`

BÁO CÁO THỰC HÀNH

Laboratory Exercise 11 - Assignment 4, 5

Học phần: Thực hành Kiến trúc máy tính

Họ tên sinh viên: Phan Minh Anh Tuấn

MSSV: 20205227

Lớp: Công nghệ thông tin Việt – Pháp 01 K65

Assignment 4

Create a new project, type in, and build the program of Home Assignment 4. Upgrade the source code so that it could detect all 16 key buttons, from 0 to F.

as4.asm* | as5.asm | as3.asm | as1.asm

```
1  # Laboratory Exercise 11, Assignment 4
2  # Phan Minh Anh Tuan - 20205227
3  .eqv IN_ADDRESS_HEXA_KEYBOARD           0xFFFF0012
4  .eqv OUT_ADDRESS_HEXA_KEYBOARD          0xFFFF0014
5  .eqv COUNTER            0xFFFF0013      # Time Counter
6  .eqv MASK_CAUSE_COUNTER                 0x00000400      # Bit 10: Counter interrupt
7  .eqv MASK_CAUSE_KEYMATRIX 0x00000800            # Bit 11: Key matrix interrupt
8
9  .data
10 msg_keypress: .asciiz "  "
11 msg_counter: .asciiz "Count inteval: "
12 #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
13 # MAIN Procedure
14 #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
15 .text
16 main:
17         #-------------------------------------------------------
18         # Enable interrupts you expect
19         #-------------------------------------------------------
20         # Enable the interrupt of Keyboard matrix 4x4 of Digital Lab Sim
21         li $t1, IN_ADDRESS_HEXA_KEYBOARD
22         li $t3, 0x80            # bit 7 = 1 to enable
23         sb $t3, 0($t1)
24         # Enable the interrupt of TimeCounter of Digital Lab Sim
25         li $t1, COUNTER
26         sb $t1, 0($t1)
27
28         xor $s0, $s0, $s0
29 Loop:   nop
30         nop
31         nop
32 sleep:  addi $v0,$zero,32       # BUG: must sleep to wait for Time Counter
```

`

```
33          li $a0,200              # sleep 300 ms
34          syscall
35          nop                     # WARNING: nop is mandatory here.
36          b Loop
37  end_main:
38  #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
39  # GENERAL INTERRUPT SERVED ROUTINE for all interrupts
40  #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
41  .ktext 0x80000180
42  IntSR:  #-----------------------------------------------------
43          # SAVE the current REG FILE to stack
44          #-----------------------------------------------------
45          addi $sp,$sp,4    # Save $ra because we may change it later
46          sw $ra,0($sp)
47          addi $sp,$sp,4    # Save $at because we may change it later
48          sw $at,0($sp)
49          addi $sp,$sp,4   # Save $v0 because we may change it later
50          sw $v0,0($sp)
51          addi $sp,$sp,4   # Save $a0, because we may change it later
52          sw $a0,0($sp)
53          addi $sp,$sp,4   # Save $t1, because we may change it later
54          sw $t1,0($sp)
55          addi $sp,$sp,4   # Save $t3, because we may change it later
56          sw $t3,0($sp)
57
58  dis_int:li $t1, COUNTER          # BUG: must disable with Time Counter
59          sb $zero, 0($t1)
60          # no need to disable keyboard matrix interrupt
61          #-----------------------------------------------------
62          # Processing
63          #-----------------------------------------------------
64  get_caus:mfc0 $t1, $13           # $t1 = Coproc0.cause
65  IsCount:li $t2, MASK_CAUSE_COUNTER# if Cause value confirm Counter..
66          and $at, $t1,$t2
67          beq $at,$t2, Counter_Intr
68  IsKeyMa:li $t2, MASK_CAUSE_KEYMATRIX # if Cause value confirm Key..
69          and $at, $t1,$t2
70          beq $at,$t2, Keymatrix_Intr
71  others: j end_process            # other cases
72
73  Keymatrix_Intr:                  # Processing Key Matrix Interrupt
74  get_cod:
75  row1:   li $t1, IN_ADDRESS_HEXA_KEYBOARD
76          li $t3, 0x81     # check row 1 and re-enable bit 7
77          sb $t3, 0($t1)   # must reassign expected row
78          li $t1, OUT_ADDRESS_HEXA_KEYBOARD
79          lb $a0, 0($t1)
80          beq $a0, $0, row2
81          j prn_cod
82          nop
83  row2:   li $t1, IN_ADDRESS_HEXA_KEYBOARD
```

`

```
        li $t3, 0x82     # check row 2 and re-enable bit 7
        sb $t3, 0($t1)   # must reassign expected row
        li $t1, OUT_ADDRESS_HEXA_KEYBOARD
        lb $a0, 0($t1)
        beq $a0, $0, row3
        j prn_cod
        nop
row3:   li $t1, IN_ADDRESS_HEXA_KEYBOARD
        li $t3, 0x84     # check row 3 and re-enable bit 7
        sb $t3, 0($t1)   # must reassign expected row
        li $t1, OUT_ADDRESS_HEXA_KEYBOARD
        lb $a0, 0($t1)
        beq $a0, $0, row4
        j prn_cod
        nop
row4:   li $t1, IN_ADDRESS_HEXA_KEYBOARD
        li $t3, 0x88     # check row 4 and re-enable bit 7
        sb $t3, 0($t1)   # must reassign expected row
        li $t1, OUT_ADDRESS_HEXA_KEYBOARD
        lb $a0, 0($t1)
        j prn_cod
        nop
prn_cod:li $v0,34
        syscall
prn_msg:addi $v0, $zero, 4
        la $a0, msg_keypress
        syscall
prn_count_intr:
        li $v0, 4
        la $a0, msg_counter
        syscall
        addi $v0, $zero, 1
        add $a0, $s0, $zero      # print auto sequence number
        syscall
        addi $v0, $zero, 11
        li $a0, '\n'             # print endofline
        syscall
        add $s0, $0, $0
        j end_process
Counter_Intr:
        addi $s0, $s0, 1         # Processing Counter Interrupt
        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

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

Chương trình thực hiện ngắt khi có phím nào đó được bấm hoặc khi hết một khoảng thời gian. Mỗi khi có phím được bấm, chương trình sẽ in mã của phím đó cùng số lần ngắt theo thời gian giữa 2 lần bấm phím ra màn hình.

Mã của các phím như sau:

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

Chương trình con thực hiện ngắt nằm ở địa chỉ cố định 0x80000180. Khi có một phím được bấm, chương trình chính sẽ ngắt vì ta đã nạp giá trị vào địa chỉ IN_ADDRESS_HEXA_KEYBOARRD là 0x80, bit số 7 bằng 1 để cho phép ngắt:

```
        # Enable the interrupt of Keyboard matrix 4x4 of Digital Lab Sim
        li $t1, IN_ADDRESS_HEXA_KEYBOARD
        li $t3, 0x80            # bit 7 = 1 to enable
        sb $t3, 0($t1)
```

`

Chương trình cũng ngắt sau mỗi khoảng thời gian nhất định vì ta bật ngắt theo thời gian của Digital Lab Sim:

```
24          # Enable the interrupt of TimeCounter of Digital Lab Sim
25          li $t1, COUNTER
26          sb $t1, 0($t1)
```

Sau đó ta gán biến đếm số lần ngắt theo thời gian bằng 0:

```
28          xor $s0, $s0, $s0
```

Tiếp theo, chương trình chính sẽ thực hiện vòng lặp vô hạn và bên cạnh đó sẽ có ngắt.

Chương trình ngắt thực hiện lần lượt các bước sau:

- Lưu các thanh ghi ở chương trình chính đang được thực hiện trước đó vào ngăn xếp vì sau đó có thể ta sẽ thay đổi chúng:

```
42  IntSR:  #-----------------------------------------------------
43          # SAVE the current REG FILE to stack
44          #-----------------------------------------------------
45          addi $sp,$sp,4    # Save $ra because we may change it later
46          sw $ra,0($sp)
47          addi $sp,$sp,4    # Save $at because we may change it later
48          sw $at,0($sp)
49          addi $sp,$sp,4   # Save $v0 because we may change it later
50          sw $v0,0($sp)
51          addi $sp,$sp,4   # Save $a0, because we may change it later
52          sw $a0,0($sp)
53          addi $sp,$sp,4   # Save $t1, because we may change it later
54          sw $t1,0($sp)
55          addi $sp,$sp,4   # Save $t3, because we may change it later
56          sw $t3,0($sp)
```

- Ngừng Time Counter:

```
58  dis_int:li $t1, COUNTER          # BUG: must disable with Time Counter
59          sb $zero, 0($t1)
60          # no need to disable keyboard matrix interrupt
```

- Quá trình xử lý ngắt như sau:

```
64  get_caus:mfc0 $t1, $13           # $t1 = Coproc0.cause
65  IsCount:li $t2, MASK_CAUSE_COUNTER# if Cause value confirm Counter..
66          and $at, $t1,$t2
67          beq $at,$t2, Counter_Intr
68  IsKeyMa:li $t2, MASK_CAUSE_KEYMATRIX # if Cause value confirm Key..
69          and $at, $t1,$t2
70          beq $at,$t2, Keymatrix_Intr
71  others: j end_process            # other cases
```

  - Lấy giá trị ở thanh ghi số 13, status trong bộ đồng xử lý C0, chứa các thiết lập về tình trạng ngắt và kiểm tra xem là ngắt gì.
  - Nếu nguyên nhân ngắt là do bộ đếm thì nhảy đến hàm Counter_Intr. Nếu nguyên nhân ngắt là do bàn phím thì nhảy đến hàm Keymatrix_Intr.
  - Nếu không phải cả 2 nguyên nhân ptreen thì đến hàm end_process
- Hàm Keymatrix_Intr xử lý ngắt do bàn phím như sau:

`

- Đi tìm code của phím vừa được bấm (get_cod) bằng cách kiểm tra từng hàng, ví dụ ở đây ta xem hàng 1 (row1):

```
74  get_cod:
75  row1:   li $t1, IN_ADDRESS_HEXA_KEYBOARD
76          li $t3, 0x81     # check row 1 and re-enable bit 7
77          sb $t3, 0($t1)   # must reassign expected row
78          li $t1, OUT_ADDRESS_HEXA_KEYBOARD
79          lb $a0, 0($t1)
80          beq $a0, $0, row2
81          j prn_cod
```

Ban đầu ta lưu địa chỉ IN_ADDRESS_HEXA_KEYBOARRD vào thanh ghi $t1 là 0x81 để kiểm tra hàng 1
Sau đó ta lưu địa chỉ OUT_ADDRESS_HEXA_KEYBOARRD vào thanh ghi $t1 và dùng lb để lấy ra giá trị số được lưu ở địa chỉ này. Cho giá trị này vào thanh ghi $a0.
So sánh $a0 với 0, nếu $a0 bằng 0 tức là không có số nào ở hàng này được bấm, ta tiếp tục kiểm tra hàng tiếp theo, ngược lại thì nhảy đến hàm prn_cod để in ra mã của số được bấm.

Các hàng còn lại được kiểm tra tương tự.

- In ra thông báo cho người dùng nếu muốn:

```
108 prn_msg:addi $v0, $zero, 4
109         la $a0, msg_keypress
110         syscall
```

- In ra số lần ngắt giữa lần bấm trước và lần này:

```
111 prn_count_intr:
112         li $v0, 4
113         la $a0, msg_counter
114         syscall
115         addi $v0, $zero, 1
116         add $a0, $s0, $zero      # print auto sequence number
117         syscall
118         addi $v0, $zero, 11
119         li $a0, '\n'             # print endofline
120         syscall
121         add $s0, $0, $0
```

- Kết thúc xử lý:

```
122         j end_process
```

- Hàm Counter _Intr xử lý ngắt do bộ đếm thời gian: Cộng thêm một vào biến đếm số lần ngắt

```
123 Counter_Intr:
124         addi $s0, $s0, 1         # Processing Counter Interrupt
125         j end_process
```

- Hàm end_process kết thúc xử lý: gán thanh ghi số 13 về 0 sau khi xử lý nguyên nhân ngắt xong

`

```
126 end_process: mtc0 $zero, $13     # Must clear cause reg
```

- Bật lại bộ đếm Time Counter:

```
127 en_int:
128         #-------------------------------------------------------
129         # Re-enable interrupt
130         #-------------------------------------------------------
131         li $t1, COUNTER
132         sb $t1, 0($t1)
```

- Tính địa chỉ lệnh quay lại của chương trình chính là epc = epc + 4:

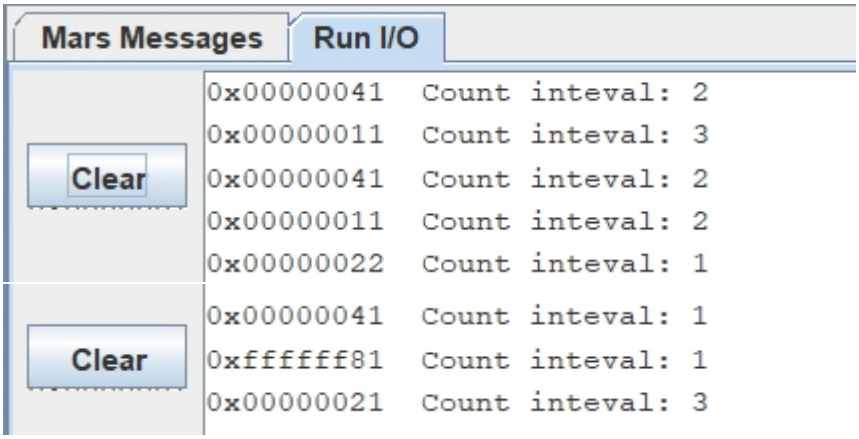
```
133         #-------------------------------------------------------
134         # Evaluate the return address of main routine
135         # epc <= epc + 4
136         #-------------------------------------------------------
137 next_pc:
138         mfc0 $at, $14           # $at <= Coproc0.$14 = Coproc0.epc
139         addi $at, $at, 4        # $at = $at + 4 (next instruction)
140         mtc0 $at, $14           # Coproc0.$14 = Coproc0.epc <= $at
```

- Khôi phục lại các thanh ghi của chương trình trước đó đã lưu và quay lại vị trí trước khi ngắt:

```
142         #-------------------------------------------------------
143         # RESTORE the REG FILE from STACK
144         #-------------------------------------------------------
145 restore:lw $t3, 0($sp)          # Restore the registers from stack
146         addi $sp,$sp,-4
147         lw $t1, 0($sp)          # Restore the registers from stack
148         addi $sp,$sp,-4
149         lw $a0, 0($sp)          # Restore the registers from stack
150         addi $sp,$sp,-4
151         lw $v0, 0($sp)          # Restore the registers from stack
152         addi $sp,$sp,-4
153         lw $at, 0($sp)          # Restore the registers from stack
154         addi $sp,$sp,-4
155         lw $ra, 0($sp)          # Restore the registers from stack
156         addi $sp,$sp,-4
157
158 return: eret # Return from exception
```

Kết quả khi thực hiện chương trình là:

`



Assignment 5

Create a new project, type in, and build the program of Home Assignment 5.

as4.asm* as5.asm* as3.asm as1.asm

```
1  # Laboratory Exercise 11, Assignment 5
2  # Phan Minh Anh Tuan - 20205227
3  .eqv KEY_CODE   0xFFFF0004              # ASCII code from keyboard, 1 byte
4  .eqv KEY_READY  0xFFFF0000              # =1 if has a new keycode ?
5                                          # Auto clear after lw
6
7  .eqv DISPLAY_CODE       0xFFFF000C      # ASCII code to show, 1 byte
8  .eqv DISPLAY_READY      0xFFFF0008      # =1 if the display has already to do
9                                          # Auto clear after sw
10
11 .eqv MASK_CAUSE_KEYBOARD 0x0000034      # Keyboard Cause
12 .text
13                 li $k0, KEY_CODE
14                 li $k1, KEY_READY
15                 li $s0, DISPLAY_CODE
16                 li $s1, DISPLAY_READY
17 loop:           nop
18 WaitForKey:     lw $t1, 0($k1)                  # $t1 = [$k1] = KEY_READY
19                 beq $t1, $zero, WaitForKey      # if $t1 == 0 then Polling
20 MakeIntR:       teqi $t1, 1 # if $t1 = 1 then raise an Interrupt
21                 j loop
22 #---------------------------------------------------------------
23 # Interrupt subroutine
24 #---------------------------------------------------------------
25 .ktext 0x80000180
26 get_caus:       mfc0 $t1, $13           # $t1 = Coproc0.cause
27 IsCount:        li $t2, MASK_CAUSE_KEYBOARD# if Cause value confirm Keyboard..
28                 and $at, $t1,$t2
29                 beq $at,$t2, Counter_Keyboard
30                 j end_process
31 Counter_Keyboard:
32 ReadKey:        lw $t0, 0($k0)                  # $t0 = [$k0] = KEY_CODE
33
```

`

```
34  WaitForDis:     lw $t2, 0($s1)                  # $t2 = [$s1] = DISPLAY_READY
35                  beq $t2, $zero, WaitForDis      # if $t2 == 0 then Polling
36
37  Encrypt:        addi $t0, $t0, 1        # change input key
38
39  ShowKey:        sw $t0, 0($s0)          # show key
40                  nop
41
42  end_process:
43  next_pc:        mfc0 $at, $14 # $at <= Coproc0.$14 = Coproc0.epc
44                  addi $at, $at, 4 # $at = $at + 4 (next instruction)
45                  mtc0 $at, $14 # Coproc0.$14 = Coproc0.epc <= $at
46  return:         eret # Return from exception
47
```

Bộ xử lý MIPS cho phép tạo ra ngắt mềm, bằng lệnh teq, hoặc teqi. Ở chương trình này, ta tạo ra ngắt mềm khi một phím được nhấn như sau:

```
17  loop:           nop
18  WaitForKey:     lw $t1, 0($k1)                  # $t1 = [$k1] = KEY_READY
19                  beq $t1, $zero, WaitForKey      # if $t1 == 0 then Polling
20  MakeIntR:       teqi $t1, 1 # if $t1 = 1 then raise an Interrupt
21                  j loop
```

- Khi có một phím được nhấn, địa chỉ KEY_READY sẽ bằng 1
- Lệnh teqi $t1,1 so sánh $t1 với 1, nếu $t1 = 1 thì dẫn đến ngắt

Ở chương trình con thực hiện ngắt ở địa chỉ 0x80000180, quá trình xử lý ngắt diễn ra như sau:

- Đầu tiên kiểm tra nguyên nhân ngắt có phải từ bàn phím hay không, nếu có thực hiện hàm Counter_Keyboard, nếu không thực hiện end_process

```
25  .ktext 0x80000180
26  get_caus:       mfc0 $t1, $13           # $t1 = Coproc0.cause
27  IsCount:        li $t2, MASK_CAUSE_KEYBOARD# if Cause value confirm Keyboard..
28                  and $at, $t1,$t2
29                  beq $at,$t2, Counter_Keyboard
30                  j end_process
```

- Hàm Counter_Keyborad:

```
31  Counter_Keyboard:
32  ReadKey:        lw $t0, 0($k0)                  # $t0 = [$k0] = KEY_CODE
33
34  WaitForDis:     lw $t2, 0($s1)                  # $t2 = [$s1] = DISPLAY_READY
35                  beq $t2, $zero, WaitForDis      # if $t2 == 0 then Polling
36
37  Encrypt:        addi $t0, $t0, 1        # change input key
38
39  ShowKey:        sw $t0, 0($s0)          # show key
40                  nop
```

  - Đọc mã của phím vừa nhấn ở địa chỉ KEY_CODE đang lưu trong thanh ghi $k0
  - Hàm WaitForDis chờ để hiển thị ký tự vừa nhấn lên màn hình

`

- Hàm Encrypt mã hóa phím vừa nhấn bằng cách cộng mã đó với 1
- Hàm Showkey thực hiện hiển thị ký tự sau khi mã hóa lên màn hình

Kết quả thực hiện chương trình là: