BÁO CÁO THỰC HÀNH
IT3280 – 156788 – THỰC HÀNH KIẾN TRÚC MÁY TÍNH

NỘI DUNG
Bài 11. Lập trình xử lý ngắt

Họ và tên	Nguyễn Minh Quân
Mã số sinh viên	20235816

Assignment 1
1. Yêu cầu

Assignment 1

Tạo project thực hiện Home Assignment 1. Cập nhật mã nguồn để chương trình có thể in ra mã của tất cả 16 nút bấm trên keypad.

```
# -----------------------------------------------------
#           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_ADDRESS_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_ADDRESS_HEXA_KEYBOARD      0xFFFF0014
```

```
.text
main:
    li  t1, IN_ADDRESS_HEXA_KEYBOARD
    li  t2, OUT_ADDRESS_HEXA_KEYBOARD
    li  t3, 0x08        # check row 4 with key C, D, E, F

polling:
    sb  t3, 0(t1 )      # must reassign expected row
    lb  a0, 0(t2)       # read scan code of key button
print:
    li  a7, 34          # print integer (hexa)
    ecall
sleep:
    li  a0, 100         # sleep 100ms
    li  a7, 32
    ecall
back_to_polling:
    j     polling       # continue polling
```

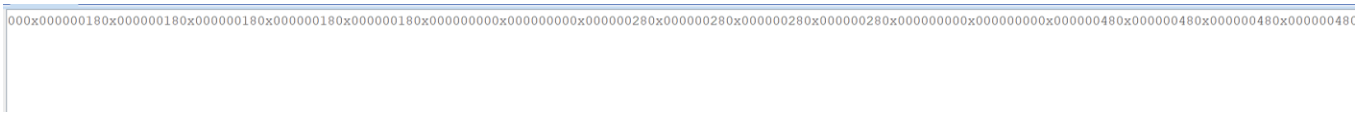
2. Thực hiện

a. Chạy chương trình ban đầu

* Chương trình

```
.eqv IN_ADDRESS_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_ADDRESS_HEXA_KEYBOARD   0xFFFF0014

.text
main:
    li t1, IN_ADDRESS_HEXA_KEYBOARD
    li t2, OUT_ADDRESS_HEXA_KEYBOARD
    li t3, 0x08             # check row 4 with key C, D, E, F

polling:
    sb t3, 0(t1)            # must reassign expected row
    lb a0, 0(t2)            # read scan code of key button

print:
    li a7, 34               # print integer (hex)
    ecall

sleep:
    li a0, 100              # sleep 100ms
    li a7, 32
    ecall

back_to_polling:
    j polling               # continue polling
```

* Kết quả



b. In ra mã 16 nút bấm trên keypad

* Chương trình:

```
.eqv IN_ADDRESS_HEXA_KEYBOARD    0xFFFF0012
.eqv OUT_ADDRESS_HEXA_KEYBOARD   0xFFFF0014

.text
main:
    li t1, IN_ADDRESS_HEXA_KEYBOARD   # t1 = input address
    li t2, OUT_ADDRESS_HEXA_KEYBOARD  # t2 = output address

polling:
    li t3, 0x1   # Bắt đầu từ hàng đầu tiên (row 0)

scan_rows:
    sb t3, 0(t1)          # gửi hàng cần quét
    lb t4, 0(t2)          # đọc mã nút bấm (0 nếu không có nút)

    beqz t4, next_row     # nếu không có nút thì quét hàng tiếp theo

    # nếu có nút thì in ra mã
    mv a0, t4
    li a7, 34             # ecall 34: print integer (hexa)
    ecall

next_row:
    slli t3, t3, 1        # dịch trái t3 (0x1 -> 0x2 -> 0x4 -> 0x8)
    li t5, 0x10           # sau 0x8 (dịch trái nữa thành 0x10)

    blt t3, t5, scan_rows # nếu chưa hết 4 hàng, quay lại quét tiếp

    # Sau khi quét xong 4 hàng, nghỉ 100ms rồi quét lại
sleep:
    li a0, 100            # sleep 100ms
    li a7, 32
    ecall

    j polling             # lặp lại từ đầu
```

* Kết quả

Run I/O

0x00000110x00000210x00000410x00000410xffffff810x00000120x00000220x00000420xffffff820x00000140x00000240x00000440xffffff840xffffff840x00000180x00000280x00000048

0210x00000410x00000410xffffff810x00000120x00000220x00000420xffffff820x00000140x00000240x00000440xffffff840xffffff840x00000180x00000280x00000480xffffff880xffffff88

Assignment 2

1. Yêu cầu

Assignment 2

Tạo project để thực hiện và thử nghiệm Home Assignment 2. Chạy ở chế độ từng dòng lệnh, quan sát giá trị của các thanh ghi để hiểu cách chương trình hoạt động.

2. Thực hiện

a. Chương trình

```
.eqv IN_ADDRESS_HEXA_KEYBOARD    0xFFFF0012

.data
message: .asciz "Someone's pressed a button.\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_HEXA_KEYBOARD
    li      t3, 0x80              # bit 7 = 1 to enable interrupt
    sb      t3, 0(t1)

    # ------------------------------------------------
    # No-end loop, main program, to demo the effective of interrupt
    # ------------------------------------------------
loop:
    nop
    nop
    nop
    j loop

end_main:

# ------------------------------------------------
# Interrupt service routine
# ------------------------------------------------
handler:
    # 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
```

ecall Issue a system call : Execute the system call specified by value in a7

b. Các giá trị thanh ghi và cách hoạt động

- Trong main: thiết lập môi trường interrupt ➔ chỉ t0, t1, t3 thay đổi.

- Trong handler: lưu và phục hồi context ➔ sp, a0, a7 thay đổi tạm thời để xử lý ngắt.

Thanh ghi	**Vị trí thay đổi**	**Ý nghĩa / Công dụng**
t0	main	Nạp địa chỉ handler vào t0 để gán cho utvec
t1	main	Nạp các giá trị cấu hình enable interrupt (0x100) và địa chỉ bàn phím
t3	main	Nạp giá trị 0x80 để enable interrupt bàn phím
sp	handler	Lưu và khôi phục ngữ cảnh (dịch con trỏ stack)

a0	handler	Lưu giá trị cũ vào stack + nạp địa chỉ message để in ra
a7	handler	Lưu giá trị cũ vào stack + nạp số dịch vụ hệ thống ecall (dịch vụ in chuỗi)

Assignment 3

1. Yêu cầu

Assignment 3

Tạo project để thực hiện và thử nghiệm Home Assignment 3. Chạy ở chế độ từng dòng lệnh, quan sát giá trị của các thanh ghi để hiểu cách chương trình hoạt động. Cập nhật mã nguồn để chương trình có thể in ra mã của tất cả 16 nút bấm trên keypad.

2. Thực hiện

a. Chương trình

```
.eqv IN_ADDRESS_HEXA_KEYBOARD 0xFFFF0012
.eqv OUT_ADDRESS_HEXA_KEYBOARD 0xFFFF0014

.data
    message: .asciz "Key scan code: "

# ---------------------------------------------------------------------------
# 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
    csrrs   zero, ustatus, 1 # ustatus - enable uie (bit 0)

    # Enable the interrupt of keypad of Digital Lab Sim
    li      t1, IN_ADDRESS_HEXA_KEYBOARD
    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
    addi    a7, zero, 11
    li      a0, '\n'         # Print EOL
    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)

prn_msg:
    addi    a7, zero, 4
    la      a0, message
    ecall
```

```
get_key_code:
    li      t1, IN_ADDRESS_HEXA_KEYBOARD
    li      t2, 0x8B        # Check row 4 and re-enable bit 7
    sb      t2, 0(t1)       # Must reassign expected row
    li      t1, OUT_ADDRESS_HEXA_KEYBOARD
    lb      a0, 0(t1)

prn_key_code:
    addi    a7, zero, 34
    ecall
    addi    a7, zero, 11
    li      a0, '\n'        # Print EOL
    ecall

    # 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
```

b. Sự thay đổi các thanh ghi

- t0: lưu địa chỉ handler
- t1: địa chỉ thiết bị ngoại vi (IN/OUT bàn phím)
- t2: đọc giá trị dòng bàn phím
- a0: dữ liệu cần in (số, chuỗi, mã phím)
- a7: syscall code (ecall)
- s0: biến đếm số vòng lặp
- sp: thay đổi khi lưu/phục hồi context

c. In ra mã 16 nút trên keypad

```
.eqv IN_ADDRESS_HEXA_KEYBOARD  0xFFFF0012
.eqv OUT_ADDRESS_HEXA_KEYBOARD 0xFFFF0014

.data
    message: .asciz "Key scan code: "

# ---------------------------------------------------------------
# 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
    li t4, 1
    csrrs   zero, ustatus, t4 # ustatus - enable uie (bit 0)

    # Enable the interrupt of keypad of Digital Lab Sim
    li      t1, IN_ADDRESS_HEXA_KEYBOARD
    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
    addi    a7, zero, 11
    li      a0, '\n'        # Print EOL
    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)

prn_msg:
    addi    a7, zero, 4
    la      a0, message
    ecall

get_key_code:
    li      t1, IN_ADDRESS_HEXA_KEYBOARD
    lb      t2, 0(t1)       # Read which row was activated

```

```
    # Send back the row value to re-enable the interrupt
    sb      t2, 0(t1)

    # Now read the key value
    li      t1, OUT_ADDRESS_HEXA_KEYBOARD
    lb      a0, 0(t1)          # a0 = key code (0x00 to 0x0F)

prn_key_code:
    addi    a7, zero, 34       # print in hexa format
    ecall
    addi    a7, zero, 11
    li      a0, '\n'
    ecall

    # Restore 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
```

Assignment 4

1. Yêu cầu

Assignment 4

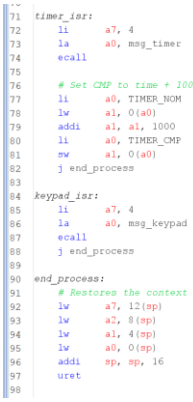
Tạo project để thực hiện và thử nghiệm Home Assignment 4. Chạy ở chế độ từng dòng lệnh, quan sát giá trị của các thanh ghi để hiểu cách chương trình hoạt động.

2. Thực hiện

a. Chương trình

```
#-----------------------------------------------------
# Define addresses
.equ IN_ADDRESS_HEXA_KEYBOARD  0xFFFF0012
.equ TIMER_NOW                 0xFFFF0018
.equ TIMER_CMP                 0xFFFF0020

.equ MASK_CAUSE_TIMER  4
.equ MASK_CAUSE_KEYPAD 8

.data
msg_keypad: .asciz "Someone has pressed a key!\n"
msg_timer:  .asciz "Time interval\n"

#-----------------------------------------------------
# MAIN Procedure
.text
main:
    la      t0, handler
    csrrs   zero, utvec, t0

    li      t1, 0x10
    csrrs   zero, uie, t1      # uie - ueie bit (bit 4) - timer interrupt
    li      t1, 0x100
    csrrs   zero, uie, t1      # uie - ueie bit (bit 8) - external interrupt
    csrrs   zero, ustatus, 1   # ustatus - enable UIE - global interrupt

    # Enable interrupts you expect
    li      t1, IN_ADDRESS_HEXA_KEYBOARD
    li      t2, 0x80           # bit 7 = 1 to enable interrupt
    sb      t2, 0(t1)

    # Enable the timer interrupt
    li      t1, TIMER_CMP
    li      t2, 1000
    sw      t2, 0(t1)
```

```
#-----------------------------------------------------
# No-end loop, main program, to demo the effective of interrupt
#
loop:
    nop
    nop
    nop
    j loop
end_main:

#-----------------------------------------------------
# Interrupt service routine
#
handler:
    # Saves the context
    addi    sp, sp, -16
    sw      a0, 0(sp)
    sw      a1, 4(sp)
    sw      a2, 8(sp)
    sw      a7, 12(sp)

    # Handles the interrupt
    csrr    a1, ucause
    li      a2, 0x7FFFFFFF
    and     a1, a1, a2         # Clear interrupt bit to get the value

    li      a2, MASK_CAUSE_TIMER
    beq     a1, a2, timer_isr

    li      a2, MASK_CAUSE_KEYPAD
    beq     a1, a2, keypad_isr

    j end_process
```



```
timer_isr:
    li      a7, 4
    la      a0, msg_timer
    ecall

    # Set CMP to time + 1000
    li      a0, TIMER_NOW
    lw      a1, 0(a0)
    addi    a1, a1, 1000
    li      a0, TIMER_CMP
    sw      a1, 0(a0)
    j end_process

keypad_isr:
    li      a7, 4
    la      a0, msg_keypad
    ecall
    j end_process

end_process:
    # Restores the context
    lw      a7, 12(sp)
    lw      a2, 8(sp)
    lw      a1, 4(sp)
    lw      a0, 0(sp)
    addi    sp, sp, 16
    uret
```

b. Các giá trị thanh ghi và cách hoạt động

* Thay đổi thanh ghi

Thanh ghi	Vai trò trong bài	Khi thay đổi
utvec	Địa chỉ hàm ngắt	Được thiết lập lúc khởi động
uie	Cho phép ngắt timer + keypad	Được bật trong main
ustatus	Bật toàn bộ user interrupt	Được bật trong main
ucause	Xác định loại ngắt xảy ra	Đọc và phân tích trong handler
sp	Lưu phục hồi context	Giảm đi/lấy lại 16 bytes
a0 a1 a2 a7	Dùng tạm cho ecall và tính toán	Bị save/restore trong handler

* Cách hoạt động

 Khởi động chương trình:

- Cài đặt địa chỉ hàm ngắt (handler) vào thanh ghi utvec.
- Bật cho phép ngắt timer + bàn phím (uie, ustatus).
- Bật ngắt riêng cho bàn phím hexa và lập lịch timer lần đầu (CMP = 1000).

 Vào vòng lặp loop::

- Chương trình chính chỉ nop (không làm gì), chờ có ngắt xảy ra.

 Khi có ngắt (timer hoặc phím):

- CPU tự động nhảy vào handler.
- handler sẽ:
    - Lưu trạng thái thanh ghi (a0, a1, a2, a7).

- Đọc nguyên nhân ngắt từ ucause.
- Nếu:
  - Ngắt timer ➔ in "Time interval", cộng thêm 1000 đơn vị vào timer.
  - Ngắt bàn phím ➔ in "Someone has pressed a key!".
- Phục hồi trạng thái thanh ghi.
- Trở lại chương trình chính bằng uret.

Assignment 5

1. Yêu cầu

Assignment 5

Tạo project để thực hiện và thử nghiệm Home Assignment 5. Chạy ở chế độ từng dòng lệnh, quan sát giá trị của các thanh ghi để hiểu cách chương trình hoạt động.

2. Thực hiện

a. Chương trình

```
.data
    message: .asciz "Exception occurred.\n"

.text
main:
try:
    la      t0, catch
    csrrw   zero, utvec, t0         # Set utvec to the handler address
    csrrsi  zero, ustatus, 1         # Set interrupt enable bit in ustatus

    lw      zero, 0                 # Trigger trap for Load access fault

finally:
    li      a7, 10                  # Exit the program
    ecall

catch:                                          ecall   Issue a system call : Execute the system call specified by value in a7
    # Show message
    li      a7, 4
    la      a0, message
    ecall

    # Since uepc contains address of the error instruction
    # Need to load finally address to uepc
    la      t0, finally
    csrrw   zero, uepc, t0
    uret
```

b. Các giá trị thanh ghi và cách hoạt động

* Thay đổi của các thanh ghi:

- t0: Dùng để lưu trữ địa chỉ của các nhãn (như catch, finally). Nó sẽ được thay đổi khi thiết lập ngoại lệ và khi thay đổi uepc.
- a0: Dùng để lưu trữ địa chỉ chuỗi thông báo ngoại lệ (message) khi gọi ecall để in thông báo.
- a7: Dùng để chỉ thị hệ thống gọi (syscall). Trong trường hợp này, nó sẽ được đặt thành 4 để gọi lệnh in chuỗi và 10 để gọi lệnh thoát.
- utvec: Lưu trữ địa chỉ của hàm xử lý ngoại lệ. Sau khi được thiết lập, nếu có ngoại lệ xảy ra, chương trình sẽ chuyển đến địa chỉ này.

- ustatus: Lưu trữ các cờ trạng thái của hệ thống, trong đó có cờ cho phép ngắt và ngoại lệ
- uepc: Lưu trữ địa chỉ của lệnh sẽ được thực thi sau khi xử lý ngoại lệ.

* Cách hoạt động:

Thiết lập bắt ngoại lệ:

- la t0, catch: Lệnh này tải địa chỉ nhãn catch vào thanh ghi t0.
- csrrw zero, utvec, t0: Lệnh này ghi giá trị trong t0 (địa chỉ của nhãn catch) vào thanh ghi utvec, thiết lập địa chỉ của bộ xử lý ngoại lệ. utvec lưu trữ địa chỉ của hàm xử lý ngoại lệ.
- csrrsi zero, ustatus, 1: Lệnh này bật bit "interrupt enable" trong thanh ghi ustatus. Điều này cho phép các ngoại lệ có thể xảy ra (nghĩa là xử lý ngắt hoặc lỗi sẽ được thực thi khi xảy ra).

Gây ra ngoại lệ (Load access fault):

- lw zero, 0: Lệnh này cố gắng tải dữ liệu từ địa chỉ 0 vào thanh ghi zero. Tuy nhiên, địa chỉ này không hợp lệ, và sẽ gây ra một ngoại lệ (Load access fault). Ngoại lệ này sẽ khiến chương trình nhảy đến địa chỉ đã được thiết lập trong utvec, đó là nhãn catch.

Xử lý ngoại lệ (catch):

- li a7, 4: Lệnh này đặt giá trị 4 vào thanh ghi a7, chỉ thị cho hệ thống gọi hệ thống (ecall) để thực hiện việc in một chuỗi (tương ứng với lệnh "write" trong syscall).
- la a0, message: Tải địa chỉ của chuỗi thông báo ngoại lệ vào thanh ghi a0, để chuỗi này có thể được in ra.
- ecall: Lệnh này gọi hệ thống, thực hiện việc in thông báo ngoại lệ.

Cuối cùng (finally):

- la t0, finally: Tải địa chỉ của nhãn finally vào thanh ghi t0.
- csrrw zero, uepc, t0: Lệnh này thay đổi giá trị của thanh ghi uepc (User Exception Program Counter) thành địa chỉ của nhãn finally. uepc lưu trữ địa chỉ của lệnh sẽ được thực thi sau khi xử lý ngoại lệ. Địa chỉ này chính là nơi tiếp tục chương trình sau khi ngoại lệ được xử lý.

- uret: Lệnh này thực hiện việc quay lại chương trình sau khi xử lý ngoại lệ. Nó sẽ đưa chương trình quay lại tiếp tục thực thi từ địa chỉ được lưu trong uepc.

Thoát chương trình

Assignment 6

1. Yêu cầu

Assignment 6: Ngắt mềm

Ngắt mềm có thể được kích hoạt bằng cách thiết lập bit **USIP** (bit 0) của thanh ghi **uip** (trước đó cần cho phép ngắt mềm bằng cách thiết lập bit **USIE** của thanh ghi **uie**). Viết chương trình kích hoạt ngắt mềm nếu xảy ra tràn số khi thực hiện việc cộng 2 số nguyên có dấu (xem lại Bài thực hành 4), chương trình con xử lý ngắt sẽ in ra thông báo lỗi tràn số và kết thúc chương trình.

2. Thực hiện

a. Chương trình (đầu vào có tràn số)

```
.data
msg:    .asciz "Lỗi tràn số\n"

.text
.globl main

main:
    # nạp hai số nguyên vào t0 và t1
    li t0, 2147483647    # 0x7FFFFFFF
    li t1, 1

    # thực hiện phép cộng
    add t3, t0, t1        # t3 = t0 + t1

    # kiểm tra tràn số
    # nếu t0 > 0, t1 > 0 và t3 < 0 => tràn dương
    # nếu t0 < 0, t1 < 0 và t3 > 0 => tràn âm
    blt t0, x0, check_negative
    blt t1, x0, continue
    blt t3, x0, interrupt_handler
    j continue

check_negative:
    bge t1, x0, continue
    bgt t3, x0, interrupt_handler
    j continue

interrupt_handler:
    # trình xử lý "giả lập ngắt"
    la a0, msg
    li a7, 4              # syscall 4: print_string
    ecall


    # thoát chương trình
    li a7, 10             # syscall 10: exit
    ecall

continue:
    # không có lỗi -> kết thúc bình thường
    li a7, 10             # syscall 10: exit
    ecall
```

b. Kết quả

```
Lỗi tràn số

-- program is finished running (0) --
```

Kết luận

1. Yêu cầu

Trước khi kết thúc bài thực hành, sinh viên trả lời các câu hỏi sau:

- Kỹ thuật thăm dò là gì?
- Ngắt là gì?
- Chương trình con xử lý ngắt là gì?
- Ưu điểm của kỹ thuật thăm dò?
- Ưu điểm của kỹ thuật xử lý ngắt?
- Nêu điểm khác nhau giữa ngắt, ngoại lệ và traps?

2. Thực hiện

- Kỹ thuật thăm dò là gì?

➔ Là cách chương trình chủ động kiểm tra liên tục trạng thái thiết bị ngoại vi.

- Ngắt là gì?

➔ Là tín hiệu yêu cầu CPU tạm dừng công việc hiện tại để xử lý một sự kiện khẩn cấp.

- Chương trình con xử lý ngắt là gì?

➔ Là đoạn mã được gọi tự động để xử lý sự kiện ngắt.

- Ưu điểm của kỹ thuật thăm dò?

➔ Dễ lập trình, không cần phần cứng hỗ trợ ngắt.

- Ưu điểm của kỹ thuật xử lý ngắt?

➔ Hiệu quả, tiết kiệm tài nguyên CPU vì CPU chỉ xử lý khi có sự kiện.

- Điểm khác nhau giữa ngắt, ngoại lệ và traps?

➔ Ngắt: do thiết bị bên ngoài yêu cầu.

Ngoại lệ: do lỗi hoặc điều kiện đặc biệt bên trong CPU.

Trap: ngoại lệ có chủ ý, thường do chương trình yêu cầu dịch vụ hệ điều hành.