

BÁO CÁO THỰC HÀNH KIẾN TRÚC MÁY TÍNH TUẦN 11

Họ và tên: Phan Khánh Vũ

MSSV: 20235880

1. Assignment 1

- Nhập chương trình:

```
.eqv IN_ADDRESS_HEX_KEYBOARD 0xFFFF0012
.eqv OUT_ADDRESS_HEX_KEYBOARD 0xFFFF0014
.eqv newline 0xa
.text
main:
    li t1, IN_ADDRESS_HEX_KEYBOARD
    li t2, OUT_ADDRESS_HEX_KEYBOARD

polling:
    li t3, 0x01 # start with row 1
row_loop:
    sb t3, 0(t1) # must reassign expected row
    lb a0, 0(t2) # read scan code of key button
    beq a0, zero, next_row # if no key pressed, continue to next row
print:
    li a7, 34 # print integer (hexa)
    ecall
    li a0, newline
```

```

        li a7, 11
        ecall
sleep:
        li a0, 100 # sleep 100ms
        li a7, 32
        ecall
        li a0, 100 # sleep 100ms
        li a7, 32
        ecall
        li a0, 100 # sleep 100ms
        li a7, 32
        ecall
next_row:
        slli t3, t3, 1 # move to next row (shift left 1 bit)
        li t4, 0x10 # maximum row mask
        blt t3, t4, row_loop # if not finished all rows, continue row_loop
        j polling # continue polling

```

- **Các bước thực hiện của chương trình:**
- Bắt đầu quét từ hàng đầu tiên (0x01).
- Gửi giá trị dòng cần quét vào địa chỉ 0xFFFF0012.
- Đọc giá trị trả về từ địa chỉ 0xFFFF0014 để biết cột nào có phím được nhấn (nếu có).
- Nếu có phím được nhấn (giá trị khác 0), in ra mã phím = hàng OR cột (ví dụ $0x02 \mid 0x04 = 0x06$).
- Chờ một chút (delay) để tránh in liên tục gây treo phần mềm.
- Chuyển sang hàng tiếp theo, lặp lại từ bước 2 đến 5.

- Khi quét xong cả 4 hàng, quay lại hàng đầu tiên → lặp vô hạn.
- TÁC DỤNG:
 - Giúp chương trình nhận diện phím được nhấn mà không cần ngắt (interrupt).
 - Được dùng trong các thiết bị đầu vào đơn giản như bàn phím ma trận, điều khiển từ xa, hoặc hệ thống nhúng đơn giản.
 - Là kỹ thuật thăm dò (polling): liên tục kiểm tra trạng thái phần cứng.
- **Chạy chương trình :**

```

Edit Execute
Lab11_1.asm Lab11_2.asm Lab11_3.asm
    li t3, 0x01 # start with row 1
row_loop:
    sb t3, 0(t1) # must reassign expected row
    lb a0, 0(t2) # read scan code of key buttonu
    beq a0, zero, next_row # if no key pressed, continue to next row
print:
    li a7, 34 # print integer (hexa)
    ecall
    li a0, newline
    li a7, 11
    ecall
sleep:
    li a0, 100 # sleep 100ms
    li a7, 32
    ecall
    li a0, 100 # sleep 100ms
    li a7, 32
    ecall
    li a0, 100 # sleep 100ms
    li a7, 32
    ecall
next_row:
    slli t3, t3, 1 # move to next row (shift left 1 bit)
    li t4, 0x10 # maximum row mask
    blt t3, t4, row_loop # if not finished all rows, continue row_loop
    j polling # continue polling
  
```

Line: 20 Column: 9 ☐ Show Line Numbers

Messages Run I/O

Assemble: assembling C:\Users\DeLL\Documents\THKMT\Lab11\Lab11_1.asm

Assemble: operation completed successfully.

Clear

- **Quan sát kết quả của các thanh ghi và chương trình:**

```

li a0, 100 # sleep 100ms
li a7, 32
ecall
li a0, 100 # sleep 100ms
li a7, 32
ecall
next_row:
slli t3, t3, 1 # move to next row
li t4, 0x10 # maximum row mask
blt t3, t4, row_loop # if not finished
j polling # continue polling

```

Line: 20 Column: 9 ☐ Show Line Numbers

Messages Run I/O

0xffffffff81
0xffffffff81
0xffffffff81
0xffffffff81
0xffffffff81
0xffffffff81

Clear

Digital Lab Sim, Version 1.0 (Didier Teifreto)

Digital Lab Sim

| | | | |
|---|---|---|---|
| 0 | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 |
| 8 | 9 | a | b |
| c | d | e | f |

Tool Control

Disconnect from Program Reset Help Close

- Sửa chương trình để hiện ra mã hexa không có 6 ký tự F ở đầu (**0xffffffff81 => 0x00000081**) ta sửa lb -> lbu

```

row_loop:
sb t3, 0(t1) # must reassign expected row
lbu a0, 0(t2) # read scan code of key buttonu
beq a0, zero, next_row # if no key pressed, continue to next row

```

Messages Run I/O

0x00000081
0x00000081
0x00000081
0x00000081
0x00000081
0x00000081

Clear

Digital Lab Sim, Version 1.0 (Didier Teifreto)

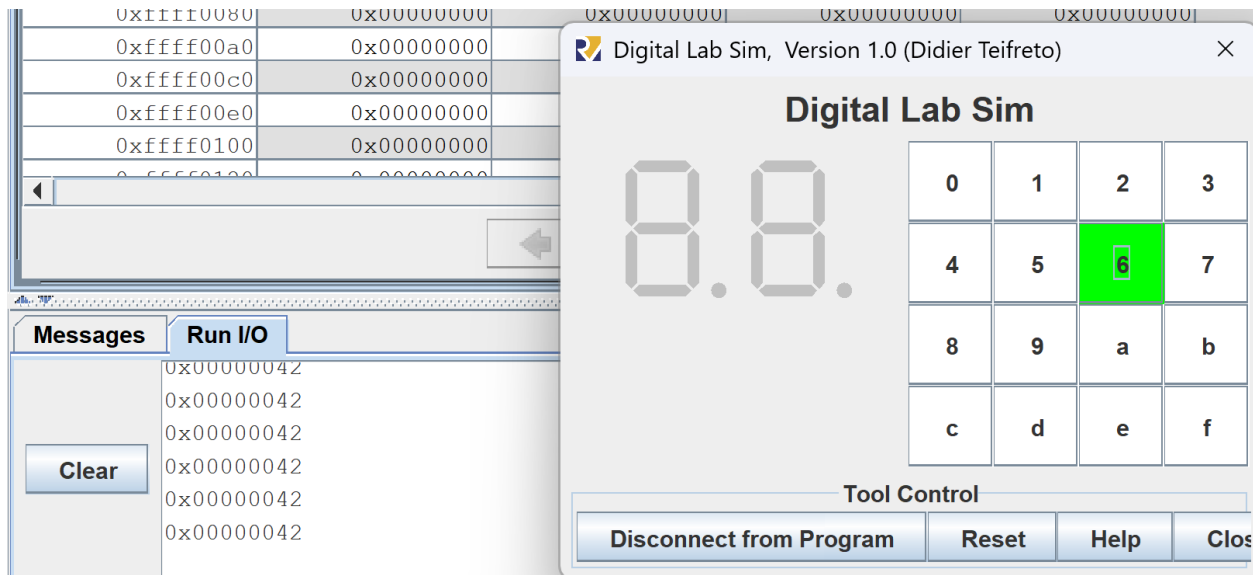
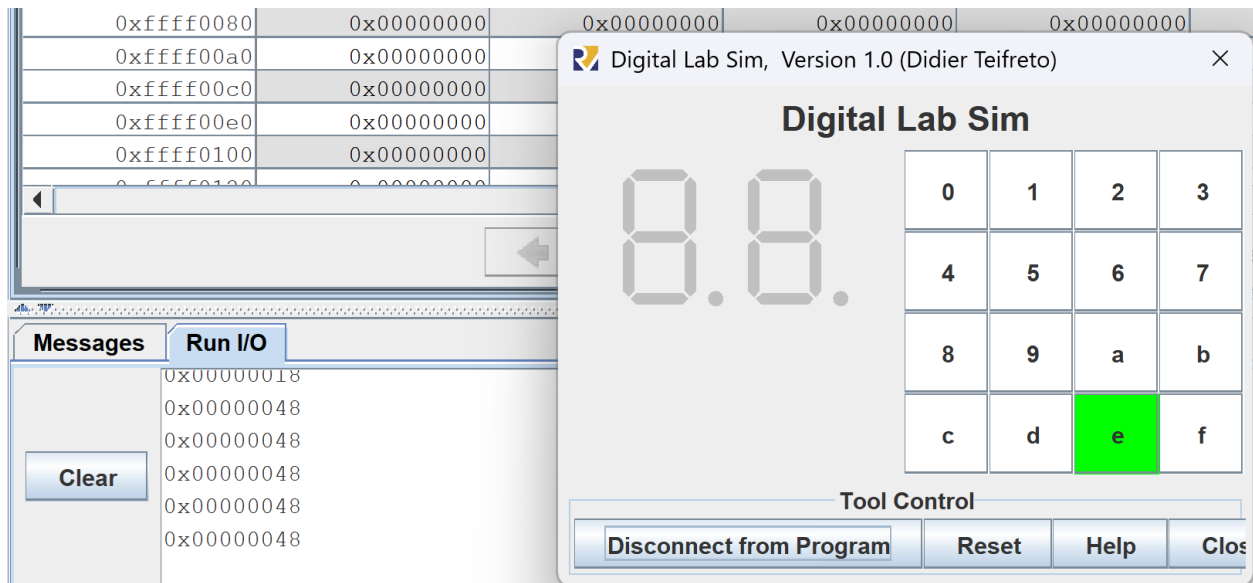
Digital Lab Sim

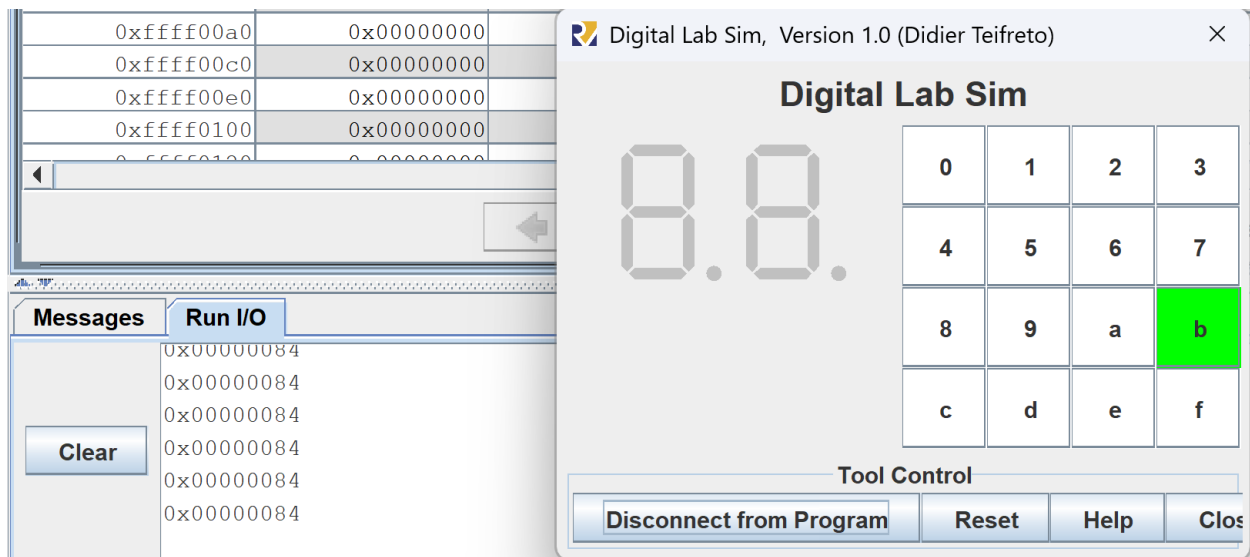
| | | | |
|---|---|---|---|
| 0 | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 |
| 8 | 9 | a | b |
| c | d | e | f |

Tool Control

Disconnect from Program Reset Help Close

- Kiểm tra một số trường hợp khác:





2. Assignment 2:

- Nhập chương trình :

```

1. .eqv IN_ADDRESS_HEXKEYBOARD 0xFFFF0012
2. .data
3. message: .asciz "Someone's pressed a button.\n"
4. # -----
5. # MAIN Procedure
6. # -----
7. .text
8. main:
9.     # Load the interrupt service routine address to the UTVEC register
10.    la t0, handler
11.    csrrs zero, utvec, t0
12.    # Set the UEIE (User External Interrupt Enable) bit in UIE register
13.    li t1, 0x100
14.    csrrs zero, uie, t1 # uie - ueie bit (bit 8)
15.    # Set the UIE (User Interrupt Enable) bit in USTATUS register
16.    csrrsi zero, ustatus, 1 # ustatus - enable uie (bit 0)
17.
18.    # Enable the interrupt of keypad of Digital Lab Sim

```

```

19.      li t1, IN_ADDRESS_HEXA_KEYBOARD
20.      li t3, 0x80 # bit 7 = 1 to enable interrupt
21.      sb t3, 0(t1)
22.# -----
23.# No-end loop, main program, to demo the effective of interrupt
24.# -----
25.loop:
26.      nop
27.      nop
28.      nop
29.      j loop
30.end_main:
31.# -----
32.# Interrupt service routine
33.# -----
34.handler:
35.      # ebreak # Can pause the execution to observe registers
36.      # Saves the context
37.      addi sp, sp, -8
38.      sw a0, 0(sp)
39.      sw a7, 4(sp)
40.      # Handles the interrupt
41.      # Shows message in Run I/O
42.      li a7, 4
43.      la a0, message
44.      ecall
45.      # Restores the context
46.      lw a7, 4(sp)
47.      lw a0, 0(sp)
48.      addi sp, sp, 8
49.      # Back to the main procedure
50.      Uret

```

- Cách hoạt động của code :

1. Chuẩn bị dữ liệu:

- Đặt địa chỉ bàn phím: 0xFFFF0012
- Khai báo chuỗi: "Someone's pressed a button.\n"

2. Cấu hình ngắt:

- Gán địa chỉ hàm handler cho thanh ghi utvec.
- Bật ngắt ngoài (uie) và tổng ngắt (ustatus).

3. Bật ngắt cho bàn phím:

- Ghi 0x80 vào địa chỉ bàn phím để bật chức năng ngắt.

4. Chờ ngắt trong vòng lặp:

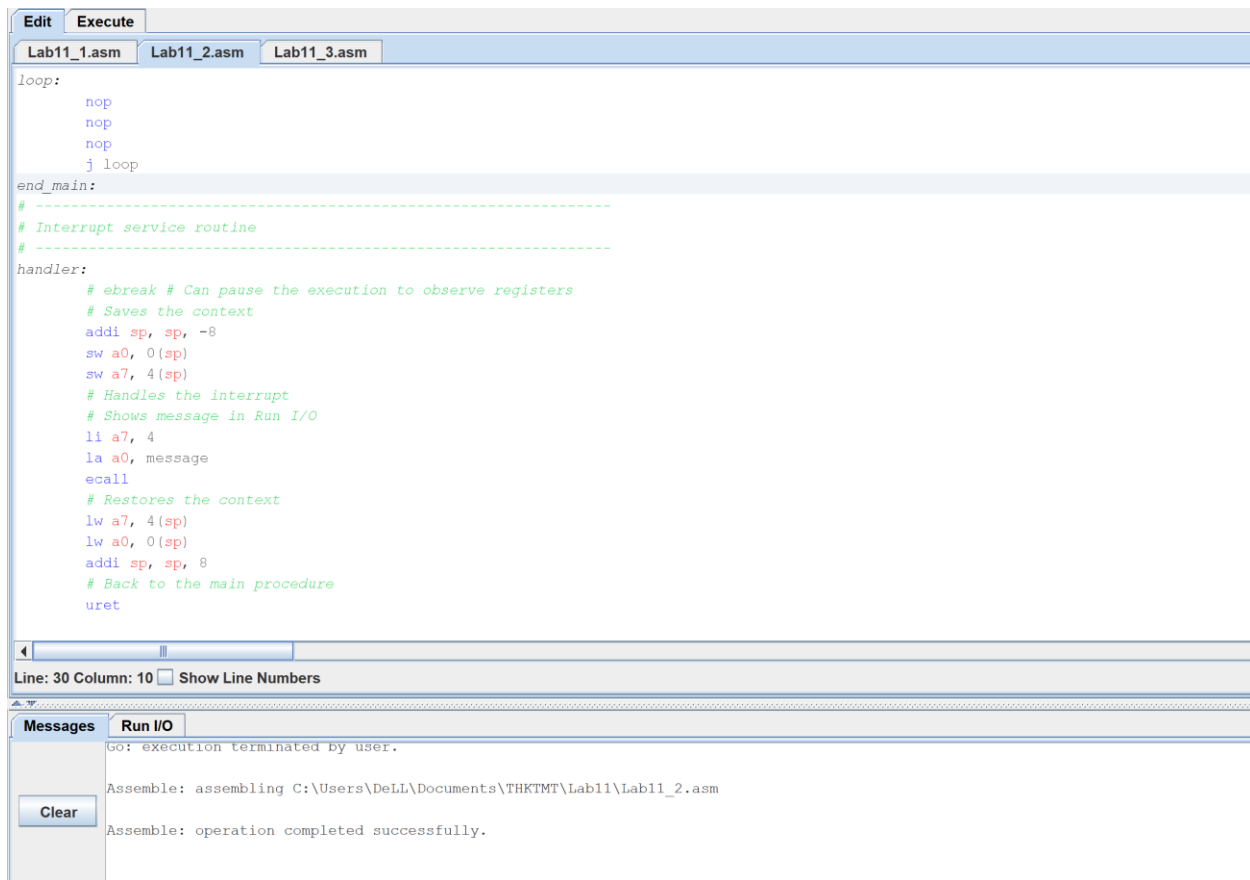
- Vòng loop chỉ chứa nop → CPU chờ sự kiện.

5. Khi phím được nhấn (ngắt xảy ra):

- CPU nhảy đến handler.

6. Xử lý ngắt (handler):

- Lưu trạng thái.
- In thông báo qua syscall.
- Khôi phục trạng thái.
- Quay lại chương trình chính.
- Tóm lại:
Chương trình dùng ngắt để phản ứng ngay khi có phím nhấn, thay vì kiểm tra liên tục.
- **Chạy chương trình:**



The screenshot shows a MIPS assembly editor with three tabs: Lab11_1.asm, Lab11_2.asm, and Lab11_3.asm. The active tab is Lab11_2.asm, which contains the following assembly code:

```
loop:
    nop
    nop
    nop
    j loop
end_main:
# -----
# Interrupt service routine
# -----
handler:
    # ebreak # Can pause the execution to observe registers
    # Saves the context
    addi sp, sp, -8
    sw a0, 0(sp)
    sw a7, 4(sp)
    # Handles the interrupt
    # Shows message in Run I/O
    li a7, 4
    la a0, message
    ecall
    # Restores the context
    lw a7, 4(sp)
    lw a0, 0(sp)
    addi sp, sp, 8
    # Back to the main procedure
    uret
```

Below the code editor, a status bar indicates "Line: 30 Column: 10" and a checkbox for "Show Line Numbers". At the bottom, a "Messages" window is open, showing the following messages:

- Go: execution terminated by user.
- Assemble: assembling C:\Users\DeLL\Documents\THKMT\Lab11\Lab11_2.asm
- Assemble: operation completed successfully.

A "Clear" button is located next to the messages.

- Các bước hoạt động của chương trình:

- + Khởi tạo hằng số là địa chỉ điều khiển ngắt từ bàn phím hexa và thông báo cần in.
- + Sau khi gán địa chỉ trình phục vụ ngắt vào thanh ghi utvec, bật các cờ cho phép xử lý ngắt trong các thanh ghi uie và ustatus.
- + Tiếp theo, bật tính năng sinh ngắt từ bàn phím bằng cách ghi giá trị thích hợp vào thanh ghi điều khiển.
- + Chương trình chính chuyển sang vòng lặp chờ để đợi ngắt xảy ra.
- + Khi người dùng nhấn phím, thiết bị tự động gửi ngắt và CPU nhảy vào trình xử lý ngắt.
- + Trong trình xử lý ngắt, chương trình sẽ hiển thị thông báo ra màn hình, sau đó quay lại chương trình chính và tiếp tục vòng lặp.

- So sánh trước và sau ngắt:

Bảng Trạng Thái Trước và Sau Ngắt (Kèm Vị Trí Code)

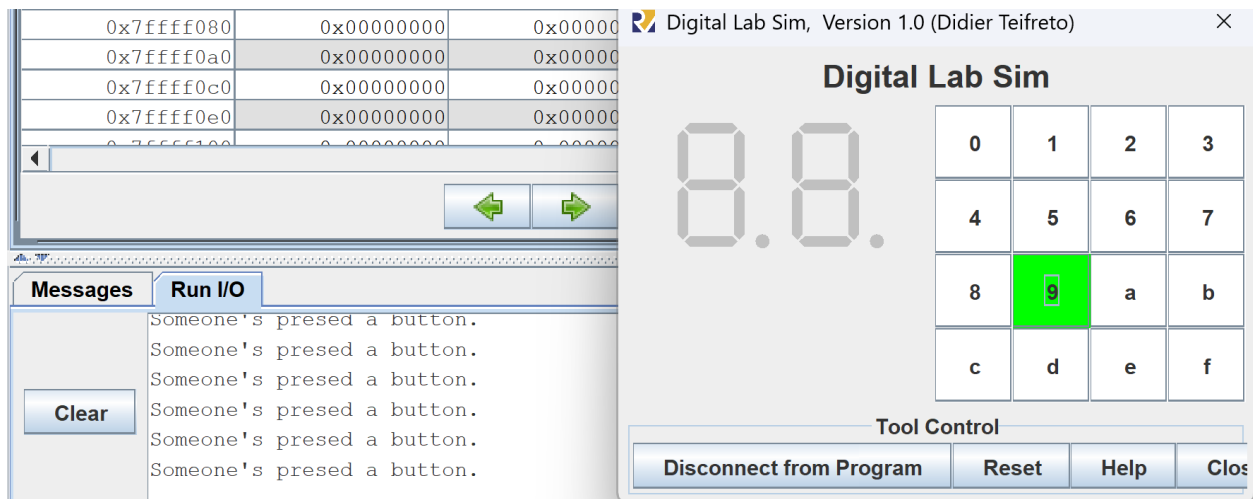
| Vị trí | Dòng lệnh | PC (Program Counter) | UEPC (User Exception PC) | Chú thích |
|-----------------------|-----------|----------------------|--------------------------|---|
| Trước khi ngắt xảy ra | 25–29 | loop | – | CPU đang chạy vòng lặp loop, thực hiện các lệnh nop. |
| Ngắt xảy ra | - | – | loop | Ngắt ngoài được kích hoạt do nhấn nút bàn phím; địa chỉ tiếp theo (PC) được lưu vào uepc. |
| Vào hàm handler | 34 | handler | loop | PC được cập nhật theo địa chỉ handler trong utvec. |
| Trong handler | 42–44 | ecall | loop | In thông báo bằng syscall thông qua ecall. |
| Kết thúc handler | 49–50 | uret | loop | Lệnh uret khôi phục PC từ uepc, quay về chương trình chính. |
| Quay lại chương trình | 25–29 | loop | – | Tiếp tục thực hiện chương trình chính tại loop, chờ ngắt tiếp theo. |

- Bảng giá trị :

| No. | Vị trí | utvec | uie | ustatus | Pc | Chú thích |
|-----|--------|-------|------|---------|------------|----------------------|
| 1 | 1 | 0x00 | 0x00 | 0x00 | 0x00400000 | Bắt đầu chương trình |

| | | | | | | |
|---|----|----------|--------|--------|------------|------------------------------------|
| 2 | 8 | 0x00 | 0x00 | 0x00 | 0x00400000 | Khai báo hằng số |
| 3 | 16 | 0x400038 | 0x0100 | 0x0001 | 0x00400018 | Khởi tạo và cấu hình ngắt |
| 4 | 21 | 0x400038 | 0x0100 | 0x0001 | 0x00400028 | Khi phát hiện ngắt |
| 5 | 50 | 0x400038 | 0x0100 | 0x0001 | 0x00400060 | Thực hiện in kết quả ra màn hình |
| 6 | 26 | 0x400038 | 0x0100 | 0x0001 | 0x00400028 | Đợi ngắt và thực hiện chương trình |

- **Kết quả chương trình:**



Kết quả đúng với yêu cầu

3. Assignment 3: Cập nhật mã nguồn để chương trình có thể in ra mã của tất cả 16 nút bấm trên keypad.

- **Nhập chương trình:**

```
.eqv IN_ADDRESS_HEX_A_KEYBOARD 0xFFFF0012
.eqv OUT_ADDRESS_HEX_A_KEYBOARD 0xFFFF0014

.data
message: .asciz "\nKey scan code: "
```

```

newline: .asciz "\n"

# -----

# MAIN Procedure
# -----

.text
main:
# Load the interrupt service routine address to the UTVEC register
    la t0, handler
    csrrs zero, utvec, t0

# Set the UEIE (User External Interrupt Enable) bit in UIE register
    li t1, 0x100
    csrrs zero, uie, t1 # uie - ueie bit (bit 8)

# Set the UIE (User Interrupt Enable) bit in USTATUS register
    csrrsi zero, ustatus, 1 # ustatus - enable uie (bit 0)

# Enable the interrupt of keypad of Digital Lab Sim
    li t1, IN_ADDRESS_HEXKEYBOARD
    li t3, 0x80 # bit 7 = 1 to enable interrupt
    sb t3, 0(t1)

# -----

# Loop to print a sequence numbers
# -----

    xor s0, s0, s0 # count = s0 = 0

```

```

loop:
    addi s0, s0, 1 # count = count + 1
prn_seq:
    addi a7, zero, 1
    add a0, s0, zero # Print auto sequence number
    ecall
# Print EOL
    addi a7, zero, 4
    la a0, newline
    ecall
sleep:
    addi a7, zero, 32
    li a0, 300 # Sleep 300 ms
    ecall
    j loop
end_main:

# -----
# Interrupt service routine
# -----

handler:
# Saves the context
    addi sp, sp, -16
    sw a0, 0(sp)

```

```
sw a7, 4(sp)
```

```
sw t1, 8(sp)
```

```
sw t2, 12(sp)
```

```
# Handles the interrupt
```

```
prn_msg:
```

```
addi a7, zero, 4
```

```
la a0, message
```

```
ecall
```

```
get_key_code:
```

```
li t1, IN_ADDRESS_HEXKEYBOARD
```

```
li t2, OUT_ADDRESS_HEXKEYBOARD
```

```
polling:
```

```
li t3, 0x01 # start with row 1
```

```
row_loop:
```

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

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

```
beqz a0, next_row # if no key pressed, continue to next row
```

```
print:
```

```
li a7, 34 # print integer (hexa)
```

```
ecall
```

```
li a0, 0xa
```

```
li a7, 11
```

```
ecall
```

```
next_row:
```

```
slli t3, t3, 1 # move to next row (shift left 1 bit)
```

```
li t4, 0x10 # maximum row mask
```

```
blt t3, t4, row_loop # if not finished all rows, continue row_loop
```

```
# After polling, must re-enable keypad interrupt
```

```
re_enable_interrupt:
```

```
li t1, IN_ADDRESS_HEXKEYBOARD
```

```
li t3, 0x80 # bit 7 = 1 to enable interrupt again
```

```
sb t3, 0(t1)
```

```
j restore
```

```
restore:
```

```
# Restores the context
```

```
lw t2, 12(sp)
```

```
lw t1, 8(sp)
```

```
lw a7, 4(sp)
```

```
lw a0, 0(sp)
```

```
addi sp, sp, 16
```

```
# Back to the main procedure
```

```
uret
```

- **Các bước thực hiện của chương trình:**

- + Khởi tạo địa chỉ ngắt và gán trình phục vụ ngắt vào thanh ghi utvec.
- + Bật cờ cho phép ngắt ngoài (UEIE trong uie) và cờ tổng cho phép xử lý ngắt (UIE trong ustatus).
- + Bật chức năng tạo ngắt từ bàn phím hexa bằng cách ghi 0x80 vào địa chỉ điều khiển.
- + Thực hiện vòng lặp in dãy số tăng dần, nghỉ mỗi 300ms giữa các lần in.
- + Khi có phím được nhấn, chương trình tự động nhảy vào handler để xử lý ngắt.
- + Trong handler, chương trình in thông báo, quét các hàng của bàn phím để tìm mã phím đã nhấn, sau đó in mã phím ra màn hình.
- + Sau khi xử lý xong, chương trình bật lại ngắt và quay về tiếp tục in dãy số.

- **Chạy chương trình :**

EditExecute

Lab11_3.asm

Text editor for composing RISC-V programs.

```
li a0, 0xa
li a7, 11
ecall

next_row:
    slli t3, t3, 1 # move to next row (shift left 1 bit)
    li t4, 0x10 # maximum row mask
    blt t3, t4, row_loop # if not finished all rows, continue row_loop

# After polling, must re-enable keypad interrupt
re_enable_interrupt:
    li t1, IN_ADDRESS_HEX_A_KEYBOARD
    li t3, 0x80 # bit 7 = 1 to enable interrupt again
    sb t3, 0(t1)

    j restore

restore:
# Restores the context
    lw t2, 12(sp)
    lw t1, 8(sp)
    lw a7, 4(sp)
    lw a0, 0(sp)
    addi sp, sp, 16
# Back to the main procedure
    uret
```

Line: 80 Column: 7 Show Line Numbers

MessagesRun I/O

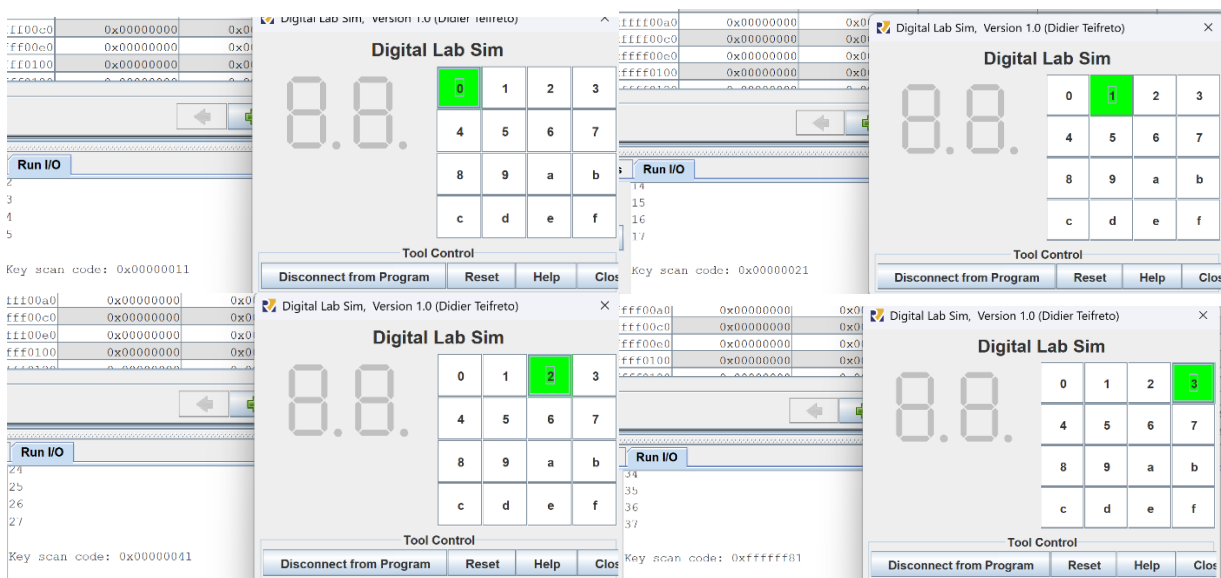
Go: execution paused by user: Lab11_2.asm

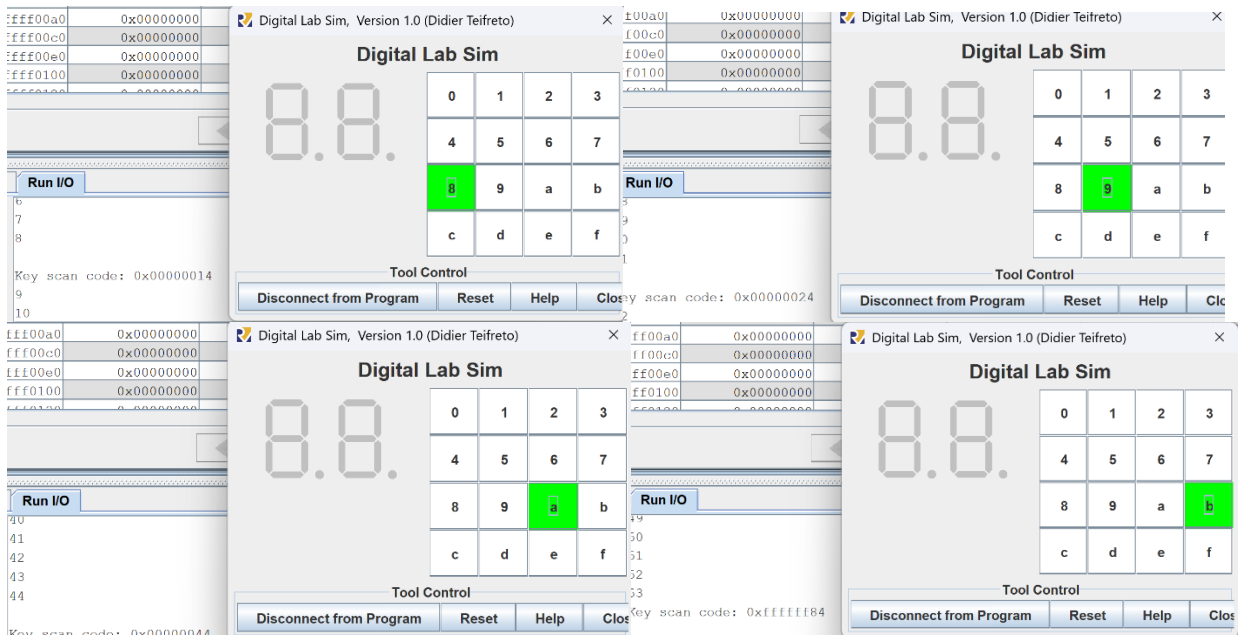
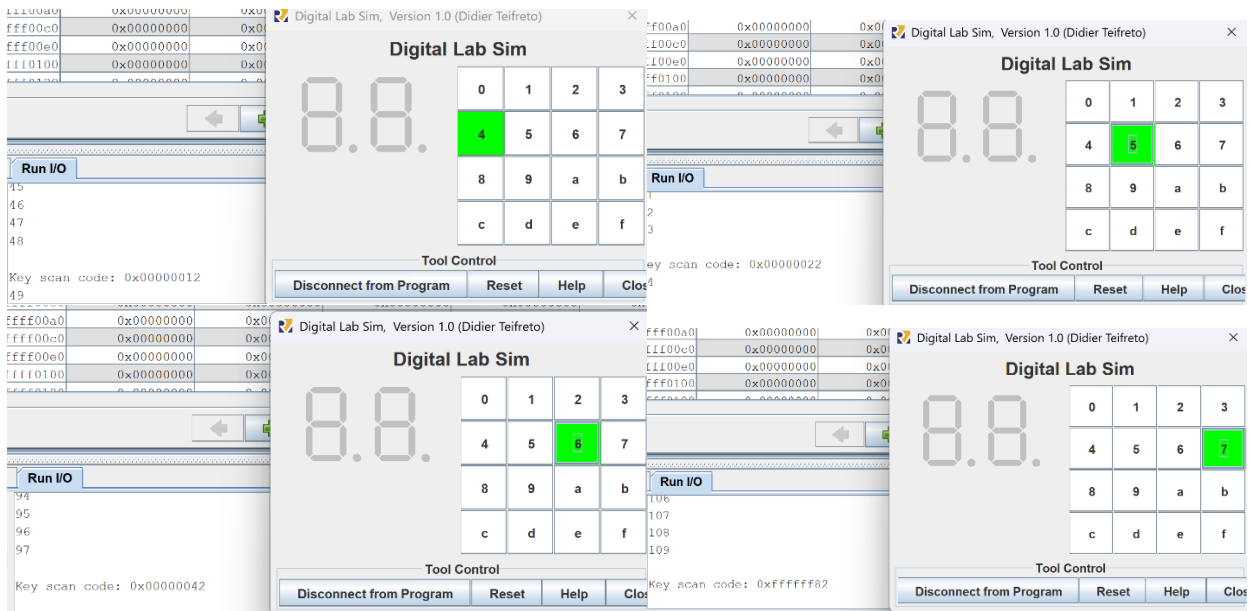
Assemble: assembling C:\Users\DeLL\Documents\THKTMT\Lab11\Lab11_3.asm

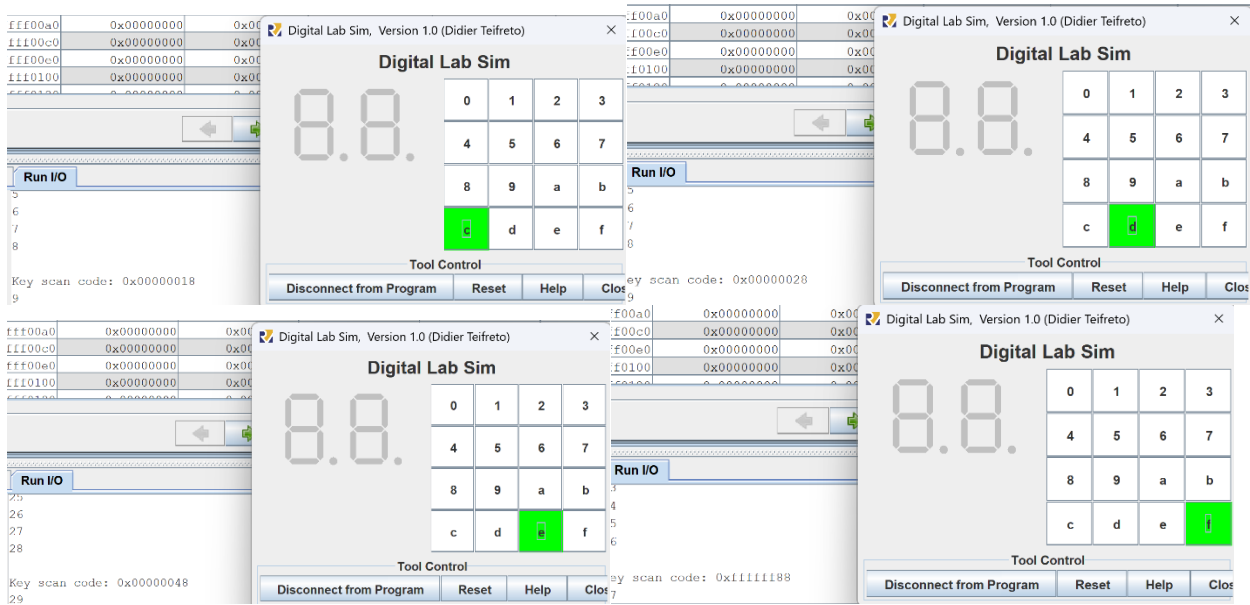
Assemble: operation completed successfully.

Clear

- Kiểm tra kết quả in ra mã của tất cả 16 nút bấm trên keypad.







4. Bài tập mở rộng:

- Nhập chương trình:

```
.eqv IN_ADDRESS_HEX_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_ADDRESS_HEX_KEYBOARD 0xFFFF0014
```

```
.eqv newline 0xa
```

```
.eqv RED 0xFF0000
```

```
.eqv BLACK 0x0
```

```
.eqv MONITOR_SCREEN_ADDR 0x10000000 #global data
```

```
.data
```

```
k11: .word RED BLACK BLACK BLACK BLACK BLACK BLACK BLACK
      BLACK BLACK BLACK BLACK BLACK BLACK BLACK BLACK
```

```
k21: .word BLACK RED BLACK BLACK BLACK BLACK BLACK BLACK
```

BLACK BLACK BLACK BLACK BLACK BLACK BLACK BLACK

k41: .word BLACK BLACK RED BLACK BLACK BLACK BLACK BLACK
BLACK BLACK BLACK BLACK BLACK BLACK BLACK BLACK

k81: .word BLACK BLACK BLACK RED BLACK BLACK BLACK BLACK
BLACK BLACK BLACK BLACK BLACK BLACK BLACK BLACK

k12: .word BLACK BLACK BLACK BLACK RED BLACK BLACK BLACK
BLACK BLACK BLACK BLACK BLACK BLACK BLACK BLACK

k22: .word BLACK BLACK BLACK BLACK BLACK RED BLACK BLACK
BLACK BLACK BLACK BLACK BLACK BLACK BLACK BLACK

k42: .word BLACK BLACK BLACK BLACK BLACK BLACK RED BLACK
BLACK BLACK BLACK BLACK BLACK BLACK BLACK BLACK

k82: .word BLACK BLACK BLACK BLACK BLACK BLACK BLACK RED
BLACK BLACK BLACK BLACK BLACK BLACK BLACK BLACK

k14: .word BLACK BLACK BLACK BLACK BLACK BLACK BLACK
BLACK RED BLACK BLACK BLACK BLACK BLACK BLACK

k24: .word BLACK BLACK BLACK BLACK BLACK BLACK BLACK
BLACK BLACK RED BLACK BLACK BLACK BLACK BLACK

k44: .word BLACK BLACK BLACK BLACK BLACK BLACK BLACK
BLACK BLACK BLACK RED BLACK BLACK BLACK BLACK

k84: .word BLACK BLACK BLACK BLACK BLACK BLACK BLACK
BLACK BLACK BLACK BLACK RED BLACK BLACK BLACK

k18: .word BLACK BLACK BLACK BLACK BLACK BLACK BLACK
BLACK BLACK BLACK BLACK BLACK RED BLACK BLACK

k28: .word BLACK BLACK BLACK BLACK BLACK BLACK BLACK
BLACK BLACK BLACK BLACK BLACK BLACK RED BLACK

k48: .word BLACK BLACK BLACK BLACK BLACK BLACK BLACK
BLACK BLACK BLACK BLACK BLACK BLACK BLACK RED

k88: .word BLACK BLACK BLACK BLACK BLACK BLACK BLACK

BLACK BLACK BLACK BLACK BLACK BLACK BLACK BLACK RED

.text

main:

li t1, IN_ADDRESS_HEXA_KEYBOARD

li t2, OUT_ADDRESS_HEXA_KEYBOARD

polling:

li t3, 0x01 # start with row 1

row_loop:

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

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

beqz a0, next_row # if no key pressed, continue to next row

print:

li a7, 34 # print integer (hexa)

ecall

j convert

sleep:

```
li a0, 100 # sleep 100ms
```

```
li a7, 32
```

```
ecall
```

```
next_row:
```

```
slli t3, t3, 1 # move to next row (shift left 1 bit)
```

```
li t4, 0x10 # maximum row mask
```

```
blt t3, t4, row_loop # if not finished all rows, continue row_loop
```

```
j polling # continue polling
```

```
convert:
```

```
li t5, 0x11
```

```
beq a0, t5, key11
```

```
li t5, 0x21
```

```
beq a0, t5, key21
```

```
li t5, 0x41
```

```
beq a0, t5, key41
```

```
li t5, 0xFFFFF81
```

```
beq a0, t5, key81
```

```
li t5, 0x12
```

beq a0, t5, key12

li t5, 0x22

beq a0, t5, key22

li t5, 0x42

beq a0, t5, key42

li t5, 0xFFFFFFFF82

beq a0, t5, key82

li t5, 0x14

beq a0, t5, key14

li t5, 0x24

beq a0, t5, key24

li t5, 0x44

beq a0, t5, key44

li t5, 0xFFFFFFFF84

beq a0, t5, key84

li t5, 0x18

```
beq a0, t5, key18
```

```
li t5, 0x28
```

```
beq a0, t5, key28
```

```
li t5, 0x48
```

```
beq a0, t5, key48
```

```
li t5, 0xFFFFFFFF88
```

```
beq a0, t5, key88
```

```
key11:
```

```
    la s0, k11
```

```
    li s1, MONITOR_SCREEN_ADDR
```

```
    li s2, 16
```

```
copy_loop11:
```

```
    lw s3, 0(s0)
```

```
    sw s3, 0(s1)
```

```
    addi s0, s0, 4
```

```
    addi s1, s1, 4
```

```
    addi s2, s2, -1
```

```
    bnez s2, copy_loop11
```

```
    j polling
```


key21:

la s0, k21

li s1, MONITOR_SCREEN_ADDR

li s2, 16

copy_loop21:

lw s3, 0(s0)

sw s3, 0(s1)

addi s0, s0, 4

addi s1, s1, 4

addi s2, s2, -1

bnez s2, copy_loop21

j polling

key41:

la s0, k41

li s1, MONITOR_SCREEN_ADDR

li s2, 16

copy_loop41:

lw s3, 0(s0)

sw s3, 0(s1)

addi s0, s0, 4

addi s1, s1, 4

addi s2, s2, -1

bnez s2, copy_loop41

j polling

key81:

la s0, k81

li s1, MONITOR_SCREEN_ADDR

li s2, 16

copy_loop81:

lw s3, 0(s0)

sw s3, 0(s1)

addi s0, s0, 4

addi s1, s1, 4

addi s2, s2, -1

bnez s2, copy_loop81

j polling

key12:

la s0, k12

li s1, MONITOR_SCREEN_ADDR

li s2, 16

copy_loop12:

lw s3, 0(s0)

sw s3, 0(s1)

addi s0, s0, 4

addi s1, s1, 4

```
    addi s2, s2, -1
    bnez s2, copy_loop12
    j polling
```

key22:

```
    la s0, k22
    li s1, MONITOR_SCREEN_ADDR
    li s2, 16
```

copy_loop22:

```
    lw s3, 0(s0)
    sw s3, 0(s1)
    addi s0, s0, 4
    addi s1, s1, 4
    addi s2, s2, -1
    bnez s2, copy_loop22
    j polling
```

key42:

```
    la s0, k42
    li s1, MONITOR_SCREEN_ADDR
    li s2, 16
```

copy_loop42:

```
    lw s3, 0(s0)
    sw s3, 0(s1)
```

```
    addi s0, s0, 4
    addi s1, s1, 4
    addi s2, s2, -1
    bnez s2, copy_loop42
    j polling
```

key82:

```
    la s0, k82
    li s1, MONITOR_SCREEN_ADDR
    li s2, 16
```

copy_loop82:

```
    lw s3, 0(s0)
    sw s3, 0(s1)
    addi s0, s0, 4
    addi s1, s1, 4
    addi s2, s2, -1
    bnez s2, copy_loop82
    j polling
```

key14:

```
    la s0, k14
    li s1, MONITOR_SCREEN_ADDR
    li s2, 16
```

copy_loop14:

```
lw s3, 0(s0)
sw s3, 0(s1)
addi s0, s0, 4
addi s1, s1, 4
addi s2, s2, -1
bnez s2, copy_loop14
j polling
```

key24:

```
la s0, k24
li s1, MONITOR_SCREEN_ADDR
li s2, 16
```

copy_loop24:

```
lw s3, 0(s0)
sw s3, 0(s1)
addi s0, s0, 4
addi s1, s1, 4
addi s2, s2, -1
bnez s2, copy_loop24
j polling
```

key44:

```
la s0, k44
li s1, MONITOR_SCREEN_ADDR
```

```

        li s2, 16
copy_loop44:
        lw s3, 0(s0)
        sw s3, 0(s1)
        addi s0, s0, 4
        addi s1, s1, 4
        addi s2, s2, -1
        bnez s2, copy_loop44
        j polling

key84:
        la s0, k84
        li s1, MONITOR_SCREEN_ADDR
        li s2, 16
copy_loop84:
        lw s3, 0(s0)
        sw s3, 0(s1)
        addi s0, s0, 4
        addi s1, s1, 4
        addi s2, s2, -1
        bnez s2, copy_loop84
        j polling

key18:

```

```
        la s0, k18
        li s1, MONITOR_SCREEN_ADDR
        li s2, 16
copy_loop18:
        lw s3, 0(s0)
        sw s3, 0(s1)
        addi s0, s0, 4
        addi s1, s1, 4
        addi s2, s2, -1
        bnez s2, copy_loop18
        j polling
```

```
key28:
        la s0, k28
        li s1, MONITOR_SCREEN_ADDR
        li s2, 16
copy_loop28:
        lw s3, 0(s0)
        sw s3, 0(s1)
        addi s0, s0, 4
        addi s1, s1, 4
        addi s2, s2, -1
        bnez s2, copy_loop28
        j polling
```

key48:

la s0, k48

li s1, MONITOR_SCREEN_ADDR

li s2, 16

copy_loop48:

lw s3, 0(s0)

sw s3, 0(s1)

addi s0, s0, 4

addi s1, s1, 4

addi s2, s2, -1

bnez s2, copy_loop48

j polling

key88:

la s0, k88

li s1, MONITOR_SCREEN_ADDR

li s2, 16

copy_loop88:

lw s3, 0(s0)

sw s3, 0(s1)

addi s0, s0, 4

addi s1, s1, 4

addi s2, s2, -1

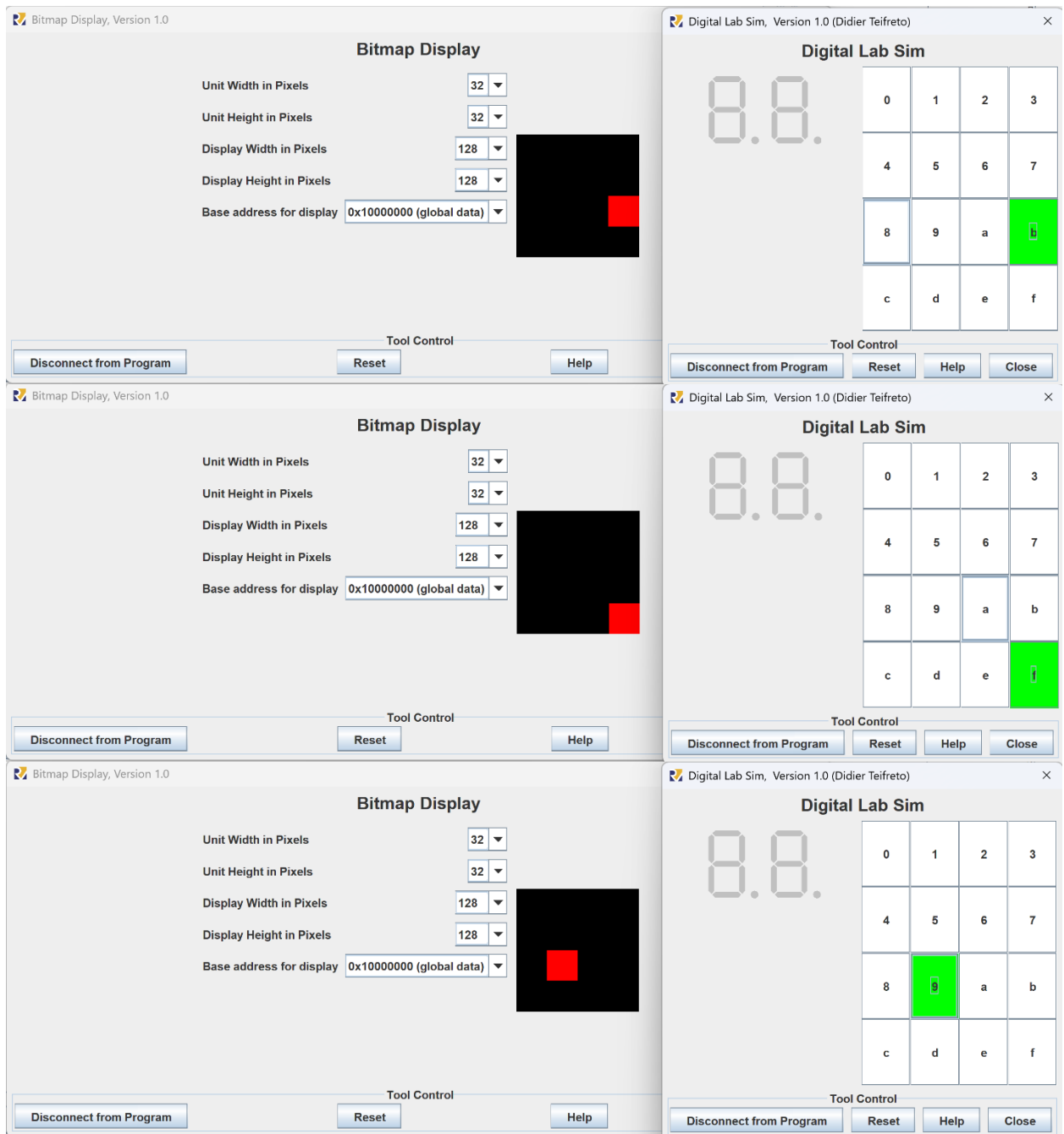

```
bnez s2, copy_loop88
```

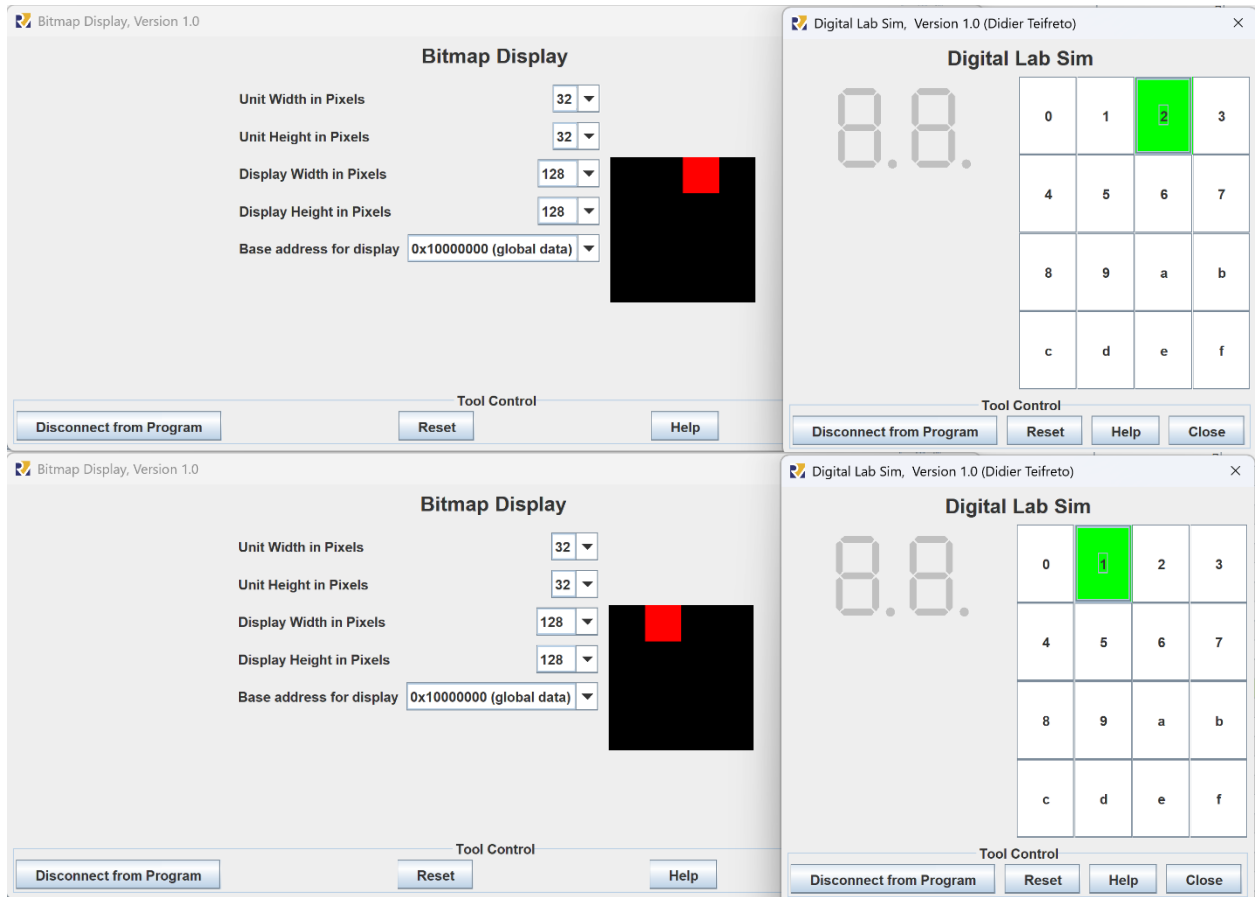
```
j polling
```

- **Các bước thực hiện của chương trình:**

- + Khởi tạo các hằng số địa chỉ bàn phím HEXA và địa chỉ màn hình (bitmap). Các dòng dữ liệu k11, k21, ..., k88 dùng để lưu cấu hình màu sắc hiển thị tương ứng với từng phím bấm.
- + Trong phần main, chương trình khởi tạo địa chỉ vào/ra của bàn phím bằng cách đưa địa chỉ `IN_ADDRESS_HEXA_KEYBOARD` và `OUT_ADDRESS_HEXA_KEYBOARD` vào các thanh ghi t1 và t2.
- + Vào vòng lặp polling: bắt đầu quét từ hàng đầu tiên (t3 = 0x01), sau đó ghi giá trị hàng vào địa chỉ điều khiển, rồi đọc mã phím đang được nhấn từ bàn phím thông qua t2. Nếu không có phím nào được nhấn thì tiếp tục quét hàng tiếp theo.
- + Nếu có phím được nhấn (a0 ≠ 0), chương trình sẽ in mã phím ở dạng hexa ra màn hình thông qua syscall 34, rồi chuyển sang phần convert.
- + Trong phần convert, chương trình so sánh mã phím đã nhấn (a0) với các giá trị cố định tương ứng với từng phím như 0x11, 0x28,... Nếu trùng thì nhảy đến nhãn tương ứng (key11, key28, v.v).
- + Mỗi nhãn keyXY tương ứng với một dòng dữ liệu màu (kXY). Chương trình sẽ nạp địa chỉ dữ liệu đó vào thanh ghi s0, địa chỉ màn hình vào s1, và lặp 16 lần: mỗi lần copy 1 word màu từ vùng dữ liệu sang màn hình (tức hiển thị dãy pixel lên màn hình).
- + Sau khi hiển thị bitmap tương ứng với phím, chương trình quay lại polling để tiếp tục quét phím tiếp theo.

- **Kết quả của chương trình:**





Kết quả thỏa mãn với yêu cầu đề bài.