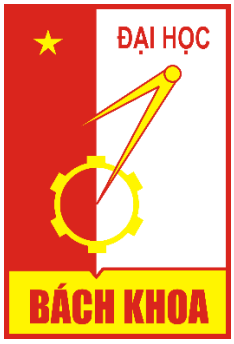
ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

----- ✿ 🕮 ✿ -----



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	Nguyễn Trí Dũng	20225613
2	Nguyễn Hồng Phúc	20225659

Hà Nội, ngày 12 tháng 6 năm 2024

Mục Lục

Bài 3: Kiểm tra tốc độ và độ chính xác khi gõ văn bản

Sinh viên thực hiện: Nguyễn Trí Dũng – 20225613

A. Đề bài

Thực hiện chương trình đo tốc độ gõ bàn phím và hiển thị kết quả bằng 2 đèn led 7 đoạn. Nguyên tắc:

- Cho một đoạn văn bản mẫu, cố định sẵn trong mã nguồn. Ví dụ *"bo mon ky thuat may tinh"*
- Sử dụng bộ định thời Timer (trong bộ giả lập Digital Lab Sim) để tạo ra khoảng thời gian để đo. Đây là thời gian giữa 2 lần ngắt, chu kì ngắt.
- Người dùng nhập các kí tự từ bàn phím. Ví dụ nhập "*bo m**O**n ky **5**huat may tinh*". Chương trình cần phải đếm số kí tự đúng (trong ví dụ trên thì người dùng gõ sai chữ **O** và **5**) mà người dùng đã gõ và hiển thị lên các đèn led.
- Chương trình đồng thời cần tính được tốc độ gõ: thời gian hoàn thành và số từ trên một đơn vị thời gian.

B. Phân tích bài toán

Cho một đoạn văn bản sẵn trong mã nguồn. Người dùng nhập các ký tự từ bàn phím. Chường trình sẽ đếm số ký tự đúng và hiện thị lên các đèn led, đồng thời hiển thị thời gian và tốc độ gõ bàn phím.

C. Cách làm và thuật toán

- Sử dụng 1 vòng lặp vô hạn.
- Trong vòng lặp có kiểm tra giá trị tại địa chỉ KEY_READY nếu bằng 1 thì thực hiện tạo ngắt bằng teqi.
- Đồng thời chương trình cũng cho phép ngắt bằng bộ đếm time counter(timer).
- Khi đã bắt được exception và con trỏ $pc nhảy đến vùng phục vụ ngắt .ktext.
- Bên trong vùng .ktext ta sẽ lấy giá trị bên trong thanh ghi Coproc0.cause($13) để kiểm tra đây là loại ngắt nào.

– Trong trường hợp lệnh ngắt được thực hiện bởi teqi(tạo ra khi nhận được ký tự từ bàn phím).

– Ta kiểm tra xem ký tự thứ i của string đã cho có phải là ký tự kết thúc hay không('\0'), nếu đúng thì kết thúc chương trình, hiển thị ra số ký tự đúng lên Digital Lab Sim và in ra thời gian hoàn thành, tốc độ gõ lên màn hình.

– Nếu chương trình chưa kết thúc, ta so sánh ký tự vừa nhập với ký tự string[i] nếu bằng nhau → tăng biến đếm số ký tự đúng lên 1.

– Tiếp tục kiểm tra xem ký tự vừa nhập vào == ' ' && ký nhập vào trước đó != ' ' nếu đúng thì tăng biến đếm số từ đã nhập lên 1.

– Sau đó tăng số ký tự nhập vào trong 1s lên 1, cập nhật giá trị của prv bằng ký tự vừa nhập vào và chuyển con trỏ của string lên 1 để phục vụ cho lần sau.

– Trong trường hợp lệnh ngắt được thực hiện bởi bộ đếm time counter (timer).

– Kiểm tra xem số lần tạo lệnh ngắt của timer đã đủ chưa (1s), nếu chưa đủ thì tăng biến đếm lên, nếu đã đủ thì hiển thị số ký tự đã gõ trong 1s lên Digital Lab Sim và khởi tạo lại biến đếm ký tự trong 1s, đồng thời tăng biến đếm thời gian hoàn thành nhập lên 1s.

– Sau khi hoàn thành các câu lệnh trong vùng .ktext.

– Thực hiện các hàm cần thiết để thiết lập lại các thông số để đón nhận lần ngắt tiếp theo.

D. Ý nghĩa các thanh ghi trong chương trình

$k0 : Ký tự nhập vào từ bàn phím

$k1: Kiểm tra bàn phím sẵn sàng

$t1 : Bộ đếm thời gian

$s0 : Đếm số ký tự từng giây

$s1 : Số ký tự đúng

$s2 : Tổng số ký tự nhập vào

$s3 : Tổng số lần ngắt Counter

$s4 : Lưu ký tự trước đó

$s5 : Tổng thời gian chạy chương trình

$a1 : Địa chỉ chuỗi cần so sánh

E. Mã nguồn

```
.eqv SEVENSEG_LEFT 0xFFFF0011          #Địa chỉ của đèn led 7 đoạn trái

.eqv SEVENSEG_RIGHT 0xFFFF0010         # Địa chỉ của đèn led 7 đoạn phải

.eqv IN_ADDRESS_HEXA_KEYBOARD 0xFFFF0012  #Địa chỉ bàn phím

.eqv MASK_CAUSE_COUNTER 0x00000400 #Bit 10: Counter interrupt

.eqv COUNTER 0xFFFF0013              #Time Counter

.eqv KEY_CODE   0xFFFF0004           #ASCII code from keyboard, 1 byte

.eqv KEY_READY  0xFFFF0000           #=1 if has a new keycode?

.data

mang_so: .byte        63, 6,  91, 79, 102, 109 ,125, 7, 127, 111   #mảng chứa số 0-9 để hiển thị ra 7segs LED (mỗi số 1 byte)

string: .asciiz "Bo mon ky thuat may tinh"

message1: .asciiz "Tổng thời gian chạy chương trình "

message2: .asciiz "(s) \nTốc độ gõ trung bình: "

message3: .asciiz " tu/phut\n"

Continue: .asciiz "Tiếp tục nhập?"

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# MAIN

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
.text
MAIN:
        # Kích hoạt ngắt COUNTER
        li      $k0,  KEY_CODE
        li      $k1,  KEY_READY
        li      $t1, COUNTER
        sb      $t1, 0($t1)             #Kích hoạt
        addi    $s0, $0, 0              #Đếm số ký tự từng giây
        addi    $s1, $0, 0              #Số ký tự đúng
        addi    $s2, $0, 0              #Tổng số ký tự nhập vào
        addi    $s3, $0, 0              #Tổng số lần ngắt Counter
        addi    $s4, $0, 0              #Lưu ký tự trước đó
        addi    $s5, $0, 0              #Tổng thời gian chạy chương trình
        la      $a1, string             #Chuỗi cần so sánh
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#Vòng lặp vô hạn
loop:
        lw      $t1, 0($k1)                     #$t1 = [$k1] = KEY_READY
        bne     $t1, $zero, make_Keyboard_Intr  #tạo interrupt khi có phím nhấn vào
        addi    $v0, $0, 32
        li      $a0, 5                          #Ngủ 5ms
```

```
        syscall
        b       loop                    #Cứ lặp 5 lần tạo 1 counter interrupt
        nop
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
make_Keyboard_Intr:
        teqi    $t1, 1                  #Check để nhảy vào ngắt .ktext
        b       loop                    # Khi return từ chương trình ngắt thì kiểm tra tiếp char tiếp theo nhập vào
        nop
end_Main:
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#PHẦN PHỤC VỤ NGẮT
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
.ktext 0x80000180
dis_int:li      $t1, COUNTER            #Vô hiệu hoá bộ đếm thời gian
        sb      $zero, 0($t1)
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#LAY GIA TRI CUA THANH GHI C0.cause DE KIEM TRA LOAI INTERRUPT
get_Caus:mfc0   $t1, $13                #$t1 = Coproc0.cause
isCount:li      $t2, MASK_CAUSE_COUNTER     #if Cause value confirm Counter..
        and     $at, $t1,$t2
        bne     $at,$t2, keyboard_Intr      #Kiểm tra xem nếu không phải do bộ đếm thì nhảy đến ngắt bàn phím
```

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

#NGẮT DO BỘ ĐẾM COUNTER

counter_Intr:

        blt     $s3, 40, continue       #40 lần ngắt tương đương 40 lần ngủ 5ms -> tổng là 200 ms ngủ -> 40 lần ngắt tương đương 1 giây chương trình-> khoi tao lai $s3, chieu toc do go ra DLS, tăng biến đếm thời gian lên 1

        jal     hien_thi                #Nếu đủ 1s thì nhảy đến phần hiển thị

        addi    $s3, $0, 0              # reset số lần ngắt

        addi    $s5, $s5, 1             # Tăng thời gian chạy

        j       en_int

        nop

continue:

        addi    $s3, $s3, 1             # Tăng số lần ngắt nếu chưa đủ 1s

        j       en_int                  #

        nop

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

#NGẮT DO BÀN PHÍM

keyboard_Intr:

kiem_tra_ky_tu:                         #Kiểm tra ký tự nhập vào

        lb      $t0, 0($a1)             #Get char string test case

        lb      $t1, 0($k0)             #Lấy kí tự nhập vào

        beq     $t1, $0, en_int                 #Check ở string test case tới NULL thì dừng

        beq     $t1, '\n', end_Program          #Kí tự là “\n” tiến hành kiểm tra và in
```

```
        bne     $t0, $t1, kiem_tra_dau_cach         #Nếu kí tự nhập vào và kí tự thứ i trong mảng giống nhau -> đếm số ký tự đúng , nếu không thì chuyển đến phần kiểm tra dấu cách

        nop

        addi    $s1, $s1, 1                 #tăng biến đếm số ký tự đúng

kiem_tra_dau_cach:                          #kiểm tra nhập vào có phải dấu cách không

#---------------------------------------------------------------

# Kiểm tra để đếm số TỪ nhập vào:

# Nếu nhập vào space cần kiểm tra 2 TH

# TH1: trước space vừa nhập là 1 char -> Tính +1 từ

# TH2: trước space vừa nhập cũng là 1 space -> Không tính từ -> về main

# Nếu KHÔNG nhập vào space thì tiếp tục đọc ký tự nhập vào tiếp (về main)

#---------------------------------------------------------------

        bne     $t1, ' ', end_Process       #Nếu ký tự nhập vào == ' ' && ký tự trước đó != ' ' thì đếm số từ đã nhập

        nop

        beq     $s4, ' ', end_Process

        nop

        addi    $s2, $s2, 1                 #Tăng số từ +1

end_Process:

        addi    $s0, $s0, 1                 #Tang so ky tu trong 1s len 1

        addi    $s4, $t1, 0                 # Save lại char vừa check

        addi    $a1, $a1, 1                 # Tăng con trỏ string test case thêm 1 byte

        j en_int
```

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# CHIẾU RA MÀN HÌNH DIGITAL LAB SIM SỐ KÝ TỰ ĐÚNG MỖI GIÂY

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

hien_thi:

        addi    $sp, $sp, -4                #Giảm giá trị 4 byte tăng kích thước ngăn
xếp  để lưu trữ giá trị mới

        sw      $ra, ($sp)                  #Lưu giá trị của thanh ghi ra ( địa chỉ trả về )
vào địa chỉ của $sp

        addi    $t0, $0, 10                 #gán giá trị 10 cho $t0

        div     $s0, $t0                    #$s0 là số ký tự đúng cần chia 10 để lấy
hàng chục và hàng đơn vị hiển thị ra đèn led

        mflo    $v1                         # Lấy số hàng chục

        mfhi    $v0                         # Lấy số hàng đơn vị

        la      $a0, mang_so                # lấy địa chỉ mảng số

        add     $a0, $a0, $v1               #Lấy địa chỉ có giá trị  hàng chục để hiển thị

        lb      $a0, 0($a0)                 #Set value for segments

        jal     SHOW_7SEG_LEFT                      #Hien thi

        la      $a0, mang_so

        add     $a0, $a0, $v0               #Lấy địa chỉ có giá trị  hàng đơn vị để hiển
thị

        lb      $a0, 0($a0)                 #Set value for segments

        jal     SHOW_7SEG_RIGHT             #Hien thi

        addi    $s0, $0, 0                  #Sau khi chiếu ra màn hình thì khởi tạo lại
biến đếm $s0

        lw      $ra, ($sp)
```

```
        addi    $sp, $sp, 4
        jr      $ra                     #Return về ngắt
SHOW_7SEG_LEFT:
        li      $t0, SEVENSEG_LEFT      #Assign port's address
        sb      $a0, 0($t0)             #Assign new value
        jr      $ra
SHOW_7SEG_RIGHT:
        li      $t0, SEVENSEG_RIGHT     #Assign port's address
        sb      $a0, 0($t0)             #Assign new value
        jr      $ra
        nop
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# KET THUC CHUONG TRINH VA HIEN THI SO KY TU DUNG
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
end_Program:
        addi    $v0, $0, 4
        la      $a0, message1
        syscall
        addi    $v0, $0, 1
        addi    $a0, $s5, 0             #in ra giá trị thời gian chạy chương trình
        syscall
        addi    $v0, $0, 4
```

```
        la      $a0, message2
        syscall
        addi    $v0, $0, 1
        addi    $a0, $0, 60
        mult    $s2, $a0            #Số từ nhân 60
        mflo    $s2
        div     $s2, $s5            #Số từ / tổng thời gian(s)
        mflo    $a0
        syscall
        addi    $v0, $0, 4
        la      $a0, message3
        syscall
        addi    $s0, $s1, 0         #Gán lại tổng số từ đúng cho $s0
        jal     hien_thi            #Hiển thị kết quả cuối cùng ra đèn led
CONTINUE:
        li $v0, 50
        la $a0, Continue            #Tiep tuc nhap hay khong
        syscall
        beq $a0, 0, MAIN            # So sánh giá trị của thanh ghi $a0 với 0 và nhảy đến nhãn MAIN nếu chúng bằng nhau.
        li $v0, 10                  # Ket thuc chuong trinh
        syscall
```

```
#Kết thúc chương trình ngắt quay về main cần tăng epc cho đúng về chương trình main

en_int:
        li      $t1, COUNTER            #Kích hoạt lại ngắt bộ đếm thời gian
        sb      $t1, 0($t1)
        mtc0    $zero, $13              #Must clear cause reg
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
return: eret                            #Return from exception
```

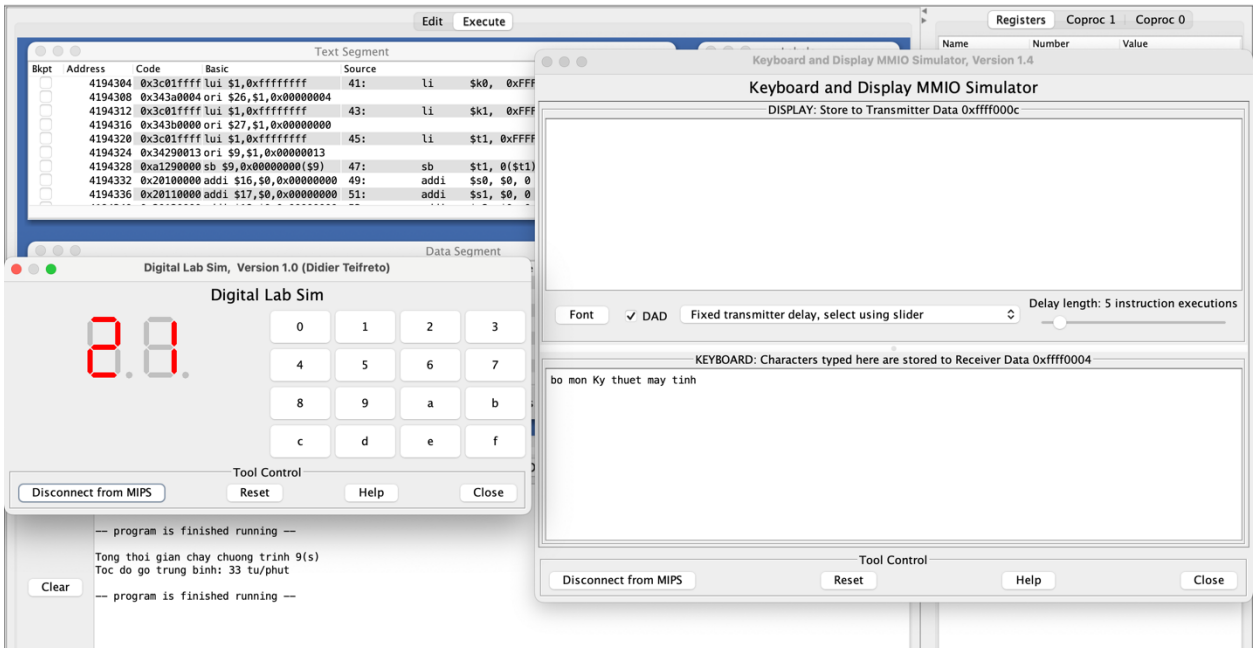
F. Kết quả chạy trên Mars Mips

- Mở phần mềm Mars Mips.
- Mở "Digital Lab Sim" và "Keyboard and Display MMIO Simulator" ở phần Tools.
- Kết nối 2 tool này với mips.
- Chạy chương trình và nhập chuỗi ký tự vào.
- Sau khi nhập xong thì chương trình sẽ hiển thị số từ đúng ở Digital Lab Sim đồng thời hiển thị thời gian chạy trung bình và tốc độ gõ trung bình ở màn hình Run I/O.

Với chuỗi ký tự nhập vào là "Bo mon ky thuat may tinh".

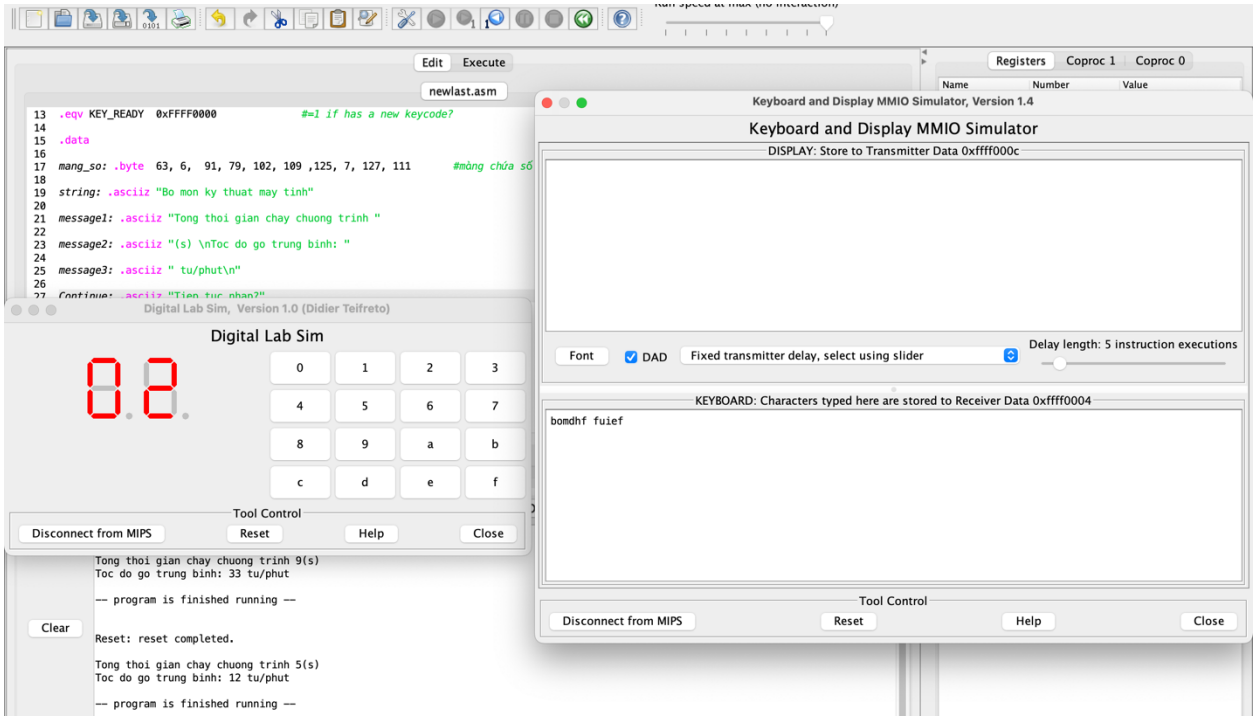
Chạy lần 1:

- Nhập vào "bo mon Ky thuet may tinh".
- Kết quả hiển thị 21 từ đúng và thời gian chạy trung bình 9(s), tốc độ gõ trung bình 33 từ/phút.



Chạy lần 2:

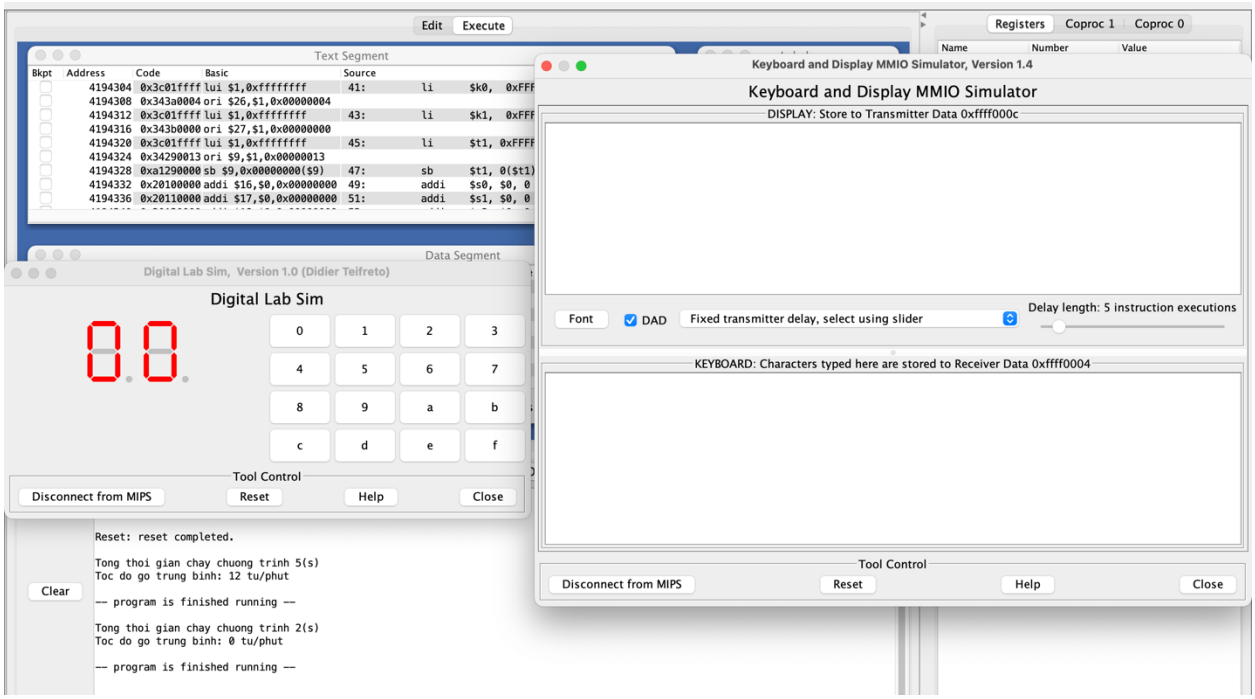
- Nhập vào “bomdhf fuief”.
- Kết quả hiển thị 02 từ đúng và thời gian chạy trung bình 5(s), tốc độ gõ trung bình 12 từ/phút.



Chạy lần 3:

- Không nhập ký tự nào.

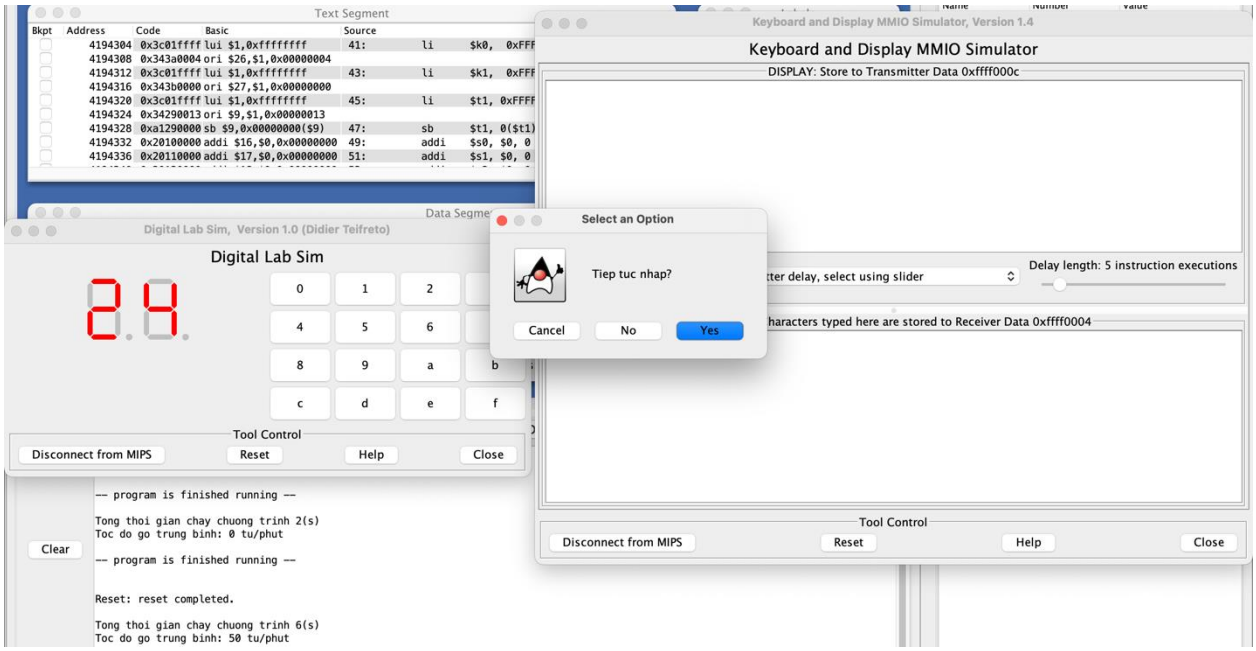
– Kết quả hiển thị 00 từ đúng và thời gian chạy trung bình 2(s), tốc độ gõ trung bình 0 từ/phút.



Chạy lần 4:

– Nhập vào chuỗi “Bo mon ky thuat may tinh”.

– Kết quả hiển thị 0 từ đúng và thời gian chạy trung bình 6(s), tốc độ gõ trung bình 50 từ/phút.



Bài 8: Mô phỏng ổ đĩa RAID 5

Sinh viên thực hiện: Nguyễn Hồng Phúc – 20225659

A. Đề bài

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 |
   ----------            ----------            ----------
```

B. Phân tích bài toán

1. Nhập 1 dãy kí tự vào sao cho độ dài của chuỗi phải chia hết cho 8. Nếu chuỗi không có độ dài hợp lệ thì mời nhập lại.
2. Đầu ra là mô phỏng ổ đĩa RAID 5

C. Ý tưởng thuật toán

Chương trình có 3 phần chính:

- Hàm tính độ dài của chuỗi kí tự nhập vào

- Hàm RAID 5 được chia thành 3 phần:

+ Phần 1 lưu chuỗi kí tự vào disk 1 và disk 2, lưu 4 byte parity xor từ 4 byte disk 1 và 4 byte disk 2 vào disk 3.

+ Phần 2 lưu chuỗi kí tự vào disk 1 và disk 3, lưu 4 byte parity xor từ 4 byte disk 1 và 4 byte disk 3 vào disk 2.

+ Phần 3 lưu chuỗi kí tự vào disk 2 và disk 3, lưu 4 byte parity xor từ 4 byte disk 2 và 4 byte disk 3 vào disk 1.

Sau khi lưu các block vào 3 phần này mà chuỗi ký tự chưa hết thì lại quay lại phần 1.

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

D. 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 parity

E. Mã nguồn chương trình

```
.data
        start: .asciiz "Nhap chuoi ky tu : "
        hex: .byte '0','1','2','3','4','5','6','7','8','9','a','b','c','d','e','f'
        d1: .space 4
        d2: .space 4
        d3: .space 4
        array: .space 32
        string: .space 5000
        enter: .asciiz "\n"
        error_length: .asciiz "Do dai chuoi khong chia het cho 8! Nhap lai.\n"
        m: .asciiz "      Disk 1              Disk 2              Disk 3\n"
        m2: .asciiz "----------------      ----------------      ----------------\n"
        m3: .asciiz "|     "
        m4: .asciiz "     |       "
        m5: .asciiz "[[ "
        m6: .asciiz "]]       "
        comma: .asciiz ","
        ms: .asciiz "Try again?"

.text
        la      $s1, d1                 # Tuong ung disk 1
```

```
        la      $s2, d2                 # Tuong ung disk 2
        la      $s3, d3                 # Tuong ung disk 3
        la      $a2, array                      # dia chi mang chua parity

input:  li      $v0, 4                  # Moi nhap chuoi ky tu
        la      $a0, start
        syscall
        li      $v0, 8
        la      $a0, string
        li      $a1, 1000
        syscall
        move    $s0, $a0                # s0 chua dia chi xau moi nhap
        li      $v0, 4
        la      $a0, m
        syscall
        li      $v0, 4
        la      $a0, m2
        syscall

#----------------------kiem tra do dai co chia het cho 8 khong-------------------------
length: addi    $t3, $zero, 0           # t3 = length
        addi    $t0, $zero, 0           # t0 = index

check_char:
        add $t1, $s0, $t0               # t1 = address of string[i]
        lb      $t2, 0($t1)                     # t2 = string[i]
        nop
        beq     $t2, 10, test_length    # t2 = '\n' ket thuc xau
        nop
        addi    $t3, $t3, 1                     # length++
        addi    $t0, $t0, 1                     # index++
        j       check_char
```

```
        nop
test_length:
        move $t5, $t3
        and     $t1, $t3, 0x0000000f        # xoa het cac byte cua $t3 ve 0, chi giu lai byte
cuoi
        bne     $t1, 0, test1               # byte cuoi bang 0 hoac 8 thi so chia het cho 8
        j       split1
test1:  beq     $t1, 8, split1
        j       error1
error1:li       $v0, 4
        la      $a0, error_length
        syscall
        j       input
#------------------------------ket thuc kiem tra do dai---------------------------------


#-----------------------lay 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 => lay byte cuoi cung cua a0
        la      $t7, hex
        add     $t7, $t7, $a0
        bgt     $t4, 1, nextc
        lb      $a0, 0($t7)                 # print hex[a0]
        li      $v0, 11
        syscall
nextc:  addi    $t4,$t4,-1
        j       loopH
endloopH: jr    $ra
#------------------------------------------------------------
```

```
#------------------------------Mo phong RAID 5-------------------------------
# xet 6 khoi dau
# lan 1: luu vao 2 disk 1,2; xor disk 3-------
split1:
        addi    $t0, $zero, 0                   # so byte duoc in ra (4 byte)
        addi    $t6, $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)              #Bắt đầu vòng lặp
        addi    $t3, $t3, -1
        sb      $t1, ($s1)
b21:    add     $s5, $s0, 4
        lb      $t2, ($s5)                  # t2 chua dia chi tung byte cua 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    $t6, $t6, 1
        addi    $s1, $s1, 1
        bgt     $t6, 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    $t6, $zero, 0
print14:lb      $t8, ($a2)
        jal     HEX
        li      $v0, 4
        la      $a0, comma
```

```
        syscall
        addi    $t6, $t6, 1
        addi    $a2, $a2, 4
        bgt     $t6, 2, end1                    # in ra 3 parity dau co dau ",", parity cuoi cung k
co
        j       print14
end1:   lb      $t8, ($a2)
        jal HEX
        li $v0, 4
        la $a0, m6
        syscall
        li $v0, 4
        la $a0, enter
        syscall
        beq $t3, 0, exit1
#-------------------------------------------------------
# lan 12: luu vao 2 khoi 1,3; xor vao 2-------
split2:  la $a2, array
        la $s1, d1
        la $s3, d3
        addi $s0, $s0, 4
        addi $t0, $zero, 0
print21:li $v0, 4
        la $a0, m3
        syscall
b12:    lb $t1, ($s0)
        addi $t3, $t3, -1
        sb $t1, ($s1)
b32:    add $s5, $s0, 4
        lb $t2, ($s5)
        addi $t3, $t3, -1
        sb $t2, ($s3)
```

```
b22:    xor $a3, $t1, $t2
        sw $a3, ($a2)
        addi $a2, $a2, 4
        addi $t0, $t0, 1
        addi $s0, $s0, 1
        addi $s1, $s1, 1
        addi $s3, $s3, 1
        bgt $t0, 3, reset2
        j b12
reset2: la $s1, d1
        la $s3, d3
        addi $t6, $zero, 0
print22:lb $a0, ($s1)
        li $v0, 11
        syscall
        addi $t6, $t6, 1
        addi $s1, $s1, 1
        bgt $t6, 3, next21
        j print22
next21:li $v0, 4
        la $a0, m4
        syscall
        la $a2, array
        addi $t6, $zero, 0
        li $v0, 4
        la $a0, m5
        syscall
print23:lb $t8, ($a2)
        jal HEX
        li $v0, 4
        la $a0, comma
        syscall
```

```
        addi $t6, $t6, 1
        addi $a2, $a2, 4
        bgt $t6, 2, next22
        j print23
next22:lb $t8, ($a2)
        jal HEX
        li $v0, 4
        la $a0, m6
        syscall
        li $v0, 4
        la $a0, m3
        syscall
        addi $t8, $zero, 0
print24:lb $a0, ($s3)
        li $v0, 11
        syscall
        addi $t8, $t8, 1
        addi $s3, $s3, 1
        bgt $t8, 3, end2
        j print24

end2:   li $v0, 4
        la $a0, m4
        syscall
        li $v0, 4
        la $a0, enter
        syscall
        beq $t3, 0, exit1
#-------------------------------------------------------
# lan 3: luu vao 2 khoi 2,3; xor vao 1-------
split3:la $a2, array
        la      $s2, d2
```

```
        la      $s3, d3
        addi    $s0, $s0, 4
        addi    $t0, $zero, 0
print31:li      $v0, 4
        la      $a0, m5
        syscall
b23:    lb      $t1, ($s0)
        addi    $t3, $t3, -1
        sb      $t1, ($s1)
b33:    add     $s5, $s0, 4
        lb      $t2, ($s5)
        addi    $t3, $t3, -1
        sb      $t2, ($s3)
b13:    xor     $a3, $t1, $t2
        sw      $a3, ($a2)
        addi    $a2, $a2, 4
        addi    $t0, $t0, 1
        addi    $s0, $s0, 1
        addi    $s1, $s1, 1
        addi    $s3, $s3, 1
        bgt     $t0, 3, reset3
        j       b23
reset3:la       $s2, d2
        la      $s3, d3
        la      $a2, array
        addi    $t6, $zero, 0
print32:lb      $t8, ($a2)
        jal     HEX
        li      $v0, 4
        la      $a0, comma
        syscall
        addi    $t6, $t6, 1
```

```
        addi    $a2, $a2, 4
        bgt     $t6, 2, next31
        j       print32
next31:lb       $t8, ($a2)
        jal HEX
        li      $v0, 4
        la      $a0, m6
        syscall
        li      $v0, 4
        la      $a0, m3
        syscall
        addi    $t6, $zero, 0
print33:lb      $a0, ($s2)
        li      $v0, 11
        syscall
        addi    $t6, $t6, 1
        addi    $s2, $s2, 1
        bgt     $t6, 3, next32
        j       print33
next32:addi     $t6, $zero, 0
        addi    $t8, $zero, 0
        li      $v0, 4
        la      $a0, m4
        syscall
        li      $v0, 4
        la      $a0, m3
        syscall
print34:lb      $a0, ($s3)
        li      $v0, 11
        syscall
        addi    $t8, $t8, 1
        addi    $s3, $s3, 1
```

```
        bgt     $t8, 3, end3
        j       print34

end3:   li      $v0, 4
        la      $a0, m4
        syscall
        li      $v0, 4
        la      $a0, enter
        syscall
        beq     $t3, 0, exit1
#------------------------------end 6 khoi dau-------------------------------------
# chuyen sang 6 khoi tiep theo
nextloop: addi $s0, $s0, 4
        j split1

exit1:  li      $v0, 4
        la      $a0, m2
        syscall
        j       ask
#------------------------------ket thuc mo phong RAID 5---------------------------------

#-----------------------------------try again-------------------------------------
ask:    li      $v0, 50
        la      $a0, ms
        syscall
        beq     $a0, 0, clear
        nop
        j       exit
        nop
# clear: dua string ve trang thai ban dau de thuc hien lai qua trinh
clear:  la      $s0, string
        add     $s3, $s0, $t5    # s3: dia chi byte cuoi cung duoc su dung trong string
```

```
        li      $t1, 0
goAgain: sb     $t1, ($s0)              # set byte o dia chi s0 thanh 0
        nop
        addi    $s0, $s0, 1
        bge     $s0, $s3, input
        nop
        j       goAgain
        nop
#---------------------------------end try again---------------------------------------


exit:   li      $v0, 10
        syscall
```

F. Giải thích chương trình

B1: Khởi tạo các giá trị và message

B2: Load địa chỉ các disk1,disk2,disk3 và địa chỉ thanh ghi chứa parity. Nhập chuỗi kí tự vào và lưu vào thanh ghi $s0.

B3: Chuyển sang hàm tính độ dài của chuỗi kí tự. Dùng $t3 để tính giá trị độ dài. Kiểm tra đến khi gặp dấu cách thì kết thúc đếm chuỗi chuyển đến nhãn kiểm tra độ dài của chuỗi có hợp lệ hay không. Nếu độ dài không hợp lệ thì hàm chuyển đến nhãn error1 để thông báo không hợp lệ rồi quay lại bước nhập chuỗi. Nếu độ dài chuỗi hợp lệ thì hàm chuyển sang phần 1 của hàm mô phỏng RAID 5

B4: phần 1 của hàm RAID 5

split1:

```
        addi    $t0, $zero, 0           # t0 = 0: Đếm số byte đã in ra (4 byte mỗi lần).
        addi    $t6, $zero, 0           # t6 = 0: Biến đếm số kí tự in ra ở disk 1.
        addi    $t8, $zero, 0           # t8 = 0: Biến đếm số kí tự in ra ở disk 2.
        la      $s1, d1                 # s1 = địa chỉ bắt đầu của d1 (đĩa 1).
        la      $s2, d2                 # s2 = địa chỉ bắt đầu của d2 (đĩa 2).
        la      $a2, array              # a2 = địa chỉ bắt đầu của mảng lưu kết quả XOR.
b11:    lb      $t1, ($s0)              #Đọc byte từ địa chỉ s0(địa chỉ của chuỗi)
        addi    $t3, $t3, -1            #Độ dài của chuỗi trừ 1
        sb      $t1, ($s1)              # Lưu byte đọc được vào địa chỉ s1 (đĩa 1).
b21:    add     $s5, $s0, 4     #địa chỉ của s5 cách s0 4 byte(lấy 4 giá trị tiếp theo lưu vào
disk 2
```

```
        lb      $t2, ($s5)                  # t2 chua dia chi tung byte cua disk 2
        addi    $t3, $t3, -1                #Độ dài của chuỗi trừ 1
        sb      $t2, ($s2)          # Lưu byte đọc được vào địa chỉ s2 (đĩa 2).
b31:
        xor $a3, $t1, $t2       # a3 = t1 XOR t2: Tính toán XOR của hai byte.
        sw $a3, ($a2)           # Lưu kết quả XOR vào mảng.
        addi $a2, $a2, 4        # Tăng địa chỉ của mảng lên 4 byte.
        addi $t0, $t0, 1        # Tăng số byte đã in ra lên 1.
        addi $s0, $s0, 1        # Tăng địa chỉ nguồn lên 1 byte.
        addi $s1, $s1, 1        # Tăng địa chỉ đĩa 1 lên 1 byte.
        addi $s2, $s2, 1        # Tăng địa chỉ đĩa 2 lên 1 byte.
        bgt $t0, 3, reset       # Nếu đã in ra 4 byte, thì nhảy tới reset.
        j b11                   # Nếu chưa in đủ, tiếp tục vòng lặp.


reset:
        la      $s1, d1                 # Đặt lại địa chỉ của đĩa 1 về giá trị ban đầu.
        la      $s2, d2                 # Đặt lại địa chỉ của đĩa 2 về giá trị ban đầu.


print12:
        lb      $a0, ($s1)              # Đọc một byte từ địa chỉ s1 (đĩa 1) và lưu vào a0.
        li      $v0, 11                 # Đặt mã syscall thành 11 (in một ký tự).
        syscall                         # Thực hiện syscall để in ký tự từ a0.
        addi    $t6, $t6, 1             # Tăng biến đếm t6 lên 1.
        addi    $s1, $s1, 1             # Tăng địa chỉ của s1 (đĩa 1) lên 1 byte.
        bgt     $t6, 3, next11          # Nếu t6 lớn hơn 3, nhảy tới next11.(Sau khi đã in đủ 4
    byte thì nhảy đến in các msg | | để mô phỏng đĩa RAID 5)
        j       print12         # Nếu chưa in đủ 4 byte, quay lại in byte tiếp theo.
next11:
        li      $v0, 4          # Đặt mã syscall thành 4 (in chuỗi).
        la      $a0, m4         # Đặt địa chỉ của chuỗi m4 vào a0.
        syscall                 # Thực hiện syscall để in chuỗi m4.
```

```
        li      $v0, 4          # Đặt mã syscall thành 4 (in chuỗi).
        la      $a0, m3         #Đặt địa chỉ của chuỗi m3 vào a0.
        syscall                 # Thực hiện syscall để in chuỗi m3.
print13:
        lb      $a0, ($s2)              # Đọc một byte từ địa chỉ s2 (đĩa 2) và lưu vào a0.
        li      $v0, 11                 # Đặt mã syscall thành 11 (in một ký tự).
        syscall                         # Thực hiện syscall để in ký tự từ a0.
        addi    $t8, $t8, 1             # Tăng biến đếm t8 lên 1.
        addi    $s2, $s2, 1             # Tăng địa chỉ của s2 (đĩa 2) lên 1 byte.
        bgt     $t8, 3, next12          # Nếu t8 lớn hơn 3, nhảy tới next12 để in các msg.
        j       print13                 # Nếu chưa in đủ 4 byte, quay lại in byte tiếp theo.
next12:
        li      $v0, 4          # Đặt mã syscall thành 4 (in chuỗi).
        la      $a0, m4                 # Đặt địa chỉ của chuỗi m4 vào a0.
        syscall                 # Thực hiện syscall để in chuỗi m4.
        li      $v0, 4          # Đặt mã syscall thành 4 (in chuỗi).
        la      $a0, m5                 # Đặt địa chỉ của chuỗi m5 vào a0.
        syscall                 # Thực hiện syscall để in chuỗi m5.

        la      $a2, array      # Đặt lại địa chỉ a2 về array (mảng lưu kết quả XOR).
        addi    $t6, $zero, 0   # Reset biến đếm t6 về 0 để dùng cho các phần sau.
print14:
        lb      $t8, ($a2)              # Đọc một byte từ địa chỉ $a2 (mảng XOR) vào $t8.
        jal     HEX                     # Gọi hàm HEX để chuyển đổi và in giá trị hex của $t8.
        li      $v0, 4                  # Đặt mã syscall thành 4 (in chuỗi).
        la      $a0, comma
        syscall                         # Thực hiện syscall để in dấu phẩy
        addi    $t6, $t6, 1             # Tăng biến đếm $t6 lên 1.
        addi    $a2, $a2, 4             # chuyển sang phần tử tiếp theo của mảng XOR
        bgt     $t6, 2, end1    # Nếu $t6 > 2, nhảy tới end1 (sau khi đã in 3 giá trị XOR).
        j       print14         # Nếu chưa in đủ 3 giá trị, quay lại in giá trị tiếp theo.
end1:
```

```
        lb      $t8, ($a2)      # Đọc một byte từ địa chỉ $a2 (giá trị XOR cuối cùng) vào $t8.
        jal HEX                 # Gọi hàm HEX để chuyển đổi và in giá trị hex của $t8.
        li $v0, 4               # Đặt mã syscall thành 4 (in chuỗi).
        la $a0, m6              # Đặt địa chỉ của chuỗi m6 vào $a0.
        syscall                 # Thực hiện syscall để in chuỗi m6.
        li $v0, 4
        la $a0, enter
        syscall                 # Thực hiện syscall để thực hiện xuống dòng
        beq $t3, 0, exit1       # Nếu độ dài chuỗi đã giảm còn 0, nhảy tới exit1(Để thực hiện
```
hỏi liệu có muốn kết thúc chương trình hay muốn tiếp tục sử dụng chương trình).

- Hàm HEX

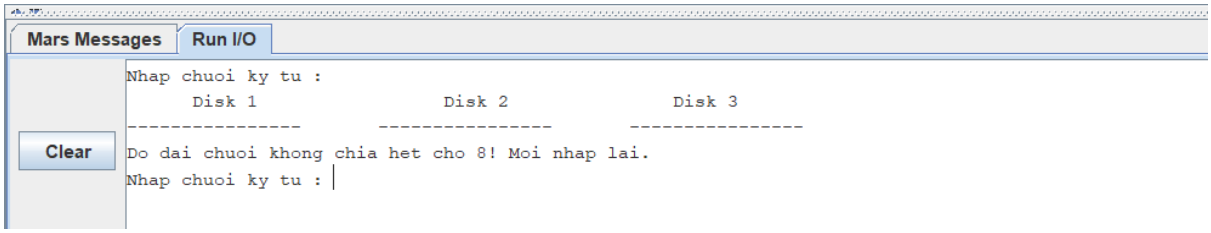
```
HEX:
        li      $t4, 7                  # Đặt $t4 bằng 7, số bit dịch chuyển cần thiết.
loopH:
        blt     $t4, $0, endloopH       # Nếu $t4 < 0, kết thúc vòng lặp.
        sll     $s6, $t4, 2             # Dịch trái $t4 lên 2 bit lưu và $s6
        srlv    $a0, $t8, $s6           # Dịch phải $t8 theo giá trị trong $s6 bit và lưu vào $a0.
        andi    $a0, $a0, 0x0000000f # Lấy 4 bit thấp nhất của $a0.
        la      $t7, hex                # Đặt địa chỉ của bảng ký tự hex vào $t7.
        add     $t7, $t7, $a0           # Thêm offset $a0 vào địa chỉ bảng ký tự hex để lấy ký
```
tự tương ứng.
```
        bgt     $t4, 1, nextc           # Nếu $t4 > 1, nhảy tới nhãn nextc.(Khi $t4=1 hoặc 0
```
mới tiến hành in ký tự)
```
        lb      $a0, 0($t7)             # Đọc ký tự từ bảng hex vào $a0.
        li      $v0, 11                 # Đặt mã syscall thành 11 (in ký tự).
        syscall                         # Thực hiện syscall để in ký tự.
nextc:
        addi    $t4, $t4, -1            # Giảm $t4 đi 1.
        j       loopH                   # Quay lại đầu vòng lặp.
endloopH:
        jr      $ra                     # Kết thúc hàm, trở về lệnh gọi.
```

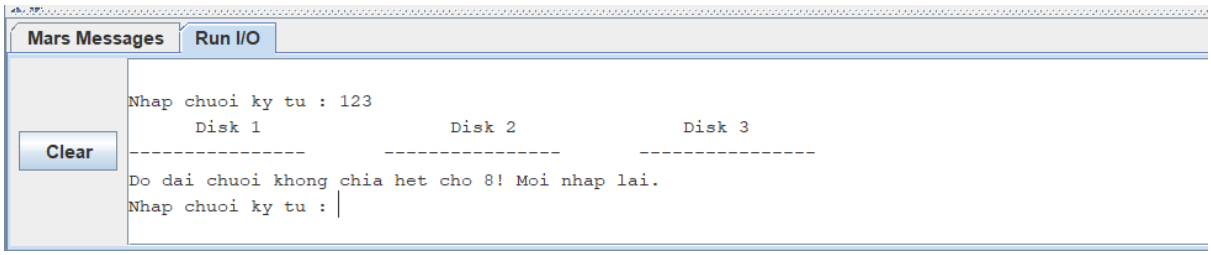
Ở phần 2,3 cách hoạt động của chương trình cũng giống tương tự phần 1.

Ở cuối phần 3, nếu $t3(độ dài chuỗi đã hết) thì chuyển sang exit1. Nếu chưa hết chuỗi thì quay lại phần 1.

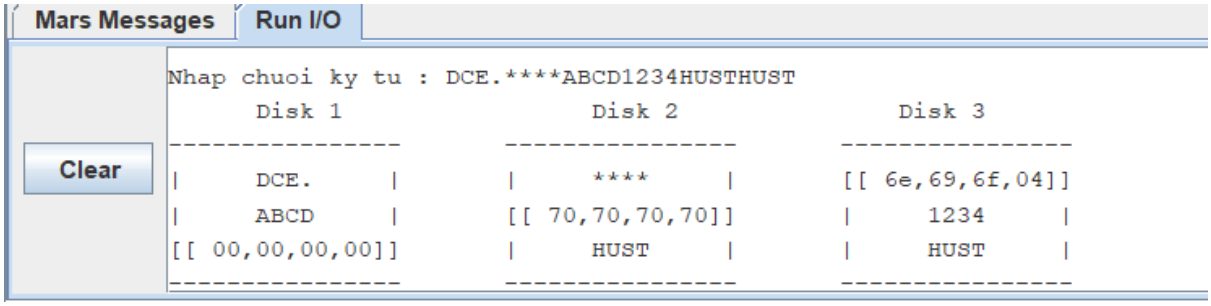
G. Kết quả chạy chương trình

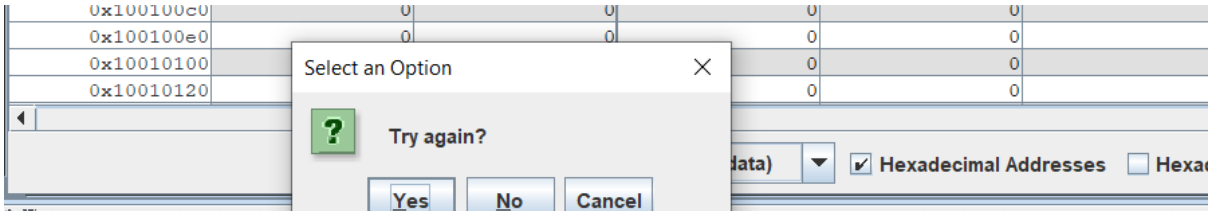
TH1: Khi nhập chuỗi kí tự rỗng hoặc xâu không có độ dài hợp lệ:

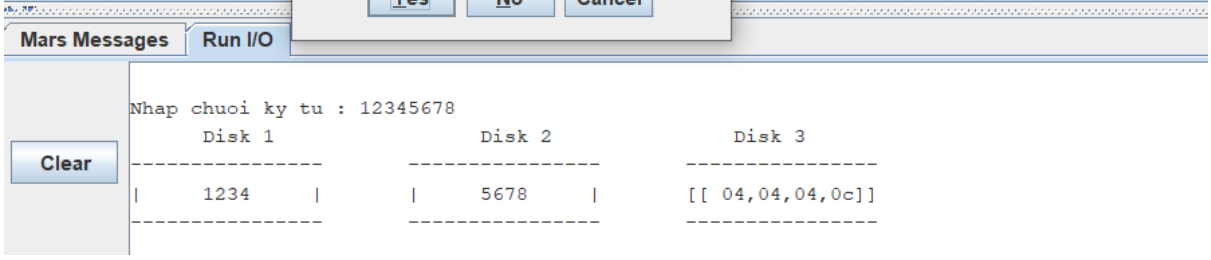


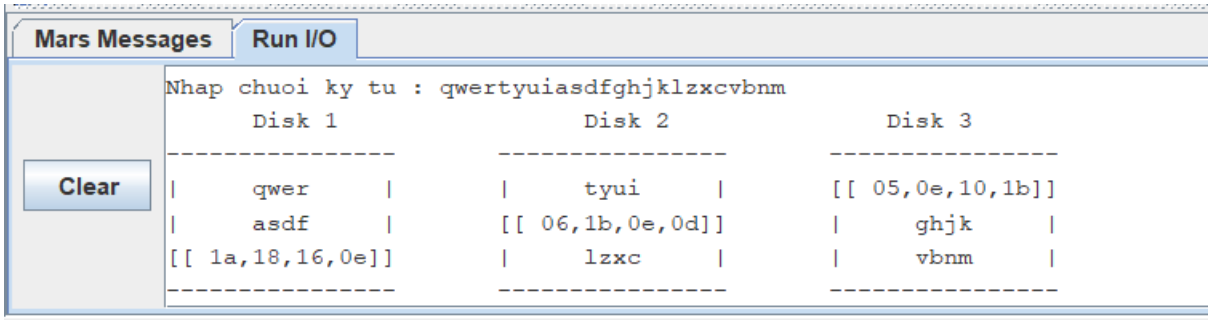


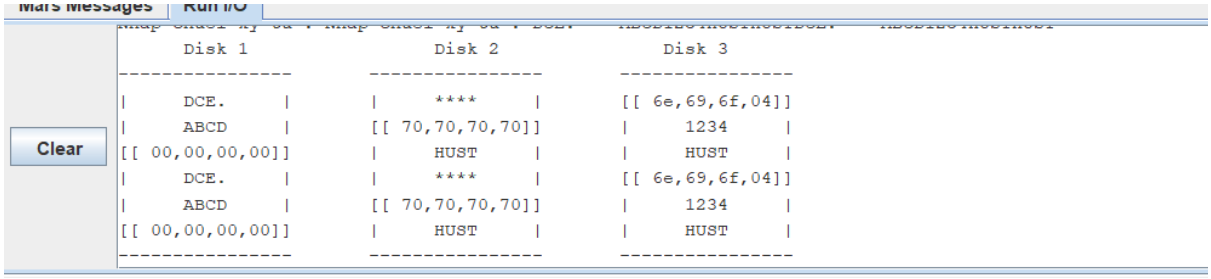
TH2: Khi nhập chuỗi kí tự hợp lệ:








Mars Messages
Run I/O
Clear
Nhap chuoi ky tu : qwertyuiasdfghjklzxcvbnm
Disk 1
Disk 2
Disk 3
qwer
asdf
[[1a,18,16,0e]]
tyui
[[06,1b,0e,0d]]
lzxc
[[05,0e,10,1b]]
ghjk
vbnm


Clear
Disk 1
Disk 2
Disk 3
DCE.
ABCD
[[00,00,00,00]]
****
[[70,70,70,70]]
HUST
[[6e,69,6f,04]]
1234
HUST