**BÁO CÁO THỰC HÀNH**

**IT3280 – 156788 – THỰC HÀNH KIẾN TRÚC MÁY TÍNH**

**NỘI DUNG**

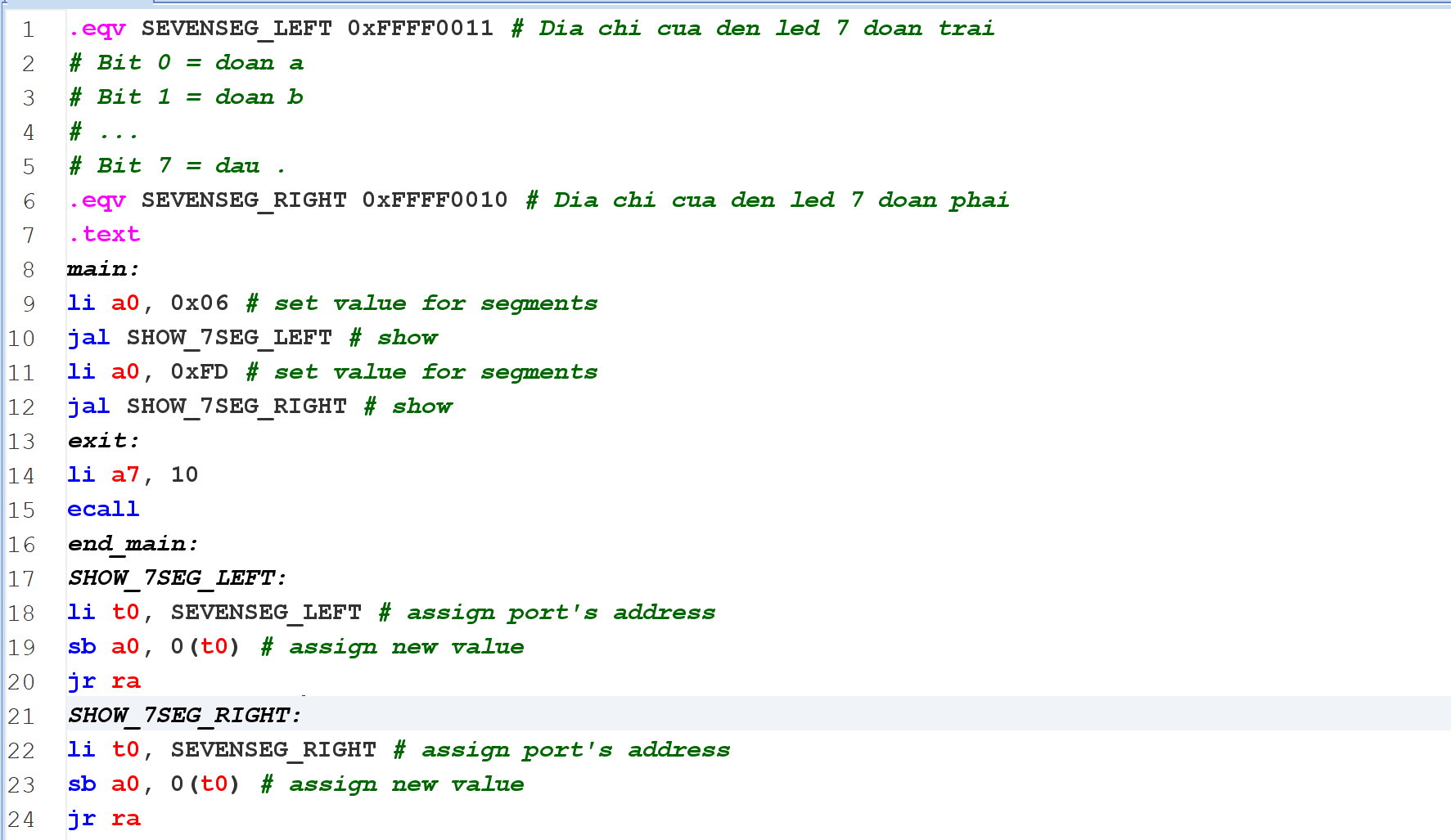
**Bài 10. Giao tiếp với các thiết bị ngoại vi**

|  |  |
| --- | --- |
| **Họ và tên** | Nguyễn Minh Quân |
| **Mã số sinh viên** | 20235816 |

**Assignment 1**

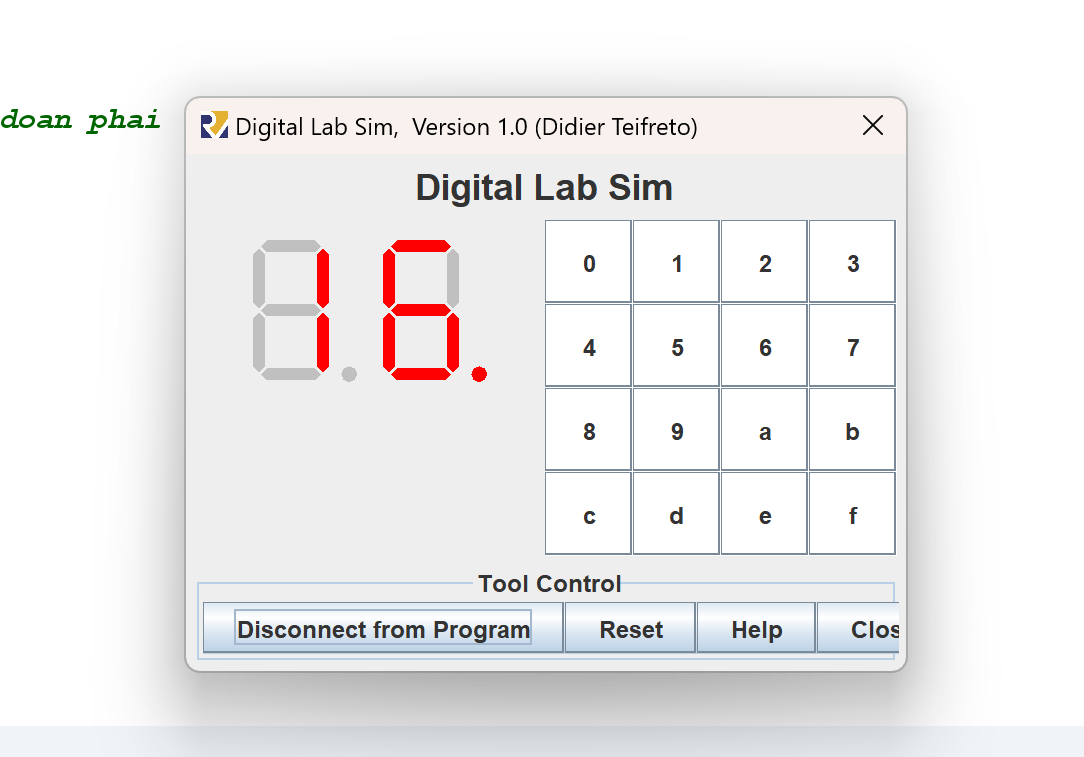
**Tạo project để thực hiện Home Assignment 1. Thay đổi các giá trị hiển thị trên LED 7 đoạn để hiển thị 2 chữ số cuối của MSSV.**

**Chương trình thực hiện :**

****

**Kết quả thu được :**

**MSSV : 20235816**

****

**Assignment 2**

**Tạo project để hiển thị trên LED 7 đoạn 2 chữ số cuối của mã ASCII (ở hệ cơ số 10) của ký tự được nhập từ bàn phím.**

**Chương trình thực hiện**

.eqv SEVENSEG\_LEFT 0xFFFF0011 # Địa chỉ của LED 7 đoạn bên trái

.eqv SEVENSEG\_RIGHT 0xFFFF0010 # Địa chỉ của LED 7 đoạn bên phải

.data

sevenseg\_map: # Bảng ánh xạ các số (0-9) tới mã nhị phân của LED 7 đoạn

.byte 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F

.text

main:

# Đọc ký tự từ bàn phím

li a7, 12 # Syscall để đọc một ký tự từ bàn phím

ecall # Kết quả trả về nằm trong thanh ghi a0

# Lấy 2 chữ số cuối của mã ASCII

li t0, 100

rem s0, a0, t0 # s0 = mã ASCII % 100 (lấy 2 chữ số cuối)

# Tách hàng chục và hàng đơn vị

li t0, 10

div t1, s0, t0 # t1 = hàng chục

rem t2, s0, t0 # t2 = hàng đơn vị

# Hiển thị số lên LED

blt s0, t0, oneDigit # Nếu số chỉ có 1 chữ số, nhảy tới oneDigit

jal twoDigits # Nếu là 2 chữ số, nhảy tới twoDigits

oneDigit:

li a0, 0x00 # Dọn dẹp LED bên trái

jal SHOW\_7SEG\_LEFT # Gọi hàm hiển thị LED bên trái

la t0, sevenseg\_map

add t0, t0, t2 # Tính địa chỉ cho số hàng đơn vị

lb a0, 0(t0) # Lấy mã nhị phân cho số hàng đơn vị

jal SHOW\_7SEG\_RIGHT # Hiển thị lên LED bên phải

j exit # Kết thúc

twoDigits:

la t0, sevenseg\_map

add t0, t0, t1 # Tính địa chỉ cho số hàng chục

lb a0, 0(t0) # Lấy mã nhị phân cho số hàng chục

jal SHOW\_7SEG\_LEFT # Hiển thị lên LED bên trái

la t0, sevenseg\_map

add t0, t0, t2 # Tính địa chỉ cho số hàng đơn vị

lb a0, 0(t0) # Lấy mã nhị phân cho số hàng đơn vị

jal SHOW\_7SEG\_RIGHT # Hiển thị lên LED bên phải

exit:

li a7, 10

ecall

SHOW\_7SEG\_LEFT:

li t0, SEVENSEG\_LEFT

sb a0, 0(t0)

jr ra

SHOW\_7SEG\_RIGHT:

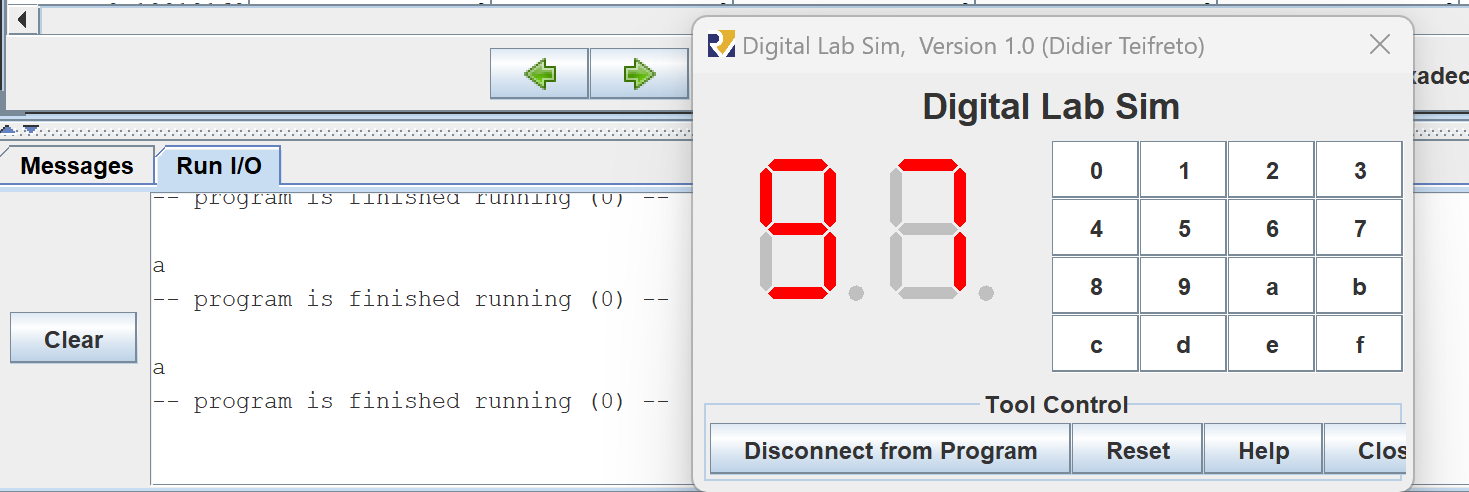
li t0, SEVENSEG\_RIGHT

sb a0, 0(t0)

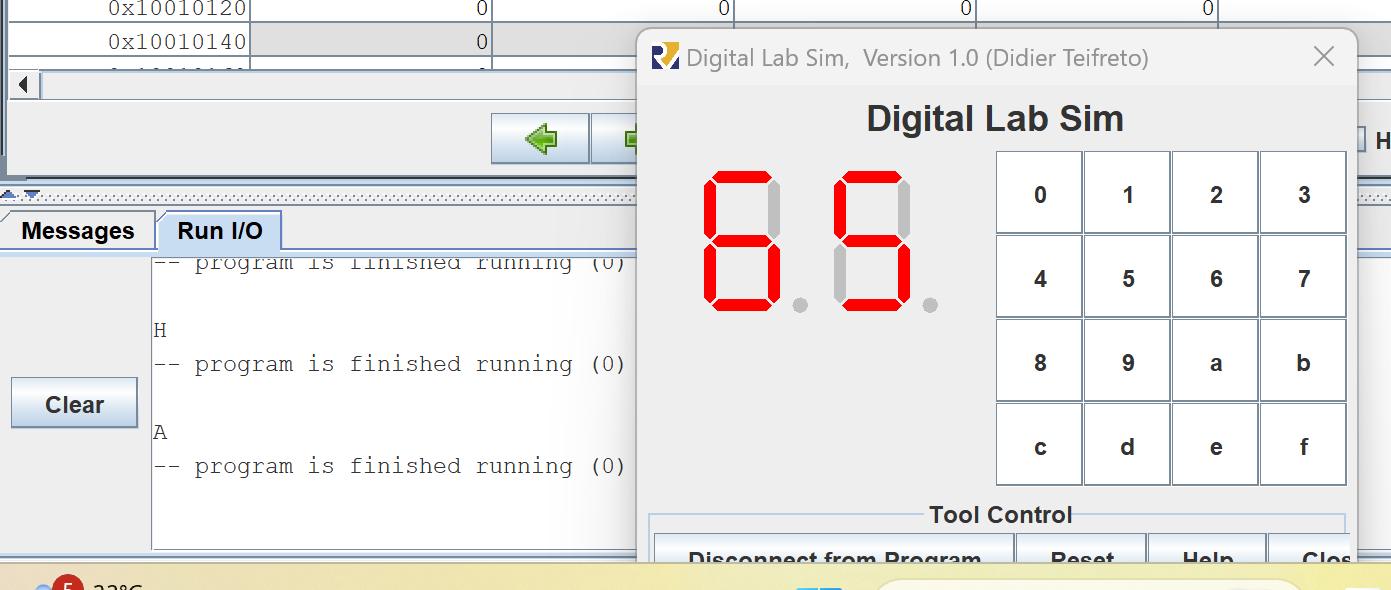
jr ra

**Kết quả thu được**

TH1: input ( a )



TH2: input (A)



**Assignment 3**

**Tạo project để thực hiện Home Assignment 2. Cập nhật mã nguồn để vẽ bàn cờ vua trên màn hình với 2 màu bất kỳ (khác màu đen).**

**Chương trình thực hiện:**

.eqv RED 0x00FF0000 # Định nghĩa màu ĐỎ (Red)

.eqv MONITOR\_SCREEN 0x10010000 # Địa chỉ bắt đầu của bộ nhớ màn hình

.eqv WHITE 0x00FFFFFF # Định nghĩa màu TRẮNG

.text

li a0, MONITOR\_SCREEN # Load địa chỉ bắt đầu màn hình vào a0

li t0, 0 # Khởi tạo chỉ số pixel (bắt đầu từ 0)

li t2, 8 # Kích thước cửa sổ: 8 hàng và 8 cột

li t3, 2 # Giá trị 2 (dùng cho phép chia modulo 2)

loop:

li t4, 64 # Tổng số pixel trong cửa sổ (8 \* 8 = 64)

bge t0, t4, exit # Nếu đã xử lý hết pixel (t0 >= 64), thoát vòng lặp

# Tính toán vị trí hàng và cột từ chỉ số pixel

div t1, t0, t2 # t1 = hàng = chỉ số pixel / 8

rem t4, t0, t2 # t4 = cột = chỉ số pixel % 8

# Xác định màu dựa trên tổng (hàng + cột) modulo 2

add t5, t1, t4 # t5 = tổng của hàng và cột

rem t5, t5, t3 # t5 = (hàng + cột) % 2

beqz t5, color\_white # Nếu (hàng + cột) % 2 == 0, chọn màu TRẮNG

j color\_pink # Ngược lại, chọn màu HỒNG

color\_white:

li t6, WHITE # Load mã màu TRẮNG vào t6

j set\_pixel # Nhảy đến đoạn code vẽ pixel

color\_pink:

li t6, RED # Load mã màu HỒNG vào t6

j set\_pixel # Nhảy đến đoạn code vẽ pixel

set\_pixel:

slli t5, t0, 2 # Tính offset byte: chỉ số pixel \* 4 (mỗi pixel 4 byte)

add t5, a0, t5 # Tính địa chỉ pixel trong bộ nhớ màn hình

sw t6, 0(t5) # Lưu mã màu vào địa chỉ bộ nhớ của pixel

addi t0, t0, 1 # Tăng chỉ số pixel để xử lý pixel tiếp theo

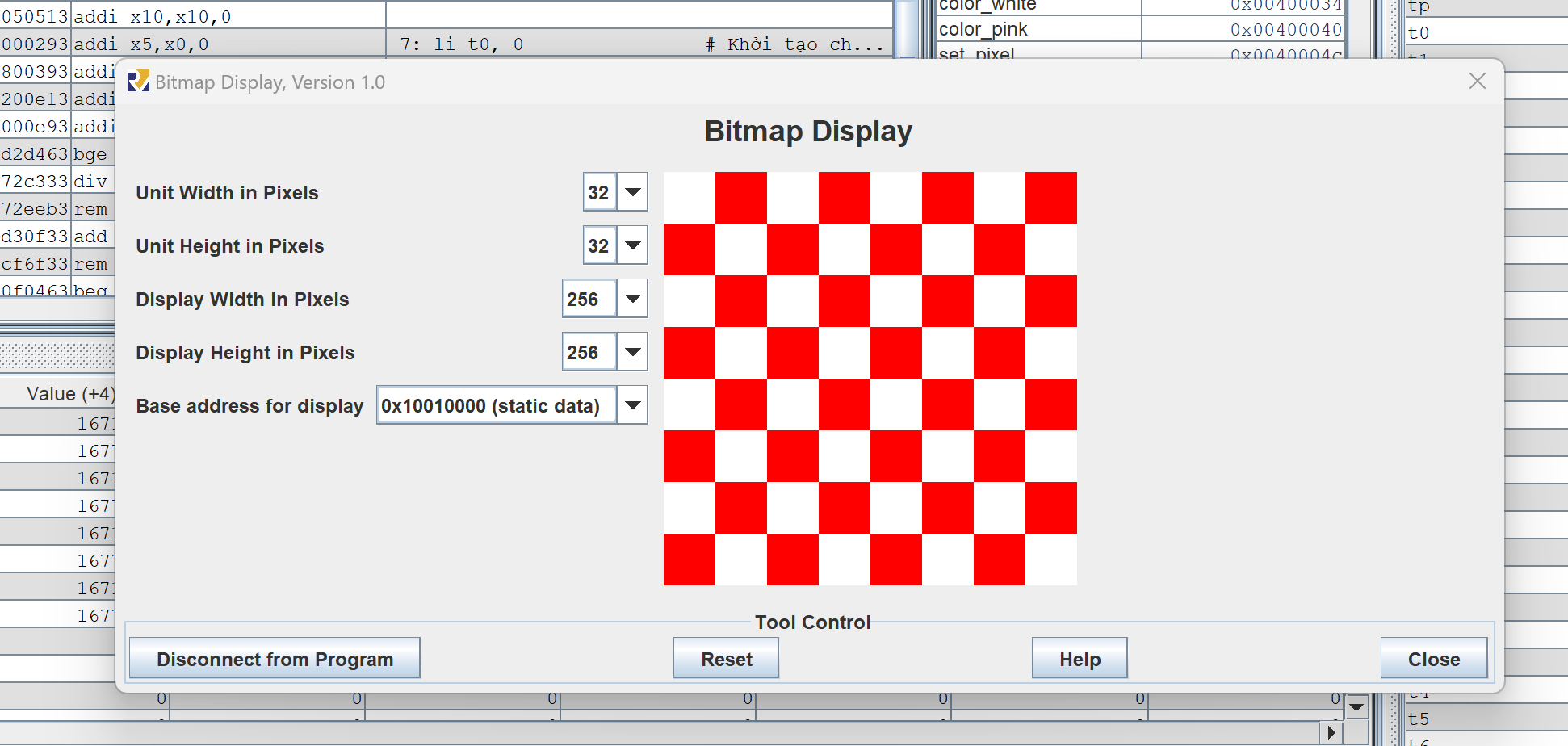
j loop # Quay lại vòng lặp

exit:

li a7, 10 # Gọi syscall để thoát chương trình

ecall

**Kết quả thu được :**

****

**Assignment 4**

**Tạo project để thực hiện Home Assignment 3. Cập nhật mã nguồn để hoàn thành yêu cầu sau: Nhập ký tự thường => hiển thị ký tự hoa tương ứng, nhập ký tự hoa => hiển thị ký tự thường tương ứng, nhập ký tự số thì giữ nguyên, nhập ký tự khác => hiển thị ký tự \*. Khi nhập chuỗi ký tự "exit" thì kết thúc chương trình.**

**Chương trình thực hiện:**

.data

exit\_str: .asciz "exit" # Chuỗi "exit" để so sánh

buffer: .space 5 # Vùng đệm chứa 4 ký tự nhập + ký tự kết thúc null

newline: .asciz "\n" # Ký tự xuống dòng

.text

.globl main

main:

# Khởi tạo

li s1, 0 # s1 = chỉ số hiện tại trong buffer

la s2, buffer # s2 = con trỏ tới buffer

main\_loop:

# Kiểm tra có phím nào được nhấn không

li t0, 0xFFFF0000 # Địa chỉ control của bàn phím

lw t1, 0(t0) # Đọc trạng thái

andi t1, t1, 1 # Kiểm tra bit sẵn sàng

beqz t1, main\_loop # Nếu không có input, tiếp tục kiểm tra

# Đọc ký tự từ bàn phím

li t0, 0xFFFF0004 # Địa chỉ data của bàn phím

lw a0, 0(t0) # Đọc ký tự

# Lưu ký tự vào buffer

sb a0, 0(s2) # Lưu ký tự vào vị trí hiện tại

addi s2, s2, 1 # Tăng con trỏ buffer

addi s1, s1, 1 # Tăng chỉ số buffer

# Xử lý ký tự

jal process\_char # Gọi hàm xử lý ký tự

# Kiểm tra nếu buffer đã có đủ 4 ký tự

li t0, 4

bne s1, t0, main\_loop # Nếu chưa đủ 4 ký tự, tiếp tục vòng lặp

# So sánh với chuỗi "exit"

la a0, buffer # Địa chỉ buffer

la a1, exit\_str # Địa chỉ chuỗi "exit"

jal strcmp # Gọi hàm so sánh chuỗi

beqz a0, exit\_program # Nếu bằng nhau thì thoát chương trình

# Dịch buffer (bỏ ký tự cũ nhất)

la s2, buffer # Đặt lại con trỏ buffer

lb t0, 1(s2) # Đọc ký tự thứ 2

sb t0, 0(s2) # Ghi vào vị trí đầu

lb t0, 2(s2) # Đọc ký tự thứ 3

sb t0, 1(s2) # Ghi vào vị trí thứ 2

lb t0, 3(s2) # Đọc ký tự thứ 4

sb t0, 2(s2) # Ghi vào vị trí thứ 3

li s1, 3 # Cập nhật chỉ số buffer = 3

addi s2, s2, 3 # Di chuyển con trỏ buffer

j main\_loop # Tiếp tục vòng lặp chính

process\_char:

# Kiểm tra nếu ký tự là chữ thường (a-z)

li t0, 'a'

li t1, 'z'

blt a0, t0, check\_upper # Nếu nhỏ hơn 'a', kiểm tra chữ hoa

bgt a0, t1, check\_digit # Nếu lớn hơn 'z', kiểm tra số

# Là chữ thường - chuyển thành chữ hoa

addi a0, a0, -32 # Trừ 32 để chuyển thành hoa

j display\_char

check\_upper:

# Kiểm tra nếu ký tự là chữ hoa (A-Z)

li t0, 'A'

li t1, 'Z'

blt a0, t0, check\_digit # Nếu nhỏ hơn 'A', kiểm tra số

bgt a0, t1, check\_digit # Nếu lớn hơn 'Z', kiểm tra số

# Là chữ hoa - chuyển thành chữ thường

addi a0, a0, 32 # Cộng 32 để chuyển thành thường

j display\_char

check\_digit:

# Kiểm tra nếu ký tự là số (0-9)

li t0, '0'

li t1, '9'

blt a0, t0, other\_char # Nếu nhỏ hơn '0', xử lý ký tự khác

bgt a0, t1, other\_char # Nếu lớn hơn '9', xử lý ký tự khác

# Là số - giữ nguyên

j display\_char

other\_char:

# Ký tự khác - thay bằng '\*'

li a0, '\*'

display\_char:

# Chờ cho đến khi màn hình sẵn sàng

li t0, 0xFFFF0008 # Địa chỉ control của màn hình

display\_wait:

lw t1, 0(t0) # Đọc trạng thái

andi t1, t1, 1 # Kiểm tra bit sẵn sàng

beqz t1, display\_wait # Nếu chưa sẵn sàng, tiếp tục chờ

# Hiển thị ký tự

li t0, 0xFFFF000C # Địa chỉ data của màn hình

sw a0, 0(t0) # Ghi ký tự ra màn hình

ret # Trở về từ hàm

strcmp:

# Hàm so sánh chuỗi

# a0 = địa chỉ chuỗi 1, a1 = địa chỉ chuỗi 2

# Trả về 0 trong a0 nếu bằng nhau

lb t0, 0(a0) # Đọc ký tự từ chuỗi 1

lb t1, 0(a1) # Đọc ký tự từ chuỗi 2

bne t0, t1, strcmp\_not\_equal # Nếu khác nhau, trả về 1

beqz t0, strcmp\_equal # Nếu gặp null terminator, trả về 0

addi a0, a0, 1 # Di chuyển tới ký tự tiếp theo chuỗi 1

addi a1, a1, 1 # Di chuyển tới ký tự tiếp theo chuỗi 2

j strcmp # Tiếp tục so sánh

strcmp\_equal:

li a0, 0 # Chuỗi bằng nhau

ret

strcmp\_not\_equal:

li a0, 1 # Chuỗi khác nhau

ret

exit\_program:

# Thoát chương trình

li a7, 10 # Syscall exit

ecall

**Kết quả thu được:**

