Lab 9 (Part 2)

Interrupts & IO programming

Đinh Ngọc Khánh Huyền – 20225726

Assigment 1

Code

```
.eqv IN_ADRESS_HEXA_KEYBOARD 0xFFFF0012
.eqv OUT_ADRESS_HEXA_KEYBOARD 0xFFFF0014
.data
Message: .asciiz "The key with a code of "
Message1: .asciiz " was pressed"
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# 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
```

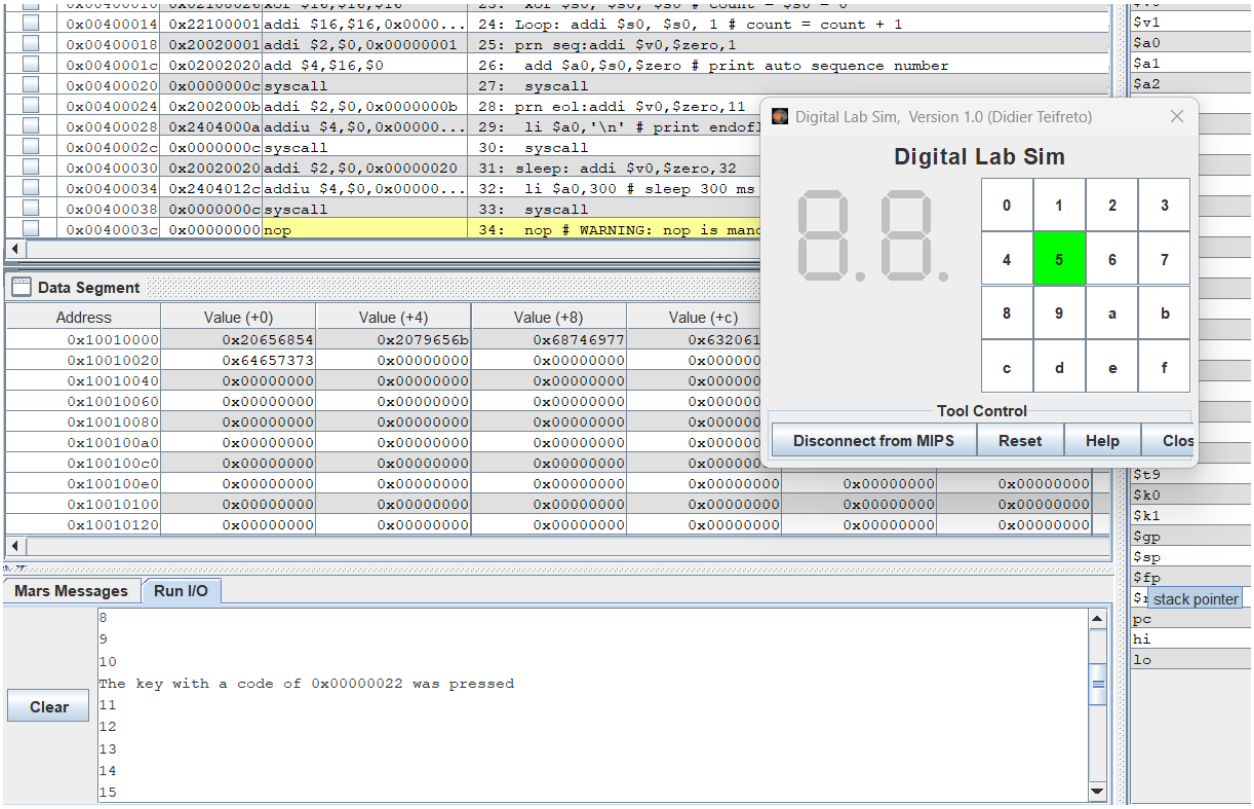
```
Loop: addi $s0, $s0, 1 # count = count + 1
prn_seq:addi $v0,$zero,1
 add $a0,$s0,$zero # print auto sequence number
 syscall
prn_eol:addi $v0,$zero,11
 li $a0,'\n' # print endofline
 syscall
sleep: addi $v0,$zero,32
 li $a0,300 # sleep 300 ms
 syscall
 nop # WARNING: nop is mandatory here.
 b Loop # 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 $ra because we may change it later
 sw $at,0($sp)
 addi $sp,$sp,4 # Save $ra 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

get_cod:
 li $t1, IN_ADRESS_HEXA_KEYBOARD
 li $t2, OUT_ADRESS_HEXA_KEYBOARD
start_interrupt_1:
        li $t3, 0x81 # check row 1 with key 0, 1, 2, 4
        sb $t3, 0($t1) # must reassign expected row
        jal interrupt

start_interrupt_2:
        li $t3, 0x82 # check row 2 with key 4, 5, 6, 7
        sb $t3, 0($t1) # must reassign expected row
        jal interrupt

start_interrupt_3:
        li $t3, 0x84 # check row 3 with key 8, 9, A, B
```

```
        sb $t3, 0($t1) # must reassign expected row
        jal interrupt

start_interrupt_4:
        li $t3, 0x88 # check row 4 with key C, D, E, F
        sb $t3, 0($t1) # must reassign expected row
        jal interrupt

check_after_interrupt_4:
        beq $a0, 0x0, prn_cod
        j next_pc

interrupt:
        lb $a0, 0($t2) # read scan code of key button
        bne $a0, 0x0, prn_cod
        jr $ra
prn_cod:li $v0,34
 syscall
prn_msg1:addi $v0, $zero, 4
 la $a0, Message1
 syscall

 li $v0,11
 li $a0,'\n' # print endofline
 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
 #---------------------------------------------------------
 # 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
```

Result



Khi ấn phím 5 => key _code = 0x22

Nên in ra “The key with a code of 0x00000022 was pressed”

Assigment 2

Code

```
# Assignment 2
.eqv IN_ADRESS_HEXA_KEYBOARD 0xFFFF0012

.eqv OUT_ADRESS_HEXA_KEYBOARD 0xFFFF0014

.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 "The key with a code of "

	msg_keypress1: .asciiz " was pressed\n"

	msg_counter: .asciiz "The counter has reached its maximum value for "

	msg_counter_end1: .asciiz " times\n"



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

# MAIN Procedure

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

.text

li $s7, 1
```

```
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 300 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_caus:

        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 $t1, IN_ADRESS_HEXA_KEYBOARD
    li $t2, OUT_ADRESS_HEXA_KEYBOARD

inter_1:
    li $t3, 0x81 # check row 1 with key 0, 1, 2, 4
    sb $t3, 0($t1) # must reassign expected row
    jal inter

inter_2:
    li $t3, 0x82 # check row 2 with key 4, 5, 6, 7
    sb $t3, 0($t1) # must reassign expected row
    jal inter
```

```
inter_3:

        li $t3, 0x84 # check row 3 with key 8, 9, A, B

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

        jal inter



inter_4:

        li $t3, 0x88 # check row 4 with key C, D, E, F

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

        jal inter



after_inter_4:

        beq $a0, 0x0, prn_cod

        j next_pc
```

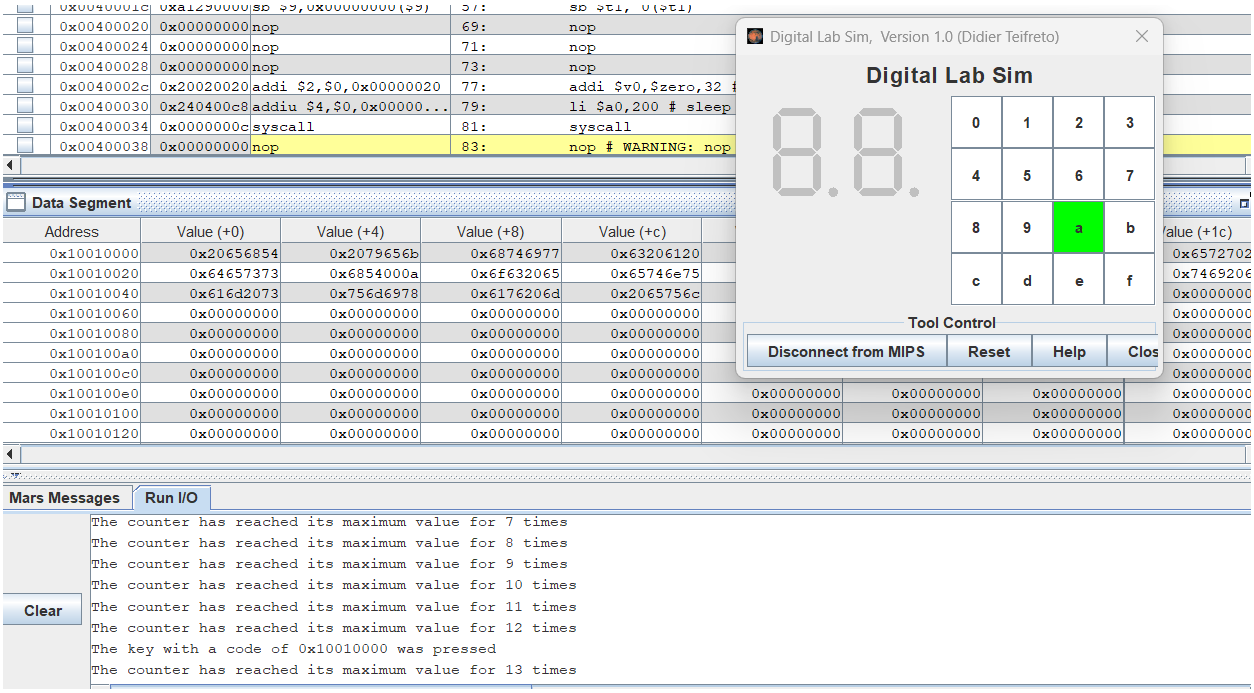
```
inter:

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

        bne $a0, 0x0, prn_cod

        jr $ra



prn_cod:

        li $v0, 4

        la $a0, msg_keypress

        syscall


        li $v0,34

        syscall



        li $v0, 4
```

```
        la $a0, msg_keypress1
        syscall


j end_process


Counter_Intr:
        li $v0, 4 # Processing Counter Interrupt
        la $a0, msg_counter
        syscall


        li $v0,1
        add $a0, $s7, $zero
        syscall
```

```
        add $s7, $s7, 1



        li $v0, 4 # Processing Counter Interrupt

        la $a0, msg_counter_end1

        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
```

Result



Trường hợp nhập vào phím a = 0x77

⇨ The key with a code of 0x1001000 was pressed

Explain

Khi đang thực hiện vòng lặp mà có tín hiệu nhấn từ ma trận Lab Sim hoặc Khoảng thời gian lập tới giới hạn thì Chương trình sẽ thực hiện ngắt.

Đầu tiên chương trình thực hiện so sánh và tìm ra nguyên nhân ngắt, nếu là vượt quá thời gian lặp thì sẽ in ra message còn nếu là tín hiệu nhấn từ Lab Sim thì sẽ in ra ký tự được nhấn (hệ cơ số 16)

Assigment 3

Code

```
# Assignment 3
.eqv KEY_CODE 0xFFFF0004 # ASCII code from keyboard, 1 byte
.eqv KEY_READY 0xFFFF0000 # =1 if has a new keycode ?
 # Auto clear after lw
.eqv DISPLAY_CODE 0xFFFF000C # ASCII code to show, 1 byte
```

```
.eqv DISPLAY_READY 0xFFFF0008 # =1 if the display has already to do
 # Auto clear after sw
.eqv MASK_CAUSE_KEYBOARD 0x0000034 # Keyboard Cause
.data
	message: .asciiz "Typing process ends"
.text
	li $k0, KEY_CODE
	li $k1, KEY_READY

	li $s0, DISPLAY_CODE
	li $s1, DISPLAY_READY
loop:
	nop
WaitForKey:
	lw $t1, 0($k1) # $t1 = [$k1] = KEY_READY
	beq $t1, $zero, WaitForKey # if $t1 == 0 then Polling
MakeIntR:
	teqi $t1, 1 # if $t1 = 1 then raise an Interrupt
	j loop
#-----------------------------------------------------------
# Interrupt subroutine
#-----------------------------------------------------------
.ktext 0x80000180
get_caus:
	mfc0 $t1, $13 # $t1 = Coproc0.cause
IsCount:
	li $t2, MASK_CAUSE_KEYBOARD# if Cause value confirm Keyboard..
```

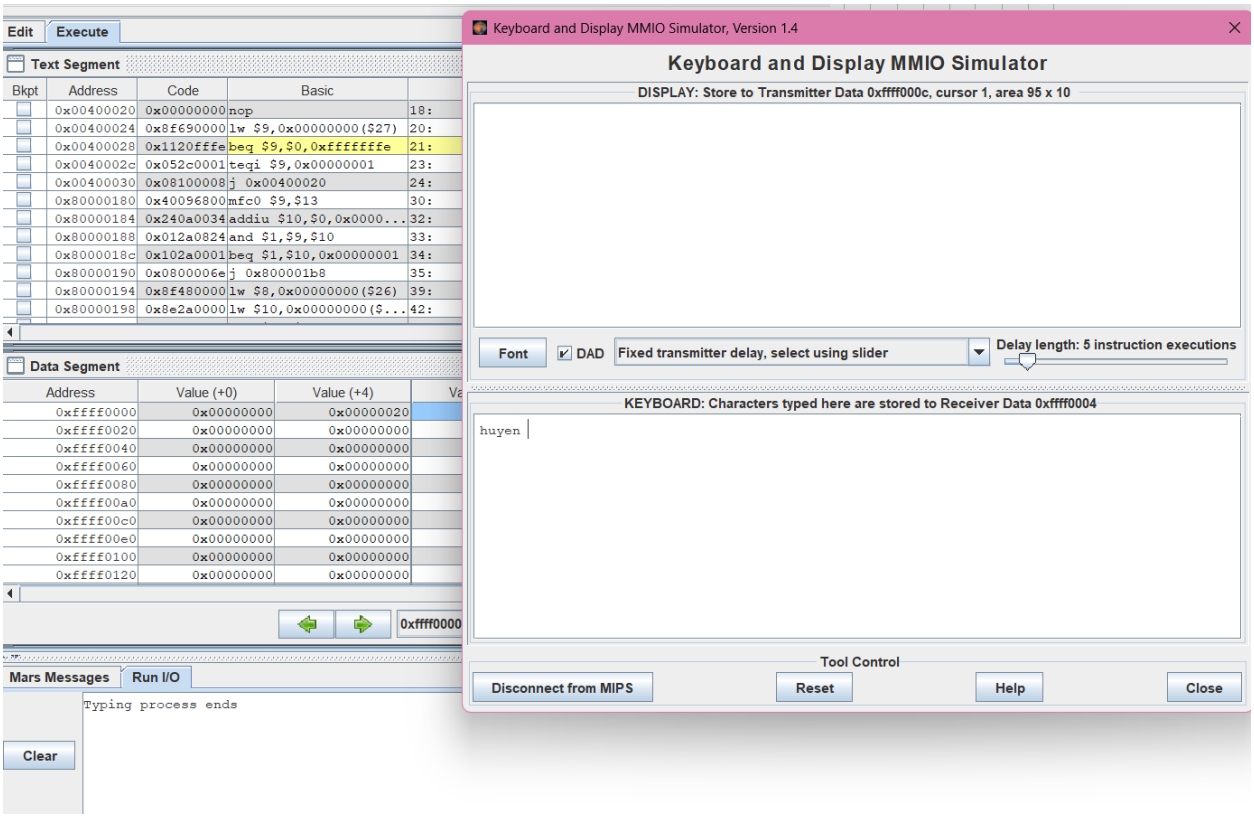
```
        and $at, $t1,$t2
        beq $at,$t2, Counter_Keyboard
        j end_process

Counter_Keyboard:
ReadKey:
        lw $t0, 0($k0) # $t0 = [$k0] = KEY_CODE

WaitForDis:
        lw $t2, 0($s1) # $t2 = [$s1] = DISPLAY_READY
        beq $t2, $zero, WaitForDis # if $t2 == 0 then Polling


ShowKey:
        sw $t0, 0($s0) # show key
        nop

prn_msg:
addi $v0, $zero, 4
 la $a0, message
 syscall

end_process:

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

Result



Quizzes

- What are the differences between interrupts, exceptions, and traps?

Ngắt	Ngoại lệ
Ngắt phần cứng	Ngắt phần mềm
Sự xuất hiện của các ngắt phần cứng thường vô hiệu hóa các ngắt phần cứng khác.	Không như vậy.

Là các yêu cầu dịch vụ bên ngoài bất đồng bộ (như bàn phím hoặc máy in cần phục vụ).	Là các yêu cầu dịch vụ bên trong đồng bộ dựa trên các sự kiện bất thường (lệnh không hợp lệ, địa chỉ không hợp lệ, tràn số, v.v.).
Vì là bất đồng bộ, ngắt có thể xảy ra ở bất kỳ đâu trong chương trình.	Vì là đồng bộ, ngoại lệ xảy ra khi có sự kiện bất thường trong chương trình của bạn như chia cho không hoặc truy cập vùng nhớ bất hợp pháp.
Đây là các sự kiện bình thường và không nên can thiệp vào việc chạy bình thường của máy tính.	Đây là các sự kiện bất thường và thường dẫn đến việc kết thúc một chương trình.

- Bẫy là một loại ngoại lệ đặc biệt, được tạo ra bởi một lệnh phần mềm cụ thể trong chương trình. Khi một bẫy được kích hoạt, nó sẽ chuyển quyền điều khiển sang chương trình xử lý bẫy để thực hiện các hoạt động cần thiết. Sau khi xử lý xong, chương trình xử lý bẫy sẽ trả lại quyền điều khiển cho chương trình gốc và chương trình gốc tiếp tục thực thi từ điểm tiếp theo sau lệnh bẫy.
- Bẫy khác với ngắt ở chỗ ngắt là một sự kiện bất đồng bộ được kích hoạt bởi phần cứng hoặc sự kiện bên ngoài, trong khi bẫy là một lệnh phần mềm được đặt trong chương trình để kích hoạt một sự kiện đồng bộ.

- In the sample code of Assignment 5, why is MASK_CAUSE_KEYBOARD set to 0x0000034?

Trong đoạn mã nguồn mẫu của Bài tập 5, giá trị của MASK_CAUSE_KEYBOARD được đặt là 0x0000034 để lọc ra nguyên nhân gây ra ngắt (interrupt) là do phím bàn phím.

Thanh ghi Cause (thanh ghi số 13 trong CPU) được sử dụng để lưu trữ mã nguyên nhân gây ra ngắt. Các bit trong thanh ghi Cause có ý nghĩa khác nhau.

Trong trường hợp này, giá trị 0x0000034 được sử dụng để mask bit tương ứng với nguyên nhân gây ra ngắt do phím bàn phím. Khi xảy ra ngắt, mã nguyên nhân sẽ được lưu trữ trong thanh ghi Cause, và giá trị 0x0000034 sẽ được sử dụng để kiểm tra xem nguyên nhân có phải là do phím bàn phím hay không.

```
get_caus:
    mfc0 $t1, $13 # $t1 = Coproc0.cause
IsCount:
```

```
    li $t2, MASK_CAUSE_KEYBOARD # $t2 = 0x0000034
    and $at, $t1, $t2 # $at = $t1 & $t2
    beq $at, $t2, Counter_Keyboard # Nếu $at == $t2 (nghĩa là nguyên nhân là do phím bàn phím) thì nhảy đến Counter_Keyboard
```

Đoạn mã trên thực hiện phép AND giữa giá trị trong thanh ghi Cause ($t1) và giá trị MASK_CAUSE_KEYBOARD ($t2). Nếu kết quả của phép AND này bằng với MASK_CAUSE_KEYBOARD, tức là nguyên nhân gây ra ngắt là do phím bàn phím, và chương trình sẽ nhảy đến nhãn Counter_Keyboard để xử lý ngắt.