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


# BÁO CÁO FINAL PROJECT

**HỌC PHẦN: Thực hành kiến trúc máy tính**

**Mã học phần: IT3280**

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

Nhóm sinh viên thực hiện:

| | Họ và tên | MSSV |
|---|---|---|
| 1 | Vũ Ngọc Đức | 20225816 |
| 2 | Hoàng Trường Giang | 20225710 |


**Hà Nội, ngày 12 tháng 6 năm 2024**

# Phần A: Chủ đề 1 – Curiosity Marsbot
# Họ và tên: Vũ Ngọc Đức
# MSSV: 20225816

## 1. Phân tích bài toán

- Xe tự hành Curiosity Marsbot chạy trên sao Hỏa, được vận hành từ xa bởi các lập trình viên trên Trái Đất bằng cách gửi các mã điều khiển.
- Các mã điều khiển được nhập từ Digital Lab Sim => cần lưu trữ các mã quét được từ Digital Lab Sim.
- Sau khi nhận mã điều khiển cần nhập lệnh kích hoạt từ Keyboard & Display MMIO Simulator:

  + Enter: Kết thúc nhập mã và yêu cầu Marsbot thực thi.
  => Trước khi thực hiện cần kiểm tra xem mã có trong kịch bản không?
  + Delete: Xóa toàn bộ mã điều khiển đang nhập.
  + Space: Lặp lại lệnh đã thực hiện trước đó
- Các hành động như di chuyển, dừng, rẽ trái,… thì chỉ cần ra lệnh trực tiếp cho Marsbot thực hiện.
- Đặc biệt có hành động quay về theo lộ trình ngược lại thì cần phải lưu trữ lịch sử di chuyển cho Marsbot.

## 2. Cách thực hiện

- Bước 1: Khi người dùng nhập 1 ký tự từ Digital Lab Sim sẽ tạo ra lệnh ngắt để lưu ký tự đó vào bộ nhớ, cứ như vậy cho tới khi người dùng nhập lệnh kích hoạt

  => Có được mã điều khiển
- Bước 2: Người dùng nhập lệnh kích hoạt thông qua Keyboard & Display MMIO Simulator suy ra cần kiểm tra liên tục xem ký tự Enter, Delete, Space có được nhập hay không ?

  + Nếu Enter được nhập chuyển sang Bước 3;
  + Nếu Delete được nhập chuyển sang Bước 4;
  + Nếu Space được nhập chuyển sang Bước 5;
  + Nếu không thì tiếp tục Bước 2
- Bước 3: Hiển thị mã điều khiển ra console
  + Kiểm tra mã điều khiển có trong kịch bản không?
  => nếu có: thực hiện hành động tương ứng

=> nếu không: in mã Error

- Bước 4: Xóa lưu trữ mã điều khiển trong bộ nhớ.
- Bước 5: Lặp lại các lệnh vừa thực hiện

## 3. Các hàm thực hiện

### Hàm main

Các nhãn và công việc tương ứng của từng nhãn trong hàm main như sau:

*start*: set góc đầu tiên của Marsbot là góc 90 độ (sang phải)
*printError*: in ra thông báo lỗi
*printCode*: in ra mã điều khiển vừa nhập vào
*repeatCode*: lặp lại mã điều khiển trước đó
*resetInput*: xóa mã điều khiển đã nhập để chuẩn bị cho mã tiếp theo
*waitForKey*: chờ phím được nhấn từ Digital Lab Sim
*readKey*: đọc ký tự được nhập vào từ Keyboard & Display MMIO Simulator
*checkCode*: kiểm tra mã điều khiển có hợp lệ về độ dài và khớp với một trong các mã đã được quy ước

*go, stop, turnLeft, turnRight, track, untrack, goBack*: thực thi mã điều khiển

### Các hàm cho Marsbot

Các hàm và chức năng tương ứng của từng hàm như sau:

*GO, STOP*: điều khiển Marsbot bắt đầu chuyển động (GO) hoặc dừng lại (STOP), lưu trạng thái đang chuyển động hay không vào isGoing
*ROTATE:* điều khiển Marsbot quay theo góc lưu ở a_cur
*TRACK, UNTRACK*: điều khiển Marsbot bắt đầu để lại vết (TRACK) hoặc dừng để lại vết (UNTRACK), lưu trạng thái đang ghi vết hay không vào isTracking
*saveHistory*: lưu tọa độ x, y và góc hiện tại trước khi Marsbot thực hiện lệnh ROTATE

### Các hàm để xử lý xâu

Các hàm và chức năng tương ứng của từng hàm như sau:

*strCmp*: so sánh xâu ở $s3 với mã điều khiển vừa nhập (curCode), trả về giá trị boolean ở $t0
*strClear*: xóa mã điều khiển vừa nhập (curCode)
*CopyPrevToCur*: sao chép mã trước vào địa chỉ mã hiện tại
*CopyCurToPrev:* sao chép mã hiện tại thành mã trước đó

## 4. Mã nguồn

```
# Định nghĩa Mars bot
.eqv   HEADING 0xffff8010          # Integer: An angle between 0 and 359
                    # 0 : Lên
                    # 90: Phải
                    # 180: Xuống
                    # 270: Trái
.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

# Định nghĩa Keyboard
.eqv   KEY_CODE 0xFFFF0004         # ASCII code from keyboard, 1 byte
.eqv   KEY_READY 0xFFFF0000        # = 1 if has a new keycode ?
                              # Auto clear after lw
# Địa chỉ Hexa Keyboard
.eqv IN_ADRESS_HEXA_KEYBOARD 0xFFFF0012
.eqv OUT_ADRESS_HEXA_KEYBOARD 0xFFFF0014

#================================================================
==================================
.data
# Lịch sử đường đi (trước khi đổi hướng)
        x_his:      .word 0 : 16       # 16: giá trị mỗi phần tử, tọa độ đổi hướng
(x; y)
        y_his:      .word 0 : 16
        a_his:      .word 0 : 16       # lịch sử góc
        l_his:      .word 4            # biến đếm độ dài
        a_cur:      .word 0            # góc hiện tại

        isGoing:        .word 0            # 1 -> chạy, 0 -> dừng
        isTracking:     .word 0            # 1 -> vết, 0 -> ngừng

        curCode:        .space 8           # lệnh nhập vào
        Length:         .word 0            # chiều dài lệnh
```

```
        prevCode:       .space 8        # lệnh trước đó

# Mã điều khiển
        MOVE_CODE:          .asciiz "1b4"               # chuyển động
        STOP_CODE:          .asciiz "c68"               # dừng
        TURN_LEFT_CODE:     .asciiz "444"               # rẽ trái
        TURN_RIGHT_CODE:    .asciiz "666"               # rẽ phải
        TRACK_CODE:         .asciiz "dad"               # tạo vết
        UNTRACK_CODE:       .asciiz "cbc"               # ngừng tạo vết
        BACK_CODE:          .asciiz "999"               # trở về
        error:              .asciiz "Khong ton tai lenh: "

#=================================================================
==================================
.text
        li $k0, KEY_CODE
        li $k1, KEY_READY

        li $t1, IN_ADRESS_HEXA_KEYBOARD         # ngắt Digital Lab Sim
        li $t3, 0x80                    # 1000 0000: bit 7 = 1 cho phép ngắt
        sb $t3, 0($t1)

start:
        addi    $t7, $zero, 4# vị trí bắt đầu: x = 0; y = 0; a = 90
        sw      $t7, l_his

        li      $t7, 90
        sw      $t7, a_cur          # a_cur = 90 -> hướng ban đầu: phải
        jal     ROTATE
        nop

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

        j       waitForKey

printError:
        li      $v0, 4
        la      $a0, error
        syscall
```

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

repeatCode:                             # quay lại lệnh trước đó
        jal CopyPrevToCur
        j checkCode

resetInput:
        jal     strClear
        nop

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

readKey:
        lw      $t6, 0($k0)             # $t6 = [$k0] = KEY_CODE
        beq     $t6, 0x7f , resetInput  # if $t6 == Delete -> Xóa toàn bộ lệnh
        beq     $t6, 0x20, repeatCode   # if $t6 == Space -> Lặp lại lệnh

        bne     $t6, 0xa, waitForKey    # if $t6 != Enter -> Chờ Enter
        nop
        bne     $t6, 0xa, waitForKey

checkCode:
        lw      $s2, Length
        bne     $s2, 3, printError      # chiều dài lệnh != 3 -> không tồn tại lệnh

        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, BACK_CODE
        jal     strCmp
        beq     $t0, 1, goBack
        nop

        j       printError

# Thực hiện lệnh
================================================================
==================
go:
        jal     CopyCurToPrev
        jal     GO
        j       printCode

stop:
        jal     CopyCurToPrev
        jal     STOP
        j       printCode

track:
        jal     CopyCurToPrev
        jal     TRACK
```

```
        j       printCode

untrack:
        jal     CopyCurToPrev
        jal     UNTRACK
        j       printCode

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

        jal     STOP
        nop
        jal     UNTRACK
        nop

        la      $s5, a_cur
        lw      $s6, 0($s5)         # $s6 = hướng hiện tại
        addi    $s6, $s6, 90        # tăng 90 độ -> phải
        sw      $s6, 0($s5)         # chuyển hướng

        jal     saveHistory
        jal     ROTATE

        beqz    $s0, noTrack1
        nop
        jal     TRACK
        noTrack1:
                nop

        beqz    $t7, noGo1
        nop
        jal     GO
        noGo1:
                nop

        j       printCode

turnLeft:
```

```
        jal     CopyCurToPrev
        lw      $t7, isGoing
        lw      $s0, isTracking

        jal     STOP
        nop
        jal     UNTRACK
        nop

        la      $s5, a_cur
        lw      $s6, 0($s5)         # $s6 = hướng hiện tại
        addi    $s6, $s6, -90       # giảm 90 độ -> trái
        sw      $s6, 0($s5)         # chuyển hướng

        jal     saveHistory
        jal     ROTATE

        beqz    $s0, noTrack2
        nop
        jal     TRACK
        noTrack2:
                nop

        beqz    $t7, noGo2
        nop
        jal     GO
        noGo2:
                nop

        j       printCode

goBack:
        jal     CopyCurToPrev
        li      $t7, IN_ADRESS_HEXA_KEYBOARD      # Không thể ngắt cho
đến khi dừng
        sb      $zero, 0($t7)

        lw      $s5, l_his                  # $s5 = biến đếm độ dài
        jal     UNTRACK
        jal     GO
```

```
goBack_turn:
        addi   $s5, $s5, -4                 # biến đếm độ dài --
        lw     $s6, a_his($s5)              # $s6 = a_his[l_his]
        addi   $s6, $s6, 180                # quay hướng ngược lại
        sw     $s6, a_cur
        jal    ROTATE
        nop

goBack_toTurningPoint:
        lw $t9, x_his($s5)                  # $t9 = x_his[i]
        get_x:
                li $t8, WHEREX              # $t8 = x_current
                lw $t8, 0($t8)

                bne $t8, $t9, get_x         # x_current == x_his[i]
                nop
                bne $t8, $t9, get_x

        lw $t7, y_his($s5)                  # $t9 = y_his[i]
        get_y:
                li $t8, WHEREY              # $t8 = y_current
                lw $t8, 0($t8)

                bne $t8, $t7, get_y         # y_current == y_his[i]
                nop
                bne $t8, $t7, get_y

        beq $s5, 0, goBack_end              # l_his == 0 -> end
        nop

        j goBack_turn                       # else -> loop

goBack_end:
        jal    STOP
        sw     $zero, a_cur                 # vị trí bắt đầu
        jal    ROTATE

        addi   $s5, $zero, 4
        sw     $s5, l_his                   # l_his = 0
```

```
        j       printCode

#=================================================================
==================================
saveHistory:
        addi    $sp, $sp, 4             # sao lưu (không bị thay đổi giá trị khi lấy thực
hiện)
        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_cur                      # s4 = a_cur

        lw      $t3, l_his                      # $t3 = l_his
        sw      $s1, x_his($t3)                 # lưu x, y, alpha
        sw      $s2, y_his($t3)
        sw      $s4, a_his($t3)

        addi    $t3, $t3, 4                     # cập nhật biến đếm độ dài
        sw      $t3, l_his

        lw      $s4, 0($sp)                     # khôi phục sao lưu
        addi    $sp, $sp, -4
        lw      $s3, 0($sp)
        addi    $sp, $sp, -4
```

```
        lw      $s2, 0($sp)
        addi    $sp, $sp, -4
        lw      $s1, 0($sp)
        addi    $sp, $sp, -4
        lw      $t4, 0($sp)
        addi    $sp, $sp, -4
        lw      $t3, 0($sp)
        addi    $sp, $sp, -4
        lw      $t2, 0($sp)
        addi    $sp, $sp, -4
        lw      $t1, 0($sp)
        addi    $sp, $sp, -4

        jr      $ra

# Cài đặt lệnh
=============================================================
====================
# GO ----------------------------------------------------------------------------------------
GO:
        addi    $sp, $sp, 4             # sao lưu
        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
        sw      $t7, isGoing            # isGoing = 1 -> di chuyển

        lw      $k0, 0($sp)             # khôi phục sao lưu
        addi    $sp, $sp, -4
        lw      $at, 0($sp)
        addi    $sp, $sp, -4

        jr      $ra

# STOP ---------------------------------------------------------------------------------------
```

```
STOP:
        addi   $sp, $sp, 4              # sao lưu
        sw     $at, 0($sp)

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

        sw     $zero, isGoing           # isGoing = 0 -> dừng

        lw     $at, 0($sp)              # khôi phục sao lưu
        addi   $sp, $sp, -4

        jr $ra

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

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

        addi   $s0, $zero, 1
        sw     $s0, isTracking          # isTracking = 1 -> tạo vết

        lw     $k0, 0($sp)              # khôi phục sao lưu
        addi   $sp, $sp, -4
        lw     $at, 0($sp)
        addi   $sp, $sp, -4

        jr $ra

# UNTRACK ----------------------------------------------------------------------------
--
UNTRACK:
        addi   $sp, $sp, 4              # sao lưu
        sw     $at, 0($sp)
```

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

        sw      $zero, isTracking   # isTracking = 0 -> ngừng tạo vết

        lw      $at, 0($sp)         # khôi phục sao lưu
        addi    $sp, $sp, -4

        jr      $ra

# ROTATE ------------------------------------------------------------------------
ROTATE:
        addi    $sp, $sp, 4         # sao lưu
        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_cur
        lw      $t3, 0($t2)
        sw      $t3, 0($t1)         # to rotate robot

        lw      $t3, 0($sp)         # khôi phục sao lưu
        addi    $sp, $sp, -4
        lw      $t2, 0($sp)
        addi    $sp, $sp, -4
        lw      $t1, 0($sp)
        addi    $sp, $sp, -4

        jr      $ra

# Các hàm xử lý xâu
================================================================
==============
# strCmp ------------------------------------------------------------------------
# Đầu vào: $s3 - địa chỉ lệnh
# Đầu ra: $t0 = 1 nếu chuỗi thỏa mãn, 0 nếu ngược lại
```

```
strCmp:
        addi   $sp, $sp, 4              # sao lưu
        sw     $t1, 0($sp)
        addi   $sp, $sp, 4
        sw     $s1, 0($sp)
        addi   $sp,$sp,4
        sw     $t2, 0($sp)
        addi   $sp, $sp, 4
        sw     $t3, 0($sp)

        addi   $t0, $zero, 0            # mặc định $t0 = 0
        addi   $t1, $zero, 0            # biến đếm $t1 = i = 0

strCmp_loop:
        beq    $t1, 3, strCmp_equal     # i = 3 -> thỏa mãn -> $t0 = 1
        nop

        lb     $t2, curCode($t1)        # $t2: lệnh nhập vào

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

        beq    $t2, $t3, strCmp_next    # if $t2 == $t3 -> loop
        nop

        j      strCmp_end

strCmp_next:
        addi   $t1, $t1, 1              # i++
        j      strCmp_loop

strCmp_equal:
        add    $t0, $zero, 1            # $t0 = 1

strCmp_end:
        lw     $t3, 0($sp)              # khôi phục sao lưu
        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                  # sao lưu
        sw     $t1, 0($sp)
        addi   $sp, $sp, 4
        sw     $t2, 0($sp)
        addi   $sp, $sp, 4
        sw     $s1, 0($sp)
        addi   $sp, $sp, 4
        sw     $t3, 0($sp)
        addi   $sp, $sp, 4
        sw     $s2, 0($sp)

        lw     $t3, Length                  # $t3 = Length
        addi   $t1, $zero, -1               # $t1 = -1 = i

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

        bne    $t1, $t3, strClear_loop      # if $t1 != 3 -> loop
        nop

        sw     $zero, Length                # Length = 0

strClear_end:
        lw     $s2, 0($sp)                  # khôi phục sao lưu
        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

# CopyPrevToCur ----------------------------------------------------------------------------
-
CopyPrevToCur:
        addi $sp, $sp, 4                    # sao lưu
        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
        la $s1, curCode                     # địa chỉ lệnh hiện tại
        la $s2, prevCode                    # địa chỉ lệnh trước đó

CopyPrevToCur_loop:
        beq $t2, 3, CopyPrevToCur_end       # if $t2 = 3 -> end

        lb $t1, 0($s2)                      # $t1 = lệnh trước[i]
        sb $t1, 0($s1)                      # lưu vào lệnh hiện tại[i]

        addi $s1, $s1, 1                    # $s1++
        addi $s2, $s2, 1                    # $s2++
        addi $t2, $t2, 1                    # $t2++

        j CopyPrevToCur_loop

CopyPrevToCur_end:
        li $t3, 3
        sw $t3, Length                      # chiều dài lệnh = 3

        lw $s2, 0($sp)                      # khôi phục sao lưu
```

```
        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

# CopyCurToPrev ----------------------------------------------------------------------
-
CopyCurToPrev:
        addi $sp, $sp, 4                # sao lưu
        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
        la $s1, prevCode                # địa chỉ lệnh trước đó
        la $s2, curCode                 # địa chỉ lệnh hiện tại

CopyCurToPrev_loop:
        beq $t2, 3, CopyCurToPrev_end   # if $t2 = 3 -> end

        lb $t1, 0($s2)                  # $t1 = lệnh hiện tại[i]
        sb $t1, 0($s1)                  # lưu vào lệnh trước[i]

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

```
        j CopyCurToPrev_loop

CopyCurToPrev_end:
        lw $s2, 0($sp)                  # khôi phục sao lưu
        addi $sp, $sp, -4
        lw $t3, 0($sp)
        addi $sp, $sp, -4
        lw $s1, 0($sp)
        addi $sp, $sp, -4
        lw $t2, 0($sp)
        addi $sp, $sp, -4
        lw $t1, 0($sp)
        addi $sp, $sp, -4

        jr $ra

#=================================================================
==================================
.ktext 0x80000180               # địa chỉ bắt đầu ngắt
backup:                         # sao lưu dữ liệu vào ngăn xếp
        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)
```

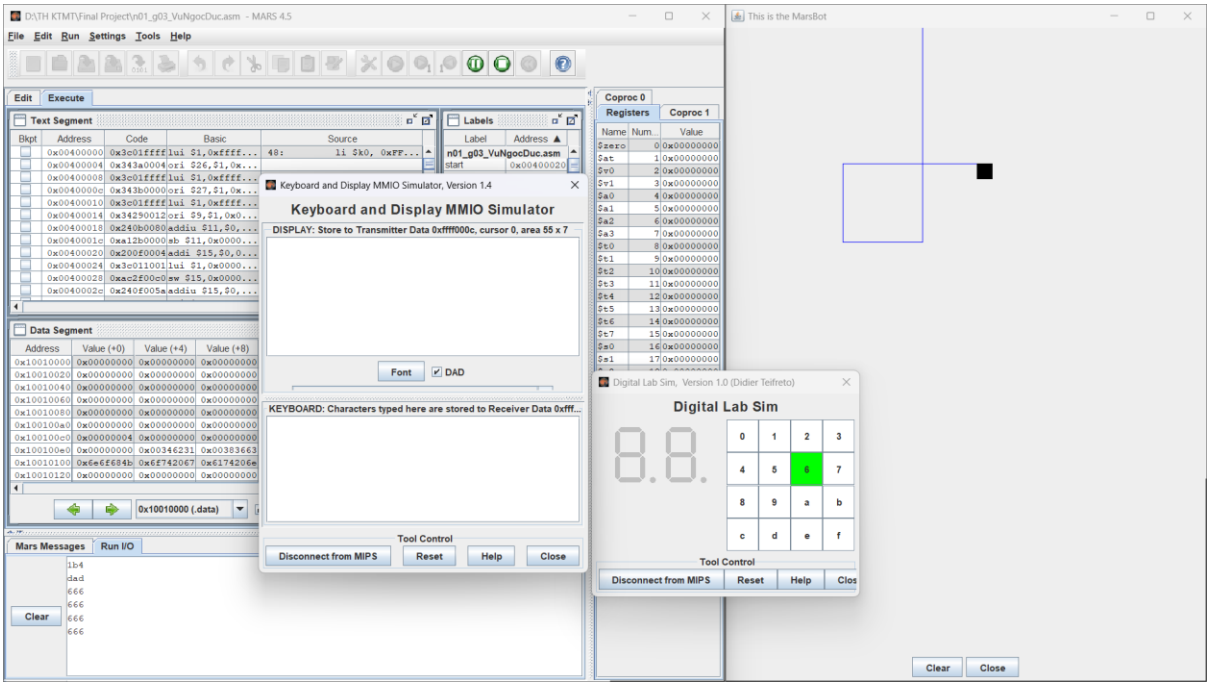
```
        addi    $sp, $sp, 4
        sw      $s3, 0($sp)

# đọc kí tự từ Digital Lab Sim   ------------------------------------------------------------
--
        li      $t1, IN_ADRESS_HEXA_KEYBOARD
        li      $t2, OUT_ADRESS_HEXA_KEYBOARD

# duyệt các hàng của Digital Lab Sim
# hàng 1
        li      $t3, 0x81
        sb      $t3, 0($t1)
        lbu     $a0, 0($t2)
        bnez    $a0, get
# hàng 2
        li      $t3, 0x82
        sb      $t3, 0($t1)
        lbu     $a0, 0($t2)
        bnez    $a0, get
# hàng 3
        li      $t3, 0x84
        sb      $t3, 0($t1)
        lbu     $a0, 0($t2)
        bnez    $a0, get
# hàng 4
        li      $t3, 0x88
        sb      $t3, 0($t1)
        lbu     $a0, 0($t2)
        bnez    $a0, get

get:
        beq     $a0, 0x11, case_0
        beq     $a0, 0x21, case_1
        beq     $a0, 0x41, case_2
        beq     $a0, 0x81, case_3
        beq     $a0, 0x12, case_4
        beq     $a0, 0x22, case_5
        beq     $a0, 0x42, case_6
        beq     $a0, 0x82, case_7
        beq     $a0, 0x14, case_8
```

```
        beq     $a0, 0x24, case_9
        beq     $a0, 0x44, case_a
        beq     $a0, 0x84, case_b
        beq     $a0, 0x18, case_c
        beq     $a0, 0x28, case_d
        beq     $a0, 0x48, case_e
        beq     $a0, 0x88, case_f

case_0:     li      $s0, '0'
        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, curCode
        la      $s2, Length
        lw      $s3, 0($s2)             # $s3 = chiều dài lệnh
        addi    $t4, $t4, -1            # $t4 = i

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

        addi    $s0, $zero, '\n'        # xuống dòng khi kết thúc 1 lệnh
        addi    $s1, $s1, 1
        sb      $s0, 0($s1)

        addi    $s3, $s3, 1
        sw      $s3, 0($s2)             # cập nhật chiều dài lệnh

#-------------------------------------------------------------------------------------------
next_pc:                                # tiếp tục lệnh tiếp theo sau khi ngắt
        mfc0    $at, $14        # $at = epc
        addi    $at, $at, 4     # $at = $at + 4
        mtc0    $at, $14        # epc = $at

#-------------------------------------------------------------------------------------------
restore:                                # khôi phục dữ liệu vào ngăn xếp
        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
eret                                    # trở về sau khi ngắt
```

## 5. Kết quả chạy thử



# Phần B: Chủ đề 8 – Mô phỏng ổ đĩa RAID 5
# Họ và tên: Hoàng Trường Giang
# MSSV: 20225710

## 1. Mô tả yêu cầu:

Hệ thống ổ đĩa RAID5 cần tối thiểu 3 ổ đĩa cứng, trong đó phần dữ liệu parity sẽ được chứa lần lượt lên 3 ổ đĩa như trong hình bên. Hãy viết chương trình mô phỏng hoạt động của RAID 5 với 3 ổ đĩa, với giả định rằng, mỗi block dữ liệu có 4 kí tự.
Giao diện như trong minh họa dưới. Giới hạn chuỗi kí tự nhập vào có độ dài là bội của 8.
Trong ví dụ sau, chuỗi kí tự nhập vào từ bàn phím (DCE.****ABCD1234HUSTHUST) sẽ được chia thành các block 4 byte. Block 4 byte đầu tiên "DCE." sẽ được lưu trên Disk 1, Block 4 byte tiếp theo "****" sẽ lưu trên Disk 2, dữ liệu trên Disk 3 sẽ là 4 byte parity được tính từ 2 block đầu tiên với mã ASCII là 6e='D' xor '*' ; 69='C' xor '*'; 6f='E' xor '*' ; 04='.' xor '*'

```
Nhap chuoi ki tu : DCE.****ABCD1234HUSTHUST
     Disk 1                Disk 2                Disk 3
 --------------        --------------        --------------
|     DCE.     |      |     ****     |      [[ 6e,69,6f,04]]
|     ABCD     |      [[ 70,70,70,70]]      |     1234     |
[[ 00,00,00,00]]      |     HUST     |      |     HUST     |
 --------------        --------------        --------------
```

## 2. Làm rõ yêu cầu:

- **Số lượng ổ đĩa**: Hệ thống RAID 5 yêu cầu tối thiểu 3 ổ đĩa.
- **Kích thước block**: Mỗi block dữ liệu có 4 ký tự.
- **Chuỗi nhập vào**: Chuỗi ký tự nhập vào phải có độ dài là bội số của 8.
- **Phân phối dữ liệu và parity**:
    - Dữ liệu sẽ được chia thành các block 4 byte và phân phối lần lượt trên các ổ đĩa.
    - Ổ đĩa thứ ba chứa dữ liệu parity được tính bằng phép XOR của các block dữ liệu trên hai ổ đĩa đầu tiên.
- **Dữ liệu đầu ra:** Màn hình in ra kết quả chạy của ổ đĩa RAID 5

## 3. Cấu trúc và cách hoạt động chương trình:

Chương trình bao gồm ba hàm chính:

- Hàm nhập chuỗi ký tự và kiểm tra chuỗi đó có số ký tự là bội của 8 hay không, hoặc là chuỗi rỗng.
- Hàm RAID 5 được chia thành ba phần:
    - split1: Tính toán 4 byte parity từ 2 block đầu tiên và lưu vào Disk 3. Nếu còn chuỗi cần lưu, hàm sẽ tiếp tục với block tiếp theo.
    - split2: Tính toán 4 byte parity và lưu vào Disk 2. Nếu còn chuỗi cần lưu, hàm sẽ tiếp tục với block tiếp theo.
    - split3: Tính toán 4 byte parity và lưu vào Disk 1. Nếu còn chuỗi cần lưu, hàm sẽ quay trở về block đầu tiên.

Kết thúc ba phần trên, nếu vẫn còn chuỗi ký tự và chuỗi parity thì chương trình sẽ lặp lại từ phần 1 cho đến khi hết ký tự để xét thì dừng.

- Hàm hex để chuyển poi 4 byte parity từ chuẩn ASCII sang định dạng Hexa.

## 4. Ý nghĩa các thanh ghi trong chương trình

- $s1: địa chỉ của Disk1
- $s2: địa chỉ của Disk2
- $s3: địa chỉ của Disk3
- $t3: độ dài chuỗi input
- $t0: index
- $t1: địa chỉ của chuỗi nhập vào
- $t2: string[i]
- $t3: length
- $t4: gán giá trị bằng 7 (cho lặp tới 0 để đủ 8 bit)
- $t7: địa chỉ của hex
- $a0: chỉ số của mảng hex
- $t8: địa chỉ của chuỗi par

## 5. Mã nguồn

```
.data
    start: .asciiz "Nhap chuoi ky tu : "    # Thông báo nhập chuỗi ký tự
    hex: .byte '0','1','2','3','4','5','6','7','8','9','a','b','c','d','e','f'  # Mảng chứa các ký tự hex
    d1: .space 4    # Dành riêng 4 byte cho disk 1
    d2: .space 4    # Dành riêng 4 byte cho disk 2
    d3: .space 4    # Dành riêng 4 byte cho disk 3
    array: .space 32    # Dành riêng 32 byte cho mảng parity
    string: .space 5000    # Dành riêng 5000 byte cho chuỗi nhập
    enter: .asciiz "\n"    # Ký tự xuống dòng
    error_length: .asciiz "Do dai chuoi khong hop le! Nhap chuoi khac\n"
    # Thông báo lỗi độ dài chuỗi không hợp lệ
    m: .asciiz "      Disk 1              Disk 2              Disk 3\n"    # Thông báo cho disk
    m2: .asciiz "----------------      ----------------      ----------------\n"    # Đường kẻ ngang
    m3: .asciiz "|     "                # Ký tự mở ngoặc cho việc in ra các giá trị của disk
    m4: .asciiz "     |      "           # Ký tự đóng ngoặc và cách ra
    m5: .asciiz "[[ "                  # Ký tự mở ngoặc cho việc in ra các giá trị của parity
    m6: .asciiz "]]       "      # Ký tự đóng ngoặc và cách ra
    comma: .asciiz ","        # Ký tự dấu phẩy
```

```
    ms: .asciiz "Try again?"        # Thông báo hỏi người dùng muốn thử lại không

.text
    la $s1, d1          # Địa chỉ tương ứng với disk 1
    la $s2, d2          # Địa chỉ tương ứng với disk 2
    la $s3, d3          # Địa chỉ tương ứng với disk 3
    la $a2, array       # Địa chỉ mảng chứa parity

input:
    li $v0, 4                   # Hệ thống gọi để in thông báo nhập chuỗi ký tự
    la $a0, start
    syscall
    li $v0, 8                   # Hệ thống gọi để nhập chuỗi ký tự
    la $a0, string
    li $a1, 1000
    syscall
    move $s0, $a0           # s0 chứa địa chỉ chuỗi vừa nhập
    li $v0, 4
    la $a0, m
    syscall
    li $v0, 4
```

```
    la $a0, m2
    syscall

#----------------------Kiểm tra độ dài có chia hết cho 8 không-----------------------
length:
    addi $t3, $zero, 0              # t3 = độ dài chuỗi
    addi $t0, $zero, 0              # t0 = chỉ số (index)

check_char:
    add $t1, $s0, $t0               # t1 = địa chỉ của string[i]
    lb $t2, 0($t1)                  # t2 = string[i]
    nop
    beq $t2, 10, test_length        # t2 = '\n' kết thúc chuỗi
    nop
    addi $t3, $t3, 1                # tăng độ dài
    addi $t0, $t0, 1                # tăng chỉ số
    j check_char
    nop

test_length:
    move $t5, $t3
```

```
    and $t1, $t3, 0x0000000f        # Xóa hết các byte của $t3 về 0, chỉ giữ lại byte cuối
    bne $t1, 0, test1               # Byte cuối bằng 0 hoặc 8 thì số chia hết cho 8
    j split1

test1:
    beq $t1, 8, split1
    j error1

error1:
    li $v0, 4
    la $a0, error_length
    syscall
    j input
```

**#----------Kết thúc kiểm tra độ dài----------**

**#----------Lấy parity----------**

```
HEX:
    li $t4, 7

loopH:
```

```
    blt $t4, $0, endloopH
    sll $s6, $t4, 2                 # s6 = t4*4
    srlv $a0, $t8, $s6              # a0 = t8>>s6
    andi $a0, $a0, 0x0000000f
# a0 = a0 & 0000 0000 0000 0000 0000 0000 0000 1111 => lấy byte cuối cùng của a0
    la $t7, hex
    add $t7, $t7, $a0
    bgt $t4, 1, nextc
    lb $a0, 0($t7)                  # in hex[a0]
    li $v0, 11
    syscall

nextc:
    addi $t4, $t4, -1
    j loopH

endloopH:
    jr $ra
```

```
#-----------------------------Mô phỏng RAID 5-------------------------------
# Xét 6 khối đầu
# Lần 1: Lưu vào 2 khối 1,2; XOR vào khối 3-------
split1:
    addi $t0, $zero, 0                  # số byte được in ra (4 byte)
    addi $t9, $zero, 0
    addi $t8, $zero, 0
    la $s1, d1
    la $s2, d2
    la $a2, array

print11:
    li $v0, 4
    la $a0, m3
    syscall

b11:
    lb $t1, ($s0)
    addi $t3, $t3, -1
    sb $t1, ($s1)
```

```
b21:
    add $s5, $s0, 4
    lb $t2, ($s5)                    # t2 chứa địa chỉ từng byte của disk 2
    addi $t3, $t3, -1
    sb $t2, ($s2)

b31:
    xor $a3, $t1, $t2
    sw $a3, ($a2)
    addi $a2, $a2, 4
    addi $t0, $t0, 1
    addi $s0, $s0, 1
    addi $s1, $s1, 1
    addi $s2, $s2, 1
    bgt $t0, 3, reset
    j b11

reset:
    la $s1, d1
    la $s2, d2
```

```
print12:
    lb $a0, ($s1)
    li $v0, 11
    syscall
    addi $t9, $t9, 1
    addi $s1, $s1, 1
    bgt $t9, 3, next11
    j print12

next11:
    li $v0, 4
    la $a0, m4
    syscall
    li $v0, 4
    la $a0, m3
    syscall

print13:
    lb $a0, ($s2)
    li $v0, 11
    syscall
```

```
    addi $t8, $t8, 1
    addi $s2, $s2, 1
    bgt $t8, 3, next12
    j print13

next12:
    li $v0, 4
    la $a0, m4
    syscall
    li $v0, 4
    la $a0, m5
    syscall

    la $a2, array
    addi $t9, $zero, 0

print14:
    lb $t8, ($a2)
    jal HEX
    li $v0, 4
    la $a0, comma
```

```
    syscall
    addi $t9, $t9, 1
    addi $a2, $a2, 4
    bgt $t9, 2, end1          # in ra 3 parity đầu có dấu ",", parity cuối cùng không có
    j print14
end1:
    lb $t8, ($a2)
    jal HEX
    li $v0, 4
```

## 6. Chạy thử chương trình

**Đầu vào:**

- Đầu vào là một chuỗi kí tự được nhập vào từ bàn phím có độ dài chia hết cho 8 và không được là xâu rỗng.

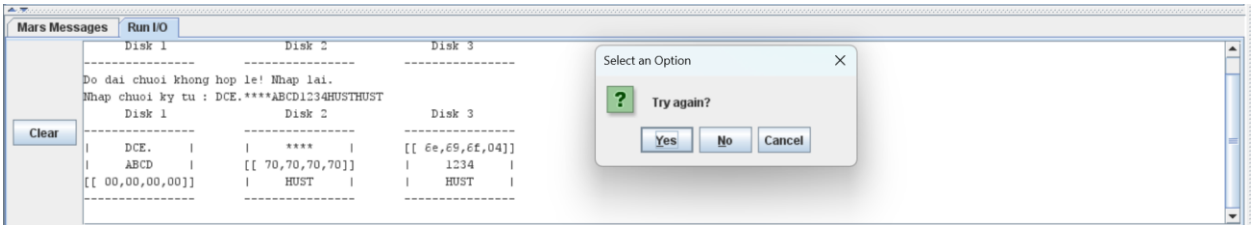
**TH1: Chuỗi không đúng định dạng là bội của 8 kí tự hoặc chuỗi là một xâu rỗng:**

- **Nếu không nhập đúng định dạng (Số phần tử trong chuỗi không là bội của 8)** thì chương trình sẽ trả về "Do dai chuoi khong hop le! Nhap chuoi khac\n"


**TH2: Khi đầu vào hợp lệ theo yêu cầu:**

- **Khi đầu vào đã nhập đúng định dạng**, ta thu được kết quả như sau:

- Ở đây lấy ví dụ đầu vào giống như đề bài: “DCE.****ABCD1234HUSTHUST”




- **Trường hợp với đầu vào dài hơn thỏa mãn điều kiện chia hết cho 8:**
  - Đầu vào là chuỗi: “hoangtruonggiang****cntt********vietnhat”:

```
Nhap chuoi ky tu : hoangtruonggiang****cntt********vietnhat
      Disk 1                 Disk 2                Disk 3
----------------       ----------------       ----------------
|     hoan     |       |     gtru     |       [[ 0f,1b,13,1b]]
|     ongg     |       [[ 06,0f,09,00]]       |     iang     |
[[ 49,44,5e,5e]]       |     ****     |       |     cntt     |
|     ****     |       |     ****     |       [[ 00,00,00,00]]
|     viet     |       [[ 18,01,04,00]]       |     nhat     |
----------------       ----------------       ----------------

-- program is finished running --
```

  - Đầu vào là chuỗi “DCE.****ABCD1234HUSTHUST DCE.****ABCD1234HUSTHUST DCE.****ABCD1234HUSTHUST”

```
Nhap chuoi ky tu : DCE.****ABCD1234HUSTHUSTDCE.****ABCD1234HUSTHUSTDCE.****ABCD1234HUSTHUST
      Disk 1                Disk 2               Disk 3
----------------       ----------------       ----------------
|     DCE.      |       |     ****     |       [[ 6e,69,6f,04]]
|     ABCD      |       [[ 70,70,70,70]]       |     1234     |
[[ 00,00,00,00]]       |     HUST     |       |     HUST     |
|     DCE.      |       |     ****     |       [[ 6e,69,6f,04]]
|     ABCD      |       [[ 70,70,70,70]]       |     1234     |
[[ 00,00,00,00]]       |     HUST     |       |     HUST     |
|     DCE.      |       |     ****     |       [[ 6e,69,6f,04]]
|     ABCD      |       [[ 70,70,70,70]]       |     1234     |
[[ 00,00,00,00]]       |     HUST     |       |     HUST     |
----------------       ----------------       ----------------

-- program is finished running --
```