ĐẠI HỌC BÁCH KHOA HÀ NỘI

TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

---📖---

BÁO CÁO CUỐI KỲ

THỰC HÀNH KIẾN TRÚC MÁY TÍNH

Giảng viên hướng dẫn: Ngô Lam Trung

Sinh viên thực hiện: Nguyễn Việt Anh 20235651

Nguyễn Thị Khánh Vân 20235869

Nhóm: 03 – Đề tài: 8, 18

Lớp: 156791 - IT3280

Mục lục

8. Máy tính bỏ túi

Sử dụng 2 ngoại vi là bàn phím keypad và led 7 thanh để xây dựng một máy tính bỏ túi đơn giản. Hỗ trợ các phép toán +, -, *, /, % với các toán hạng là số nguyên. Do trên bàn phím không có các phím trên nên sẽ dùng các phím:

- Bấm phím a để nhập phép tính +
- Bấm phím b để nhập phép tính –
- Bấm phím c để nhập phép tính *
- Bấm phím d để nhập phép tính /
- Bấm phím e để nhập phép tính %
- Bấm phím f để nhập phép =

Yêu cầu cụ thể như sau:

- Khi nhấn các phím số, hiển thị lên LED, do chỉ có 2 LED nên chỉ hiển thị 2 số cuối cùng. Ví dụ khi nhấn phím 1 → hiển thị 01. Khi nhấn thêm phím 2 → hiển thị 12. Khi nhấn thêm phím 3 → hiển thị 23.
- Sau khi nhập số, sẽ nhập phép tính + - * / %
- Sau khi nhấn phím f (dấu =), tính toán và hiển thị kết quả lên LED.
- Có thể thực hiện các phép tính liên tiếp (tham khảo ứng dụng Calculator trên hệ điều hành Windows)

1. Mã nguồn

```
.eqv SEVENSEG_RIGHT    0xFFFF0010        # địa chỉ của led 7 đoạn phải

.eqv SEVENSEG_LEFT     0xFFFF0011        # địa chỉ của led 7 đoạn trái
.eqv IN_ADDRESS_HEXA_KEYBOARD   0xFFFF0012      # địa chỉ cổng vào của bàn
phím -> quét dòng
.eqv OUT_ADDRESS_HEXA_KEYBOARD  0xFFFF0014  # địa chỉ cổng ra của bàn phím ->
đọc cột
.eqv MASK_CAUSE_KEYPAD          8       # nguyên nhân ngắt do nhấn phím

.data
    A: .byte 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F    #
Chứa mã hiển thị số từ 0-9 cho LED 7 đoạn
    error: .asciz "\nERROR! Không thể chia cho 0 (˘·_·˘)\nHãy tính lại phép
tính khác\n"
.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, ustatus, 1        # ustatus - enable uie - global interrupt

    li      t1, IN_ADDRESS_HEXA_KEYBOARD
    li      t2, 0x80        # bit 7 of = 1 to enable interrupt
    sb      t2, 0(t1)

    li s1, 0            # chứa số hiện tại đang nhập
    li s2, 10           # hệ số nhân -> để tìm số nhấn
    li s0, 0            # thanh ghi chứa kết quả cuối cùng

    li t3, 0            # thanh ghi để kiểm tra các điều kiện
    li t4, 0            # chứa các số từ bàn phím để kiểm tra
    li s4, 0            # check số đầu tiên
    li s5, 0            # dấu = -> kiểm tra nếu như không tính tiếp thì reset
lại các giá trị

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_KEYPAD
    beq     a1, a2, keypad_isr
    j       end_process

keypad_isr:

    li t1, IN_ADDRESS_HEXA_KEYBOARD
    li t2, 0x81                 # Check row 1 and re-enable bit 7
    sb t2, 0(t1)                # Must reassign expected row
```

```
li t1, OUT_ADDRESS_HEXA_KEYBOARD
lb a0, 0(t1)

li t1, 0x00000011           # kiem tra so 0
li t4, 0
beq a0, t1, capnhat

li t1, 0x00000021           # kiem tra so 1
li t4, 1
beq a0, t1, capnhat

li t1, 0x00000041           # kiem tra so 2
li t4, 2
beq a0, t1, capnhat

li t1, 0xFFFFFF81           # kiem tra so 3
li t4, 3
beq a0, t1, capnhat

li t1, IN_ADDRESS_HEXA_KEYBOARD
li t2, 0x82                 # Check row 2 and re-enable bit 7
sb t2, 0(t1)                # Must reassign expected row
li t1, OUT_ADDRESS_HEXA_KEYBOARD
lb a0, 0(t1)

li t1, 0x00000012           # kiem tra so 4
li t4, 4
beq a0, t1, capnhat

li t1, 0x00000022           # kiem tra so 5
li t4, 5
beq a0, t1, capnhat

li t1, 0x00000042           # kiem tra so 6
li t4, 6
beq a0, t1, capnhat

li t1, 0xFFFFFF82
li t4, 7
beq a0, t1, capnhat         # kiem tra so 7

li t1, IN_ADDRESS_HEXA_KEYBOARD
li t2, 0x84                 # Check row 2 and re-enable bit 7
sb t2, 0(t1)                # Must reassign expected row
li t1, OUT_ADDRESS_HEXA_KEYBOARD
lb a0, 0(t1)

li t1, 0x00000014           # kiem tra so 8
```

```
    li t4, 8
    beq a0, t1, capnhat

    li t1, 0x00000024           # kiem tra so 9
    li t4, 9
    beq a0, t1, capnhat

    li t1, 0x00000044           # kiem tra a (cong +)
    li t4, 10
    beq a0, t1, phep_tinh

    li t1, 0xFFFFFF84
    li t4, 11
    beq a0, t1, phep_tinh           # kiem tra b (tru -)

    li t1, IN_ADDRESS_HEXA_KEYBOARD
    li t2, 0x88                 # Check row 2 and re-enable bit 7
    sb t2, 0(t1)                # Must reassign expected row
    li t1, OUT_ADDRESS_HEXA_KEYBOARD
    lb a0, 0(t1)

    li t1, 0x00000018           # kiem tra c (nhan *)
    li t4, 12
    beq a0, t1, phep_tinh

    li t1, 0x00000028           # kiem tra d (chia /)
    li t4, 13
    beq a0, t1, phep_tinh

    li t1, 0x00000048           # kiem tra e (lay du %)
    li t4, 14
    beq a0, t1, phep_tinh

    li t1, 0xFFFFFF88
    li t4, 15
    beq a0, t1, phep_bang       # kiem tra f (bang =)
    j end_process
#-------------------------------------------------------------------------
----
# Nhập số
# Nhấn các phím 0 – 9, giá trị sẽ được xây dựng dần và hiển thị trên led 7
thanh
# Nếu nhập tiếp tục, chỉ hai chữ số sau cùng được hiển thị
#-------------------------------------------------------------------------
----
capnhat:
    bnez s5, reset_all      # Nếu mới nhấn = xong và nhập số mới → tự động
reset mọi biến, bắt đầu phép tính mới
```

```
new:
    mul s1, s1, s2          # nhân số cũ với 10
    add s1, s1, t4          # cộng tiếp với số vừa nhấn
    addi s3, s1, 0          # ta được kết quả là số mà mình muốn nhấn (vd:
1234)

    j hienthi           # hiển thị giá trị số nhập vào

#-------------------------------------------------------------------------
------------------------
# Nhập phép toán
# Sau khi nhập số, bấm phím a đến e để chọn phép toán (+, -, *, /, %)
# Nếu đã có phép toán trước, chương trình sẽ tự động tính phép toán trước đó
trước khi cập nhật phép toán mới
#-------------------------------------------------------------------------
------------------------
phep_tinh:
    li a7, 1            # in ra màn hình Run I/O giá trị của số vừa nhập
    mv a0, s1
    ecall

    li s5, 0            # reset s5 khi bắt đầu phép tính mới

    bnez s4, calc_before        # kiểm tra xem số vừa nhấn có phải là số đầu
tiên không
    mv s0, s1           # nếu là số đầu tiên thì gán cho s0 = s1

    j update_op

# xử lý phép toán trước đó trước khi cập nhật toán tử mới
calc_before:
    li t3, 10           # kiểm tra dấu trước đó là dấu +
    beq a4, t3, calc_add

    li t3, 11           # kiểm tra dấu trước đó là dấu -
    beq a4, t3, calc_sub

    li t3, 12           # kiểm tra dấu trước đó là dấu *
    beq a4, t3, calc_mul

    li t3, 13           # kiểm tra dấu trước đó là dấu /
    beq a4, t3, calc_div

    li t3, 14           # kiểm tra dấu trước đó là dấu %
    beq a4, t3, calc_rem

    j update_op
```

```
calc_add:
    add s0, s0, s1          # tính kết quả nếu trước đó là dấu +

    j update_op

calc_sub:
    sub s0, s0, s1

    j update_op

calc_mul:
    mul s0, s0, s1

    j update_op

calc_div:
    beqz s1, chia0_error        # lỗi chia cho 0
    div s0, s0, s1

    j update_op

calc_rem:
    beqz s1, chia0_error        # lỗi chia cho 0
    rem s0, s0, s1

update_op:
    # In ký hiệu toán tử vừa nhấn
    li a7, 11

    li t3, 10
    beq t4, t3, print_cong

    li t3, 11
    beq t4, t3, print_tru

    li t3, 12
    beq t4, t3, print_nhan

    li t3, 13
    beq t4, t3, print_chia

    li t3, 14
    beq t4, t3, print_du

    j end_process

print_cong:
```

```
    li a0, '+'          # in ra dấu vừa nhấn
    ecall

    j set_op
print_tru:
    li a0, '-'
    ecall

    j set_op
print_nhan:
    li a0, '*'
    ecall

    j set_op
print_chia:
    li a0, '/'
    ecall

    j set_op
print_du:
    li a0, '%'
    ecall
set_op:
    li s4, 1                # Đã có số đầu tiên
    li s1, 0                # Reset số đang nhập
    mv a4, t4               # Lưu lại phép toán mới (lưu lại dấu cuối cùng đã
nhấn)

    j end_process

#---------------------------------------------------------------------------
----------
# Nhấn '='
# Thực hiện phép toán cuối cùng và hiển thị kết quả
# Cho phép thực hiện tiếp phép tính mới dựa trên kết quả cũ
#---------------------------------------------------------------------------
----------
phep_bang:
    li s5, 1            # vừa mới nhấn '='

    li t3, 10           # kiểm tra dấu của phép trước đó là dấu +
    beq a4, t3, cong

    li t3, 11
    beq a4, t3, tru

    li t3, 12
    beq a4, t3, nhan
```

```
    li t3, 13
    beq a4, t3, chia

    li t3, 14
    beq a4, t3, du

    # nếu như không thực hiện phép tính nào mà nhấn bằng luôn
    li a7, 1            # in ra số đang nhập
    mv a0, s1
    ecall

    mv s0, s1           # thanh ghi kết quả được gán bằng số đang nhấn
    j ketqua            # nhảy đến đây để hiển thị kết quả
cong:
    add s0, s0, s1          # thực hiện phép toán cuối cùng trước khi hiển thị
kết quả

    li a7, 1            # in ra phần tử số cuối cùng
    mv a0, s1
    ecall

    li s4, 0
    j ketqua
tru:
    sub s0, s0, s1

    li a7, 1
    mv a0, s1
    ecall

    li s4, 0
    j ketqua
nhan:
    mul s0, s0, s1

    li a7, 1
    mv a0, s1
    ecall

    li s4, 0
    j ketqua

chia:
    beqz s1, chia0_error        # lỗi chia cho 0

    div s0, s0, s1
```

```
    li a7, 1
    mv a0, s1
    ecall

    li s4, 0
    j ketqua

du:
    beqz s1, chia0_error        # lỗi chia cho 0

    rem s0, s0, s1

    li a7, 1
    mv a0, s1
    ecall

ketqua:
    li a7, 11
    li a0, '='
    ecall

    li a7, 1            # in ra kết quả vừa tính
    mv a0, s0
    ecall

    li a7, 11
    li a0, '\n'
    ecall

    mv s3, s0           # hiển thị kết quả khi nhấn =

    mv s1, s0           # gán s1 = s0 để tự động lấy kết quả trước làm toán
hạng đầu mà không cần phải nhấn lại
    li a4, 0            # reset thanh ghi chứa dấu của phép toán gần nhất

#-------------------------------------------------------------------------
--------------------------------
# Hiển thị
# Nếu kết quả dương: hiển thị 2 chữ số cuối cùng
# Nếu kết quả âm: + Nếu số > -10: LED trái hiển thị dấu -, LED phải hiển thị
số
#         + Nếu số <= -10: chuyển thành số dương rồi hiển thị 2 số cuối
#-------------------------------------------------------------------------
--------------------------------
hienthi:
    la a3, A            # lấy địa chỉ đầu mảng A

    bge s3, zero, hienthisoduong
```

```
    sub s3, zero, s3        # nếu số âm <= - 10 thì biến thành số dương để
hiển thị 2 số cuối

    blt s3, s2, hienthisoam     # nếu như số âm lớn hơn -10 thì nhảy đến hàm
này để khi in số âm có cả dấu -

hienthisoduong:
    li t3, 100
    rem s3, s3, t3          # t4 = s3 % 100

    li t3, 10

    div t5, s3, t3
        add t5, t5, a3              # Địa chỉ của A[t5]
        lb a0, 0(t5)                # set value for segments
        jal SHOW_7SEG_LEFT          # show

    rem t6, s3, t3
        add t6, t6, a3              # Địa chỉ của A[t6]
        lb a0, 0(t6)                # set value for segments
        jal SHOW_7SEG_RIGHT         # show

    j end_process

hienthisoam:
    li t3, 10

    li a0, 0x40         # hiển thị dấu -
        jal SHOW_7SEG_LEFT          # show

        rem t6, s3, t3
        add t6, t6, a3              # Địa chỉ của A[t6]
        lb a0, 0(t6)                # set value for segments
        jal SHOW_7SEG_RIGHT         # show

        j end_process

SHOW_7SEG_LEFT:
        li t0, SEVENSEG_LEFT        # assign port's address
        sb a0, 0(t0)                # assign new value
        jr ra

SHOW_7SEG_RIGHT:
        li t0, SEVENSEG_RIGHT       # assign port's address
        sb a0, 0(t0)                # assign new value
        jr ra
```

```
reset_all:
    li s1, 0            # reset số đang nhập
    li s0, 0            # reset kết quả
    li s4, 0            # đánh dấu chưa có số đầu tiên
    li s5, 0            # reset trạng thái '='

    j new               # nhảy về chỗ xử lý số

#---------------------------------------------------------------------
# Xử lý chia cho 0
# Nếu chia cho 0, in ra thông báo lỗi và reset lại
#---------------------------------------------------------------------
chia0_error:
    li a7, 1
    mv a0, s1
    ecall

    li a7, 4            # in ra thông báo lỗi
    la a0, error
    ecall

    li s0, 0            # reset kết quả về 0
    li s4, 0            # bắt đầu nhập lại số khác

end_process:
    lw      a7, 12(sp)
    lw      a2, 8(sp)
    lw      a1, 4(sp)
    lw      a0, 0(sp)
    addi    sp, sp, 16
    uret                # Quay lại chương trình chính
```

2. Ý tưởng

- Sử dụng ngắt từ bàn phím keypad interrupt:
  - Khi nhấn phím bất kỳ trên keypad, xảy ra ngắt ngoài
  - Trong quá trình xảy ra ngắt, mã phím được đọc
    - Các phím số 0-9: cập nhật các số đang nhập
    - Các phím từ a-e: tính các phép toán +, -, *, /, %
    - Phím f: thực hiện phép tính và hiển thị kết quả
- Xử lý quá trình nhập số liên tục:
  - Các số được nhập vào bằng cách nhấn liên tiếp các phím số
  - Sau mỗi lần nhấn số mới thì cần nhân 10 với số đang nhập và cộng thêm số mới

    - Để hiển thị trên led 7 thanh cần mod 100 để lọc ra 2 số cuối của số nhập vào
    - Thanh ghi s1 được sử dụng để lưu các số đang nhập
- Xử lý lưu và tính toán các phép tính:
    - Thanh ghi s0 dùng để lưu kết quả, nếu như chưa được tính toán thì thanh ghi s0 chứa giá trị 0
    - Khi chưa có số nào được nhập vào trước đó thì s0 sẽ được gán bằng giá trị số nhập vào đầu tiên
    - Chương trình thực hiện việc tính toán liên tục ngay khi có toán tử được nhấn
    - Thanh ghi a4 để lưu toán tử gần nhất được nhấn
    - Khi chuẩn bị nhập một số mới thì s1 cập nhật lại s1 = 0
- Tính kết quả khi nhấn "=":
    - Dựa vào giá trị được lưu trong thanh ghi a4 để tính kết quả của s0 và s1 rồi lưu vào thanh ghi s0 chứa kết quả cuối cùng. Nếu như không tính toán mà nhấn "=" thì in ra kết quả luôn (ví dụ 4=4)
    - Hiển thị 2 số cuối của kết quả trên led 7 thanh
        - Với kết quả là số nguyên dương: thực hiện mod 100 để lọc 2 số cuối
        - Nếu kết quả là số âm: Nếu s0 ≤ -10 thì lấy trị tuyệt đối của nó để thành số âm và hiển thị 2 số cuối như là một số nguyên dương. Ngược lại, -9 ≤ s0 ≤ -1 thì trên led 7 thanh đoạn trái sẽ hiện dấu "-" và bên phải là số của kết quả
- Xử lý việc nhập phép tính sau khi nhấn dấu "=":
    - Nếu như muốn thực hiện phép tính liên tiếp giống calculator, gán giá trị thanh ghi s1 bằng kết quả s0 để tự động lấy kết quả trước làm toán hạng đầu mà không cần phải nhấn lại
    - Ngược lại, sau khi hiện kết quả khi nhấn "=", nếu không muốn tính tiếp từ kết quả trước đó mà nhập luôn số mới thì các thanh ghi sẽ được reset lại toàn bộ để tính toán phép tính mới
- Xử lý lỗi chia cho 0:
    - Trong các nhãn thực hiện việc chia "/" và chia lấy dư "%" ta thêm lệnh kiểm tra xem s1 = 0 hay không
    - Nếu s1 = 0, in ra thông báo lỗi và yêu cầu thực hiện phép tính mới

3. Phân tích

❖ Định nghĩa các địa chỉ và khai báo dữ liệu

- Sử dụng các định nghĩa cho địa chỉ của đèn led 7 đoạn trái và phải, địa chỉ điều khiển đầu vào của keypad và địa chỉ đọc đầu ra từ keypad
- Định nghĩa nguyên nhân ngắt: 8 -> ngắt từ bàn phím
- Mảng A chứ các mã hiển thị cho led 7 đoạn từ 0-9 tùy vào các đoạn (a, b, c, d, e, f, g) được bật lên để tạo thành số tương ứng
- Chuỗi error .asciz chứa thông báo lỗi khi chia cho 0 kết thúc bằng ký tự null

❖ Chương trình chính (main)
- Gán địa chỉ của handler (thủ tục phục vụ ngắt) vào thanh ghi t0
- Ghi địa chỉ của handler vào thanh ghi utvec để khi có ngắt xảy ra, CPU sẽ nhảy đến địa chỉ được lưu trong utvec
- Bật bit 8 của thanh ghi uie cho phép ngắt ngoài ở chế độ User
- Bật bit uie trong thanh ghi ustatus -> bật ngắt tổng
- Nạp giá trị 0x80 vào thanh ghi t2 rồi đẩy vào địa chỉ cổng vào của bàn phím ma trận hexa. Bit 7 được bật -> phát sinh ngắt khi có một phím được nhấn và bật chức năng ngắt
- Khởi tạo các thanh ghi được sử dụng trong quá trình xử lý phép tính
    - s1: thanh ghi để chứa giá trị của số đang nhập
    - s0: chứa kết quả của phép tính
    - s2: hệ số nhân 10 -> dùng để nhân số hiện tại với 10 trước khi cộng thêm số mới giúp tạo nên số từ các chữ số riêng lẻ
    - t3: dùng để kiểm tra điều kiện
    - t4: khi nhấn phím nào thì t4 sẽ lưu giá trị của phím đó ở hệ 10
    - s4: để kiểm tra xem số đang nhập có là số đầu tiên của phép tính không
    - s5: kiểm tra dấu “=” được nhấn hay chưa (0-> chưa nhấn; 1-> đã nhấn). Được sử dụng để quyết định nên reset lại các giá trị cho phép tính mới hay tiếp tục tính tiếp từ kết quả cũ
- Vòng lặp loop: không làm gì cả -> chương trình thực hiện vòng lặp này trong khi chờ ngắt từ bàn phím

❖ Thủ tục phục vụ ngắt handler: khi có một ngắt xảy ra, CPU sẽ tự động nhảy đến địa chỉ được lưu trong utvec
- Giảm con trỏ stack để cấp phát không gian lưu trữ cho các thanh ghi. Lưu các giá trị của thanh ghi vào stack để khi quay lại chương trình chính giá trị của chúng được khôi phục và không bị ảnh hưởng
- a1 chứa thông tin về nguyên nhân gây ra ngắt. Bit 31 là 1 -> do ngắt, bit 31 là 0 -> do ngoại lệ

- a2 = 0x7FFFFFFF -> tất cả các bit còn lại đều là 1 trừ bit thứ 31 là 0
- Thực hiện and giữa a1 và s2 để xác định nguyên nhân ngắt thực sự
- So sánh với ngắt do bàn phím MASK_CAUSE_KEYPAD = 8 rồi nhảy đến keypad_isr
- Thủ tục ngắt bàn phím (keypad_isr)
    - Lần lượt nạp các giá trị 0x81, 0x82, 0x84, 0x88 vào địa chỉ cổng vào của bàn phím để quét từng dòng
    - Lấy ra từ địa chỉ cổng ra của bàn phím để xem mã phím nào được nhấn. Kiểm tra từng cột một và so sánh với các giá trị mã phím của hàng đang quét. Nếu khớp thì ta gán cho t4 giá trị tương ứng với phím đó (0-9: là số từ 0-9; 10-15: toán tử và "=")

❖ Xử lý logic

- Cập nhật số: việc nhấn số liên tục cần được xử lý để xây dựng số từ các chữ số riêng lẻ
    - Kiểm tra trước đó đã nhấn dấu "=" chưa -> nếu nhấn rồi mà muốn thực hiện phép tính mới (tức là nhấn luôn số mới) thì nhảy đến reset_all để khởi tạo lại các giá trị cho biến
    - new: Khi nhập số nhảy đến nhãn này. Thực hiện nhân số hiện tại (s1) với 10 (s2) rồi cộng với số hiện tại (s1) và lưu lại vào số hiện tại. Ví dụ: nhấn 5 rồi nhấn 6-> 5 * 10 = 50 -> 50 + 6 = 56 -> 56 sẽ được lưu vào thanh ghi s1 để được số hiện tại
    - Sau mỗi lần nhấn số đều nhảy đến hienthi để hiển thị các số vừa nhấn trên led 7 đoạn
- Xử lý các phép toán khi người dùng nhấn các phím toán tử
    - In ra giá trị của số hiện tại s1 trên màn hình Run I/O
    - Đặt s5 = 0 để báo hiệu đang thực hiện phép toán mới
    - Kiểm tra đã có số đầu tiên của phép tính chưa. Nếu chưa thì gán s0 = s1 -> s0 được gán là giá trị đầu tiên của phép tính luôn và nhảy đến update_op. Ngược lại nhảy đến calc_before để tính toán phép tính trước đó
- Xử lý phép toán trước đó (calc_before)
    - Thanh ghi a4 được dùng để lưu phép toán trước đó. So sánh để nhảy đến hàm tính toán tương ứng giữa kết quả cũ (s0) với số vừa nhập (s1)
        - calc_add: thực hiện phép toán + trước đó
        - calc_ sub: thực hiện phép toán – trước đó
        - calc_div: thực hiện phép toán / trước đó

        - calc_rem: thực hiện phép toán lấy dư %
    - Trong calc_div và calc_rem cần thêm điều kiện kiểm tra chia cho 0 vì không thể chia cho 0 -> in ra thông báo lỗi và reset lại các biến để thực hiện phép tính mới
- Cập nhật phép toán (update_op)
    - In ra toán tử vừa nhấn: so sánh t4 (10-14: là các dấu +,–, *,/,%)
    - Dùng lệnh gọi hệ thống in ra 1 ký tự
    - Sau khi in xong nhảy đến set_op
- Thiết lập phép toán (set_op)
    - s4 = 1: phép tính đã tồn tại số đầu tiên
    - Reset s1 = 0 để thực hiện cho việc nhập số tiếp theo
    - Lưu phép toán vừa in ra màn hình vào thanh ghi a4 (mv a4 = t4) -> phục vụ khi nhấn dấu "="
    - Kết thúc xử lý quay lại chương trình chính: j end_process
- Xử lý nhấn "=": tính toán kết quả và hiển thị trên màn hình
    - Tương tự như calc_before: kiểm tra phép toán cuối cùng lưu trong a4 và nhảy đến nhãn tương ứng để tính toán trước khi in ra màn hình Run I/O và led 7 đoạn
    - Khi nhấn số xong mà không nhấn các toán tử +, -, *, /, % mà nhấn "=" thì in ra Run I/O số vừa nhập đồng thời gán cho s0 = s1 rồi nhảy đến nhãn hiển thị kết quả
    - Đặt s5 = 1 -> phím "=" vùa được nhấn
- Hiển thị kết quả (hienthi) trên màn hình và led 7 đoạn
    - In ký tự "=" và in kết quả cuối cùng (s0) ra màn hình, in thêm ký tự xuống dòng '\n' để dễ dàng theo dõi các phép tính
    - s3 = s0 -> s3 được sử dụng để hiển thị 2 số trên led 7 đoạn
    - s1 = s0 -> gán kết quả cũ cho s1 để có thể tiếp tục tính toán với kết quả vừa rồi mà không cần nhập lại
    - a4 = 0 -> reset lại thanh ghi chứa dấu của phép toán
    - Kiểm tra dấu của kết quả s0
        - Nếu s0 ≥ 0 -> nhảy đến hienthisoduong
        - Nếu -10 < s0 < 0 -> đổi sang số dương rồi nhảy đến hienthisoam
        - Nếu s0 ≤ -10 -> đổi sang số dương để hiển thị 2 số cuối mà ko hiển thị ra dấu "–" trên led 7 đoạn
    - hienthisoduong: lấy phần dư của s3 khi mod 100 -> lọc ra 2 chữ số cuối cùng. Sau đó lọc ra chữ số hàng chục (/10) rồi nhảy đến

chương trình con để hiển thị trên led đoạn trái và lọc chữ số hàng đơn vị (%10) nhảy đến chương trình con thực hiện hiển thị trên led đoạn phải

- hienthisoam: xử lý hiển thị số âm (-10 < s0 < 0)
  - Gán cho thanh ghi a0 = 0x40 (đoạn g: tức là dấu '–') hiển thị trên led đoạn trái
  - Tương tự hiển thị số trên led đoạn phải

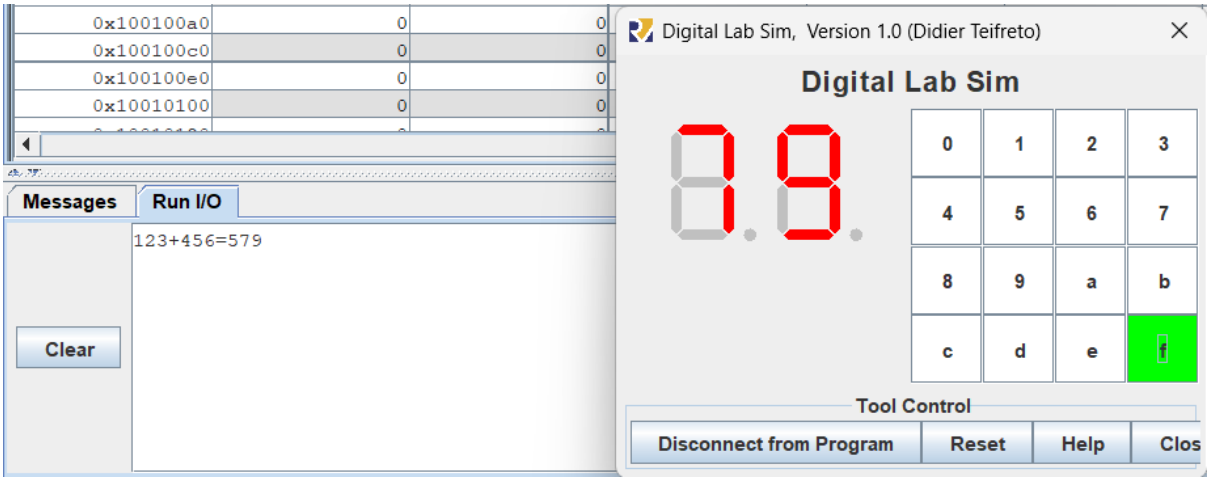
- Chương trình con hiển thị trên led 7 đoạn (SHOW_7SEG_LEFT, SHOW_7SEG_RIGHT)
  - Nạp địa chỉ của led trái vào thanh ghi t0
  - Ghi giá trị mã hiển thị trong a0 vào địa chỉ led trái
  - Quay lại chương trình jr ra
  - Tương tự với led đoạn phải
- Reset lại tất cả (reset_all)
  - Đặt lại số đang nhập s1 = 0, thanh ghi chứa kết quả s0 = 0
  - Trạng thái số đầu tiên s4 = 0, trạng thái nhấn dấu "=" s5 = 0
- Xử lý lỗi chia cho 0 (chia0_error)
  - Sử dụng lệnh gọi hệ thống in ra chuỗi thông báo lỗi (syscall = 4)
  - Đồng thời reset lại các giá trị để nhập lại phép tính toán mới

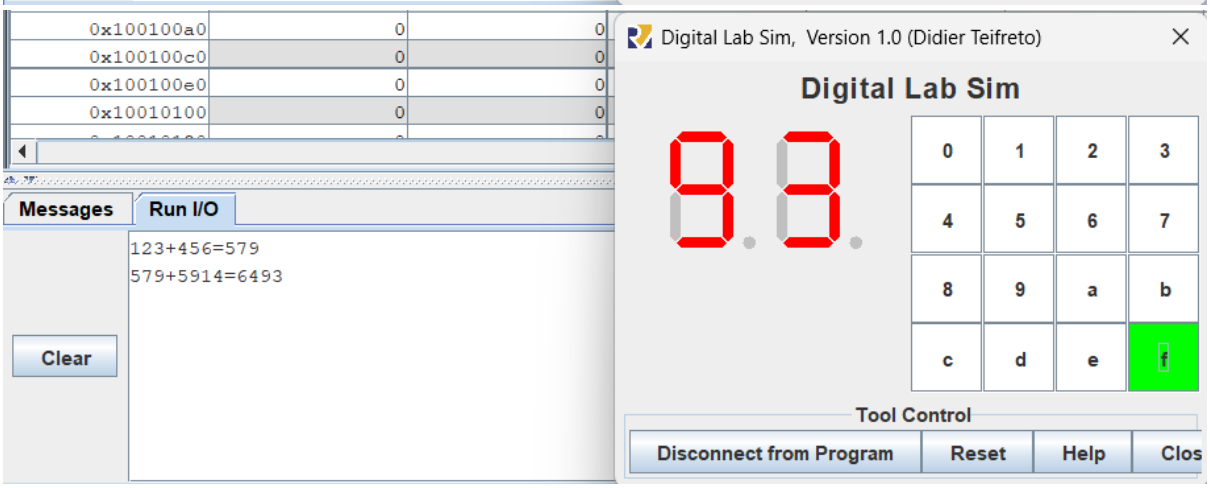
❖ Kết thúc thủ tục xử lý ngắt (end_process)

- Khôi phục lại các giá trị của các thanh ghi từ stack về các giá trị ban đầu trước khi xảy ra ngắt bàn phím
- Khôi phục lại vùng nhớ đã cấp phát của stack
- uret: khôi phục lại trạng thái của CPU về vị trí trước khi xảy ra ngắt -> thực hiện tiếp ở chương trình chính

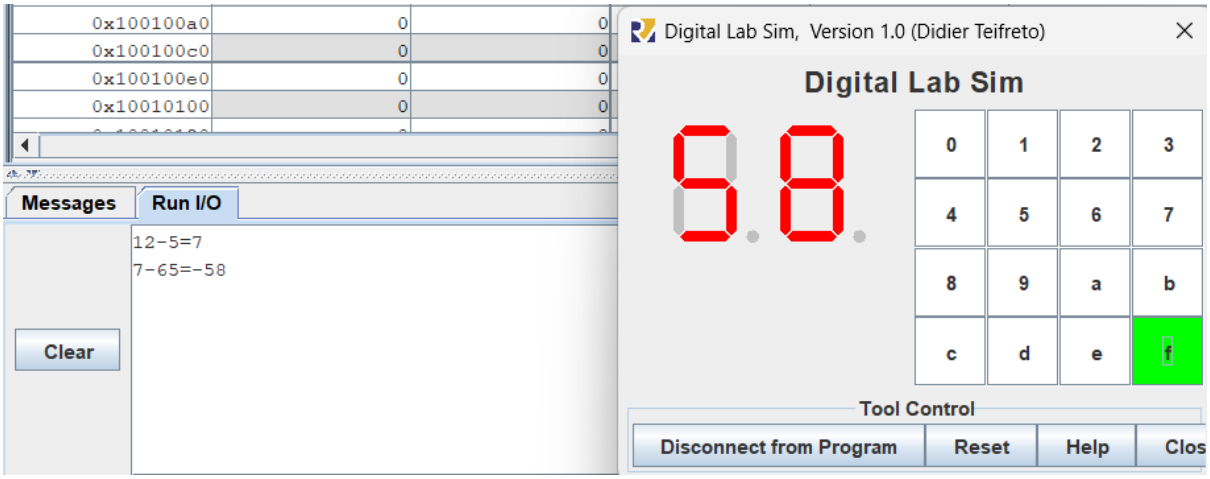
4. Kết quả

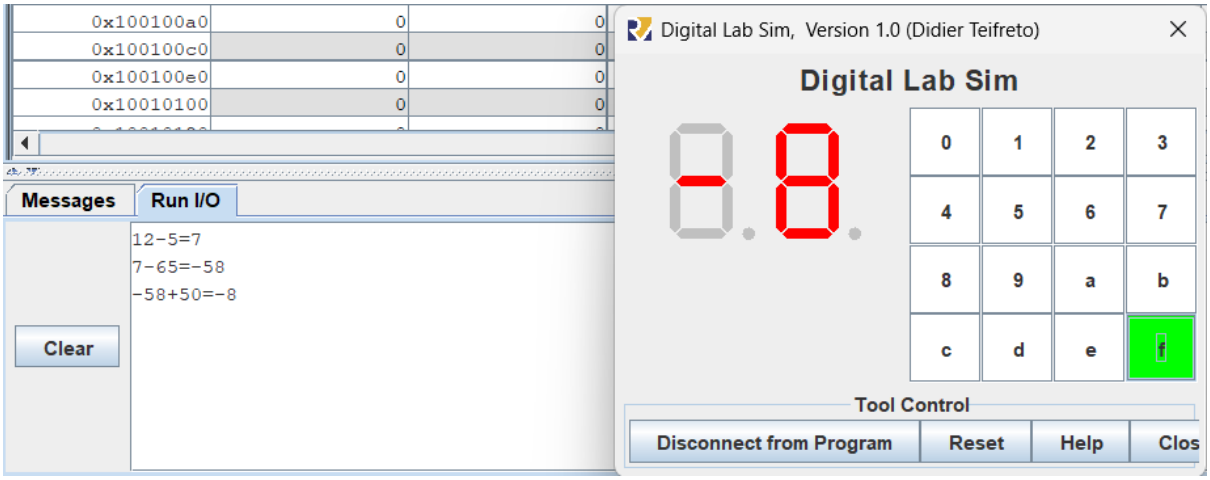
- Video kết quả của chương trình thực hiện đầy đủ các chức năng chính của chương trình
- Phép tính 1: thực hiện các phép toán cộng



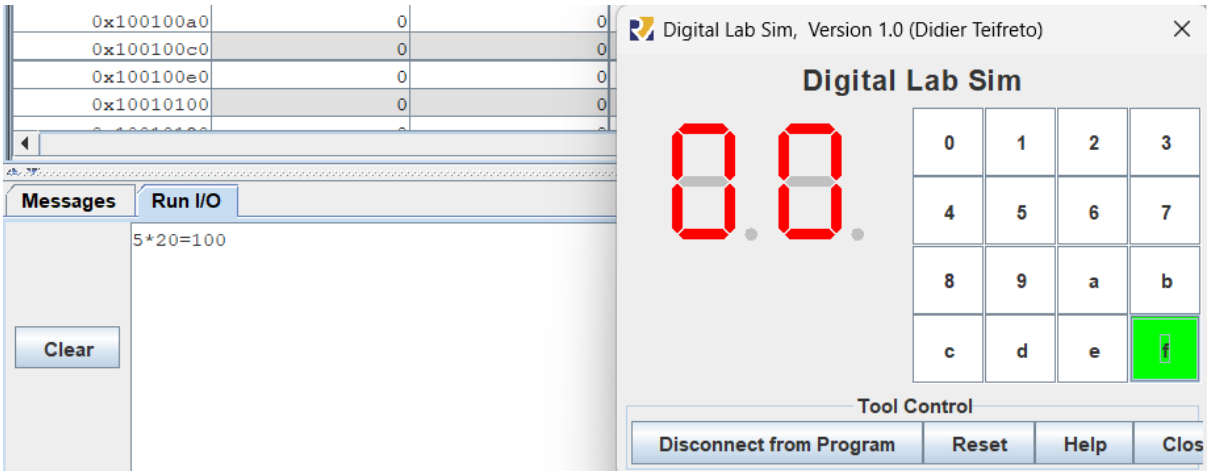


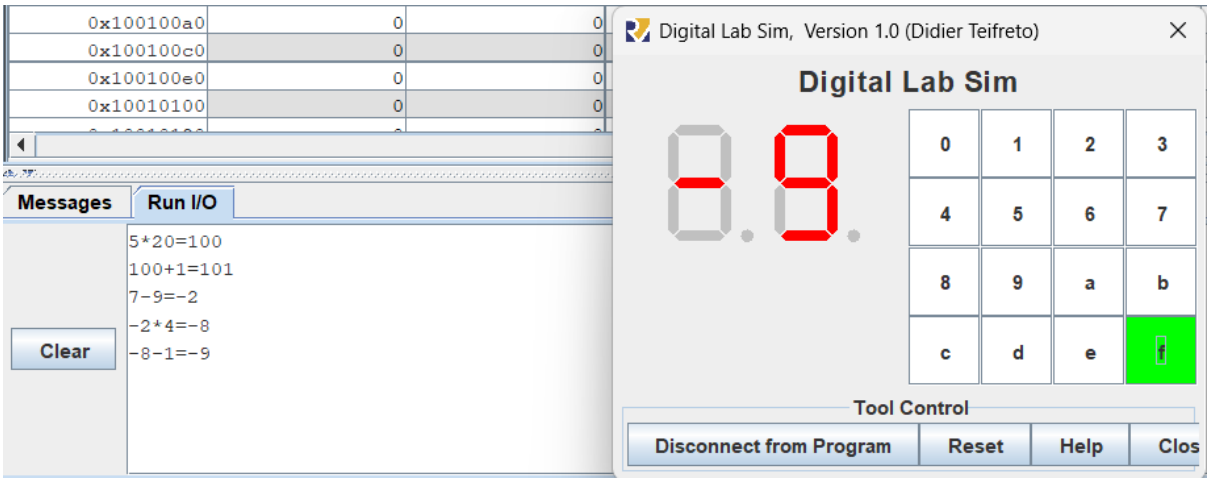
- Phép tính 2: thực hiện các phép toán trừ



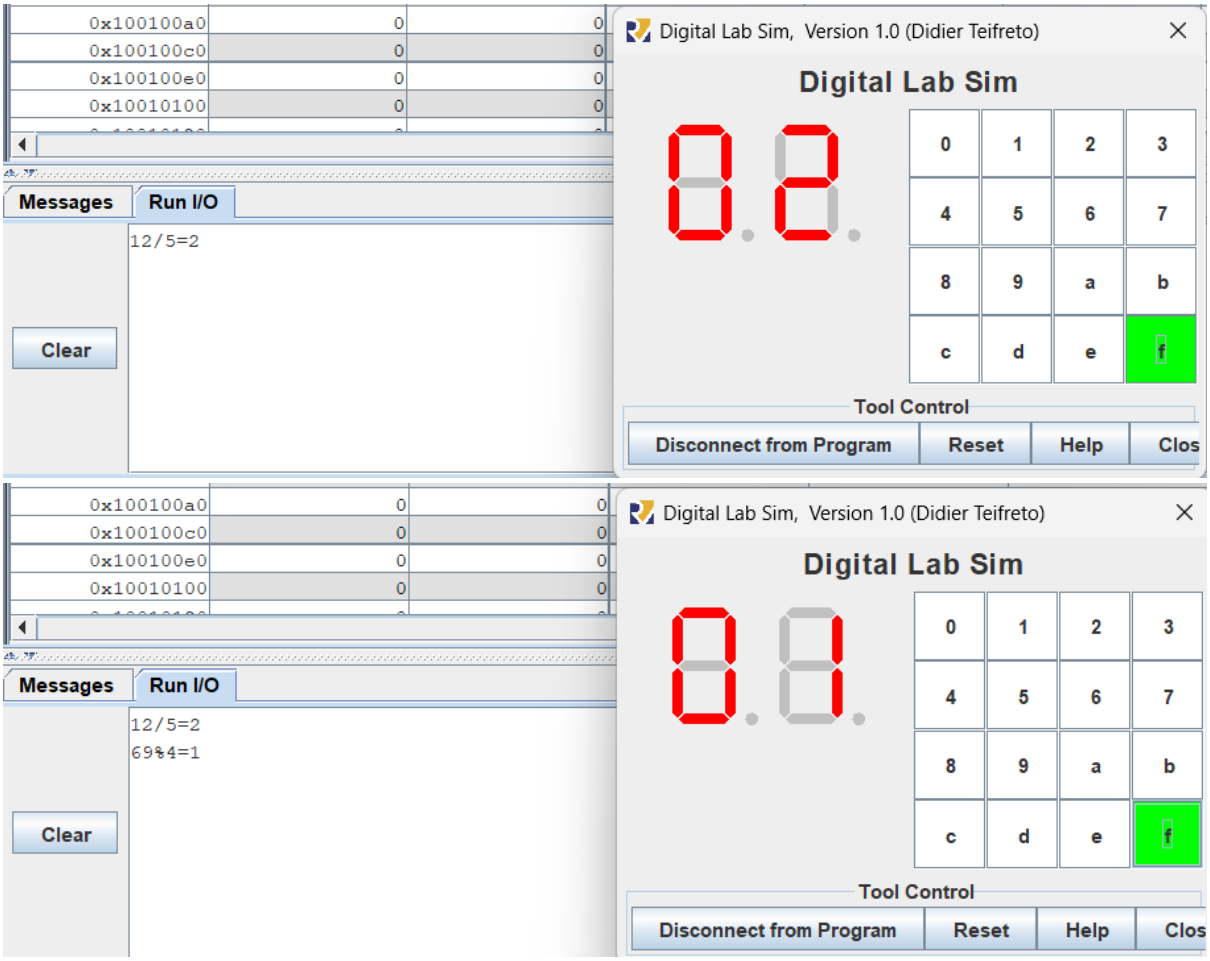


- Phép tính 3: thực hiện các phép nhân

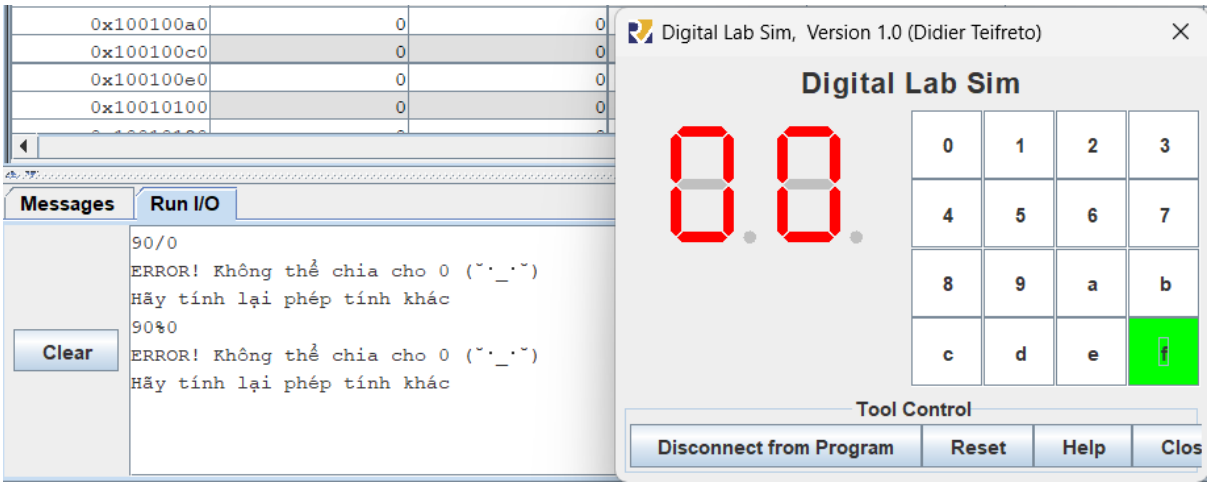




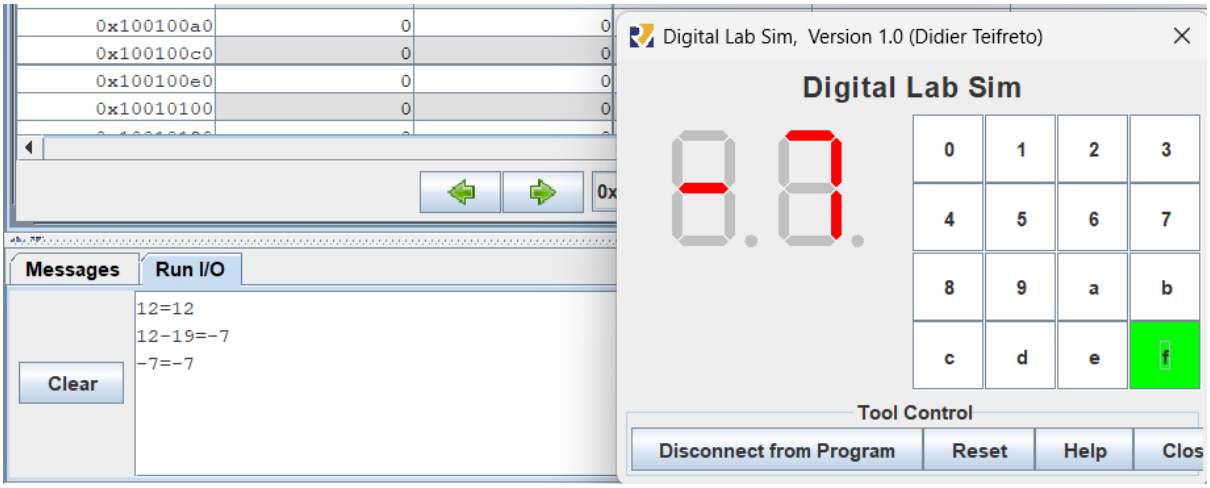
- Phép tính 4: thực hiện các phép tính chia và chia lấy dư



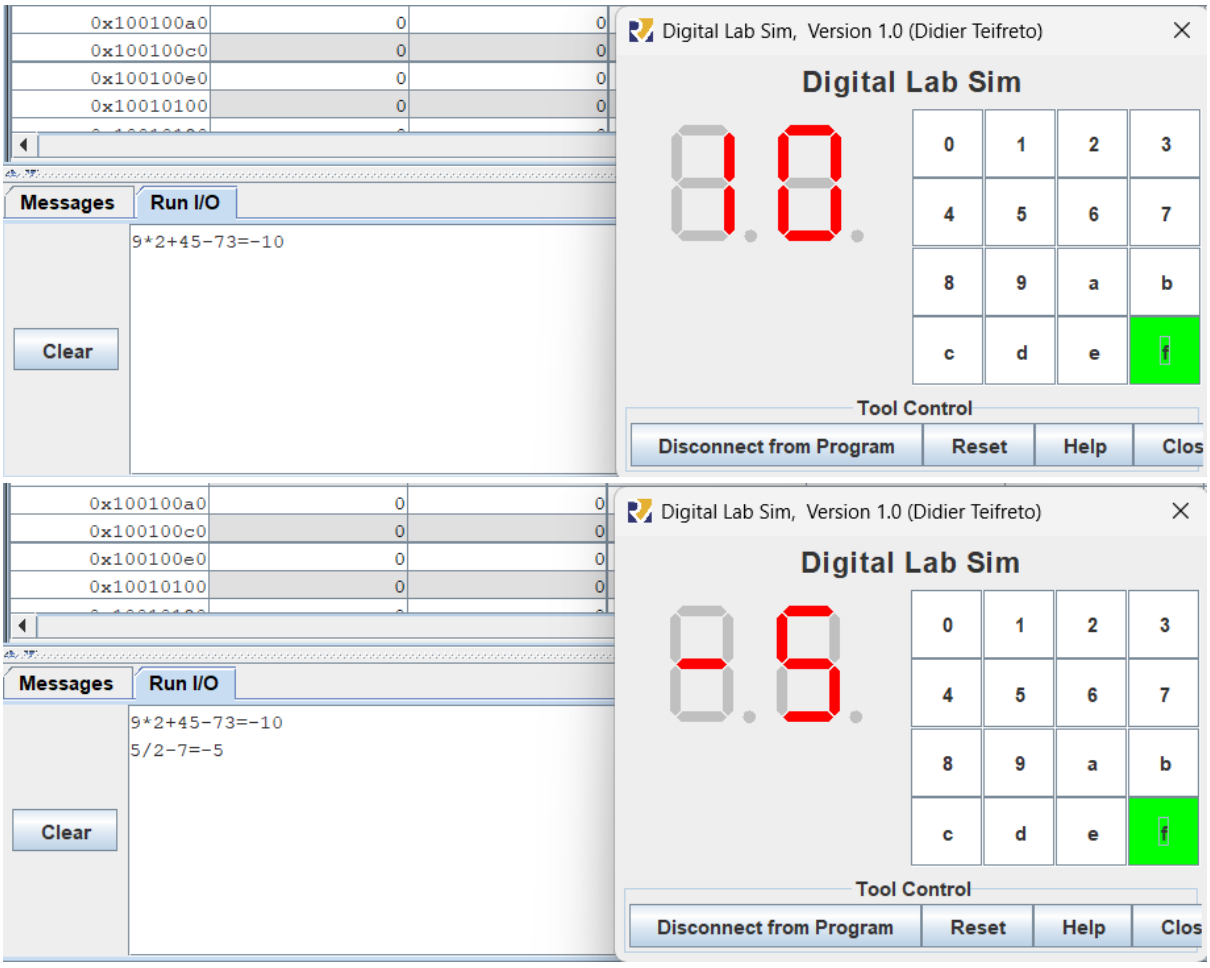
- Phép toán 5: thực hiện chia cho 0

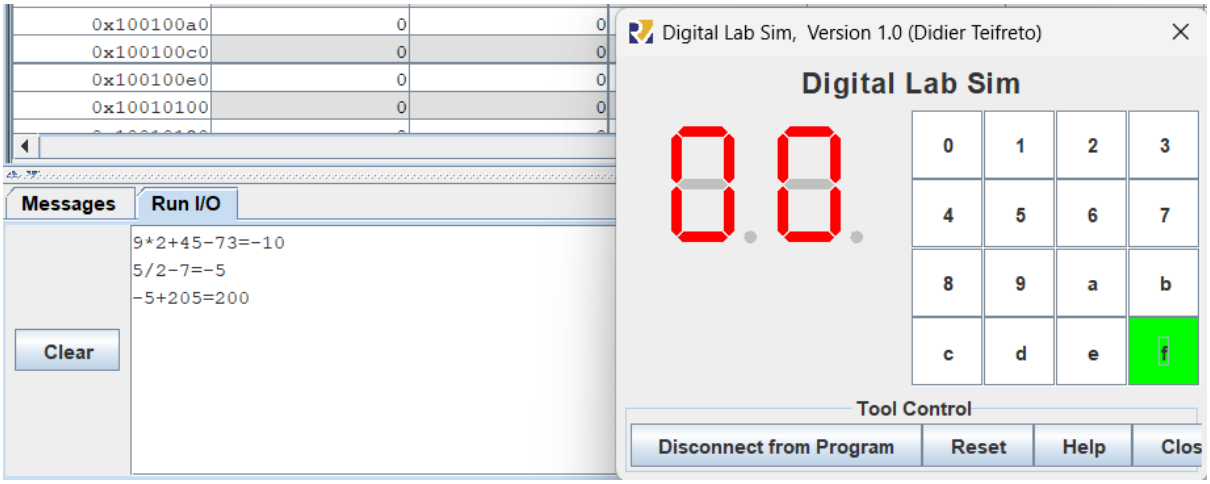


- Phép toán 6: phép tính không thực hiện tính toán (nhấn = liên tục)



- Phép toán 7: thực hiện các phép tính liên tiếp giống calculator




0x100100a0 0 0
0x100100c0 0 0
0x100100e0 0 0
0x10010100 0 0
Messages
Run I/O
9*2+45-73=-10
5/2-7=-5
-5+205=200
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
Clos

18. Đồng hồ điện tử

- Hiển thị thời gian hiện tại lên đèn LED 7 đoạn
- Bấm nút trên bàn phím KeyMatrix để chuyển chế độ hiển thị
  - Số 1: Hiển thị giờ
  - Số 2: Hiển thị phút
  - Số 3: Hiển thị giây
  - Số 4: Hiển thị ngày
  - Số 5: Hiển thị tháng
  - Số 6: Hiển thị 2 số cuối của năm
- Khi mỗi giây trôi qua cần cập nhật thời gian và hiển thị
- Khi chẵn một phút thì phát âm thanh báo hiệu

1. Mã nguồn

```
.eqv SEVENSEG_RIGHT    0xFFFF0010        # Dia chi cua den led 7 doan phai
.eqv SEVENSEG_LEFT    0xFFFF0011        # Dia chi cua den led 7 doan trai
.eqv IN_ADDRESS_HEXA_KEYBOARD   0xFFFF0012
.eqv OUT_ADDRESS_HEXA_KEYBOARD  0xFFFF0014
.eqv TIMER_NOW                  0xFFFF0018
.eqv TIMER_CMP                  0xFFFF0020
.eqv MASK_CAUSE_TIMER           4
.eqv MASK_CAUSE_KEYPAD          8

.data
    A: .byte 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F  # Các
số từ 0 đến 9
.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)

    li s2, 3            # kiểm tra xem nhấn phím nào (3: giây)
    li s4, -1           # lưu phút trước đó để kiểm tra xem đã trôi qua 1 phút
chưa

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, 30
    ecall                   # a0 = timestamp (ms)

    li a1, 0xD13DA400       # tính từ ngày 1/1/1970 đến 15/5/2025 có mã hexa
64bit là 0x196D13DA400 miliseconds
    sub t6, a0, a1          # số mili giây tính từ thời điểm 0:0:0 ngày
15/5/2025

    li s10, 25          # s10: năm -> cố định là 2025
```

```
    li s11, 86400000        # 1 ngày = 86400000 mili giây
    div s8, t6, s11         # s8: ngày

    li s11, 1468800000      # khoảng mili giây từ 15/5 đến 1/6
    beq t6, s11, thang6
    li s9, 5            # s9: tháng -> để tháng hiện tại là tháng 5

    j next
thang6:
    addi s9, s9, 1
    addi s8, s8, -15        # đặt lại mốc về đầu tháng 6

next:
    li s11, 86400000
    mul s11, s11, s8
    sub s11, t6, s11
    li s5, 3600000          # 1 giờ = 3600000 mili giây
    div s5, s11, s5         # s5: giờ

    li s11, 24          # một ngày có 24 giờ
    mul s6, s8, s11         # số giờ đã qua trong s8 ngày trước
    add s6, s6, s5          # số giờ đã qua tính đến thời điểm hiện tại
    li s11, 3600000
    mul s11, s6, s11
    sub s11, t6, s11
    li s6, 60000
    div s6, s11, s6         # s6: phut

    li s11, 24
    mul s7, s8, s11
    add s7, s7, s5
    li s11, 60          # một ngày có 60 phút
    mul s7, s7, s11
    add s7, s7, s6
    li s11, 60000           # 1 phút = 60000 mili giây
    mul s7, s7, s11
    sub s7, t6, s7
    li s11, 1000            # 1 giây = 1000 mili giây
    div s7, s7, s11         #s7: giay

    beqz s7, phat_am        # nếu như qua phút mới (s7 = 00) -> phát ra âm
thanh
    j chon_che_do

phat_am:
    li a0, 60           # nốt nhạc C4
    li a1, 400          # thời gian phát 400 ms
    li a2, 0            # loại nhạc cụ: piano
```

```
    li a3, 100          # volume
    li a7, 31
    ecall

chon_che_do:
# ---- Lấy giá trị tương ứng để hiển thị theo chế độ ----
    li t1, 10           # để lấy 2 số bên trái và phải của led 7 thanh
    la s0, A            # Lấy địa chỉ đầu tiên của A

    li s1, 1
    beq s2, s1, hien_thi_gio    # nếu s1 = 1 (nhấn phím 1) thì hiển thị trên
bàn phím giờ hiện tại

    li s1, 2
    beq s2, s1, hien_thi_phut   # nếu s1 = 2 (nhấn phím 2) thì hiển thị trên
bàn phím phút hiện tại

    li s1, 3
    beq s2, s1, hien_thi_giay   # nếu s1 = 3 (nhấn phím 3) thì hiển thị trên
bàn phím giờ hiện tại

    li s1,4
    beq s2, s1, hien_thi_ngay   # nếu s1 = 4 (nhấn phím 4) thì hiển thị trên
bàn phím giây hiện tại

    li s1, 5
    beq s2, s1, hien_thi_thang  # nếu s1 = 5 (nhấn phím 5) thì hiển thị trên
bàn phím tháng hiện tại

    li s1, 6
    beq s2, s1, hien_thi_nam    # nếu s1 = 6 (nhấn phím 6) thì hiển thị trên
bàn phím 2 số cuối của năm hiện tại

hien_thi_gio:
    addi s3, s5, 7          # cộng thêm 7 vì việt nam lệch 7 múi giờ
    j show
hien_thi_phut:
    mv s3, s6
    j show
hien_thi_giay:
    mv s3, s7
    j show
hien_thi_ngay:
    addi s3, s8, 15         # vì lấy mốc thời gian là 15/5/2025 nên ngày cần
cộng thêm 15
    j show
hien_thi_thang:
    mv s3, s9
```

```
    j show
hien_thi_nam:
    mv s3, s10
show:
    div t4, s3, t1
        add t4, t4, s0              # Địa chỉ của A[t4]
        lb a0, 0(t4)                # set value for segments
        jal SHOW_7SEG_LEFT          # show

    rem t5, s3, t1
        add t5, t5, s0              # Địa chỉ của A[t5]
        lb a0, 0(t5)                # set value for segments
        jal SHOW_7SEG_RIGHT         # show

    # 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      t1, IN_ADDRESS_HEXA_KEYBOARD
    li      t2, 0x81            # Check row 1 and re-enable bit 7
    sb      t2, 0(t1)           # Must reassign expected row
    li      t1, OUT_ADDRESS_HEXA_KEYBOARD
    lb      a0, 0(t1)

    li t1, 0x00000021       # kiem tra so 1
    beq a0, t1, gio

    li t1, 0x00000041       # kiem tra so 2
    beq a0, t1, phut

    li t1, 0xFFFFFF81       # kiem tra so 3
    beq a0, t1, giay

    li      t1, IN_ADDRESS_HEXA_KEYBOARD
    li      t2, 0x82            # Check row 2 and re-enable bit 7
    sb      t2, 0(t1)           # Must reassign expected row
    li      t1, OUT_ADDRESS_HEXA_KEYBOARD
    lb      a0, 0(t1)

    li t1, 0x00000012       # kiem tra so 4
    beq a0, t1, ngay
```

```
    li t1, 0x00000022       # kiem tra so 5
    beq a0, t1, thang

    li t1, 0x00000042       # kiem tra so 6
    beq a0, t1, nam

    j end_process

gio:
    li s2, 1
    j end_process
phut:
    li s2, 2
    j end_process
giay:
    li s2, 3
    j end_process
ngay:
    li s2, 4
    j end_process
thang:
    li s2, 5
    j end_process

nam:
    li s2, 6
    j end_process

#---------------------------------------------------------------
# Chương trình con thực hiện hiển thji số trên led 7 đoạn
#---------------------------------------------------------------
SHOW_7SEG_LEFT:
        li t0, SEVENSEG_LEFT        # assign port's address
        sb a0, 0(t0)                # assign new value
        jr ra
SHOW_7SEG_RIGHT:
        li t0, SEVENSEG_RIGHT       # assign port's address
        sb a0, 0(t0)                # assign new value
        jr ra

end_process:
    lw      a7, 12(sp)
    lw      a2, 8(sp)
    lw      a1, 4(sp)
    lw      a0, 0(sp)
    addi    sp, sp, 16
    uret                # quay lại chương trình chính
```

2. Ý tưởng

- Chương trình sử dụng ngắt timer và ngắt bàn phím để hiển thị thời gian thực trên LED 7 đoạn
- Xử lý hiển thị thời gian hiện tại trên LED 7 đoạn
  - Dữ liệu thời gian được tính dựa trên số mili giây kể từ mốc 1/1/1970
  - Chọn mốc thời gian gốc là 15/5/2025 00:00:00 và tính số mili giây trôi qua kể từ thời điểm đó -> chỉ làm việc với thanh ghi a1
- Sử dụng ngắt thời gian (Timer Interrupt):
  - Cập nhật thời gian mỗi giây
  - Chuyển đổi mili giây thành các đơn vị thời gian theo đề bài
  - Kiểm tra khi có sự thay đổi phút để phát âm thanh báo hiệu
  - Cập nhật hiển thị trên led 7 thanh theo các chế độ tùy chọn
- Sử dụng ngắt bàn phím (Keypad interrupt)
  - Phát hiện người dùng nhấn phím và thay đổi chế độ hiển thị
    - Các phím từ 1-6 lần lượt là các chế độ hiển thị: giờ, phút, giây, ngày, tháng, năm
    - Thanh ghi s2 được sử dụng để phục vụ logic chế độ được chọn
- Phát âm thanh bằng syscall MIDI khi trôi qua 1 phút
  - Sử dụng hàm hệ thống phát âm thanh (syscall 31)
  - Nếu trôi qua phút mới (s7=0 -> trôi qua 1 phút) thì phát ra nốt nhạc

3. Phân tích

❖ Định nghĩa các địa chỉ và khai báo dữ liệu

- Sử dụng các định nghĩa cho địa chỉ của đèn led 7 đoạn trái và phải, địa chỉ điều khiển đầu vào của keypad và địa chỉ đọc đầu ra từ keypad, địa chỉ lưu trữ thời gian hiện tại của bộ đếm thời gian, địa chỉ so sánh thời gian của bộ đếm-> nếu vượt quá thì xảy ra ngắt
- Định nghĩa nguyên nhân ngắt: 8 -> ngắt từ bàn phím; 4 -> ngắt định thời
- Mảng A chứ các mã hiển thị cho led 7 đoạn từ 0-9 tùy vào các đoạn (a, b, c, d, e, f, g) được bật lên để tạo thành số tương ứng

❖ Chương trình chính

- Gán địa chỉ của handler (thủ tục phục vụ ngắt) vào thanh ghi t0
- Ghi địa chỉ của handler vào thanh ghi utvec để khi có ngắt xảy ra, CPU sẽ nhảy đến địa chỉ được lưu trong utvec
- Bật bit 8 của thanh ghi uie cho phép ngắt ngoài ở chế độ User

- Bật bit 4 trong thanh ghi uie cho phép ngắt định thời
- Bật bit uie trong thanh ghi ustatus -> bật ngắt tổng
- Kích hoạt ngắt bàn phím: nạp giá trị 0x80 vào thanh ghi t2 rồi đẩy vào địa chỉ cổng vào của bàn phím ma trận hexa. Bit 7 được bật -> phát sinh ngắt khi có một phím được nhấn và bật chức năng ngắt
- Kích hoạt ngắt định thời: nạp địa chỉ so sánh thời gian vào t1, t2 lưu giá trị thời gian 1000ms. Sau đó ghi vào địa chỉ TIMER_CMP
- Khởi tạo các thanh ghi s2: lưu chế độ hiển thị thời gian, được mặc định giá trị là 3 tức là giây
- Vòng lặp loop: vòng lặp vô hạn không làm gì cả. Chương trình chạy liên tục ở đây chờ đợi ngắt từ timer hoặc bàn phím

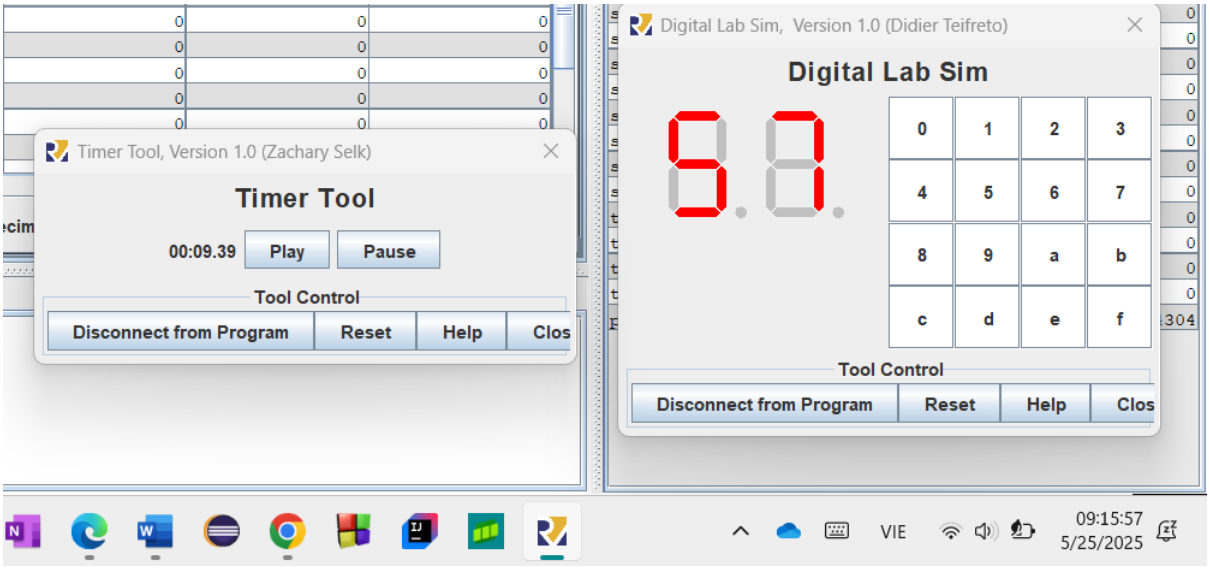
❖ Thủ tục phục vụ ngắt (handler) khi có một ngắt xảy ra do timer hoặc bàn phím thì CPU sẽ nhảy đến đây
- Giảm con trỏ stack để cấp phát không gian lưu trữ cho các thanh ghi. Lưu các giá trị của thanh ghi vào stack để khi quay lại chương trình chính giá trị của chúng được khôi phục và không bị ảnh hưởng
- a1 chứa thông tin về nguyên nhân gây ra ngắt. Bit 31 là 1 -> do ngắt, bit 31 là 0 -> do ngoại lệ
- a2 = 0x7FFFFFFF -> tất cả các bit còn lại đều là 1 trừ bit thứ 31 là 0
- Thực hiện and giữa a1 và s2 để xác định nguyên nhân ngắt thực sự
- So sánh với ngắt do bàn phím MASK_CAUSE_KEYPAD = 8 hay do timer MASK_CAUSE_TIMER = 4 rồi nhảy đến nhãn tương ứng. Nếu không thuộc nguyên nhân nào trong 2 nguyên nhân trên thì j end_process để khôi phục ngữ cảnh và quay lại chương trình chính

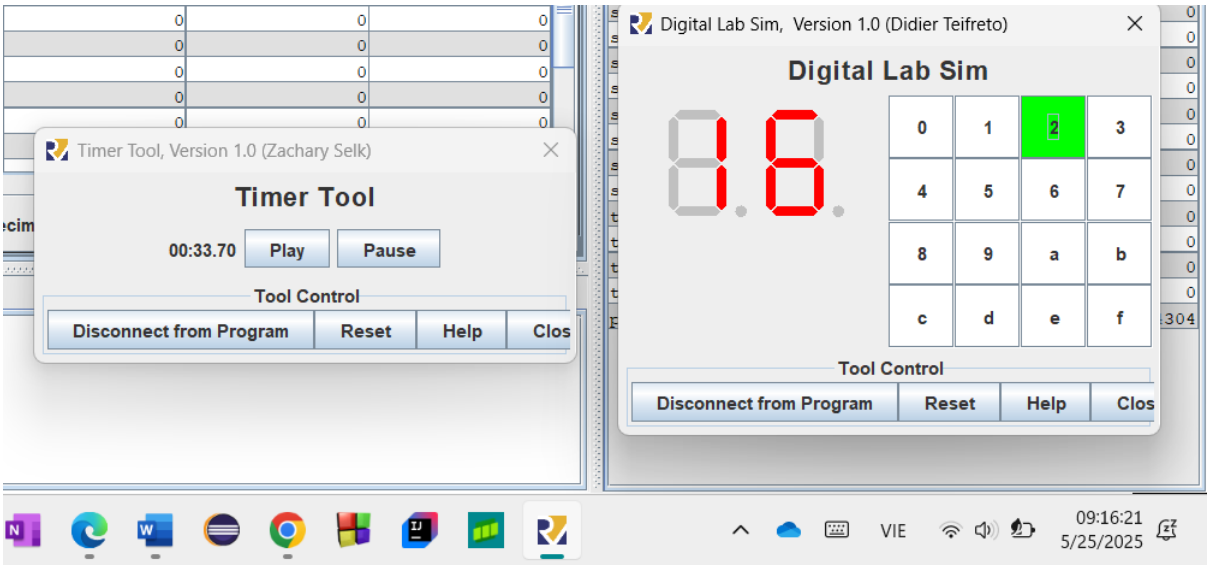
❖ Các thủ tục phục vụ ngắt
- Ngắt định thời (timer_isr) được kích hoạt mỗi giây để cập nhật thời gian và hiển thị trên led 7 đoạn
    - Gọi hàm hệ thống lấy thời gian hiện tại li a7, 30
    - Thời gian được tính bằng mili giây kể từ thời điểm 1/1/1970 với
        - a0: lưu trữ 32 bit thấp
        - a1: lưu trữ 32 bit cao
- Tính toán thời gian kể từ thời điểm 00:00:00 15/5/2025 (tức là chỉ làm việc với 32 bit thấp)
    - Nạp vào thanh ghi a1 giá trị mili giây tại thời điểm 15/5/2025
    - Lấy thời gian hiện tại trừ đi thời gian mốc (a1) rồi lưu vào thanh ghi t6 để phục vụ cho việc tính toán thời gian sau này
    - Cố định thời gian là năm 2025

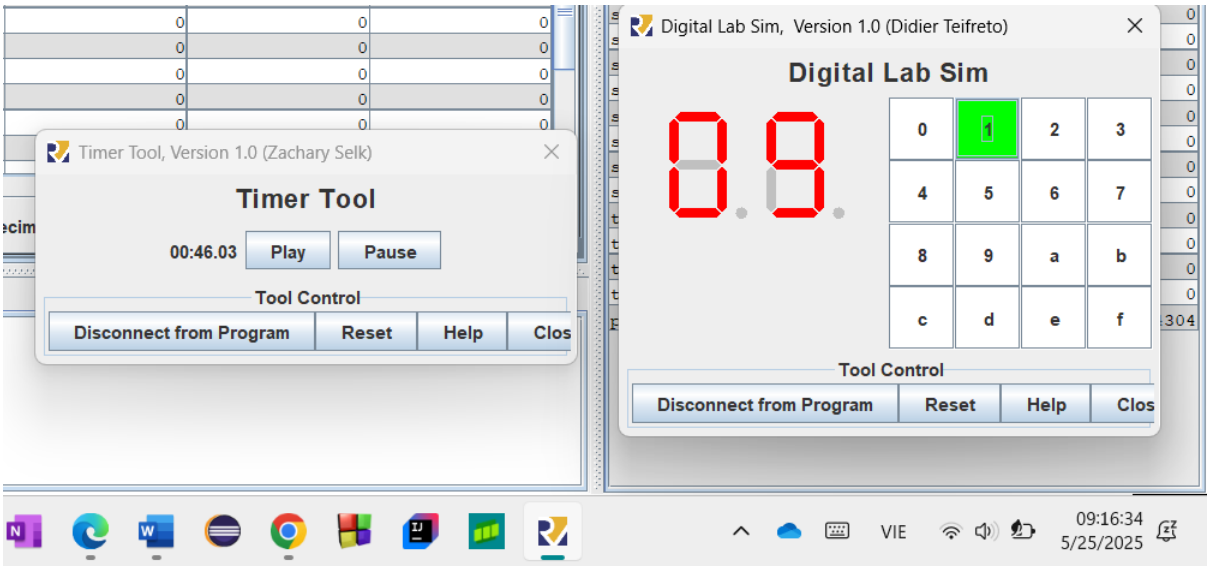
    - s8: lưu giá trị ngày hiện tại. lấy t6 / 86400000 (sử dụng công cụ để chuyển đổi số mili giây trong 1 ngày) ta được số ngày tính từ mốc thời gian
    - s9: lưu tháng hiện tại, ta gán cho giá trị 5 (tháng 5). kiểm tra t6 có vượt qua mốc 1/6/2025 hay không-> nếu có thì +1 vào s9
    - s5: lưu giờ hiện tại. lấy mili giây trong 1 ngày nhân với số ngày đã qua (s8); sau đó lấy t6 trừ đi rồi chia cho số mili giây trong 1 giờ -> ta được giờ hiện tại
    - Xử lý tương tự với phút s6; và giây s7
    - Kiểm tra xem s7 = 0 -> trôi qua phút mới -> phát âm thanh
- Phát âm thanh (phat_am)
    - Dịch vụ hệ thống (syscall) số 31 trong RARS là một chức năng cho phép tạo âm thanh MIDI được mô phỏng bằng card âm thanh của hệ thống thông qua thư viện javax.sound.midi của Java
    - Cần truyền 4 tham số đầu vào cho các thanh ghi
        - a0 – pitch (cao độ): giá trị trong khoảng 0-127 -> 60 : nốt C4
        - a1 - duration (thời lượng): thời gian phát nốt nhạc. Nếu giá trị âm thì mặc định là 1000ms
        - a2 - instrument (nhạc cụ): giá trị trong khoảng 0-127. Nếu giá trị không hợp lệ, hệ thống mặc định dùng 0 (piano)
        - a3 - volume (âm lượng): tăng dần từ 0-127. Nếu nằm ngoài phạm vi, hệ thống sẽ dùng mặc định là 100
- Chọn chế độ hiển thị (chon_che_do)
    - Tải địa chỉ đầu mảng A vào s0
    - So sánh s2 với các giá trị từ 1-6 để lựa chọn chế độ hiển thị
    - s3 được sử dụng để gán các giá trị thời gian hiển thị trên led 7 đoạn
- Hiển thị trên led 7 đoạn (show)
    - Chia s3 /10 để lấy chữ số hàng chục -> cộng với địa chỉ đầu mảng A để lấy ra mã LED của số
    - Nhảy đến chương trình con thực hiện hiển thị trên led đoạn trái
    - Tương tự với led đoạn phải, thực hiện s3%10 rồi nhảy đến chương trình con hiển thị led đoạn phải
- Cài đặt ngắt định thời tiếp theo: thiết lập thời điểm cho ngắt timer tiếp theo, đảm bảo nó xảy ra chính xác sau 1 giây nữa
    - Đọc giá trị thời gian hiện tại từ địa chỉ TIMER_NOW và lưu vào a1

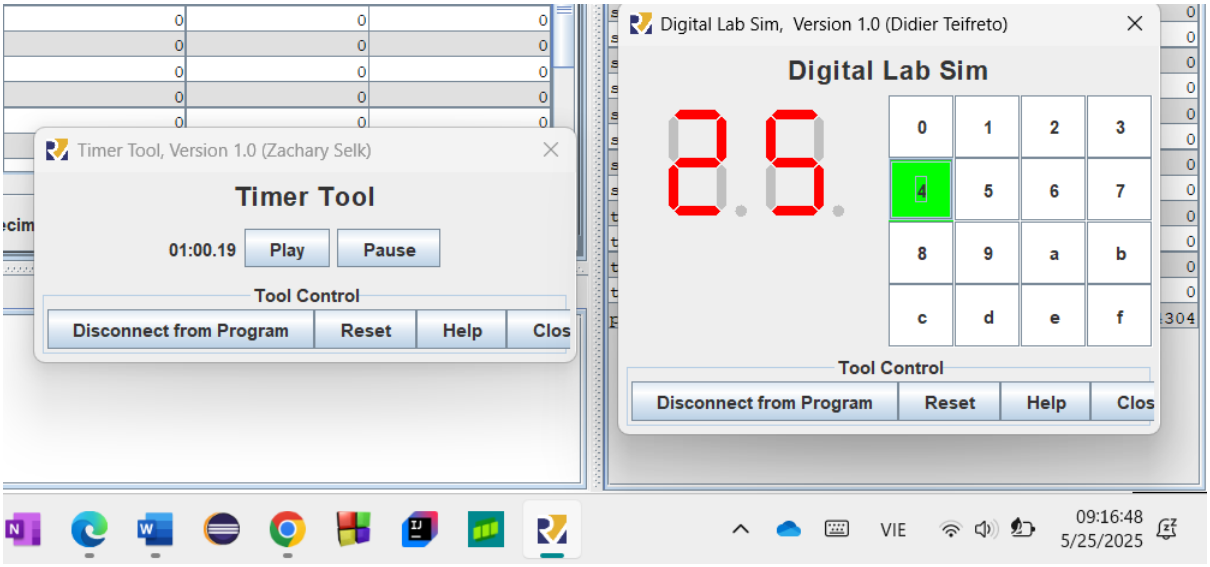
        - Cộng thêm 1000ms vào a1
        - Ghi giá trị a1 vào địa chỉ của TIMER_CMP
    - Ngắt bàn phím (keypad_isr) được kích hoạt khi nhấn phím
        - Nạp địa chỉ cổng vào bàn phím vào thanh ghi t1
        - t2 = 0x81 -> bit 7 bật để kích hoạt ngắt, bit 0 bật để kiểm tra hàng 1 của bàn phím
        - Ghi t2 vào địa chỉ của cổng vào bàn phím để kích hoạt ngắt
        - Đọc giá trị cột được nhấn từ cổng ra bàn phím và so sánh với các mã phím để nhảy đến nhãn tương ứng
        - Tương tự với hàng thứ hai t2 = 0x82
        - Các nhãn gio (giờ), phut (phút), giay (giây), ... thanh ghi t2 lưu trữ chế độ hiển thị của phím nhấn
    - Chương trình con thực hiện hiển thị trên led 7 đoạn (SHOW_7SEG_LEFT, SHOW_7SEG_RIGHT)
        - Nạp địa chỉ led trái vào thanh ghi t0
        - Ghi giá trị mã hiển thị vào t0
        - Quay lại từ hàm con
        - Tương tự với led đoạn phải
- ❖ Kết thúc thủ tục xử lý ngắt (end_process)
    - Khôi phục lại các giá trị của các thanh ghi từ stack về các giá trị ban đầu trước khi xảy ra ngắt bàn phím
    - Khôi phục lại vùng nhớ đã cấp phát của stack
    - uret: khôi phục lại trạng thái của CPU về vị trí trước khi xảy ra ngắt -> thực hiện tiếp ở chương trình chính

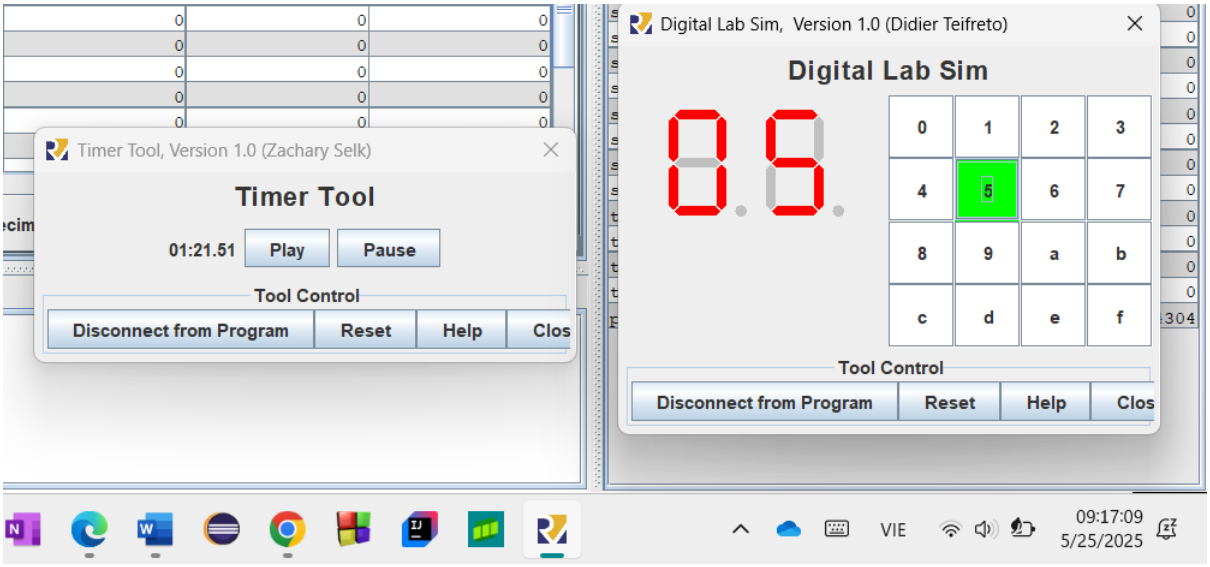
4. Kết quả
    - Video kết quả của chương trình có bao gồm âm thanh phát nhạc
    - Kết quả tương ứng với thời gian thực

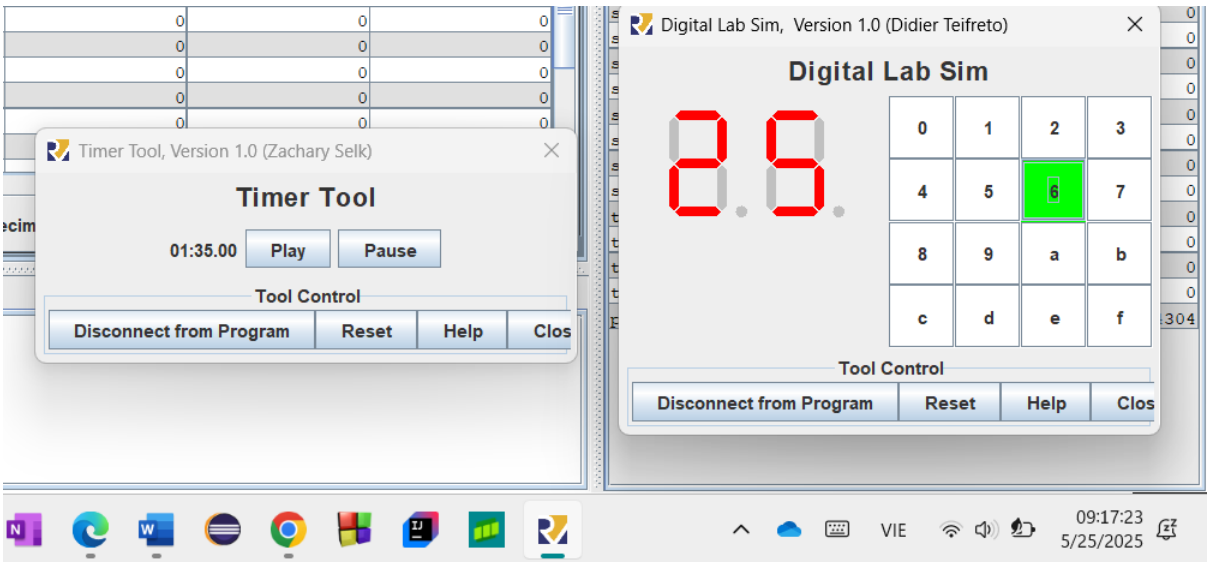

Timer Tool, Version 1.0 (Zachary Selk)
Timer Tool
00:09.39
Play
Pause
Tool Control
Disconnect from Program
Reset
Help
Clos
Digital Lab Sim, Version 1.0 (Didier Teifreto)
Digital Lab Sim
0 1 2 3
4 5 6 7
8 9 a b
c d e f
Tool Control
Disconnect from Program
Reset
Help
Clos
VIE
09:15:57
5/25/2025


Timer Tool, Version 1.0 (Zachary Selk)
Timer Tool
00:33.70
Play
Pause
Tool Control
Disconnect from Program
Reset
Help
Clos
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
Clos
VIE
09:16:21
5/25/2025


Timer Tool, Version 1.0 (Zachary Selk)
Timer Tool
00:46.03
Play
Pause
Tool Control
Disconnect from Program
Reset
Help
Clos
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
Clos
VIE
09:16:34
5/25/2025


Timer Tool, Version 1.0 (Zachary Selk)
Timer Tool
01:00.19
Play
Pause
Tool Control
Disconnect from Program
Reset
Help
Clos
Digital Lab Sim, Version 1.0 (Didier Teifreto)
Digital Lab Sim
2.6.
0 1 2 3
4 5 6 7
8 9 a b
c d e f
Tool Control
Disconnect from Program
Reset
Help
Clos
VIE
09:16:48
5/25/2025


Timer Tool, Version 1.0 (Zachary Selk)
Timer Tool
01:21.51
Play
Pause
Tool Control
Disconnect from Program
Reset
Help
Clos
Digital Lab Sim, Version 1.0 (Didier Teifreto)
Digital Lab Sim
8.6.
0 1 2 3
4 5 6 7
8 9 a b
c d e f
Tool Control
Disconnect from Program
Reset
Help
Clos
VIE
09:17:09
5/25/2025


Timer Tool, Version 1.0 (Zachary Selk)
Timer Tool
01:35.00
Play
Pause
Tool Control
Disconnect from Program
Reset
Help
Clos
Digital Lab Sim, Version 1.0 (Didier Teifreto)
Digital Lab Sim
2.6.
0 1 2 3
4 5 6 7
8 9 a b
c d e f
Tool Control
Disconnect from Program
Reset
Help
Clos
VIE
09:17:23
5/25/2025