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:

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, số

Nội dung do AI tạo ra có thể không chính xác.

**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

Ảnh có chứa văn bản, ảnh chụp màn hình, màn hình, Phông chữ

Nội dung do AI tạo ra có thể không chính xác.

Kết quả là:

Ảnh có chứa văn bản, ảnh chụp màn hình, số, biểu đồ

Nội dung do AI tạo ra có thể không chính xác.

Nhập kí tự x:

Ảnh có chứa văn bản, ảnh chụp màn hình, màn hình, số

Nội dung do AI tạo ra có thể không chính xác.

Kết quả là:

Ảnh có chứa văn bản, ảnh chụp màn hình, số, Phông chữ

Nội dung do AI tạo ra có thể không chính xác.

Nhập kí tự ?:

Ảnh có chứa văn bản, ảnh chụp màn hình, màn hình, Phông chữ

Nội dung do AI tạo ra có thể không chính xác.

Kết quả là:

Ảnh có chứa văn bản, ảnh chụp màn hình, số, biểu đồ

Nội dung do AI tạo ra có thể không chính xác.

**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:

Ảnh có chứa văn bản, ảnh chụp màn hình

Nội dung do AI tạo ra có thể không chính xác.

**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:

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, màn hình

Nội dung do AI tạo ra có thể không chính xác.

**ASSIGNMENT 5:**

Đoạn mã:

.data

msg\_max: .asciz # Chuỗi để in giá trị lớn nhất

msg\_min: .asciz # Chuỗi để in giá trị nhỏ nhất

comma: .asciz # Chuỗi dấu phẩy

newline: .asciz # Chuỗi xuống dòng

.text

variable: # Nhãn khởi tạo các biến

addi a0, zero, 2 # a0 = 2

addi a1, zero, 6 # a1 = 6

addi a2, zero, 1 # a2 = 1

addi a3, zero, -3 # a3 = -3

addi a4, zero, 5 # a4 = 5

addi a5, zero, 3 # a5 = 3

addi a6, zero, 2 # a6 = 2

addi a7, zero, 9 # a7 = 9

main:

addi sp, sp, -32 # Dịch con trỏ stack xuống 32 byte (8 words)

sw a0, 0(sp) # Lưu a0 vào stack[0]

sw a1, 4(sp) # Lưu a1 vào stack[4]

sw a2, 8(sp) # Lưu a2 vào stack[8]

sw a3, 12(sp) # Lưu a3 vào stack[12]

sw a4, 16(sp) # Lưu a4 vào stack[16]

sw a5, 20(sp) # Lưu a5 vào stack[20]

sw a6, 24(sp) # Lưu a6 vào stack[24]

sw a7, 28(sp) # Lưu a7 vào stack[28]

li t6, 8 # t6 = 8 (số lượng phần tử)

addi a0, sp, 0 # Truyền địa chỉ mảng vào a0

jal find\_max\_min # Gọi hàm tìm max/min

li a7, 10 # Chuẩn bị syscall exit

ecall # Kết thúc chương trình

find\_max\_min:

lw s0, 0(a0) # Khởi tạo max = phần tử đầu (s0)

lw s2, 0(a0) # Khởi tạo min = phần tử đầu (s2)

li s1, 0 # Chỉ số của max (s1)

li s3, 0 # Chỉ số của min (s3)

li t0, 1 # Bộ đếm vòng lặp i = 1 (t0)

loop:

bge t0, t6, done # Nếu i >= số phần tử -> kết thúc

add t1, t0, t0 # t1 = i\*2

add t1, t1, t1 # t1 = i\*4 (offset phần tử thứ i)

add t2, a0, t1 # t2 = địa chỉ phần tử thứ i

lw t3, 0(t2) # t3 = giá trị phần tử thứ i

bge t3, s0, update\_max # Nếu t3 >= max -> cập nhật max

ble t3, s2, update\_min # Nếu t3 <= min -> cập nhật min

j next # Nhảy qua next

update\_max:

mv s0, t3 # Cập nhật giá trị max mới

mv s1, t0 # Cập nhật chỉ số max mới

j next # Tiếp tục vòng lặp

update\_min:

mv s2, t3 # Cập nhật giá trị min mới

mv s3, t0 # Cập nhật chỉ số min mới

next:

addi t0, t0, 1 # Tăng bộ đếm i++

j loop # Lặp lại

done:

jr ra # Trở về hàm gọi

Kết quả:

| STT | Vị trí | Thanh ghi | Giá trị thanh ghi | Giá trị vùng nhớ stack | Ghi chú chi tiết |
| --- | --- | --- | --- | --- | --- |
| 1 | variable: | pc | 0x00400000 |  | Bắt đầu chương trình |
| 2 | addi a0, zero, 2 | a0 | 0x00000002 |  | Khởi tạo a0 = 2 |
| 3 | addi a1, zero, 6 | a1 | 0x00000006 |  | Khởi tạo a1 = 6 |
| 4 | addi a2, zero, 1 | a2 | 0x00000001 |  | Khởi tạo a2 = 1 |
| 5 | addi a3, zero, -3 | a3 | 0xFFFFFFFD |  | Khởi tạo a3 = -3 |
| 6 | addi a4, zero, 5 | a4 | 0x00000005 |  | Khởi tạo a4 = 5 |
| 7 | addi a5, zero, 3 | a5 | 0x00000003 |  | Khởi tạo a5 = 3 |
| 8 | addi a6, zero, 2 | a6 | 0x00000002 |  | Khởi tạo a6 = 2 |
| 9 | addi a7, zero, 9 | a7 | 0x00000009 |  | Khởi tạo a7 = 9 |
| 10 | main: | pc | 0x00400020 |  | Bắt đầu hàm main |
| 11 | addi sp, sp, -32 | sp | 0x7fffefdc |  | Cấp phát 32 byte stack |
| 12 | sw a0, 0(sp) |  |  | [sp]=0x00000002 | Lưu a0 vào stack |
| 13 | sw a1, 4(sp) |  |  | [sp+4]=0x00000006 | Lưu a1 vào stack |
| 14 | sw a2, 8(sp) |  |  | [sp+8]=0x00000001 | Lưu a2 vào stack |
| 15 | sw a3, 12(sp) |  |  | [sp+12]=0xFFFFFFFD | Lưu a3 vào stack |
| 16 | sw a4, 16(sp) |  |  | [sp+16]=0x00000005 | Lưu a4 vào stack |
| 17 | sw a5, 20(sp) |  |  | [sp+20]=0x00000003 | Lưu a5 vào stack |
| 18 | sw a6, 24(sp) |  |  | [sp+24]=0x00000002 | Lưu a6 vào stack |
| 19 | sw a7, 28(sp) |  |  | [sp+28]=0x00000009 | Lưu a7 vào stack |
| 20 | li t6, 8 | t6 | 0x00000008 |  | Gán t6 = 8 (số phần tử) |
| 21 | addi a0, sp, 0 | a0 | 0x7fffefdc |  | Truyền địa chỉ mảng vào a0 |
| 22 | jal find\_max\_min | ra | 0x00400034 |  | Gọi hàm find\_max\_min, lưu return address |
| 23 | find\_max\_min: | pc | 0x00400040 |  | Bắt đầu hàm find\_max\_min |
| 24 | lw s0, 0(a0) | s0 | 0x00000002 |  | Khởi tạo max = phần tử đầu |
| 25 | lw s2, 0(a0) | s2 | 0x00000002 |  | Khởi tạo min = phần tử đầu |
| 26 | li s1, 0 | s1 | 0x00000000 |  | Chỉ số max ban đầu = 0 |
| 27 | li s3, 0 | s3 | 0x00000000 |  | Chỉ số min ban đầu = 0 |
| 28 | li t0, 1 | t0 | 0x00000001 |  | Bộ đếm vòng lặp i = 1 |
| 29 | loop: | pc | 0x00400054 |  | Bắt đầu vòng lặp |
| ... | ... | ... | ... | ... | ... (tiếp tục cho các vòng lặp) |
| 40 | done: | pc | 0x00400078 |  | Kết thúc hàm find\_max\_min |
| 41 | jr ra | pc | 0x00400034 |  | Trở về hàm main |
| 42 | li a7, 10 | a7 | 0x0000000A |  | Chuẩn bị syscall exit |
| 43 | ecall |  |  |  | Kết thúc chương trình |