**REPORT LAB 11**

**ASSEMBLY LANGUAGE AND COMPUTER ARCHITECTURE LAB**

**STUDENT NAME: PHI QUOC HUNG**

**STUDENT ID:20184270**

**Assignment 1:**

- Source Code:

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

# 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. assign0x1, 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

li $t3, 0x8 # check row 4 with key C, D, E, F

li $t4, 0x4

li $t5, 0x2

li $t6, 0x1

polling:

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

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

bne $a0,$zero,print

sb $t4, 0($t1) # must reassign expected row

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

bne $a0,$zero,print

sb $t5, 0($t1) # must reassign expected row

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

bne $a0,$zero,print

sb $t6, 0($t1) # must reassign expected row

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

bne $a0,$zero,print

print:

li $v0, 34 # print integer (hexa)

syscall

sleep:

li $a0, 2000 # sleep 100ms

li $v0, 32

syscall

back\_to\_polling:

j polling # continue polling

- The result observed:

Graphical user interface, application

Description automatically generated

- Explanation:

We can only check 1 row at a time for input key, so in order to detect all 16 key, check all 4 rows every polling iteration. If the output is 0, move on to check the next row, if it’s not, then that means a key in that row is pressed, jump to **print** to output the key button.

**Assignment 2:**

- Source 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:

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

# 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

- The result observed: Graphical user interface, application, table

Description automatically generated

-Explanation:

**Assginment 3:**

- Source 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 LabSim

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

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

li $t1, OUT\_ADRESS\_HEXA\_KEYBOARD

lb $a0, 0($t1)

bne $a0, $zero, prn\_cod

nop

li $t1, IN\_ADRESS\_HEXA\_KEYBOARD

li $t3, 0x82 # check row 2 and re-enable bit 7

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

li $t1, OUT\_ADRESS\_HEXA\_KEYBOARD

lb $a0, 0($t1)

bne $a0, $zero, prn\_cod

nop

li $t1, IN\_ADRESS\_HEXA\_KEYBOARD

li $t3, 0x84 # check row 3 and re-enable bit 7

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

li $t1, OUT\_ADRESS\_HEXA\_KEYBOARD

lb $a0, 0($t1)

bne $a0, $zero, prn\_cod

nop

li $t1, IN\_ADRESS\_HEXA\_KEYBOARD

li $t3, 0x88 # check row 4 and re-enable bit 7

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

li $t1, OUT\_ADRESS\_HEXA\_KEYBOARD

lb $a0, 0($t1)

prn\_cod: li $v0,34

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 $at, 0($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

- The result observed:

Graphical user interface, application

Description automatically generated

- Explanation:

In this sample code, when the keyboard interrupts, the interrupt program saves all registers that it is likely going to change to the stack (just like when calling procedure, then it polls keyboard input using the register, print the result and restore values back to the registers.

Just like in assignment 1, we are going to modify the code that do the polling so that instead of checking the last row only, it checks all the rows to detect all 16 keys.

**Assignment 4:**

- Source 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 $a0,2000 # 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\_caus:

mfc0 $t1, $13 # $t1 = Coproc0.causeIsCount:

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

- The result observed:

Graphical user interface, application, table

Description automatically generated

- Explanation:

The register **$13** (Coproc0.cause) holds the information about what causes the interrupt, bit number 10 set to 1 means that the interrupt is caused by counter, bit number 11 set to 1 means that the interrupt is caused by the key matrix.

In the interrupt handling program, we check register **$13** to determine the kind of interrupt and perform the right action

**Assignment 5:**

- Source code:

(Same as the source code given)

- The result observed:

Graphical user interface, text, application

Description automatically generated

- Explanation:

Keyboard input doesn’t cause interrupt when pressed so we have to trigger a trap when we want a keyboard press to behave like an interrupt. Use instruction **teqi** to interrupt whenever a key is pressed, and write a interrupt handling program at 0x80000180 like normal interrupt.