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:130938

Giảng viên hướng dẫn: ThS. Lê Bá Vui
Nhóm sinh viên thực hiện: Nguyễn Gia Lộc 20194607 (chủ đề 4)
Nguyễn Thị Phương Lý 20194612 (chủ đề 1)

Nhóm: 17

Hà Nội, ngày 22 tháng 7 năm 2022.

Chủ đề 4: Postscript CNC Marsbot
Nguyễn Gia Lộc - 20194607

1. Phân tích bài toán:

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>,<Thời gian>,<Cắt / Không cắt >
- Trong đó là góc của hàm HEADING của Marsbot - là thời gian duy trì quá trình vận hành hiện tại - thiết lập lưu vết/không lưu vết

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)

2. Cách thực hiện:

-Thủ tục polling: phát hiện ra phím được ấn trong Digi Lab Sim (Home Assignment 1 – POOLING- Week11), các phím 0, 4, 8 sẽ tương ứng với các postscript khác nhau

```
polling:
        li $t5, 0x01            # check row 1 with key 0, 1, 2, 3
        sb $t5, 0($t3)          # must reassign expected row
        lb $a0, 0($t4)          # $a0 = read scan code of key button
        bne $a0, 0x11, NOT_NUMPAD_0 # if not numpad 0
        la $a1, postscript1
        j START

        NOT_NUMPAD_0:
        li $t5, 0x02            # check row 2 with key 4, 5, 6, 7
        sb $t5, 0($t3)          # must reassign expected row
        lb $a0, 0($t4)          # read scan code of key button
        bne $a0, 0x12, NOT_NUMPAD_4  # if not numpad 4
        la $a1, postscript2
        j START

        NOT_NUMPAD_4:
        li $t5, 0X04                    # check row 3 with key 8, 9, a, b
        sb $t5, 0($t3)                  # must reassign expected row
        lb $a0, 0($t4)                  # read scan code of key button
        bne $a0, 0x14, back_to_polling  # if not numpad 8
        la $a1, postscript3
        j START
back_to_polling: j polling                  # neu numpad 0, 4, 8 khong duoc chon -> quay lai doc tiep
```

-Thủ tục READ_POSTCRIPT: gồm các thủ tục con READ_ROTATE, READ_TRACK, READ_TIME

+ READ_ROTATE: thanh ghi $t0 lưu giữ giá trị góc quay, thanh ghi $t5 lưu giữ mã ASCII của mỗi kí tự khi đọc vào, vì thế sau mỗi lần đọc kí tự lưu vào thanh ghi $t5, cần trừ đi 48 đơn vị để $t5 lưu giữ giá trị của kí tự đọc vào. Sau khi đọc xong góc quay $t0, sẽ nhảy đến thủ tục ROTATE (jal ROTATE)

```
READ_ROTATE:                    # doc goc quay
                                # $t6 = i = 0 , i la chi so duyet mang xau
        add $t7, $a1, $t6       # $t7= $a1 + $t6 =  postscript[0] + i = address of postscript[i], dich bit
        lb $t5, 0($t7)          # $t5= value at $t7 = postscript[i], doc cac ki tu cua pscript
        beq $t5, 0, END_PROGRAM # if $t5 == null -> ket thuc doc pscript
        beq $t5, 44, READ_TRACK # if $t5 == ',' -> chuyen den READ_TRACK
        mul $t0, $t0, 10        # $t0 = $t0 * 10
        addi $t5, $t5, -48      # $t5 = $t5 - 48 -> $t5 = ma ASCII -48 = gia tri ki tu can tim
        add $t0, $t0, $t5       # $t0 = $t0 + $t5
        addi $t6, $t6, 1        # $t6 = $t6 +1 ->  tang so bit can dich chuyen len 1
        j READ_ROTATE           # quay lai doc tiep den khi nao gap ','
```

+ READ_TRACK: thanh ghi $t9 có giá trị 1 hoặc 0, cho phép Robot có thiết lập lưu vết hay không

```
READ_TRACK:
        add $a0, $zero, $t0     # Marsbot rotates $t0* and start running
        jal ROTATE

        addi $t6, $t6, 1        # $t6 = $t6 + 1 -> tang so bit can dich chuyen len 1
        add $t7, $a1, $t6       # $t7= $a1 + $t6 =  postscript[0] + i = address of postscript[i], dich bit
        lb $t9, 0($t7)          # #t9  (1 OR 0)
        addi $t9, $t9, -48      # $t9 = $t9 - 48 -> $t5 = ma ASCII -48 = gia tri ki tu can tim
        addi $t6, $t6, 1        # $t6 = $t6 + 1 -> tang so bit can dich chuyen len 1
```

+ READ_TIME: tương tự giống READ_TRACK.

```
READ_TIME:                      # doc thoi gian chuyen dong.
        addi $t6, $t6, 1        # $t6 = $t6 + 1 -> tang so bit can dich chuyen len 1
        add $t7, $a1, $t6       # $t7= $a1 + $t6 =  postscript[0] + i = address of postscript[i]
        lb $t5, 0($t7)          # $t5= value at $t7 = postscript[i], doc cac ki tu cua pscript
        beq $t5, 59, RUNNING    # if $t5 == ';' -> chuyen den cau truc moi
        mul $t1, $t1, 10        # $t1 = $t1 * 10
        addi $t5, $t5, -48      # $t5 = $t5 - 48 -> $t5 = ma ASCII -48 = gia tri ki tu can tim
        add $t1, $t1, $t5       # $t1 = $t1 + $t5
        j READ_TIME             # quay lai doc tiep den khi gap dau ','
```

+RUNNING: Sau khi đọc xong thời gian $t1, gán giá trị $a0 = $t1, gọi service sleeping với thời gian = giá trị $a0

-

```
RUNNING:
        addi $v0,$zero,32       # Keep running by sleeping in $t1 ms
        add $a0, $zero, $t1     # $a0 = $t1
        beq $t9, $zero, Khong_Cat # 1=cat | 0=khongcat
        jal UNTRACK             # keep old track
        jal TRACK               # draw new track
        j READ_NEXT_PHASE
Khong_Cat:
        jal UNTRACK             # keep old track
```

+READ_NEXT_PHASE: Sau khi đọc xong 3 phần tử, tiếp tục đọc 3 phần từ tiếp theo, cứ như vậy cho đến khi đọc hết xâu phần tử thì kết thúc chương trình

```
READ_NEXT_PHASE:
        syscall
        addi $t6, $t6, 1        # tang so bit can dich chuyen len 1, bo qua dau ';'
        j READ_POSTCRIPT        # quay lai doc tiep postscript
```

3. Mã nguồn:

```
.eqv HEADING 0xffff8010      # Integer: An angle between 0 and 359
.eqv MOVING 0xffff8050       # Boolean: whether or not to move
.eqv LEAVETRACK 0xffff8020  # Boolean (0 or non-0):
.eqv WHEREX 0xffff8030       # Integer: Current x-location of MarsBot
.eqv WHEREY 0xffff8040       # Integer: Current y-location of MarsBot
# Key matrix
.eqv OUT_ADRESS_HEXA_KEYBOARD 0xFFFF0014
.eqv IN_ADRESS_HEXA_KEYBOARD 0xFFFF0012

.data
# postscript-DCE => key 0
postscript1: .asciiz
"90,0,2000;180,0,3000;180,1,5790;80,1,500;70,1,500;60,1,500;50,1,500;40,1,500;30,1,5
00;20,1,500;10,1,500;0,1,500;350,1,500;340,1,500;330,1,500;320,1,500;310,1,500;300,
1,500;290,1,500;280,1,490;90,0,8000;270,1,500;260,1,500;250,1,500;240,1,500;230,1,5
00;220,1,500;210,1,500;200,1,500;190,1,500;180,1,500;170,1,500;160,1,500;150,1,500;
140,1,500;130,1,500;120,1,500;110,1,500;100,1,500;90,1,1000;90,0,5000;270,1,2000;0,
1,5800;90,1,2000;180,0,2900;270,1,2000;90,0,3000;"
# postscript-LOC => key 4
postscript2: .asciiz
"90,0,2000;180,0,3000;180,1,5790;90,1,3000;90,0,6000;270,1,500;280,1,500;290,1,500;
300,1,500;310,1,500;320,1,500;330,1,500;340,1,500;350,1,500;0,1,500;10,1,500;20,1,5
00;30,1,500;40,1,500;50,1,500;60,1,500;70,1,500;80,1,500;90,1,500;100,1,500;110,1,50
0;120,1,500;130,1,500;140,1,500;150,1,500;160,1,500;170,1,500;180,1,500;190,1,500;2
00,1,500;210,1,500;220,1,500;230,1,500;240,1,500;250,1,500;260,1,500;90,0,11000;27
0,1,1500;280,1,500;290,1,500;300,1,500;310,1,500;320,1,500;330,1,500;340,1,500;350,
1,500;360,1,500;0,1,500;10,1,500;20,1,500;30,1,500;40,1,500;50,1,500;60,1,500;70,1,5
00;80,1,500;90,1,1000;90,0,3000;"
# postscript-LÝ =>  key 8
postscript3: .asciiz
"90,0,2000;180,0,3000;180,1,5790;90,1,3000;90,0,5000;0,1,2895;315,1,4170;90,0,3000;
180,0,2890;45,1,4170;270,0,3000;45,1,1800;"
.text
# nguoi dung chon phim key-matrix
        li $t3, IN_ADRESS_HEXA_KEYBOARD
        li $t4, OUT_ADRESS_HEXA_KEYBOARD
polling:
```

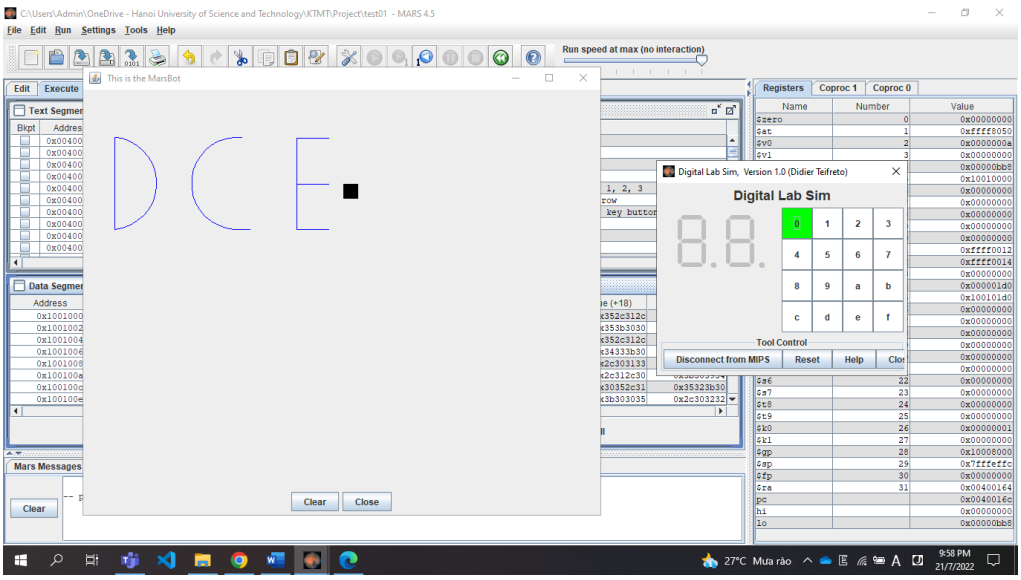
```
        li $t5, 0x01            # check row 1 with key 0, 1, 2, 3
        sb $t5, 0($t3)          # must reassign expected row
        lb $a0, 0($t4)  # $a0 = read scan code of key button
        bne $a0, 0x11, NOT_NUMPAD_0 # if not numpad 0
        la $a1, postscript1
        j START

        NOT_NUMPAD_0:
        li $t5, 0x02            # check row 2 with key 4, 5, 6, 7
        sb $t5, 0($t3)          # must reassign expected row
        lb $a0, 0($t4)  # read scan code of key button
        bne $a0, 0x12, NOT_NUMPAD_4  # if not numpad 4
        la $a1, postscript2
        j START

        NOT_NUMPAD_4:
        li $t5, 0X04                # check row 3 with key 8, 9, a, b
        sb $t5, 0($t3)          # must reassign expected row
        lb $a0, 0($t4)              # read scan code of key button
        bne $a0, 0x14, back_to_polling      # if not numpad 8
        la $a1, postscript3
        j START
back_to_polling: j polling              # neu numpad 0, 4, 8 khong duoc chon ->
quay lai doc tiep



#  Xu ly CNC MARSBOT
START:
        jal GO

READ_POSTCRIPT:
        addi $t0, $zero, 0      # $t0 luu gia tri rotate
        addi $t1, $zero, 0      # $t1 luu gia tri time

READ_ROTATE:                    # doc goc quay
                                # $t6 = i = 0 , i la chi so duyet mang xau
        add $t7, $a1, $t6       # $t7= $a1 + $t6 =  postscript[0] + i = address of
postscript[i], dich bit
        lb $t5, 0($t7)   # $t5= value at $t7 = postscript[i], doc cac ki tu cua pscript
        beq $t5, 0, END_PROGRAM # if $t5 == null -> ket thuc doc pscript
        beq $t5, 44, READ_TRACK # if $t5 == ',' -> chuyen den READ_TRACK
        mul $t0, $t0, 10        # $t0 = $t0 * 10
        addi $t5, $t5, -48      # $t5 = $t5 - 48 -> $t5 = ma ASCII -48 = gia tri ki tu can tim
```

```
        add $t0, $t0, $t5       # $t0 = $t0 + $t5
        addi $t6, $t6, 1        # $t6 = $t6 +1 ->  tang so bit can dich chuyen len 1
        j READ_ROTATE                   # quay lai doc tiep den khi nao gap ','

READ_TRACK:
        add $a0, $zero, $t0     # Marsbot rotates $t0* and start running
        jal ROTATE

        addi $t6, $t6, 1        # $t6 = $t6 + 1 -> tang so bit can dich chuyen len 1
        add $t7, $a1, $t6       # $t7= $a1 + $t6 =  postscript[0] + i = address of
postscript[i], dich bit
        lb $t9, 0($t7)          # #t9  (1 OR 0)
        addi $t9, $t9, -48      # $t9 = $t9 - 48 -> $t5 = ma ASCII -48 = gia tri ki tu can tim
        addi $t6, $t6, 1        # $t6 = $t6 + 1 -> tang so bit can dich chuyen len 1
READ_TIME:                      # doc thoi gian chuyen dong.
        addi $t6, $t6, 1        # $t6 = $t6 + 1 -> tang so bit can dich chuyen len 1
        add $t7, $a1, $t6       # $t7= $a1 + $t6 =  postscript[0] + i = address of
postscript[i]
        lb $t5, 0($t7)   # $t5= value at $t7 = postscript[i], doc cac ki tu cua pscript
        beq $t5, 59, RUNNING            # if $t5 == ';' -> chuyen den cau truc moi
        mul $t1, $t1, 10        # $t1 = $t1 * 10
        addi $t5, $t5, -48      # $t5 = $t5 - 48 -> $t5 = ma ASCII -48 = gia tri ki tu can tim
        add $t1, $t1, $t5       # $t1 = $t1 + $t5
        j READ_TIME             # quay lai doc tiep den khi gap dau ','

RUNNING:
        addi $v0,$zero,32       # Keep running by sleeping in $t1 ms
        add $a0, $zero, $t1     # $a0 = $t1
        beq $t9, $zero, Khong_Cat # 1=cat | 0=khongcat
        jal UNTRACK             # keep old track
        jal TRACK               # draw new track
        j READ_NEXT_PHASE
Khong_Cat:
        jal UNTRACK             # keep old track

READ_NEXT_PHASE:
        syscall
        addi $t6, $t6, 1        # tang so bit can dich chuyen len 1, bo qua dau ';'
        j READ_POSTCRIPT        # quay lai doc tiep postscript



GO:
        li $at, MOVING
```

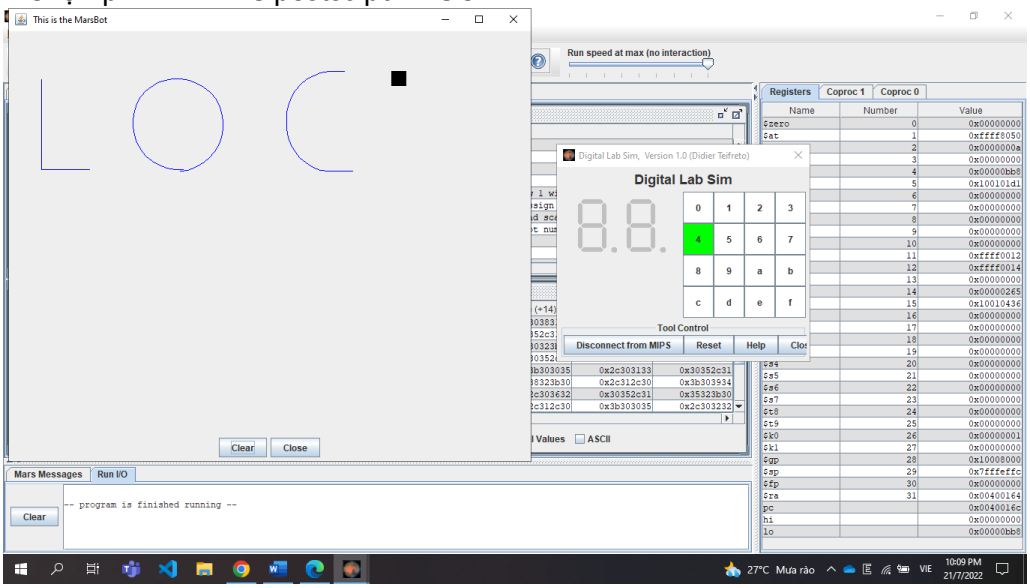
```
        addi $k0, $zero,1
        sb $k0, 0($at)
        jr $ra

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

END_PROGRAM:
        jal STOP
        li $v0, 10
        syscall
        j polling
```

4. Mô phỏng chương trình:

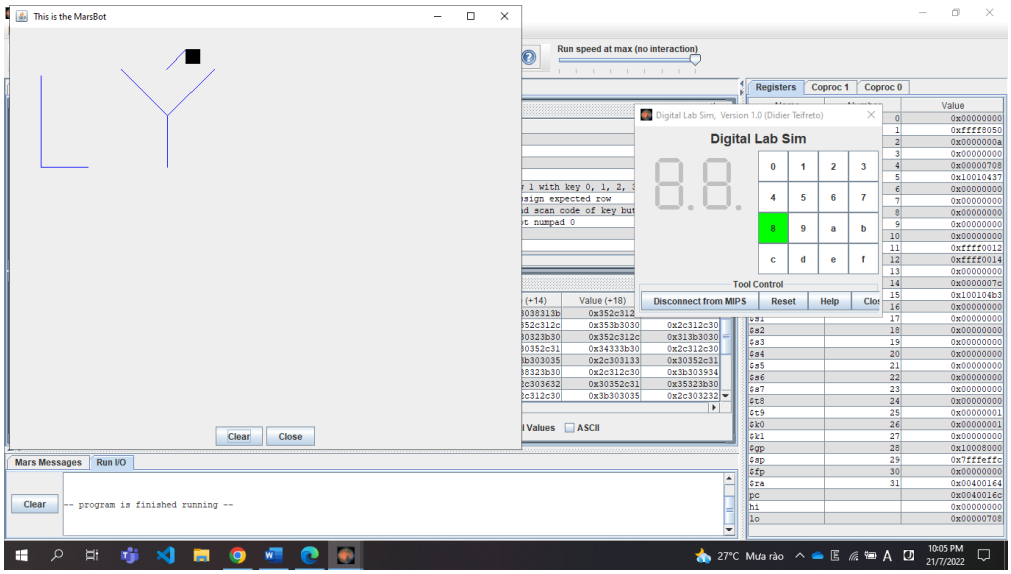
+ Chọn phím 0 -> Vẽ postscipt1: DCE



+ Chọn phím 4 -> Vẽ postscipt2: LOC



+ Chọn phím 8 -> Vẽ postscipt3: LÝ



Chủ đề 1: Curiosity Marsbot
Nguyễn Thị Phương Lý – 20194612

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.

+ Space: Lặp lại lệnh đã thực hiện trước đó

- 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, Space có được nhập hay không ?

+ Nếu Enter được nhập chuyển sang *Bước 3*;
+ Nếu Delete được nhập chuyển sang *Bước 4*;
+ Nếu Space được nhập chuyển sang *Bước 5*;
+ 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:

```
# 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
#---------------------------------------------------------------
--------
.data
     x_history:       .word 0 : 16    # = 16 for easier debugging
     y_history:       .word 0 : 16
     a_history:       .word 0 : 16
     l_history:       .word 4    # history length
     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"
#-----------------------------------------------------------------
--------
.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:
      lw    $t7, l_history              # l_history += 4 tang vung
nhÆ¡ de co the luu xya
      addi  $t7, $zero, 4               # to save 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          # a_history[1] = 0

      j     waitForKey

print_error: li $v0, 4
      la    $a0, INVALID_CODE
      syscall

print_current_code: li     $v0, 4
      la    $a0, current_code
      syscall

resetInput: jal strClear
      nop

waitForKey: lw $t5, 0($k1)              # $t5 = [$k1] = KEY_READY
```

```
      beq   $t5, $zero, waitForKey      # if $t5 == 0 -> Polling
      nop
      beq   $t5, $zero, waitForKey

readKey: lw      $t6, 0($k0)                 # $t6 = [$k0] =
KEY_CODE
      beq   $t6, 0x7f , resetInput      # $t6 == 'DEL' -> reset
input

      beq   $t6, 32, replay             # $t6 == " "  -> Replay
      nop

      beq   $t6, 0x0a, check_code             # $t6 =='\n'
      nop


check_code: lw   $s2, length                  # length != 3 ->
invalid cmd
      bne   $s2, 3, print_error

      la    $s3, MOVE_CODE
      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

      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*
      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)
      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 = the reverse
direction of alpha
      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]
      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]
      nop                         # -> turn or end
      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 print_current_code
#-------------------------------------------------------------
# saveHistory()
#-------------------------------------------------------------
saveHistory: addi $sp, $sp, 4         # backup: de khong bá»‹
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)
      addi  $sp, $sp, 4
      sw    $k0, 0($sp)

      li    $at, MOVING           # MOVING == 1 -> DI CHUYEN
      addi  $k0, $zero, 1
      sb    $k0, 0($at)

      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 drawing tail

      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)                 # to rotate robot

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

```
      addi  $sp, $sp, -4
ROTATE_end: jr $ra
#================================================================
========
# Procedure for string
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~
# strcmp()
# - input: $s3 = string to compare with 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: