Bài thực hành số 11  
Lớp: 139365 – Học phần : Thực Hành Kiến Trúc Máy Tính

Họ và tên: Hạ Hồng Sơn MSSV: 20215130

**Assignment 1**: Xác định phím được nhập trong ma trận phím.

1. **Đoạn code:**

# CHUONG TRINH KIEM TRA PHIM DUOC NHAP BANG POLLING

#**------------------------------------------------------**

# 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

#**------------------------------------------------------**

# command row number of hexadecimal keyboard **(**bit 0 to 3**)**

# Eg. assign 0x1**,** to get key button 0**,** 1**,** 2**,** 3

# assign 0x2**,** to get key button 4**,** 5**,** 6**,** 7

# NOTE must reassign value for this address before reading**,**

# eventhough you only want to scan 1 row

.eqv IN\_ADRESS\_HEXA\_KEYBOARD 0xFFFF0012

# receive row **and** column of the key pressed**,** 0 if **not** key pressed

# Eg. equal 0x11**,** means that key button 0 pressed.

# Eg. equal 0x28**,** means that key button D pressed.

.eqv OUT\_ADRESS\_HEXA\_KEYBOARD 0xFFFF0014

**.text**

main**:** li $t1**,** IN\_ADRESS\_HEXA\_KEYBOARD

li $t2**,** OUT\_ADRESS\_HEXA\_KEYBOARD

set**:** li $t3**,** 0x01 # check row 4 with key 0, 1, 2, 3

li $t4**,** 0x10 # danh dau da duyet het cac hang tron ma tran

polling**:**

sb $t3**,** 0**(**$t1**)** # must reassign expected row

lb $a0**,** 0**(**$t2**)** # read scan code of key button

print**:** beqz $a0**,** nextrow # check co phim nao trong ham duoc nhap khong

li $v0**,** 34 # print integer **(**hexa**)**

**syscall**

sleep**:** li $a0**,** 500 # sleep 500ms

li $v0**,** 32

**syscall**

nextrow**:** sll $t3**,** $t3**,** 1

beq $t3**,** $t4**,** reset

j polling

reset**:** li $t3**,** 0x01

j polling # continue polling

1. **Giải thích:**

* IN\_ADRESS\_HEXA\_KEYBOARD 0xFFFF0012: nhận 1 byte chứa giá trị hàng của ma trận key matrix
* OUT\_ADRESS\_HEXA\_KEYBOARD 0xFFFF0014: chứa giá trị hexa của số vừa được bấm

Chương trình thực hiện in ra các phím tự đã được nhập theo ma trận phím.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Col 0x1 | Col 0x2 | Col 0x4 | Col 0x8 |
| row 0x1 | 0  0x11 | 1  0x21 | 2  0x41 | 3  0x81 |
| row 0x2 | 4  0x12 | 5  0x22 | 6  0x42 | 7  0x82 |
| row 0x4 | 8  0x14 | 9  0x22 | A  0x42 | B  0x82 |
| row 0x8 | C  0x18 | D  0x28 | E  0x48 | F  0x88 |

**Bảng giá trị của ma trận phím**

* Ma trận phím cho ta biết được giá trị của hàng i, cột j: với giá trị mỗi ô được định nghĩa 4 bits cao dành cho hang và 4 bits thấp dành cho cột (0xicoljrow).

***Note:*** Khi thực hiện chạy chương trình in ra các giá trị thuộc cột 0x8, ta thấy màn hình in ra 3 bytes đầu là 0xffffff là vì 0x8 ứng với nhị phân 4 bits có dấu là 10002 khi chuyển sang 32 bits máy sẽ tự động thêm các ký tự 1 và trước nên khi in ra màn hình ta được kết quả như vậy

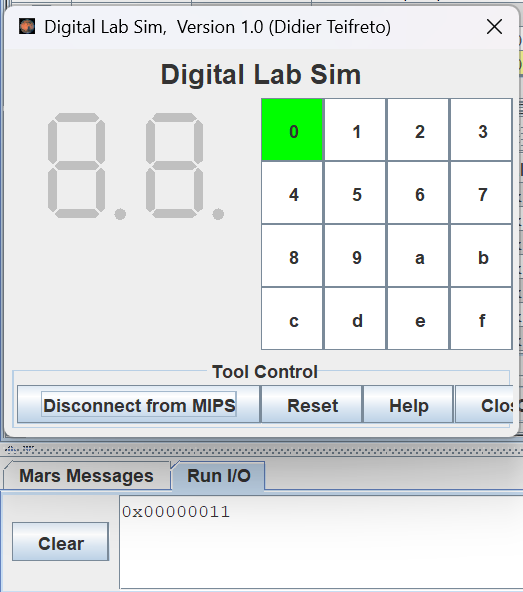
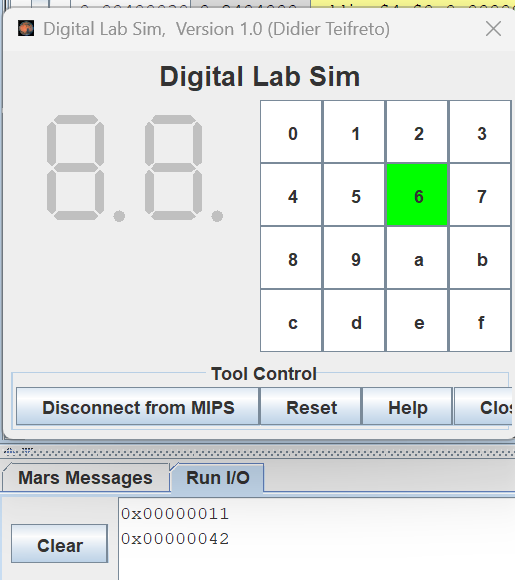
* Chương trình sẽ thực hiện in ra mã Hexa của ký tự được nhấn vào trong ma trận phím (Digital Lab Sim):
* $t1, $t2 lưu địa chỉ của: 0xFFFF0012 và 0xFFFF0014. Địa chỉ 0xFFFF0012 sẽ nhận 1 byte chứa giá trị hàng của ma trận key matrix trên, và 0xFFFF0014 sẽ chứa giá trị hexa của số vừa được bấm.
* $a0 lưu giá trị của phím được nhập
* sb $t3, 0($t1): cần được thực hiện trước khi đọc mã quét của nút bàn phím (scan code) để đảm bảo rằng chúng ta đang quét đúng hàng trên bàn phím. Sau đó đọc phím đã được nhập.
* Chương trình kiểm tra xem tại mỗi hang ký tự nhập vào có khác 0 hay không, nếu khác 0 🡺 in ra màn hình mã hexa
* Sau đó thực hiện sleep, tạm dừng thực thi trong vòng 500ms. Rồi cập nhập đến hàng tiếp theo và tiếp tục kiểm tra cho đến khi

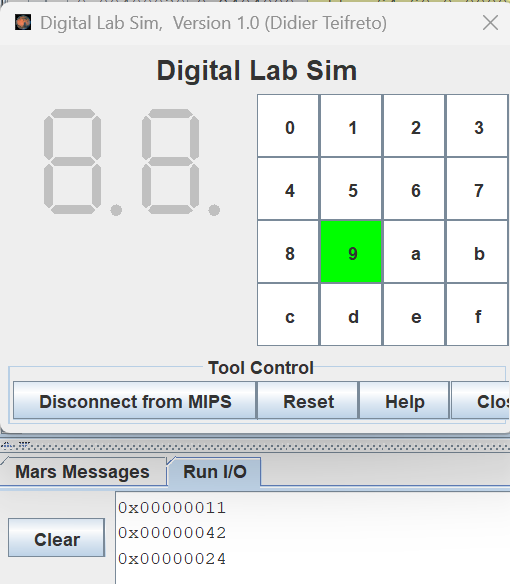
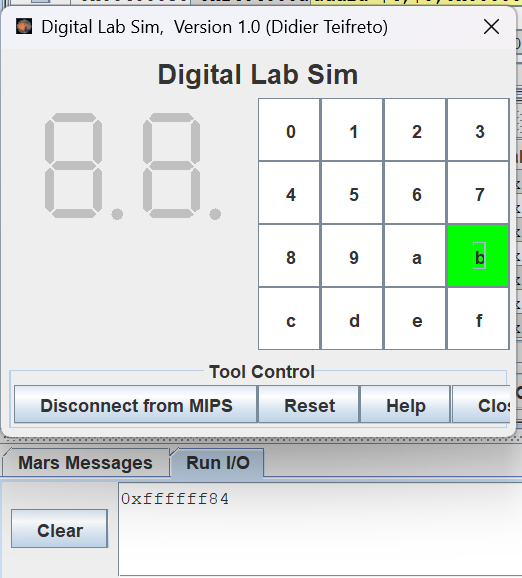
$t3 == $t4 thì cập nhập trở về hàng đầu tiên.

* Chức năng chính của polling: là liên tục kiểm tra xem có ký tự phím nào được nhập không thông qua các vòng lặp. Điều này dẫn đến số lần gọi vô cùng lớn khi CPU phải liên tục vào ra bộ nhớ.

1. **Kết quả:**

Khi nhập ký tự 1, 6, 9, B ta thu được kết quả ứng với mã hexa**:**

** **

**** 

🡺Kết quả hiển thị đúng với mong đợi.

**Assignment 2**: Exception khi nhập ký tự từ matrix key

1. **Đoạn code:**

.eqv IN\_ADRESS\_HEXA\_KEYBOARD 0xFFFF0012

.data

Message**:** .asciiz "Oh my god. Someone's presed a button.\n"

#**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

# MAIN Procedure

#**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

**.text**

main**:**

#**---------------------------------------------------------**

# Enable interrupts you expect

#**---------------------------------------------------------**

# Enable the interrupt of Keyboard matrix 4x4 of Digital Lab

Sim**:**

li $t1**,** IN\_ADRESS\_HEXA\_KEYBOARD

li $t3**,** 0x80 # bit 7 of **=** 1 to enable interrupt

sb $t3**,** 0**(**$t1**)**

#**---------------------------------------------------------**

# No**-**end **loop,** main program**,** to demo the effective of interrupt

#**---------------------------------------------------------**

**Loop:** **nop**

**nop**

**nop**

**nop**

b **Loop** # **Wait** for interrupt

end\_main**:** li $v0**,** 10

**syscall**

#**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~** **~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

# GENERAL INTERRUPT SERVED ROUTINE for **all** interrupts

#**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

.ktext 0x80000180

#**--------------------------------------------------------**

# Processing

#**--------------------------------------------------------**

IntSR**:** addi $v0**,** $zero**,** 4 # show message

la $a0**,** Message

**syscall**

#**--------------------------------------------------------**

# Evaluate the return address of main routine

# 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

return**:** eret # Return from exception

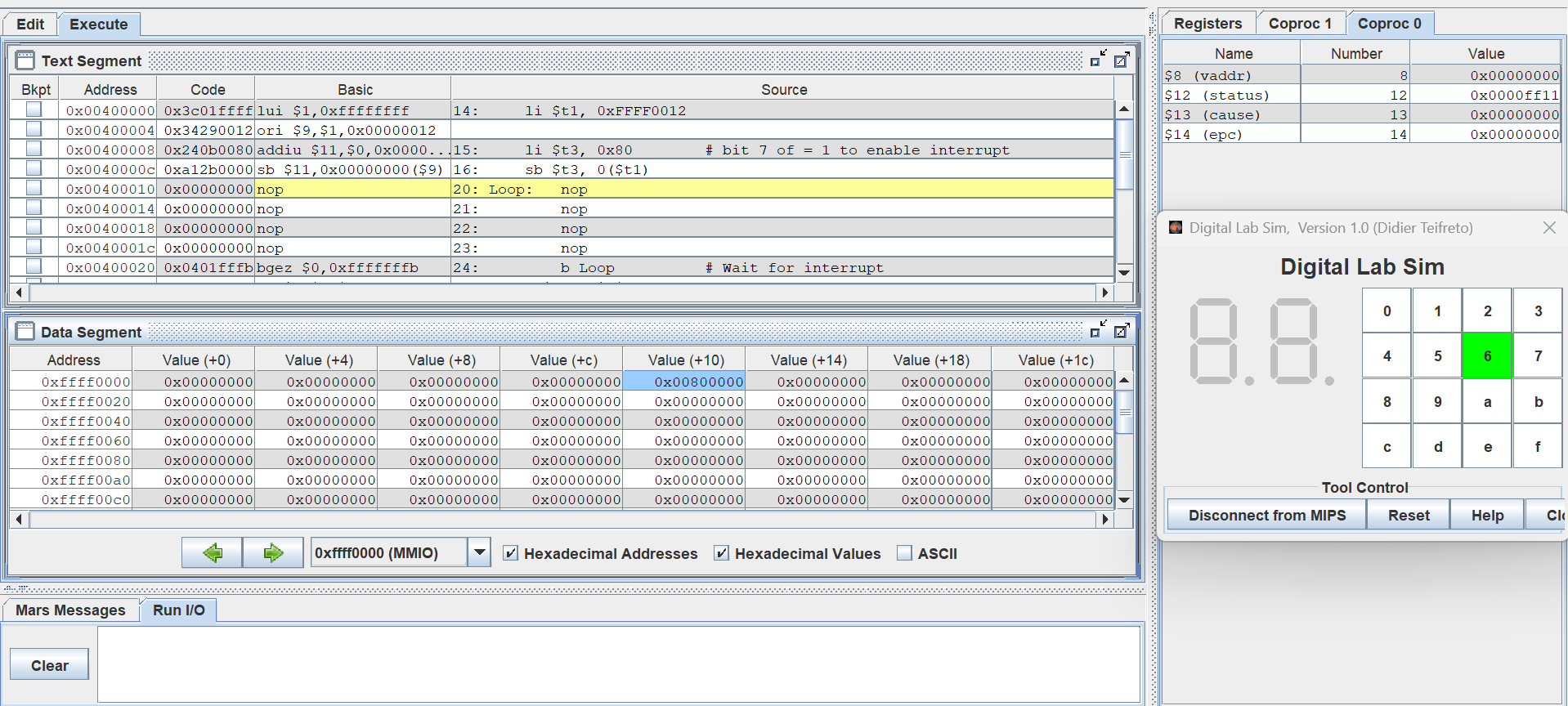
1. **Giải thích:**
   * Các thanh ghi được sử dụng trong bộ đồng sử lý C0:
     + $12 (status): Trong thanh ghi chứa bit thứ 1 được gọi là bit cho phép ngắt.
       - Khi bit này được bật lên (bit1 = 1) 🡪 ngắt (interrupt)
       - Khi bit này được tắt (bit1 = 0) 🡪 Không xảy ra hiện tượng gì.
     + $13 (cause):
       - Trường hợp thanh ghi $13 (cause) cho biết nguyên nhân làm tham chiếu địa chỉ bộ nhớ không hợp lệ, thanh ghi $8 (vaddr) sẽ chứa địa chỉ lỗi đó
     + $14 (epc) : chứa địa chỉ kế tiếp của chương trình chính, để quay trở về sau khi xử lý các đoạn mã Exception xong. Sử dụng lệnh eret đề trở về
   * Nguyên lý hoạt động:
     + sb $t3, 0($t1) lưu giá trị của thanh ghi $t3 (Chứa bit 7 1) vào địa chỉ của ma trận phím. Điều này có tác dụng kích hoạt ngắt từ bàn phím bằng cách gửi một tín hiệu hoặc trạng thái cho thiết bị bàn phím.
     + Set ngắt: Kích hoạt bít thứ 7 bằng 1
     + Khi phát hiện ngắt (người dùng nhập vào matrix key)
       - Tự chuyển xuống địa chỉ ngắt và $14 (epc) lưu địa chỉ của câu lệnh hiện tại.
     + Trong chương trình ngắt
       - Truy cập đến địa chỉ ngắt: .ktext 0x80000180
       - Tăng lên $14 cộng thêm 4 để thực hiện sau câu lênh ngắt trong chương trình chính
         * Để lấy được giá trị của $14 trong C0 ta thực hiện lênh : mfc0 $at, $14
         * Và lưu giá trị trả lại vào $14 của C0 ta thực hiện: mtc0 $at, $14
       - Trở về câu lệnh tiếp theo của chương trình chính với cú pháp eret

Cơ chế cài đặt ngắt:

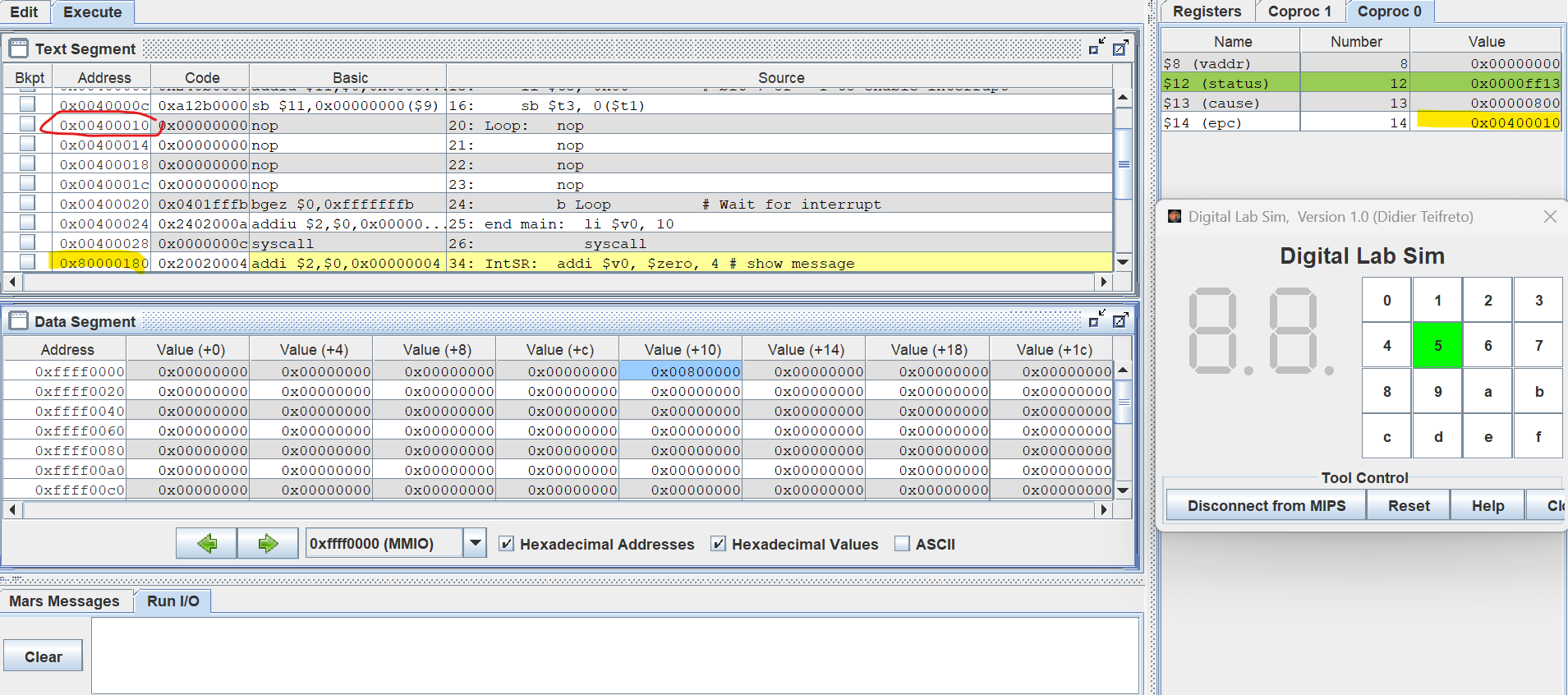
* Kích hoạt bít ngắt bằng 1
* Khai báo địa chỉ ngắt .ktext
* Đưa ra các câu lệnh cảnh báo,…
* Thực hiện quay về chương trình chính:
  + Lấy giá trị của $14 (epc) cập nhập +4 và lưu trở lại với mfc0 và mtc0
* Trở về hàm chính: eret

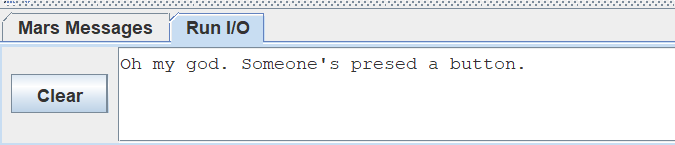
1. **Kết quả và thực hiện chương trình:**

Ban đầu tại Coproc 0:

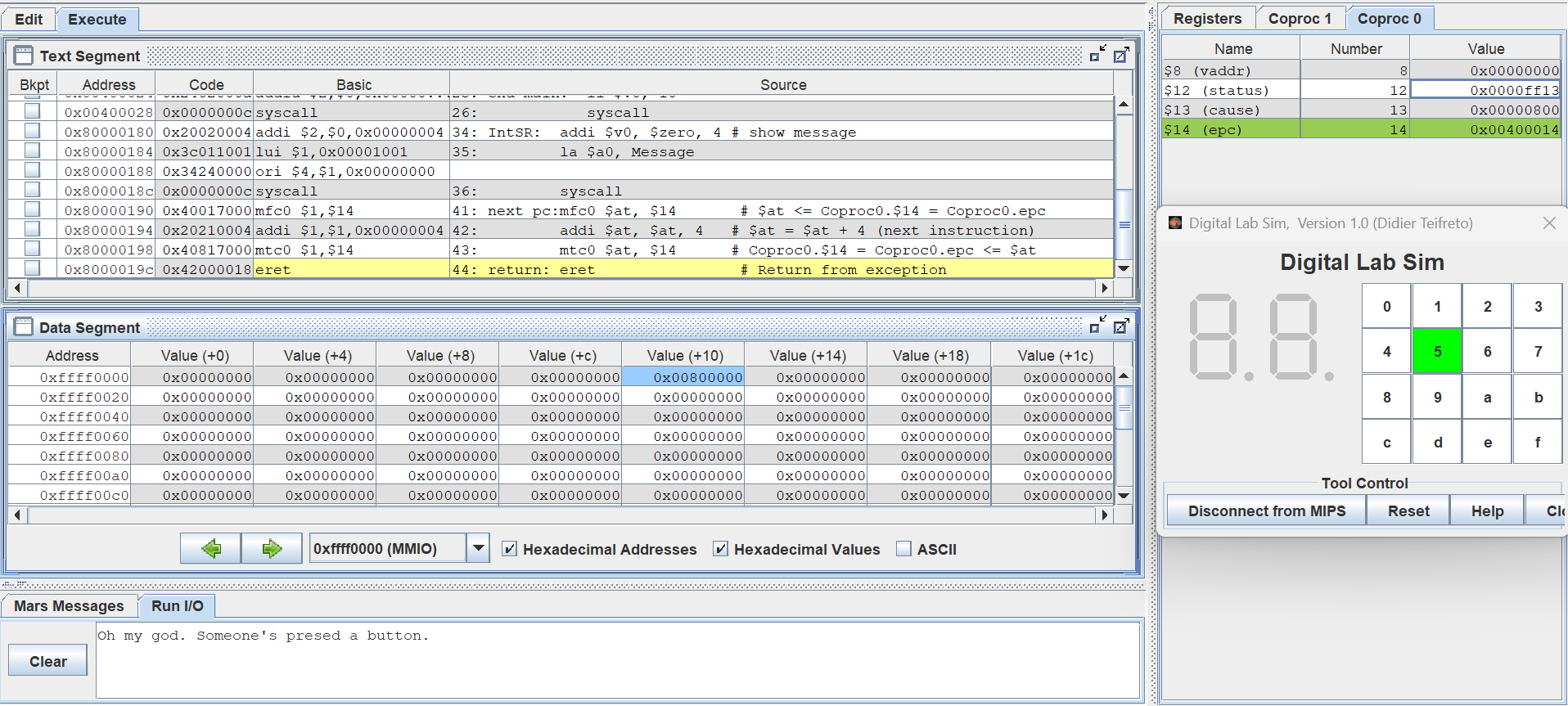


Khi bắt đầu thực hiện câu lệnh ngắt bằng kích thích nhập vào ma trận phím:

+) $14 (epc) thay đổi   
+) Chương trình nhảy đến địa chỉ ngắt

Và in ra màn hình kết quả:  


Sau đó cập nhập $14 = $14 + 4 và trở về hàm chính

****

**Assignment 3**: Ngắt bằng ma trận phím và đưa ra màn hinh phím được bấm

1. **Đoạn code:**

.eqv IN\_ADRESS\_HEXA\_KEYBOARD 0xFFFF0012

.eqv OUT\_ADRESS\_HEXA\_KEYBOARD 0xFFFF0014

.data

Message**:** .asciiz "Key scan code "

#**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

# MAIN Procedure

#**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

**.text**

main**:**

#**---------------------------------------------------------**

# Enable interrupts you expect

#**---------------------------------------------------------**

# Enable the interrupt of Keyboard matrix 4x4 of Digital Lab

Sim**:**

li $t1**,** IN\_ADRESS\_HEXA\_KEYBOARD

li $t3**,** 0x80 # bit 7 **=** 1 to enable

sb $t3**,** 0**(**$t1**)**

#**---------------------------------------------------------**

# **Loop** an print sequence numbers

#**---------------------------------------------------------**

**xor** $s0**,** $s0**,** $s0 # count **=** $s0 **=** 0

**Loop:** addi $s0**,** $s0**,** 1 # count **=** count **+** 1

prn\_seq**:**addi $v0**,**$zero**,**1

**add** $a0**,**$s0**,**$zero # print auto sequence number

**syscall**

prn\_eol**:**addi $v0**,**$zero**,**11

li $a0**,**'\n' # print endofline

**syscall**

sleep**:** addi $v0**,**$zero**,**32

li $a0**,**500 # sleep 500 ms

**syscall**

**nop** # WARNING**:** **nop** is mandatory here.

b **Loop** # **loop**

end\_main**:**

#**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

# GENERAL INTERRUPT SERVED ROUTINE for **all** interrupts

#**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

.ktext 0x80000180

#**-------------------------------------------------------**

# SAVE the current REG FILE to stack

#**-------------------------------------------------------**

IntSR**:** addi $sp**,**$sp**,**4 # Save $ra because we may change it later

sw $ra**,**0**(**$sp**)**

addi $sp**,**$sp**,**4 # Save $at because we may change it later

sw $at**,**0**(**$sp**)**

addi $sp**,**$sp**,**4 # Save $v0 because we may change it later

sw $v0**,**0**(**$sp**)**

addi $sp**,**$sp**,**4 # Save $a0**,** because we may change it later

sw $a0**,**0**(**$sp**)**

addi $sp**,**$sp**,**4 # Save $t1**,** because we may change it later

sw $t1**,**0**(**$sp**)**

addi $sp**,**$sp**,**4 # Save $t3**,** because we may change it later

sw $t3**,**0**(**$sp**)**

#**--------------------------------------------------------**

# Processing

#**--------------------------------------------------------**

prn\_msg**:**addi $v0**,** $zero**,** 4

la $a0**,** Message

**syscall**

li $t3**,** 0x81 # check row 1 **and** re**-**enable bit 7

get\_cod**:**li $t1**,** IN\_ADRESS\_HEXA\_KEYBOARD

sb $t3**,** 0**(**$t1**)** # must reassign expected row

li $t1**,** OUT\_ADRESS\_HEXA\_KEYBOARD

lb $a0**,** 0**(**$t1**)**

bnez $a0**,** prn\_cod

**and** $t4**,** $t3**,** 0x0F

**add** $t3**,** $t3**,** $t4 # next row

j get\_cod

prn\_cod**:**li $v0**,**34

**syscall**

li $v0**,**11

li $a0**,**'\n' # print endofline

**syscall**

#**--------------------------------------------------------**

# RESTORE the REG FILE from STACK

#**--------------------------------------------------------**

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 $ra**,** 0**(**$sp**)** # Restore the registers from stack

addi $sp**,**$sp**,-**4

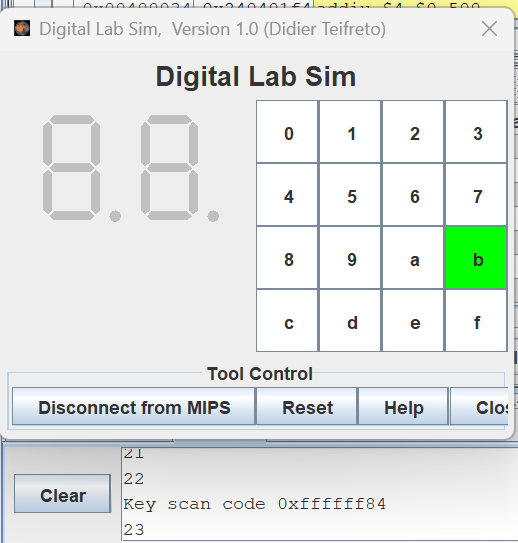
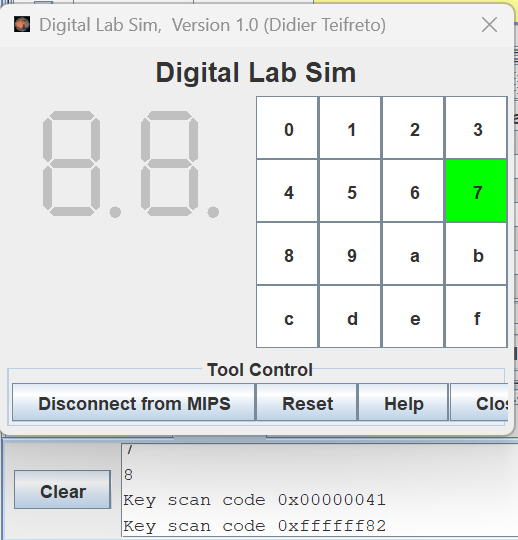
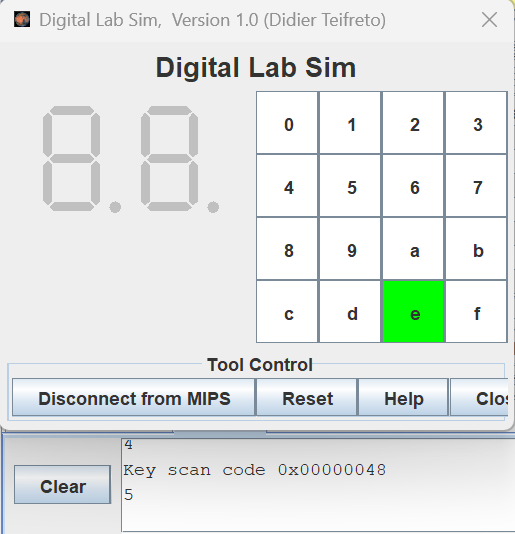
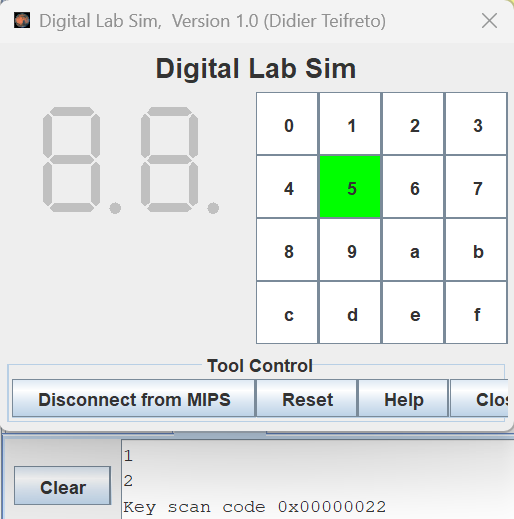
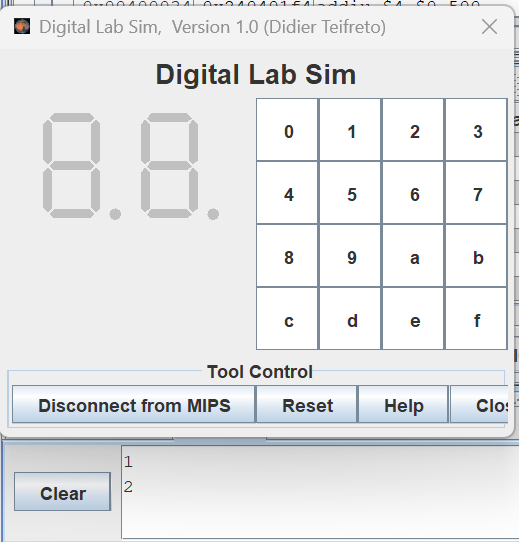
return**:** eret # Return from exception

1. **Giải thích:**

* Cơ chế hoạt động:
  + Tương tự như bài trên ta sẽ lưu địa chỉ lưu giá trị được bấm trên ma trận phím vào $t1. Kích hoạt bit ngắt bằng 1 và lưu vào địa chỉ đang được trỏ bởi $t1.
  + Khởi tạo biến tăng $s0, và bắt đầu vòng lặp. Vòng lặp thực hiện in ra chuỗi số tăng dần, và sleep khoảng 500ms sau khi in.
* Cơ chế ngắt
  + Trong vòng lặp đang in các chuỗi số tăng ra màn hình, nếu nhận được kích thích là bấm vào ma trận phím 🡪 kích hoạt ngắt chạy đến địa chỉ ngắt .ktext 0x80000180, và $14 (epc) trong Coproc sẽ có giá trị là địa chỉ câu lệnh hiện tại.
  + Trong chương trình ngắt ta thực hiện push từng giá trị của các thanh ghi vào stack để tránh mất dữ liệu ($ra: giá trị trả về hàm , $at, $v0: giá trị thực hiện syscall, $a0: value in ra màn hình, $t1: địa chỉ lưu giá trị được nhập từ ma trận phím, $t3: kích hoạt bit ngắt). Sau đó chương trình sẽ in ra màn hình “Key scan code ”.
  + Tiếp theo, scan ký tự được bấm trong ma trận phím, bắt đầu từ hàng đầu tiên (0x81) lưu trong thanh ghi $t3. Thanh ghi $t3 thực hiện đồng thời 2 chức năng: kích hoạt bit ngắt và trỏ đến hàng i của ma trận phím. Tiếp tục scan các hàng tiếp theo: xác định hàng i (0x8i) sau đó hàng tiếp theo là 0x8j (với j = i+i). Lấy giá trị i bằng phép AND $t3 với 0x0F đề lấy 4 bít cuối của $t3.
  + Chương trình sẽ quét cho đến khi gặp một hàng có ký tự được nhập rồi in ra màn hình.
  + Kết thúc, pop các giá trị trong stack trả về các thanh ghi ban đầu và tiếp tục câu lệnh trong hàm main

1. **Kết quả:**

Khi nhập vào 5, E, 2, 7, B kết quả thu được là:

****

* + - **Đúng với kết quả**

**Assignment 4**: Đưa ra nguyên nhân ngắt (Time counter hay Key matrix interrupt)

1. **Đoạn code:**

.eqv IN\_ADRESS\_HEXA\_KEYBOARD 0xFFFF0012

.eqv COUNTER 0xFFFF0013 # Time Counter

.eqv MASK\_CAUSE\_COUNTER 0x00000400 # Bit 10**:** Counter interrupt

.eqv MASK\_CAUSE\_KEYMATRIX 0x00000800 # Bit 11**:** Key matrix interrupt

.data

msg\_keypress**:** .asciiz "Someone has pressed a key!\n"

msg\_counter**:** .asciiz "Time inteval!\n"

#**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

# MAIN Procedure

#**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

**.text**

main**:**

#**---------------------------------------------------------**

# Enable interrupts you expect

#**---------------------------------------------------------**

# Enable the interrupt of Keyboard matrix 4x4 of Digital Lab

Sim**:**

li $t1**,** IN\_ADRESS\_HEXA\_KEYBOARD

li $t3**,** 0x80 # bit 7 **=** 1 to enable

sb $t3**,** 0**(**$t1**)**

# Enable the interrupt of TimeCounter of Digital Lab Sim

li $t1**,** COUNTER

sb $t1**,** 0**(**$t1**)**

#**---------------------------------------------------------**

# **Loop** an print sequence numbers

#**---------------------------------------------------------**

**Loop:** **nop**

**nop**

**nop**

sleep**:** addi $v0**,**$zero**,**32 # BUG**:** must sleep to **wait** for Time Counter

li $a0**,**200 # sleep 200 ms

**syscall**

**nop** # WARNING**:** **nop** is mandatory here.

b **Loop**

end\_main**:**

#**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

# GENERAL INTERRUPT SERVED ROUTINE for **all** interrupts

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

.ktext 0x80000180

IntSR: #--------------------------------------------------------

# Temporary disable interrupt

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

dis\_int:li $t1, COUNTER # BUG: must disable with Time Counter

sb $zero, 0($t1)

# no need to disable keyboard matrix interrupt

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

# Processing

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

get\_cause: mfc0 $t1, $13 # $t1 = Coproc0.cause

IsCount: li $t2, MASK\_CAUSE\_COUNTER# if Cause value confirm Counter..

and $at, $t1,$t2

beq $at,$t2, Counter\_Intr

IsKeyMa:li $t2, MASK\_CAUSE\_KEYMATRIX # if Cause value confirm Key..

and $at, $t1,$t2

beq $at,$t2, Keymatrix\_Intr

others: j end\_process # other cases

Keymatrix\_Intr: li $v0, 4 # Processing Key Matrix Interrupt

la $a0, msg\_keypress

syscall

j end\_process

Counter\_Intr: li $v0, 4 # Processing Counter Interrupt

la $a0, msg\_counter

syscall

j end\_process

end\_process:

mtc0 $zero, $13 # Must clear cause reg

en\_int: #--------------------------------------------------------

# Re-enable interrupt

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

li $t1, COUNTER

sb $t1, 0($t1)

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

# Evaluate the return address of main routine

# 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

return: eret # Return from exception

1. **Giải thích:**

* Nguyên lý hoạt động: Với đoạn code trên chương trình sẽ thực hiện xác định đây là ngắt do key matrix hay do Bộ đếm thời gian gây ra.
  + Đầu tiên ta khởi tạo 2 giá trị địa chỉ cho IN\_ADDRESS\_ HEXA\_KEYBOARD = 0xFFFF0012 là địa chỉ của ma trận phím và COUNTER = 0xFFFF0013 là địa chỉ của bộ đếm thời gian.
  + Sử dụng bit mask số 10 và 11 bằng 1: nếu ngắt bằng Time Counter thì $13 sẽ có giá trị là 0x400 hay bit 10 là 1, còn ngắt do ma trận phím sẽ có giá trị là 0x800 hay bit 11 là 1. Tạo ra hai mặt nạ bit: MASK\_CAUSE\_COUNTER và MASK\_CAUSE\_KEYMATRIX phục vụ xác định nguyên nhân ngắt
  + Trong hàm main xác định Bit cho phép ngắt
    - Ngắt của key matric: gán bit thứ 7 tại vị trí IN\_ADDRESS bằng 1

li $t1, IN\_ADRESS\_HEXA\_KEYBOARD

li $t3, 0x80 # bit 7 = 1 to enable

sb $t3, 0($t1)

* + - Ngắt của time interval: gán 0xFFFF0013 vào địa chỉ COUNTER

li $t1, COUNTER

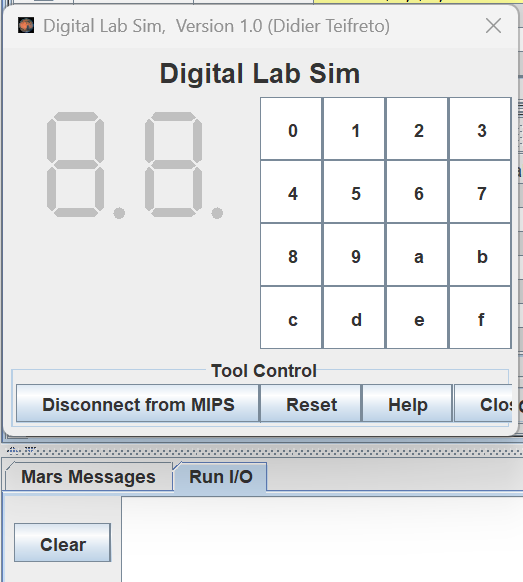
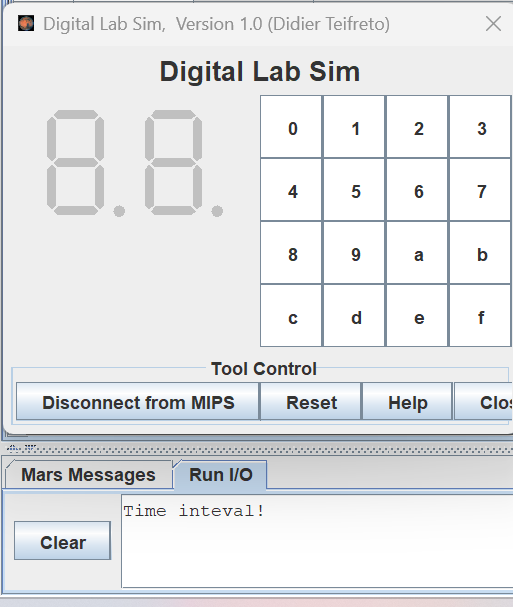
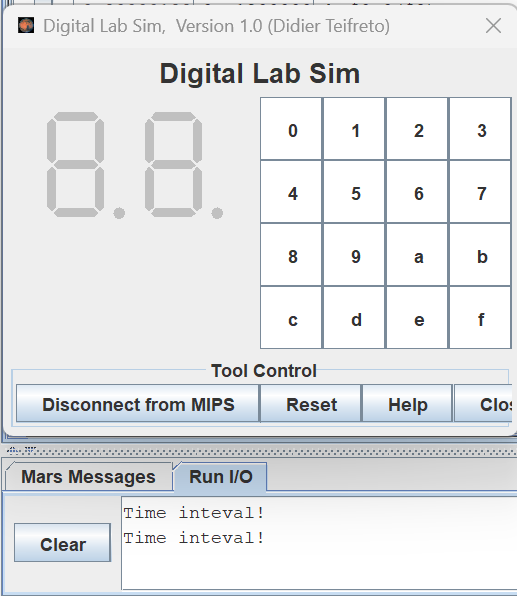
sb $t1, 0($t1)

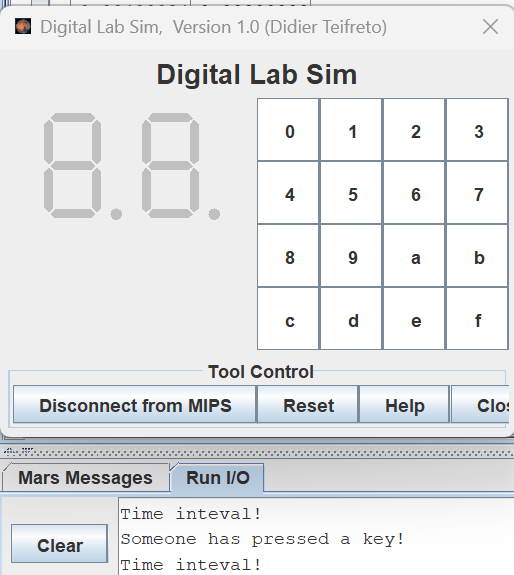
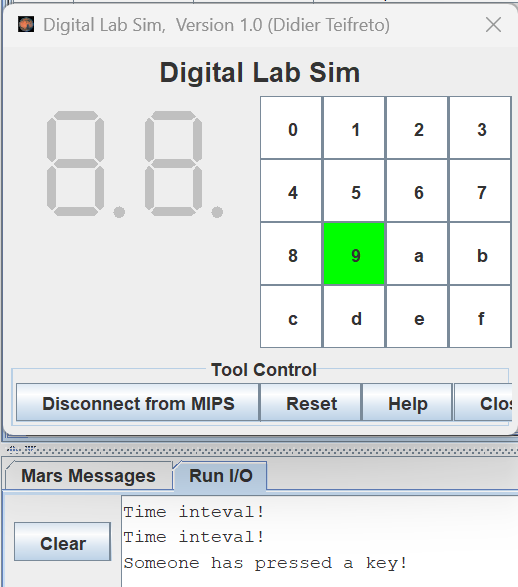
* + Sau đó ta sẽ có 1 vòng lặp vô hạn Loop thực hiện một vài lệnh nop và sleep. Khi hết thời gian đếm hoặc người dùng bâm một phím nào đó trong ma trân phím thì sẽ thực hiện ngắt và nhảy đến chương trình ngắt.
* Cơ chế kiểm tra ngắt:
  + Khi nhận được kích thích chương trình nhảy đến địa chỉ ngắt, .ktext 0x80000180.
  + Kiểm tra xem switch case là gì (2 nguồn ngắt)? Nạp mặt nạ sau đó kiểm tra xem là ngắt key hay time. Nếu bit 10 🡪 Key ,còn bit 11 🡪 counter sử dụng AND thanh nghi nguyên nhân ngắt $13 (mfc0) 🡪 nhảy đến sử lý tương ứng (in ra màn hình thông báo)
    - Lấy nguyên nhân gây ngắt tại $13 (Coproc0.cause) lưu giá trị vào $t1 đem đi AND với mặt nạ bit couter (MASK\_CAUSE\_COUNTER). Nếu kết quả thu được trùng với $t2 ->lỗi Time inteval xử lý in ra màn hình.
    - Nếu không bằng tiếp tục kiểm tra với mặt nạ bit keymatrix, AND MASK\_CAUSE\_KEYMATRIX (lưu tại $t2) với nguyên nhân ngắt, nếu kết quả trùng với $t2 🡪 lỗi bởi matrix key 🡪xử lý in ra màn hình. Còn nếu kết quả khác $t2 🡪 Rơi vào trường hợp ngắt khác.
  + Sau mỗi lần ngắt xóa đi ( xóa thành ghi nguyên nhân ngắt $13 bằng cách lưu giá trị 0 vào đó). Sau đó lưu lại $t1 vào địa chỉ ở $t1 để bật lại cho phép ngắt bằng thời gian.
  + Sau đó cập nhập giá trị thanh ghi $14 (epc) cộng thêm 4 để tiếp tục thực hiện câu lệnh tiếp theo trong chương trình chính với câu lệnh quay lại eret.

1. **Kết quả:**

Chạy chương trình:

1. Không bấm gì 🡪 sau 3 vòng lặp ngắt “Time interval”
2. Sau khi bấm phím 9 🡪 Ngắt “Key matrix”
3. Tiếp tục 🡪 ngắt “Time interval”

**** **** 



**Kết quả**

**Assignment 5**:

1. **Đoạn code:**
2. **Giải thích:**

Cơ chế hoạt động:

* Bàn phím không tự động tạo ra ngắt. Thay vì phải kiểm tra liên tục trạng thái của bàn phím để xem có phím nào được nhấn hay không (polling), ngắt mềm cho phép chương trình chờ đợi ngắt được kích hoạt khi có phím từ bàn phím.
* Bằng cách sử dụng ngắt mềm, công việc xử lý bàn phím và hiển thị được đưa vào một ngắt riêng biệt. Trái với polling khi việc đọc và ghi ra màn hình phải thực hiện một cách tuần tự, thì ngắt mềm cho phép vừa đọc và vừa ghi ra màn hình ký tự trước đó
* Đầu tiên ta khởi tạo:
  + KEY\_CODE (0xFFFF0004) giá trị ASCII ký tự được nhập từ bàn phím
  + KEY\_READY (0xFFFF0000) = 1 khi có ký tự được nhập và bằng 0 sau lệnh lw
    - Được gán lần lượt vào: $k0 và $k1
  + DISPLAY\_CODE (0xFFFF000C) giá trị ASCII của ký được được in ra màn hình
  + DISPLAY\_READY (0xFFFF0008) = 1 khi chuẩn bị in và sẽ = 0 sau lệnh sw
    - Được gán lần lượt vào: $s0 và $s1
* Tạo ra vòng lặp để đọc ký tự vào, nếu $t1 (value của KEY\_READY) == 0 tức chưa có ký tự được nhập thì ta sẽ tiếp tục lặp cho đến khi $t1 != 0 (khi nhập ký tự từ bàn phím tại địa chỉ KEY\_CODE sẽ lưu giá trị vừa được nhập, và tại KEY\_READY sẽ bằng 1). Sau khi xác nhận là có ký tự được nhập chương trình sẽ nhảy đến nhiệm vụ ngắt mềm bằng cách kiểm tra teqi $t1, 1 và nhảy đến địa chỉ ngắt .ktext 0x80000180

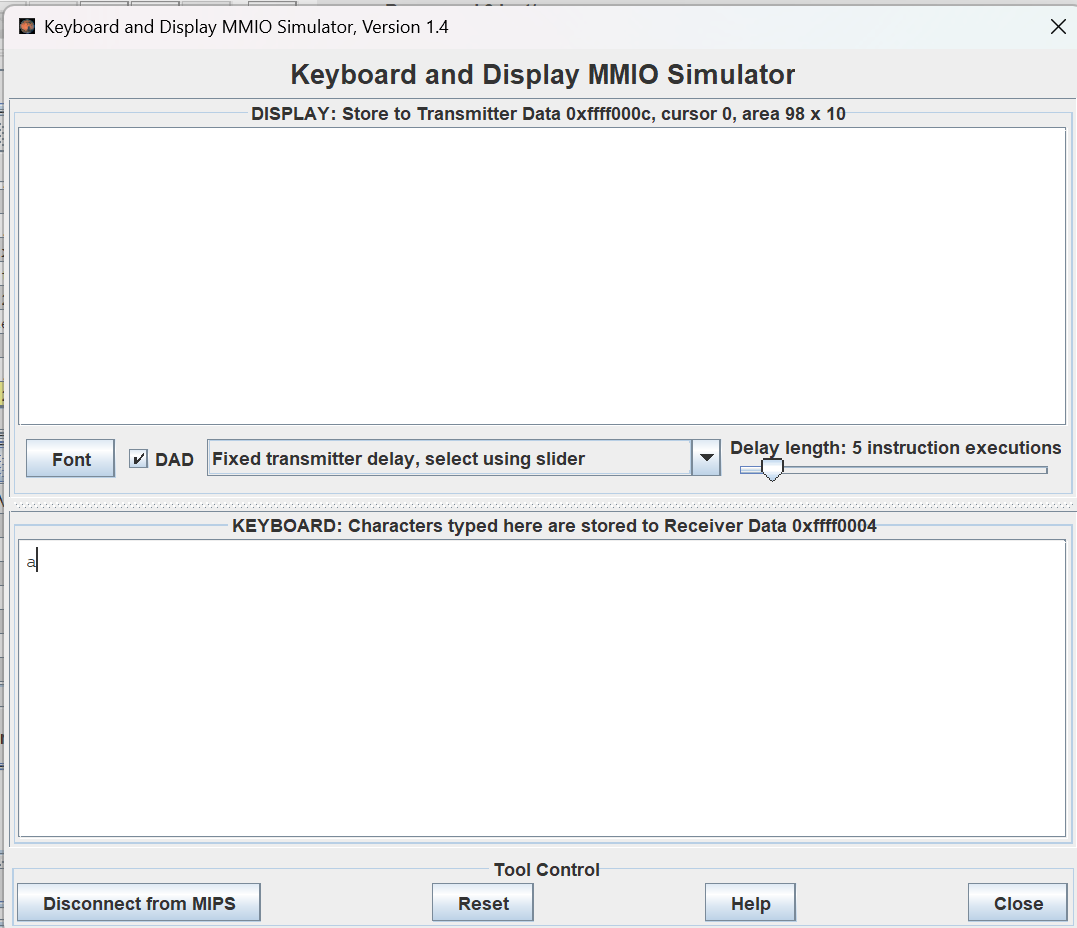
Cơ chế ngắt:

Cơ chế kích hoạt ngăt: khi có một ký tự được nhập vào từ bàn phím và KEY\_READY == 1

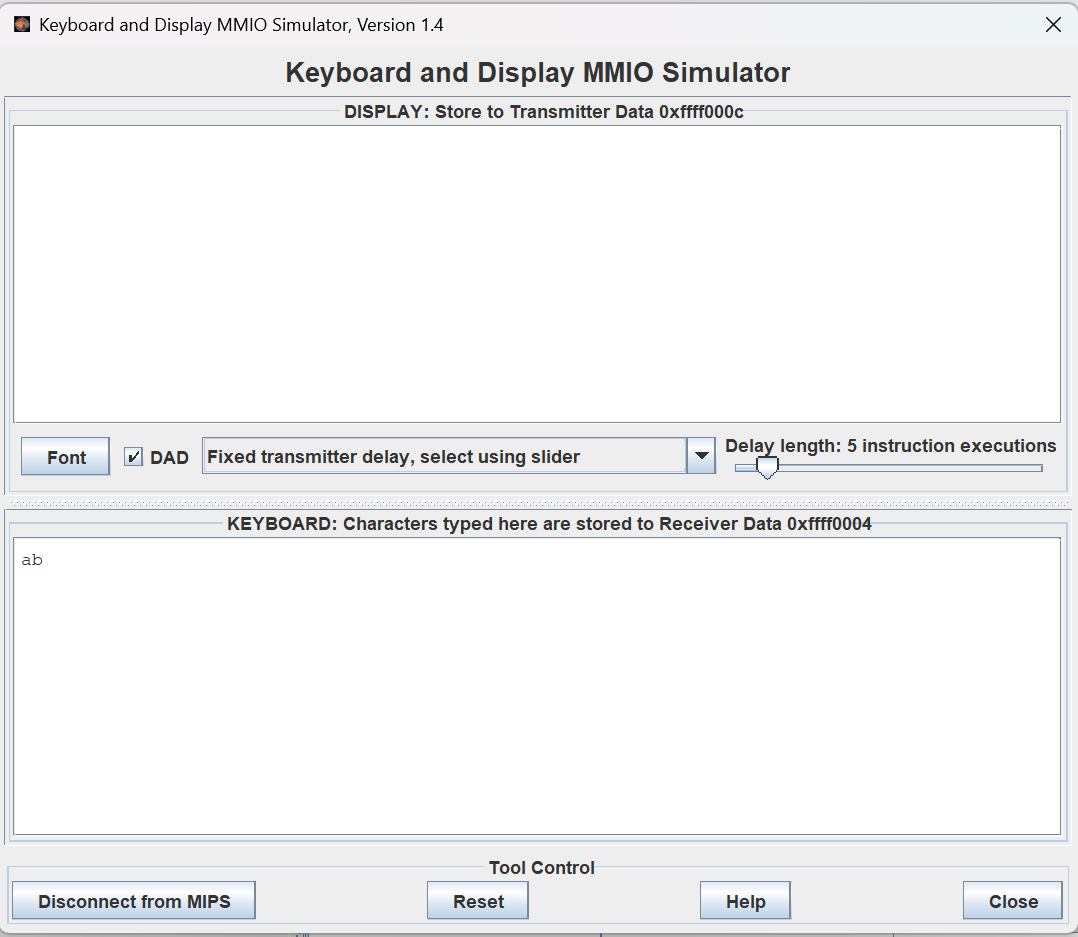
* Kiểm tra nguyên nhân ngắt từ bộ xử lý Coproc0($13) lưu vào $t1, và mặt nạ bit ngắt bàn phím vào $t2 🡺 Rồi sử dụng phép AND, nếu kết quả trả về trùng với $t2 tức là ngắt do bản phím.
* Sau đó ta đọc ký được đã được ghi vào từ bàn phìm và kiểm tra xem DISPLAY\_READY có sẵn sang không (nếu value tại DISPLAY\_READY == 1 tức là sẵn sang in ra màn hình, còn nếu bằng 0 thì sẽ tiếp tục polling bằng vòng lặp).
* Khi DISPLAY\_READY == 1, ta sẽ mã hóa ký tự bằng cách + mã ASCII của nó lên 1 và in kết quả ra màn hình.
* Kết thúc ngắt, ta sẽ cập nhập giá trị $14 (epc) cộng thêm 4 (địa chỉ lệnh của câu lệnh tiếp theo) rồi dùng lệnh eret để trở về chương trình chính.

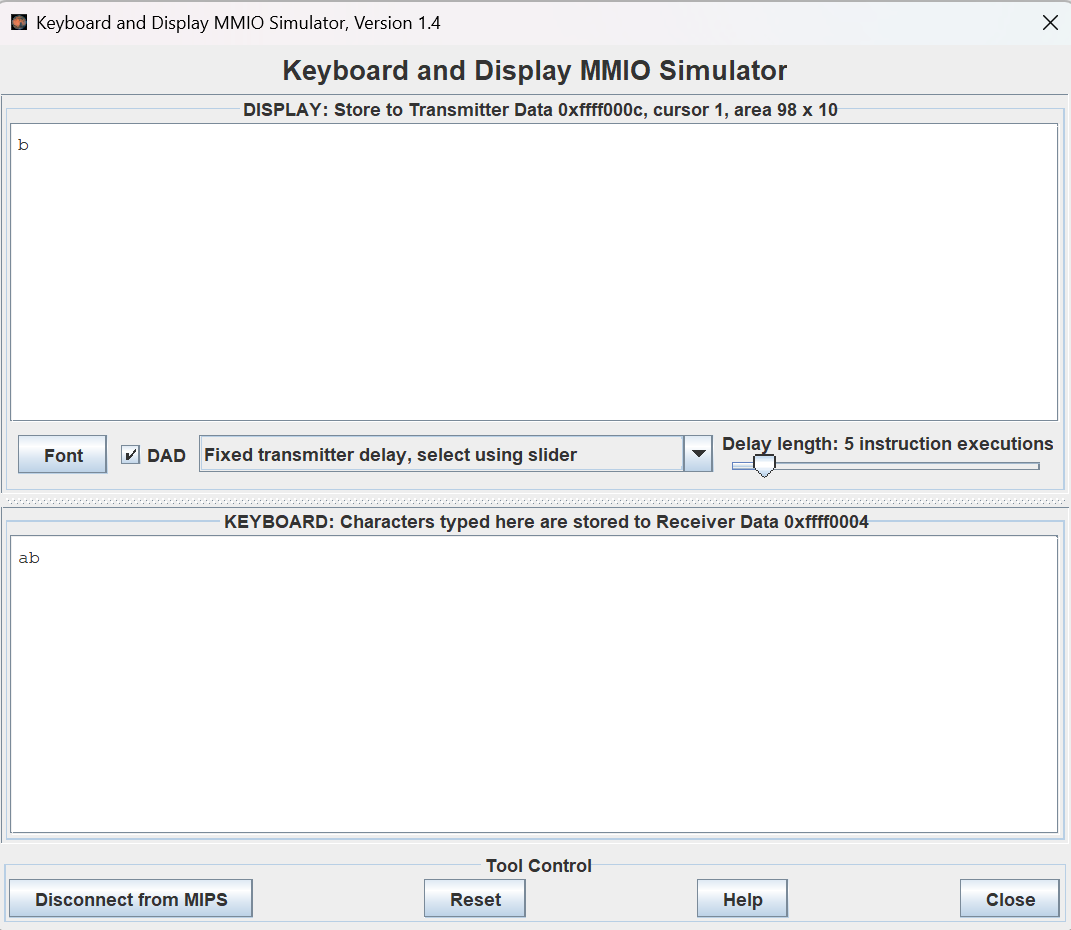
1. **Kết quả:**

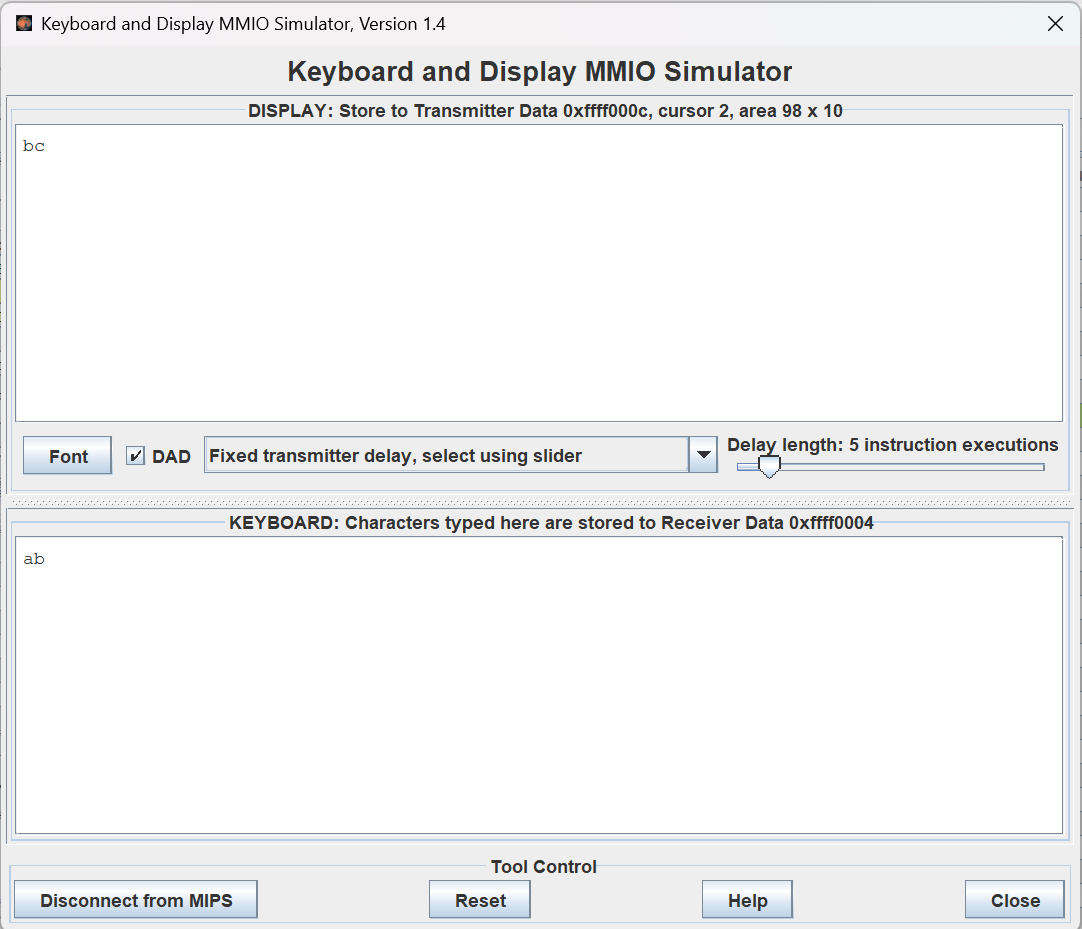
Ban đầu ta nhập ký tự ‘a’



Trong lúc chương trình đang xử lý in ta tiếp tục nhập ký tự ‘b’



Sau đó màn hinh hiện lên lần lượt các ký tự ‘b’, ‘c’  




**🡺 Kết quả đúng**