HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

FINAL PROJECT REPORT

IT 3280 - THỰC HÀNH KIẾN TRÚC MÁY TÍNH

Course information

Course ID	Course title	Class ID
IT3280	Thực hành kiến trúc máy tính	147796

Student information

Student's full name	Class	Student ID
Đinh Ngọc Khánh Huyền	Việt Nhật 01 - K67	20225726
Đỗ Quang Huy	Việt Nhật 04 - K67	20225854

Instructor: **Đỗ Công Thuần**

HA NOI, 5/24

Mục lục

Task 1: Curiosity Marsbot

1. Giới thiệu vấn đề

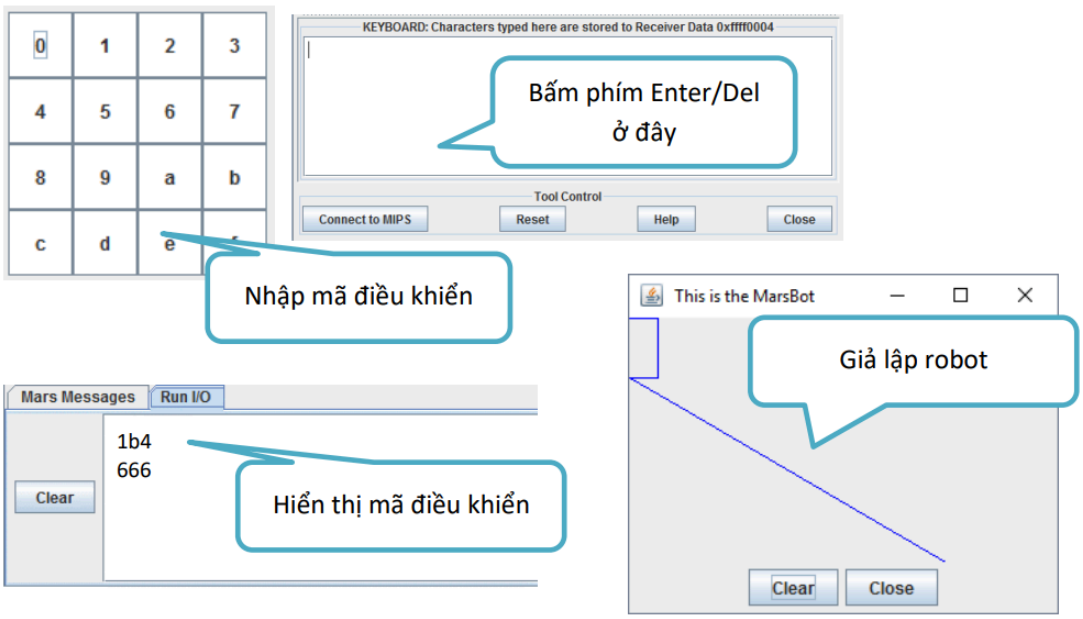
Xe tự hành Curioisity Marsbot chạy trên sao Hỏa, được vận hành từ xa bởi các lập trình viên trên Trái Đất. Bằng cách gửi đi các mã điều khiển từ một bàn phím ma trận, lập trình viên điều khiển quá trình di chuyển của Marbot như sau:

Mã điều khiển	**Ý nghĩa**
1b4	Marbot bắt đầu chuyển động.
c68	Marbot đứng im.
444	Rẽ trái 90 độ so với phương chuyển động gần nhất và giữ hướng mới.
666	Rẽ phải 90 độ so với phương chuyển động gần nhất và giữ hướng mới.
dad	Bắt đầu để lại vết trên đường.
cbc	Chấm dứt để lại vết trên đường.
999	Tự động quay trở lại theo lộ trình ngược lại. Không vẽ vết, không nhận mã khác cho tới khi kết thúc lộ trình ngược.

Khi khởi động chương trình thì Curioisity Marsbot sẽ tự động di chuyển theo một quỹ đạo (tuỳ chọn) rồi dừng lại chờ nhận mã điều khiển. Sau khi nhận mã điều khiển, Curioisity Marsbot sẽ không xử lý ngay, mà phải đợi lệnh kích hoạt mã từ bàn phím Keyboard & Display MMIO Simulator. Có 2 lệnh như vậy:

Kích hoạt mã	**Ý nghĩa**
Phím Enter	Kết thúc nhập mã và yêu cầu Marbot thực thi.
Phím Del	Xóa toàn bộ mã điều khiển đang nhập.

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.



2. Triển khai dự án

2.1 Ý tưởng chính

Xử lý ngắt ở Digital Lab Sim để đọc mã điều khiển từ bàn phím, lưu vào 1 mảng rồi so sánh với các mã lệnh đã định nghĩa để xác định mã điều khiển. Đọc lệnh từ bàn phím MMIO để xác định lệnh sẽ được kích hoạt theo cách nào. Sau đó tùy thuộc vào mã lệnh, chương trình thực hiện các hành động như di chuyển, dừng, xoay, hoặc vẽ dấu vết cho Marsbot.

2.2 Thuật toán chi tiết

Để chạy, chương trình phải kích hoạt chức năng ngắt của bàn phím ma trận 4x4 của Digital Lab.

Chương trình sẽ chạy trong vòng lặp vô hạn, chờ đợi nhập liệu từ bàn phím MMIO. Khi một nút trên Digital Lab được nhấn, một tín hiệu ngắt sẽ được kích hoạt và cho phép chương trình kiểm tra phím đã nhấn trên Digital Lab.

Chờ đợi nhập dữ liệu từ bàn phím MMIO. Khi một phím được nhấn, chương trình sẽ xác định lệnh và thực hiện:

- Enter: Chương trình sẽ thực thi dựa trên tham số control_code. Nếu mã lệnh không khả dụng, lỗi sẽ được in ra. Nếu không, Marsbot sẽ thực hiện hành động dựa trên control_code.

- Delete: Chạy hàm strClear để đặt lại tham số control_code.

Nếu một phím trên Digital Lab được nhấn, chương trình sẽ bị ngắt và chạy hàm getCode để lấy phím nhập. Phím sẽ được nối thêm vào control_code.

Trong quá trình chạy, mỗi lần Marsbot thực hiện một hành động, tọa độ và góc sẽ được lưu vào một mảng lịch sử. Cho phép theo dõi ngược lại hành động khi robot phải đi theo quỹ đạo ngược lại.

2.3 Các chương trình con

- Trong hàm main:
    + init: khởi tạo các giá trị, thiết lập trạng thái ban đầu.
    + printError: in ra error message
    + printCode: in câu lệnh
    + resetInput: xóa mã điều khiển để sẵn sàng cho câu lệnh tiếp theo
    + waitForKey: chờ đợi phím được nhấn ở Digital Lab Sim
    + readKey: đọc phím được nhấn
    + checkCode: kiểm tra xem mã điều khiển có hợp lệ hay không.
    + go, stop, track, untrack, turnRight, turnLeft, goBackward: thực hiện các lệnh tương ứng và in mã điều khiển.
    + getCode: quét xem có phím nào được nhấn
    + getCodeInChar: đổi sang mã ASCII
    + storeCode: lưu mã điều khiển vào mảng
- Các hàm điều khiển Marsbot:
    + saveHistory: lưu trạng thái hiện tại (các thông số x, y, góc quay)
    + GO, STOP: Điều khiển Marsbot di chuyển hoặc dừng lại và lưu trạng thái trong biến is_going.
    + ROTATE: Điều khiển Marsbot xoay theo góc được lưu trữ trong biến a_current.
    + TRACK, UNTRACK: Điều khiển Marsbot để bắt đầu hoặc ngừng lưu vết và lưu trạng thái trong biến is_tracking.
- Các hàm làm việc với string:

+ strcmp: so sánh chuỗi nhận được với mã điều khiển hiện tại.
+ strClear: xóa mã điều khiển hiện tại.
+ strCpy: Sao chép từ chuỗi hiện tại sang chuỗi trước đó.

2.4 Giải thích mã

- Khởi tạo các giá trị, trạng thái ban đầu

```
.data
        # CODE
        MOVE_CODE: .asciiz "1b4" # command code
        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"

        error_msg:  .asciiz "Invalid command code: "
```

```

    # HISTORY
    # save history before changing direction

    x_history: .word 0 : 16 # = 16 for easier debugging
    y_history: .word 0 : 16

    # For rotation
    a_history: .word 0 : 16
    l_history: .word 4              # history length

    a_current: .word 0              # current alpha

    is_going: .word 0
    is_tracking: .word 0

    # Code properties
    control_code: .space 8          # input command code
    code_length: .word 0            # input command length

```

```
82  # run at start of program
83  init:
84          # increase length history by 4
85          # (as saving current state: x = 0; y = 0; a = 90)
86
87          lw $t7, l_history # l_history += 4
88          addi $t7, $zero, 4
89          sw $t7, l_history
90
91          li $t7, 90
92          sw $t7, a_current # a_current = 90 -> head to the right
93          jal ROTATE
94          nop
95
96          sw $t7, a_history # a_history[0] = 90
97
98          j waitForKey
```

- Kiểm tra xem có giá trị nhập vào từ bàn phím hay không? Đọc mã ASCII của phím vừa được nhấn

```
116 # Take input
117 waitForKey:
118         lw $t5, 0($k1)    # $t5 = [$k1] = KEY_READY
119         beq $t5, $zero, waitForKey  # if $t5 == 0 -> Polling
120         nop
121         beq $t5, $zero, waitForKey
122
123 readKey:
124         lw $t6, 0($k0)    # $t6 = [$k0] = KEY_CODE
125         # if $t6 == 'DEL' -> reset input
126         beq $t6, 0x8, resetInput
127
128         # if $t6 != 'ENTER' -> Polling
129         bne $t6, 0x0a, waitForKey
130         nop
131         bne $t6, 0x0a, waitForKey
```

- Đọc các phím nhấn từ Digital Lab Sim

```
653 getCode:
654         li $t1, IN_ADRESS_HEXA_KEYBOARD
655         li $t2, OUT_ADRESS_HEXA_KEYBOARD
656
657         # scan row 1
658         li $t3, 0x81
659         sb $t3, 0($t1)
660         lbu $a0, 0($t2)
661         bnez $a0, getCodeInChar
662
663         # scan row 2
664         li $t3, 0x82
665         sb $t3, 0($t1)
666         lbu $a0, 0($t2)
667         bnez $a0, getCodeInChar
```

```
681 getCodeInChar:
682         beq $a0, KEY_0, case_0
683         beq $a0, KEY_1, case_1
684         beq $a0, KEY_2, case_2
685         beq $a0, KEY_3, case_3
686         beq $a0, KEY_4, case_4
```

```
699 case_0:
700         li $s0, '0'  # $s0 store code in char type
701         j storeCode
702 case_1:
703         li $s0, '1'
704         j storeCode
705 case_2:
706         li $s0, '2'
```

- Lưu mã ASCII của các phím được nhấn vào mảng control_code

```
748 storeCode:
749         la  $s1, control_code
750         la $s2, code_length
751         lw $s3, 0($s2)    # $s3 = strlen(control_code)
752         addi $t4, $t4, -1    # $t4 = i
753 
754 storeCodeLoop:
755         addi $t4, $t4, 1
756         bne $t4, $s3, storeCodeLoop
757         add $s1, $s1, $t4  # $s1 = control_code + i
758         sb $s0, 0($s1)    # control_code[i] = $s0
759 
760         addi $s0, $zero, '\n'  # add '\n' character to end of string
761         addi $s1, $s1, 1
762         sb $s0, 0($s1)
763 
764         addi $s3, $s3, 1
765         sw $s3, 0($s2)    # update code_length
766 
```

- Kiểm tra xem control_code có khớp với các lệnh đã định nghĩa trước hay không?

```
133 checkCode:
134         lw $s2, code_length   # code_length != 3 -> invalid code
135         bne $s2, 3, printError
136 
137         la $s3, MOVE_CODE
138         jal strcmp
139         beq $t0, 1, go
140 
141         la $s3, STOP_CODE
142         jal strcmp
143         beq $t0, 1, stop
144 
145         la $s3, TURN_LEFT_CODE
146         jal strcmp
147         beq $t0, 1, turnLeft
148 
149         la $s3, TURN_RIGHT_CODE
150         jal strcmp
151         beq $t0, 1, turnRight
```

- Hàm so sánh mã điều khiển được nhập vào và các mã đã được định nghĩa sẵn

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

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

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

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

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

        j strcmp_end

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

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

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

        jr $ra
```

- Sao chép lại code, thực hiện các thao tác cần thiết, in ra mã điều khiển

```
169 # Perform function and print code
170 go:
171         jal strCpy      #chuan bi hoac sao chep chuoi can thiet
172         jal GO          #thuc hien cac thao tac can thiet
173         j printCode     #in ra ma dieu khien
174
175 stop:
176         jal strCpy
177         jal STOP
178         j printCode
179
180 track:
181         jal strCpy
182         jal TRACK
183         j printCode
184
```

- In ra error message hoặc mã điều khiển, sau đó reset lại Input

```
100 # Function: print error to console
101 printError:
102         li $v0, 4
103         la $a0, error_msg
104         syscall
105
106 printCode:
107         li $v0, 4
108         la $a0, control_code
109         syscall
110         j resetInput
111
112 resetInput:
113         jal strClear
114         nop
115
```

- Thủ tục để điều khiển Marbot

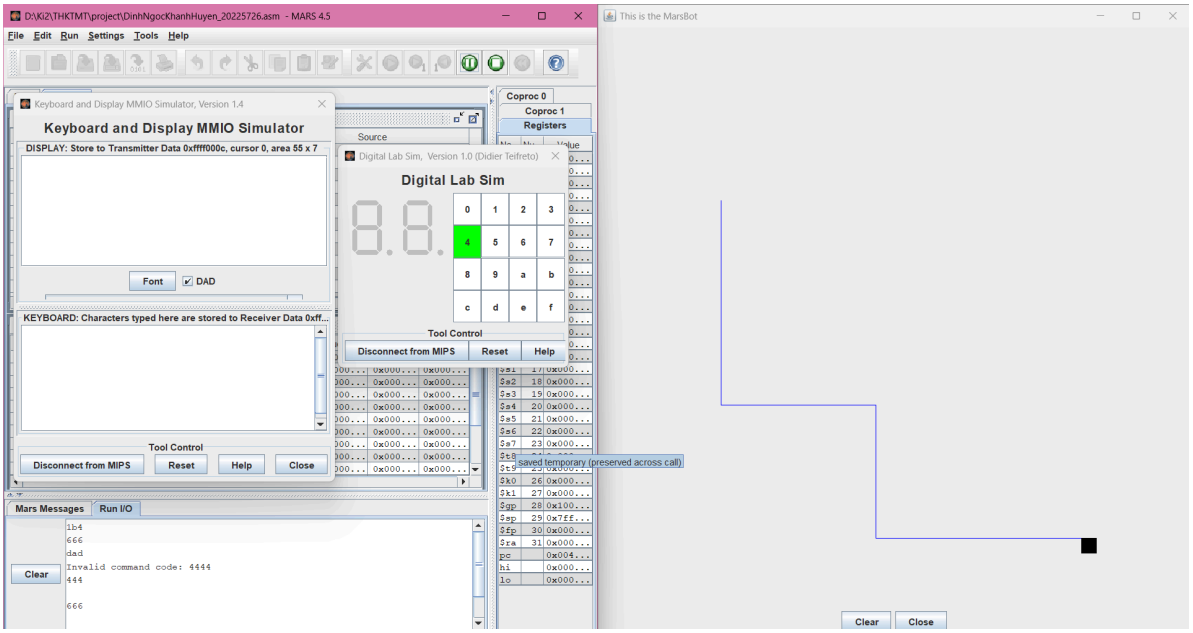
```
GO:
        addi $sp, $sp, 4    # backup
        sw $at, 0($sp)
        addi $sp, $sp, 4
        sw $k0, 0($sp)

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

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

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

GO_end:
        jr $ra

```

- Hàm saveHistory phục vụ cho mã kích hoạt 999

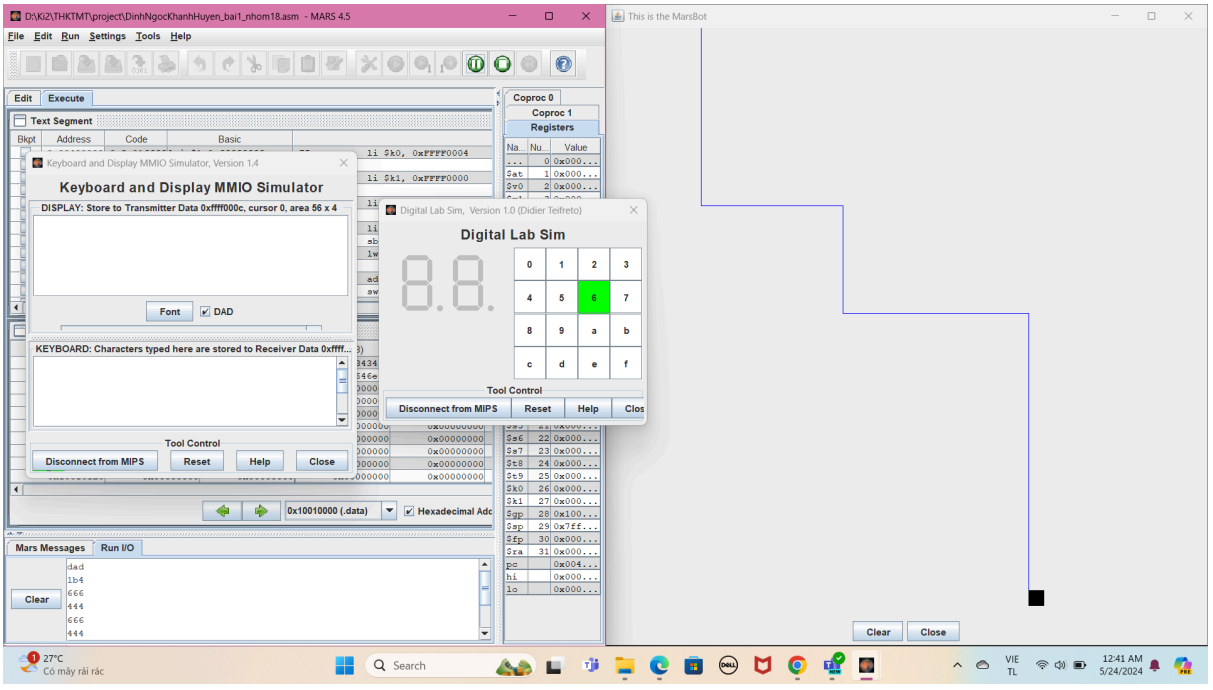
```
saveHistory:
        addi $sp, $sp, 4    # backup
        sw $t1, 0($sp)
        addi $sp, $sp, 4
        sw $t2, 0($sp)
        addi $sp, $sp, 4
        sw $t3, 0($sp)
        addi $sp, $sp, 4
        sw $t4, 0($sp)
        addi $sp, $sp, 4
        sw $s1, 0($sp)
        addi $sp, $sp, 4
      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

3. Trình bày dự án

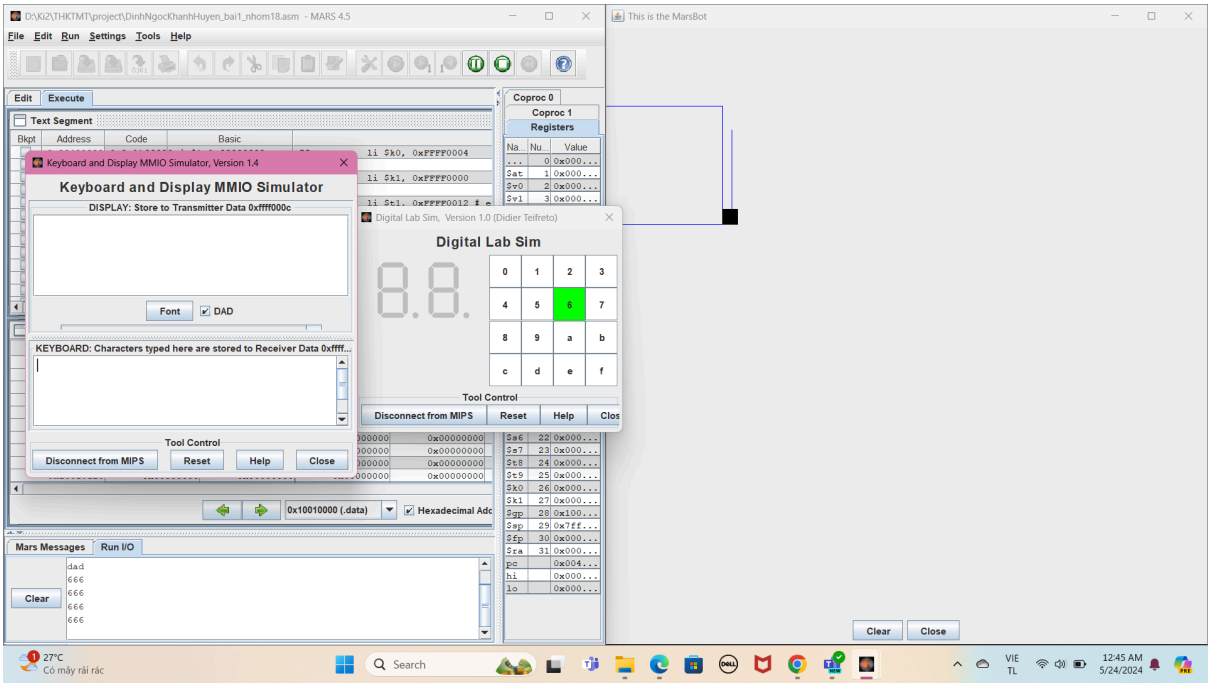
- Demo chương trình:



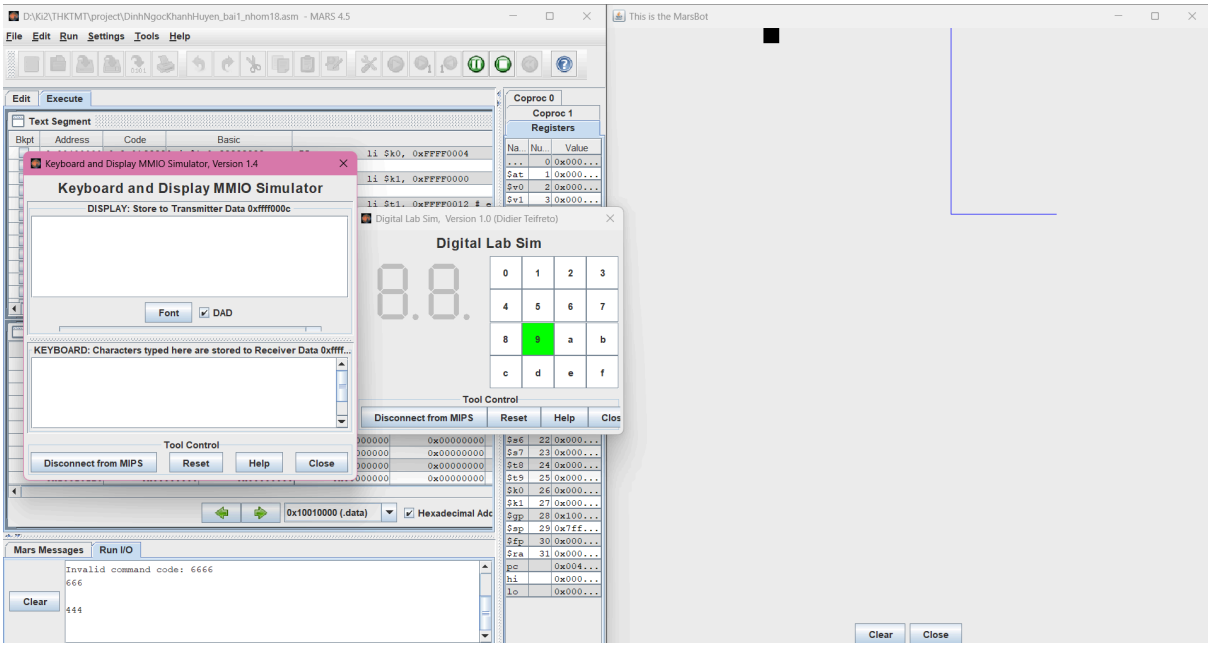
Chương trình sau khi thực hiện các mã điều khiển: 1b4, 666, dad, 444, 666.
Khi nhập mã lệnh sai, chương trình sẽ in ra mã lệnh sai đó cùng dòng Invalid command value:



Chương trình sau khi thực hiện các mã điều khiển: dad, 1b4, 666, 444, 666, 444, 666



Chương trình sau khi thực hiện các mã điều khiển: 1b4, 666, c68, 1b4, dad, 666, 666, 666, 666



Chương trình sau khi thực hiện các mã điều khiển: 1b4, dad, 666, 444, 999

Task 2: Moving a ball on the BITMAP Display

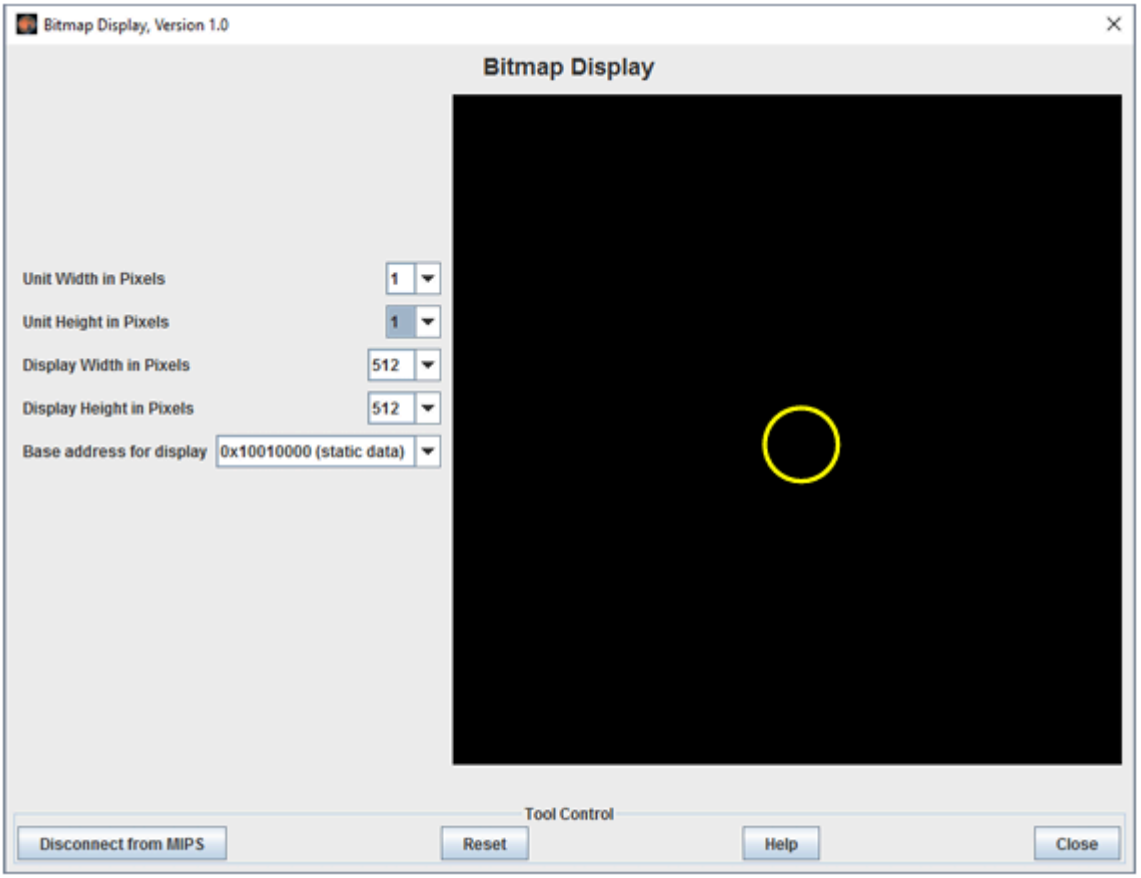
1. Giới thiệu vấn đề

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

Yêu cầu:

- Thiết lập màn hình ở kích thước 512x512. Kích thước pixel 1x1.

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



2. Triển khai dự án

2.1 Ý tưởng chính cho vấn đề

- Sử dụng MMIO Keyboard và Display Simulator để phát hiện phím được nhấn.

- Sự di chuyển của quả bóng có thể được biểu diễn bằng cách vẽ một hình tròn ở vị trí mới và vẽ một hình tròn khác cùng màu với màn hình ở vị trí trước đó.
- Để làm một đối tượng di chuyển thì chúng ta sẽ xóa đối tượng ở vị trí cũ và vẽ đối tượng ở vị trí mới. Để xóa đối tượng chúng ta chỉ cần vẽ đối tượng đó với màu là màu nền.

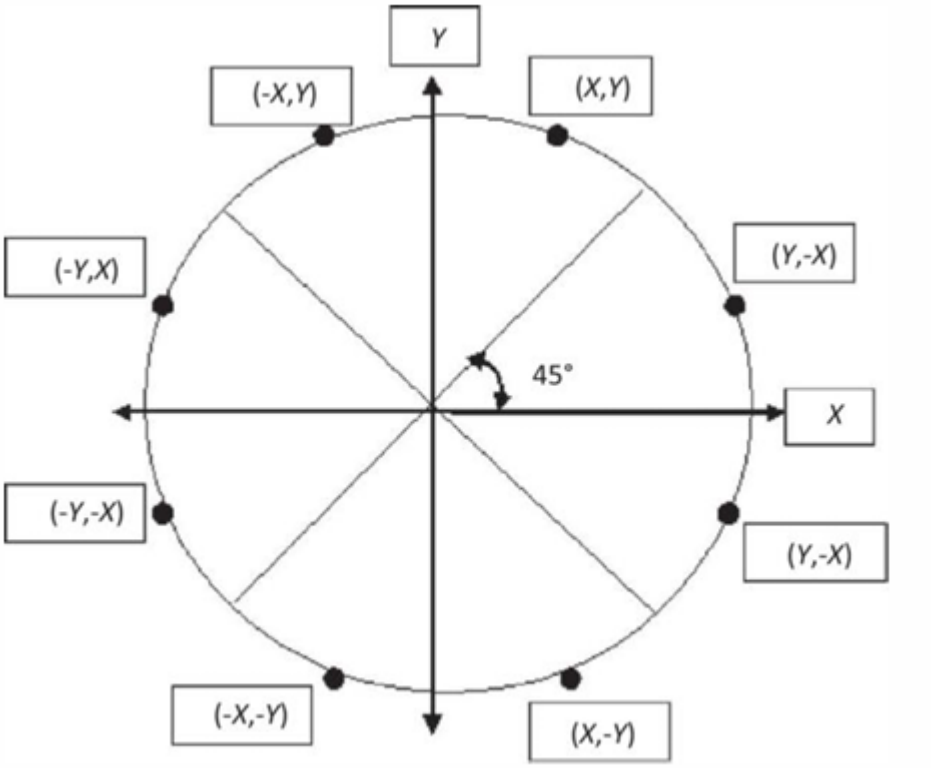
2.2 Thuật toán chi tiết

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

- Tạo một con trỏ để lưu dữ liệu về tất cả các điểm trong hình tròn vào mảng hình tròn.

Lặp giá trị tọa độ x từ 0 đến R, tính giá trị tọa độ y theo công thức $y = \sqrt{R^2 - x^2}$.

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



Kiểm tra nếu quả bóng chạm vào cạnh của màn hình. Nếu tọa độ của tâm cộng R lớn hơn giá trị giới hạn của màn hình (1 cho giới hạn dưới và 511 cho giới hạn trên), quả bóng sẽ di chuyển theo hướng ngược lại.
Vẽ một hình tròn mới: Xóa hình tròn cũ bằng cách thay đổi màu của hình tròn trước đó thành màu đen. Thay đổi màu thành màu vàng, cập nhật giá trị mới cho các điểm trong hình tròn và vẽ lại.

2.3 Các chương trình con

initial_declare

- Khởi tạo các giá trị ban đầu cho tọa độ, vận tốc, bán kính và thời gian khung hình.
- Gọi hàm circle_data để tạo dữ liệu cho hình tròn.

input

- Kiểm tra xem có dữ liệu nhập từ bàn phím hay không.
- Nếu có, gọi hàm direction_change để thay đổi hướng di chuyển dựa trên phím nhập.

edge_check

- Kiểm tra xem hình tròn có chạm vào các cạnh của màn hình không.
- Gọi các nhãn right, left, down, up để kiểm tra từng cạnh.

right

- Kiểm tra cạnh phải của màn hình, nếu không chạm, chuyển sang kiểm tra cạnh trái (left).
- Nếu chạm, gọi check_right.

left

- Kiểm tra cạnh trái của màn hình, nếu không chạm, chuyển sang kiểm tra cạnh dưới (down).
- Nếu chạm, gọi check_left.

down

- Kiểm tra cạnh dưới của màn hình, nếu không chạm, chuyển sang kiểm tra cạnh trên (up).
- Nếu chạm, gọi check_down.

up

- Kiểm tra cạnh trên của màn hình, nếu không chạm, gọi move_circle.
- Nếu chạm, gọi check_up.

move_circle

- Xóa hình tròn cũ bằng cách vẽ lại nó với màu đen.
- Cập nhật tọa độ trung tâm của hình tròn theo vận tốc hiện tại.
- Vẽ hình tròn mới với màu vàng.
- Quay lại nhãn loop.

loop

- Đưa chương trình vào chế độ ngủ trong một khoảng thời gian ngắn (50ms).
- Quay lại nhãn input để làm mới chu kỳ.

circle_data

- Tạo dữ liệu cho hình tròn (tọa độ của các pixel tạo thành hình tròn).
- Lưu địa chỉ cuối cùng của mảng dữ liệu circle vào circle_end.

pixel_data_loop

- Tạo dữ liệu pixel cho hình tròn dựa trên bán kính.
- Gọi hàm root để tính căn bậc hai của giá trị cần thiết.
- Gọi hàm pixel_save để lưu tọa độ pixel vào mảng circle.
- Lặp lại cho đến khi hoàn thành.

root

- Tìm căn bậc hai của giá trị trong thanh ghi $a2.(gần đúng)
- Sử dụng phương pháp bisection để tìm căn bậc hai.

pixel_save

- Lưu tọa độ pixel vào mảng circle.

direction_change

- Kiểm tra mã phím nhập và thay đổi vận tốc của hình tròn tương ứng.
- Nếu phím d được nhấn, di chuyển sang phải.
- Nếu phím a được nhấn, di chuyển sang trái.
- Nếu phím s được nhấn, di chuyển xuống dưới.
- Nếu phím w được nhấn, di chuyển lên trên.
- Nếu phím x được nhấn, tăng thời gian khung hình.
- Nếu phím z được nhấn, giảm thời gian khung hình (nếu thời gian > 0).

check_right

- Kiểm tra cạnh phải của màn hình.
- Nếu chạm, gọi reverse_direction.
- Nếu không chạm, gọi move_circle.

check_left

- Kiểm tra cạnh trái của màn hình.
- Nếu chạm, gọi reverse_direction.
- Nếu không chạm, gọi move_circle.

check_down

- Kiểm tra cạnh dưới của màn hình.
- Nếu chạm, gọi reverse_direction.
- Nếu không chạm, gọi move_circle.

check_up

- Kiểm tra cạnh trên của màn hình.
- Nếu chạm, gọi reverse_direction.
- Nếu không chạm, gọi move_circle.

reverse_direction

- Đảo ngược hướng di chuyển của hình tròn.
- Quay lại nhãn move_circle.

draw_circle

- Vẽ hình tròn trên màn hình.
- Sử dụng dữ liệu từ mảng circle để vẽ từng pixel của hình tròn.
- Lặp lại cho đến khi vẽ xong tất cả các pixel.

pixel_draw

- Vẽ một pixel lên màn hình tại tọa độ cụ thể.
- Tính toán địa chỉ pixel trên màn hình và lưu giá trị màu vào địa chỉ đó.

2.4 Giải thích mã

bai2(project)

```
1   .eqv KEY_CODE 0xFFFF0004
2   .eqv KEY_READY 0xFFFF0000
3   .eqv YELLOW 0x00FFFF00
4   .eqv SCREEN_MONITOR 0x10010000
5
6   .data
7   circle_end:     .word   1       # Con trỏ trỏ đến cuối của chu kỳ
8   circle:         .word           # Con trỏ trỏ đến mảng lưu dữ liệu của chu kỳ
9   .text
10  initial_declare:
11          addi    $s0, $0, 255    # x = 255
12          addi    $s1, $0, 255    # y = 255
13          addi    $s2, $0, 0      # dx = 0
14          add     $s3, $0, $0     # dy = 0
15          addi    $s4, $0, 20     # r = 20
16          addi    $a0, $0, 50     # t = 50ms/frame
17          jal     circle_data
18
19  input:
20          li      $k0, KEY_READY  # Kiểm tra xem có dữ liệu nhập không
21          lw      $t0, 0($k0)
22          bne     $t0, 1, edge_check
23          jal     direction_change
24
25  # Kiểm tra xem hình tròn có chạm vào cạnh không
26  edge_check:
```

-Đoạn mã này được sử dụng để thiết lập giá trị ban đầu cho một số biến:
$s0 là tọa độ x của tâm, $s1 là tọa độ y của tâm, thiết lập ban đầu là tâm của màn hình (255, 255). $s2 (dx), $s3 (dy) là khoảng cách di chuyển theo trục x và trục y (ban đầu bằng 0). $s4 là bán kính của hình tròn, $a0 là thời gian chờ.

```
19  input:
20          li      $k0, KEY_READY  # Kiểm tra xem có dữ liệu nhập không
21          lw      $t0, 0($k0)
22          bne     $t0, 1, edge_check
23          jal     direction_change
24
```

Kiểm tra đầu vào từ bàn phím. Nếu không có phím được nhấn, chương trình sẽ chuyển đến phần ‘edge_check’ để kiểm tra xem quả bóng có chạm vào các cạnh không.

```
edge_check:

right:
        bne     $s2, 1, left # Nếu bóng di chuyển sang trái, không kiểm tra cạnh phải, kiểm tra cạnh trái
        j       check_right

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

down:
        bne     $s3, 1, up
        j       check_down

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

check_right:
        add     $t0, $s0, $s4   # Đặt $t0 sang phải của hình tròn (trung tâm + R)
        beq     $t0, 511, reverse_direction     # Đảo ngược hướng nếu cạnh chạm vào viền
        j       move_circle     # Quay lại nếu không

check_left:
        sub     $t0, $s0, $s4   # Đặt $t0 sang trái của hình tròn
        beq     $t0, 0, reverse_direction       # Đảo ngược hướng nếu cạnh chạm vào viền
        j       move_circle     # Quay lại nếu không

check_down:
        add     $t0, $s1, $s4   # Đặt $t0 xuống dưới của hình tròn
        beq     $t0, 511, reverse_direction     # Đảo ngược hướng nếu cạnh chạm vào viền
        j       move_circle     # Quay lại nếu không

check_up:
        sub     $t0, $s1, $s4   # Đặt $t0 lên trên của hình tròn
        beq     $t0, 0, reverse_direction       # Đảo ngược hướng nếu cạnh chạm vào viền
        j       move_circle

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

-Đầu tiên, kiểm tra hướng di chuyển. Nếu quả bóng di chuyển sang trái, chỉ kiểm tra với cạnh trái; nếu quả bóng di chuyển sang phải, chỉ kiểm tra với cạnh phải. Tương tự với trường hợp di chuyển lên và xuống.

-Trong hàm kiểm tra (check_right, check_left, check_up, check_down), tạo một biến tạm thời để lưu giá trị của điểm xa nhất bên phải của hình tròn (tọa độ tâm + bán kính) để kiểm tra.

```
move_circle:
        add     $s5, $0, $0     # Đặt màu thành đen
        jal     draw_circle     # Xóa hình tròn cũ

        add     $s0, $s0, $s2   # Cập nhật giá trị mới cho điểm trung tâm của chu kỳ
        add     $s1, $s1, $s3
        li      $s5, YELLOW     # Đặt màu thành vàng
        jal     draw_circle     # Vẽ hình tròn mới

```

-Mã để di chuyển hình tròn: thiết lập màu thành đen và vẽ một hình tròn. Tiếp theo, cập nhật tọa độ tâm mới, đổi màu thành vàng và vẽ lại hình tròn mới.

```
pixel_data_loop:
        bgt     $s7, $s4, data_end
        mul     $t0, $s7, $s7   # $t0 = px^2
        sub     $a2, $a3, $t0   # $a2 = r^2 - px^2 = py^2
        jal     root            # $a2 = py

        add     $a1, $0, $s7    # $a1 = px
        add     $s6, $0, $0     # Sau khi lưu (px, py), (-px, py), (-px, -py), (px, -py), hoán đổi px và py, sau đó lưu (-py, px), (py, px), (py, -px), (-py, -px)

```

```
symmetric:
        beq     $s6, 2, finish
        jal     pixel_save      # px, py >= 0
        sub     $a1, $0, $a1
        jal     pixel_save      # px <= 0, py >= 0
        sub     $a2, $0, $a2
        jal     pixel_save      # px, py <= 0
        sub     $a1, $0, $a1
        jal     pixel_save      # px >= 0, py <= 0

        add     $t0, $0, $a1    # Hoán đổi px và -py
        add     $a1, $0, $a2
        add     $a2, $0, $t0
        addi    $s6, $s6, 1
        j       symmetric

finish:
        addi    $s7, $s7, 1 #px = px + 1
        j       pixel_data_loop

data_end:
        la      $t0, circle_end
        sw      $s5, 0($t0)     # Lưu địa chỉ cuối của mảng "circle"
        lw      $ra, 0($sp)
        addi    $sp, $sp, 4
        jr      $ra
```

-Đoạn này dùng để lưu dữ liệu của tất cả các điểm trong một hình tròn. Đầu tiên, gán $s7 (px) = 0 để lặp từ 0 đến R.

-Trong khối pixel_data_loop: tính toán $a2 (py) bằng khối root. Lưu 7 điểm khác bằng cách đổi dấu và hoán đổi giá trị của px và py vào mảng hình tròn. Việc lưu kết thúc khi px tăng lớn hơn R.

```
pixel_save:
        sw      $a1, 0($s5)     # Lưu px trong mảng "circle"
        sw      $a2, 4($s5)     # Lưu py trong mảng "circle"
        addi    $s5, $s5, 8     # Di chuyển con trỏ đến khối null
        jr      $ra
```

-Hàm để lưu tọa độ pixel của x và y .

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

case_d:
        bne     $t0, 'd', case_a
        addi    $s2, $0, 1      # dx = 1
        add     $s3, $0, $0     # dy = 0
        jr      $ra

case_a:
        bne     $t0, 'a', case_s
        addi    $s2, $0, -1     # dx = -1
        add     $s3, $0, $0     # dy = 0
        jr      $ra

case_s:
        bne     $t0, 's', case_w
        add     $s2, $0, $0     # dx = 0
        addi    $s3, $0, 1      # dy = 1
        jr      $ra
```

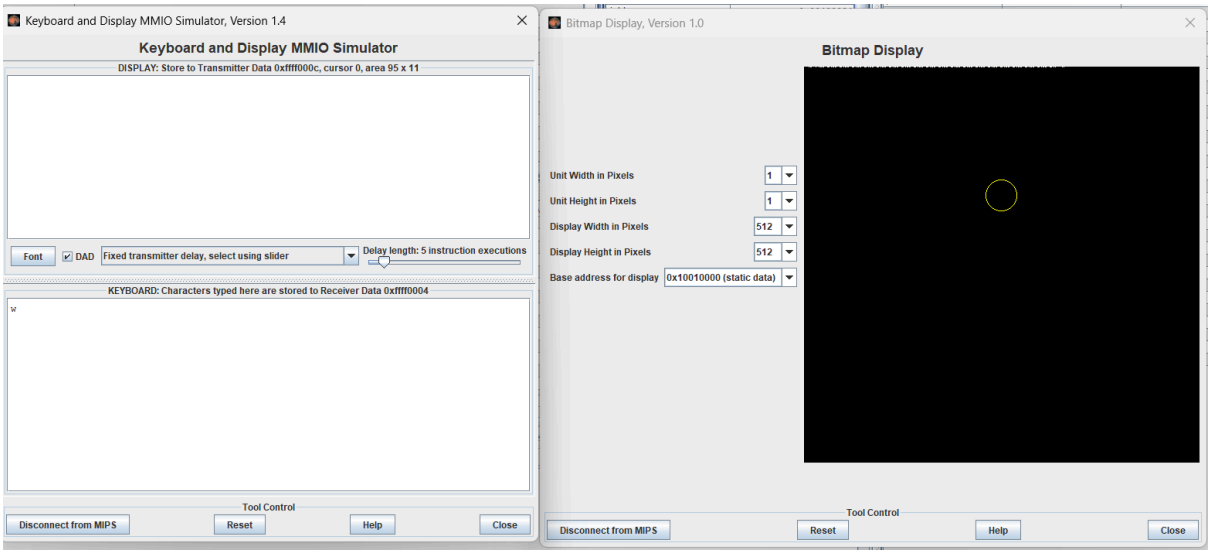
```

case_w:
        bne     $t0, 'w', case_x
        add     $s2, $0, $0     # dx = 0
        addi    $s3, $0, -1     # dy = -1
        jr      $ra

case_x:
        bne     $t0, 'x', case_z
        addi    $a0, $a0, 10    # t += 10
        jr      $ra

case_z:
        bne     $t0, 'z', default
        beq     $a0, 0, default # Chỉ giảm t khi t >= 0
        addi    $a0, $a0, -10   # t -= 10

default:
        jr      $ra

```

-Kiểm tra phím được nhấn từ bàn phím để cập nhật chuyển động của hình tròn bằng cách cập nhật khoảng cách di chuyển $s2 (dx), $s3 (dy). Nếu tùy chọn là ‘x’ hoặc ‘z’, cập nhật giá trị thời gian ngủ.

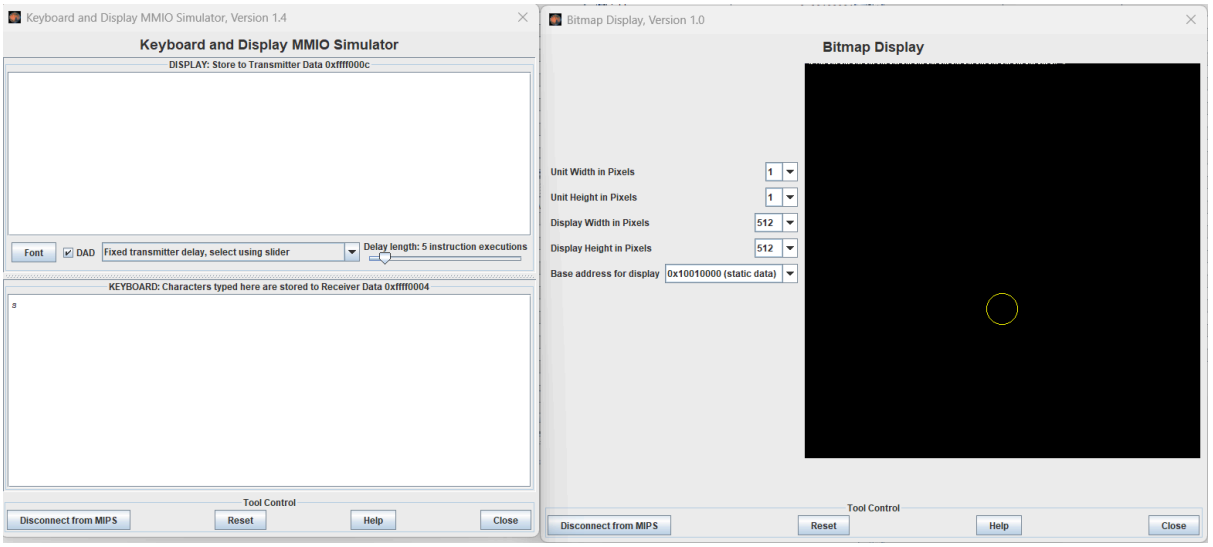
```
206 draw_circle:
207         addi    $sp, $sp, -4    # Luu $ra
208         sw      $ra, 0($sp)
209         la      $s6, circle_end
210         lw      $s7, 0($s6)     # $s7 trở thành địa chỉ cuối của mảng "circle"
211         la      $s6, circle     # $s6 trở thành con trỏ đến mảng "circle"
212
213 draw_loop:
214         beq     $s6, $s7, draw_end      # Dừng lại khi $s6 = $s7
215         lw      $a1, 0($s6)             # Lấy px
216         lw      $a2, 4($s6)             # Lấy py
217         jal     pixel_draw
218         addi    $s6, $s6, 8             # Đến pixel tiếp theo
219         j       draw_loop
220
221 draw_end:
222         lw      $ra, 0($sp)
223         addi    $sp, $sp, 4
224         jr      $ra
225
226 pixel_draw:
227         li      $t0, SCREEN_MONITOR
228         add     $t1, $s0, $a1           # x cuối cùng (fx) = x + px
229         add     $t2, $s1, $a2           # fy = y + py
230         sll     $t2, $t2, 9             # $t2 = fy * 512
231         add     $t2, $t2, $t1           # $t2 += fx
232         sll     $t2, $t2, 2             # $t2 *= 4
233         add     $t0, $t0, $t2
234         sw      $s5, 0($t0)
235         jr      $ra
236
```

-Lưu địa chỉ bắt đầu của mảng hình tròn vào $s6 và địa chỉ kết thúc của mảng hình tròn vào $s7. Chuyển đến khối ‘pixel_draw’.
- Lặp lại cho tất cả các điểm, lấy 2 biến tạm thời $a1 để lưu px, $a2 để lưu py. Tính toán vị trí của điểm ảnh bằng 2 tọa độ px, py. Vẽ điểm ảnh này với màu sắc.

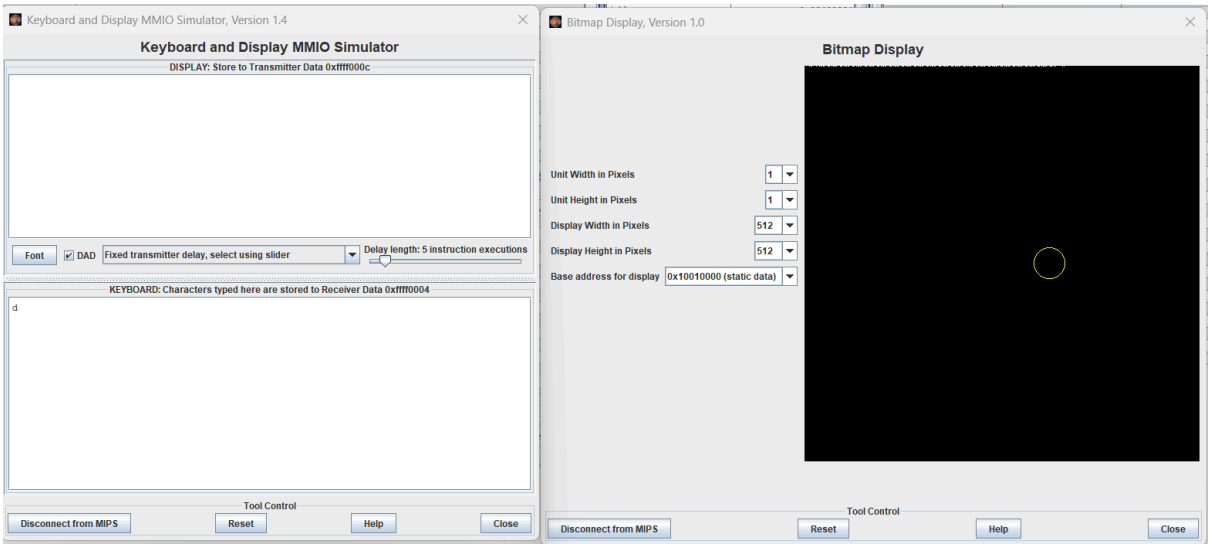
3.Trình bày dự án



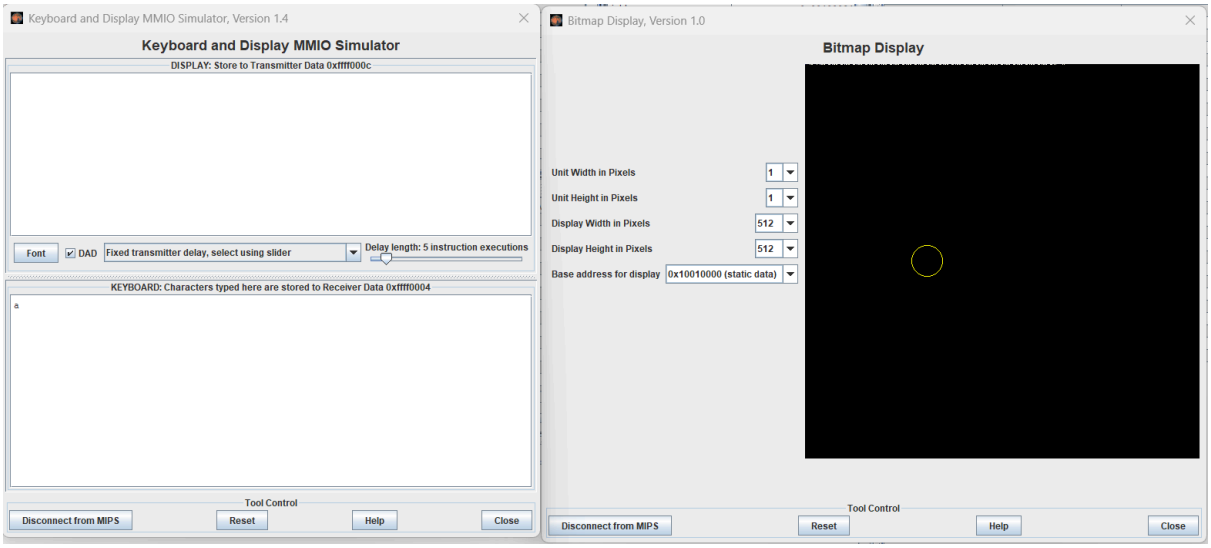
-Đi lên



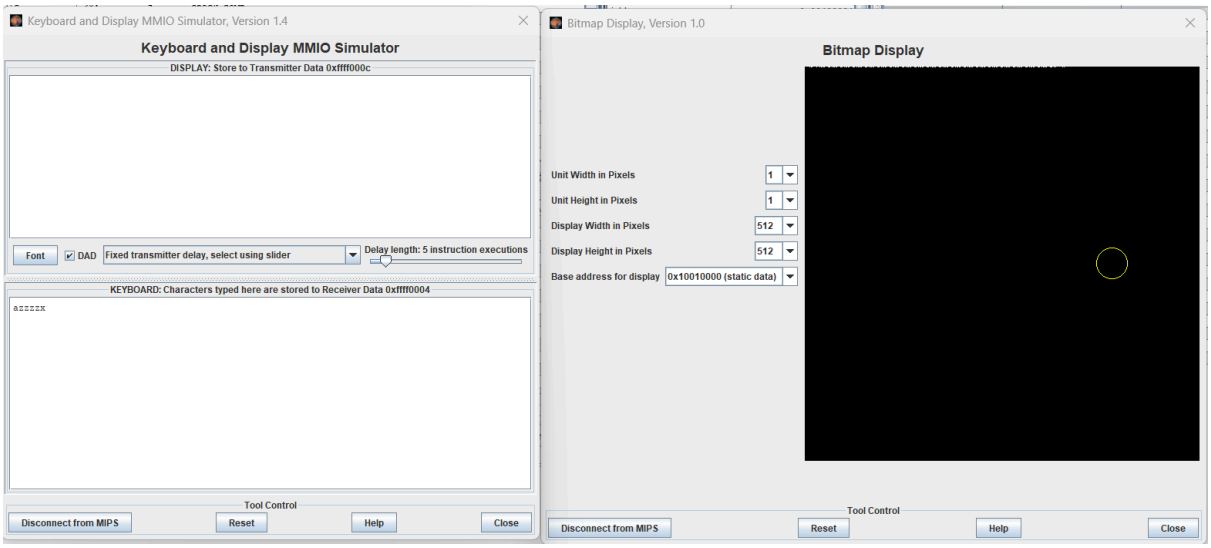
-Đi xuống



-Sang trái



-Sang phải



-

Quả bóng nảy sang hướng ngược lại khi chạm vào các cạnh.

Source Code

Task 1: Curiosity Marsbot

```
# eqv for 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        # Integer: An angle between 0 and 359
                               # 0 : North (up)
                               # 90: East (right)
                               # 180: South (down)
                               # 270: West (left)
.eqv MOVING 0xffff8050                 # Boolean: whether or not to move
.eqv LEAVETRACK 0xffff8020             # Boolean: whether or not to leave a track
.eqv WHEREX 0xffff8030                 # Integer: Current x-location of MarsBot
.eqv WHEREY 0xffff8040                 # Integer: Current y-location of MarsBot

#-------------------------------------------------------------------------------
.data
        # CODE
        MOVE_CODE: .asciiz "1b4" # command code
        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"
```

```
        error_msg:  .asciiz "Invalid command code: "


        # HISTORY
        # save history before changing direction

        x_history: .word 0 : 16 # = 16 for easier debugging
        y_history: .word 0 : 16

        # For rotation
        a_history: .word 0 : 16
        l_history: .word 4                  # history length

        a_current: .word 0                  # current alpha

        is_going: .word 0
        is_tracking: .word 0

        # Code properties
        control_code: .space 8   # input command code
        code_length: .word 0                # input command length

        prev_control_code: .space 8       # store previous input code

.text

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

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

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

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

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

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

        j waitForKey

# Function: print error to console
printError:
```

```
        li $v0, 4
        la $a0, error_msg
        syscall

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

resetInput:
        jal strClear
        nop

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

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

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

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

        la $s3, 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 printError


# Perform function and print code
go:
        jal strCpy      #chuan bi hoac sao chep chuoi can thiet
        jal GO          #thuc hien cac thao tac can thiet
        j printCode     #in ra ma dieu khien

stop:
        jal strCpy
        jal STOP
        j printCode

track:
        jal strCpy
        jal TRACK
        j printCode

untrack:
        jal strCpy
        jal UNTRACK
        j printCode


turnRight:
        jal strCpy
        lw $t7, is_going
        lw $s0, is_tracking

        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 printCode


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

        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 printCode


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

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

goBackward_turn:
        addi $s5, $s5, -4   # code_length--
        lw $s6, a_history($s5)  # $s6 = a_history[code_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]
        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 printCode

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

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

```
        addi $sp, $sp, 4
        sw $s3, 0($sp)
        addi $sp, $sp, 4
        sw $s4, 0($sp)

        lw $s1, WHEREX   # s1 = x
        lw $s2, WHEREY   # s2 = y
        lw $s4, a_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   # change MOVING port
        addi $k0, $zero, 1  # to logic 1,
        sb $k0, 0($at)   # to start running

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

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

GO_end:
        jr $ra

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

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

        sw $zero, is_going  # is_going = 0

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

STOP_end:
        jr $ra

#-------------------------------------------------------------
# TRACK()
#-------------------------------------------------------------
TRACK:
        addi $sp, $sp, 4   # backup
        sw $at, 0($sp)
        addi $sp, $sp, 4
        sw $k0, 0($sp)

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

        addi $s0, $zero, 1
        sw $s0, is_tracking

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

TRACK_end:
        jr $ra

#-------------------------------------------------------------
# UNTRACK()
#-------------------------------------------------------------
UNTRACK:
```

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

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

        sw $zero, is_tracking

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

UNTRACK_end:
        jr $ra

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

        li $t1, HEADING # change HEADING port
        la $t2, a_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 control_code
# - output: $t0 = 0 if not equal, 1 if equal
#-----------------------------------------------------------
strcmp:
        addi $sp, $sp, 4   # back up
        sw $t1, 0($sp)
        addi $sp, $sp, 4
        sw $s1, 0($sp)
        addi $sp,$sp,4
        sw $t2, 0($sp)
```

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

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

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

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

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

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

        j strcmp_end

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

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

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

        jr $ra

#-------------------------------------------------------------
# strClear()
#-------------------------------------------------------------
strClear:
        addi $sp, $sp, 4   # backup
        sw $t1, 0($sp)
        addi $sp, $sp, 4
        sw $t2, 0($sp)
        addi $sp, $sp, 4
        sw $s1, 0($sp)
        addi $sp, $sp, 4
        sw $t3, 0($sp)
        addi $sp, $sp, 4
        sw $s2, 0($sp)

        lw $t3, code_length   # $t3 = code_length
```

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

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

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

        sw $zero, code_length  # reset code_length = 0

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

        jr $ra


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

        li $t2, 0
        # load address of prev_control_code
        la $s1, prev_control_code

        # load address of control_code
        la $s2, control_code

strCpy_loop:
        beq $t2, 3, strCpy_end

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

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

        j strCpy_loop

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

        jr $ra

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

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

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

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

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

        addi $sp, $sp, 4
        sw $at, 0($sp)
        addi $sp, $sp, 4
        sw $s0, 0($sp)
        addi $sp, $sp, 4
        sw $s1, 0($sp)
        addi $sp, $sp, 4
        sw $s2, 0($sp)
        addi $sp, $sp, 4
        sw $t4, 0($sp)
        addi $sp, $sp, 4
```

```
        sw $s3, 0($sp)
        #-----------------------------------------------------
        # Processing
        #-----------------------------------------------------
getCode:
        li $t1, IN_ADRESS_HEXA_KEYBOARD
        li $t2, OUT_ADRESS_HEXA_KEYBOARD

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

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

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

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

getCodeInChar:
        beq $a0, 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 storeCode
case_1:
        li $s0, '1'
```

```
        j storeCode
case_2:
        li $s0, '2'
        j storeCode
case_3:
        li $s0, '3'
        j storeCode
case_4:
        li $s0, '4'
        j storeCode
case_5:
        li $s0, '5'
        j storeCode
case_6:
        li $s0, '6'
        j storeCode
case_7:
        li $s0, '7'
        j storeCode
case_8:
        li $s0, '8'
        j storeCode
case_9:
        li $s0, '9'
        j storeCode
case_a:
        li $s0, 'a'
        j storeCode
case_b:
        li $s0, 'b'
        j storeCode
case_c:
        li $s0, 'c'
        j storeCode
case_d:
        li $s0, 'd'
        j storeCode
case_e:
        li $s0, 'e'
        j storeCode
case_f:
        li $s0, 'f'
        j storeCode

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

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

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

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

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

Task 2: Moving a ball on the Bitmap Display

```
.eqv KEY_CODE 0xFFFF0004
.eqv KEY_READY 0xFFFF0000
.eqv YELLOW 0x00FFFF00
.eqv SCREEN_MONITOR 0x10010000

.data
circle_end:     .word   1       # Con trỏ trỏ đến cuối của chu kỳ
circle:         .word           # Con trỏ trỏ đến mảng lưu dữ liệu của chu kỳ
.text
initial_declare:
        addi    $s0, $0, 255    # x = 255
        addi    $s1, $0, 255    # y = 255
        addi    $s2, $0, 0      # dx = 0
        add     $s3, $0, $0     # dy = 0
        addi    $s4, $0, 20     # r = 20
        addi    $a0, $0, 50     # t = 50ms/frame
        jal     circle_data

input:
        li      $k0, KEY_READY      # Kiểm tra xem có dữ liệu nhập không
        lw      $t0, 0($k0)
        bne     $t0, 1, edge_check
        jal     direction_change

# Kiểm tra xem hình tròn có chạm vào cạnh không
edge_check:

right:
        bne     $s2, 1, left # Nếu bóng di chuyển sang trái, không kiểm tra cạnh phải, kiểm tra cạnh
trái
        j       check_right

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

down:
        bne     $s3, 1, up
        j       check_down

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

move_circle:
        add     $s5, $0, $0     # Đặt màu thành đen
        jal     draw_circle     # Xóa hình tròn cũ
```

```
        add     $s0, $s0, $s2   # Cập nhật giá trị mới cho điểm trung tâm của chu kỳ
        add     $s1, $s1, $s3
        li      $s5, YELLOW # Đặt màu thành vàng
        jal     draw_circle    # Vẽ hình tròn mới

loop:
        li $v0, 32              # Giá trị syscall để ngủ
        syscall
        j       input           # Làm mới chu kỳ

# Thủ tục bên dưới

circle_data:
        addi    $sp, $sp, -4    # Lưu $ra
        sw      $ra, 0($sp)
        la      $s5, circle     # $s5 trở thành con trỏ của mảng "circle"
        mul     $a3, $s4, $s4   # $a3 = r^2
        add     $s7, $0, $0     # pixel x (px) = 0

pixel_data_loop:
        bgt     $s7, $s4, data_end
        mul     $t0, $s7, $s7   # $t0 = px^2
        sub     $a2, $a3, $t0   # $a2 = r^2 - px^2 = py^2
        jal     root            # $a2 = py

        add     $a1, $0, $s7    # $a1 = px
        add     $s6, $0, $0     # Sau khi lưu (px, py), (-px, py), (-px, -py), (px, -py), hoán đổi px và
py, sau đó lưu (-py, px), (py, px), (py, -px), (-py, -px)

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

        add     $t0, $0, $a1    # Hoán đổi px và -py
        add     $a1, $0, $a2
        add     $a2, $0, $t0
        addi    $s6, $s6, 1
        j       symmetric

finish:
        addi    $s7, $s7, 1 #px = px + 1
        j       pixel_data_loop

data_end:
        la      $t0, circle_end
        sw      $s5, 0($t0)     # Lưu địa chỉ cuối của mảng "circle"
        lw      $ra, 0($sp)
        addi    $sp, $sp, 4
```

```
        jr      $ra

root:                           # Tìm căn bậc hai của $a2
        add     $t0, $0, $0     # Đặt $t0 = 0
        add     $t1, $0, $0     # $t1 = $t0^2

root_loop:
        beq     $t0, $s4, root_end      # Nếu $t0 vượt quá 20, 20 sẽ là căn bậc hai
        addi    $t2, $t0, 1             # $t2 = $t0 + 1
        mul     $t2, $t2, $t2           # $t2 = ($t0 + 1)^2
        sub     $t3, $a2, $t1           # $t3 = $a2 - $t0^2
        bgez    $t3, continue           # Nếu $t3 < 0, $t3 = -$t3
        sub     $t3, $0, $t3

continue:
        sub     $t4, $a2, $t2           # $t4 = $a2 - ($t0 + 1)^2
        bgez    $t4, compare            # Nếu $t4 < 0, $t4 = -$t4
        sub     $t4, $0, $t4

compare:
        blt     $t4, $t3, root_continue  # Nếu $t3 >= $t4, $t0 không gần căn bậc hai của $a2 hơn $t0
+ 1
        add     $a2, $0, $t0            # Nếu không, $t0 là số gần nhất với căn bậc hai của $a2
        jr      $ra

root_continue:
        addi    $t0, $t0, 1
        add     $t1, $0, $t2
        j       root_loop

root_end:
        add     $a2, $0, $t0
        jr      $ra

pixel_save:
        sw      $a1, 0($s5)     # Lưu px trong mảng "circle"
        sw      $a2, 4($s5)     # Lưu py trong mảng "circle"
        addi    $s5, $s5, 8     # Di chuyển con trỏ đến khối null
        jr      $ra

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

case_d:
        bne     $t0, 'd', case_a
        addi    $s2, $0, 1      # dx = 1
        add     $s3, $0, $0     # dy = 0
        jr      $ra

case_a:
        bne     $t0, 'a', case_s
        addi    $s2, $0, -1     # dx = -1
        add     $s3, $0, $0     # dy = 0
        jr      $ra
```

```
case_s:
        bne     $t0, 's', case_w
        add     $s2, $0, $0     # dx = 0
        addi    $s3, $0, 1      # dy = 1
        jr      $ra

case_w:
        bne     $t0, 'w', case_x
        add     $s2, $0, $0     # dx = 0
        addi    $s3, $0, -1     # dy = -1
        jr      $ra

case_x:
        bne     $t0, 'x', case_z
        addi    $a0, $a0, 10    # t += 10
        jr      $ra

case_z:
        bne     $t0, 'z', default
        beq     $a0, 0, default # Chỉ giảm t khi t >= 0
        addi    $a0, $a0, -10   # t -= 10

default:
        jr      $ra

check_right:
        add     $t0, $s0, $s4   # Đặt $t0 sang phải của hình tròn (trung tâm + R)
        beq     $t0, 511, reverse_direction     # Đảo ngược hướng nếu cạnh chạm vào viền
        j       move_circle     # Quay lại nếu không

check_left:
        sub     $t0, $s0, $s4   # Đặt $t0 sang trái của hình tròn
        beq     $t0, 0, reverse_direction# Đảo ngược hướng nếu cạnh chạm vào viền
        j       move_circle     # Quay lại nếu không

check_down:
        add     $t0, $s1, $s4   # Đặt $t0 xuống dưới của hình tròn
        beq     $t0, 511, reverse_direction     # Đảo ngược hướng nếu cạnh chạm vào viền
        j       move_circle     # Quay lại nếu không

check_up:
        sub     $t0, $s1, $s4   # Đặt $t0 lên trên của hình tròn
        beq     $t0, 1, reverse_direction# Đảo ngược hướng nếu cạnh chạm vào viền
        j       move_circle

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

draw_circle:
        addi    $sp, $sp, -4    # Lưu $ra
        sw      $ra, 0($sp)
        la      $s6, circle_end
```

```
        lw      $s7, 0($s6)     # $s7 trở thành địa chỉ cuối của mảng "circle"
        la      $s6, circle     # $s6 trở thành con trỏ đến mảng "circle"

draw_loop:
        beq     $s6, $s7, draw_end      # Dừng lại khi $s6 = $s7
        lw      $a1, 0($s6)             # Lấy px
        lw      $a2, 4($s6)             # Lấy py
        jal     pixel_draw
        addi    $s6, $s6, 8             # Đến pixel tiếp theo
        j       draw_loop

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

pixel_draw:
        li      $t0, SCREEN_MONITOR
        add     $t1, $s0, $a1           # x cuối cùng (fx) = x + px
        add     $t2, $s1, $a2           # fy = y + py
        sll     $t2, $t2, 9             # $t2 = fy * 512
        add     $t2, $t2, $t1           # $t2 += fx
        sll     $t2, $t2, 2             # $t2 *= 4
        add     $t0, $t0, $t2
        sw      $s5, 0($t0)
        jr      $ra
```