

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



BÁO CÁO BÀI TẬP LỚN
IT3280-THỰC HÀNH KIẾN TRÚC MÁY TÍNH

Thông tin lớp học

Học phần	Tên học phần	Mã lớp
IT3280	Thực hành kiến trúc máy tính	147789

Thông tin sinh viên

Tên sinh viên	Lớp	MSSV
Phan Thanh Thắng	VN 03-K67	20225927
Đặng Minh Trí	VN 05-K67	20225939

Giảng viên hướng dẫn : Lê Bá Vui

Trợ giảng : Đỗ Gia Huy

Mục lục

Bài 4 : Postscript CNC Marsbot.....	3
1. Giới thiệu vấn đề.....	3
2. Triển khai dự án.....	3
2.1 Ý tưởng bài toán	4
2.2 Phân tích bài toán	4
2.3 Mã nguồn.....	9
3. Demo sản phẩm.....	15
3.1 Các trường hợp ngoại lệ.....	15
3.2 Kết quả.....	16
Bài 8 : Mô phỏng ổ đĩa RAID 5.....	18
1. Giới thiệu vấn đề.....	18
2. Triển khai dự án.....	18
2.1 Ý tưởng thuật toán.....	18
2.2 Phân tích bài toán	18
2.3 Mã nguồn.....	26
3. Demo sản phẩm.....	32
3.1 Các trường hợp ngoại lệ.....	32
3.2 Kết quả.....	32

Bài 4 : Postscript CNC Marsbot

1. Giới thiệu vấn đề

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. Triển khai dự án

2.1 Ý tưởng thuật toán

- Sử dụng 1 vòng lặp vô hạn để thực thi chương trình và xử lý bài toán bằng chương trình phục vụ ngắt khi chọn phím từ keyboard , khi ngắt chương trình sẽ được xử lý tại địa chỉ 0x80000180 bằng cách sử dụng .ktext 0x80000180 , bằng cách này khi keyboard được người sử dụng chọn phím bất kì thì chương trình phục vụ ngắt xảy ra và từ đây marsbot sẽ bắt đầu thực thi theo yêu cầu

- Sử dụng keyboard để người dùng chọn lựa các chữ mình cần cắt , cài đặt bit 7 = 1 vào IN_ADDRESS_HEXA_KEYBOARD 0xFFFF0012 để chương trình phục vụ ngắt được thực thi khi người nhập đã chọn phím
- Lưu các chỉ số <góc chuyển động>,<cắt/không cắt>,<thời gian> vào 1 mảng , khi marsbot bắt đầu được kích hoạt thì duyệt mảng từ đầu cho tới phần tử cuối cùng của mảng , lưu lần lượt 3 chỉ số <góc chuyển động>,<cắt/không cắt>,<thời gian> vào 3 phần tử của mỗi vòng lặp. Mỗi vòng lặp cài lại bộ nhớ của MOVING 0xffff8050 = 1 để marsbot di chuyển, cho marsbot di chuyển bằng cách sleep trong syscall v0 = 32, và a0 = <thời gian>, bộ nhớ của HEADING 0xffff8010 chứa <góc chuyển động> , bộ nhớ của LEAVETRACK 0xffff8020 chứa <cắt/không cắt> để quyết định có để lại 1 track hay không. Khi duyệt tới phần tử cuối cùng cài lại bộ nhớ của MOVING 0xffff8050 = 0 để marsbot dừng di chuyển

Sau khi cắt xong điều hướng chương trình quay về vòng lặp để đợi người dùng muốn cắt hình gì tiếp theo

2.2 Phân tích bài toán

2.2.1 Gán các địa chỉ đầu vào , đầu ra của keyboard và marsbot

```
# Mars Bot
.eqv HEADING 0xffff8010 # Integer: An angle between 0 and 359
.eqv MOVING 0xffff8050 # Boolean: whether or not to move
.eqv LEAVETRACK 0xffff8020 # Boolean (0 or non-0): 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
#Key Matrix
.eqv IN_ADDRESS_HEXA_KEYBOARD 0xFFFF0012
.eqv OUT_ADDRESS_HEXA_KEYBOARD 0xFFFF0014
```

2.2.2 Lưu các mảng và các xâu kí tự cần thiết để khi keyboard bắt kí tự thì sẽ có các chuỗi kí tự in ra để hỗ trợ người dùng

```
.data
# postscript DCE
postscript0: .word 90,0,3000,180,0,6000,180,1,13750,80,1,1200,70,1,1200,60,1,1200,50,1,1200,40,1,1200,30,1,12
end0: .word
# postscript PTT
postscript4: .word 90,0,3000,180,0,6000,180,1,13750,0,0,13750,90,1,750,100,1,750,110,1,750,120,1,750,130,1,75
end4: .word
# postscript hcn hcn hv
postscript8: .word 90,0,6000,180,0,3000,260,1,500,250,1,500,240,1,500,230,1,500,220,1,500,210,1,500,200,1,500
end8: .word
message1 : .asciiz "Vui long nhap 0 de marsbot cat DCE \n"
message2 : .asciiz "Vui long nhap 4 de marsbot cat PTT \n"
message3 : .asciiz "Vui long nhap 8 de marsbot cat hcn,hcn,hv \n"
track1 : .asciiz "Dang Track DCE \n"
track2 : .asciiz "Dang Track PTT \n"
track3 : .asciiz "Dang Track hcn,hcn,hv \n"
```

2.2.3 Trong hàm main gán bit 7 của địa chỉ đầu vào IN_ADDRESS_HEX_KEYBOARD = 1 để khi người dùng nhập phím bất kì chương trình phục vụ ngắt sẽ hoạt động

```
#-----  
#Cho phép ngắt của Bàn phím ma tran 4x4 của Digital Lab Sim  
    li $t1, IN_ADDRESS_HEX_KEYBOARD  
    li $t3, 0x80 # bit 7 = 1 (01000000) thì kích hoạt ngắt  
    sb $t3, 0($t1)  
#-----  
# vòng lặp vô hạn  
#-----  
Loop:  
    nop  
    nop  
    addi $v0, $zero, 32  
    li $a0, 200  
    syscall  
    nop  
    nop  
    b Loop  
end_main:
```

2.2.4 Sử dụng vòng lặp vô hạn và có thời gian ngủ để đợi người dùng nhập số từ keyboard để chương trình phục vụ ngắt được thực thi

```
#-----  
#Cho phép ngắt của Bàn phím ma tran 4x4 của Digital Lab Sim  
    li $t1, IN_ADDRESS_HEX_KEYBOARD  
    li $t3, 0x80 # bit 7 = 1 (01000000) thì kích hoạt ngắt  
    sb $t3, 0($t1)  
#-----  
# vòng lặp vô hạn  
#-----  
Loop:  
    nop  
    nop  
    addi $v0, $zero, 32  
    li $a0, 200  
    syscall  
    nop  
    nop  
    b Loop  
end_main:
```

2.2.5 Bắt đầu chương trình ngắt các giá trị trong thanh ghi mình đang dùng trong vòng lặp ở hàm main, các giá trị trước khi chương trình ngắt bắt đầu có thể bị thay đổi, nên phải cất dữ liệu vào stack

```
# Chương trình phục vụ ngắt chung khi lỗi xảy ra
#-----
.ktext 0x80000180
#-----
# lưu các giá trị vào stack, sau khi thực hiện xong trả lại sau
#-----
IntSR:
    addi $sp,$sp,4 # lưu $at vào stack vì sau này có thể dữ liệu bị thay đổi
    sw $at,0($sp)
    addi $sp,$sp,4 # lưu $v0 vào stack vì sau này có thể dữ liệu bị thay đổi
    sw $v0,0($sp)
    addi $sp,$sp,4 # lưu $a0 vào stack vì sau này có thể dữ liệu bị thay đổi
    sw $a0,0($sp)
    addi $sp,$sp,4 # lưu $t1 vào stack vì sau này có thể dữ liệu bị thay đổi
    sw $t1,0($sp)
    addi $sp,$sp,4 # Save $t3 because we may change it later
    sw $t3,0($sp)
#-----
```

2.2.6 Lần lượt gán các địa chỉ hàng mong muốn người dùng nhập và kích hoạt lại bit 7 để nếu không có phím 0,4,8 được nhấn thì chương trình phục vụ ngắt vẫn đang đợi người dùng nhập lại phím khác. Nếu phím được nhấn (tức là giá trị OUT_ADDRESS_HEX_KEYBOARD sẽ khác 0x0) thì sẽ tiếp tục kiểm tra đây là phím gì

```
get_key:
    li $t1, IN_ADDRESS_HEX_KEYBOARD
    li $t3, 0x81 # kiểm tra hàng 1 và kích hoạt lại bit 7
    sb $t3, 0($t1)
    li $t1, OUT_ADDRESS_HEX_KEYBOARD
    lb $a0, 0($t1)
    bne $a0, 0x0, key_pressed # kiểm tra đã được nhấn phím hay chưa

    li $t1, IN_ADDRESS_HEX_KEYBOARD
    li $t3, 0x82 # kiểm tra hàng 2 và kích hoạt lại bit 7
    sb $t3, 0($t1)
    li $t1, OUT_ADDRESS_HEX_KEYBOARD
    lb $a0, 0($t1)
    bne $a0, 0x0, key_pressed

    li $t1, IN_ADDRESS_HEX_KEYBOARD
    li $t3, 0x84 # kiểm tra hàng 3 và kích hoạt lại bit 7
    sb $t3, 0($t1)
    li $t1, OUT_ADDRESS_HEX_KEYBOARD
    lb $a0, 0($t1)
    bne $a0, 0x0, key_pressed
```

2.2.7 Kiểm tra phím vừa được nhấn có phải là số 0 (0x11) , số 4 (0x12) số 8 (0x14) hay không , nếu phải thì xử lí riêng từng phím , còn không phải thì hướng dẫn người dùng phải chọn option khác

```
key_pressed:
    beq $a0, 0x11, key_0 # 0 is pressed
    beq $a0, 0x12, key_4 # 4 is pressed
    beq $a0, 0x14, key_8 # 8 is pressed
    j end_script
```

2.2.8 Sau khi các số được chọn là 0,4,8 thì Marsbot sẽ thực thi

```
key_0:
    la $a0, track1
    li $v0, 4
    syscall
    la $a2, postscript0 # lưu địa chỉ của postscript0 vào a2
    la $a1, end0 # kết thúc lưu vào a1
    j MarsBot_Draw

key_4:
    la $a0, track2
    li $v0, 4
    syscall
    la $a2, postscript4 # lưu địa chỉ của postscript4 vào a2
    la $a1, end4 # kết thúc lưu vào a1
    j MarsBot_Draw

key_8:
    la $a0, track3
    li $v0, 4
    syscall
    la $a2, postscript8 # lưu địa chỉ của postscript8 vào a2
    la $a1, end8 # kết thúc lưu vào a1
    j MarsBot_Draw
```

2.2.9 Duyệt mảng gồm <góc chuyển động>,<cắt/không cắt>,<thời gian> từ phần tử đầu tới cuối, tại mỗi vòng lặp lấy ra 3 giá trị để cài vào HEADING,MOVING, LEAVETRACK

```
MarsBot_Draw: # Bat dau chay marsbot
read_script: # doc postscript
    beq $a2, $a1, end_script
read_angle:
    lw $a0, 0($a2) # load goc , lưu vào heading
    jal ROTATE
    addi $a2, $a2, 4 # a2 = a2+4 để đọc cut or uncut
read_cut_uncut: # cut if 1, uncut if 0
    lw $s0, 0($a2)
    beq $s0, $0, read_duration # s0 = 0 => untrack
    jal TRACK # s0 = 1 => track = 1
read_duration:
    jal GO
    addi $a2, $a2, 4 # a2 = a2 + 4
    lw $a0, 0($a2) # load time go
    addi $v0, $zero, 32 # go bằng cách sleep
    syscall
    jal UNTRACK
    addi $a2, $a2, 4 # a2 = a2 + 4
    j read_script # lặp lại cho tới lúc hết postscript
```

2.2.10 Khi mảng rỗng thì in ra các thông báo và các option để người dùng vẫn có thể tiếp tục chọn lựa

```
end_script:
    li $v0,4
    la $a0,message1
    syscall
    li $v0,4
    la $a0,message2
    syscall
    li $v0,4
    la $a0,message3
    syscall
    jal STOP
```

2.2.11 Sau khi chương trình phục vụ ngắt kết thúc, thanh ghi PC vẫn chứa địa chỉ của lệnh ngắt xảy ra (lệnh đã thực hiện xong) nên phải tăng địa chỉ chứa trong thanh ghi epc lên , trả lại các dữ liệu đã lưu trong stack và thoát chương trình ngắt

```
#-----
# sau khi xu ly ngat , tang lai thanh ghi epc len 4 , $14 chua dia chi tiep theo
# epc <= epc + 4
#-----
next_pc:
    mfc0 $at, $14 # $at <= Coproc0.$14 = Coproc0.epc
    addi $at, $at, 4 # $at = $at + 4 (next instruction)
    mtc0 $at, $14 # Coproc0.$14 = Coproc0.epc <= $at
#-----
# tra lai du lieu ban dau
#-----
restore:
    lw $t3, 0($sp) # Restore the registers from stack
    addi $sp, $sp, -4
    lw $t1, 0($sp) # Restore the registers from stack
    addi $sp, $sp, -4
    lw $a0, 0($sp) # Restore the registers from stack
    addi $sp, $sp, -4
    lw $v0, 0($sp) # Restore the registers from stack
    addi $sp, $sp, -4
    lw $at, 0($sp) # Restore the registers from stack
    addi $sp, $sp, -4
return:
    eret # Quay tro ve chuong trinh chinh
```

2.2.12 Các nhãn của Marsbot để cài lại góc chuyển động,cắt ,không cắt,di chuyển,dừng lại

```
GO:
    li $at, MOVING # thay doi cong MOVING = 1;
    addi $k0, $zero,1
    sb $k0, 0($at) # bat dau di chuyen
    jr $ra

STOP:
    li $at, MOVING # thay doi cong MOVING = 0;
    sb $zero, 0($at) # dung lai
    jr $ra

TRACK:
    li $at, LEAVETRACK # thay doi cong LEAVETRACK = 1
    addi $k0, $zero,1 # to logic 1,
    sb $k0, 0($at) # bat dau ve
    jr $ra

UNTRACK:
    li $at, LEAVETRACK # thay doi cong LEAVETRACK = 0
    sb $zero, 0($at) # dung ve
    jr $ra

ROTATE:
    li $at, HEADING # thay doi goc xoay cua robot
    sw $a0, 0($at)
    jr $ra
```


2.3 Mã nguồn

Mars Bot

```
.eqv HEADING 0xffff8010 # Integer: An angle between 0 and 359
```

```
.eqv MOVING 0xffff8050 # Boolean: whether or not to move
```

```
.eqv LEAVETRACK 0xffff8020 # Boolean (0 or non-0): 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
```

#Key Matrix

```
.eqv IN_ADDRESS_HEX keyboard 0xFFFF0012
```

```
.eqv OUT_ADDRESS_HEX keyboard 0xFFFF0014
```

.data

postscript DCE

postscript0: .word

90,0,3000,180,0,6000,180,1,13750,80,1,1200,70,1,1200,60,1,1200,50,1,1200,40,1,1200,30,1,1200,20,1,1200,10,1,1200,0,1,1200,350,1,1200,340,1,1200,330,1,1200,320,1,1200,310,1,1200,300,1,1200,290,1,1200,280,1,1150,90,0,14000,260,1,1200,250,1,1200,240,1,1200,230,1,1200,220,1,1200,210,1,1200,200,1,1200,190,1,1200,180,1,1200,170,1,1200,160,1,1200,150,1,1200,140,1,1200,130,1,1200,120,1,1200,110,1,1200,100,1,1200,90,0,4000,90,1,5000,270,0,5000,0,1,7000,90,1,5000,270,0,5000,0,1,7000,90,1,5000,0,0,3000

end0: .word

postscript PTT

postscript4: .word

90,0,3000,180,0,6000,180,1,13750,0,0,13750,90,1,750,100,1,750,110,1,750,120,1,750,130,1,750,140,1,750,150,1,750,160,1,750,170,1,750,180,1,750,190,1,750,200,1,750,210,1,750,220,1,750,230,1,750,240,1,750,250,1,750,260,1,750,270,1,750,180,0,500,0,0,9000,90,0,6000,90,1,10000,270,0,5000,180,1,13000,0,0,13000,90,0,7000,90,1,10000,270,0,5000,180,1,13000,90,0,2000

end4: .word

```
# postscript hcn hcn hv
```

postscript8: .word

90,0,6000,180,0,3000,260,1,500,250,1,500,240,1,500,230,1,500,220,1,500,210,1,500,200,1,500,190,1,500,180,1,500,170,1,500,160,1,500,150,1,500,140,1,500,130,1,500,120,1,500,110,1,500,100,1,500,90,1,500,80,1,500,70,1,500,60,1,500,50,1,500,40,1,500,30,1,500,20,1,500,10,1,500,0,1,500,350,1,500,340,1,500,330,1,500,320,1,500,310,1,500,300,1,500,290,1,500,280,1,500,270,1,500,90,0,9000,270,1,1500,240,1,1500,210,1,1500,180,1,1500,150,1,1500,120,1,1500,90,1,1500,60,1,150

0,30,1,1500,0,1,1500,330,1,1500,300,1,1500,90,0,6000,90,1,5000,180,1,5000,270
,1,5000,0,1,5000,0,0,2000

end8: .word

message1 : .ascii "Vui long nhap 0 de marsbot cat DCE \n"

message2 : .ascii "Vui long nhap 4 de marsbot cat PTT \n"

message3 : .ascii "Vui long nhap 8 de marsbot cat hcn,hcn,hv \n"

track1 : .ascii "Dang Track DCE \n"

track2 : .ascii "Dang Track PTT \n"

track3 : .ascii "Dang Track hcn,hcn,hv \n"

#~~~~~
~~~

# MAIN Procedure

#~~~~~  
~~~

.text

main:

#-----

Cho phép ngat

#-----

col 0x1 col 0x2 col 0x4 col 0x8

#

row 0x1 0 1 2 3

0x11 0x21 0x41 0x81

#

row 0x2 4 5 6 7

0x12 0x22 0x42 0x82

#

row 0x4 8 9 a b

0x14 0x24 0x44 0x84

#

row 0x8 c d e f

0x18 0x28 0x48 0x88

#

#-----

#-----

#Cho phép ngat của Bàn phím ma tran 4x4 của Digital Lab Sim

li \$t1, IN_ADDRESS_HEX_KEYBOARD

li \$t3, 0x80 # bit 7 = 1 (01000000) thi kích hoạt ngat

sb \$t3, 0(\$t1)

```

#-----
# vong lap vo han
#-----
Loop:
    nop
    nop
    addi $v0, $zero, 32
    li $a0, 200
    syscall
    nop
    nop
    b Loop
end_main:

#-----
~~~
# Chuong trinh phuc vu ngat chung khi loi xay ra
#-----
~~~
.ktext 0x80000180
#-----
# luu cac gia tri vao stack , sau khi thuc hien xong tra lai sau
#-----
IntSR:
    addi $sp,$sp,4 # luu $at vao stack vi sau nay co the du lieu bi thay doi
    sw $at,0($sp)
    addi $sp,$sp,4 # luu $v0 vao stack vi sau nay co the du lieu bi thay doi
    sw $v0,0($sp)
    addi $sp,$sp,4 # luu $a0 vao stack vi sau nay co the du lieu bi thay doi
    sw $a0,0($sp)
    addi $sp,$sp,4 # luu $t1 vao stack vi sau nay co the du lieu bi thay doi
    sw $t1,0($sp)
    addi $sp,$sp,4 # Save $t3 because we may change it later
    sw $t3,0($sp)

#-----
# Xu li khi chon phim
#-----
get_key:
    li $t1, IN_ADDRESS_HEX_KEYBOARD
    li $t3, 0x81 # kiem tra hang 1 va kich hoat lai bit 7

```

```

sb $t3, 0($t1)
li $t1, OUT_ADDRESS_HEX_KEYBOARD
lb $a0, 0($t1)
bne $a0, 0x0, key_pressed #kiem tra da duoc nhan phim hay chua

```

```

li $t1, IN_ADDRESS_HEX_KEYBOARD
li $t3, 0x82 # kiem tra hàng 2 và kích hoạt lại bit 7
sb $t3, 0($t1)
li $t1, OUT_ADDRESS_HEX_KEYBOARD
lb $a0, 0($t1)
bne $a0, 0x0, key_pressed

```

```

li $t1, IN_ADDRESS_HEX_KEYBOARD
li $t3, 0x84 # kiem tra hàng 3 và kích hoạt lại bit 7
sb $t3, 0($t1)
li $t1, OUT_ADDRESS_HEX_KEYBOARD
lb $a0, 0($t1)
bne $a0, 0x0, key_pressed

```

key_pressed:

```

beq $a0, 0x11, key_0 # 0 is pressed
beq $a0, 0x12, key_4 # 4 is pressed
beq $a0, 0x14, key_8 # 8 is pressed
j end_script

```

key_0:

```

la $a0 ,track1
li $v0,4
syscall
la $a2, postscript0 # lưu địa chỉ của postscript0 vào a2
la $a1, end0 # kết thúc lưu vào a1
j MarsBot_Draw

```

key_4:

```

la $a0 ,track2
li $v0,4
syscall
la $a2, postscript4 # lưu địa chỉ của postscript4 vào a2
la $a1, end4 # kết thúc lưu vào a1
j MarsBot_Draw

```

key_8:

```

la $a0 ,track3

```

```

    li $v0,4
    syscall
    la $a2, postscript8 # luu dia chi cua postscript8 vao a2
    la $a1, end8 # ket thuc luu vao a1
    j MarsBot_Draw

```

MarsBot_Draw: # Bat dau chay marsbot

read_script: # doc postscript

```
    beq $a2, $a1, end_script
```

read_angle:

```
    lw $a0, 0($a2) # load goc , luu vao heading
```

```
    jal ROTATE
```

```
    addi $a2, $a2, 4 # a2 = a2+4 de doc cut or uncut
```

read_cut_uncut: # cut if 1, uncut if 0

```
    lw $s0, 0($a2)
```

```
    beq $s0, $0, read_duration #s0 = 0 ==> untrack
```

```
    jal TRACK # s0 = 1 ==>> track = 1
```

read_duration:

```
    jal GO
```

```
    addi $a2, $a2, 4 # a2 = a2 + 4
```

```
    lw $a0, 0($a2) # load time go
```

```
    addi $v0,$zero,32 # go bang cach sleep
```

```
    syscall
```

```
    jal UNTRACK
```

```
    addi $a2, $a2, 4 # a2 = a2 + 4
```

```
    j read_script # lap lai cho toi luc het postscript
```

end_script:

```
    li $v0,4
```

```
    la $a0,message1
```

```
    syscall
```

```
    li $v0,4
```

```
    la $a0,message2
```

```
    syscall
```

```
    li $v0,4
```

```
    la $a0,message3
```

```
    syscall
```

```
    jal STOP
```

#-----

```
# sau khi xu ly ngat , tang lai thanh ghi epc len 4 , $14 chua dia chi tiep theo
# epc <= epc + 4
```

```
#-----
```

```
next_pc:
```

```
    mfc0 $at, $14 # $at <= Coproc0.$14 = Coproc0.epc
    addi $at, $at, 4 # $at = $at + 4 (next instruction)
    mtc0 $at, $14 # Coproc0.$14 = Coproc0.epc <= $at
```

```
#-----
```

```
# tra lai du lieu ban dau
```

```
#-----
```

```
restore:
```

```
    lw $t3, 0($sp) # Restore the registers from stack
    addi $sp, $sp, -4
    lw $t1, 0($sp) # Restore the registers from stack
    addi $sp, $sp, -4
    lw $a0, 0($sp) # Restore the registers from stack
    addi $sp, $sp, -4
    lw $v0, 0($sp) # Restore the registers from stack
    addi $sp, $sp, -4
    lw $at, 0($sp) # Restore the registers from stack
    addi $sp, $sp, -4
```

```
return:
```

```
    eret # Quay tro ve chuong trinh chinh
```

```
GO:
```

```
    li $at, MOVING # thay doi cong MOVING = 1;
    addi $k0, $zero, 1
    sb $k0, 0($at) # bat dau di chuyen
    jr $ra
```

```
STOP:
```

```
    li $at, MOVING # thay doi cong MOVING = 0;
    sb $zero, 0($at) # dung lai
    jr $ra
```

```
TRACK:
```

```
    li $at, LEAVETRACK # thay doi cong LEAVETRACK = 1
    addi $k0, $zero, 1 # to logic 1,
    sb $k0, 0($at) # bat dau ve
    jr $ra
```

```
UNTRACK:
```

```
    li $at, LEAVETRACK # thay doi cong LEAVETRACK = 0
```

sb \$zero, 0(\$at) # dung ve

jr \$ra

ROTATE:

li \$at, HEADING # thay doi goc xoay cua robot

sw \$a0, 0(\$at)

jr \$ra

3. Demo sản phẩm

3.1 Các trường hợp ngoại lệ

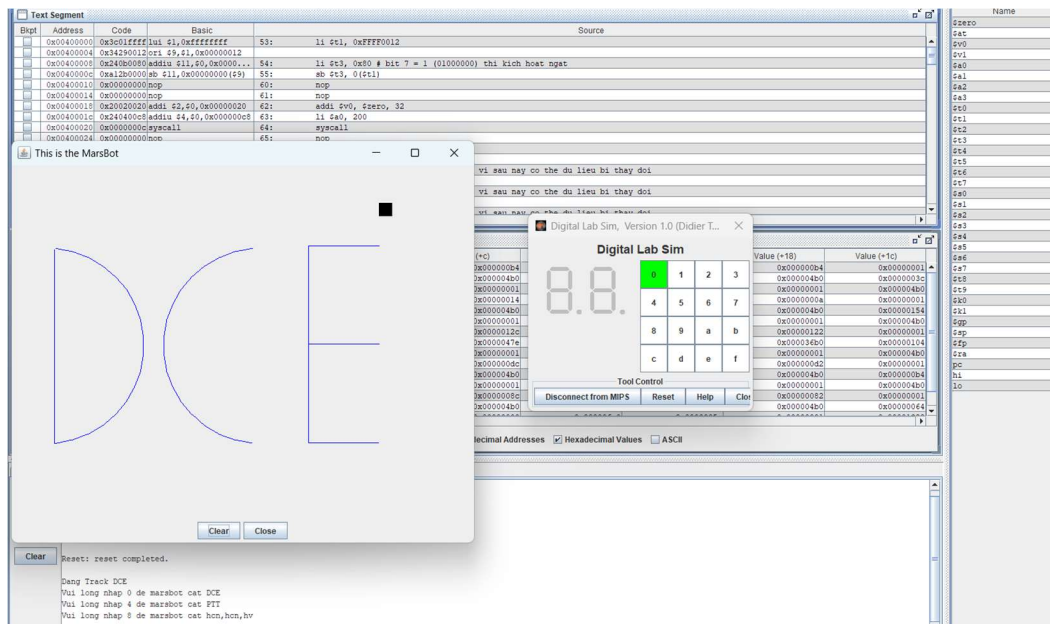
Khi chọn các phím không phải 0,4,8 thì in ra các thông báo để người dùng có thể chọn lại để marsbot bắt đầu hoạt động

The screenshot displays a Digital Lab Sim interface. The main window is divided into several sections:

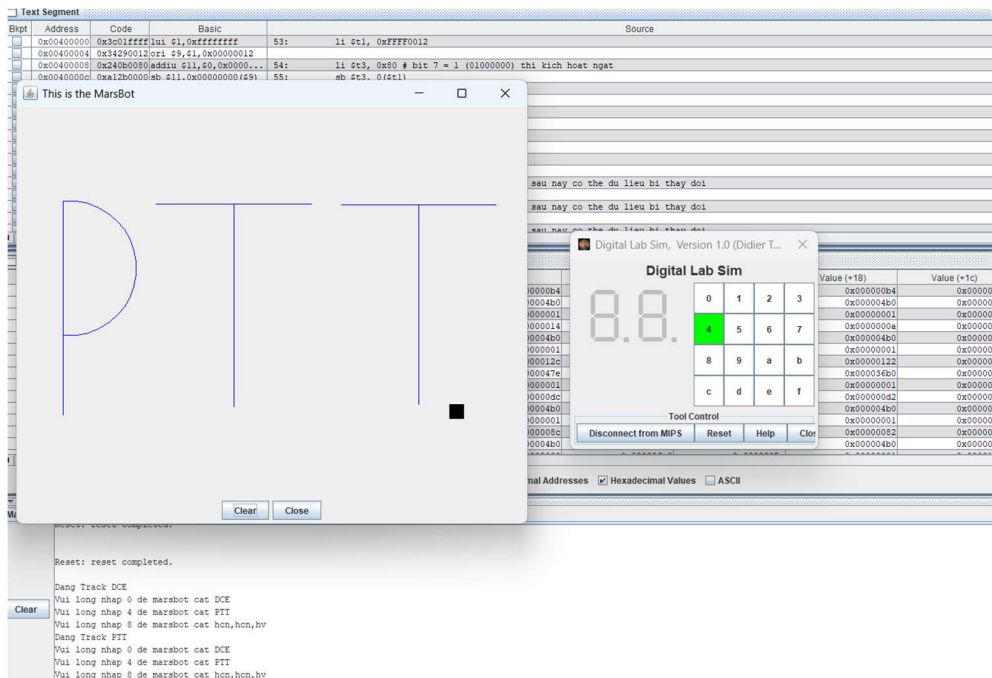
- Text Segment:** A table of MIPS assembly instructions. The first instruction is `li $t1, 0xFFFF0012`. Other instructions include `ori $9,$1,0x00000012`, `addiu $11,$0,0x0000...`, `sb $t3, 0($t1)`, `nop`, `addi $v0,$zero, 32`, `li $a0, 200`, `syscall`, `b Loop`, `addi $sp,$sp,4`, `sw $at,0($sp)`, `addi $sp,$sp,4`, `sw $v0,0($sp)`, and `addi $an,$an,4`.
- Data Segment:** A table of memory addresses and their corresponding values in hexadecimal, decimal, and ASCII. The first row shows address `0x10010000` with value `0x0000005a`.
- Digital Display:** A central window showing the number `8.8` on a 7-segment display. Below the display is a numeric keypad with buttons 0-9, a, b, c, d, e, f, and a **Tool Control** section with buttons **Disconnect from MIPS**, **Reset**, **Help**, and **Close**.
- Mars Messages:** A log at the bottom showing messages like `Vui long nhap 0 de marsbot cat DCE`, `Vui long nhap 4 de marsbot cat PTI`, and `Vui long nhap 8 de marsbot cat hcn,hcn,hv`.

3.2 Kết quả

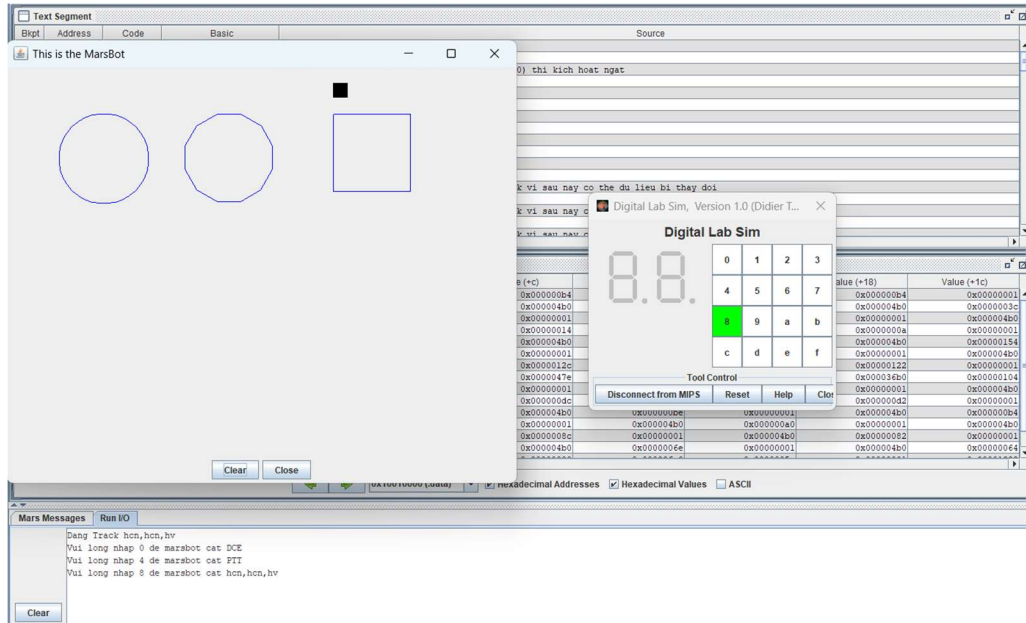
3.2.1 Cắt chữ DCE



3.2.2 Cắt chữ PTT



3.2.3 Cắt hình chữ nhật, hình chữ nhật, hình vuông



Bài 8 : Mô phỏng ổ đĩa RAID 5

1. Giới thiệu vấn đề

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 $6e = 'D' \text{ xor } '*'$; $69 = 'C' \text{ xor } '*'$; $6f = 'E' \text{ xor } '*'$; $04 = '.' \text{ xor } '*'$

2. Khai triển dự án

2.1 Ý tưởng thuật toán:

Lấy mỗi lần 8 kí tự sau đó lưu vào lần lượt thanh ghi \$s1,\$s2 , parity sẽ lưu vào \$s3 , ta chỉ cần hoán vị địa chỉ của Disk1, Disk2, Disk3 vào 3 thanh ghi trên và lần lượt in ra chúng theo yêu cầu của đề bài.

2.2 Phân tích bài toán :

2.2.1 type:

li \$v0, 8

la \$a0, input

li \$a1, 100

syscall

Yêu cầu nhập vào một xâu kí tự sau đó lưu vào địa chỉ cơ sở input

2.2.2 retype:

li \$v0, 4

la \$a0, Nhaplai

syscall

Chương trình này in ra chuỗi nhaplai đã được tiền xử lí trong .data, chương trình được gọi khi yêu cầu nhập trước đó nhận được 1 string có độ dài không chia hết cho 8

2.2.3 loop:

```

addi $t2, $t2, 1
lb $t1, 0($a0)
addi $a0, $a0, 1
bne $t0, $t1, loop
addi $t2, $t2, -1
srl $t0, $t2, 3
sll $t0, $t0, 3
bne $t0, $t2, retype

```

Nửa trên ta đếm độ dài chuỗi sau đó lưu vào \$t2, nửa dưới sẽ kiểm tra xem độ dài chuỗi có chia hết cho 8 không, nếu không ta jump tới *retype*

2.2.4 header:

```

li $v0, 4
la $a0, Header
add $a1, $t2, $zero
syscall
la $s0, input
In ra phần như trong hình

```

Nhập chuỗi kí tự : 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
-----	-----	-----

2.2.5. save:

Ở đây tôi sẽ giải thích từng phần, tôi sẽ thao tác trên 3 thanh ghi \$s1, \$s2, \$s3 như ý tưởng tôi đã nêu ở đầu

```
li $t0, 4
```

```
add $t6, $s0, $zero
```

```
addi $t7, $t6, 4
```

Phần này tôi dùng \$t6,\$t7 để lưu địa chỉ cơ sở để lấy được 4 kí tự đầu tiên và 4 kí tự tiếp theo,\$t0 để khởi tạo cho vòng for bên dưới

```
loopS:
```

```
lb $t1, 0($t6)
```

```
lb $t2, 0($t7)
```

```
xor $t3, $t1, $t2
```

```
sb $t1, 0($s1)
```

```
sb $t2, 0($s2)
```

```
sb $t3, 0($s3)
```

```
addi $t0, $t0, -1
```

```
addi $t6, $t6, 1
```

```
addi $t7, $t7, 1
```

```
addi $s1, $s1, 1
```

```
addi $s2, $s2, 1
```

```
addi $s3, $s3, 1
```

```
bne $t0, $zero, loopS
```

```
jr $ra
```

Phần này tôi lưu lần lượt các kí tự vào thanh ghi \$s1,\$s2,sau đó tính xor của chúng rồi lưu vào \$s3

2.2.6. normal:

li \$v0, 4

la \$a0, lNormal

syscall

add \$a0, \$zero, \$t0

syscall

la \$a0, rNormal

syscall

la \$a0, kc

syscall

jr \$ra

Phần này tôi in ra giá trị của thanh ghi không phải parity , ví dụ như trong hình (những phần khoanh bên dưới)

Nhap chuỗi kí tự : 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
-----	-----	-----

2.2.7. spec:

Phần này sẽ in những thanh ghi lưu giá trị parity

Tôi sẽ giải thích từng phần:

printHexa:

srl \$t6, \$t3, 4

add \$s6, \$zero, \$ra

jal beauti

sll \$t6,\$t3, 28

srl \$t6, \$t6,28

jal beauti

add \$ra, \$zero, \$s6

jr \$ra

beauti:

slti \$s5, \$t6, 10

beq \$s5, \$zero,skip

li \$v0, 1

add \$a0, \$zero, \$t6

syscall

jr \$ra

skip:

slti \$s5,\$t6,10

```
addi $t6, $t6, 87
```

```
li $v0, 11
```

```
add $a0, $zero, $t6
```

```
syscall
```

```
jr $ra
```

Phần này in ra số dưới dạng hexa, vấn đề lớn nhất là những số ≥ 10 sẽ không hiển thị là a,b,c,d,e,f tương ứng, vì thế ta có phần beauti

Nếu như số đó nhỏ hơn 10 thì sẽ gọi skip mà không biến đổi nó thành các chữ cái tương ứng rồi in ra.

Những phần save,normal,spec đều được thực hiện bằng jal nên nếu bên trong có gọi jal khác thì cần lưu lại giá trị của \$ra hiện tại để sau đó còn trả lại , bạn nên chú ý điều này

2.2.8.End:

```
li $v0, 4
```

```
la $a0, Footer
```

```
syscall
```

```
li $v0, 10
```

```
syscall
```

In ra phần như trong hình

Nhap chuoì 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
-----	-----	-----

2.2.9.Main

Main:

```
beq $a1, $zero, End
```

```
#load du lieu
```

```

la $s1, disk1
la $s2, disk2
la $s3, disk3
jal save
#in ra hang 1
la $t0, disk1
jal normal
la $t0, disk2
jal normal
la $t0, disk3
jal spec
li $t1, '\n'
li $v0, 11
add $a0, $zero, $t1
syscall
addi $s0, $s0, 8
addi $a1, $a1, -8
beq $a1, $zero, End
#load du lieu
la $s1, disk1
la $s2, disk3
la $s3, disk2
jal save
#in ra hang 2
la $t0, disk1
jal normal
la $t0, disk2

```



```

jal spec
la $t0, disk3
jal normal
li $t1, '\n'
li $v0, 11
add $a0, $zero, $t1
syscall

addi $s0, $s0, 8
addi $a1, $a1, -8
beq $a1, $zero, End
#load du lieu
la $s1, disk2
la $s2, disk3
la $s3, disk1
jal save
# in ra hang 3
la $t0, disk1
jal spec
la $t0, disk2
jal normal
la $t0, disk3
jal normal
li $t1, '\n'
li $v0, 11
add $a0, $zero, $t1
syscall

```

```
addi $s0, $s0, 8
```

```
addi $a1, $a1, -8
```

```
j Main
```

Lấy 3 lần 8 kí tự trong chuỗi nhập vào, kiểm tra xem chuỗi đã hết chưa sau mỗi lần lấy, mỗi lần lấy ta sẽ xáo trộn lại thành parity từ disk3, disk2 rồi disk1, sau đó in ra tương ứng.

2.3 Mã nguồn

```
.data
```

```
Message: .asciiz "Nhập chuỗi kí tự : "
```

```
Nhaplai: .asciiz "Độ dài chuỗi kí tự phải chia hết cho 8 : "
```

```
Header: .asciiz "   Disk 1           Disk 2           Disk 3 \n -----  
----- \n"
```

```
Footer: .asciiz " -----           -----           ----- "
```

```
lNormal: .asciiz "|   "
```

```
rNormal: .asciiz "   |"
```

```
kc: .asciiz "   "
```

```
lSpec: .asciiz "[[ "
```

```
rSpec: .asciiz "]]"
```

```
input: .space 100 # Bộ đệm lưu chuỗi đầu vào
```

```
disk1: .space 100 # Không gian lưu trữ cho Disk 1
```

```
disk2: .space 100 # Không gian lưu trữ cho Disk 2
```

```
disk3: .space 100 # Không gian lưu trữ cho Disk 3
```

```
buffer: .space 100
```

```
.text
```

```
# in ra chuỗi kí tự Message
```

```
li $v0, 4
```

```
la $a0, Message
```

```
syscall
```

```
j type
```

```
# đọc vào chuỗi kí tự đầu vào
```

```
retype:
```

```
li $v0, 4
```

```
la $a0, Nhaplai
```

```
syscall
```

```
type:
```

```
li $v0, 8
```

```

    la $a0, input
    li $a1, 100
    syscall
#Kiem tra do dai xau ki tu, neu khong chia het cho 8 yeu cau nhap lai
    li $t0, '\n'
    li $t2, 0
loop:
    addi $t2, $t2, 1
    lb $t1, 0($a0)
    addi $a0, $a0, 1
    bne $t0, $t1, loop
    addi $t2, $t2, -1

    srl $t0, $t2, 3
    sll $t0, $t0, 3
    bne $t0, $t2, retype

#in ra header
header:
    li $v0, 4
    la $a0, Header
    add $a1, $t2, $zero
    syscall
    la $s0, input
Main:
    beq $a1, $zero, End
    #load du lieu
    la $s1, disk1
    la $s2, disk2
    la $s3, disk3
    jal save
    #in ra hang 1
    la $t0, disk1
    jal normal
    la $t0, disk2
    jal normal
    la $t0, disk3
    jal spec
    li $t1, '\n'
    li $v0, 11

```

```

add $a0, $zero, $t1
syscall
addi $s0, $s0, 8
addi $a1, $a1, -8
beq $a1, $zero, End
#load du lieu
la $s1, disk1
la $s2, disk3
la $s3, disk2
jal save
#in ra hang 2
la $t0, disk1
jal normal
la $t0, disk2
jal spec
la $t0, disk3
jal normal
li $t1, '\n'
li $v0, 11
add $a0, $zero, $t1
syscall

```

```

addi $s0, $s0, 8
addi $a1, $a1, -8
beq $a1, $zero, End
#load du lieu
la $s1, disk2
la $s2, disk3
la $s3, disk1
jal save
# in ra hang 3
la $t0, disk1
jal spec
la $t0, disk2
jal normal
la $t0, disk3
jal normal
li $t1, '\n'
li $v0, 11
add $a0, $zero, $t1

```

```

        syscall

        addi $s0, $s0, 8
        addi $a1, $a1, -8
        j Main
# in ra footer
End:
        li $v0, 4
        la $a0, Footer
        syscall

        li $v0, 10
        syscall

save:
        li $t0, 4
        add $t6, $s0, $zero
        addi $t7, $t6, 4

loopS:
        lb $t1, 0($t6)
        lb $t2, 0($t7)
        xor $t3, $t1, $t2

        sb $t1, 0($s1)
        sb $t2, 0($s2)
        sb $t3, 0($s3)

        addi $t0, $t0, -1
        addi $t6, $t6, 1
        addi $t7, $t7, 1
        addi $s1, $s1, 1
        addi $s2, $s2, 1
        addi $s3, $s3, 1
        bne $t0, $zero, loopS
        jr $ra

normal:
        li $v0, 4

```

```
la $a0, lNormal  
syscall
```

```
add $a0, $zero, $t0  
syscall
```

```
la $a0, rNormal  
syscall
```

```
la $a0, kc  
syscall
```

```
jr $ra
```

spec:

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

```
li $t1, 3  
li $t2, ','  
add $s4, $zero, $ra
```

loopSpec:

```
lb $t3, 0($t0)
```

```
jal printHexa
```

```
li $v0, 11  
add $a0, $zero, $t2  
syscall
```

```
addi $t0, $t0, 1  
addi $t1, $t1, -1  
bne $t1, $zero, loopSpec
```

```
lb $t3, 0($t0)
```

```
jal printHexa
```

```
li $v0, 4  
la $a0, rSpec
```

syscall

**la \$a0, kc
syscall**

**add \$ra, \$zero, \$s4
jr \$ra**

printHexa:

srl \$t6, \$t3, 4

**add \$s6, \$zero, \$ra
jal beauti**

**sll \$t6, \$t3, 28
srl \$t6, \$t6, 28**

jal beauti

**add \$ra, \$zero, \$s6
jr \$ra**

beauti:

**slti \$s5, \$t6, 10
beq \$s5, \$zero, skip
li \$v0, 1
add \$a0, \$zero, \$t6
syscall
jr \$ra**

skip:

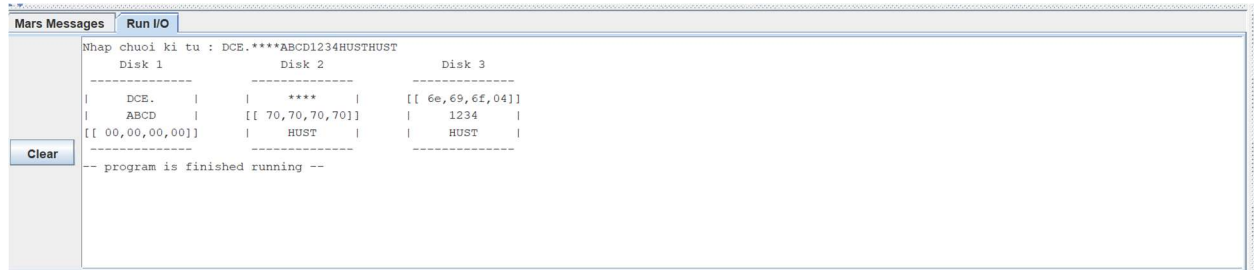
**slti \$s5, \$t6, 10
addi \$t6, \$t6, 87
li \$v0, 11
add \$a0, \$zero, \$t6
syscall
jr \$ra**

3. Demo sản phẩm

3.1 Các trường hợp ngoại lệ

Không có trường hợp ngoại lệ vì đã chuẩn hóa đầu vào từ đầu

3.2 Kết quả



The screenshot shows a window titled "Mars Messages" with a "Run I/O" button. The output text is as follows:

```
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  |
-----
-- program is finished running --
```

On the left side of the window, there is a "Clear" button.