TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

TRƯỜNG CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG

*****📖*****

BÁO CÁO CUỐI KÌ

IT3280

THỰC HÀNH KIẾN TRÚC MÁY TÍNH

Nhóm sinh viên thực hiện: **Nhóm 18**

Giảng viên hướng dẫn: **ThS. Lê Bá Vui**

1. Nguyễn Ngọc Lan - 20225730
2. Nguyễn Trung Sơn - 20226124

Năm học 2023 - 2024

MỤC LỤC

PHẦN 1: Nguyễn Ngọc Lan - 20225730

1. Đề bài (Đề 4)

Postscript CNC Marsbot

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

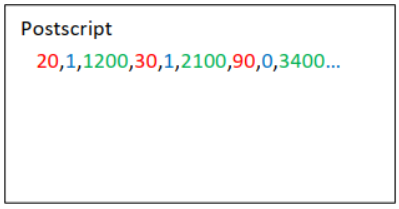
- Nếu lưỡi cắt dịch chuyển nhưng không cắt tấm kim loại, tức là Marsbot di chuyển nhưng không để lại vết (Track)
- Nếu lưỡi cắt dịch chuyển và cắt tấm kim loại, tức là Marsbot di chuyển và có để lại vết.

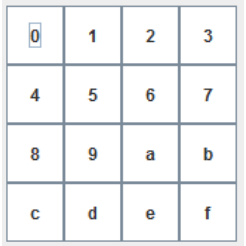
Để điều khiển Marsbot cắt đúng như hình dạng mong muốn, người ta nạp vào Marsbot một kịch bản là chuỗi ký tự bao gồm liên tiếp bộ 3 tham số (các giá trị phân cách nhau bởi dấu phẩy)

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

Hãy lập trình để CNC Marsbot có thể:

- Thực hiện cắt kim loại như đã mô tả
- Nội dung postscript được lưu trữ cố định bên trong mã nguồn
- Mã nguồn chứa 3 postscript và người dùng sử dụng 3 phím 0, 4, 8 trên bàn phím Key Matrix để chọn postscript nào sẽ được gia công.
- Một postscript chứa chữ DCE cần gia công. Hai script còn lại sinh viên tự đề xuất (tối thiểu 10 đường cắt)





2. Mã nguồn

```
.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 (0 or non-0): whether or not to leave a track
.eqv IN_ADDRESS_HEXA_KEYBOARD 0xFFFF0012
.eqv OUT_ADDRESS_HEXA_KEYBOARD 0xFFFF0014
.eqv MASK_CAUSE_KEYMATRIX 0x00000800
# Bit 11: Key matrix interrupt
.data
        script0: .asciiz
"71,1,1700,37,1,1700,17,1,1700,0,1,1700,341,1,1700,320,1,1700,295,1,1700,180,1
,8820,90,0,7000,270,1,2300,345,1,4520,15,1,4000,75,1,2500,90,0,2000,180,1,882
0,90,1,2666,0,0,4410,270,1,2666,0,0,4410,90,1,2666"

        script4: .asciiz
"315,1,2000,270,1,2000,225,1,2000,180,1,2000,135,1,2000,90,1,2000,135,1,2000,
180,1,2000,225,1,2000,270,1,2000,315,1,2000,90,0,6000,135,1,2000,90,1,2000,45
,1,2000,0,1,6800,315,1,2000,270,1,2000,225,1,2000,180,1,6800,90,0,6000,180,0,1
650,0,1,10000,90,0,6000,255,1,4000,195,1,4500,165,1,4500,90,1,4000,90,0,4000,
180,0,100,0,1,10000,270,0,3000,90,1,6000"

        script8: .asciiz
"90,1,12000,180,1,8000,270,1,12000,0,1,8000,90,0,5800,180,0,2200,162,1,1200,9
0,1,1200,234,1,1200,162,1,1200,306,1,1200,234,1,1200,18,1,1200,306,1,1200,90,
1,1200,18,1,1200"

        StringWrong:        .asciiz "Postscript "

        StringAllwrong:     .asciiz "Tat ca Postscript deu sai"
```

```
    Reasonwrong1:    .asciiz "sai do loi cu phap\n"
    Reasonwrong2:    .asciiz "sai do thieu bo so\n"
    EndofProgram:    .asciiz "Ket thuc chuong trinh!"
    checkNOTdone:    .asciiz "Chua check xong xin hay doi mot lat"
    Done:            .asciiz "Da cat xong!"
    Choose:      .asciiz "----------------------MENU----------------------\nVui long chon phim tren Digital Lab Sim\n0: DCE\n4: SOICT\n8: Co Viet Nam\nC: Thoat chuong trinh"
    NotNormal: .asciiz "Xay ra loi bat thuong! Vui long thu lai chuong trinh!"
    Array:       .word


.text
main:       li $v0, 55 #Thông báo MENU ra màn hình
            la $a0, Choose
            li $a1, 1
            syscall
            li $t1, IN_ADDRESS_HEXA_KEYBOARD
            li $t2, OUT_ADDRESS_HEXA_KEYBOARD
            li $t3, 0x80 #bit 7 = 1 để bật ngắt
            sb $t3, 0($t1)
            la $k0, Array
            li $s0, 4
            div $k0, $s0
            mfhi $s1 #Gán s1 = địa chỉ mảng % 4
            beqz $s1, StrChk
            #Nếu địa chỉ mảng chia hết cho 4 rồi thì thôi, nếu không thì cộng thêm để chia hết cho 4
            sub $s0, $s0, $s1 #Gán s0 = 4 - (địa chỉ mảng % 4)
```

```
            add $k0, $k0, $s0
            #Gán địa chỉ mảng mới = địa chỉ mảng cũ +  4 - (địa chỉ mảng % 4)
StrChk:     jal StringCheck

Loop:       nop
            addi $v0, $zero, 32
            li $a0, 200
            syscall
            nop
            nop
            b Loop #Đợi người dùng nhấn phím trên Digital Lab Sim
            b Loop #Chỉ được gọi tới khi ngắt đúng vào lệnh b Loop ở trên

exit:       li $v0, 55
            la $a0, EndofProgram
            li $a1, 1
            syscall
            li $v0, 10
            syscall
endMain:

#---------------------------------------------------------
#StringCheck: Kiểm tra dữ liệu đầu vào
#Input: Địa chỉ các chuỗi 0, 4, 8
#Output: Thông báo ra màn hình nếu chuỗi sai (Kết thúc chương trình nếu tất cả các chuỗi đều sai)
#Các thanh ghi sử dụng:
```

```
#       a0:     - chứa địa chỉ các chuỗi ban đầu
#               - sau khi chạy hàm check thì = 1 hoặc 2 nếu chuỗi sai, = địa chỉ mảng
#               nếu chuỗi đúng
#       a1: chứa giá trị đúng/sai của chuỗi
#               (0: đúng, 1: lỗi cú pháp, 2: lỗi thiếu bộ số, 3: tất cả các chuỗi đều sai)
#       t7, t8, t9: bằng 1 hoặc 2 nếu chuỗi 0, 4, 8 sai (tùy lỗi); là địa chỉ của mảng số
#       nếu chuỗi đúng
#       s0: đếm số chuỗi sai
#       k1: Chứa chuỗi sai (0, 4 hoặc 8)
#-------------------------------------------------------------
StringCheck:    li $s0, 0
                addi $sp, $sp, 4 #Lưu $ra vào stack để dùng sau
                sw $ra, 0($sp)
Check_script0:  la $a0, script0
                jal Check
                add $t7, $a0, $zero
                #t7 = 1 hoặc 2 nếu chuỗi sai, = địa chỉ mảng nếu chuỗi đúng
                beqz $a1, Check_script4
                #Nếu chuỗi không sai thì tiếp tục kiểm tra chuỗi tiếp theo
                addi $k1, $zero, 0
                jal WrongMessage
Check_script4:  la $a0, script4
                jal Check
                addi $t8, $a0, 0
                #t8 = 1 hoặc 2 nếu chuỗi sai, = địa chỉ mảng nếu chuỗi đúng
                beqz $a1, Check_script8
                #Nếu chuỗi không sai thì tiếp tục kiểm tra chuỗi tiếp theo
```

```
                addi $k1, $zero, 4
                jal WrongMessage
Check_script8:  la $a0, script8
                jal Check
                addi $t9, $a0, 0
                #t9 = 1 hoặc 2 nếu chuỗi sai, = địa chỉ mảng nếu chuỗi đúng
                beqz $a1, restoreRA
                #Nếu chuỗi không sai thì khôi phục $ra và kết thúc kiểm tra
                addi $k1, $zero, 8
                jal WrongMessage
AllWrong:       bne $s0, 3, restoreRA
#Nếu tất cả các chuỗi đều sai thì thông báo ra màn hình và kết thúc chương trình
                addi $a1, $zero, 3
                jal WrongMessage
                j exit
restoreRA:      lw $ra, 0($sp) #Khôi phục $ra
                addi $sp, $sp, -4
end_of_StringCheck:     addi $t6, $t6, 1 #Gán t6 = 1 → Check xong
                        jr $ra
#-------------------------------------------------------
#WrongMessage: Thông báo lỗi sai
#Input: Chuỗi sai (k1) + Lỗi sai của chuỗi đó (a1)
#Output: Thông báo chuỗi sai và lỗi sai ra màn hình
#Các thanh ghi sử dụng:
#       k1: Chứa chuỗi sai (0, 4 hoặc 8)
#       a1: Chứa giá trị sai của chuỗi
#               (1: lỗi cú pháp, 2: lỗi thiếu bộ số, 3: tất cả các chuỗi đều sai)
```

```
#-----------------------------------------------------
WrongMessage:   beq $a1, 3, Reason3
                #Nếu a1 = 3 thì thông báo tất cả các chuỗi đều sai
                li $v0, 4 #In chuỗi sai
                la $a0, StringWrong
                syscall
                li $v0, 1
                add $a0, $k1, $zero
                syscall
                li $v0, 11
                addi $a0, $zero, 0x20 #print space
                syscall
                beq $a1, 1, Reason1 #Nếu a1 = 1 thì thông báo lỗi do cú pháp
                beq $a1, 2, Reason2 #Nếu a1 = 2 thì thông báo lỗi thiếu bộ số
Reason1:        li $v0, 4
                la $a0, Reasonwrong1 #Sai do lỗi cú pháp
                syscall
                j endWM
Reason2:        li $v0, 4
                la $a0, Reasonwrong2 #Sai do lỗi thiếu bộ số
                syscall
                j endWM
Reason3:        li $v0, 55
                la $a0, StringAllwrong
                li $a1, 0
                syscall
endWM:          jr $ra
```

```
#-------------------------------------------------------
#Check: Kiểm tra 1 chuỗi có hợp lệ hay không
#Input: Địa chỉ chuỗi (a0)
#Output:    a0:    (chuỗi đúng) là địa chỉ mảng của chuỗi đang xét;
#                   (chuỗi sai) bằng 1 hoặc 2
#           a1: (chuỗi đúng) bằng 0; (chuỗi sai) bằng 1 hoặc 2
#Các thanh ghi sử dụng:
#       a0: địa chỉ ban đầu của script, gọi là s
#       a1: 0: chuỗi đúng; 1: lỗi cú pháp; 2: lỗi thiếu bộ số
#       a2: giá trị đang xét trên script, gọi là s[i]
#       v0: giữ giá trị trước của a2, gọi là s[i-1]
#       a3: đếm số dấu phẩy
#       s0: đếm số chuỗi sai
#       k0: địa chỉ mảng của chuỗi tiếp theo (nếu chuỗi đang xét đúng)
#-------------------------------------------------------
Check:      li $a3, 0                   #Khởi tạo số dấu phẩy = 0
            lb $a2, 0($a0)              #a2 = s[0]
            beq $a2, 0x2C, wrong1       #Nếu ký tự đầu tiên là phẩy → Lỗi cú pháp
loop_Check:lb $a2, 0($a0)               #a2 = s[i]
            beq $a2, 0x2C, is_comma #Nếu s[i] = ',' thì nhảy đến is_comma
            beq $a2, 0x20, continue #Nếu s[i] = ' ' thì bỏ qua nhảy đến continue
            beq $a2, 0x00, end_Check #Nếu s[i] = '\0' thì nhảy đến end_Check
            blt $a2, 0x30, wrong1 #Nếu s[i] không thuộc ['0', '9'] → Lỗi cú pháp
            bgt $a2, 0x39, wrong1
            j continue
is_comma:   beq $v0, 0x2C, wrong1   #Nếu có 2 dấu phẩy liên tiếp → Lỗi cú pháp
            addi $a3, $a3, 1        #Không thì tăng số dấu phẩy lên 1
```

```
continue:    addi $v0, $a2, 0     #v0 = s[i-1]
             addi $a0, $a0, 1     #Nếu không có lỗi nào thì chuyển sang s[i+1]
j loop_Check
wrong1:      li $a1, 1            #a1 = 1, lỗi cú pháp
             li $a0, 1
             addi $s0, $s0, 1     #Tăng số chuỗi sai lên 1
             jr $ra
wrong2:      li $a1, 2            #a1 = 2, lỗi thiếu bộ số
             li $a0, 2
             addi $s0, $s0, 1     #Tăng số chuỗi sai lên 1
             jr $ra
end_Check:   beq $v0, 0x2C, wrong1   #Nếu ký tự cuối cùng là ',' → Lỗi cú pháp
             li $a2, 3               #Gán a2 = 3
             div $a3, $a2
             mfhi $a2                #a2 = a3 % 3 = số dấu phẩy % 3
             bne $a2, 2, wrong2      #Nếu a2 % 3 != 2 → Lỗi không đủ bộ số
             li $a1, 0 #Nếu không gặp lỗi nào ở trên thì chuỗi đúng → Gán a1 = 0
             addi $a0, $k0, 0        #và gán a0 = địa chỉ mảng của chuỗi đang xét
             addi $a3, $a3, 3        #Gán a3 = số phần tử của mảng = Số dấu phẩy + 3
#(+1 → số chữ số, dành 1 ô đầu tiên đánh dấu chuỗi đã chuyển thành mảng hay
chưa, dành 1 ô cuối cùng lưu ký tự đánh dấu kết thúc chuỗi)
             sll $a3, $a3, 2         #a3*4
             add $k0, $k0, $a3       #k0 = địa chỉ mảng của chuỗi tiếp theo
             li $a2, -1              #Ký tự đánh dấu kết thúc chuỗi
             sw $a2, -4($k0)
             jr $ra
```

```
#Chương trình xử lý ngắt
.ktext 0x80000180
Check_Cause:    mfc0 $t4, $13
                li $t3, MASK_CAUSE_KEYMATRIX
                and $at, $t4, $t3
                bne $at, $t3, Unusual
#Nếu không phải ngắt do nhấn phím trên Digital Lab Sim thì nhảy đến Unusual
                bne $t6, 1, notDone
#Nếu chưa check xong thì thông báo ra màn hình
                j Keymatrix_Intr
Unusual:        li $v0, 55          #Thông báo ra màn hình là
                la $a0, NotNormal #lỗi bất thường
                li $a1, 0
                syscall
                li $v0, 10 #exit
                syscall
notDone:        addi $sp, $sp, 4 #Lưu v0 của chương trình chính vào stack
                sw $v0, 0($sp)
                addi $sp, $sp, 4 #Lưu a0 của chương trình chính vào stack
                sw $a0, 0($sp)
                addi $sp, $sp, 4 #Lưu a1 của chương trình chính vào stack
                sw $a1, 0($sp)
                li $v0, 55
                la $a0, checkNOTdone #Chưa check xong
                li $a1, 1
                syscall
                lw $a1, 0($sp)
```

```
                addi $sp, $sp, -4 #Khôi phục a1
                lw $a0, 0($sp)
                addi $sp, $sp, -4 #Khôi phục a0
                lw $v0, 0($sp)
                addi $sp, $sp, -4 #Khôi phục v0
                j return
#---------------------------------------------------------
#s0: là địa chỉ mảng (nếu có gọi là s); là 1 hoặc 2 nếu lỗi
#s7: địa chỉ chuỗi
#k1: Tên postscript đang xét
#---------------------------------------------------------
Keymatrix_Intr:  li $t3, 0x81
#Kiểm tra xem có phải phím ở hàng 1 0, 1, 2, 3 được nhấn không
                sb $t3, 0($t1)
                lb $a0, 0($t2)
                bne $a0, 0x11, Row2
#Nếu 0 không được nhấn thì kiểm tra hàng tiếp theo
                add $s0, $t7, $zero #s0 = địa chỉ mảng script 0 (nếu có)
                addi $k1, $zero, 0
                la $s7, script0
                jal runScript       #Nếu 0 được nhấn thì chạy script 0
                j next_pc
Row2:           li $t3, 0x82
#Kiểm tra xem có phải phím ở hàng 2 4, 5, 6, 7 được nhấn không
                sb $t3, 0($t1)
                lb $a0, 0($t2)
                bne $a0, 0x12, Row3
```

```
#Nếu 4 không được nhấn thì kiểm tra hàng tiếp theo
                add $s0, $t8, $zero #s0 = địa chỉ mảng script 4 (nếu có)
                addi $k1, $zero, 4
                la $s7, script4
                jal runScript       #Nếu 4 được nhấn thì chạy script 4
                j next_pc
Row3:           li $t3, 0x84
#Kiểm tra xem có phải phím ở hàng 3 8, 9, A, B được nhấn không
                sb $t3, 0($t1)
                lb $a0, 0($t2)
                bne $a0, 0x14, Row4
#Nếu 8 không được nhấn thì kiểm tra hàng tiếp theo
                add $s0, $t9, $zero #s0 = địa chỉ mảng script 8 (nếu có)
                addi $k1, $zero, 8
                la $s7, script8
                jal runScript       #Nếu 8 được nhấn thì chạy script 8
                j next_pc
Row4:           li $t3, 0x88
#Kiểm tra xem có phải phím ở hàng 4 C, D, E, F được nhấn không
                sb $t3, 0($t1)
                lb $a0, 0($t2)
                beq $a0, 0x18, exit #Nếu C được nhấn thì kết thúc chương trình
next_pc:        mfc0 $at, $14       #$at ← Coproc0.$14 = Coproc0.epc
                addi $at, $at, 4    #$at = $at + 4 (lệnh tiếp theo)
                mtc0 $at, $14       #Coproc0.$14 = Coproc0.epc ← $at
return:         eret                #Quay về chương trình chính
#------------------------------------------------------------
```

```
#runScript: Chạy Script
#Input:        s0: địa chỉ mảng (nếu có gọi là a); 1 hoặc 2 nếu lỗi
#              s7: địa chỉ chuỗi muốn chạy
#Output:       Hình được cắt xong (Nếu chuỗi chưa chuyển thành mảng thì chuyển)
#              hoặc
#              Thông báo chuỗi sai
#Các thanh ghi sử dụng:
#       s1: giá trị phần tử của mảng, gọi là a[i]
#       s2: biến chạy, gọi là i
#-------------------------------------------------------------
runScript:    beq $s0, 1, Wrong
              beq $s0, 2, Wrong
              lw $s1, 0($s0)
              beq $s1, 1, run #Nếu chuỗi đã chuyển thành mảng số thì chạy script
              addi $sp, $sp, 4          #Lưu $ra vào stack để dùng sau
              sw $ra, 0($sp)
              add $s4, $s0, $zero       #Truyền địa chỉ mảng
              add $t5, $s7, $zero       #và địa chỉ chuỗi vào hàm StringSolve
              jal StringSolve           #Nếu chưa chuyển thì nhảy đến hàm chuyển
run:          add $s2, $zero, $zero     #Gán s2 = i = 0
              add $s3, $zero, $zero     #Gán s3 = địa chỉ (a[i]) = 0
              li $a0, 135               #Quay Marsbot 135* và bắt đầu chạy
              jal ROTATE
              jal GO
              addi $v0, $zero, 32       #Trong 14000ms
              li $a0, 14000
              syscall
```

```
DRAW:       jal getVALUE            #Lấy góc
            beq $s1, -1, endDRAW    #Nếu s1 = -1 thì kết thúc vẽ
            add $a0, $zero, $s1     #Quay Marsbot
            jal ROTATE
            jal getVALUE            #Bật TRACK hoặc không
            beq $s1, $zero, KeepRunning
            #Nếu s1 = 0 thì không bật TRACK
            jal TRACK
KeepRunning: jal getVALUE       #Lấy thời gian
            addi $v0, $zero, 32#Tiếp tục chạy trong (s1)ms
            add $a0, $zero, $s1
            syscall
            jal UNTRACK         #Nếu không bật track thì tắt cũng không sao
            j DRAW
endDRAW:    jal STOP
            li $v0, 55          #Thông báo đã cắt hình xong ra màn hình
            la $a0, Done
            li $a1, 1
            syscall
            j resRA
Wrong:      li $v0, 59
            la $a0, StringWrong
            beq $s0, 2, error2
            #Nếu s0 = 1 hoặc 2 thì chuỗi sai và thông báo ra màn hình
error1:     la $a1, Reasonwrong1
            syscall
            j endRun
```

```
error2:         la $a1, Reasonwrong2
                syscall
                j endRun
resRA:          lw $ra, 0($sp) #Khôi phục $ra
                addi $sp, $sp, -4
endRun:         jr $ra
#---------------------------------------------------------
#StringSolve: Biến chuỗi thành mảng số
#Input:     Địa chỉ chuỗi cần chuyển ($t5)
#           Địa chỉ mảng của chuỗi đó ($s4)
#Output: Mảng đã chuyển xong (phần tử đầu tiên từ 0 thành 1 (mảng đã chuyển))
#Các thanh ghi sử dụng:
#       t5: Địa chỉ chuỗi, gọi là s
#       s4: Địa chỉ mảng
#       s6: Giá trị của chuỗi, gọi là s[i]
#       s1: Đếm số chữ số của 1 số
#       s2: Giá trị số (từ chuỗi chuyển sang)
#       s3: 10
#       s5: 10^k (bắt đầu từ 10^0 rồi tăng dần lên)
#---------------------------------------------------------
StringSolve: addi $sp, $sp, 4
             addi $s3, $zero, 1
             sw $s3, 0($s4)
             #Lưu 1 vào phần tử đầu tiên của mảng để xác nhận chuỗi đã chuyển
             addi $s4, $s4, 4     #Lưu giá trị vào mảng từ vị trí thứ 2
             li $s3, 10
mainSS:      li $s2, 0
```

```
                li $s1, 0               #Đếm số chữ số (khởi tạo = 0)
                li $s5, 1               #Lưu giá trị 10^k (khởi tạo = 10^0 = 1)
SS_loop:        lb $s6, 0($t5)          #Lấy s[i]
                beq $s6, 0x20, SS_next          #Nếu s[i] = ' ' thì bỏ qua
                beq $s6, 0x2C, String2Array     #Nếu s[i] = ','
                beq $s6, 0x00, String2Array
                #hoặc s[i] = '\0' thì chuyển chuỗi số trước đó thành số
                addi $sp, $sp, 1
                sb $s6, 0($sp)          #Lưu s[i] vào stack
                addi $s1, $s1, 1        #Tăng số chữ số lên 1
SS_next:        addi $t5, $t5, 1        #Tăng địa chỉ trỏ đến phần tử s[i+1]
                j SS_loop
String2Array: add $v0, $s6, $zero #Gán v0 = s[i-1]
str2a_loop:     ble $s1, $zero, Save2Array
                #Lặp lại việc chuyển chuỗi thành số cho đến khi số chữ số = 0
                lb $s6, 0($sp)
                #Lấy các chữ số ra khỏi stack → lấy từ hàng đơn vị trở lên gán cho s6
                addi $sp, $sp, -1
                addi $s6, $s6, -48  #s[i] = s[i] - '0' (chuyển ký tự số → số)
                mult $s6, $s5
                mflo $s6 #s[i] = s[i]*(10^k) (ban đầu là 10^0 = 1 rồi tăng dần)

                add $s2, $s6, $s2   #s2 = s2 + s[i]*(10^k)
                mult $s5, $s3
                mflo $s5            #Gán s5 = (10^k)*10 = 10^(k+1)
                addi $s1, $s1, -1   #Giảm số chữ số đi 1 và tiếp tục vòng lặp
                j str2a_loop
```

```
Save2Array: sw $s2, 0($s4)      #Lưu số đã chuyển xong vào mảng
            add $s4, $s4, 4     #Tăng địa chỉ mảng lên 1, trỏ đến ô nhớ tiếp theo
            addi $t5, $t5, 1    #Tăng địa chỉ chuỗi lên 1, trở đến phần tử tiếp theo
            beq $v0, 0, end_of_StringSolve
            #Nếu s[i-1] = '\0' thì kết thúc hàm chuyển
            j mainSS
end_of_StringSolve:     addi $sp, $sp, -4
                        jr $ra

#-------------------------------------------------------------
#getVALUE: Lấy giá trị từ mảng
#Input:     s2: Vị trí trong mảng
#           s0: Địa chỉ gốc của mảng
#Output:    s1: Giá trị phần tử tại vị trí a[i]
#-------------------------------------------------------------
getVALUE: addi $s2, $s2, 1      #i++
          sll $s4, $s2, 2       #Gán s4 = 4i
          add $s3, $s4, $s0     #s3 = 4i + địa chỉ mảng a = địa chỉ cùa a[i]
          lw $s1, 0($s3)        #s1 = a[i]
          jr $ra

GO:       li $at, MOVING    # change MOVING port
          addi $k0, $zero, 1  # to  logic 1,
          sb $k0, 0($at)        # to start running
          nop
          jr $ra
          nop
```

```
STOP:       li $at, MOVING          # change MOVING port to 0
            sb $zero, 0($at)        # to stop
            nop
            jr $ra
            nop

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

UNTRACK:li $at, LEAVETRACK      # change LEAVETRACK port to 0
            sb $zero, 0($at)        # to stop drawing tail
            nop
            jr $ra
            nop
#-----------------------------------------------------------
#ROTATE: Quay Marsbot
#Input:     a0: Góc quay từ 0 đến 359
#           0 : North (up)
#           90: East  (right)
#           180: South (down)
#           270: West  (left)
#-----------------------------------------------------------
```

```
ROTATE:  li $at, HEADING   # change HEADING port
         sw $a0, 0($at)    # to rotate robot
         nop
         jr $ra
         nop
```

3. Phân tích

3.1. Các bước thực hiện

Bước 1: Thông báo MENU ra màn hình, người dùng có thể chọn 1 trong 4 phím trên Digital Lab Sim để chạy Marsbot cắt hình tương ứng:

0: Cắt hình DCE

4: Cắt hình SOICT

8: Cắt hình cờ Việt Nam

C: Thoát chương trình

Lưu ý: *Ấn các phím khác Marsbot sẽ không chạy*

Bước 2: Kiểm tra các postscript có hợp lệ hay không, nếu không hợp lệ chương trình thông báo chuỗi không hợp lệ theo lỗi ra phần Run I/O trên phần mềm MARS.

Bước 3: Đợi người dùng nhấn phím trên Digital Lab Sim.

Bước 4: Sau khi người dùng nhấn phím trên Digital Lab Sim, chương trình ngắt sẽ được thực hiện. Có nhiều kiểu ngắt:

- Ngắt do ấn phím trên Digital Lab Sim:
  - Trường hợp chương trình chưa kiểm tra postscript xong sẽ thông báo ra màn hình và quay lại chương trình chính tiếp tục kiểm tra.
  - Trường hợp đã kiểm tra postscript xong rồi chương trình sẽ kiểm tra phím nào được ấn và chạy postscript tương ứng với phím đấy.
- Ngắt trong trường hợp khác: Lỗi bất thường, thông báo ra màn hình rồi kết thúc chương trình.

Bước 5: Khi chạy postscript:

- Postscript được ấn nếu *sai* chương trình sẽ:
  - Thông báo ra màn hình lỗi sai.

    - Quay về chương trình chính đợi người dùng ấn phím khác.
- Postscript được ấn nếu *đúng* chương trình sẽ:
    - Kiểm tra xem postscript đã chuyển thành mảng hay chưa:
        - Chưa chuyển: chương trình sẽ gọi đến hàm chuyển chuỗi thành mảng.
        - Chuyển rồi: chương trình sẽ di chuyển Marsbot theo postscript đã chọn.
    - Thông báo ra màn hình đã cắt xong hình.
    - Quay về chương trình chính đợi người dùng ấn phím khác.

Bước 6: Tiếp tục lặp lại các bước **3** đến **5** cho đến khi người dùng ấn phím C kết thúc chương trình.

3.2. Ý nghĩa các chuỗi được khai báo

Tên chuỗi	Ý nghĩa
script0	Postscript tương ứng với phím 0 trên Digital Lab Sim
script4	Postscript tương ứng với phím 4 trên Digital Lab Sim
script8	Postscript tương ứng với phím 8 trên Digital Lab Sim
StringWrong	Kết hợp với Reasonwrong1 và Reasonwrong2 thông báo postscript sai do lỗi ra màn hình
StringAllWrong	Thông báo tất cả postscript đều sai
Reasonwrong1	Thông báo lỗi sai do cú pháp
Reasonwrong2	Thông báo lỗi sai do thiếu bộ số (không đủ 3n số)
EndofProgram	Thông báo kết thúc chương trình
checkNOTdone	Thông báo chưa check xong postscript
Done	Thông báo đã cắt hình xong
Choose	Hiện MENU những phím có thể chọn trên Digital Lab Sim và ý nghĩa của chúng
NotNormal	Thông báo nếu có lỗi bất thường xảy ra
Array	Mảng lưu các giá trị số từ chuỗi chuyển thành

3.3. Các thanh ghi sử dụng cố định

Thanh ghi	Vai trò
$t1	Lưu địa chỉ IN_ADDRESS_HEXA_KEYBOARD
$t2	Lưu địa chỉ OUT_ADDRESS_HEXA_KEYBOARD
$t6	Đánh dấu postscript đã được check xong hay chưa

$t7	Bằng 1 hoặc 2 nếu postscript 0 lỗi Là địa chỉ mảng nếu postscript 0 đúng
$t8	Bằng 1 hoặc 2 nếu postscript 4 lỗi Là địa chỉ mảng nếu postscript 4 đúng
$t9	Bằng 1 hoặc 2 nếu postscript 8 lỗi Là địa chỉ mảng nếu postscript 8 đúng

3.4. Ý nghĩa các chương trình con

StringCheck: Kiểm tra dữ liệu đầu vào

Input: Địa chỉ các postscript 0, 4, 8.

Output:

- $t7, $t8, $t9: bằng 1 hoặc 2 nếu postscript 0, 4, 8 sai (tùy lỗi); là địa chỉ của mảng số nếu postscript đúng.
- Thông báo ra màn hình nếu postscript sai (Kết thúc chương trình nếu tất cả các postscript đều sai).

Các thanh ghi sử dụng:

- $a0:
  - Chứa địa chỉ các postscript ban đầu.
  - Sau khi chạy hàm ***Check*** thì bằng 1 hoặc 2 nếu postscript sai, là địa chỉ mảng nếu postscript đúng.
- $a1: Chứa giá trị đúng/sai của postscript:
  0: đúng
  1: lỗi cú pháp
  2: lỗi thiếu bộ số
  3: tất cả các postscript đều sai
- $t7, $t8, $t9:
  - Postscript sai: Là 1 hoặc 2 (tùy lỗi)
  - Postscript đúng: Là địa chỉ của mảng số
- $s0: Đếm số postscript sai.
- $k1: Chứa postscript sai (0, 4 hoặc 8).

Giải thích:

- Đầu tiên khởi tạo số postscript sai bằng 0 ($s0 = 0).
- Rồi lưu lại địa chỉ trở về ở thanh ghi $ra vào stack (do trong hàm này sẽ gọi đến hàm khác).

- Tiếp theo là kiểm tra script 0:
    - Gọi đến hàm *Check* với đầu vào là địa chỉ postsrcipt 0. Nếu postsrcipt sai thì tăng $s0 lên 1.
    - Lấy output $a0 của hàm Check gán cho $t7.
    - Kiểm tra nếu $a1 ≠ 0 thì gọi đến hàm *WrongMessage* để thông báo lỗi sai với postscript sai được lưu trong $k1.
    - Tiếp tục kiểm tra postscript 4.
- Tiếp đến là kiểm tra script 4:
    - Gọi đếm hàm *Check* với đầu vào là địa chỉ postsrcipt 4. Nếu postsrcipt sai thì tăng $s0 lên 1.
    - Lấy output $a0 của hàm Check gán cho $t8.
    - Kiểm tra nếu $a1 ≠ 0 thì gọi đến hàm *WrongMessage* để thông báo lỗi sai với postscript sai được lưu trong $k1.
    - Tiếp tục kiểm tra postscript 8.
- Kiểm tra script 8:
    - Gọi đếm hàm *Check* với đầu vào là địa chỉ postsrcipt 8. Nếu postsrcipt sai thì tăng $s0 lên 1.
    - Lấy output $a0 của hàm Check gán cho $t9.
    - Kiểm tra nếu $a1 ≠ 0 thì gọi đến hàm *WrongMessage* để thông báo lỗi sai với postscript sai được lưu trong $k1 rồi kiểm tra xem có phải tất cả các postscript đều sai hay không, nếu đúng thì gán $a1 = 3 rồi gọi đến hàm *WrongMessage* để thông báo lỗi ra màn hình và nhảy đến exit kết thúc chương trình.
- Khôi phục địa chỉ trả về vào thanh ghi $ra, gán $t6 = 1 (Đánh dấu đã check xong) rồi quay về chương trình chính.

WrongMessage: Thông báo lỗi sai

Input:

- Postscript sai ($k1)
- Lỗi sai của postscript đó ($a1)

Output: Thông báo postscript sai và lỗi sai ra màn hình.

Các thanh ghi sử dụng:

- $k1: Chứa postscript sai (0, 4 hoặc 8)
- $a1: Chứa giá trị sai của postscript

    1: lỗi cú pháp

2: lỗi thiếu bộ số

3: tất cả các postscript đều sai

Giải thích:

- Đầu tiên kiểm tra xem nếu $a1 = 3 thì thông báo "Tất cả postscript đều sai" ra màn hình.
- Nếu không in chuỗi "Postscript ($k1) " ra phần Run I/O rồi so sánh $a1.
- Nếu $a1 = 1 thì in chuỗi "sai do lỗi cú pháp" ra Run I/O.
- Nếu $a1 = 2 thì in chuỗi "sai do thiếu bộ số" ra Run I/O.
- Kết thúc hàm *WrongMessage*.

Check: Kiểm tra 1 postscript có hợp lệ hay không

Input: Địa chỉ chuỗi ($a0)

Output:

- $a0: Postscript đúng: là địa chỉ mảng của chuỗi đang xét

  Postscript sai: bằng 1 hoặc 2
- $a1: Postscript đúng: bằng 0

  Postscript sai: bằng 1 hoặc 2

Các thanh ghi sử dụng:

- $a0: Địa chỉ ban đầu của postscript, gọi là s
- $a1: 0: Postscript đúng

  1: Lỗi cú pháp

  2: Lỗi thiếu bộ số
- $a2: Giá trị đang xét trên postscript, gọi là s[i]
- $v0: Giữ giá trị trước của a2, gọi là s[i-1]
- $a3: Đếm số dấu phẩy
- $s0: Đếm số chuỗi sai
- $k0: Địa chỉ mảng của postscript tiếp theo (nếu postscript đang xét đúng)

Giải thích:

- Coi $a0 như s, $a2 như s[i], $v0 như s[i-1].
- Nếu s[0] = ',' là lỗi cú pháp → nhảy đến *wrong1*.

- Vòng lặp:
    - Load s[i] vào $a2 từ địa chỉ $a0.
    - Nếu s[i] = ‘,’ thì kiểm tra xem s[i-1] = ‘,’ hay không. Nếu có (tức là *2 dấu phẩy liên tiếp*) là lỗi cú pháp → nhảy đến *wrong1*.
    - Nếu s[i] = ‘ ’ (dấu cách) thì bỏ qua.
    - Nếu s[i] = ‘\0’ (NULL) thì:
        - Kiểm tra nếu ký tự cuối cùng s[i-1] = ‘,’ là lỗi cú pháp → nhảy đến *wrong1*.
        - Kiểm tra nếu số lượng dấu phẩy không phải 3n + 2 là thiếu bộ số → nhảy đến *wrong2*.
        - Nếu không phải 1 trong 2 lỗi trên thì postscript đúng:
            - Gán $a1 = 0.
            - Gán $a0 = địa chỉ mảng postscript đang xét.
        - Tăng $a3 lên 3 vì:
            - Số dấu phẩy + 1 = Số lượng số của postscript.
            - Dành vị trí đầu tiên để đánh dấu postscript đã chuyển thành mảng hay chưa.
            - Dành vị trí cuối cùng để đánh dấu kết thúc mảng.

            ➔ Khi đó $a3 sẽ thành số phần tử của mảng.
        - Gán $k0 = địa chỉ mảng của postscript tiếp theo (Ngay sau phần tử cuối cùng mảng trước).
        - Lưu $a2 = -1 vào vị trí cuối cùng của mảng để đánh dấu kết thúc mảng.
        - Kết thúc hàm *Check*.
    - Nếu s[i] < ‘0’ hay s[i] > ‘9’ là lỗi cú pháp → nhảy đến *wrong1*.
    - Nếu không có lỗi nào thì trỏ đến phần tử tiếp theo của chuỗi ($a0++).
- Lặp lại vòng lặp cho đến khi gặp NULL hoặc lỗi.
- Giải thích wrong1 và wrong2:
    - wrong1: Gán $a1, $a0 = 1 (*lỗi cú pháp*), tăng số postscript sai ($s0) lên 1 rồi quay lại hàm *StringCheck*.
    - wrong2: Gán $a1, $a0 = 2 (*lỗi thiếu bộ số*), tăng số postscript sai ($s0) lên 1 rồi quay lại hàm *StringCheck*.

Check_Cause: Kiểm tra nguyên nhân gây ra ngắt

Giải thích:

- Đầu tiên kiểm tra nếu không phải ngắt do nhấn phím trên Digital Lab Sim thì nhảy đến *Unusual* thông báo xảy ra lỗi bất thường rồi thoát chương trình.
- Nếu là ngắt do nhấn phím trên Digital Lab Sim mà kiểm tra $t6 ≠ 1 (chưa check xong) thì thông báo ra màn hình rồi quay lại chương trình chính.
  Lưu ý: Trước khi thông báo phải lưu lại các thanh ghi $v0, $a0, $a1 vào stack (do chương trình chính cũng sử dụng những thanh ghi này) và thông báo xong thì khôi phục lại những thanh ghi này.
- Nếu check postscript xong rồi thì nhảy đến *Keymatrix_Intr* để kiểm tra phím được nhấn.

Keymatrix_Intr: Kiểm tra phím được nhấn trên Digital Lab Sim

Giải thích: Kiểm tra từng hàng một:

- Hàng 1:
  - Nếu phím được nhấn không phải *0* thì kiểm tra tiếp hàng 2.
  - Nếu phím *0* được nhấn thì gọi đến hàm *runScript* với đầu vào là địa chỉ mảng script 0 được lưu trong $t7, tên postscript được lưu trong $k1 và địa chỉ script 0 được lưu trong $s7.
  - Chạy xong Script thì quay lại chương trình chính đợi người dùng nhấn phím khác.
- Hàng 2:
  - Nếu phím được nhấn không phải *4* thì kiểm tra tiếp hàng 3.
  - Nếu phím *4* được nhấn thì gọi đến hàm *runScript* với đầu vào là địa chỉ mảng script 4 được lưu trong $t7, tên postscript được lưu trong $k1 và địa chỉ script 4 được lưu trong $s7.
  - Chạy xong Script thì quay lại chương trình chính đợi người dùng nhấn phím khác.
- Hàng 3:
  - Nếu phím được nhấn không phải *8* thì kiểm tra tiếp hàng 4.
  - Nếu phím *8* được nhấn thì gọi đến hàm *runScript* với đầu vào là địa chỉ mảng script 8 được lưu trong $t7, tên postscript được lưu trong $k1 và địa chỉ script 8 được lưu trong $s7.
  - Chạy xong Script thì quay lại chương trình chính đợi người dùng nhấn phím khác.
- Hàng 4:
  - Nếu phím được nhấn không phải *C* thì kiểm tra tiếp hàng 3.
  - Nếu phím *C* được nhấn thì kết thúc chương trình.

- Nếu không thì quay lại chương trình chính đợi người dùng nhấn phím khác.

runScript: Chạy Script

Input:

- $s0: Địa chỉ mảng (nếu có gọi là a); 1 hoặc 2 nếu lỗi
- $s7: Địa chỉ mảng postscript muốn chạy, gọi là a

Output:

- Postscript đúng: Hình được cắt xong
- Postscript sai: Thông báo postscript sai do lỗi gì

Các thanh ghi sử dụng:

- $s1: Giá trị phần tử của mảng, gọi là a[i]
- $s2: Biến chạy, gọi là i

Giải thích:

- Nếu $s0 = 1 hoặc 2 → Lỗi → nhảy đến *Wrong*.
- Nếu a[0] = 0 (chuỗi chưa chuyển thành mảng) → gọi đến hàm *StringSolve* với đầu vào là $t5 (địa chỉ postscript cần chuyển) và $s4 (địa chỉ mảng tương ứng postscript).
- Lưu $ra vào stack (Do trong hàm này gọi đến hàm khác).
- Tiếp theo quay Marsbot 135° rồi cho nó chạy 14s để bắt đầu cắt hình.
- Vòng lặp:
  - Đầu tiên là lấy giá trị từ mảng lưu vào $s1 (góc).
  - Nếu $s1 = -1 (đã lấy hết giá trị trong mảng) thì kết thúc cắt (Dừng Marsbot và thông báo đã cắt xong ra màn hình) rồi khôi phục $ra và kết thúc hàm runScript.
  - Nếu không thì quay Marsbot theo góc ($s1).
  - Tiếp theo lại lấy giá trị từ mảng lưu vào $s1 (cắt hay không).
  - Nếu cắt thì gọi hàm TRACK.
  - Rồi lại lấy giá trị từ mảng lưu vào $s1 (thời gian). Tiếp tục chạy trong ($s1)ms.
  - Gọi hàm UNTRACK (Nếu không cắt thì vẫn UNTRACK).
- Quay lại vòng lặp cho đến khi $s1 = -1.
- Wrong: Hiện ra lỗi sai của postscript tùy lỗi.

StringSolve: Biến chuỗi thành mảng số

Input: $t5: Địa chỉ chuỗi cần chuyển

$s4: Địa chỉ mảng của chuỗi đó

Output: Mảng đã chuyển xong (phần tử đầu tiên đổi từ 0 thành 1)

Các thanh ghi sử dụng:

- $t5: Địa chỉ chuỗi, gọi là s
- $s4: Địa chỉ mảng
- $s6: Giá trị của chuỗi, gọi là s[i]
- $s1: Đếm số chữ số của 1 số
- $s2: Giá trị số (từ chuỗi chuyển sang)
- $s3: 10
- $s5: 10^k (bắt đầu từ 10^0 rồi tăng dần lên)

Ý tưởng:

Khi người dùng nhấn vào phím 0/4/8 *lần đầu tiên*, postscript tương ứng với phím được nhấn sẽ được chuyển thành mảng bằng cách gọi đến hàm này (Địa chỉ mảng đã được xác định (nếu postscript đúng) khi chạy hàm StringCheck) và được đánh dấu đã chuyển bằng phần tử đầu tiên của của mảng. Những lần ấn sau không cần chạy hàm này nữa.

Giải thích:

- Đầu tiên gán a[0] = 1 (Đánh dấu chuỗi đã chuyển thành mảng).
- Khởi tạo các giá trị:
  - $s3 = 10 (để nhân 10 trong String2Array).
  - $s2 = 0 (Giá trị số cuối cùng sau khi chuyển từ các ký tự sang).
  - $s1 = 0 (Đếm số chữ số).
  - $s5 = 1 ($10^k$ với k bắt đầu từ 0).
- Vòng lặp (SS_loop): *Dùng để đếm số chữ số của 1 số đằng trước dấu phẩy*
  - Lấy s[i] lưu vào $s6.
  - Nếu s[i] = ‘ ’ (dấu cách) thì bỏ qua.
  - Nếu s[i] = ‘,’ thì bắt đầu chuyển những ký tự số trước đó thành chuỗi. → nhảy đến *String2Array*.
  - Nếu s[i] = ‘\0’ (NULL) thì chuyển những ký tự số trước đó thành chuỗi → nhảy đến *String2Array* rồi kết thúc hàm StringSolve.

    - Nếu không phải những trường hợp trên thì s[i] thuộc [‘0’, ‘9’]. Lưu s[i] vào stack.
    - Tăng số chữ số lên 1 ($s1++).
    - Trỏ đến phần tử tiếp theo của chuỗi ($t5++).
- Lặp lại vòng lặp cho đến khi gặp dấu phẩy hoặc NULL.
- String2Array: Chuyển từng ký tự thành số rồi cộng vào thành 1 số hoàn chỉnh.
    - Gán $v0 = s[i-1] (để kiểm tra sau).
    - Vòng lặp (str2a_loop):
        - Lấy từng ký tự ra khỏi stack (Lấy từ hàng đơn vị rồi tăng dần lên) gán cho $s6.
        - Chuyển $s6 từ ký tự thành số bằng cách trừ đi ‘0’.
        - Nhân $s6 với 10^k ($s5) (k bắt đầu từ 0 rồi tăng dần lên).
        - Cộng tất cả vào $s2.
        - Rồi nhân $s5 với 10 ($10^{k+1}$).
        - Trừ số chữ số đi 1 ($s1--).
        - Và lặp lại vòng lặp cho đến khi số chữ số $s1 ≤ 0.
- Sau khi chuyển ký tự thành số xong (số chuyển xong được lưu trong $s2) ta phải lưu số đó vào mảng.
- Sau đó trỏ mảng và chuỗi đến phần tử tiếp theo.
- Lúc này nếu $v0 = ‘\0’ (NULL) thì kết thúc hàm StringSolve.
- Không thì tiếp túc lặp lại từ khởi tạo các giá trị cho đến khi gặp ký tự NULL.

getVALUE: Lấy giá trị từ mảng

Input: $s2: Vị trí trong mảng (i)

$s0: Địa chỉ gốc của mảng

Output: $s1: Giá trị phần tử tại vị trí $s2 (a[i])

GO: Di chuyển Marsbot

STOP: Dừng Marsbot

TRACK: Để lại vết

UNTRACK: Không để lại vết

ROTATE: Quay Marsbot

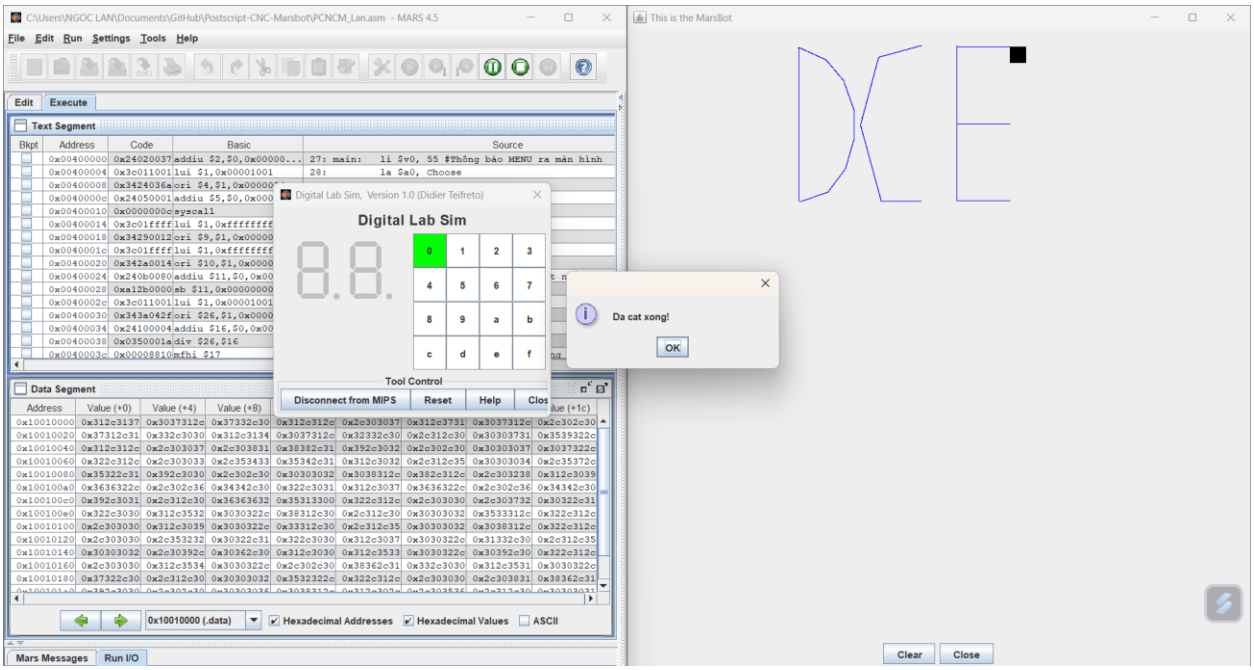
Input: $a0: Góc quay từ 0 đến 359 với:

- 0 : North (up)

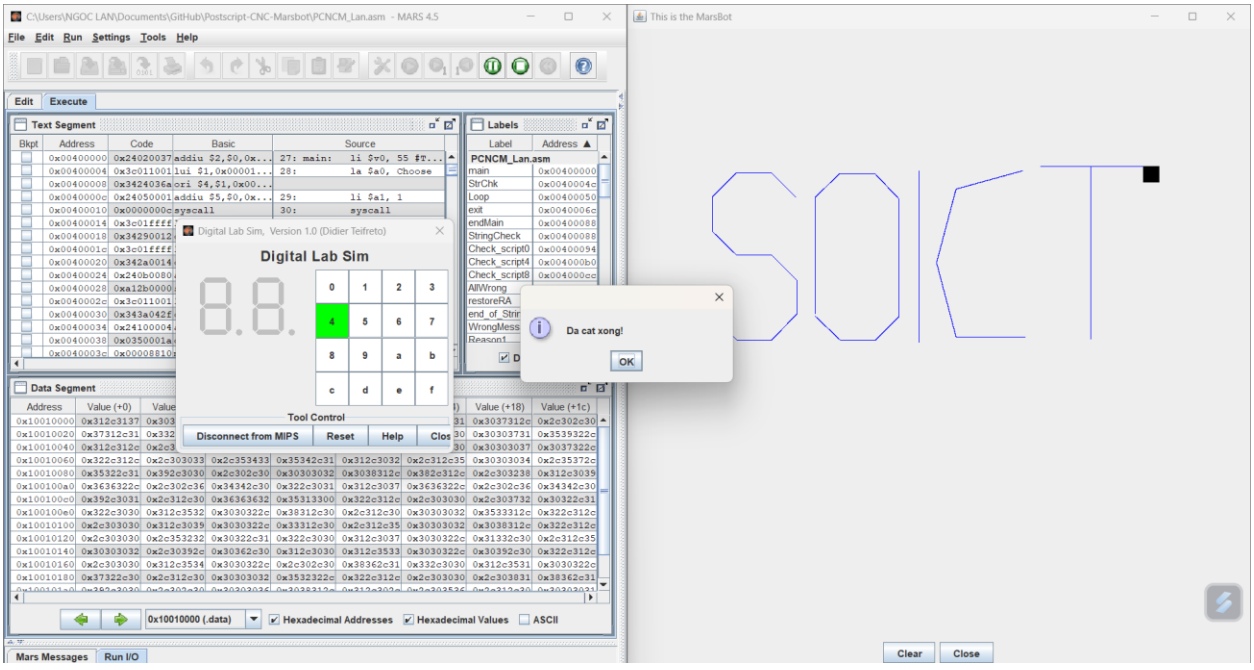
- 90: East (right)
- 180: South (down)
- 270: West (left)

4. Kết quả

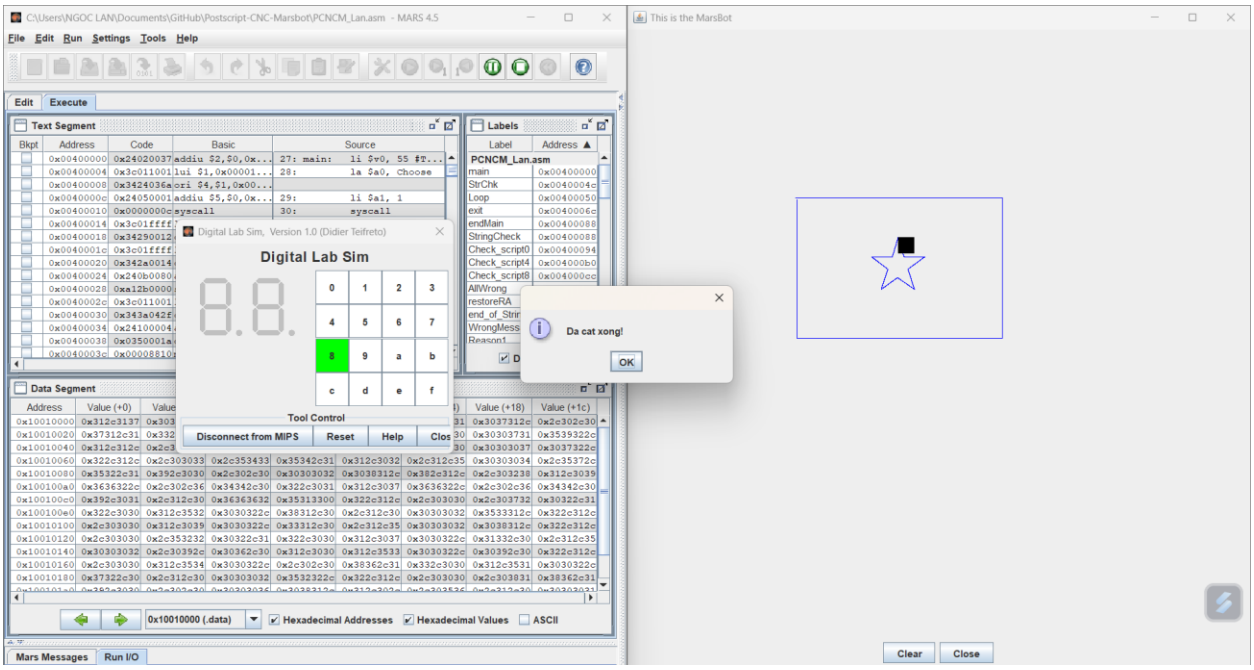
a. Postscript 0: DCE



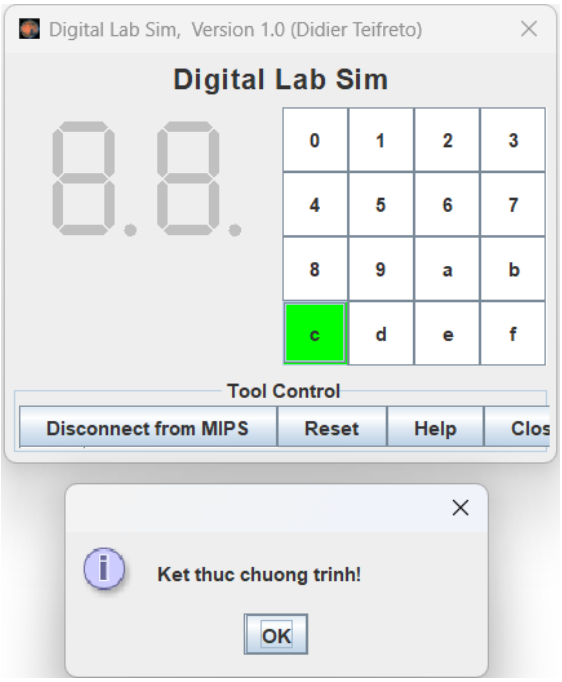
b. Postscript 4: SOICT



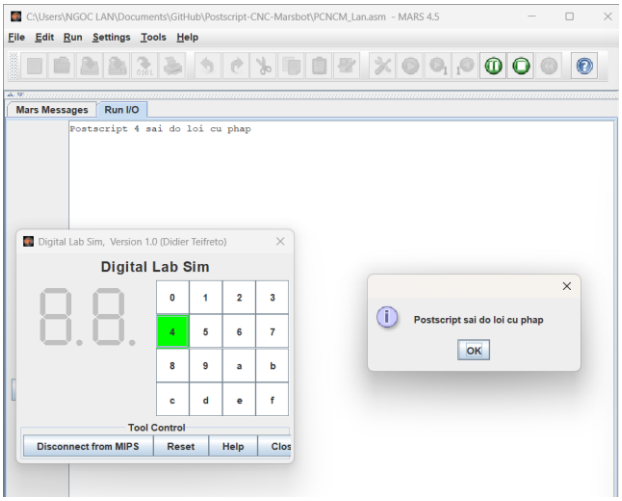
c. Postscript 8: Cờ Việt Nam



d. Khi ấn phím C

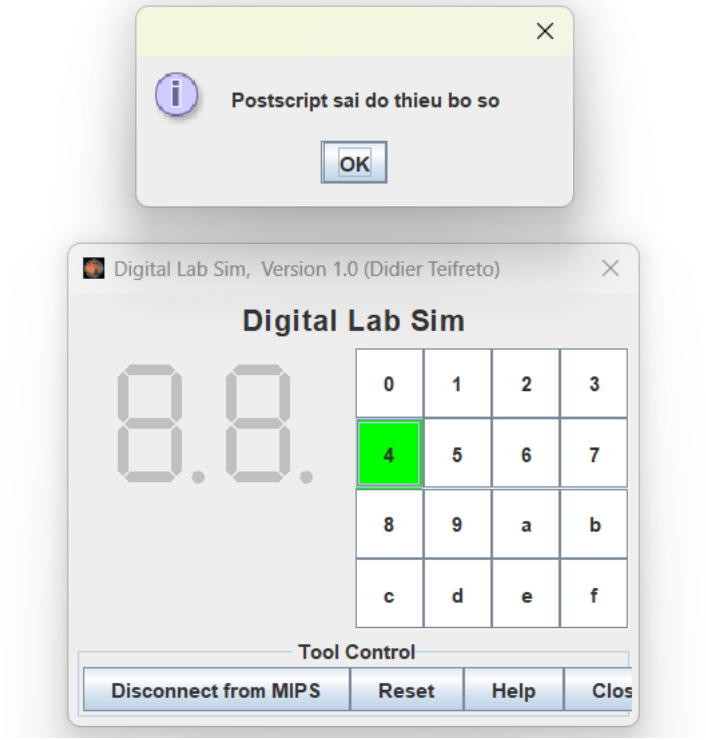


e. Lỗi cú pháp



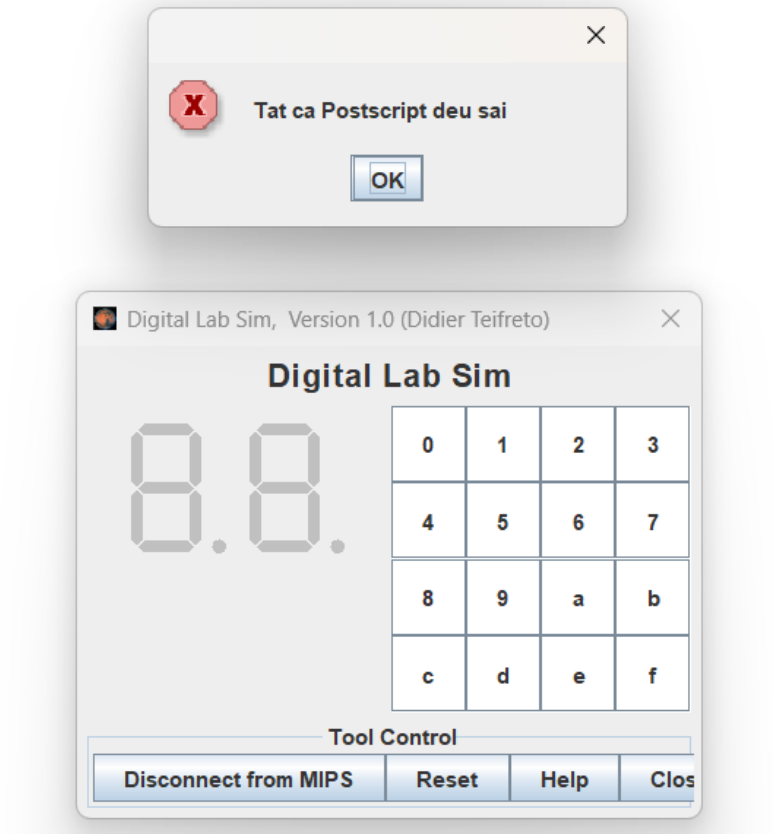
f. Lỗi thiếu bộ số

```
Postscript 4 sai do thieu bo so
```

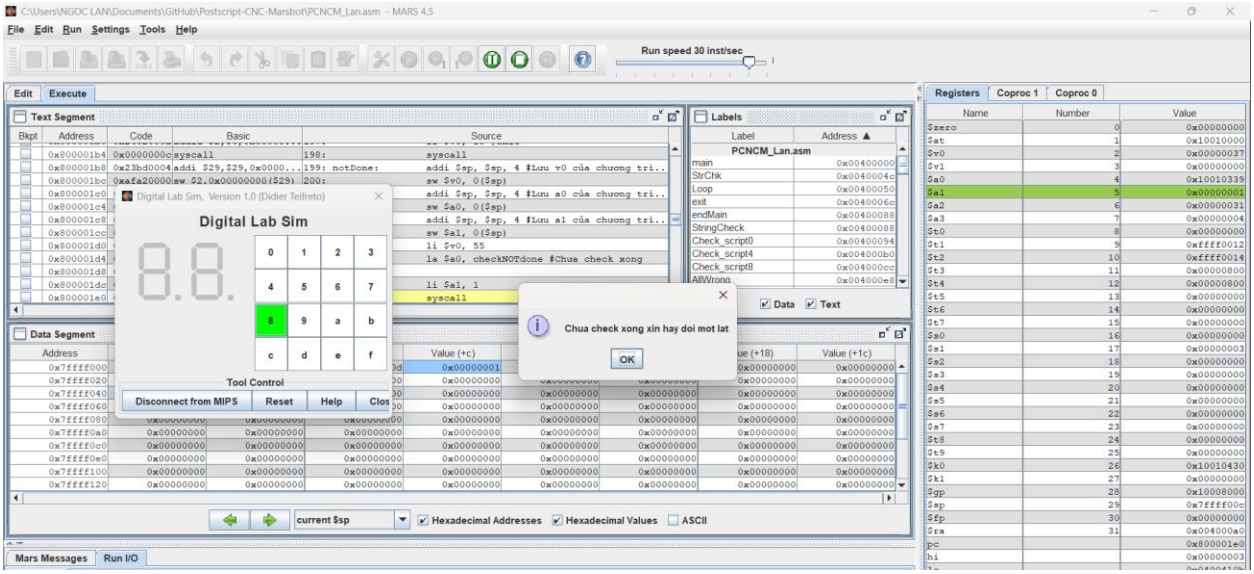


g. Lỗi tất cả các Postscript đều sai

```
Postscript 0 sai do loi cu phap
Postscript 4 sai do thieu bo so
Postscript 8 sai do loi cu phap
```



h. Ấn phím khi chưa check xong



PHẦN 2: Nguyễn Trung Sơn - 20226124

1. Đề bài (Đề 2)

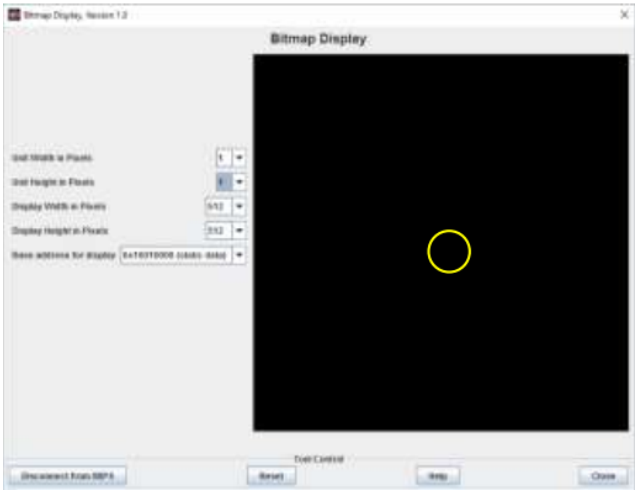
Vẽ hình trên màn hình Bitmap

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

Yêu cầu:

- Thiết lập màn hình ở kích thước 512x512. Kích thước pixel 1x1.
- Chiều di chuyển phụ thuộc vào phím người dùng bấm, gồm có (di chuyển lên (W), di chuyển xuống (S), sang trái (A), sang phải (D), tăng tốc độ (Z), giảm tốc độ (X) trong bộ giả lập Keyboard and Display MMIO Simulator).
- Vị trí bóng ban đầu ở giữa màn hình.

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



2. Mã nguồn

```
.eqv SCREEN         0x10010000
.eqv YELLOW         0x00FFFF66
.eqv BACKGROUND     0x00000000
# Thiet lap ky tu
.eqv KEY_A          0x00000061      # di chuyen sang trai
```

```
.eqv KEY_D              0x00000064      # di chuyen sang phai
.eqv KEY_S              0x00000073      # di chuyen xuong duoi
.eqv KEY_W              0x00000077      # di chuyen len tren
.eqv KEY_Z              0x0000007A      # tang toc do di chuyen
.eqv KEY_X              0x00000078      # giam toc do di chuyen
.eqv KEY_ENTER          0x0000000A      # chuong trinh dung lai
# thiet lap khoang cach giua hai duong tron
.eqv khoang_cach        20
.eqv KEY_CODE           0xFFFF0004
.eqv KEY_READY          0xFFFF0000
#=============================================================
.data
        Array: .space 512  # cap bo nho luu toa do cac diem cua duong tron
.text
        li      $s0, 256    # x = 256 khoi tao toa do x ban dau cua tam duong tron
        li      $s1, 256    # y = 256 khoi tao toa do y ban dau cua tam duong tron
        li      $s2, 25     # R = 25  R la ban kinh cua duong tron
        li      $s3, 512    # SCREEN_WIDTH = 512 chieu rong man hinh
        li      $s4, 512    # SCREEN_HEIGHT = 512chieu dai man hinh
        li      $s5, YELLOW             # duong tron co mau vang
        li      $t6, khoang_cach        # Khoang cach giua cac hinh tron
        li      $t7, 0
        # dx = 0 gia tri dich chuyen theo chieu ngang tai thoi diem hien tai
        li      $t8, 0
        # dy = 0 gia tri dich chuyen theo chieu doc tai thoi diem hien tai
        li      $t9, 50                 # Thanh ghi luu tru thoi gian delay
```

```
#=====================================================
# HAM KHOI TAO TOA DO DUONG TRON
#=====================================================
khoi_tao:
        li      $t0, 0                  # khoi tao i = 0
        la      $t5, Array              # luu dia chi cua mang vao thanh ghi $t5
loop:  # tao vong lap chay tu i den R
        slt     $v0, $t0, $s2           # v0 = 1 neu  i<R
        beq     $v0, $zero, ket_thuc    # v0 = 0 <=> i >= R thi nhay den ket_thuc
        mul     $s6, $s2, $s2           # s6 = R*R = R^2
        mul     $t3, $t0, $t0           # t3 = i*I = i^2
        sub     $t3, $s6, $t3           # $t3 = R^2 - i^2
        move    $v1, $t3                # v1 = t3
        jal     sqrt                    # nhay den ham tinh can cua t3
        sw      $a0, 0($t5)
        # lay gia tri cua thanh ghi a0 = sqrt(R^2 - i^2) luu vao mang du lieu
        addi    $t0, $t0, 1             # i = i+1
        add     $t5, $t5, 4             # di den vi tri tiep theo cua mang du lieu
        j       loop
ket_thuc:
#-------------------------------------------------------------
# tao ham lam cho chuong trinh dung chay trong 1 khoang thoi gian
# thoi gian co gia tri luu o thanh ghi %r khi goi ham
.macro delay(%r)
        addi    $a0, %r, 0
        li      $v0, 32
        syscall
```

```
.end_macro
# tao ham de dat lai mau va ve them duong tron o vi tri moi
# dia chi cua mau luu o thanh ghi %color khi goi ham
.macro datmauveduongtron(%color)
        li      $s5, %color
        jal     ham_ve_duong_tron
.end_macro

#=========================================================
# HAM NHAP DU LIEU TU BAN PHIM
#=========================================================
Start:
doc_ky_tu:
        lw      $k1, KEY_READY      # kiem tra da nhap ki tu nao chua?
        beqz    $k1, check_vi_tri
        # Neu k1! = 0 => da nhap ky tu thi nhay den ham kiem tra vi tri
        lw      $k0, KEY_CODE       # thanh ghi k0 luu gia tri ki tu nhap vao
        beq     $k0, KEY_A, case_a  # di chuyen qua trai
        beq     $k0, KEY_D, case_d  # di chuyen qua phai
        beq     $k0, KEY_S, case_s  # di chuyen xuong duoi
        beq     $k0, KEY_W, case_w  # di chuyen len tren
        beq     $k0, KEY_X, case_x  # Giam toc do
        beq     $k0, KEY_Z, case_z  # Tang toc do
        beq     $k0, KEY_ENTER, case_enter # Dung chuong trinh
        j       check_vi_tri
        nop
case_a:
```

```
        jal     di_sang_trai
        j       check_vi_tri
case_d:
        jal     di_sang_phai
        j       check_vi_tri
case_s:
        jal     di_chuyen_xuong
        j       check_vi_tri
case_w:
        jal     di_chuyen_len
        j       check_vi_tri
case_z:
        addi    $t9,$t9,-10
        j       check_vi_tri
case_x:
        addi    $t9,$t9,20
        j       check_vi_tri
case_enter:
        j       endProgram
endProgram:
        li      $v0, 10
        syscall

#=====================================================
# HAM KIEM TRA VI TRI
#=====================================================
check_vi_tri:
```

```
phia_ben_phai:
        add     $v0, $s0, $s2           # v0 = x0 + R, toa do tam hien tai+ ban kinh
        add     $v0, $v0,$t7
        # neu x0 + R + khoang_cach > 512 thi nhay den ham di_sang_trai
        slt     $v1, $v0,$s3            # v1 = 1 neu v0 < 512
        bne     $v1, $zero,phia_ben_trai
        jal     di_sang_trai
        nop
phia_ben_trai:
        sub     $v0, $s0, $s2           # v0 = x0 - R
        add     $v0, $v0, $t7
        # neu x0 - R + khoang_cach < 0 thi nhay den ham di_sang_phai
        slt     $v1, $v0, $zero         # v1 = 1 neu v0 < 0
        beq     $v1, $zero, phia_tren
        jal     di_sang_phai
        nop
phia_tren:
        sub     $v0, $s1, $s2           # v0 = y0 - R
        add     $v0, $v0, $t8
        # neu y0 - R + khoang_cach < 0 thi nhay den ham di chuyen len
        slt     $v1, $v0, $zero         # v1 = 1 neu v0 < 0
        beq     $v1, $zero, phia_duoi
        jal     di_chuyen_xuong
        nop
phia_duoi:
        add     $v0, $s1, $s2           # v0 = y0 + R
        add     $v0, $v0, $t8
```

```
        # neu y0 + R + khoang_cach > 512 thi nhay den ham di_chuyen_xuong
        slt     $v1, $v0, $s4           # v1 = 1 neu v0 < 512
        bne     $v1, $zero, draw
        jal     di_chuyen_len
        nop

#=========================================================
# HAM VE DUONG TRON
#=========================================================

draw:
        datmauveduongtron(BACKGROUND)       # ve duong tron trung mau nen
        add     $s0, $s0, $t7       # Cap nhat toa do x cua duong tron
        add     $s1, $s1, $t8       # cap nhat toa do y cua duong tron

        datmauveduongtron(YELLOW)           # ve duong tron moi mau vang
        delay($t9)                  # dung 1 khoang thoi gian roi ve duong tron moi
        j Start

ham_ve_duong_tron:
        add     $sp, $sp, -4
        sw      $ra, 0($sp)
        li      $t0, 0              # khoi tao bien i = 0
loop_ve_duong_tron:
        slt     $v0, $t0, $s2       # v0 = 1 neu i < R
        beq     $v0, $zero, ket_thuc_ve # neuv0 = 0 <=> I >= R => ket_thuc_ve
        sll     $t5, $t0, 2
```

```
        lw      $t3, Array($t5)     # nap sqrt(R^2-i^2) luu o Array vao thanh ghi $t3
        move    $a0, $t0            # i = $t0 = $a0
        move    $a1, $t3            # j = $t3 = $a1
        jal     ve_diem # ve 2 diem (x0+i, y0+j), (x0+j, y0+i) tren phan tu thu I
        sub     $a1, $zero, $t3
        jal     ve_diem # ve 2 diem (x0+i, y0-j), (x0+j, y0-i) tren phan tu thu II
        sub     $a0, $zero, $t0
        jal     ve_diem # ve 2 diem (x0-i, y0-j), (x0-j, y0-i) tren phan tu thu III
        add     $a1, $zero, $t3
        jal     ve_diem # ve 2 diem (x0-i, y0+j), (x0-j, y0+i) tren phan tu thu IV
        addi    $t0, $t0, 1
        j       loop_ve_duong_tron
 ket_thuc_ve:
        lw      $ra, 0($sp)
        add     $sp, $sp, 0
        jr      $ra
# Ham ve diem tren duong tron
 ve_diem:
        add     $t1, $s0, $a0           # xi = x0 + i
        add     $t4, $s1, $a1           # yi = y0 + j
        mul     $t2, $t4, $s3           # yi * SCREEN_WIDTH
        add     $t1, $t1, $t2
        # yi * SCREEN_WIDTH + xi (Toa do 1 chieu cua diem anh)
        sll     $t1, $t1, 2             # dia chi tuong doi cua diem anh
        sw      $s5, SCREEN($t1)        # ve anh
        add     $t1, $s0, $a1           # xi = x0 + j
        add     $t4, $s1, $a0           # yi = y0 + i
```

```
        mul     $t2, $t4, $s3               # yi * SCREEN_WIDTH
        add     $t1, $t1, $t2
        # yi * SCREEN_WIDTH + xi (Toa do 1 chieu cua diem anh)
        sll     $t1, $t1, 2                 # dia chi tuong doi cua diem anh
        sw      $s5, SCREEN($t1)            # ve anh
        jr      $ra
#-------------------------------------------------------------------------------------------
# Ham tinh can cua t3
sqrt:
        mtc1  $v1, $f1              # dua gia tri trong thanh ghi v0 vao thanh ghi f0
        cvt.s.w $f1, $f1
        # chuyen gia tri cua f0 tuong duong voi gia tri so nguyen 32 bit
        sqrt.s  $f1, $f1            # Tinh can bac hai cua gia tri thanh ghi f0
        cvt.w.s $f1, $f1            # Chuyen f0 ve dang 32-bit
        mfc1  $a0, $f1              # dat gia tri thanh ghi a0 = f0
        jr      $ra

#=========================================================
# CAC HAM DI CHUYEN
#=========================================================
di_sang_trai:
        sub     $t7, $zero, $t6
        li      $t8, 0
        jr      $ra
di_sang_phai:
        add     $t7, $zero, $t6
        li      $t8, 0
```

```
        jr      $ra
di_chuyen_len:
        li      $t7, 0
        sub     $t8, $zero, $t6
        jr      $ra
di_chuyen_xuong:
        li      $t7, 0
        add     $t8, $zero, $t6
        jr      $ra
```

3. Phân tích

3.1. Cách thực thi chương trình

Bước 1: Chúng ta sẽ kết nối đồng thời chương trình với keyboard và bitmap display dịch và chạy chương trình.

Bước 2: Điều khiển bằng các phím “a”, “b”, “c”, “d”, tăng tốc phím “z”, giảm tốc phím “x”. Nếu nhập phím khác sẽ ko thực hiện gì cả. Enter để kết thúc chương trình.

Bước 3: Hình tròn sẽ nảy khi có va chạm với cạnh.

3.2. Giải thích các hàm

Cách hoạt động chi tiết của bài toán:

Chúng ta sẽ khai báo các nút cần thiết để chạy chương trình theo yêu cầu bài toán

- Khởi tạo tọa độ ban đầu là (x, y) = (256, 256) bán kính đường tròn là 25 khoảng cách các đường tròn, chiều dài và rộng của màn hình 512x512, hình tròn màu vàng và kch khi vẽ liên tục là 20 (thanh ghi ($s0 - $s5, $t6)).
- Thanh ghi $t7 và $t8 ghi lại giá trị dịch chuyển theo chiều ngang dọc (hiện tại cấp là 0) và thanh ghi $t9 lưu thời gian delay.

1. Hàm khởi tạo tọa độ đường tròn

Mục đích: Xác định tọa độ các điểm của đường tròn và lưu vào bộ nhớ trong mảng array phục vụ việc vẽ đường tròn ở phần sau

Chúng ta sẽ khởi tạo i, chạy vòng lặp từ i → R dừng khi i = R tính sqrt(R^2 - i^2), và lưu vào mảng dữ liệu.

2. Hàm delay

Mục đích: Khi bóng dịch chuyển từ vị trí này qua vị trí khác thì chương trình sẽ dừng khoảng 50ms sau đó mới vẽ đường tròn ở vị trí mới. Thời gian delay lưu ở thanh ghi %r

3. Hàm datmauveduongtron

Mục đích: Khi bóng dịch chuyển đi, thì mình sẽ vẽ đường tròn cũ màu vàng thay bằng màu trùng với màu đen của màn hình, còn đường tròn mới màu vàng sẽ được vẽ ở vị trí khác

4. Hàm nhập dữ liệu từ bàn phím

Mô tả: Có 7 ký tự sẽ xuất hiện trong bài toán

Ký tự chữ a: Di chuyển qua bên trái

Ký tự chữ d: Di chuyển qua bên phải

Ký tự chữ s: Di chuyển xuống dưới

Ký tự chữ w: Di chuyển lên trên

Ký tự chữ x: Giảm tốc độ dịch chuyển của bóng

Ký tự chữ z: Tăng tốc độ dịch chuyển của bóng

Enter: Kết thúc chương trình

Chúng ta sẽ có 7 case cho 7 nút bấm. Và sẽ xử lí việc mà gặp cạnh như sau:

- Ở hàm phia_ben_phai nếu x0 + R + khoang_cach > 512 thi nhảy đến hàm di_sang_trai.
- Ở hàm phia_ben_trai nếu x0 - R + khoang_cach < 0 thi nhảy đến hàm di_sang_phai.
- Ở hàm phia_tren nếu y0 - R + khoang_cach < 0 thi nhảy đến hàm di_chuyen_len.
- Ở hàm phia_duoi nếu y0 + R + khoang_cach > 512 thi nhảy đến hàm di_chuyen_xuong.

5. Hàm di chuyển vị trí

6. Hàm vẽ điểm trên đường tròn

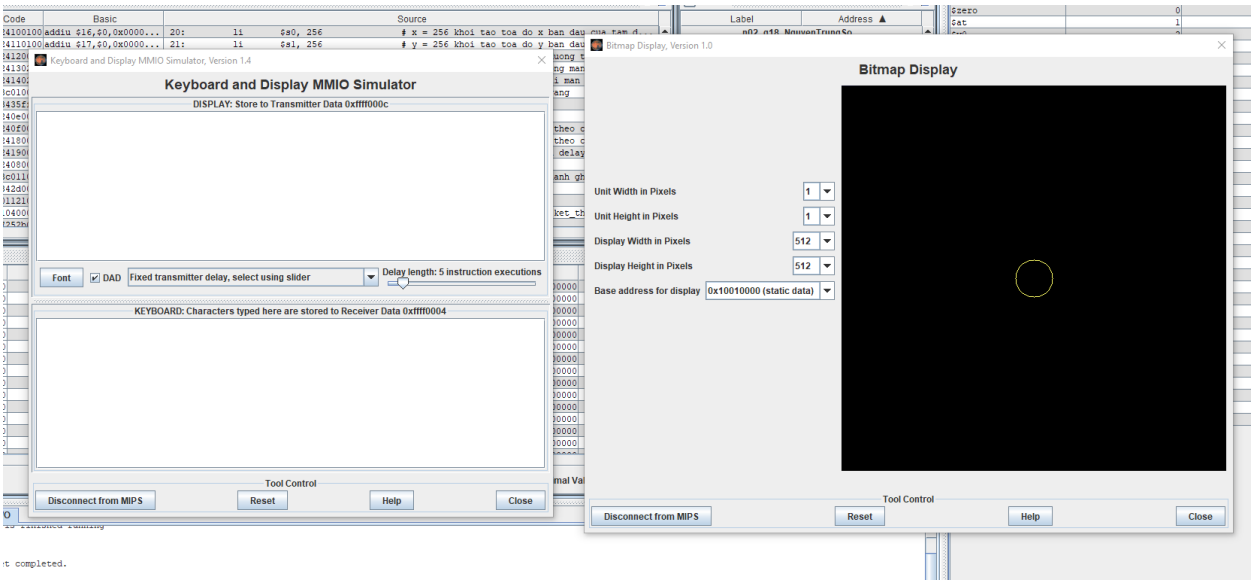
- Tính tọa độ x của điểm cần vẽ xi bằng cách cộng i (được lưu trong $a0) với tọa độ x của tâm x0 (được lưu trong $s0).
- Tính tọa độ y của điểm cần vẽ yi bằng cách cộng j (được lưu trong $a1) với tọa độ y của tâm y0 (được lưu trong $s1).
- Nhân tọa độ y của điểm cần vẽ với độ rộng màn hình để chuyển đổi tọa độ 2D thành tọa độ 1D trong bộ nhớ video.
- Thêm tọa độ x của điểm cần vẽ để có được tọa độ 1D của điểm ảnh trong bộ nhớ video.
- Dịch trái tọa độ 1D này 2 bit để nhân với 4, vì mỗi điểm ảnh chiếm 4 byte trong bộ nhớ.
- Lưu giá trị màu (được lưu trong $s5) vào bộ nhớ video tại vị trí đã tính toán, tức là vẽ điểm ảnh này lên màn hình.
- Tính tọa độ x của điểm đối xứng bằng cách cộng j (được lưu trong $a1) với tọa độ x của tâm x0 (được lưu trong $s0).
- Tính tọa độ y của điểm đối xứng bằng cách cộng i (được lưu trong $a0) với tọa độ y của tâm y0 (được lưu trong $s1).
- nh tọa độ y của điểm đối xứng bằng cách cộng i (được lưu trong $a0) với tọa độ y của tâm y0 (được lưu trong $s1).
- Thêm tọa độ x của điểm đối xứng để có được tọa độ 1D của điểm ảnh trong bộ nhớ.
- Dịch trái tọa độ 1D này 2 bit để nhân với 4, vì mỗi điểm ảnh chiếm 4 byte trong bộ nhớ.
- Lưu giá trị màu (được lưu trong $s5) vào bộ nhớ video tại vị trí đã tính toán, tức là vẽ điểm ảnh này lên màn hình.

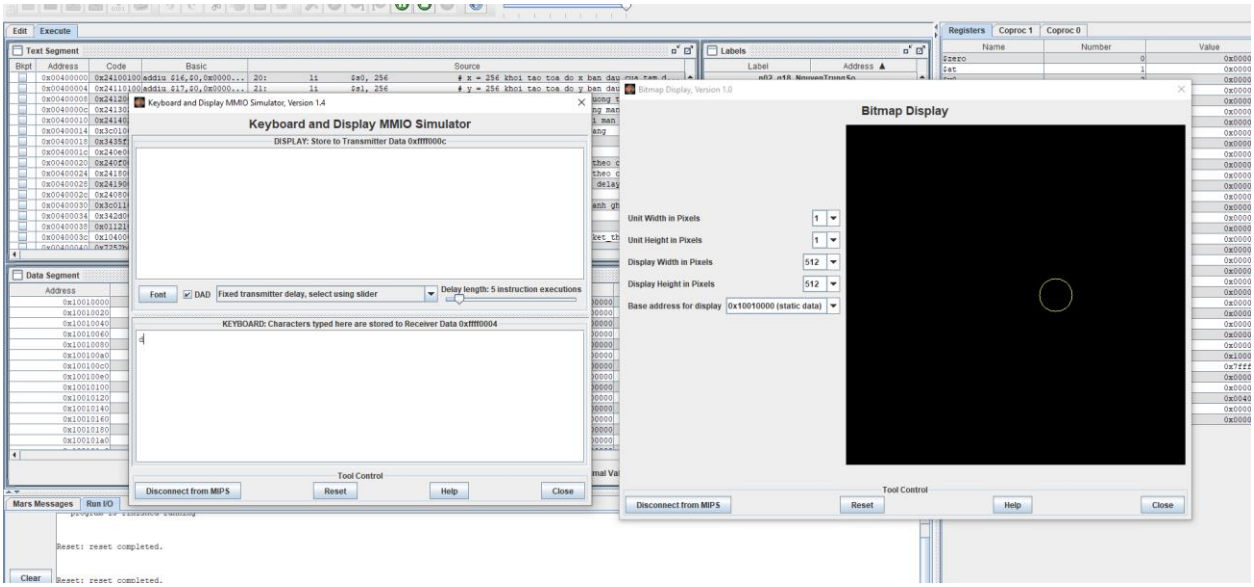
Hàm vẽ điểm của đường tròn sẽ chia đường tròn thành 4 phần, phần tư thứ I, II, III, IV và áp dụng thuật toán tìm tọa độ điểm để vẽ được toàn bộ đường tròn.

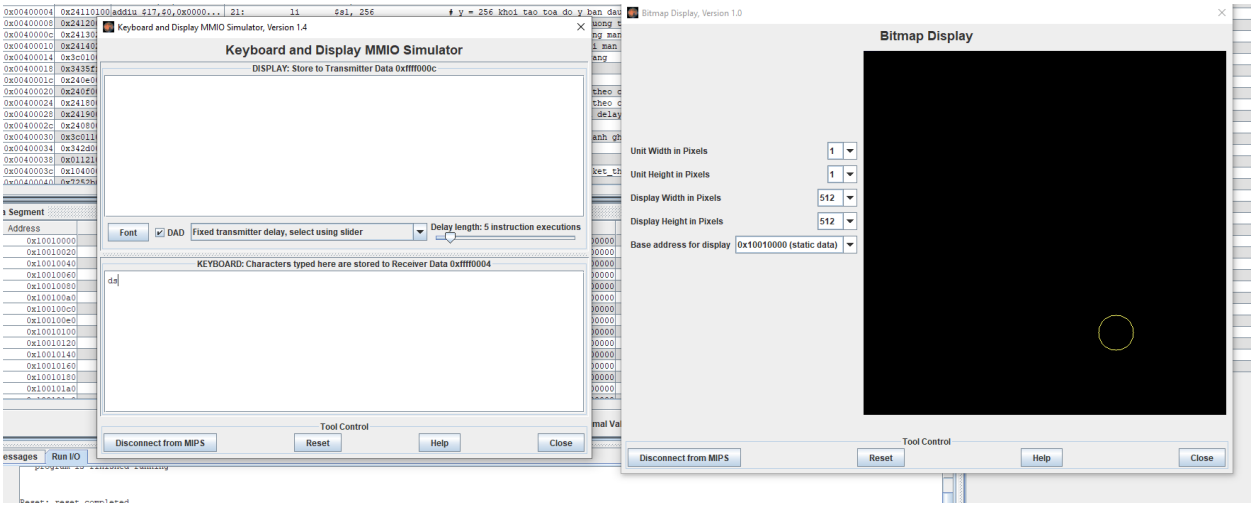
7. Hàm vẽ đường tròn

- Nạp sqrt(R^2 - i^2) ở mảng vào thanh ghi $t3.
- Vòng lặp loop để thực hiện liên tục việc vẽ điểm trên 4 góc phần tư.
- Hàm tính căn bậc 2.

4. Kết quả


Keyboard and Display MMIO Simulator, Version 1.4
Keyboard and Display MMIO Simulator
DISPLAY: Store to Transmitter Data 0xffff000c
Font
DAD
Fixed transmitter delay, select using slider
Delay length: 5 instruction executions
KEYBOARD: Characters typed here are stored to Receiver Data 0xffff0004
Tool Control
Disconnect from MIPS
Reset
Help
Close
Bitmap Display, Version 1.0
Bitmap Display
Unit Width in Pixels
Unit Height in Pixels
Display Width in Pixels
Display Height in Pixels
Base address for display
0x10010000 (static data)
Tool Control
Disconnect from MIPS
Reset
Help
Close


Edit
Execute
Text Segment
Data Segment
Labels
Registers
Coproc 1
Coproc 0
Keyboard and Display MMIO Simulator, Version 1.4
Keyboard and Display MMIO Simulator
DISPLAY: Store to Transmitter Data 0xffff000c
Font
DAD
Fixed transmitter delay, select using slider
Delay length: 5 instruction executions
KEYBOARD: Characters typed here are stored to Receiver Data 0xffff0004
Tool Control
Disconnect from MIPS
Reset
Help
Close
Bitmap Display, Version 1.0
Bitmap Display
Unit Width in Pixels
Unit Height in Pixels
Display Width in Pixels
Display Height in Pixels
Base address for display
0x10010000 (static data)
Tool Control
Disconnect from MIPS
Reset
Help
Close
Mars Messages
Run I/O
Clear
Reset: reset completed.
Reset: reset completed.



Enter để dừng