Thực hành kiến trúc máy tính

Báo cáo thực hành

Bài 10. Giao tiếp với các thiết bị ngoại vi

Họ Tên	Lê Thành An
MSSV	20235631

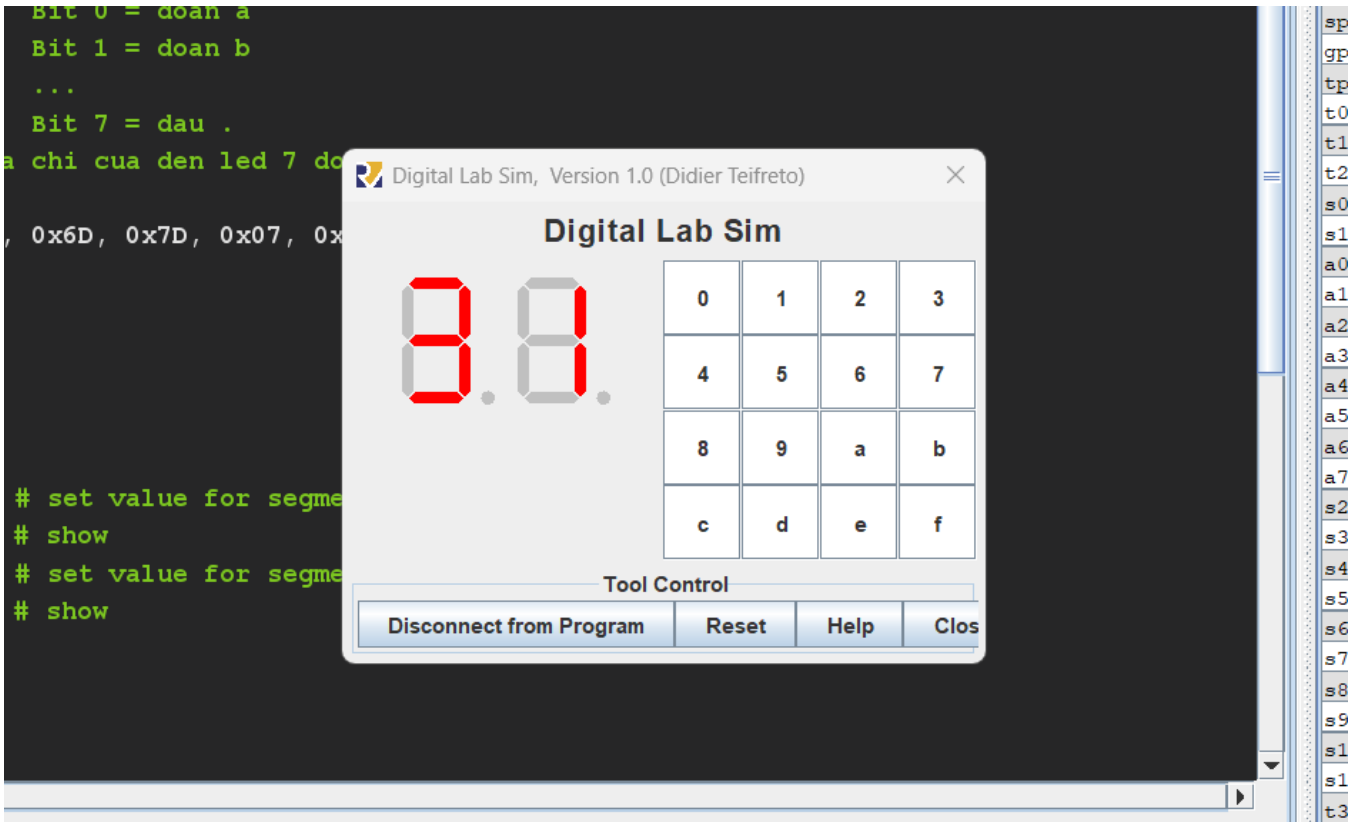
ASSIGNMENT 1

ĐOẠN MÃ :

```
.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
.data
char: .word 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F
.text
main:
    la      s0,  char
    lw      s1, 4(s0)
    lw      s2, 12(s0)
print:
    add      a0,  zero, s2              # set value for segments
    jal     SHOW_7SEG_LEFT             # show
    add      a0,  zero, s1              # set value for segments
    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
```

Kết quả:



ASSIGNMENT 2

ĐOẠN MÃ :

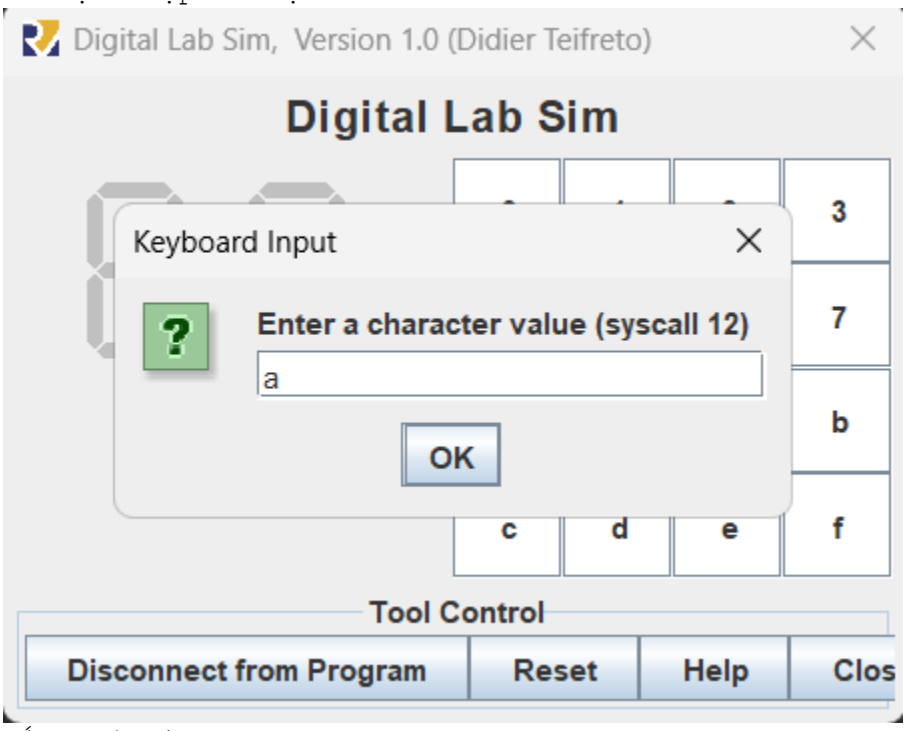
```
.eqv SEVENSEG_LEFT    0xFFFF0011    # Địa chỉ của đèn LED 7 đoạn bên trái
                                    #     Bit 0 = đoạn a
                                    #     Bit 1 = đoạn b
                                    #     ...
                                    #     Bit 7 = dấu chấm
.eqv SEVENSEG_RIGHT   0xFFFF0010    # Địa chỉ của đèn LED 7 đoạn bên phải

.data
# Mảng mã hiển thị cho các số từ 0-9 trên LED 7 đoạn (dạng Common Anode)
A: .word 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F

.text
main:
    # Thiết lập tham số cho hệ thống
    li a7, 12                       # Số hiệu syscall để đọc số nguyên từ
bàn phím
    li a6, 10                       # Hằng số 10 dùng để chia
```

```
    la t2, A                        # Tải địa chỉ bảng mã LED 7 đoạn

    li t1, 4                        # Kích thước mỗi phần tử trong mảng (4
byte)

    ecall                           # Đọc số từ bàn phím (kết quả lưu vào
a0)

    # Xử lý chữ số hàng đơn vị (hiển thị trên LED bên phải)
    rem a2, a0, a6                  # Lấy chữ số hàng đơn vị (a0 % 10)
    div a0, a0, a6                  # Chia số cho 10 để lấy chữ số hàng
chục
    mul a2, a2, t1                  # Tính offset trong mảng A (index * 4)
    add a2, a2, t2                  # Tính địa chỉ phần tử trong mảng A
    lw a1, 0(a2)                    # Lấy mã hiển thị cho chữ số này

    jal SHOW_7SEG_RIGHT             # Hiển thị chữ số hàng đơn vị trên LED
phải

    # Xử lý chữ số hàng chục (hiển thị trên LED bên trái)
    rem a2, a0, a6                  # Lấy chữ số hàng chục (a0 % 10)
    div a0, a0, a6                  # Chia số cho 10 (ở đây chỉ để minh
họa)
    mul a2, a2, t1                  # Tính offset trong mảng A
    add a2, a2, t2                  # Tính địa chỉ phần tử trong mảng A
    lw a1, 0(a2)                    # Lấy mã hiển thị cho chữ số này

    jal SHOW_7SEG_LEFT              # Hiển thị chữ số hàng chục trên LED
trái

exit:
    li a7, 10                       # Số hiệu syscall để kết thúc chương
trình
    ecall
end_main:

# ---------------------------------------------------------------
# Hàm SHOW_7SEG_LEFT: Bật/tắt các đoạn của LED 7 đoạn trái
# Tham số đầu vào:
#    a1 - Giá trị cần hiển thị (mã các đoạn)
# Ghi chú: Thanh ghi t0 bị thay đổi
# ---------------------------------------------------------------
SHOW_7SEG_LEFT:
    li t0, SEVENSEG_LEFT           # Gán địa chỉ thanh ghi điều khiển LED
trái
    sb a1, 0(t0)                    # Ghi giá trị hiển thị vào cổng LED
    jr ra                           # Trở về từ hàm con

# ---------------------------------------------------------------
# Hàm SHOW_7SEG_RIGHT: Bật/tắt các đoạn của LED 7 đoạn phải
# Tham số đầu vào:
#    a1 - Giá trị cần hiển thị (mã các đoạn)
# Ghi chú: Thanh ghi t0 bị thay đổi
# ---------------------------------------------------------------
SHOW_7SEG_RIGHT:
    li t0, SEVENSEG_RIGHT          # Gán địa chỉ thanh ghi điều khiển LED
```

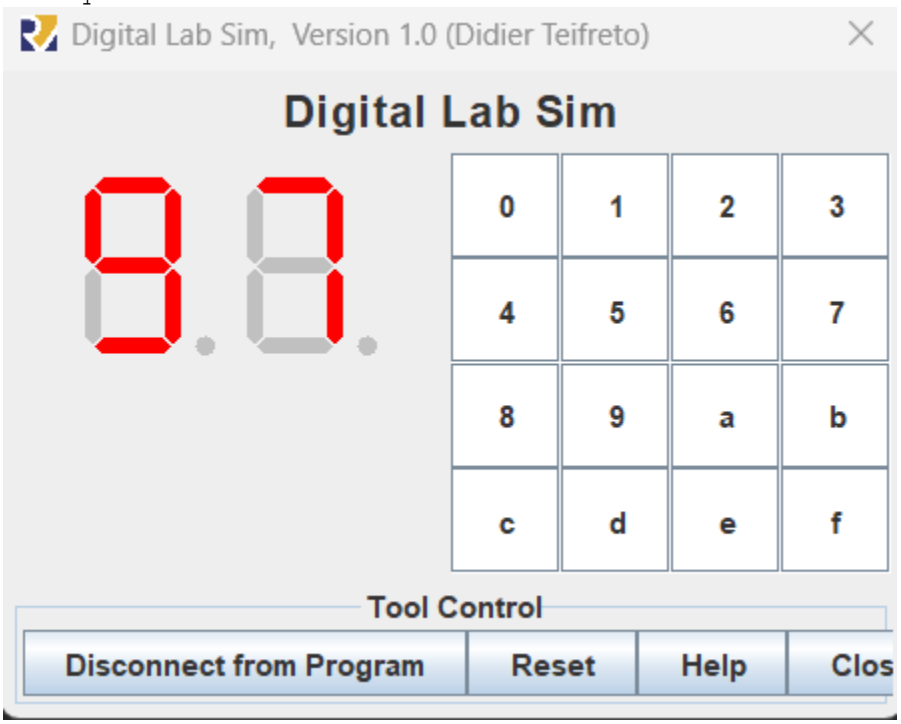
```
phải
    sb a1, 0(t0)                    # Ghi giá trị hiển thị vào cổng LED
    jr ra                           # Trở về từ hàm con
```

Kết quả:

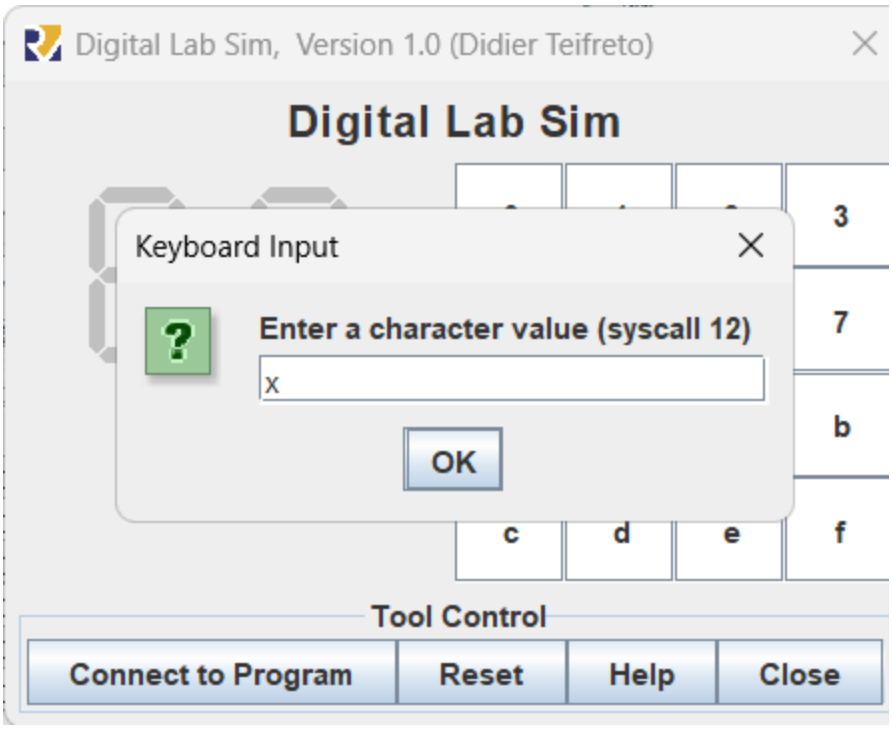
Ví dụ: Nhập kí tự a



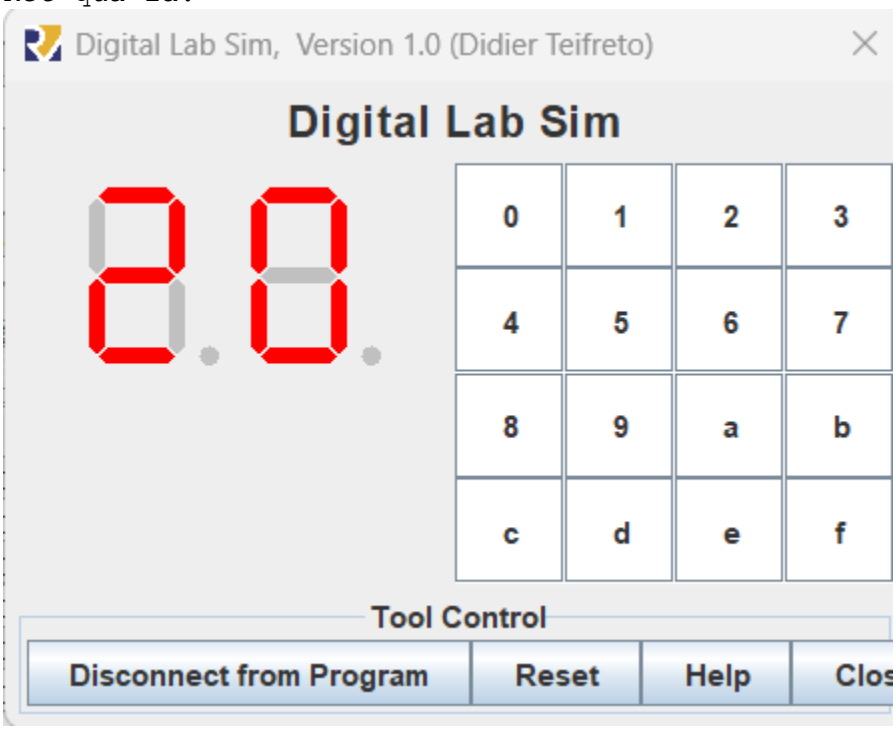
Kết quả là:



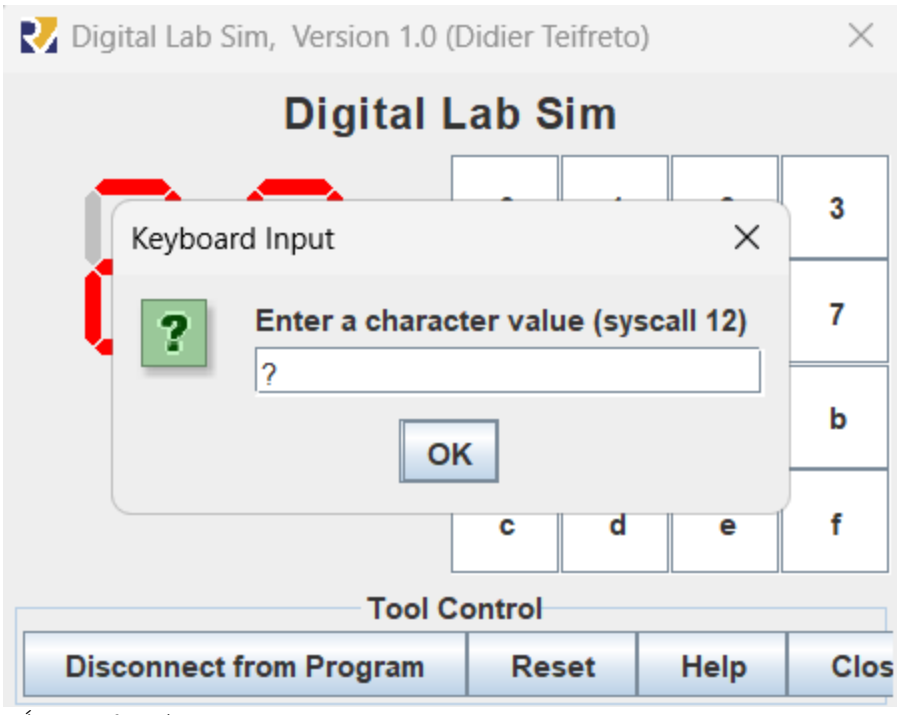
Nhập kí tự x:



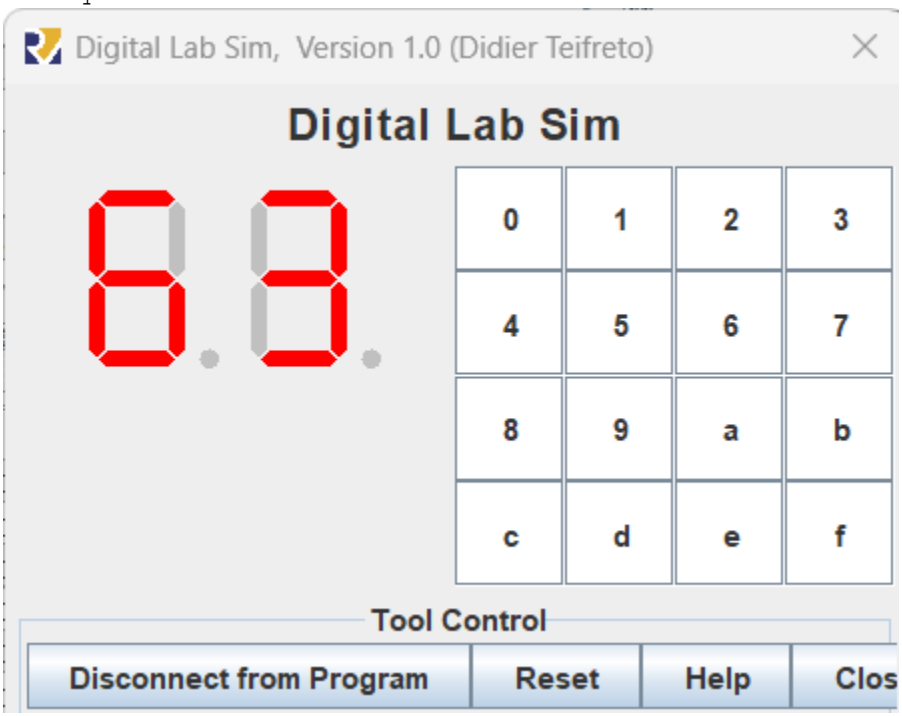
Kết quả là:



Nhập kí tự ?:



Kết quả là:



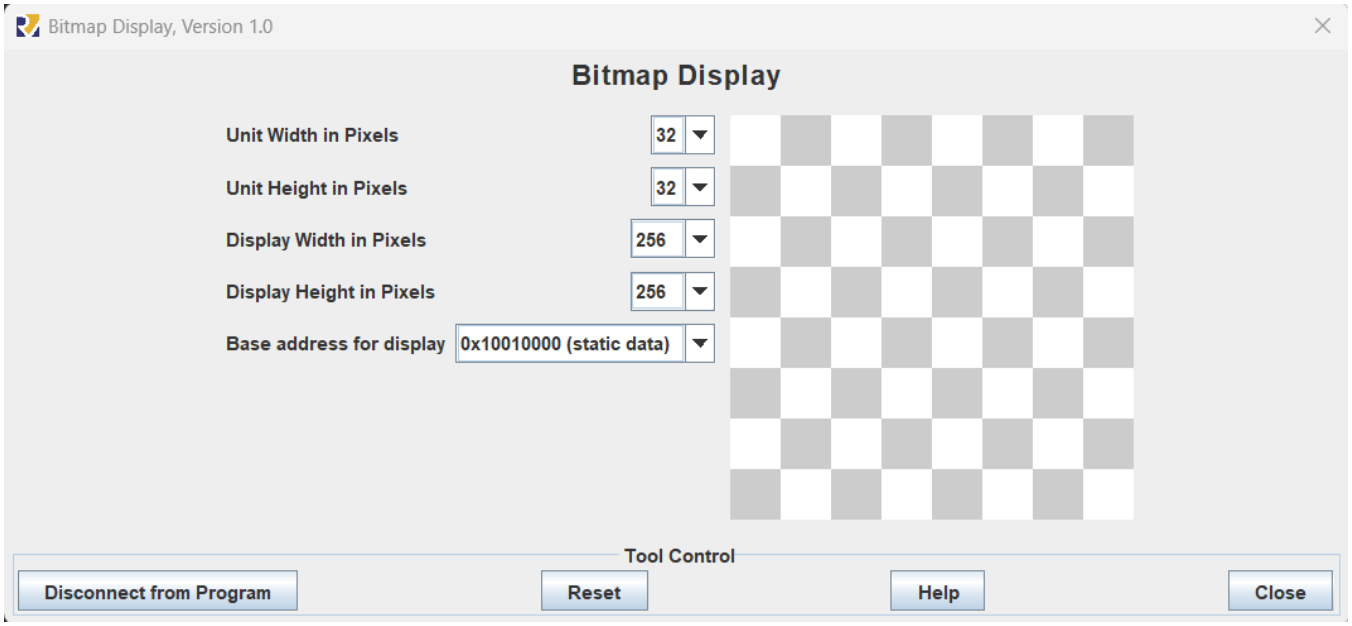
ASSIGNMENT 3

ĐOẠN MÃ :

```
.eqv MONITOR_SCREEN 0x10010000   # Địa chỉ bắt đầu của bộ nhớ màn hình
.eqv RED            0x00FF0000    # Các giá trị màu thường sử dụng
.eqv GREEN          0x0000FF00
.eqv BLUE           0x000000FF
.eqv WHITE          0x00FFFFFF
.eqv YELLOW         0x00FFFF00
.eqv GREY           0xCCCCCCCC    # Màu xám
```

```
.text
   # Khởi tạo các biến
   li t1, 0                    # Con trỏ pixel (đếm số ô đã xử lý)
   li a0, MONITOR_SCREEN       # Địa chỉ cơ sở của màn hình
   li s1, 2                    # Hằng số 2 dùng để kiểm tra chẵn/lẻ
   li s2, 8                    # Kích thước bàn cờ (8x8)
   li s3, 1                    # Biến đếm hàng (row), bắt đầu từ 1

# Vòng lặp chính - duyệt qua các hàng
loop2:
   bgt s3, s2, end             # Nếu đã xử lý hết 8 hàng thì kết thúc

   li s4, 1                    # Biến đếm cột (column), bắt đầu từ 1

# Vòng lặp con - duyệt qua các cột trong hàng hiện tại
loop1:
   bgt s4, s2, tiep            # Nếu đã xử lý hết 8 cột thì chuyển sang
hàng tiếp theo

   slli t2, t1, 2              # Tính offset bộ nhớ: t2 = t1 * 4 (mỗi
pixel 4 byte)
   add t2, t2, a0              # t2 = địa chỉ pixel hiện tại (a0 + offset)

   # Xác định màu cho ô hiện tại (trắng hoặc xám)
   add s5, s3, s4              # Tính tổng chỉ số hàng và cột
   rem s5, s5, s1              # Lấy phần dư khi chia cho 2 (kiểm tra
chẵn/lẻ)
   beq s5, zero, white         # Nếu chẵn (tổng hàng + cột chia hết cho 2)
-> màu trắng

# Trường hợp ô màu xám
grey:
   li t0, GREY                 # Nạp màu xám
   sw t0, 0(t2)                # Lưu màu vào vị trí pixel hiện tại

# Tiếp tục xử lý ô tiếp theo
continue:
   addi t1, t1, 1              # Tăng con trỏ pixel
   addi s4, s4, 1              # Tăng biến đếm cột
   j loop1                     # Lặp lại vòng lặp cột

# Trường hợp ô màu trắng
white:
   li t0, WHITE                # Nạp màu trắng
   sw t0, 0(t2)                # Lưu màu vào vị trí pixel hiện tại
   j continue                  # Quay lại tiếp tục vòng lặp

# Chuyển sang hàng tiếp theo
tiep:
   addi s3, s3, 1              # Tăng biến đếm hàng
   j loop2                     # Lặp lại vòng lặp hàng

# Kết thúc chương trình
end:
   li a7, 10                   # Gọi hệ thống để kết thúc chương trình
   ecall
```

Kết quả:



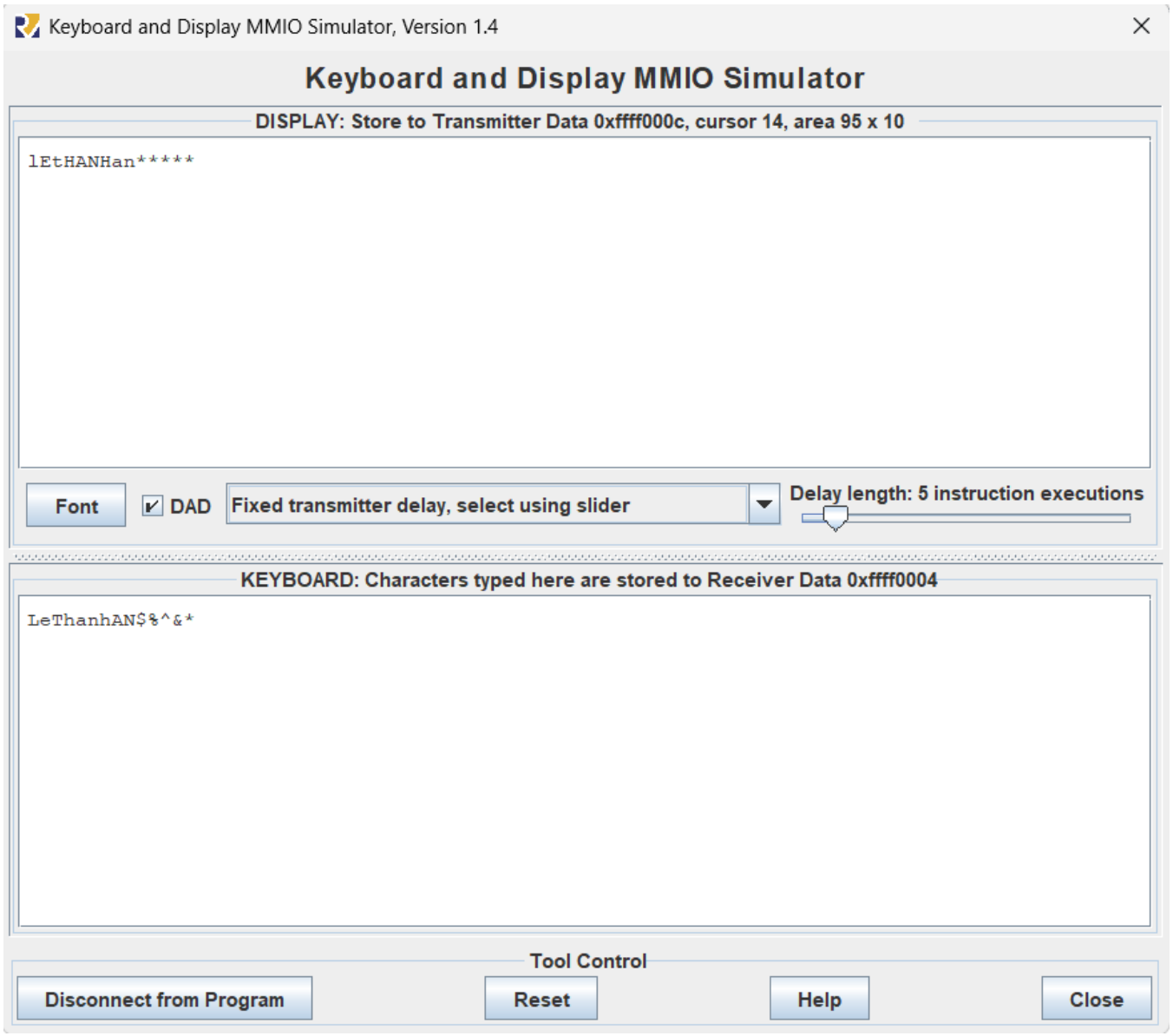
ASSIGNMENT 4

ĐOẠN MÃ :

```
.eqv KEY_CODE       0xFFFF0004   # ASCII code from keyboard, 1 byte
.eqv KEY_READY      0xFFFF0000   # =1 if has a new keycode (Auto clear after
lw)
.eqv DISPLAY_CODE   0xFFFF000C   # ASCII code to show, 1 byte
.eqv DISPLAY_READY  0xFFFF0008   # =1 if display is ready (Auto clear after sw)

.text
main:
    # Initialize port addresses
    li    a0, KEY_CODE
    li    a1, KEY_READY
    li    s0, DISPLAY_CODE
    li    s1, DISPLAY_READY

main_loop:
    # Wait for key press
    lw    t1, 0(a1)             # Check KEY_READY
    beqz  t1, main_loop         # Keep waiting if no key

    # Read key code
    lw    t0, 0(a0)             # t0 = ASCII character

    # Check lowercase letters (a-z)
    li    t3, 'a'
    blt   t0, t3, check_upper
    li    t3, 'z'
    bgt   t0, t3, check_upper
    addi  t0, t0, -32           # Convert to uppercase
    j     wait_display
```

```
check_upper:
    # Check uppercase letters (A-Z)
    li    t3, 'A'
    blt   t0, t3, check_digit
    li    t3, 'Z'
    bgt   t0, t3, check_digit
    addi  t0, t0, 32           # Convert to lowercase
    j     wait_display

check_digit:
    # Check digits (0-9)
    li    t3, '0'
    blt   t0, t3, other_char
    li    t3, '9'
    bgt   t0, t3, other_char
    j     wait_display          # Leave digits unchanged

other_char:
    li    t0, '*'              # Replace other chars with *

wait_display:
    # Wait until display is ready
    lw    t3, 0(s1)
    beqz  t3, wait_display

    # Display the character
    sw    t0, 0(s0)
    j     main_loop
```

Kết quả:


Keyboard and Display MMIO Simulator, Version 1.4
Keyboard and Display MMIO Simulator
DISPLAY: Store to Transmitter Data 0xffff000c, cursor 14, area 95 x 10
lEtHANHan*****
Font
DAD
Fixed transmitter delay, select using slider
Delay length: 5 instruction executions
KEYBOARD: Characters typed here are stored to Receiver Data 0xffff0004
LeThanhAN$%^&*
Tool Control
Disconnect from Program
Reset
Help
Close