **length** mới . Sau khi lưu thì sẽ phải **restore** lại stack.

Mảng **history** lưu thông tin về đường đi hay đúng hơn là thông tin về các cạnh của đường đi của Marsbot. Mỗi một cạnh gồm 3 thông tin: tọa độ x và y của điểm đầu tiên, hướng đi của cạnh đó.

4. back

```
back:      .text
           sll $a3,$a2,2
           addi $t7,$a1,0
           li $v0,30
           syscall
           sub $a1,$a0,$t7
           sw $a1,timeHistory($a3)
           addi $a1,$a0,0
           li $v0,7
           sw $v0,historyControl($a3)
           addi $a2,$a2,1

           #backup
           addi $sp,$sp,4
           sw $s5, 0($sp)
           addi $sp,$sp,4
           sw $s6, 0($sp)
           addi $sp,$sp,4
           sw $s7, 0($sp)
           addi $sp,$sp,4
           sw $t8, 0($sp)
           addi $sp,$sp,4
           sw $t9, 0($sp)
           jal UNTRACK
           jal GO
           la $s7, history
           la $s5, length
           lw $s5, 0($s5)
           add $s7, $s7, $s5

start:     addi $s5, $s5, -12          #lui lai 1 structure
           addi $s7, $s7, -12          #vi tri cua thong tin ve canh cuoi cung
           lw $s6, 8($s7)             #huong cua canh cuoi cung
           addi $s6, $s6, 180         #nguoc lai huong cua canh cuoi cung
           la $t8, nowState          #marsbot quay nguoc lai
           sw $s6, 0($t8)
           jal NAVIGATE

firstPointEdge:
           lw $t9, 0($s7)            #toa do x cua diem dau tien cua canh
           li $t8, AXISX              #toa do x hien tai
           lw $t8, 0($t8)
           bne $t8, $t9, firstPointEdge
           nop
           bne $t8, $t9, firstPointEdge
           lw $t9, 4($s7)             #toa do y cua diem dau tien cua canh
           li $t8, AXISY              #toa do y hien tai
           lw $t8, 0($t8)
           bne $t8, $t9, firstPointEdge
           nop
           bne $t8, $t9, firstPointEdge
           beq $s5, 0, end
           beq $s5, 0, end
           j start
           nop
           j start
```

```

end:
jal STOP
la $t8, nowState
add $s6, $zero, $zero
sw $s6, 0($t8)           #update heading
la $t8, length
addi $s5, $zero, 12
sw $s5, 0($t8)           #update length = 12
#restore
lw $t9, 0($sp)
addi $sp,$sp,-4
lw $t8, 0($sp)
addi $sp,$sp,-4
lw $s7, 0($sp)
addi $sp,$sp,-4
lw $s6, 0($sp)
addi $sp,$sp,-4
lw $s5, 0($sp)
addi $sp,$sp,-4
jal NAVIGATE
li $a3,7
beq $v1,$a3,point
j consolog

```

Ý nghĩa: điều khiển Marsbot đi ngược lại theo lộ trình nó đã đi và về điểm xuất phát

Đầu vào: mảng **history** lưu thông tin đường đi, biến **length** lưu kích cỡ của mảng **history** theo byte.

Mảng **history** lưu thông tin đường đi. Mỗi thông tin về 1 cạnh gồm tọa độ x và y và hướng đi - 3 số nguyên. Do đó mỗi thông tin đường đi sẽ chiếm 12 byte. Do đó **length** sẽ có giá trị là bội của 12.

Mỗi khi muốn quay ngược lại và đi về điểm đầu tiên của 1 cạnh trên đường đi, ta sẽ lấy hướng đi của cạnh đó và đi ngược lại, đến khi nào gặp điểm có tọa độ như đã lưu thì kết thúc việc đi ngược trên cạnh đó, tiếp tục trên cạnh khác.

5. do_one_more

```
do_one_more:
    li $t1,0
    addi $t9,$a2,0
loopWhile:
    beq $t1,$t9,endLoop
    addi $a3,$t1,0
    sll $a3,$a3,2
    lw $t8,historyControl($a3)
    #case check
    li $v1,1
    beq $t8,$v1,go
    li $v1,2
    beq $t8,$v1,stop
    li $v1,3
    beq $t8,$v1,left
    li $v1,4
    beq $t8,$v1,right
    li $v1,5
    beq $t8,$v1,track
    li $v1,6
    beq $t8,$v1,untrack
    li $v1,7
    beq $t8,$v1,back
point:
    subi $v1,$a2,1
    beq $t1,$v1,endLoop
    addi $a3,$t1,1
    sll $a3,$a3,2
    lw $a0,timeHistory($a3)
    li $v0,32
    syscall
    addi $t1,$t1,1
    j loopWhile
endLoop:
    j continue
```

Ý nghĩa: chạy lại câu lệnh trước đó bằng cách nhấn phím space ở keyboard

Dựa vào mảng chứa các câu lệnh đã hoạt động(**historyControl**) và thời gian nó hoạt động(**timeHistory**). Sau chờ 1 thời gian thì bot sẽ tự chạy hoạt động gần nhất bằng cách so sánh với trạng thái hiện tại.

6. right và left

```
left:
    sll $a3,$a2,2
    addi $t7,$a1,0
    li $v0,30
    syscall
    sub $a1,$a0,$t7
    sw $a1,timeHistory($a3)
    addi $a1,$a0,0
    li $v0,3
    sw $v0,historyControl($a3)
    addi $a2,$a2,1
    #
    addi $sp,$sp,4
    sw $s5, 0($sp)
    addi $sp,$sp,4
    sw $s6, 0($sp)
    #processing
    la $s5, nowState
    lw $s6, 0($s5) #$s6 is state at now
    addi $s6, $s6, -90 #increase state by 90*
    sw $s6, 0($s5) # update nowState
    #restore
    lw $s6, 0($sp)
    addi $sp,$sp,-4
    lw $s5, 0($sp)
    addi $sp,$sp,-4
    jal saveHistory
    jal NAVIGATE
    li $a3,3
    beq $v1,$a3,point
    j consolog
```

```

right:      -----
          sll $a3,$a2,2

          addi $t7,$a1,0
          li $v0,30
          syscall
          sub $a1,$a0,$t7
          sw $a1,timeHistory($a3)
          addi $a1,$a0,0

          li $v0,4
          sw $v0,historyControl($a3)
          addi $a2,$a2,1
          #
          addi $sp,$sp,4
          sw $s5, 0($sp)
          addi $sp,$sp,4
          sw $s6, 0($sp)
          #restore
          la $s5, nowState
          lw $s6, 0($s5) #$s6 is state at now
          addi $s6, $s6, 90 #increase state by 90*
          sw $s6, 0($s5) # update nowState
          #restore
          lw $s6, 0($sp)
          addi $sp,$sp,-4
          lw $s5, 0($sp)
          addi $sp,$sp,-4

          jal saveHistory
          jal NAVIGATE
          li $a3,4
          beq $v1,$a3,point
          j consolog

```

Ý nghĩa:

right : điều khiển marsbot sang trái và in mã điều khiển. Vì là sang trái nên sẽ đi 1 góc 90. Các biến sử dụng được lưu trong stack .Sau khi dùng xong sẽ **restore** lại stack .Cuối cùng là lưu lại đường đi qua hàm **saveHistory** và gọi **NAVIGATE** để cập nhật lại **STATE** rồi thực hiện thao tác theo lệnh.

left : điều khiển marsbot sang trái và in mã điều khiển. Vì là sang trái nên sẽ đi 1 góc -90. Các biến sử dụng được lưu trong stack .Sau khi dùng xong sẽ **restore** lại stack .Cuối cùng là lưu lại đường đi qua hàm **saveHistory** và gọi **NAVIGATE** để cập nhật lại **STATE** rồi thực hiện thao tác theo lệnh.

Đầu vào: biến **nowState**

7. NAVIGATE

```

.. . . .
NAVIGATE:
    #backup
    addi $sp,$sp,4
    sw $t1,0($sp)
    addi $sp,$sp,4
    sw $t2,0($sp)
    addi $sp,$sp,4
    sw $t3,0($sp)
    #processing
    li $t1, STATE # change STATE port
    la $t2, nowState
    lw $t3, 0($t2) #$t3 is heading at now
    sw $t3, 0($t1) # to navigate robot
    #restore
    lw $t3, 0($sp)
    addi $sp,$sp,-4
    lw $t2, 0($sp)
    addi $sp,$sp,-4
    lw $t1, 0($sp)
    addi $sp,$sp,-4

    jr $ra
    nop
    jr $ra

```

Ý nghĩa: quay Marsbot theo hướng có số độ lưu trong **nowState**

Đầu vào: biến **nowState**

Load biến **nowState** và lưu vào địa chỉ **STATE** (0xfffff8010) để Marsbot chuyển hướng.

8. TRACK, UNTRACK

track:

```

    sll $a3,$a2,2

    addi $t7,$a1,0
    li $v0,30
    syscall
    sub $a1,$a0,$t7
    sw $a1,timeHistory($a3)
    addi $a1,$a0,0

    li $v0,5
    sw $v0,historyControl($a3)

    addi $a2,$a2,1
    jal TRACK
    li $a3,5
    beq $v1,$a3,point
    j consolog

```

```

.. -- -
TRACK: #backup
    addi $sp,$sp,4
    sw $at,0($sp)
    addi $sp,$sp,4
    sw $k0,0($sp)
    #processing
    li $at, LEAVETRACK # change LEAVETRACK port
    addi $k0, $zero,1 # to logic 1,
    sb $k0, 0($at) # to start tracking
    #restore
    lw $k0, 0($sp)
    addi $sp,$sp,-4
    lw $at, 0($sp)
    addi $sp,$sp,-4
    jr $ra
    nop
    jr $ra

untrack:
    sll $a3,$a2,2

    addi $t7,$a1,0
    li $v0,30
    syscall
    sub $a1,$a0,$t7
    sw $a1,timeHistory($a3)
    addi $a1,$a0,0

    li $v0,6
    sw $v0,historyControl($a3)
    addi $a2,$a2,1
    jal UNTRACK
    li $a3,6
    beq $v1,$a3,point
    j consolog

UNTRACK:#backup
    addi $sp,$sp,4
    sw $at,0($sp)
    #processing
    li $at, LEAVETRACK # change LEAVETRACK port to 0
    sb $zero, 0($at) # to stop drawing tail
    lw $at, 0($sp) #save du lieu sau khi jal
    addi $sp,$sp,-4
    jr $ra
    nop
    jr $ra

```

Ý nghĩa: điều khiển Marsbot bắt đầu đê lại vết (TRACK) hoặc kết thúc đê lại vết (UNTRACK)

Load 1 vào địa chỉ **LEAVETRACK** (**0xffff8020**) nếu muốn đê lại vết và load 0 nếu muốn kết thúc vết.

9. go, stop

```

go:
    sll $a3,$a2,2

    addi $t7,$a1,0
    li $v0,30
    syscall
    sub $a1,$a0,$t7
    sw $a1,timeHistory($a3)
    addi $a1,$a0,0

    li $v0,1
    sw $v0,historyControl($a3)
    addi $a2,$a2,1
    jal GO
    li $a3,1
    beq $v1,$a3,point
    j consolog
    j continue
GO:   #backup
    addi $sp,$sp,4
    sw $at,0($sp)
    addi $sp,$sp,4
    sw $k0,0($sp)
    #processing
    li $at, MOVING # change MOVING port
    addi $k0, $zero,1 # to logic 1,
    sb $k0, 0($at) # to start running
    #restore
    lw $k0, 0($sp)
    addi $sp,$sp,-4
    lw $at, 0($sp)
    addi $sp,$sp,-4
    jr $ra
    nop
    jr $ra
    ...
    ...

stop:  j consolog
      sll $a3,$a2,2

      addi $t7,$a1,0
      li $v0,30
      syscall
      sub $a1,$a0,$t7
      sw $a1,timeHistory($a3)
      addi $a1,$a0,0

      li $v0,2
      sw $v0,historyControl($a3)
      addi $a2,$a2,1
      jal STOP
      li $a3,2
      beq $v1,$a3,point
      j consolog

```

```

STOP:    #backup
        addi $sp,$sp,4
        sw $at,0($sp)
#processing
        li $at, MOVING # change MOVING port to 0
        sb $zero, 0($at) # to stop
#restore
        lw $at, 0($sp)
        addi $sp,$sp,-4
        jr $ra
        nop
        jr $ra
...

```

Ý nghĩa: điều khiển Marsbot bắt đầu chuyển động (GO) hoặc dừng lại (STOP)
Load 1 vào địa chỉ **MOVING** (**0xffff8050**) nếu muốn để lại vết và load 0 nếu muốn kết thúc vết.

go: dùng stack cấp các biến cần sử dụng .\$k0 sẽ được đổi giá trị thành 1 và \$at sẽ nhận giá trị này rồi chạy MOVING .Sau khi sử dụng thì restore lại stack

stop: Load địa chỉ MOVING và sửa giá trị hiện tại thành 0 . Marsbot sẽ dừng.

10. consologError

```

consologError:
        li $v0, 4
        la $a0, inputCode
        syscall
        nop
        li $v0, 4
        la $a0, Error
        syscall
        nop
        nop
        j continue
        nop
        j continue
...

```

Ý nghĩa: hiện thông báo dialog khi người dùng nhập code điều khiển không đúng.
Sử dụng các syscall 4

11. checkCase

```

checkCase:
    #moi khi dung jal phai luu lai cac bien s,sp.....
    addi $sp,$sp,4
    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)
    #tien hanh check
    addi $t1, $zero, -1
    add $t0, $zero, $zero
    la $s1, inputCode
    loopStringCheck:           # check cho den khi phat hien 1 ky tu khac
        addi $t1, $t1, 1
        add $t2, $s1, $t1
        lb $t2, 0($t2)
        add $t3, $s3, $t1
        lb $t3, 0($t3)
        bne $t2, $t3, falseCase
        bne $t1, 2, loopStringCheck
        nop
        bne $t1, 2, loopStringCheck

trueCase:
    lw $t3, 0($sp) # gan lai cac gia tri truoc da save o truoc
    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
    li $t0,1 #return true false bang $t0
    jr $ra
    nop
    jr $ra

falseCase:
    #restore
    lw $t3, 0($sp)
    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
    li $t0,0      #return true false bang $t0
    jr $ra
    nop
    jr $ra

```

Ý nghĩa: kiểm tra chuỗi *inputCode* bằng với chuỗi s (lưu trữ trong \$s3). Đέ 2 chuỗi bằng nhau đầu tiên sẽ check độ dài 2 chuỗi, sau đó check từng ký tự trong chuỗi. Đầu tiên sử dụng stack để khởi tạo cho các thanh ghi sử dụng đến. Công việc chính ở đây là lưu dữ liệu input code vào thanh ghi \$s1 thì sẽ so sánh từng ký tự với thanh ghi \$s3. Sau ghi đã xác định được 2 string bằng nhau hoặc không bằng nhau thì sẽ *restore* lại stack và trả lại biến trạng thái trong thanh ghi \$t0 (=1 nếu string bằng nhau, =0 nếu không bằng nhau).

12. remove

```

remove:
#luu gia tri jal
addi $sp,$sp,4
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)
#
la $s2, lengthInputCode
lw $t3, 0($s2)                                #$t3 = lengthInputCode
addi $t1, $zero, -1                            #$t1 = -1 = i
addi $t2, $zero, 0                             #$t2 = '\0'
la $s1, inputCode
addi $s1, $s1, -1
for_loop_to_remove:
    addi $t1, $t1, 1                           #i++
    add $s1, $s1, 1                           #$s1 = inputCode + i
    sb $t2, 0($s1)                           #inputCode[i] = '\0'
    bne $t1, $t3, for_loop_to_remove        #if $t1 <=3 continue loop
    nop
    bne $t1, $t3, for_loop_to_remove

add $t3, $zero, $zero
sw $t3, 0($s2)                                #lengthInputCode = 0

#tra lai cac gia tri
lw $s2, 0($sp)
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
nop
jr $ra

```

Ý nghĩa: xóa xâu *inputCode* (nơi lưu trữ code điều khiển nhập vào). Đầu tiên sử dụng stack để cấp các biến ta sử dụng. Sử dụng biến độ dài *lengthInputCode* của *inputCode* để dùng vòng lặp gán từng phần tử trong mảng thành ký tự “\n”. Sau khi gán hết xong thì gán lại độ dài = 0. Cuối cùng là *restore* lại stack.

13. interrupt

```

get_cod:
    li $t1, HEXA_KEYBOARD_IN
    li $t2, HEXA_KEYBOARD_OUT
scan_row1:
    li $t3, 0x81
    sb $t3, 0($t1)
    lbu $a0, 0($t2)
    bnez $a0, get_char
scan_row2:
    li $t3, 0x82
    sb $t3, 0($t1)
    lbu $a0, 0($t2)
    bnez $a0, get_char
scan_row3:
    li $t3, 0x84
    sb $t3, 0($t1)
    lbu $a0, 0($t2)
    bnez $a0, get_char
scan_row4:
    li $t3, 0x88
    sb $t3, 0($t1)
    lbu $a0, 0($t2)
    bnez $a0, get_char
get_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'
        j saveCode
case_1: li $s0, '1'
        j saveCode
case_2: li $s0, '2'
        j saveCode
case_3: li $s0, '3'
        j saveCode
case_4: li $s0, '4'
        j saveCode
case_5: li $s0, '5'
        j saveCode
case_6: li $s0, '6'
        j saveCode
case_7: li $s0, '7'
        j saveCode
case_8: li $s0, '8'
        j saveCode

```

```

case_9: li $s0, '9'
        j saveCode
case_a: li $s0, 'a'
        j saveCode
case_b: li $s0, 'b'
        j saveCode
case_c: li $s0, 'c'
        j saveCode
case_d: li $s0, 'd'
        j saveCode
case_e: li $s0, 'e'
        j saveCode
case_f: li $s0, 'f'
        j saveCode
saveCode:
    la $s1, inputCode
    la $s2, lengthInputCode
    lw $s3, 0($s2)           #$s3 = strlen(inputCode)
    addi $t4, $t4, -1         #$t4 = i
    forLoopSaveCode:
        addi $t4, $t4, 1
        bne $t4, $s3, forLoopSaveCode
        add $s1, $s1, $t4       #$s1 = inputCode + i
        sb $s0, 0($s1)          #inputCode[i] = $s0

        addi $s3, $zero, '\n'    #add '\n' character to end of string
        addi $s1, $s1, 1          #add '\n' character to end of string
        sb $s0, 0($s1)          #add '\n' character to end of string

    addi $s3, $s3, 1
    sw $s3, 0($s2)           #update length of input code

```

Ý nghĩa: Lần lượt quét các hàng của Digital Lab Sim để xem phím nào được bấm. Tiếp đó dựa vào mã được trả về ghi kí tự tương ứng vào bộ nhớ, cụ thể là ghi vào cuối xâu *inputCode*.

14. Backup và Restore

```

    mller $dtl, $t4 # coproc0.$t4 = coproc0.sp <= dtl
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

```

```
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)
```

Ý nghĩa:

Backup : lưu lại những giá trị trong các thanh ghi đang được sử dụng vào stack

Restore : giải phóng thanh ghi.

IV. MÃ NGUỒN

```
.eqv HEXA_KEYBOARD_IN 0xFFFF0012
.eqv HEXA_KEYBOARD_OUT 0xFFFF0014
.eqv HEX_CODE 0xFFFF0004
.eqv HEX_READY 0xFFFF0000

#-----
# Key
.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

#-----
# Marsbot
.eqv STATE 0xffff8010
.eqv MOVING 0xffff8050
.eqv LEAVETRACK 0xffff8020
.eqv AXISX 0xffff8030
.eqv AXISY 0xffff8040
#=====
```

```
.data
#cac lenh dieu khien
```

```
inputCode: .space 50
lengthInputCode: .word 0
```

```

nowState: .word 0
timeHistory:word 0:100
historyControl:word 0:100 # luu cac buoc da su dung
history: .space 600      # luu toa do cua con bot
length: .word 12          #bytes
moveCode: .asciiz "1b4"
stopCode: .asciiz "c68"
leftCode: .asciiz "444"
rightCode: .asciiz "666"
trackCode: .asciiz "dad"
UNtrackCode: .asciiz "cbc"
backCode: .asciiz "999"
Error: .asciiz "Error! An error occurred.\n"
SpaceAsciiiz: .asciiz " "
{text
main:
    li $k0, HEX_CODE
    li $k1, HEX_READY
    li $a2,0      # so thao tac
#
#-----#
# Enable the interrupt of Keyboard matrix 4x4 of Digital Lab Sim
#-----#
    li $t1, HEXA_KEYBOARD_IN
    li $t3, 0x80 # bit 7 = 1 to enable
    sb $t3, 0($t1)
#
#-----#
loop:    nop
WAIT:    lw $t5, 0($k1)           #$t5 = [$k1] = HEX_READY
        beq $t5, $zero, WAIT      #if $t5 == 0 then Polling
        nop
        beq $t5, $zero, WAIT
READ:   lw $t6, 0($k0)           #$t6 = [$k0] = HEX_CODE
        beq $t6, 127 , continue  #if $t6 == delete key then remove input , 127 is delete
key in ascii
        beq $t6, 32, do_one_more #if $t6 == space key then run previous command, 32 is space
key in ascii
        bne $t6, '\n' , loop     #if $t6 != '\n' then Polling
        nop
        bne $t6, '\n' , loop

```

checkControl:

```
la $s2, lengthInputCode # kiem tra xem ma dieu khien co do dai = 3 hay khong
lw $s2, 0($s2)
bne $s2, 3, consologError # in loi
la $s3, moveCode
jal checkCase
beq $t0, 1, go

la $s3, stopCode
jal checkCase
beq $t0, 1, stop

la $s3, leftCode
jal checkCase
beq $t0, 1, left

la $s3, rightCode
jal checkCase
beq $t0, 1, right

la $s3, trackCode
jal checkCase
beq $t0, 1, track

la $s3, UNtrackCode
jal checkCase
beq $t0, 1, untrack

la $s3, backCode
jal checkCase
beq $t0, 1, back

beq $t0, 0, consologError # neu khong khop duoc lenh nao thi se bao loi
```

consolog:

```
li $v0, 4
la $a0, inputCode
syscall
nop
```

continue:

jal remove

nop

j loop

nop

j loop

#

saveHistory:

#luu du lieu khi dung jal

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 \$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)

#-----

li \$t1, AXISX

lw \$s1, 0(\$t1) #s1 = x

li \$t2, AXISY

lw \$s2, 0(\$t2) #s2 = y

la \$s4, nowState

lw \$s4, 0(\$s4) #s4 = now state

la \$t3, length

lw \$s3, 0(\$t3) #\$s3 = length (dv: byte)

la \$t4, history

add \$t4, \$t4, \$s3 #position to save

sw \$s1, 0(\$t4) #save x

sw \$s2, 4(\$t4) #save y

```

sw $s4, 8($t4)      #save state
addi $s3, $s3, 12    #update length 12 = 3 (word) x 4 (bytes)
sw $s3, 0($t3)
#tra lai du lieu
lw $s4, 0($sp)
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

jr $ra
nop
jr $ra
do_one_more:
li $t1,0
addi $t9,$a2,0
loopWhile:
beq $t1,$t9,endLoop
addi $a3,$t1,0
sll $a3,$a3,2
lw $t8,historyControl($a3)
#case check
li $v1,1
beq $t8,$v1,go
li $v1,2
beq $t8,$v1,stop
li $v1,3
beq $t8,$v1,left

```

```
li $v1,4
beq $t8,$v1,right
li $v1,5
beq $t8,$v1,track
li $v1,6
beq $t8,$v1,untrack
li $v1,7
beq $t8,$v1,back
point:
subi $v1,$a2,1
beq $t1,$v1,endLoop
addi $a3,$t1,1
sll $a3,$a3,2
lw $a0,timeHistory($a3)
li $v0,32
syscall
addi $t1,$t1,1
j loopWhile
endLoop:
j continue
back:
```

```
sll $a3,$a2,2

addi $t7,$a1,0
li $v0,30
syscall
sub $a1,$a0,$t7
sw $a1,timeHistory($a3)
addi $a1,$a0,0
li $v0,7
sw $v0,historyControl($a3)
```

```
addi $a2,$a2,1
```

```
#backup
addi $sp,$sp,4
```

```

sw $s5, 0($sp)
addi $sp,$sp,4
sw $s6, 0($sp)
addi $sp,$sp,4
sw $s7, 0($sp)
addi $sp,$sp,4
sw $t8, 0($sp)
addi $sp,$sp,4
sw $t9, 0($sp)
jal UNTRACK
jal GO
la $s7, history
la $s5, length
lw $s5, 0($s5)
add $s7, $s7, $s5

```

start:

addi \$s5, \$s5, -12	#lui lai 1 structure
addi \$s7, \$s7, -12	#vi tri cua thong tin ve canh cuoi cung
lw \$s6, 8(\$s7)	#huong cua canh cuoi cung
addi \$s6, \$s6, 180	#nguoc lai huong cua canh cuoi cung
la \$t8, nowState	#marsbot quay nguoc lai
sw \$s6, 0(\$t8)	
jal NAVIGATE	

firstPointEdge:

lw \$t9, 0(\$s7)	#toa do x cua diem dau tien cua canh
li \$t8, AXISX	#toa do x hien tai
lw \$t8, 0(\$t8)	
bne \$t8, \$t9, firstPointEdge	
nop	
bne \$t8, \$t9, firstPointEdge	
lw \$t9, 4(\$s7)	#toa do y cua diem dau tien cua canh
li \$t8, AXISY	#toa do y hien tai
lw \$t8, 0(\$t8)	
bne \$t8, \$t9, firstPointEdge	
nop	
bne \$t8, \$t9, firstPointEdge	
beq \$s5, 0, end	
nop	
beq \$s5, 0, end	

```

j start
nop
j start
end:
jal STOP
la $t8, nowState
add $s6, $zero, $zero
sw $s6, 0($t8)      #update heading
la $t8, length
addi $s5, $zero, 12
sw $s5, 0($t8)      #update length = 12
#restore
lw $t9, 0($sp)
addi $sp,$sp,-4
lw $t8, 0($sp)
addi $sp,$sp,-4
lw $s7, 0($sp)
addi $sp,$sp,-4
lw $s6, 0($sp)
addi $sp,$sp,-4
lw $s5, 0($sp)
addi $sp,$sp,-4
jal NAVIGATE
li $a3,7
beq $v1,$a3,point
j consolog
#control
track:
sll $a3,$a2,2

addi $t7,$a1,0
li $v0,30
syscall
sub $a1,$a0,$t7
sw $a1,timeHistory($a3)
addi $a1,$a0,0

li $v0,5
sw $v0,historyControl($a3)

```

```
addi $a2,$a2,1  
jal TRACK  
li $a3,5  
beq $v1,$a3,point  
j consolog
```

untrack:

```
sll $a3,$a2,2
```

```
addi $t7,$a1,0  
li $v0,30  
syscall  
sub $a1,$a0,$t7  
sw $a1,timeHistory($a3)  
addi $a1,$a0,0
```

```
li $v0,6  
sw $v0,historyControl($a3)  
addi $a2,$a2,1  
jal UNTRACK  
li $a3,6  
beq $v1,$a3,point  
j consolog
```

go:

```
sll $a3,$a2,2
```

```
addi $t7,$a1,0  
li $v0,30  
syscall  
sub $a1,$a0,$t7  
sw $a1,timeHistory($a3)  
addi $a1,$a0,0
```

```
li $v0,1  
sw $v0,historyControl($a3)  
addi $a2,$a2,1  
jal GO  
li $a3,1  
beq $v1,$a3,point
```

```
j consolog
```

```
stop:
```

```
sll $a3,$a2,2
```

```
addi $t7,$a1,0
```

```
li $v0,30
```

```
syscall
```

```
sub $a1,$a0,$t7
```

```
sw $a1,timeHistory($a3)
```

```
addi $a1,$a0,0
```

```
li $v0,2
```

```
sw $v0,historyControl($a3)
```

```
addi $a2,$a2,1
```

```
jal STOP
```

```
li $a3,2
```

```
beq $v1,$a3,point
```

```
j consolog
```

```
right:
```

```
sll $a3,$a2,2
```

```
addi $t7,$a1,0
```

```
li $v0,30
```

```
syscall
```

```
sub $a1,$a0,$t7
```

```
sw $a1,timeHistory($a3)
```

```
addi $a1,$a0,0
```

```
li $v0,4
```

```
sw $v0,historyControl($a3)
```

```
addi $a2,$a2,1
```

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

```
addi $sp,$sp,4
```

```
sw $s5, 0($sp)
```

```
addi $sp,$sp,4
```

```
sw $s6, 0($sp)
```

```
#restore
```

```
la $s5, nowState
```

```
lw $s6, 0($s5) #$s6 is state at now
```

```

addi $s6, $s6, 90 #increase state by 90*
sw $s6, 0($s5) # update nowState
#restore
lw $s6, 0($sp)
addi $sp,$sp,-4
lw $s5, 0($sp)
addi $sp,$sp,-4

```

```

jal saveHistory
jal NAVIGATE
li $a3,4
beq $v1,$a3,point
j consolog

```

left:

```
sll $a3,$a2,2
```

```

addi $t7,$a1,0
li $v0,30
syscall
sub $a1,$a0,$t7
sw $a1,timeHistory($a3)
addi $a1,$a0,0

```

```

li $v0,3
sw $v0,historyControl($a3)
addi $a2,$a2,1
#-----

```

```

addi $sp,$sp,4
sw $s5, 0($sp)
addi $sp,$sp,4
sw $s6, 0($sp)
#processing
la $s5, nowState
lw $s6, 0($s5) #$s6 is state at now
addi $s6, $s6, -90 #increase state by 90*
sw $s6, 0($s5) # update nowState
#restore
lw $s6, 0($sp)
addi $sp,$sp,-4

```

```
lw $s5, 0($sp)
addi $sp,$sp,-4
```

```
jal saveHistory
jal NAVIGATE
li $a3,3
beq $v1,$a3,point
j consolog
```

remove:

```
#luu gia tri khi jal
addi $sp,$sp,4
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)
```

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

```
la $s2, lengthInputCode
```

```
lw $t3, 0($s2)           #$t3 = lengthInputCode
```

```
addi $t1, $zero, -1      #$t1 = -1 = i
```

```
addi $t2, $zero, 0        #$t2 = '\0'
```

```
la $s1, inputCode
```

```
addi $s1, $s1, -1
```

```
for_loop_to_remove:
```

```
    addi $t1, $t1, 1          #i++
```

```
    add $s1, $s1, 1           #$s1 = inputCode + i
```

```
    sb $t2, 0($s1)           #inputCode[i] = '\0'
```

```
    bne $t1, $t3, for_loop_to_remove  #if $t1 <=3 continue loop
```

```
    nop
```

```
    bne $t1, $t3, for_loop_to_remove
```

```
add $t3, $zero, $zero
```

```
sw $t3, 0($s2)           #lengthInputCode = 0
```

```
#tra lai cac gia tri
```

```
lw $s2, 0($sp)
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
nop
jr $ra
```

checkCase:

```
#moi khi dung jal phai luu lai cac bien s,sp.....  
addi $sp,$sp,4  
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)  
#tien hanh check  
addi $t1, $zero, -1  
add $t0, $zero, $zero  
la $s1, inputCode  
loopStringCheck:      # check cho den khi phat hien 1 ky tu khac  
addi $t1, $t1, 1  
add $t2, $s1, $t1  
lb $t2, 0($t2)  
add $t3, $s3, $t1  
lb $t3, 0($t3)  
bne $t2, $t3, falseCase  
bne $t1, 2, loopStringCheck  
nop  
bne $t1, 2, loopStringCheck
```

trueCase:

```
lw $t3, 0($sp) # gan lai cac gia tri truoc da save o truoc
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
li $t0,1 #return true false bang $t0
jr $ra
nop
jr $ra
```

falseCase:

```
#restore
lw $t3, 0($sp)
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
li $t0,0 #return true false bang $t0
jr $ra
nop
jr $ra
```

consologError:

```
li $v0, 4
la $a0, inputCode
syscall
nop
li $v0, 4
la $a0, Error
syscall
nop
nop
j continue
nop
j continue
```

```

GO: #backup
    addi $sp,$sp,4
    sw $at,0($sp)
    addi $sp,$sp,4
    sw $k0,0($sp)
#processing
    li $at, MOVING # change MOVING port
    addi $k0, $zero,1 # to logic 1,
    sb $k0, 0($at) # to start running
#restore
    lw $k0, 0($sp)
    addi $sp,$sp,-4
    lw $at, 0($sp)
    addi $sp,$sp,-4
    jr $ra
    nop
    jr $ra

STOP:      #backup
    addi $sp,$sp,4
    sw $at,0($sp)
#processing
    li $at, MOVING # change MOVING port to 0
    sb $zero, 0($at) # to stop
#restore
    lw $at, 0($sp)
    addi $sp,$sp,-4
    jr $ra
    nop
    jr $ra

TRACK:     #backup
    addi $sp,$sp,4
    sw $at,0($sp)
    addi $sp,$sp,4
    sw $k0,0($sp)
#processing
    li $at, LEAVETRACK # change LEAVETRACK port
    addi $k0, $zero,1 # to logic 1,
    sb $k0, 0($at) # to start tracking
#restore

```

```

lw $k0, 0($sp)
addi $sp,$sp,-4
lw $at, 0($sp)
addi $sp,$sp,-4
jr $ra
nop
jr $ra

UNTRACK:#backup
addi $sp,$sp,4
sw $at,0($sp)
#processing
li $at, LEAVETRACK # change LEAVETRACK port to 0
sb $zero, 0($at) # to stop drawing tail
lw $at, 0($sp) #save du lieu sau khi jal
addi $sp,$sp,-4
jr $ra
nop
jr $ra

```

NAVIGATE:

```

#backup
addi $sp,$sp,4
sw $t1,0($sp)
addi $sp,$sp,4
sw $t2,0($sp)
addi $sp,$sp,4
sw $t3,0($sp)
#processing
li $t1, STATE # change STATE port
la $t2, nowState
lw $t3, 0($t2) #$t3 is heading at now
sw $t3, 0($t1) # to navigate robot
#restore
lw $t3, 0($sp)
addi $sp,$sp,-4
lw $t2, 0($sp)
addi $sp,$sp,-4
lw $t1, 0($sp)
addi $sp,$sp,-4

```

```

jr $ra
nop
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)

get_cod:
    li $t1, HEXA_KEYBOARD_IN
    li $t2, HEXA_KEYBOARD_OUT

scan_row1:
    li $t3, 0x81
    sb $t3, 0($t1)
    lbu $a0, 0($t2)
    bnez $a0, get_char

scan_row2:
    li $t3, 0x82
    sb $t3, 0($t1)

```

```

lbu $a0, 0($t2)
bnez $a0, get_char
scan_row3:
    li $t3, 0x84
    sb $t3, 0($t1)
    lbu $a0, 0($t2)
    bnez $a0, get_char
scan_row4:
    li $t3, 0x88
    sb $t3, 0($t1)
    lbu $a0, 0($t2)
    bnez $a0, get_char
get_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'
             j saveCode
case_1:      li $s0, '1'
             j saveCode
case_2:      li $s0, '2'
             j saveCode
case_3:      li $s0, '3'
             j saveCode
case_4:      li $s0, '4'
             j saveCode

```

```

case_5:      li $s0, '5'
              j saveCode
case_6:      li $s0, '6'
              j saveCode
case_7:      li $s0, '7'
              j saveCode
case_8:      li $s0, '8'
              j saveCode
case_9:      li $s0, '9'
              j saveCode
case_a:      li $s0, 'a'
              j saveCode
case_b:      li $s0, 'b'
              j saveCode
case_c:      li $s0, 'c'
              j saveCode
case_d:      li $s0, 'd'
              j saveCode
case_e:      li $s0, 'e'
              j saveCode
case_f:      li $s0, 'f'
              j saveCode
saveCode:
    la $s1, inputCode
    la $s2, lengthInputCode
    lw $s3, 0($s2)           #$s3 = strlen(inputCode)
    addi $t4, $t4, -1        #$t4 = i
forLoopSaveCode:
    addi $t4, $t4, 1
    bne $t4, $s3, forLoopSaveCode
    add $s1, $s1, $t4          #$s1 = inputCode + i
    sb $s0, 0($s1)            #inputCode[i] = $s0

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

```

addi \$s3, \$s3, 1

```
sw $s3, 0($s2) #update length of input code
```

next_pc:

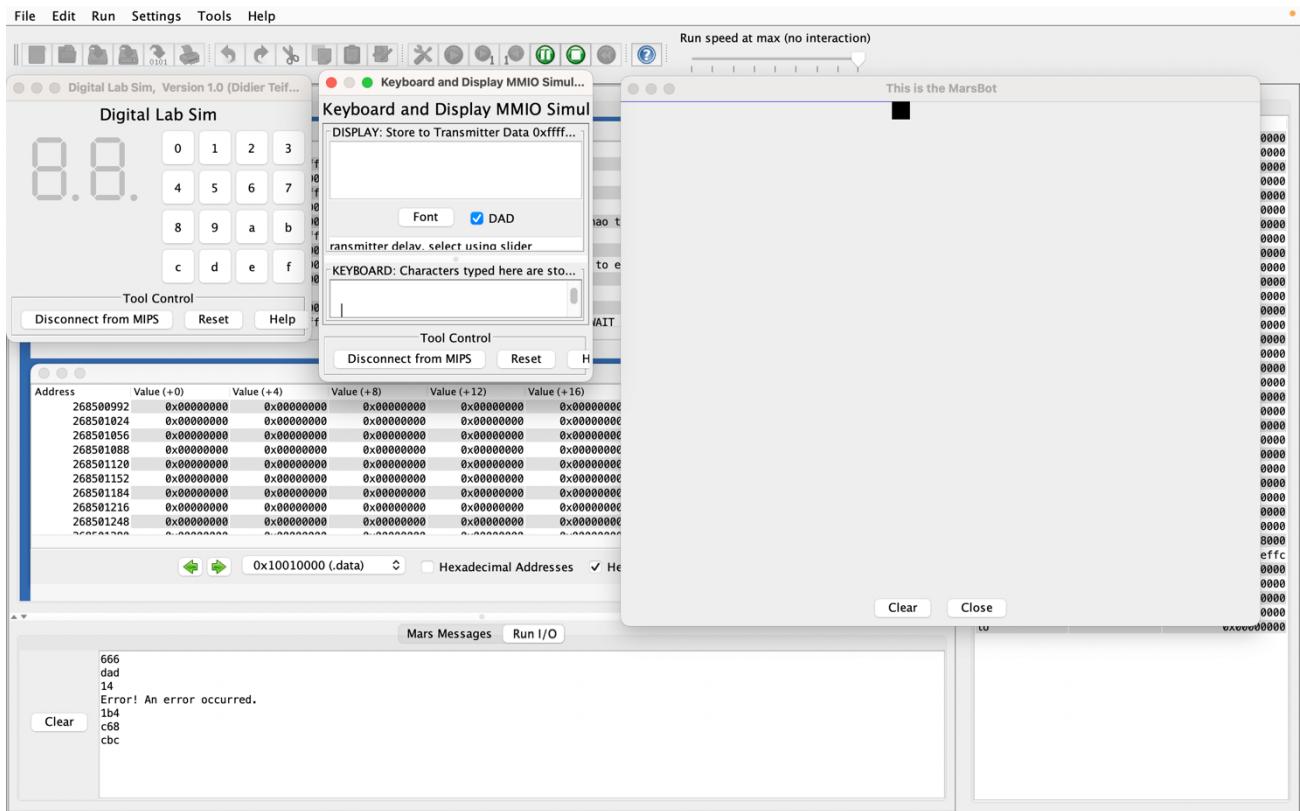
```
mfc0 $at, $14 # $at <= Coproc0.$14 = Coproc0.epc  
addi $at, $at, 4 # $at = $at + 4 (next instruction)  
mtc0 $at, $14 # Coproc0.$14 = Coproc0.epc <= $at
```

restore:

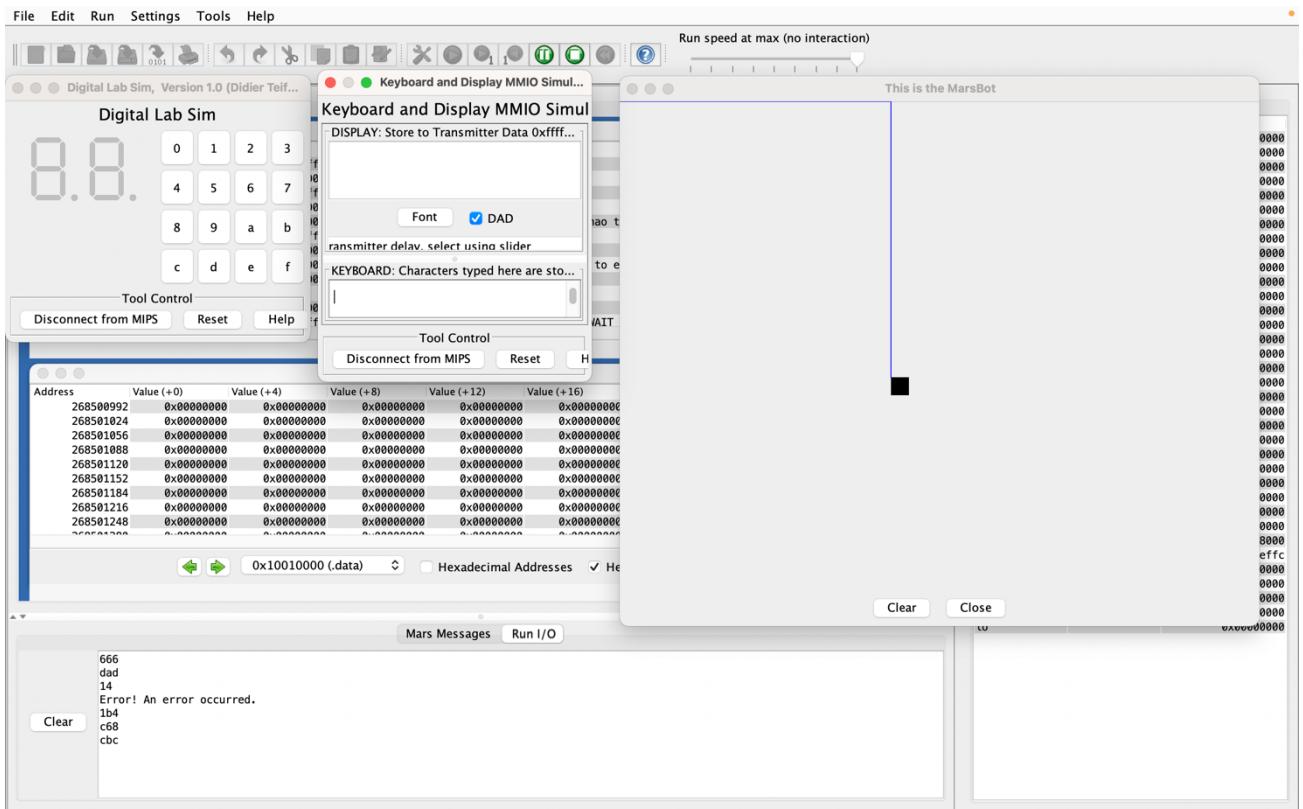
```
lw $s3, 0($sp)  
addi $sp,$sp,-4  
lw $t4, 0($sp)  
addi $sp,$sp,-4  
lw $s2, 0($sp)  
addi $sp,$sp,-4  
lw $s1, 0($sp)  
addi $sp,$sp,-4  
lw $s0, 0($sp)  
addi $sp,$sp,-4  
lw $at, 0($sp)  
addi $sp,$sp,-4  
lw $a0, 0($sp)  
addi $sp,$sp,-4  
lw $t3, 0($sp)  
addi $sp,$sp,-4  
lw $t2, 0($sp)  
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

HÌNH ẢNH MÔ PHỎNG



*Điều khiển Marsbot bằng các câu lệnh.
In ra lỗi nếu nhập sai
In các câu lệnh ra console.*



MarsBot tự chạy những câu lệnh trước sau khi ấn dấu cách.

B. Chủ đề 4

I. Đề bài

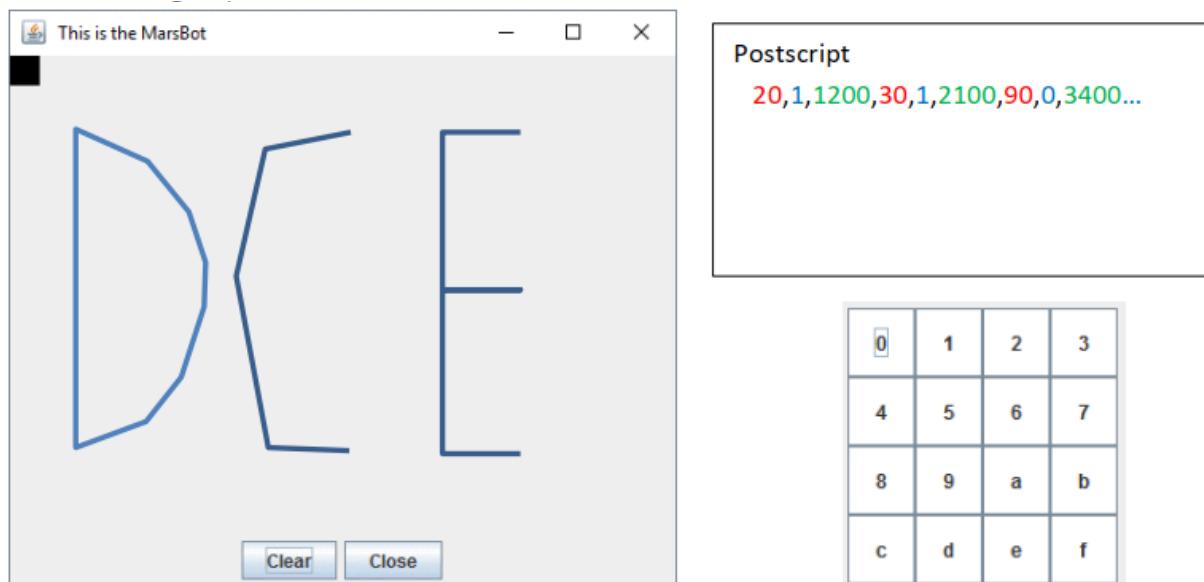
Postscript CNC Marsbot

Máy gia công cơ khí chính xác CNC Marsbot được dùng để cắt tấm kim loại theo các đường nét được qui định trước. CNC Marsbot có một lưỡi cắt dịch chuyển trên tấm kim loại, với giả định rằng:

- Nếu lưỡi cắt dịch chuyển nhưng không cắt tấm kim loại, tức là Marsbot di chuyển nhưng không để lại vết (Track)
- Nếu lưỡi cắt dịch chuyển và cắt tấm kim loại, tức là Marsbot di chuyển và có để lại vết.

Để điều khiển Marsbot cắt đúng như hình dạng mong muốn, người ta nạp vào Marsbot một mảng cấu trúc gồm 3 phần tử:

- <Góc chuyển động>,<Cắt/Không cắt>,<Thời gian>
 - Trong đó <Góc chuyển động> là góc của hàm HEADING của Marsbot
 - <Cắt/Không cắt> thiết lập lưu vết/không lưu vết
 - <Thời gian> là thời gian duy trì quá trình vận hành hiện tại
- Hãy lập trình để CNC Marsbot có thể:
- Thực hiện cắt kim loại như đã mô tả
 - Nội dung postscript được lưu trữ cố định bên trong mã nguồn
 - Mã nguồn chứa 3 postscript và người dùng sử dụng 3 phím 0, 4, 8 trên bàn phím Key Matrix để chọn postscript nào sẽ được gia công.
 - Một postscript chứa chữ DCE cần gia công. Hai script còn lại sinh viên tự đề xuất (tối thiểu 10 đường cắt)



II. Hướng giải quyết

```

.data
pscript1: .word 90,0,2000,180,0,3000,180,1,10000, 90,1,2500, 45,1,3520, 0,1,5000, 315,1,3520, 270,1,2500, 90,0,15000, 270,1,2500, 225,1,3520, 180,1,5000, 135,1,3520, 90,1,2500, 90,0,10000,
pscript1_end: .word #DCE
pscript2: .word 90,0,2000,180,0,13000, 0,1,10000, 143,1,12500, 0,1,10000, 90,0,8000, 206,1,11180, 26,0,11180, 154,1,11180, 334,0,5590, 270,1,4900, 90,0,4900, 334,0,5590, 90,0,10000, 180,1,50
pscript2_end: .word #NAV
pscript3: .word 90,0,2000,180,0,3000, 180,1,10000, 90,1,2500, 45,1,3520, 0,1,5000, 315,1,3520, 270,1,2500, 90,0,10000, 180,0,10000, 0,1,10000, 143,1,12500, 0,1,10000, 90,0,8000, 180,1,10000,
pscript3_end: .word #DNB

```

- Dùng mảng pscript để lưu rotate, track, time và pscript_end để đánh dấu là kết thúc mảng.

- Sử dụng Digital Lab Sim để chọn pscript và MarsBot để tiến hành cắt hình dạng mà mình mong muốn.

```

- - - -
GET_INPUT:
    li $t5, 0x1
    sb $t5, 0($t3)
    lb $a0, 0($t4)
    bne $a0, 0x11, NOT_0 # kiem tra xem co phai phim 0 khong
    la $a1, pscript1
    la $a2, pscript1_end
    j INIT
NOT_0:
    li $t5, 0x2
    sb $t5, 0($t3)
    lb $a0, 0($t4)
    bne $a0, 0x12, NOT_4 # kiem tra xem co phai phim 4 khong
    la $a1, pscript2
    la $a2, pscript2_end
    j INIT
NOT_4:
    li $t5, 0x4
    sb $t5, 0($t3)
    lb $a0, 0($t4)
    bne $a0, 0x14, LOOP # kiem tra xem co phai phim 8 khong
    la $a1, pscript3
    la $a2, pscript3_end
    j INIT
LOOP: j GET_INPUT # Neu cac phim 0, 4, 8 khong duoc nhan thi quay lai doc tiep

```

- Chạy một vòng lặp cho đến khi có một phím 0 hoặc 4 hoặc 8 được nhập để chọn pscript.

```

PSCRIPT_TRAVERSAL:

    # rotate
    addi $t0, $t0, 3
    sll $t1, $t0, 2
    add $t2, $a1, $t1
    lw $t3, 0($t2) # pscript i
    add $a0, $zero, $t3
    jal ROTATE

    # lay track
    addi $t2, $t2, 4
    lw $t3, 0($t2)
    beq $t3, $zero, NO_CUT
    jal UNTRACK
    jal TRACK # and draw new track line
    NO_CUT:
    # lay time
    addi $t2, $t2, 4
    lw $t3, 0($t2)

SLEEP:
    addi $v0,$zero,32 # Keep running by sleeping
    add $a0, $zero, $t3
    syscall
    jal UNTRACK # keep old track
    addi $t2, $t2, 4
    bne $t2, $a2, PSCRIPT_TRAVERSAL

```

- Chạy một vòng lặp hết pscript để lấy 3 phần tử 1 lần để lấy rotate, track và time và cắt

III. Ý nghĩa các hàm trong mã nguồn

```

GET_INPUT:
    li $t5, 0x1
    sb $t5, 0($t3)
    lb $a0, 0($t4)
    bne $a0, 0x11, NOT_0 # kiem tra xem co phai phim 0 khong
    la $a1, pscript1
    la $a2, pscript1_end
    j INIT
NOT_0:
    li $t5, 0x2
    sb $t5, 0($t3)
    lb $a0, 0($t4)
    bne $a0, 0x12, NOT_4 # kiem tra xem co phai phim 4 khong
    la $a1, pscript2
    la $a2, pscript2_end
    j INIT
NOT_4:
    li $t5, 0x4
    sb $t5, 0($t3)
    lb $a0, 0($t4)
    bne $a0, 0x14, LOOP # kiem tra xem co phai phim 8 khong
    la $a1, pscript3
    la $a2, pscript3_end
    j INIT
LOOP: j GET_INPUT # Neu cac phim 0, 4, 8 khong duoc nhan thi quay lai doc tiep

```

Ý nghĩa: Chạy một vòng lặp cho đến khi có một phím 0 hoặc 4 hoặc 8 được nhập để chọn pscript.

- + Nếu 0 được chọn thì \$a1 sẽ lưu địa chỉ của mảng pscript1 và \$a2 sẽ lưu địa chỉ tiếp theo sau phần tử cuối của mảng pscript1
- + Nếu 4 được chọn thì \$a1 sẽ lưu địa chỉ của mảng pscript2 và \$a2 sẽ lưu địa chỉ tiếp theo sau phần tử cuối của mảng pscript2
- + Nếu 8 được chọn thì \$a1 sẽ lưu địa chỉ của mảng pscript3 và \$a2 sẽ lưu địa chỉ tiếp theo sau phần tử cuối của mảng pscript3

```

INIT:
    li $t0, -3 # i
    jal GO

```

Ý nghĩa: Khởi tạo i và cho Bot bắt đầu chạy

PSCRIPT_TRAVERSAL:

```
# rotate
addi $t0, $t0, 3
sll $t1, $t0, 2
add $t2, $a1, $t1
lw $t3, 0($t2) # pscript i
add $a0, $zero, $t3
jal ROTATE

# lay track
addi $t2, $t2, 4
lw $t3, 0($t2)
beq $t3, $zero, NO_CUT
jal UNTRACK
jal TRACK # and draw new track line
NO_CUT:
# lay time
addi $t2, $t2, 4
lw $t3, 0($t2)
```

Ý nghĩa:

- + Rotate ở đây sẽ lấy phần tử mảng ở vị trí i (tức là góc xoay)
- + Lấy phần tử mảng i + 1 để lấy track
- + Lấy phần tử mảng i + 2 để lấy time
- + Sau đó tăng i lên 3 để duyệt tiếp 3 phần tử tiếp theo

SLEEP:

```
addi $v0,$zero,32 # Keep running by sleeping
add $a0, $zero, $t3
syscall
jal UNTRACK # keep old track
addi $t2, $t2, 4
bne $t2, $a2, READ_PSCRIPT
```

Ý nghĩa: Sleep với time lấy được ở trên để Bot vẽ.

END:

```
jal STOP
li $v0, 10
syscall
```

Ý nghĩa: Stop Bot và dừng chương trình.

IV. Mă nguồn

```
.eqv HEADING 0xfffff8010
.eqv MOVING 0xfffff8050
.eqv LEAVETRACK 0xfffff8020
.eqv WHEREX 0xfffff8030
.eqv WHEREY 0xfffff8040

.eqv OUT_ADDRESS_HEXA_KEYBOARD 0xFFFFF0014
.eqv IN_ADDRESS_HEXA_KEYBOARD 0xFFFFF0012

.data
pscript1: .word 90,0,2000,180,0,3000,180,1,10000, 90,1,2500, 45,1,3520, 0,1,5000, 315,1,3520,
270,1,2500, 90,0,15000, 270,1,2500, 225,1,3520, 180,1,5000, 135,1,3520, 90,1,2500, 90,0,10000,
270,1,5000, 0,1,10000, 90,1,5000, 180,0,5000, 270,1,5000, 90,0,10000
pscript1_end: .word #DCE
pscript2: .word 90,0,2000,180,0,13000, 0,1,10000, 143,1,12500, 0,1,10000, 90,0,8000, 206,1,11180,
26,0,11180, 154,1,11180, 334,0,5590, 270,1,4900, 90,0,4900, 334,0,5590, 90,0,10000, 180,1,5000,
135,1,7071, 45,1,7071, 0,1,5000, 90,0,10000
pscript2_end: .word #NAV
pscript3: .word 90,0,2000,180,0,3000, 180,1,10000, 90,1,2500, 45,1,3520, 0,1,5000, 315,1,3520,
270,1,2500, 90,0,10000, 180,0,10000, 0,1,10000, 143,1,12500, 0,1,10000, 90,0,6000, 180,1,10000,
90,1,2500, 45,1,3520, 315,1,3520, 270,1,2500, 90,0,2500, 45,1,3520, 315,1,3520, 270,1,2500,
90,0,10000
pscript3_end: .word #DNB

.text
    li $t3, IN_ADDRESS_HEXA_KEYBOARD
    li $t4, OUT_ADDRESS_HEXA_KEYBOARD
GET_INPUT:
    li $t5, 0x1
    sb $t5, 0($t3)
    lb $a0, 0($t4)
    bne $a0, 0x11, NOT_0 # kiem tra xem co phai phim 0 khong
    la $a1, pscript1
    la $a2, pscript1_end
    j INIT
NOT_0:
    li $t5, 0x2
    sb $t5, 0($t3)
    lb $a0, 0($t4)
    bne $a0, 0x12, NOT_4 # kiem tra xem co phai phim 4 khong
    la $a1, pscript2
    la $a2, pscript2_end
    j INIT
NOT_4:
    li $t5, 0x4
```

```

sb $t5, 0($t3)
lb $a0, 0($t4)
bne $a0, 0x14, LOOP # kiem tra xem co phai phim 8 khong
la $a1, pscript3
la $a2, pscript3_end
j INIT
LOOP: j GET_INPUT # Neu cac phim 0, 4, 8 khong duoc nhan thi quay lai doc tiep

```

INIT:

```

li $t0, -3 # i
jal GO

```

PSCRIPT_TRAVERSAL:

```

# rotate
addi $t0, $t0, 3
sll $t1, $t0, 2
add $t2, $a1, $t1
lw $t3, 0($t2) # pscript i
add $a0, $zero, $t3
jal ROTATE

# lay track
addi $t2, $t2, 4
lw $t3, 0($t2)
beq $t3, $zero, NO_CUT
jal UNTRACK
jal TRACK # and draw new track line
NO_CUT:
# lay time
addi $t2, $t2, 4
lw $t3, 0($t2)

```

SLEEP:

```

addi $v0,$zero,32 # Keep running by sleeping
add $a0, $zero, $t3
syscall
jal UNTRACK # keep old track
addi $t2, $t2, 4
bne $t2, $a2, PSCRIPT_TRAVERSAL

```

END:

```

jal STOP
li $v0, 10
syscall

```

GO:

```

li $at, MOVING
addi $k0, $zero,1

```

```
sb $k0, 0($at)
jr $ra
```

STOP:

```
li $at, MOVING
sb $zero, 0($at)
jr $ra
```

TRACK:

```
li $at, LEAVETRACK
addi $k0, $zero,1
sb $k0, 0($at)
jr $ra
```

UNTRACK:

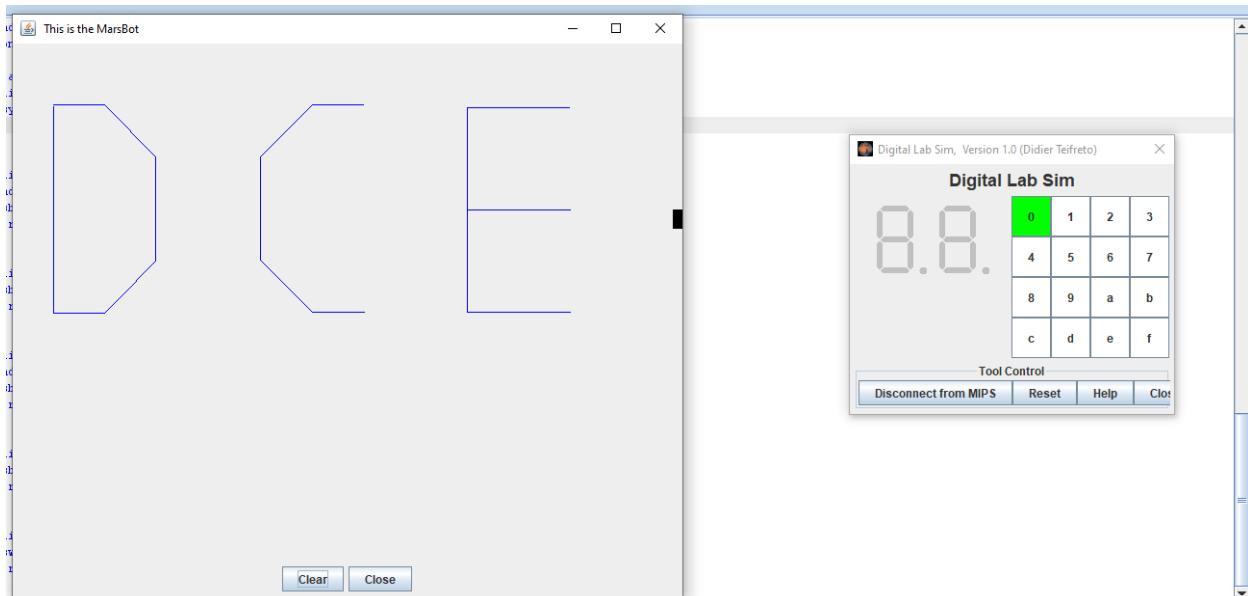
```
li $at, LEAVETRACK
sb $zero, 0($at)
jr $ra
```

ROTATE:

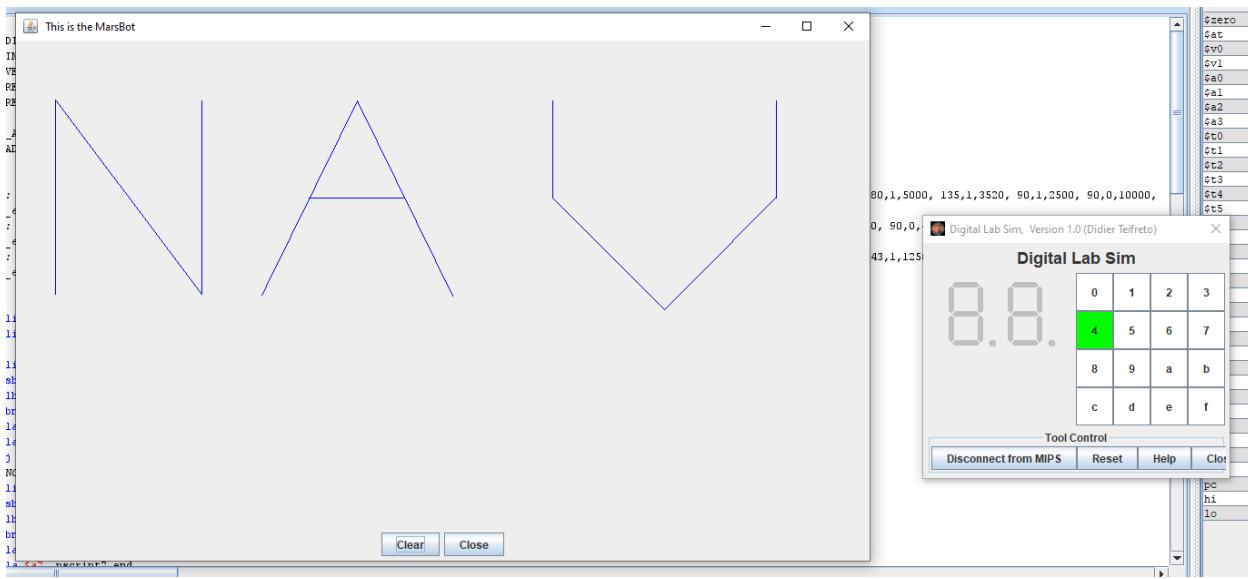
```
li $at, HEADING
sw $a0, 0($at)
jr $ra
```

V. Demo

- Phím 0:



- Phím 4:



- Phím 8:

