Bài 10. Giao tiếp với các thiết bị ngoại vi

Họ và tên: Nguyễn Thành Duy

MSSV: 20235696

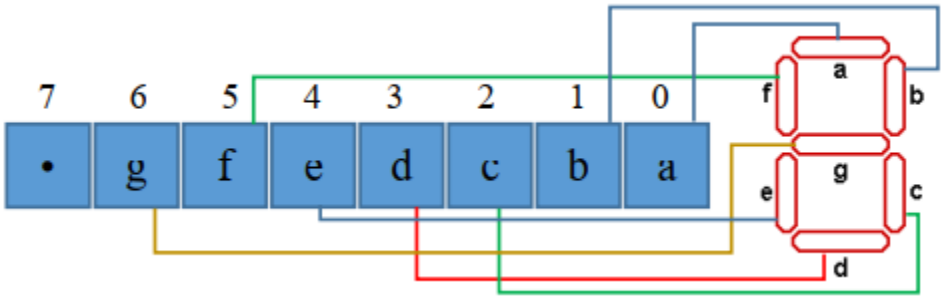
Assignment 1

Code:

```
.eqv SEVENSEG_LEFT 0xFFFF0011 # Dia chi cua den led 7 doan trai
# Bit 0 = doan a
# Bit 1 = doan b
# ...
# Bit 7 = dau .
.eqv SEVENSEG_RIGHT 0xFFFF0010 # Dia chi cua den led 7 doan phai
.text
main:
li a0, 0x6F # set value for segments
jal SHOW_7SEG_LEFT # show
li a0, 0x7D # set value for segments 01111101
jal SHOW_7SEG_RIGHT # show
exit:
li a7, 10
ecall
end_main:
# ---------------------------------------------------------------
# Function SHOW_7SEG_LEFT : turn on/off the 7seg
# param[in] a0 value to shown
# remark t0 changed
# ---------------------------------------------------------------
SHOW_7SEG_LEFT:
li t0, SEVENSEG_LEFT # assign port's address
sb a0, 0(t0) # assign new value
jr ra
# ---------------------------------------------------------------
# Function SHOW_7SEG_RIGHT : turn on/off the 7seg
# param[in] a0 value to shown
# remark t0 changed
# ---------------------------------------------------------------
SHOW_7SEG_RIGHT:
li t0, SEVENSEG_RIGHT # assign port's address
sb a0, 0(t0) # assign new value
jr ra
```

Giải thích:

- MSSV: 20235696 -> 2 số cuối là 96



Ở thanh leb bên trái để hiển thị số 9: thanh e = 0; a,b,c,d,f,g = 1

➔ 01101111 = 0x6F

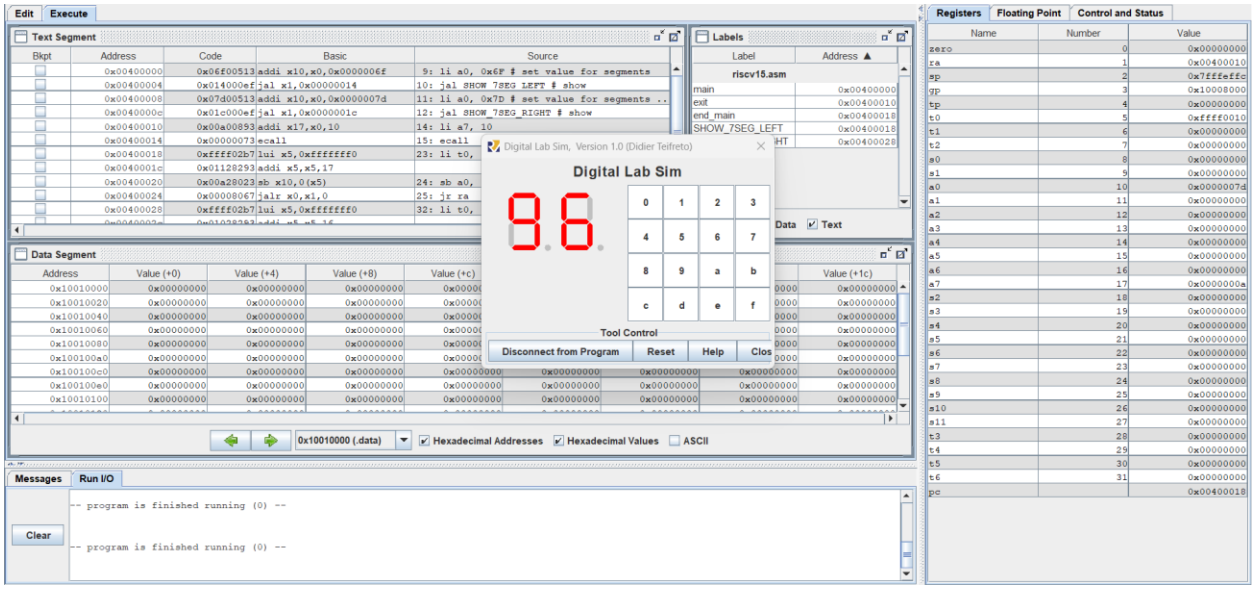
```
li a0, 0x6F # set value for segments
```

Ở thanh leb bên phải để hiển thị số 6: thanh b = 0; a,c,d,e,f,g = 1

➔ 01111101 = 0x7D

```
li a0, 0x7D # set value for segments 01111101
```

Kết quả sau khi chạy chương trình:



Assignment 2

Code:

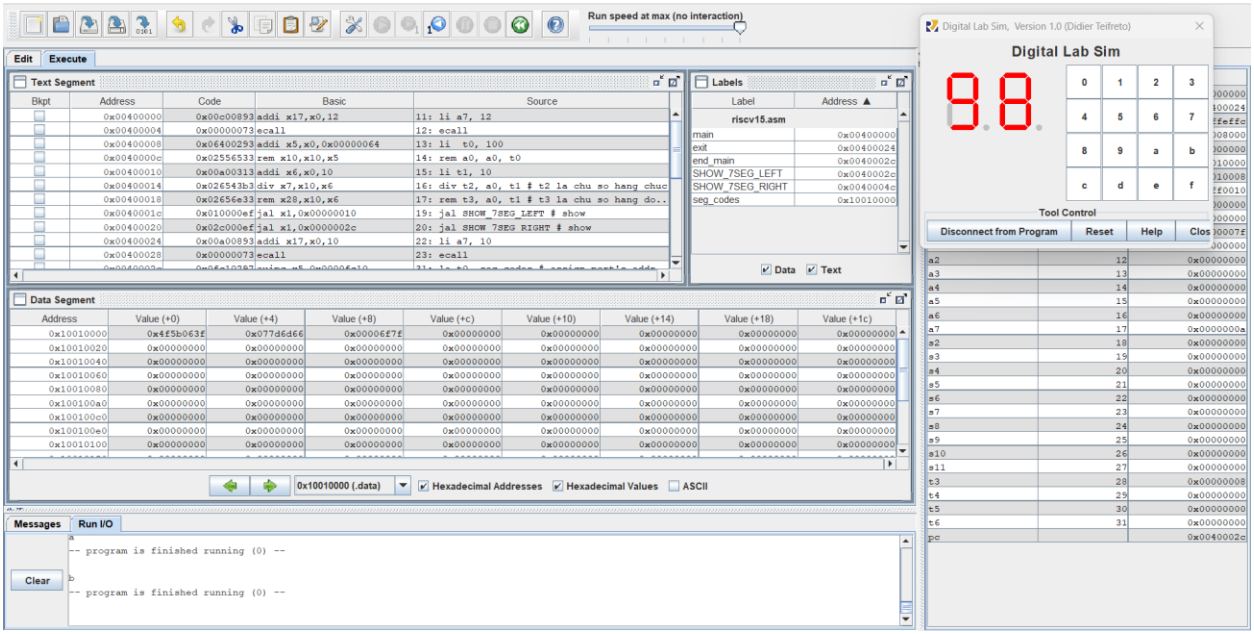
```
.data
seg_codes: .byte 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F
.eqv SEVENSEG_LEFT 0xFFFF0011 # Dia chi cua den led 7 doan trai
# Bit 0 = doan a
# Bit 1 = doan b
# ...
# Bit 7 = dau .
.eqv SEVENSEG_RIGHT 0xFFFF0010 # Dia chi cua den led 7 doan phai
.text
main:
li a7, 12
ecall
li  t0, 100
rem a0, a0, t0
li t1, 10
div t2, a0, t1 # t2 la chu so hang chuc
rem t3, a0, t1 # t3 la chu so hang don vi
#---------------------------------------
jal SHOW_7SEG_LEFT # show
jal SHOW_7SEG_RIGHT # show
exit:
li a7, 10
ecall
end_main:
# ---------------------------------------------------------------
# Function SHOW_7SEG_LEFT : turn on/off the 7seg
# param[in] a0 value to shown
# remark t0 changed
# ---------------------------------------------------------------
SHOW_7SEG_LEFT:
la t0, seg_codes # assign port's address
add t1, t0, t2
lb a0, 0(t1)
li t2, SEVENSEG_LEFT
sb a0, 0(t2) # assign new value
jr ra
# ---------------------------------------------------------------
# Function SHOW_7SEG_RIGHT : turn on/off the 7seg
# param[in] a0 value to shown
# remark t0 changed
```

```
# ---------------------------------------------------------------
SHOW_7SEG_RIGHT:
la t0, seg_codes # assign port's address
add t1, t0, t3
lb a0, 0(t1)
li t2, SEVENSEG_RIGHT
sb a0, 0(t2) # assign new value
jr ra
```

Kết quả sau khi chạy chương trình:

Input: b

Output: 98 (Mã ASCII của b)



Giải thích:

Để hiển thị đúng yêu cầu cần lấy được 2 chữ số cuối từ mã ASCII của ký tự được nhập vào

```
li a7, 12
ecall
li  t0, 100
rem a0, a0, t0
li t1, 10
div t2, a0, t1 # t2 la chu so hang chuc
rem t3, a0, t1 # t3 la chu so hang don vi
```

Đoạn code trên lấy 2 chữ số cuối bằng cách chia mã ASCII cho 100, rồi chia dư cho 10 để được chữ số hàng đơn vị, chia cho 10 để lấy hàng chục.

Sau khi đã tách được 2 chữ số để hiển thị ở 2 thanh Led thì cần chuyển qua segcode thì để dễ dàng thì tạo một mảng các segcode tương ứng 0->9

```
.data
seg_codes: .byte 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F
```

Sau khi lấy được segcode thì đưa vào từng thanh Leb để hiển thị.

```
SHOW_7SEG_LEFT:
la t0, seg_codes # assign port's address
add t1, t0, t2
lb a0, 0(t1)
li t2, SEVENSEG_LEFT
sb a0, 0(t2) # assign new value
jr ra
```

```
SHOW_7SEG_RIGHT:
la t0, seg_codes # assign port's address
add t1, t0, t3
lb a0, 0(t1)
li t2, SEVENSEG_RIGHT
sb a0, 0(t2) # assign new value
jr ra
```

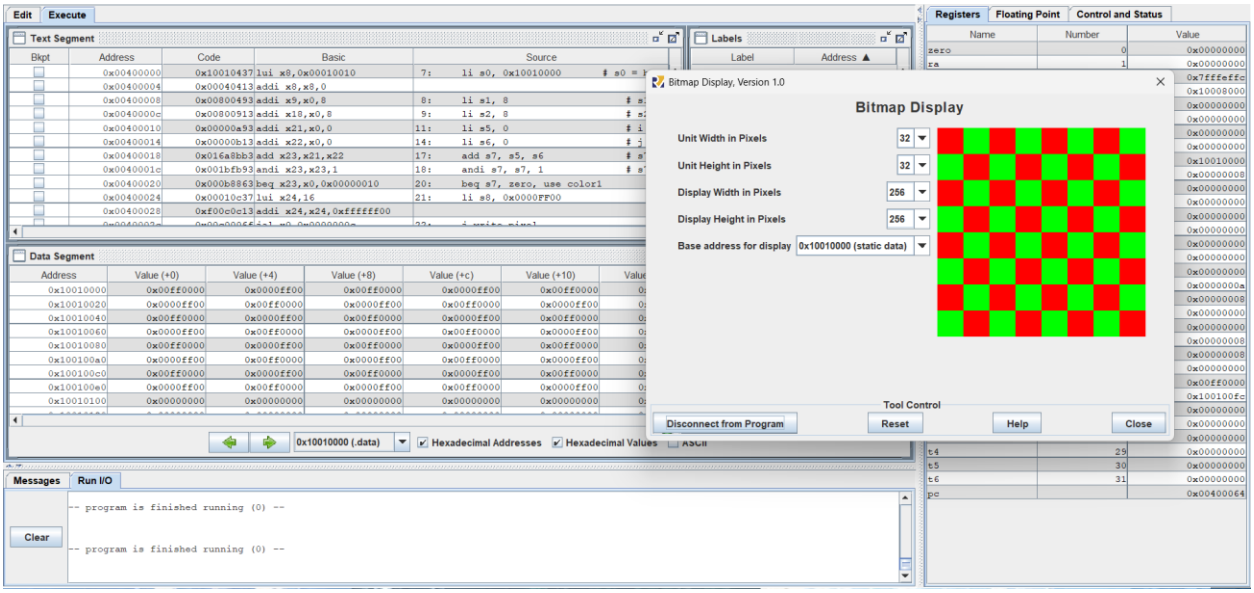
Assignment 3

Code:

```
.eqv MONITOR_SCREEN 0x10010000
.eqv RED   0x00FF0000
.eqv GREEN  0x0000FF00

.text
main:
  li s0, MONITOR_SCREEN     # s0 = base address màn hình
  li s1, 8                  # s1 = số hàng
  li s2, 8                  # s2 = số cột

  li s5, 0                  # i = row

outer_loop:
  li s6, 0                  # j = col
```

```
inner_loop:
   add s7, s5, s6          # s7 = row + col
   andi s7, s7, 1          # s7 = (row + col) % 2

   beq s7, zero, use_color1
   li s8, GREEN            # s8 = màu 2
   j write_pixel

use_color1:
   li s8, RED              # s8 = màu 1

write_pixel:
   mul s9, s5, s2          # s9 = row * 8
   add s9, s9, s6          # s9 = row * 8 + col
   slli s9, s9, 2          # s9 = byte offset
   add s9, s0, s9          # s9 = địa chỉ pixel

   sw s8, 0(s9)            # ghi màu

   addi s6, s6, 1          # j++
   blt s6, s2, inner_loop

   addi s5, s5, 1          # i++
   blt s5, s1, outer_loop

exit:
   li a7, 10
   ecall
```

Kết quả sau chạy chương trình:



Giải thích:

Cấu trúc bàn cờ:

- Bàn cờ là hình vuông 8x8 ô.
- Mỗi ô là một **pixel màu 32-bit**, lưu tại vùng nhớ MONITOR_SCREEN.
- Các ô được tô màu **xen kẽ** theo quy tắc:
  +) Nếu (row + col) % 2 == 0 → dùng **màu A**
  +) Ngược lại -> dùng **màu B**

Vị trí của bộ nhớ của mỗi ô:

address = MONITOR_SCREEN + ((row * 8 + col) * 4)

Ý tưởng thuật toán:

Duyệt qua từng hàng i từ 0 → 7 (biến row)

- Trong mỗi hàng, duyệt qua từng cột j từ 0 → 7 (biến col)
  - Tính màu: nếu (row + col) % 2 == 0 → màu 1, ngược lại → màu 2
  - Tính địa chỉ pixel → ghi giá trị màu vào bộ nhớ tại địa chỉ đó

Assignment 4

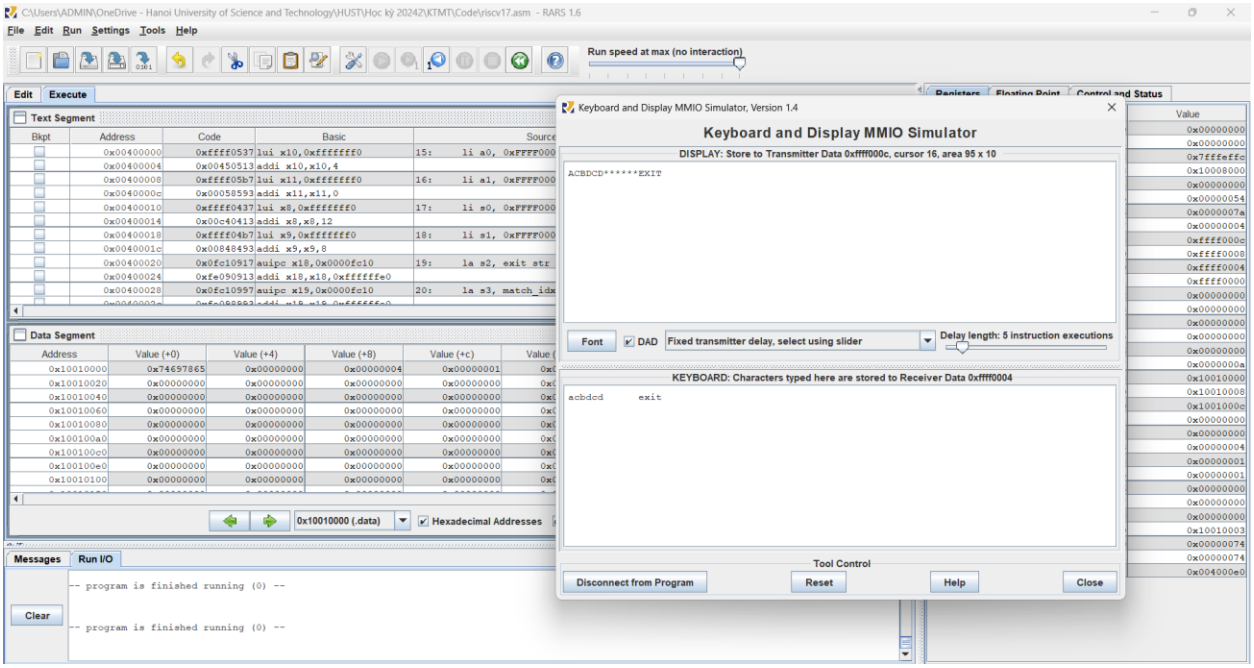
Code:

```
.eqv KEY_CODE,      0xFFFF0004
.eqv KEY_READY,     0xFFFF0000
.eqv DISPLAY_CODE,  0xFFFF000C
.eqv DISPLAY_READY, 0xFFFF0008

.data
exit_str:   .asciz "exit"
match_idx:  .word 0     # chỉ số so khớp chuỗi "exit"
exit_flag:  .word 0     # cờ báo sẽ thoát sau khi hiển thị

.text
.globl main
main:
   # Khởi tạo
   li a0, KEY_CODE
   li a1, KEY_READY
   li s0, DISPLAY_CODE
   li s1, DISPLAY_READY
   la s2, exit_str
   la s3, match_idx
   la s4, exit_flag

main_loop:
wait_key:
   lw t1, 0(a1)
   beqz t1, wait_key

   lw t0, 0(a0)    # load 4 byte từ KEY_CODE (vì KEY_CODE chứa 4 byte)
   andi t0, t0, 0xFF # chỉ lấy 1 byte thấp nhất

   # Trước tiên xử lý ký tự để hiển thị
   li t1, 'a'
   blt t0, t1, check_upper
   li t1, 'z'
   bgt t0, t1, check_upper
   addi t0, t0, -32          # lowercase -> uppercase
   j display

check_upper:
   li t1, 'A'
```

```
    blt t0, t1, check_digit
    li t1, 'Z'
    bgt t0, t1, check_digit
    addi t0, t0, 32          # uppercase -> lowercase
    j display

check_digit:
    li t1, '0'
    blt t0, t1, other_char
    li t1, '9'
    bgt t0, t1, other_char
    j display

other_char:
    li t0, '*'

display:
wait_display:
    lw t2, 0(s1)
    beqz t2, wait_display
    sw t0, 0(s0)     # Hiển thị ký tự đã xử lý

    # Sau khi hiển thị xong mới kiểm tra exit
    j check_exit

check_exit:
    lw t2, 0(s3)    # match_idx
    add t4, s2, t2
    lb t5, 0(t4)    # exit_str[match_idx]

    # So sánh ký tự vừa nhập (gốc, trước xử lý) với exit_str
    lw t6, 0(a0)
    andi t6, t6, 0xFF   # chỉ lấy 1 byte
    bne t6, t5, reset_match

    # Nếu đúng
    addi t2, t2, 1
    sw t2, 0(s3)

    li s7, 4
    bne t2, s7, main_loop

    # Nếu match_idx == 4, bật cờ exit
    li s8, 1
```

```
    sw s8, 0(s4)

    # Kiểm tra exit_flag để kết thúc
check_exit_flag:
    lw s9, 0(s4)
    beqz s9, main_loop

    li a7, 10
    ecall

reset_match:
    sw zero, 0(s3)
    j main_loop
```

Kết quả sau khi chạy chương trình:



Giải thích:

- Nhập ký tự từ bàn phím, kiểm tra nếu có ký tự mới qua KEY_READY.
- Kiểm tra chuỗi "exit": Đếm ký tự đã nhập và dừng nếu chuỗi "exit" được nhập.
- Chuyển đổi ký tự:
    - Thường (a–z) → thành hoa (A–Z) bằng cách trừ 32.
    - Hoa (A–Z) → thành thường (a–z) bằng cách cộng 32.
    - Số (0–9) → giữ nguyên.
    - Ký tự khác → hiển thị *.

- Hiển thị ký tự: Gửi ký tự đã chuyển đổi vào bộ nhớ hiển thị (DISPLAY_CODE) khi màn hình sẵn sàng.
- Vòng lặp tiếp tục cho đến khi nhập "exit" thì kết thúc chương trình.

Assignment yêu nước: