Final Project

Group 17

Ex1:
Code:

```
.eqv KEY_CODE 0xFFFF0004  # Address for key code input
.eqv KEY_READY 0xFFFF0000  # Address to check if a key is ready

.eqv SCREEN_MONITOR 0x10010000  # Base address for screen monitor
.data
circle_end: .word 1       # End marker for circle data
data circle: .word        # Circle data buffer

.text
setup:
    # Initialize variables for circle movement and size
    addi s0, zero, 255     # x = 255 (initial x position)
    addi s1, zero, 255     # y = 255 (initial y position)
    add s2, zero, zero     # dx = 1 (x-direction velocity)
    addi s3, zero, -1      # dy = 0 (y-direction velocity)
    addi s4, zero, 20      # r = 20 (radius of the circle)
    addi a0, zero, 50      # t = 50ms/frame (frame delay)
    jal circle_data        # Generate circle pixel data

main_loop:
    # Main loop for the game
    jal input              # Check for key inputs
    jal move_circle        # Update circle position based on velocity
    jal draw_circle        # Draw the circle at the new position

    # Add delay for smooth animation
    li a7, 32              # ecall for sleep
    ecall

    j main_loop            # Repeat the loop

input:
    # Handle keyboard input
    li t0, KEY_READY       # Load KEY_READY address
    lw t1, 0(t0)           # Check if a key is ready
    bne t1, zero, process_key  # If a key is pressed, process it
```

```
    j edge_check          # Otherwise, check for edges

process_key:
    # Process key press to change direction
    li t0, KEY_CODE       # Load KEY_CODE address
    lw t1, 0(t0)          # Get the key code
    jal direction_change  # Change direction based on the key
    jal wait_for_key_clear # Wait until the key is cleared
    j edge_check          # Continue with edge checking

wait_for_key_clear:
    # Wait until the key press is cleared
    li t0, KEY_READY
clear_loop:
    lw t1, 0(t0)
    bne t1, zero, clear_loop
    jr ra

edge_check:
    # Check and handle boundary conditions
    li t2, 1
    li t3, -1

right:
    bne s2, t2, left_check
    j check_right
left_check:
    bne s2, t3, down_check
    j check_left
down_check:
    bne s3, t2, up_check
    j check_down
up_check:
    bne s3, t3, move_circle
    j check_up

move_circle:
    # Update position and redraw circle
    add s5, zero, zero
    jal draw_circle       # Clear the old circle
    add s0, s0, s2        # Update x position
    add s1, s1, s3        # Update y position
    li s5, 0x00FFFF00     # Set color for the circle
    jal draw_circle       # Draw the new circle
```

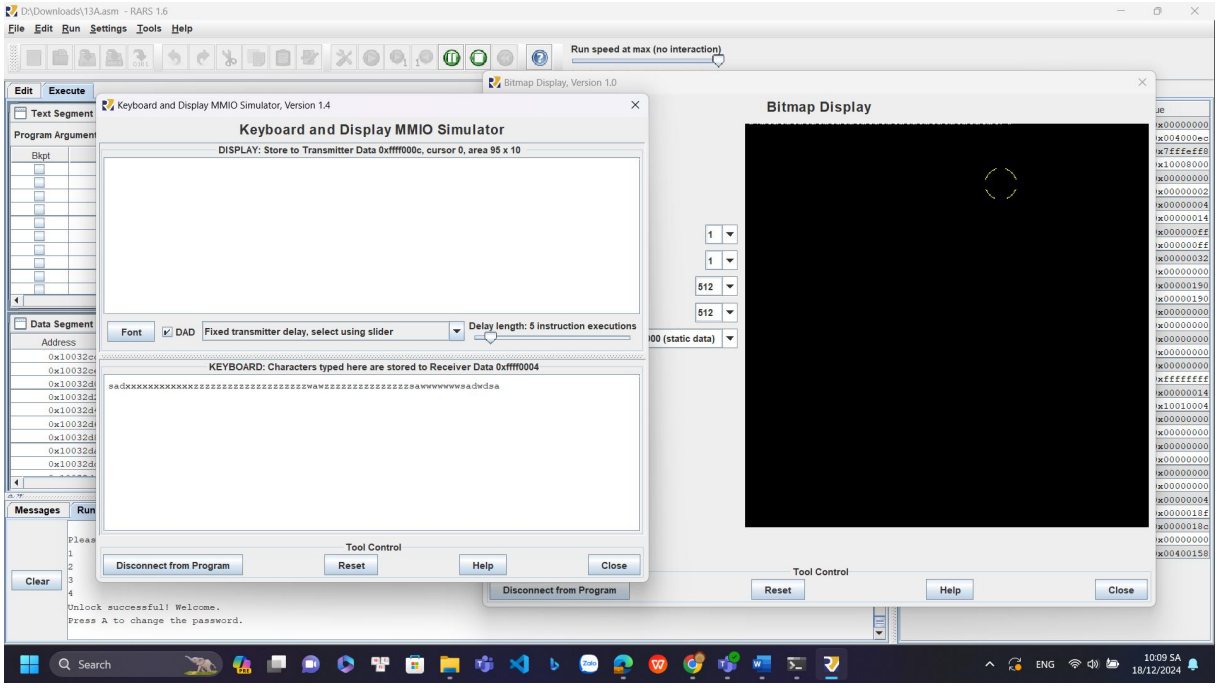
```
    loop:
    li a7, 32
    ecall
    j input

circle_data:
    # Generate pixel data for the circle
    addi sp, sp, -4
    sw ra, 0(sp)
    la s5, circle
    mul a3, s4, s4        # a3 = r^2
    add s7, zero, zero
pixel_data_loop:
    bgt s7, s4, data_end
    mul t0, s7, s7        # t0 = y^2
    sub a2, a3, t0        # a2 = r^2 - y^2
    jal root              # Compute sqrt(r^2 - y^2)
    add a1, zero, s7
    add s6, zero, zero
symmetric:
    # Save pixels for symmetry
    li t3, 2
    beq s6, t3, finish
    jal pixel_save
    sub a1, zero, a1
    jal pixel_save
    sub a2, zero, a2
    jal pixel_save
    sub a1, zero, a1
    jal pixel_save
    add t0, zero, a1
    add a1, zero, a2
    add a2, zero, t0
    addi s6, s6, 1
    j symmetric
finish:
    addi s7, s7, 1
    j pixel_data_loop
data_end:
    la t0, circle_end
    sw s5, 0(t0)
    lw ra, 0(sp)
    addi sp, sp, 4
    jr ra
```

```
root:
    # Approximation for square root
    add t0, zero, zero
    add t1, zero, zero
root_loop:
    li t2, 20
    beq t0, t2, root_end
    addi t3, t0, 1
    mul t3, t3, t3
    sub t4, a2, t1
    bgez t4, continue
    sub t4, zero, t4
continue:
    sub t5, a2, t3
    bgez t5, compare
    sub t5, zero, t5
compare:
    blt t5, t4, root_continue
    add a2, zero, t0
    jr ra
root_continue:
    addi t0, t0, 1
    add t1, zero, t3
    j root_loop
root_end:
    add a2, zero, t0
    jr ra

pixel_save:
    # Save a pixel coordinate pair
    sw a1, 0(s5)
    sw a2, 4(s5)
    addi s5, s5, 8
    jr ra

direction_change:
    # Change direction based on key press
    li t0, KEY_CODE
    lw t1, 0(t0)
    li t2, 'd'
    li t3, 'a'
    li t4, 's'
    li t5, 'w'
    li t6, 'x'   # Extra input for functionality
```

```
    li t0, 'z'   # Reverse speed functionality

case_d:
    bne t1, t2, case_a
    li s2, 1            # dx = 1
    li s3, 0            # dy = 0
    jr ra
case_a:
    bne t1, t3, case_s
    li s2, -1           # dx = -1
    li s3, 0
    jr ra
case_s:
    bne t1, t4, case_w
    li s2, 0            # dx = 0
    li s3, 1            # dy = 1
    jr ra
case_w:
    bne t1, t5, case_x
    li s2, 0
    li s3, -1           # dy = -1
    jr ra
case_x:
    bne t1, t6, case_z
    addi a0, a0, 10     # Increase speed
    jr ra
case_z:
    bne t1, t0, default
    beq a0, zero, default
    addi a0, a0, -10    # Decrease speed

default:
    jr ra

check_right:
    # Handle collision with right boundary
    add t0, s0, s4
    li t1, 511
    beq t0, t1, reverse_direction
    j move_circle
check_left:
    # Handle collision with left boundary
    sub t0, s0, s4
    beq t0, zero, reverse_direction
    j move_circle
```

```
check_down:
    # Handle collision with bottom boundary
    add t0, s1, s4
    li t1, 511
    beq t0, t1, reverse_direction
    j move_circle
check_up:
    # Handle collision with top boundary
    sub t0, s1, s4
    blez t0, reverse_direction
    j move_circle

reverse_direction:
    # Reverse velocity when hitting boundaries
    sub s2, zero, s2
    sub s3, zero, s3
    j move_circle

draw_circle:
    # Draw the circle at the current position
    addi sp, sp, -4
    sw ra, 0(sp)
    la s6, circle_end
    lw s7, 0(s6)
    la s6, circle
draw_loop:
    beq s6, s7, draw_end
    lw a1, 0(s6)
    lw a2, 4(s6)

    # Bounds checking for screen edges
    add t1, s0, a1
    blt t1, zero, skip_pixel
    li t2, 511
    bgt t1, t2, skip_pixel

    add t3, s1, a2
    blt t3, zero, skip_pixel
    li t4, 511
    bgt t3, t4, skip_pixel

    jal pixel_draw
skip_pixel:
    addi s6, s6, 8
    j draw_loop
```

```
draw_end:
    lw ra, 0(sp)
    addi sp, sp, 4
    jr ra

pixel_draw:
    # Draw a single pixel on the screen
    li t0, SCREEN_MONITOR
    add t1, s0, a1
    add t2, s1, a2
    slli t2, t2, 9
    add t2, t2, t1
    slli t2, t2, 2
    add t0, t0, t2
    sw s5, 0(t0)

    jr ra
```



Constants and Data Section

```
.eqv KEY_CODE 0xFFFF0004
.eqv KEY_READY 0xFFFF0000
.eqv SCREEN_MONITOR 0x10010000
.data

circle_end: .word 1
circle: .word
```

1. `KEY_CODE` and `KEY_READY`: Define the memory-mapped I/O addresses for reading keyboard input.
    - `KEY_READY`: Indicates whether a keypress is available.
    - `KEY_CODE`: Stores the ASCII code of the pressed key.
2. `SCREEN_MONITOR`: Base address for the virtual screen, where pixels are drawn.
3. `circle` and `circle_end`: Memory locations for storing the precomputed offsets of the circle's shape.

Setup Section

```
setup:
addi s0, zero, 255 # x = 255
addi s1, zero, 255 # y = 255
add s2, zero, zero   # dx = 1
addi s3, zero, -1 # dy = 0
addi s4, zero, 20  # r = 20
addi a0, zero, 50  # t = 50ms/frame
jal circle_data
```

- **Initialization of Variables**:
    - `s0` and `s1`: Set the circle's starting position to the screen center (255, 255).
    - `s2` and `s3`: Direction vectors (`dx`, `dy`) initialized to move diagonally down-left.
    - `s4`: Circle radius, set to 20 pixels.
    - `a0`: Frame delay, set to 50ms.
- **Precomputing Circle Data**:
    - The `circle_data` subroutine computes the circle's pixel offsets relative to its center based on its radius (`r`).
    - These offsets are stored in memory to speed up the drawing process.

Main Loop

```
main_loop:
    jal input            # Check for key presses
    jal move_circle      # Update position based on direction
    jal draw_circle      # Draw the circle at the new position

    li a7, 32            # ecall for sleep
    ecall
    j main_loop          # Repeat the loop
```

- The `main_loop` governs the program's flow:
    1. **Input Handling** (`input`): Processes user input to adjust direction or speed.
    2. **Movement Update** (`move_circle`): Updates the circle's position based on current direction.
    3. **Drawing** (`draw_circle`): Redraws the circle at its updated position.
    4. **Delay**: Introduces a frame delay to control animation speed.

The loop repeats indefinitely, simulating continuous animation.

Input Handling

Input Subroutine

```
input:
    li t0, KEY_READY    # Load the address of KEY_READY
    lw t1, 0(t0)        # Read KEY_READY value
    bne t1, zero, process_key
    j edge_check
```

- Reads the value of `KEY_READY`. If a key is pressed (`KEY_READY != 0`), it jumps to `process_key`.
- Otherwise, it skips to the `edge_check` routine.

Key Processing

```
process_key:
    li t0, KEY_CODE     # Load the address of KEY_CODE
    lw t1, 0(t0)        # Load the ASCII key code
    jal direction_change
    jal wait_for_key_clear
    j edge_check
```

- Reads the ASCII code of the pressed key from `KEY_CODE`.
- Calls `direction_change` to adjust the circle's direction or speed based on the input.
- Waits for the key to be cleared (`KEY_READY = 0`) using `wait_for_key_clear`.

Direction Control

Direction Change Subroutine

```
direction_change:
    li t2, 'd'
    li t3, 'a'
    li t4, 's'
    li t5, 'w'
    li t6, 'x'
    li t0, 'z'
```

- Maps keys (`w`, `a`, `s`, `d`) to specific movement directions:
    - `w`: Move up (`dy = -1`).
    - `a`: Move left (`dx = -1`).
    - `s`: Move down (`dy = 1`).
    - `d`: Move right (`dx = 1`).
- `z` and `x`: Adjust animation speed (`a0`):
    - `z`: Decrease speed (increase delay).
    - `x`: Increase speed (decrease delay).

Movement Update

```
move_circle:
```

```
add s0, s0, s2    # Update X position
add s1, s1, s3    # Update Y position
j draw_circle
```

- Updates the circle's position by adding `dx` (`s2`) to `x` (`s0`) and `dy` (`s3`) to `y` (`s1`).

Edge Checking

```
check_right:
add t0, s0, s4
li t1, 511
beq t0, t1, reverse_direction
```

- Checks whether the circle's edge exceeds the screen boundary (511x511).
- If a boundary collision occurs, the direction is reversed using `reverse_direction`.

Drawing the Circle

Circle Data Precomputation

```
circle_data:
mul a3, s4, s4       # Calculate r^2
pixel_data_loop:
# Compute offsets for all pixels within the circle
```

- Computes offsets of circle pixels relative to its center using the equation x2+y2=r2x^2 + y^2 = r^2x2+y2=r2.
- Uses symmetry to store only one-eighth of the circle's offsets, then mirrors them.

Drawing Subroutine

```
draw_circle:
lw a1, 0(s6)         # Load x offset
lw a2, 4(s6)         # Load y offset
pixel_draw:
add t1, s0, a1       # Calculate absolute X coordinate
add t3, s1, a2       # Calculate absolute Y coordinate
```

- Iterates through the precomputed offsets and computes the absolute pixel positions.
- Ensures pixels fall within the screen bounds (0 to 511).
- Draws valid pixels by writing to the `SCREEN_MONITOR` memory-mapped region.

Ex10:

Code:

```
.eqv IN_ADDRESS_HEXA_KEYBOARD 0xFFFF0012

.eqv OUT_ADDRESS_HEXA_KEYBOARD 0xFFFF0014

.eqv SEVENSEG_LEFT 0xFFFF0011        # Address of the LED on the left

# Bit 0 = segment a

# Bit 1 = segment b

# ...

# Bit 7 = dot sign

.eqv SEVENSEG_RIGHT 0xFFFF0010       # Address of the LED on the right

.data

password: .byte 1, 2, 3, 4             # Mật khẩu ban đầu

buffer_password: .space 32

len: .word 4

buffer: .space 32                  # Lưu mật khẩu tối đa 16 ký tự

index: .word 0                     # Chỉ số hiện tại trong buffer, nơi lưu mật khẩu nhập vào

wrong_attempts: .word 0            # Biến đếm số lần nhập sai mật khẩu

msg_enter_password: .asciz "Please enter your password:¥n"

msg_unlock_success: .asciz "Unlock successful! Welcome.¥n"

msg_enter_old_password: .asciz "Please enter your old password:¥n"

msg_enter_new_password: .asciz "Please enter your new password:¥n"

msg_password_updated: .asciz "Password has been successfully updated.¥n"

msg_password_wrong: .asciz "Incorrect password. Please try again.¥n"

msg_lock_suspended: .asciz "Too many incorrect attempts. Lock is suspended for 1
minute.¥n"
```

```
msg_press_A_to_change: .asciz "Press A to change the password.¥n"
msg_short_password: .asciz "Password is too short. It must be at least 4 characters long.¥n"
.text
main:
    li t1, IN_ADDRESS_HEXA_KEYBOARD   # Input address for row assignment
    li t2, OUT_ADDRESS_HEXA_KEYBOARD  # Output address for reading key pressed
start:
    la s0, buffer                 # Điểm bắt đầu của buffer
    la s1, index                  # Địa chỉ lưu chỉ số
loop_main:
    # Display the message to enter the password
    la a0, msg_enter_password
    jal print_message
    # Read and check password
    jal READ_PASSWORD
    jal CHECK_PASSWORD
    # If password is correct, show LED
    beqz a0, handle_password_incorrect  # Nếu a0 = 0, mật khẩu sai
    la t6, wrong_attempts
    sw zero, 0(t6)                # Khôi phục số lần nhập sai về 0
    la a0, msg_unlock_success
    jal print_message
    jal LED_ON
    la a0, msg_press_A_to_change
    jal print_message
    j wait_for_A
```

```
end_main:
    li a7, 10
    ecall
handle_password_incorrect:
    la t6, wrong_attempts
    lw t4, 0(t6)
    addi t4, t4, 1
    sw t4, 0(t6)
    la a0, msg_password_wrong
    jal print_message
    # Nếu đã nhập sai quá 3 lần, tạm ngưng chương trình
    li t6, 3
    bge t4, t6, lock_suspended
    jal LED_OFF
    j loop_main
lock_suspended:
    # Hiển thị thông báo tạm ngưng khóa
    la a0, msg_lock_suspended
    jal print_message
# Tạm dừng mọi nút bấm trong 1 phút (60 giây)
wait_1_minute:
    li a0, 60000                # Giả lập thời gian đợi 1 phút
    li a7, 32
    ecall
    # Sau 1 phút, khôi phục lại số lần nhập sai
    la t6, wrong_attempts
```

```
    sw zero, 0(t6)
    j loop_main
print_message:
    addi sp, sp, -4
    sw a0, 0(sp)                # Lưu giá trị của a0 (địa chỉ chuỗi)
    li a7, 4                    # Syscall: Print string
    ecall                       # In chuỗi
    lw a0, 0(sp)
    addi sp, sp, 4
    jr ra
back_to_program:
    jr ra
# ------------------------------------------------------------
# Function LED_ON: Bật LED (Hiển thị tất cả các segment)
# ------------------------------------------------------------
LED_ON:
    li t0, SEVENSEG_LEFT        # Địa chỉ LED trái
    li a0, 0x7F                 # LED = 8 (Hiển thị tất cả các segment)
    sb a0, 0(t0)                # Ghi giá trị vào LED trái
    li t0, SEVENSEG_RIGHT       # Địa chỉ LED phải
    li a0, 0x7F                 # LED = 8 (Hiển thị tất cả các segment)
    sb a0, 0(t0)                # Ghi giá trị vào LED phải
    jr ra                       # Quay lại gọi hàm
# ------------------------------------------------------------
# Function LED_OFF: Tắt LED (Tắt tất cả các segment)
# ------------------------------------------------------------
```

```
LED_OFF:
    li t0, SEVENSEG_LEFT            # Địa chỉ LED trái
    li a0, 0x3F
    sb a0, 0(t0)                    # Ghi giá trị vào LED trái
    li t0, SEVENSEG_RIGHT           # Địa chỉ LED phải
    li a0, 0x71
    sb a0, 0(t0)                    # Ghi giá trị vào LED phải
    jr ra                           # Quay lại gọi hàm
# ------------------------------------------------------------
# Function READ_PASSWORD : Read password input from keypad
# ------------------------------------------------------------
READ_PASSWORD:
    li t5, 0x0
    la s0, buffer
    la s1, index
    sw t5, 0(s1)
loop:
    li t3, 0x1                      # Quét từ row 0x1 (row đầu tiên)
scan_rows:
    sb t3, 0(t1)                    # Gán giá trị hàng hiện tại
    lb a0, 0(t2)                    # Đọc mã nút bấm
    beqz a0, next_row               # Không có nút nào được bấm -> kiểm tra hàng tiếp theo
    li t4, 0x88                     # Kiểm tra nếu nút bấm là F
    andi a1, a0, 0xFF
    beq a1, t4, back_to_program     # Nếu nút F được bấm, quay lại luồng chương trình để
kiểm tra mật khẩu
    addi sp, sp, -4
```

```
    sw ra, 0(sp)
    jal decode
    lw ra, 0(sp)
    addi sp, sp, 4
    # Kiểm tra nếu mã phím không hợp lệ
    li t4, 0xFF
    beq a0, t4, invalid_key          # Nếu a0 = 0xFF, gọi đến nhãn invalid_key
    sb a0, 0(s0)                     # Lưu mã nút bấm vào buffer
    addi s0, s0, 1                   # Tăng địa chỉ buffer
    lw t5, 0(s1)                     # Đọc chỉ số hiện tại
    addi t5, t5, 1                   # Tăng chỉ số
    sw t5, 0(s1)                     # Lưu lại chỉ số mới
    addi sp, sp, -4
    sw ra, 0(sp)
    jal print_number
    lw ra, 0(sp)
    addi sp, sp, 4
    sb zero, 0(t2)
    li a0, 100                       # Sleep 100ms (debounce)
    li a7, 32
    ecall
next_row:
    slli t3, t3, 1                   # Chuyển sang row tiếp theo (0x1 -> 0x2 -> 0x4 -> 0x8)
    li t4, 0x10                      # Hết tất cả các row (sau 0x8)
    blt t3, t4, scan_rows            # Tiếp tục quét nếu còn row
    j loop                           # Quay lại vòng lặp chính
```

```
decode:                 # Chuyển đổi số nhập vào từ vị trí nhận được khi nhập
    andi a1, a0, 0xFF
    li t4, 0x11
    li t6, 0x0
    beq a1, t4, save
    li t4, 0x21
    li t6,0x1
    beq a1, t4, save
    li t4, 0x41
    li t6, 0x2
    beq a1, t4, save
    li t4, 0x81
    li t6, 0x3
    beq a1, t4, save
    li t4, 0x12
    li t6, 0x4
    beq a1, t4, save
    li t4, 0x22
    li t6, 0x5
    beq a1, t4, save
    li t4, 0x42
    li t6, 0x6
    beq a1, t4, save
    li t4, 0x82
    li t6, 0x7
    beq a1, t4, save
```

```
    li t4, 0x14
    li t6, 0x8
    beq a1, t4, save
    li t4, 0x24
    li t6, 0x9
    beq a1, t4, save
    # Nếu không khớp bất kỳ phím nào, trả về 0xFF để báo lỗi
    li a0, 0xFF                 # Đặt a0 = 0xFF khi không khớp phím nào
    jr ra                       # Quay lại
invalid_key:
    sb zero, 0(t2)
    j loop
save:
    add a0, zero, t6
    jr ra
print_number:
    addi sp, sp, -4
    sw a0, 0(sp)
    li a7, 1                    # Syscall: Print integer
    ecall                       # In giá trị trong a0
    # In ký tự xuống dòng
    li a0, 10                   # Mã ASCII của '¥n'
    li a7, 11                   # Syscall: Print character
    ecall
    lw a0, 0(sp)
    addi sp, sp, 4
```

```
    jr ra
# ------------------------------------------------------------
# Function CHECK_PASSWORD: Compare entered password with stored password
# ------------------------------------------------------------
CHECK_PASSWORD:
    la s0, buffer                  # Bắt đầu đọc buffer
    la s1, index                   # Bắt đầu đọc mật khẩu lưu trữ
    la s2, password
    lw t5, 0(s1)
    addi sp, sp, -4
    sw t5, 0(sp)
    la t6, len
    lw t6, 0(t6)                   # Đặt chiều dài mật khẩu tối thiểu
    bne t5, t6, password_incorrect    # Nếu số ký tự nhập vào khác số kí tự của mật khẩu -> sai
compare_loop:
    beqz t5, password_correct        # Nếu không còn ký tự, mật khẩu đúng
    lb a0, 0(s0)                   # Lấy ký tự từ buffer
    lb a1, 0(s2)                   # Lấy ký tự từ password
    bne a0, a1, password_incorrect    # Nếu khác nhau, mật khẩu sai
    addi s0, s0, 1                 # Tăng địa chỉ buffer
    addi s2, s2, 1                 # Tăng địa chỉ password
    addi t5, t5, -1                # Giảm số lượng ký tự còn lại
    j compare_loop                 # Quay lại so sánh ký tự tiếp theo
password_correct:
    li a0, 1
    j recover
```

```
password_incorrect:
    li a0, 0
    j recover
recover:
    lw t5, 0(sp)
    sw t5, 0(s1)
    addi sp, sp, 4
    j back_to_program
# wait for change password
wait_for_A:
    li t3, 0x1                  # Quét từ row 0x1 (row đầu tiên)
scan_rows_A:
    sb t3, 0(t1)                # Gán giá trị hàng hiện tại
    lb a0, 0(t2)                # Đọc mã nút bấm
    beqz a0, next_row_A         # Không có nút nào được bấm -> kiểm tra hàng tiếp theo
    li t4, 0x44                 # Mã phím A
    andi a1, a0, 0xFF
    beq a1, t4, change_password # Nếu phím A được bấm, xử lý đổi mật khẩu
    sb zero, 0(t2)
    li a0, 100                  # Sleep 100ms (debounce)
    li a7, 32
    ecall
next_row_A:
    slli t3, t3, 1              # Chuyển sang hàng tiếp theo
    li t4, 0x10                 # Hết tất cả các hàng
    blt t3, t4, scan_rows_A     # Tiếp tục quét nếu còn hàng
```

```
    j wait_for_A                    # Quay lại quét hàng đầu tiên
change_password:
    # Chuyển đến hàm đổi mật khẩu
    jal LED_OFF
    jal CHANGE_PASSWORD
    j start                         # Quay lại vòng lặp chính
# ---------------------------------------------------------------
# Function CHANGE_PASSWORD: Handle changing the password
# ---------------------------------------------------------------
CHANGE_PASSWORD:
    li a2, 0                        # Số lần nhập sai mật khẩu
    li a3, 3                                # Cho phép nhập mật khẩu sai tối đa bao nhiêu lần
    addi sp, sp -4
    sw ra, 0(sp)
retry_old_password:
    # Hiển thị thông báo nhập mật khẩu cũ
    la a0, msg_enter_old_password
    jal print_message
    # Đọc mật khẩu cũ từ người dùng
    jal READ_PASSWORD
    # Kiểm tra mật khẩu cũ
    jal CHECK_PASSWORD
    beqz a0, change_password_wrong      # Nếu a0 = 0, mật khẩu sai
update_new_password:
    # Hiển thị thông báo nhập mật khẩu mới
    la a0, msg_enter_new_password
```

```
    jal print_message
    # Đọc mật khẩu mới từ người dùng
    jal READ_PASSWORD
    la s0, buffer                # Địa chỉ của buffer chứa mật khẩu mới
    la s1, index
    lw t5, 0(s1)                 # Lấy chiều dài mật khẩu mới từ index
    li t6, 4
    blt t5, t6, password_too_short
    # Lưu mật khẩu mới vào vùng nhớ password
    la s2, password              # Địa chỉ của password cũ
    la t6, len
    sw t5, 0(t6)                 # Cập nhật chiều dài mới vào len
copy_new_password:
    beqz t5, password_updated    # Nếu không còn ký tự, cập nhật xong
    lb a0, 0(s0)                 # Lấy từng ký tự từ buffer
    sb a0, 0(s2)                 # Ghi vào vùng nhớ password
    addi s0, s0, 1               # Tăng địa chỉ buffer
    addi s2, s2, 1               # Tăng địa chỉ password
    addi t5, t5, -1              # Giảm số lượng ký tự còn lại
    j copy_new_password          # Lặp lại cho ký tự tiếp theo
password_updated:
    # Hiển thị thông báo mật khẩu đã cập nhật thành công
    la a0, msg_password_updated
    jal print_message
    lw ra, 0(sp)
    addi sp, sp, 4
```

```
    jr ra
password_too_short:
    la a0, msg_short_password       # Hiển thị thông báo mật khẩu quá ngắn
    jal print_message
    j update_new_password           # Yêu cầu nhập lại mật khẩu mới
change_password_wrong:
    addi a2, a2, 1
    bge a2, a3, lock_user
    la a0, msg_password_wrong
    jal print_message
    j retry_old_password            # Yêu cầu nhập lại
lock_user:
    # Hiển thị thông báo khóa
    la a0, msg_lock_suspended
    jal print_message
    li a0, 60000                    # Giả lập thời gian đợi 1 phút
    li a7, 32
    ecall
    # Sau 1 phút, khôi phục lại số lần nhập sai
    li a2, 0
    j retry_old_password            # Yêu cầu nhập lại sau khi hết khóa
```

1. Password Input

The system employs a hexadecimal keypad for user input, supported by the following features:

- **Buffer Management**:
  - User input is stored in a memory buffer (`buffer`) with an index (`index`) tracking the current position.
  - The function `READ_PASSWORD` handles keypad scanning and input decoding. It also debounces keys by incorporating a 100 ms delay after each input.
- **Key Decoding**:

    - Keycodes from the keypad are mapped to corresponding numerical values through the `decode` subroutine. Unsupported keys return an error code (`0xFF`).
- **End-of-Input Detection**:
    - The program listens for the "F" key to signal completion of input.

2. Password Verification

The `CHECK_PASSWORD` subroutine compares the entered password against the stored password:

- **Length Check**:
    - Ensures the entered password matches the stored password's length.
- **Character-by-Character Comparison**:
    - Iterates through the stored password and buffer to check for equality.
- **Result Handling**:
    - If passwords match, the subroutine returns success (`a0 = 1`); otherwise, it returns failure (`a0 = 0`).

3. LED Status Indication

Two 7-segment LEDs visually indicate the system's status:

- **`LED_ON`**:
    - Activates all segments on both LEDs, signaling successful password entry.
- **`LED_OFF`**:
    - Display OF.

4. Incorrect Password Handling

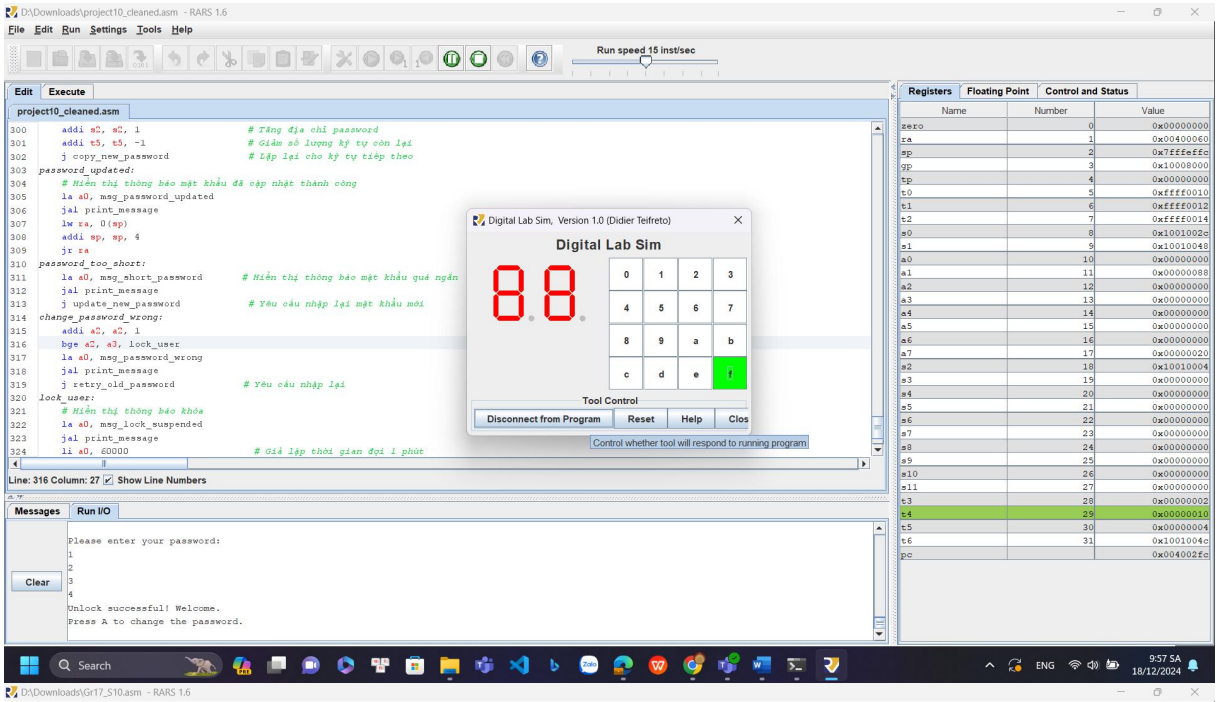
Repeated incorrect attempts trigger specific actions:

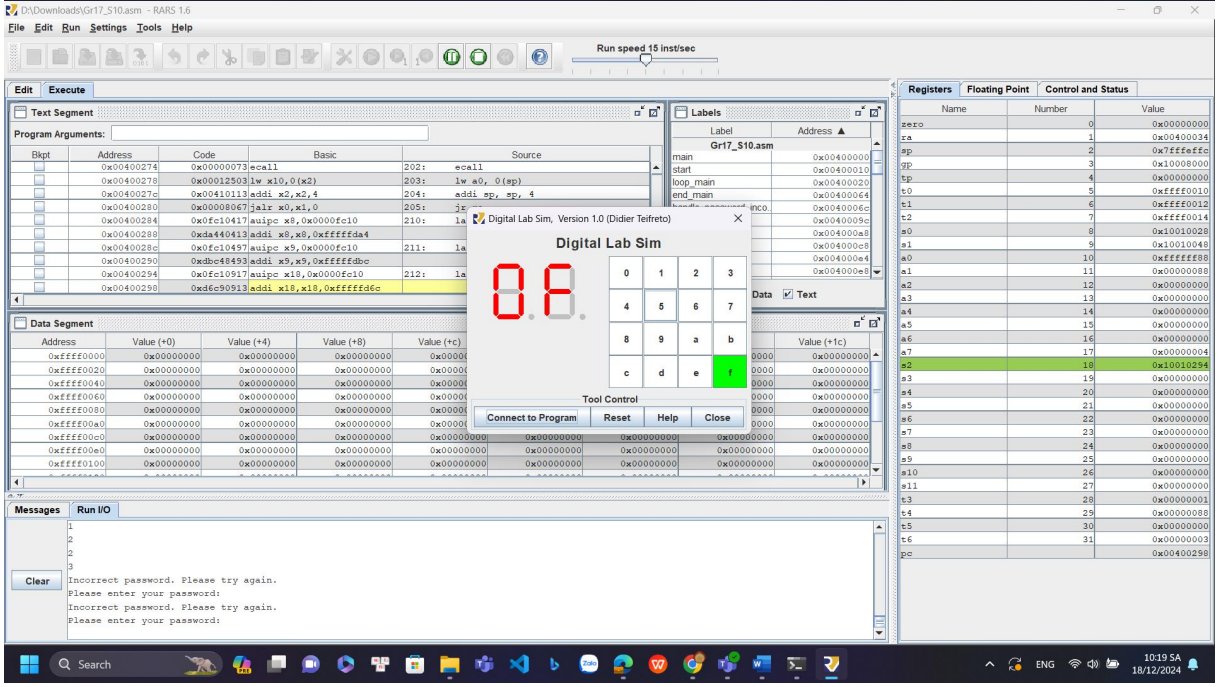
- **Attempt Tracking**:
    - A counter (`wrong_attempts`) tracks failed password entries.
- **Lock Suspension**:
    - After three incorrect attempts, the system enters a suspended state for one minute.
    - The suspension is simulated using a delay (`ecall` for 60,000 ms).

5. Password Change Mechanism

Users can update the password by pressing the "A" key after successful login:

- **Old Password Verification**:
    - Users must verify the old password before entering a new one.
- **New Password Validation**:
    - Ensures the new password meets a minimum length of four characters.
- **Password Update**:
    - Copies the new password into the `password` memory segment and updates its length (`len`).
- **Retry Handling**:
    - Allows three attempts to enter the old password before suspending the user for one minute.


D:\Downloads\project10_cleaned.asm - RARS 1.6
File Edit Run Settings Tools Help
Run speed 15 inst/sec
Edit Execute
project10_cleaned.asm
addi s2, s2, 1 # Tăng địa chỉ password
addi t5, t5, -1 # Giảm số lượng ký tự còn lại
j copy_new_password # Lặp lại cho ký tự tiếp theo
password_updated:
Hiển thị thông báo mật khẩu đã cập nhật thành công
la a0, msg_password_updated
jal print_message
lw ra, 0(sp)
addi sp, sp, 4
jr ra
password_too_short:
la a0, msg_short_password # Hiển thị thông báo mật khẩu quá ngắn
jal print_message
j update_new_password # Yêu cầu nhập lại mật khẩu mới
change_password_wrong:
addi a2, a2, 1
bge a2, a3, lock_user
la a0, msg_password_wrong
jal print_message
j retry_old_password # Yêu cầu nhập lại
lock_user:
Hiển thị thông báo khóa
la a0, msg_lock_suspended
jal print_message
li a0, 60000 # Giả lập thời gian đợi 1 phút
Line: 316 Column: 27 Show Line Numbers
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
Control whether tool will respond to running program
Registers Floating Point Control and Status
Name Number Value
zero 0 0x00000000
ra 1 0x00400060
sp 2 0x7fffeffc
gp 3 0x10008000
tp 4 0x00000000
t0 5 0xffff0010
t1 6 0xffff0012
t2 7 0xffff0014
s0 8 0x1001002c
s1 9 0x10010048
a0 10 0x00000000
a1 11 0x00000088
a2 12 0x00000000
a3 13 0x00000000
a4 14 0x00000000
a5 15 0x00000000
a6 16 0x00000000
a7 17 0x00000020
s2 18 0x10010004
s3 19 0x00000000
s4 20 0x00000000
s5 21 0x00000000
s6 22 0x00000000
s7 23 0x00000000
s8 24 0x00000000
s9 25 0x00000000
s10 26 0x00000000
s11 27 0x00000000
t3 28 0x00000002
t4 29 0x00000010
t5 30 0x00000004
t6 31 0x1001004c
pc 0x004002fc
Messages Run I/O
Clear
Please enter your password:
1
2
3
4
Unlock successful! Welcome.
Press A to change the password.
Search
ENG
9:57 SA
18/12/2024


D:\Downloads\Gr17_S10.asm - RARS 1.6
File Edit Run Settings Tools Help
Run speed 15 inst/sec
Edit Execute
Text Segment
Program Arguments:
Bkpt Address Code Basic Source
0x00400274 0x00000073 ecall 202: ecall
0x00400278 0x00012503 lw x10,0(x2) 203: lw a0, 0(sp)
0x0040027c 0x00410113 addi x2,x2,4 204: addi sp, sp, 4
0x00400280 0x00008067 jalr x0,x1,0 205: jr
0x00400284 0x0fc10417 auipc x8,0x0000fc10 210: la
0x00400288 0xda440413 addi x8,x8,0xffffda4
0x0040028c 0x0fc10497 auipc x9,0x0000fc10 211: la
0x00400290 0xdbc48493 addi x9,x9,0xffffdbc
0x00400294 0x0fc10917 auipc x18,0x0000fc10 212: la
0x00400298 0xd6c90913 addi x18,x18,0xffffd6c
Labels
Label Address
Gr17_S10.asm
main 0x00400000
start 0x00400010
loop_main 0x00400020
end_main 0x00400064
Data Text
Data Segment
Address Value (+0) Value (+4) Value (+8) Value (+c) Value (+1c)
0xffff0000 0x00000000 0x00000000 0x00000000 0x00000000
0xffff0020 0x00000000 0x00000000 0x00000000 0x00000000
0xffff0040 0x00000000 0x00000000 0x00000000 0x00000000
0xffff0060 0x00000000 0x00000000 0x00000000 0x00000000
0xffff0080 0x00000000 0x00000000 0x00000000 0x00000000
0xffff00a0 0x00000000 0x00000000 0x00000000 0x00000000
0xffff00c0 0x00000000 0x00000000 0x00000000 0x00000000
0xffff00e0 0x00000000 0x00000000 0x00000000 0x00000000
0xffff0100 0x00000000 0x00000000 0x00000000 0x00000000
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
Connect to Program Reset Help Close
Registers Floating Point Control and Status
Name Number Value
zero 0 0x00000000
ra 1 0x00400034
sp 2 0x7fffeffc
gp 3 0x10008000
tp 4 0x00000000
t0 5 0xffff0010
t1 6 0xffff0012
t2 7 0xffff0014
s0 8 0x10010028
s1 9 0x10010048
a0 10 0xffffff88
a1 11 0x00000088
a2 12 0x00000000
a3 13 0x00000000
a4 14 0x00000000
a5 15 0x00000000
a6 16 0x00000000
a7 17 0x00000004
s2 18 0x10010294
s3 19 0x00000000
s4 20 0x00000000
s5 21 0x00000000
s6 22 0x00000000
s7 23 0x00000000
s8 24 0x00000000
s9 25 0x00000000
s10 26 0x00000000
s11 27 0x00000000
t3 28 0x00000001
t4 29 0x00000088
t5 30 0x00000000
t6 31 0x00000003
pc 0x00400298
Messages Run I/O
Clear
1
2
2
3
Incorrect password. Please try again.
Please enter your password:
Incorrect password. Please try again.
Please enter your password:
Search
ENG
10:19 SA
18/12/2024