TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG
Khoa:VIỆT – NHẬT

- - - - - - - - - - - - - - - -

BÁO CÁO BÀI TẬP
MÔN KIẾN TRÚC MÁY TÍNH
ĐỀ TÀI SỐ 1, 8

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

Sinh viên : **Phan Hải Dương - 20194533**

Nguyễn Việt Anh - 20194477

Nhóm: **15** - Lớp: **Việt Nhật K64**

Contents1

A. BÀI TẬP 1

I. ĐỀ BÀI

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

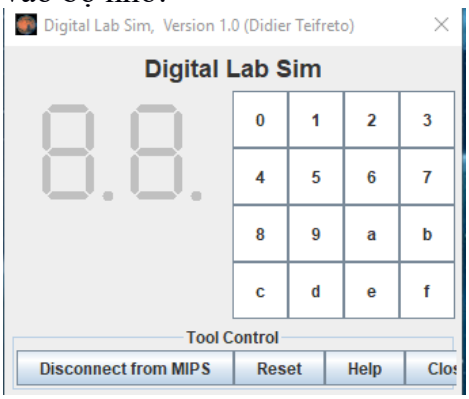
Mã điều khiển	Ý nghĩa
1b4	Marsbot bắt đầu chuyển động
c68	Marsbot đứng im
444	Rẽ trái 90 độ so với phương chuyển động gần nhất
666	Rẽ phải 90 độ so vơi phương chuyển động gần nhất
dad	Bắt đầu để lại vết trên đường
cbc	Chấm dứt để lại vết trên đường
999	Tự động đi theo lộ trình ngược lại. Không vẽ vết, không nhận mã khác cho tới khi kết thúc lộ trình ngược. Mô tả: Marsbot được lập trình để nhớ lại toàn bộ lịch sử các mã điều khiển và khoảng thời gian giữa các lần đổi mã. Vì vậy, nó có thể đảo ngược lại lộ trình để quay về điểm xuất phát.

Sau khi nhận mã điều khiển, Curiosity Marsbot sẽ không xử lý ngay, mà phải đợi lệnh kích hoạt mã từ bàn phím Keyboard & Display MMIO Simulator. Có 3 lệnh như vậy:

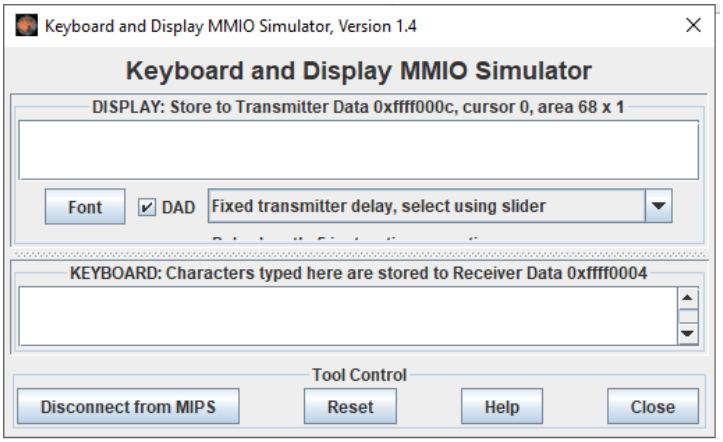
Kích hoạt mã	Ý nghĩa
Phím Enter	Kết thúc nhập mã và yêu cầu Marsbot thực thi.
Phím Delete	Xóa toàn bộ mã điều khiển đang nhập.
Phím Space	Lặp lại lệnh đã thực hiện trước đó.

II. ĐỊNH HƯỚNG

Bước 1(Nhập từ Digital Lab Sim): Cho người dùng nhập từ Digital Lab Sim và lấy dữ liệu vào bộ nhớ.



Bước 2(Sử dụng Keyboard & Display MMIO Simulator): Người dùng sau khi ấn ở Bước 1 sẽ dùng Sử dụng Keyboard & Display MMIO Simulator để thao tác tiếp như mô tả của đề bài.



Bước 3(Kiểm tra code nhập): Nếu ấn Enter ở Bước 2 ta xuống bước 3. Bước này sẽ kiểm tra xem code ở bước 1 có đúng theo quy chuẩn không (đủ 3 ký tự và trùng với 1 mã lệnh), nếu đúng thì in code ra màn hình nếu sai thì in ra màn hình kèm lỗi đồng thời xóa code hiện tại ra khỏi bộ nhớ.

Bước 4(Cài đặt bot): Sau khi nhận lệnh Bot sẽ được nhận lệnh qua các case có sẵn và lưu trữ các bước trong các mảng có sẵn.

III. Ý NGHĨA MÃ NGUỒN

1.Hàm ***loadKeyCase*** : Hàm dùng để nhậm lệnh từ Sử dụng Keyboard & Display MMIO Simulator để chương trình thực hiện các mã kích hoạt bằng cách load dữ liệu từ KEY_CODE

```
loadKeyCase:    lw $t6, 0($k1)                  #nhap tung ki ty cho den khi gap dau \n delete space
                beq $t6, $zero, loadKeyCase
                nop
                beq $t6, $zero, loadKeyCase
                lw $t5, 0($k0)                  #doc tung gia tri cua nut
                beq $t5, 127 , resetCode                #delete case
                beq $t5, 32, doAgain            #space case
                bne $t5, '\n' , loadKeyCase     #gap /n thi dung va chay xuong checkLengt
                nop
```

2.Hàm ***CheckLenght->ReadCase***:2 hàm này dùng để kiểm tra mã điều khiển nếu đúng thì chúng sẽ được kích hoạt vào các hàm điều khiển

3.Tổ hợp các hàm ***printf*** :Các hàm này được sử dụng để in ra màn hình chính các thông tin, trạng thái của chương trình

4.Hàm ***doAgain***: Đây là hàm khi nhận mã kích hoạt Space, hàm sẽ lấy thông tin từ mảng save **saveControlCase(**mảng dùng để lưu các bước đã sử dụng**)** sau đó load từng phần tử và gọi lại hàm điều khiển, sau đó lấy thông tin từ mảng **saveTimeArray(**mảng dùng để lưu trữ thời gian giữa các câu lệnh**)** sau đó sẽ bắt hệ thống chờ để tiếp tục lấy ra lệnh tiếp theo

5. Các tổ hợp hàm điều khiển 7 hàm (**goCase, stopCase, goLeftCase, goRightCase, trackCase, untrackCase, goBackCase**):Các hàm ứng với 7 lệnh điều khiển, khi gọi đến các hàm thì sẽ xét các trạng thái tương tự cho bot thực hiện lệnh đồng thời sẽ lưu lại trạng thái của bot và các mảng **saveStateBot, saveControlCase, saveTimeArray, currentLocation** để phục vụ cho các hàm Space

6.Hàm ***checkCaseEvent***: Hàm này dùng để bổ trợ cho hàm ***ReadCase*** để xem rằng mã

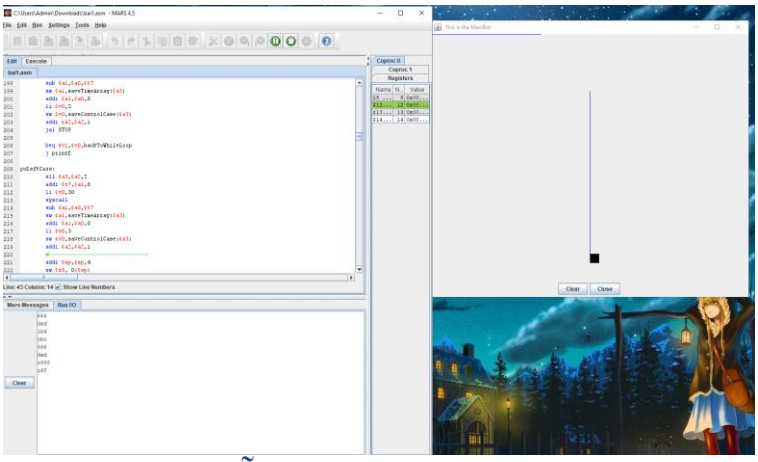
nhận vào có giống một trong các hàm điều khiển hay không
7.Hàm ***saveStatus***: Hàm này dùng để lưu lại các trạng thái của bot trong hàm điều khiển dùng mảng **saveStateBot** chia thành 3 word 1 lưu các giá trị x, y và hướng hiện tại của bot

IV. MÃ NGUỒN

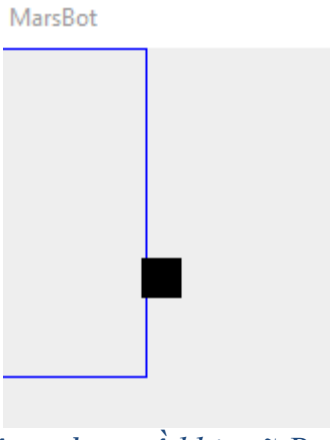
Mã nguồn đã được lưu và gửi cho thầy
n01_g15_NguyenVietAnh.asm

V. HÌNH ẢNH MÔ PHỎNG

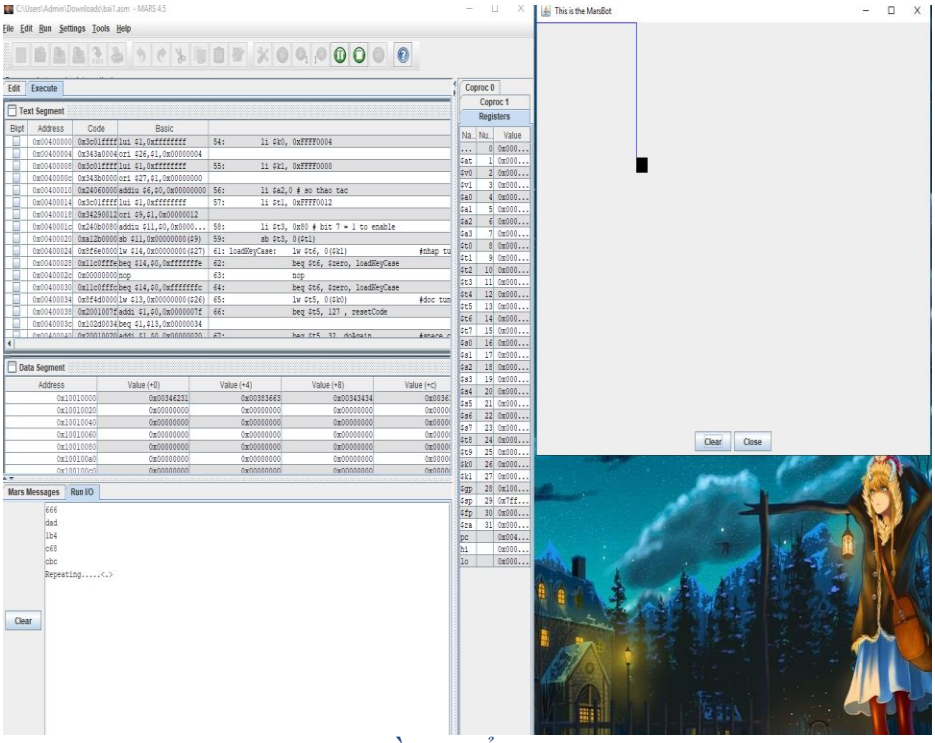
Lỗi



Chạy mẫu với các lệnh đơn giản



Bot tự động chạy về khi mã Back "999"



Bot khi nhận được mã điều khiển Space và lặp lại hoạt động

B. BÀI TẬP 8

I. ĐỀ BÀI

Bài 8: Mô phỏng ổ đĩa RAID 5

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.

Ví dụ:

Input:

string1: DCE.****ABCD1234HUSTHUST

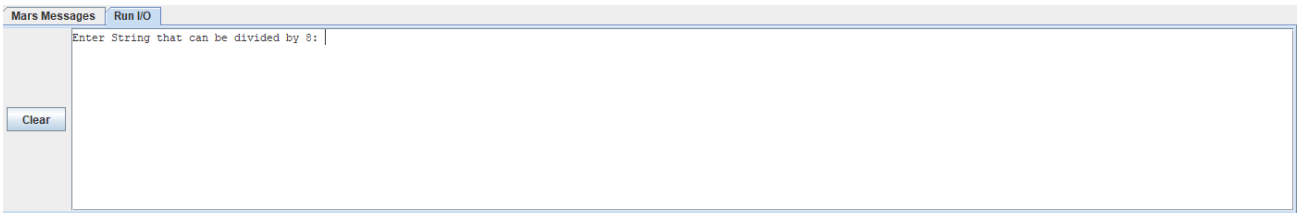
Output:

Enter string that can be divided by 8: 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

II. ĐỊNH HƯỚNG

B1: Nhập liệu

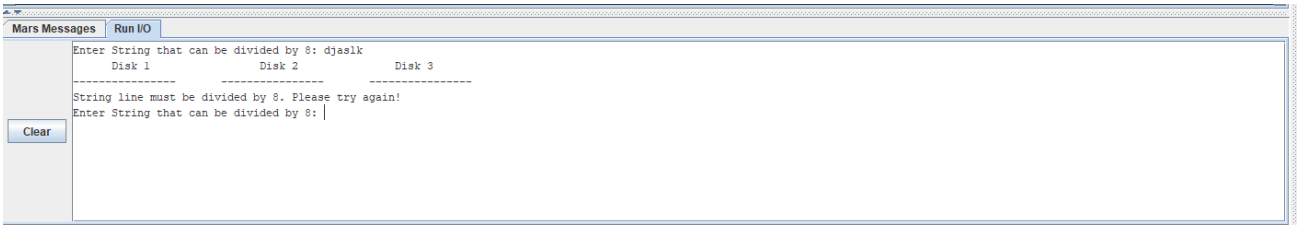


B2: Thuật toán thực hiện:

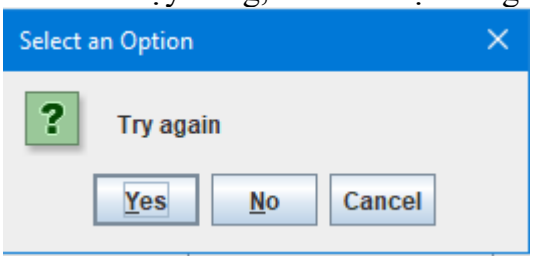
1. **Input String có số byte chia hết cho 8**
   - **TH 1: chia hết cho 8**

   Chuyển sang hàm **parityCheck** để tiếp tục check bit và print.
   - **TH 2: không chia hết cho 8**

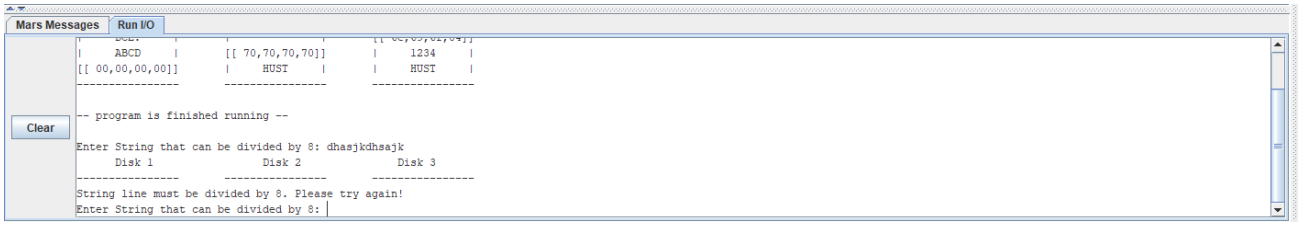
   Chuyển sang hàm **divideError** để in **errorMessage** và bắt nhập lại.

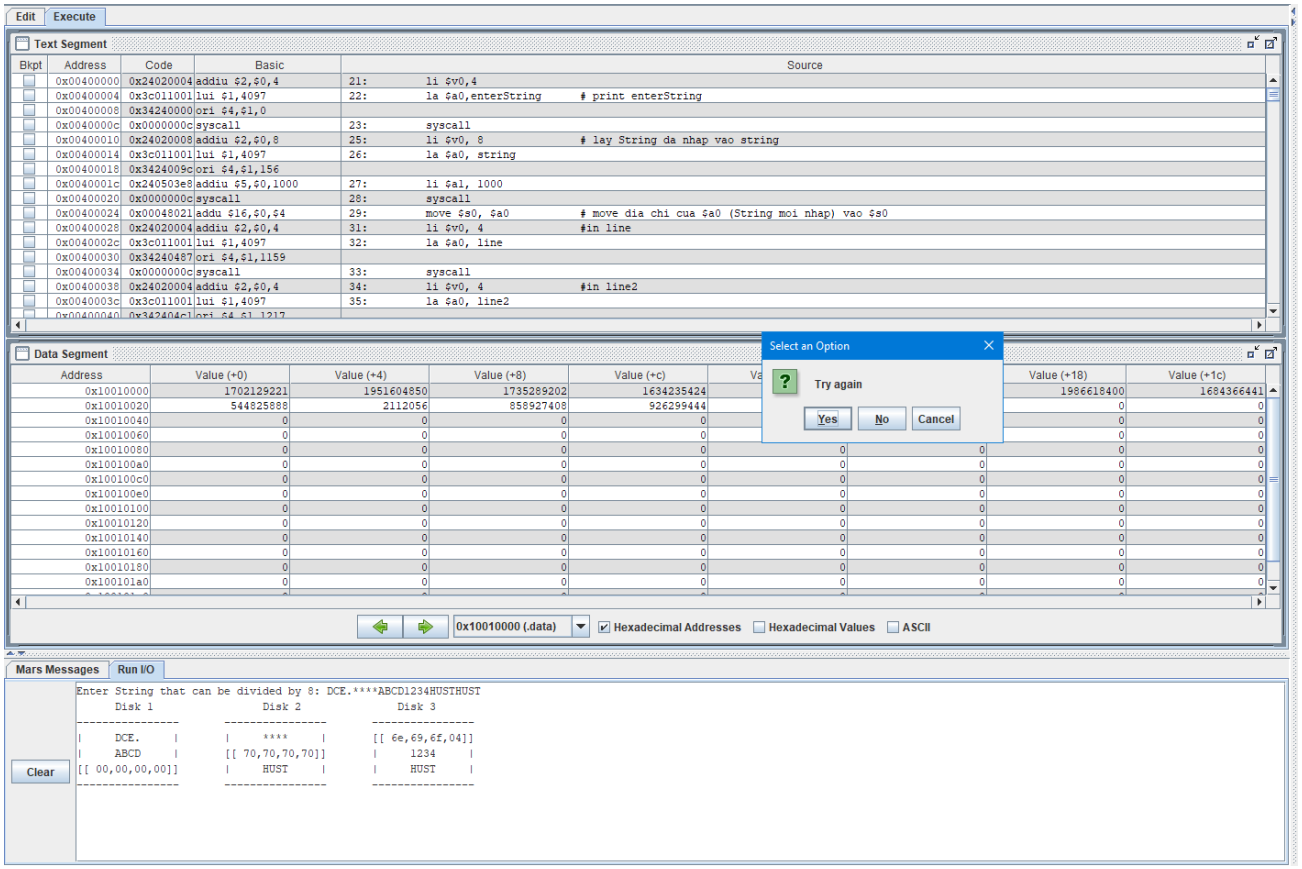


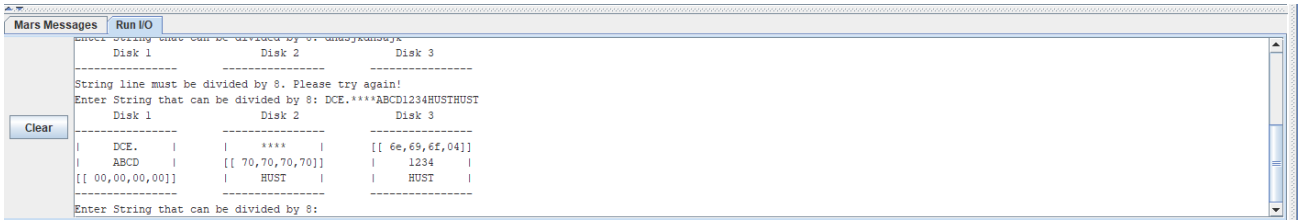
2. **parityCheck**
   1 String 8 byte sẽ được chia làm 2 block gồm 4 byte và bitwise xor từng byte 1 của mỗi block với nhau. 2 byte đầu được lưu vào disk 1 và 2 riêng và bitwise sẽ được lưu dưới dạng Hex ASCII ở disk 3. 2 byte sau đó được lưu vào disk 1,3 và bitwise sẽ được lưu dưới dạng Hex ASCII ở disk 2. 2 byte cuối được lưu vào disk 2,3 và bitwise sẽ được lưu dưới dạng Hex ASCII ở disk 1. Sau đó là loop.
3. **HEXASCII**
   Sau khi được bitwise xor, sẽ chuyển thành Hex ASCII và print
   VD: DCE.****
   D là 68, * là 42. Sau khi bitwise xor sẽ là 110 chuyển sang hex là: 6E
4. **tryAgain**
   Sau khi chạy xong, sẽ xuất hiện bảng notification **Try Again** Cho người dùng chọn.



III. HÌNH ẢNH MÔ PHỎNG







IV. MÃ NGUỒN

Lưu trong file n15_g08_PhanHaiDuong.asm đi kèm.