THỰC HÀNH KIẾN TRÚC MÁY TÍNH BUỔI 12

Họ và tên: Nguyễn Thành Duy

MSSV: 20235696

Assignment 4

Code:

```
.eqv IN_ADDRESS_HEXA_KEYBOARD 0xFFFF0012
.eqv TIMER_NOW 0xFFFF0018
.eqv TIMER_CMP 0xFFFF0020
.eqv MASK_CAUSE_TIMER 4
.eqv MASK_CAUSE_KEYPAD 8
.data
msg_keypad: .asciz "Someone has pressed a key!\n"
msg_timer: .asciz "Time inteval!\n"
# ---------------------------------------------------------------
# MAIN Procedure
# ---------------------------------------------------------------
.text
main:
la t0, handler
csrrs zero, utvec, t0
li t1, 0x100
csrrs zero, uie, t1 # uie - ueie bit (bit 8) - external interrupt
csrrsi zero, uie, 0x10 # uie - utie bit (bit 4) - timer interrupt
csrrsi zero, ustatus, 1 # ustatus - enable uie - global interrupt
# -----------------------------------------------------
# Enable interrupts you expect
# -----------------------------------------------------
# Enable the interrupt of keypad of Digital Lab Sim
li t1, IN_ADDRESS_HEXA_KEYBOARD
li t2, 0x80 # bit 7 of = 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
```

- Đoạn code lấy địa chỉ hàm handler (trình phục vụ ngắt), thiết lập địa chỉ xử lý ngắt vào thanh ghi utvec

```
la t0, handler
csrrs zero, utvec, t0
```

- Bật external interrupt (bit 8 = 0x100) trong thanh ghi uie. Bật **timer interrupt** (bit 4 = 0x10) trong thanh ghi uie.

```
li t1, 0x100
csrrs zero, uie, t1 # uie - ueie bit (bit 8) - external interrupt
```

- Gửi giá trị 0x80 vào địa chỉ keypad để bật ngắt thiết bị keypad (bit 7 = enable interrupt).

```
# Enable the interrupt of keypad of Digital Lab Sim
li t1, IN_ADDRESS_HEXA_KEYBOARD
li t2, 0x80 # bit 7 of = 1 to enable interrupt
sb t2, 0(t1)
```

- Đặt giá trị TIMER_CMP = 1000, tức là tạo mốc thời gian đầu tiên để timer sinh ngắt.

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

- Dành chỗ và lưu giá trị của các thanh ghi a0, a1, a2, a7 vào stack để bảo toàn trạng thái trước khi xử lý ngắt

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

- Đọc mã nguyên nhân ngắt từ `ucause`.

- Dùng `AND` với `0x7FFFFFFF` để xóa **interrupt bit** (bit 31), chỉ lấy cause code (0-30).

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

- So sánh `a1` với `MASK_CAUSE_TIMER` (4) và `MASK_CAUSE_KEYPAD` (8).
  - Nhảy đến `timer_isr` hoặc `keypad_isr` nếu phù hợp.
  - Nếu không khớp ngắt nào thì đi tới `end_process`.

```
li a2, MASK_CAUSE_TIMER
beq a1, a2, timer_isr
li a2, MASK_CAUSE_KEYPAD
beq a1, a2, keypad_isr
j end_process
```

- Phục hồi giá trị của các thanh ghi `a0`, `a1`, `a2`, `a7`.
  - Khôi phục lại con trỏ stack.
  - Trở về chương trình chính bằng `uret` (User return from interrupt).

```
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
```

Assignment 5

Code:

```
.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:
# 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
```

- Load địa chỉ của nhãn catch vào thanh ghi t0.

```
la t0, catch
csrrw zero, utvec, t0 # Set utvec to the handler address
```

- Bật bit 0 của `ustatus` (gọi là UIE – User Interrupt Enable).
    - Cho phép nhận ngắt và exception ở chế độ người dùng (user-mode).

```
csrrsi zero, ustatus, 1 # Set interrupt enable bit in ustatus
```

- Khi xảy ra trap, địa chỉ lệnh gây lỗi sẽ được lưu vào **uepc**.
    - Để tiếp tục thực thi chương trình (bỏ qua lệnh lỗi), ta gán lại `uepc` thành địa chỉ của `finally`.
    - `csrrw zero, uepc, t0`: ghi địa chỉ `finally` vào `uepc`

```
la t0, finally
csrrw zero, uepc, t0
uret
```

Assignment 6: Ngắt mềm

Code:

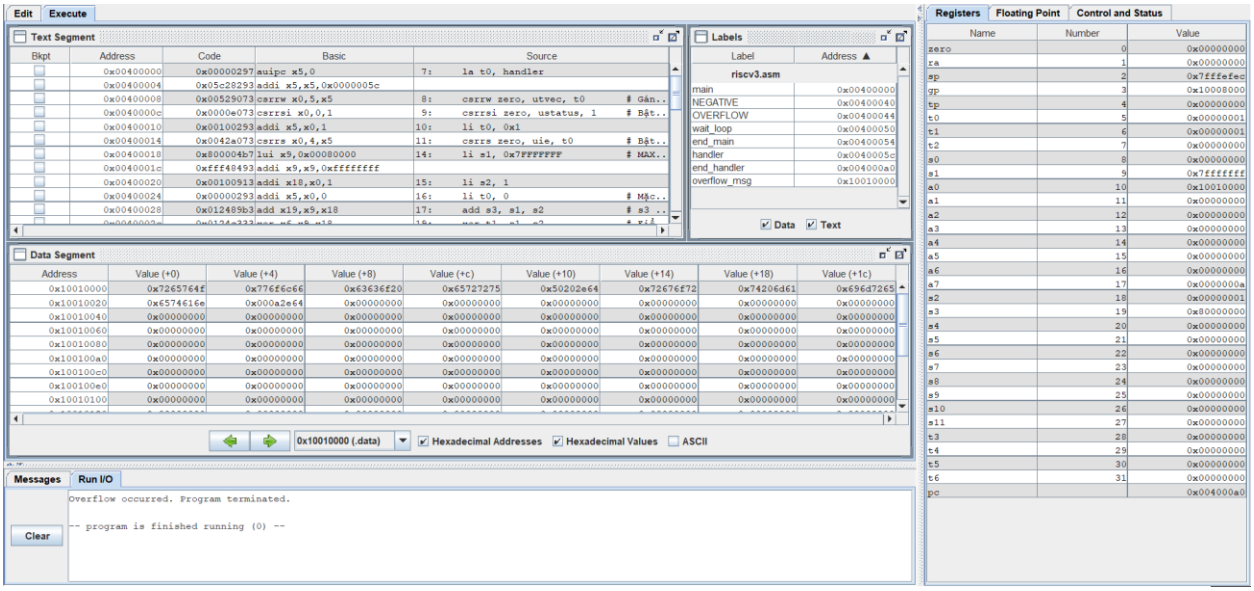
```
.data
overflow_msg: .asciz "Overflow occurred. Program terminated.\n"

.text
.globl main
main:
  # ----- Đặt địa chỉ trình xử lý ngắt (trap handler)
  la t0, handler
  csrrw zero, utvec, t0      # Gán utvec = handler
```

```
    csrrsi zero, ustatus, 1     # Bật global interrupt (UIE)
    li t0, 0x1
    csrrs zero, uie, t0         # Bật USIE (bit 0) trong uie → cho phép ngắt mềm

    # ----- Cộng hai số nguyên có dấu (có thể thay đổi giá trị để kiểm thử)
    li s1, 0x7FFFFFFF           # MAX_INT
    li s2, 1
    li t0, 0                    # Mặc định: không tràn
    add s3, s1, s2              # s3 = s1 + s2

    xor t1, s1, s2              # Kiểm tra dấu của s1 và s2
    blt t1, zero, end_main      # Nếu khác dấu → không thể tràn

    blt s1, zero, NEGATIVE      # Nếu s1 âm → kiểm tra theo nhánh âm
    bge s3, s1, end_main        # Nếu s3 >= s1 → không tràn
    j OVERFLOW

NEGATIVE:
    bge s1, s3, end_main        # Nếu s1 >= s3 → không tràn

OVERFLOW:
    li t0, 1                    # Đánh dấu tràn

    # Nếu t0 == 1 thì kích hoạt ngắt mềm
    li t1, 1
    csrrs zero, uip, t1         # Thiết lập USIP = 1

wait_loop:
    j wait_loop                 # Đợi ngắt được xử lý

end_main:
    li a7, 10                   # Không tràn → thoát chương trình
    ecall

# ----------------------------
# Interrupt/trap handler
# ----------------------------
handler:
    # Lưu context
    addi sp, sp, -16
    sw a0, 0(sp)
```

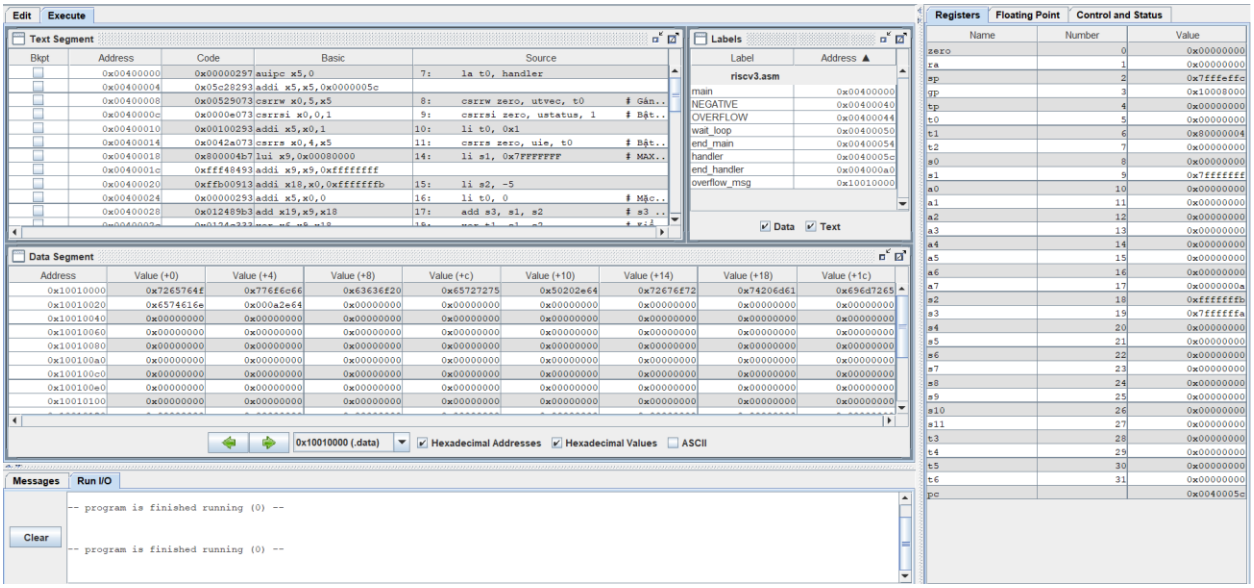
```
    sw a1, 4(sp)
    sw a7, 8(sp)
    sw t0, 12(sp)

    # Xác định nguyên nhân trap
    csrr a0, ucause
    li a1, 0x7FFFFFFF
    and a0, a0, a1          # Xóa bit MSB (interrupt flag)
    li a1, 0              # Mã ngắt mềm = 0 (USIP)
    bne a0, a1, end_handler    # Nếu không phải ngắt mềm → bỏ qua

    # ----- In thông báo lỗi
    li a7, 4
    la a0, overflow_msg
    ecall

    # ----- Thoát chương trình
    li a7, 10
    ecall

end_handler:
    # Khôi phục context
    lw a0, 0(sp)
    lw a1, 4(sp)
    lw a7, 8(sp)
    lw t0, 12(sp)
    addi sp, sp, 16
    uret
```

Kết quả:

Với s1 = 0x7FFFFFFF, s2 = 1 => s1 + s2 sẽ bị tràn số



Với s1 = 0x7FFFFFFF, s2 = -5 => s1 + s2 không tràn số nên không in ra thông báo và kết thúc chương trình



Giải thích:

- Kiểm tra tổng 2 số

+) Nếu như không xảy ra hiện tượng tràn số thì chương trình tiếp tục thực hiện các lệnh tiếp theo

+) Khi phát hiện tràn → **kích hoạt ngắt mềm** → in thông báo và kết thúc chương trình

Bài tập bổ sung:

Code:

```
.eqv IN_ADDRESS_HEXA_KEYBOARD      0xFFFF0012  # địa chỉ truy cập thanh ghi làm
việc với ngắt bàn phím
.eqv OUT_ADDRESS_HEXA_KEYBOARD     0xFFFF0014
.eqv TIMER_NOW                0xFFFF0018  # địa chỉ truy cập thanh ghi lưu, đọc thời gian
hiện tại
.eqv TIMER_CMP                0xFFFF0020  # địa chỉ truy cập thanh ghi so sánh
.eqv MASK_CAUSE_TIMER           4         # mã nguyên nhân ngắt do bộ định thời
.eqv MASK_CAUSE_KEYPAD          8         # mã nguyên nhân ngắt do bàn phím
.eqv SEVENSEG_LEFT            0xFFFF0011
.eqv SEVENSEG_RIGHT            0xFFFF0010  # Gán địa chỉ của 2 đèn LED

.data
  increase_msg:   .asciz "Đếm tăng dần.\n"
  decrease_msg:   .asciz "Đếm giảm dần.\n"
  slow_down_msg:  .asciz "Giảm tốc. "
  speed_up_msg:   .asciz "Tăng tốc. "
  speed_info: .asciz "Thời gian 1 chu kỳ hiện tại là: "
  msg1:      .asciz " ms.\n"
  led:       .word 0x3F 0x06 0x5B 0x4F 0x66 0x6D 0x7D 0x07 0x7F 0x6F  # Mảng mã của số
hiện trên đèn LED từ 0-9
  count:     .word 0      # nơi lưu biến đếm hiện tại
  cycle1:     .word 1000    # nơi lưu chu kỳ bộ định thời
  direction:  .word 0      # nơi lưu hướng đếm hiện tại, 0 là tăng, 1 là giảm
.text
main:
  la     t0, handler
  csrrs   zero, utvec, t0   # lấy địa chỉ của hàm xử lý ngắt

  li     t1, 0x100
  csrrs   zero, uie, t1    # bật bit 8 của uie - ueie cho phép xử lý ngắt ngoài
  csrrsi  zero, uie, 0x10   # bật bit 4 của uie - utie cho phép xử lý ngắt thời gian

  csrrsi  zero, ustatus, 1  # bật bit 0 của ustatus cho phép xử lý ngắt ở User Mode

  li     t1, IN_ADDRESS_HEXA_KEYBOARD
  li     t2, 0x80
  sb     t2, 0(t1)       # bật ngắt của bàn phím

  li     t1, TIMER_CMP
  li     t2, 1000
  sw     t2, 0(t1)       # đặt thời gian ngắt mặc định là 1s
```

```
loop:
  nop
  nop
  nop
  j     loop          # Vòng lặp vô hạn

#--------------Trình xử lý ngắt-------------------
handler:
  csrr   a1, ucause
  li     a2, 0x7FFFFFFF
  and    a1, a1, a2      # loại bỏ bit 31

  li     a2, MASK_CAUSE_TIMER    # nếu là bit 2, tức kq a1 = 4 thì là ngắt thời gian
  beq    a1, a2, timer_isr
  li     a2, MASK_CAUSE_KEYPAD   # nếu là bit 3, tức kq a1 = 8 thì là ngắt ngoài do bàn
phím
  beq    a1, a2, keypad_isr
  j      end_handler          # nếu không phải 2 loại ngắt này thì không xử lý

#----Xử lý ngắt thời gian, mỗi chu kỳ cập nhật lại LED 1 lần----------
timer_isr:
  la     t1, count
  lw     t2, 0(t1)          # đọc giá trị count từ bộ nhớ

  li     a7, 1
  mv     a0, t2
  ecall                  # in count hiện tại ra màn hình

  li     a7, 11
  li     a0, 32            # thêm dấu cách giữa các số
  ecall

  li     t1, 10
  div    t3, t2, t1         # lấy chữ số hàng chục
  rem    t4, t2, t1         # lấy chữ số hàng đơn vị

  la     t1, led
  slli   t3, t3, 2          # t3 = 4*t3
  add    t3, t3, t1         # địa chỉ của mã trong mảng
  lw     t5, 0(t3)          # Lấy mã LED của số hàng chục từ mảng led
  li     t6, SEVENSEG_LEFT
  sb     t5, 0(t6)          # hiện đèn bên trái

  slli   t4, t4, 2
```

```
   add    t4, t4, t1          # địa chỉ của mã trong mảng
   lw     t5, 0(t4)           # Lấy mã LED của số đơn vị từ mảng led
   li     t6, SEVENSEG_RIGHT
   sb     t5, 0(t6)           # hiện đèn bên phải

   la     t1, direction
   lw     t3, 0(t1)           # đọc hướng đếm hiện tại

   beq    t3, zero, increase_count    # hàm direction = 0, đang đếm tăng thì nhảy hàm
tương ứng

decrease_count:
   addi   t2, t2, -1              # direction khác 0 là bằng 1 thì giảm biến đếm
   li     t4, 0
   bge    t2, t4, update_count       # nếu count hiện tại >= 0 thì cập nhật luôn
   li     t2, 99                  # nếu count < 0 thì cần đặt lại là 99 sau đó mới cập nhật
   j      update_count

increase_count:
   addi   t2, t2, 1               # tăng biến đếm
   li     t4, 99
   ble    t2, t4, update_count       # nếu count hiện tại <=99 thì cập nhật luôn
   li     t2, 0                   # nếu đã lớn hơn 99 thì đặt về 0 và cập nhật
   j      update_count            # dòng hơi thừa nhưng nhét vào cho cân xứng :))

update_count:
   la     t1, count
   sw     t2, 0(t1)               # cập nhật lại count vào bộ nhớ

   # Cập nhật chu kỳ của bộ định thời
   li     t1, TIMER_NOW
   lw     t2, 0(t1)
   la     t3, cycle1
   lw     t4, 0(t3)               # lấy tốc độ hiện tại
   add    t2, t2, t4              # đặt thời điểm ngắt kế tiếp
   li     t1, TIMER_CMP
   sw     t2, 0(t1)

   j      end_handler

#------Xử lý ngắt từ bàn phím, khi nhấn các phím 0, 1, 2, 3-----------
keypad_isr:
   li     t1, IN_ADDRESS_HEXA_KEYBOARD
   li     t2, 0x81
```

```
    sb     t2, 0(t1)
    li     t1, OUT_ADDRESS_HEXA_KEYBOARD
    lb     t2, 0(t1)               # đọc mã của phím được nhấn

    # Kiểm tra phím được nhấn và thực hiện hành động tương ứng
    li     t3, 0x11                # Phím 0: Tăng dần
    beq    t2, t3, increase

    li     t3, 0x21                # Phím 1: Giảm dần
    beq    t2, t3, decrease

    li     t3, 0x41                # Phím 2: Giảm tốc độ (tăng chu kỳ)
    beq    t2, t3, slow_down

    li     t3, 0xffffff81              # Phím 3: Tăng tốc độ (giảm chu kỳ)
    beq    t2, t3, speed_up

    j      end_handler

increase:
    la     t1, direction
    li     t2, 0
    sw     t2, 0(t1)           # đặt giá trị direction về 0
    li     a7, 4
    la     a0, increase_msg
    ecall                      # in ra thông báo
    j      end_handler
decrease:
    la     t1, direction
    li     t2, 1
    sw     t2, 0(t1)           # đặt giá trị direction về 1
    li     a7, 4
    la     a0, decrease_msg         # in ra thông báo
    ecall
    j      end_handler
slow_down:
    la     t1, cycle1
    lw     t2, 0(t1)           # Load chu kỳ hiện tại
    li     t3, 2
    mul    t2, t2, t3           # Tăng thời gian chu kỳ gấp đôi
    sw     t2, 0(t1)           # lưu lại vào bộ nhớ

    li     a7, 4
    la     a0, slow_down_msg         # In ra thông báo
```

```
  ecall

  li    a7, 4
  la    a0, speed_info
  ecall
  li    a7, 1
  mv    a0, t2
  ecall
  li    a7, 4
  la    a0, msg1
  ecall                 # Thông báo chu kỳ hiện tại
  j     end_handler
speed_up:
  la    t1, cycle1
  lw    t2, 0(t1)           # Load chu kỳ hiện tại
  li    t3, 2
  div   t2, t2, t3          # Giảm thời gian chu kỳ một nửa
  sw    t2, 0(t1)           # lưu lại vào bộ nhớ

  li    a7, 4
  la    a0, speed_up_msg        # In ra thông báo
  ecall

  li    a7, 4
  la    a0, speed_info
  ecall
  li    a7, 1
  mv    a0, t2
  ecall
  li    a7, 4
  la    a0, msg1
  ecall                 # Thông báo chu kỳ hiện tại
  j     end_handler         # Thêm vô cho đều chứ cũng không cần lắm
end_handler:
  uret
```

- **Ý tưởng:**
  - Thiết lập ngắt sau:
    - Ngắt từ bộ định thời, 1 giây ngắt 1 lần.
    - Ngắt từ bàn phím.
  - Kiểm tra dữ liệu lấy được từ bàn phím để điều hướng tới hàm con để xử lý.
  - Cách để 2 đèn LED chạy từ 00 đến 99. Khai báo ở .data một mảng chứa mã đèn LED của các số từ 0 – 9, sử dụng vòng lặp lấy biến đếm của vòng lặp làm chữ số hiển thị trên đèn LED.

- Xử lý phím được nhấn:

+ Nếu phím được nhấn là 0, biến đếm i = i + 1 sau mỗi vòng lặp. Nếu i = 100 đặt lại i = 0.

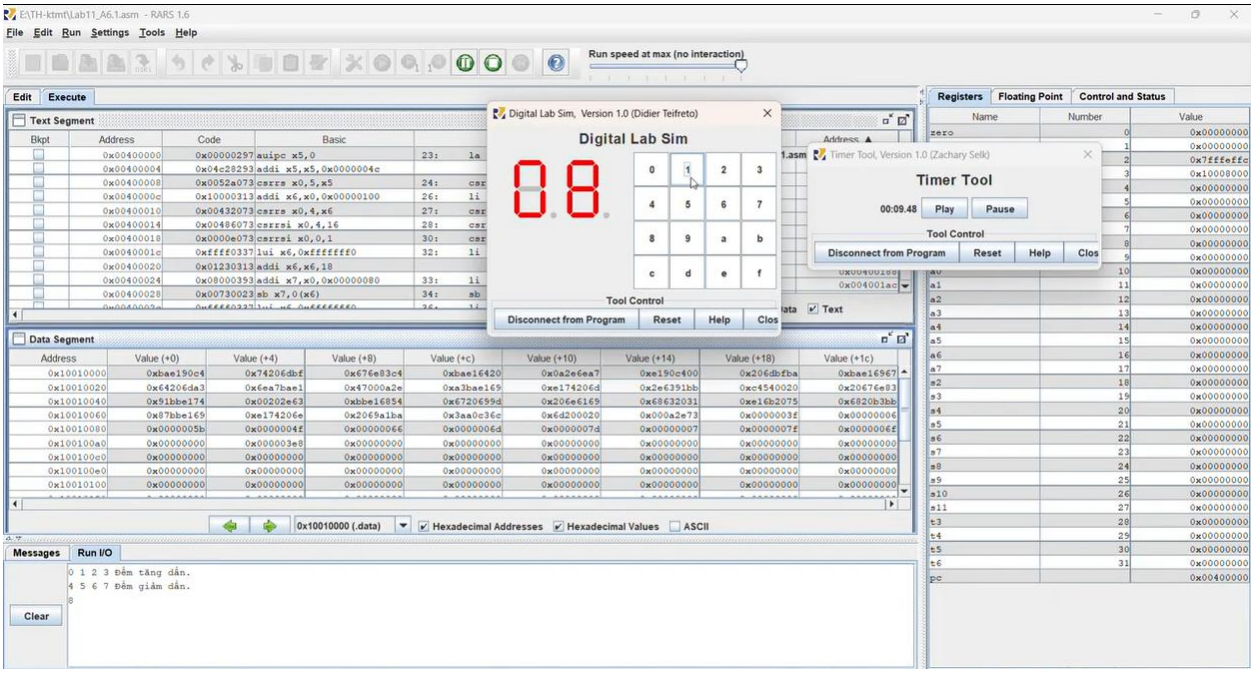
+ Nếu phím được nhấn là 1, biến đếm i = i - 1 sau mỗi vòng lặp. Nếu i = 0 thì đặt lại i = 99. + Nếu phím được nhấn là 2, tăng chu kỳ tại bộ định thời bằng cách tăng thời gian lên gấp đôi, tốc độ nhảy số sẽ giảm đi.

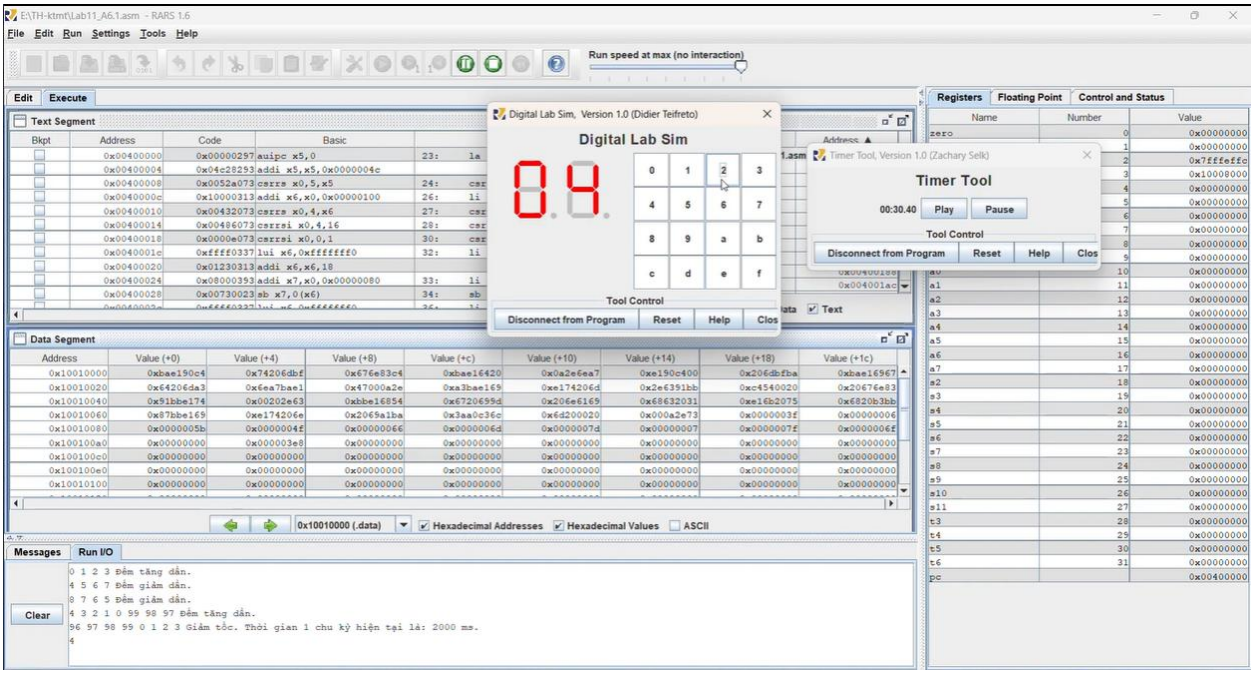
+ Nếu phím được nhấn là 3, giảm chu kỳ tại bộ định thời bằng cách giảm thời gian đi 1 nửa, tốc độ nhảy số sẽ tăng lên.

+ Để dễ dàng quan sát thông qua báo cáo, mỗi xử lý với phím được đưa ra 1 thông báo tương ứng, và mỗi chu kỳ sẽ in ra số trên đèn LED.

- Kết quả:

```
0 1 2 3 Đếm tăng dần.
4 5 6 7 Đếm giảm dần.
8 7 6 5 Đếm giảm dần.
4 3 2 1 0 99 98 97 Đếm tăng dần.
96 97 98 99 0 1 2 3 Giảm tốc. Thời gian 1 chu kỳ hiện tại là: 2000 ms.
4 5 6 Giảm tốc. Thời gian 1 chu kỳ hiện tại là: 4000 ms.
7 8 Tăng tốc. Thời gian 1 chu kỳ hiện tại là: 2000 ms.
9 10 11 Tăng tốc. Thời gian 1 chu kỳ hiện tại là: 1000 ms.
12 13 Tăng tốc. Thời gian 1 chu kỳ hiện tại là: 500 ms.
14 15 16 17 18 Tăng tốc. Thời gian 1 chu kỳ hiện tại là: 250 ms.
19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 Đếm tăng dần.
34 35 36 37 38 39 40 41 42 Đếm giảm dần.
43 42 41 40 39 38 37 36 35 34 33 32 31 Giảm tốc. Thời gian 1 chu kỳ hiện tại là: 500 ms.
30 29 28 27 26 25 24 23 22 21 20 19 Tăng tốc. Thời gian 1 chu kỳ hiện tại là: 250 ms.
18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83
```




E:\TH-ktmt\Lab11_A6.1.asm - RARS 1.6
File Edit Run Settings Tools Help
Run speed at max (no interaction)
Edit Execute
Text Segment
Bkpt Address Code Basic
Digital Lab Sim, Version 1.0 (Didier Teifreto)
Digital Lab Sim
Tool Control
Disconnect from Program Reset Help Clos
Timer Tool, Version 1.0 (Zachary Selk)
Timer Tool
00:30.40 Play Pause
Tool Control
Disconnect from Program Reset Help Clos
Data Segment
Address Value (+0) Value (+4) Value (+8) Value (+c) Value (+10) Value (+14) Value (+18) Value (+1c)
0x10010000 0xbae190c4 0x74206dbf 0x676e83c4 0xbae16420 0x0a2e6ea7 0xe190c400 0x206dbfba 0xbae16967
0x10010020 0x64206da3 0x6ea7bae1 0x47000a2e 0xa3bae169 0xe174206d 0x2e6391bb 0xc4540020 0x20676e83
0x10010040 0x91bbe174 0x00202e63 0xbbe16854 0x6720699d 0x206e6169 0x68632031 0xe16b2075 0x6820b3bb
0x10010060 0x87bbe169 0xe174206e 0x2069a1ba 0x3aa0c36c 0x6d200020 0x000a2e73 0x0000003f 0x00000064
0x10010080 0x0000005b 0x0000004f 0x00000066 0x0000006d 0x0000007d 0x00000007 0x0000007f 0x0000006f
0x100100a0 0x00000000 0x000003e8 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x100100c0 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x100100e0 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010100 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010000 (.data) Hexadecimal Addresses Hexadecimal Values ASCII
Messages Run I/O
Clear
0 1 2 3 Đếm tăng dần.
4 5 6 7 Đếm giảm dần.
8 7 6 5 Đếm giảm dần.
4 3 2 1 0 99 98 97 Đếm tăng dần.
96 97 98 99 0 1 2 3 Giảm tốc. Thời gian 1 chu kỳ hiện tại là: 2000 ms.
4
Registers Floating Point Control and Status
Name Number Value
zero 0 0x00000000
pc 0x00400000