ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CNTT & TRUYỀN THÔNG

BÁO CÁO CUỐI KỲ

MÔN: KIẾN TRÚC MÁY TÍNH

Giảng viên hướng dẫn: TS. Lê Bá Vui

Mã lớp: 156790 – Nhóm 02

Sinh viên thực hiện:

Nguyễn Thiệu Thành – 20235832

Đông Xuân Đức – 20235679

Hà Nội, 5/2024

Mục lục

1 Kiểm tra tốc độ và độ chính xác khi gõ văn bản

1.1 Mô tả

Xây dựng một chương trình "typing test" trên RISC-V với các yêu cầu:

- Hiển thị một chuỗi mẫu để người dùng gõ lại.
- Chương trình sẽ đếm:
  - Số từ người dùng gõ được (word count).
  - Số ký tự gõ đúng so với mẫu (correct character count).
  - Thời gian hoàn thành bài kiểm tra (tính bằng giây).
  - Tốc độ gõ thực tế quy đổi ra words per minute (WPM).
- Hiển thị kết quả ra console và cập nhật số ký tự đúng liên tục lên hai thanh module 7-seg.
- Hỏi người dùng có muốn thực hiện lại bài kiểm tra hay không.

1.2 Ý tưởng

Giải pháp đề xuất là xây dựng một chương trình có thể đáp ứng được tất cả yêu cầu của đề bài, với các phần chương trình con có vai trò tổng quát như sau:

1. **Khởi tạo:** Reset bộ đếm và hiển thị, in prompt, đợi phím bất kỳ → ghi nhận thời gian bắt đầu.
2. **Vòng lặp:** Liên tục đọc ký tự từ bàn phím, in lại, so sánh với mẫu để đếm ký tự đúng/đếm từ. Dừng khi gặp Enter.
3. **Kết thúc:** Tính thời gian, chuyển ms→giây, in kết quả (words, correct chars, elapsed seconds, WPM), hỏi retry/exit.

$$\text{Tốc độ gõ (WPM)} = \frac{\text{Số từ gõ được} \times 60}{\text{Thời gian (giây)}} \tag{1}$$

1.3 Chương trình

1.3.1 Data

Khai báo dữ liệu:

- `sample_text`: Xâu mẫu để người dùng gõ lại.
- `prompt_start, word_count_msg,...`: Các thông báo in ra màn hình qua ecall.
- `word_count`: Biến nguyên (4 byte) lưu số từ đã đếm.
- `seven_seg_patterns`: Mã bật/tắt các đoạn để hiển thị 0–9 trên module 7-seg.

Các `.eqv` gán tên cho các thanh ghi điều khiển và dữ liệu của bàn phím, màn hình ký tự và hiển thị LED 7 đoạn.

```
.data
    sample_text:         .asciz "The quick brown fox jumps over the lazy dog"
    prompt_start:        .asciz "Press any key to start typing test:\n"
    word_count_msg:      .asciz "Number of words: "
    correct_chars_msg:   .asciz "Number of correct characters: "
    elapsed_time_msg:    .asciz "Elapsed time (seconds): "
    typing_speed_msg:    .asciz "Typing Speed (words/min): "
    retry_prompt:        .asciz "Do you want to try again?"
    newline:             .asciz "\n"
    word_count:          .word 0

# Memory-mapped I/O addresses
.eqv KEYBOARD_CONTROL   0xFFFF0000
.eqv KEYBOARD_DATA      0xFFFF0004
.eqv DISPLAY_CONTROL    0xFFFF0008
.eqv DISPLAY_DATA       0xFFFF000C
.eqv DISPLAY_7SEG_LEFT  0xFFFF0011
.eqv DISPLAY_7SEG_RIGHT 0xFFFF0010

# Seven-segment display patterns (0-9)
seven_seg_patterns:
    .byte 0x3F, 0x06, 0x5B, 0x4F, 0x66  # 0-4
    .byte 0x6D, 0x7D, 0x07, 0x7F, 0x6F  # 5-9
```

1.3.2 Initialization

Chức năng:

- Reset các bộ đếm: ký tự đúng (s0), tổng ký tự (s1) và số từ `word_count`.
- Đặt màn hình bảy đoạn về 0.
- Hiển thị prompt bắt đầu và đợi phím bất kỳ.
- Ghi nhận thời gian bắt đầu và chuẩn bị.

Chi tiết:

- Chương trình đặt tất cả giá trị đếm về 0 và đồng bộ hóa LED bảy đoạn để đảm bảo hiển thị ban đầu là số không.
- Sau đó, thông báo được in ra, chương trình chờ đến khi phát hiện phím bất kỳ được nhấn, rồi ghi nhận thời điểm hiện tại để tính toán khoảng thời gian gõ.

```
retry_start:
    # Initialize variables
    li s0, 0            # s0 = correct character counter
    li s1, 0            # s1 = total characters typed
```

```
    sw zero, word_count, t0  # Reset word count

    # Reset seven-segment display
    jal reset_display_chars

    # Display start prompt
    li a7, 4
    la a0, prompt_start
    ecall

    # Wait for key press to start
    jal wait_for_key

    # Get start time
    li a7, 30
    ecall
    mv s2, a0        # s2 = start time

    # Initialize text pointer
    la s3, sample_text
    li s4, 0         # Previous character (for word counting)
```

1.3.3 Typing Loop

Chức năng:

- Đọc từng ký tự người dùng gõ.
- Hiển thị ký tự và so sánh với mẫu.
- Tăng bộ đếm ký tự đúng và đếm số từ.
- Tiếp tục cho đến khi nhấn Enter.

Chi tiết:

- Mỗi lần lặp, hệ thống kiểm tra bàn phím cho đến khi có ký tự nhập vào, sau đó hiển thị ký tự đó ngay lập tức
- Ký tự vừa nhập được so sánh với chuỗi mẫu, nếu giống nhau, số ký tự đúng được cộng và màn hình bảy đoạn được cập nhật.
- Song song, khi gặp dấu cách và trước đó không phải là dấu cách, số từ sẽ tăng, rồi chương trình tiếp tục với ký tự mẫu kế tiếp.

```
typing_loop:
    # Get keyboard input
    jal get_keyboard_input
    mv s5, a0        # Save current character

    # Display typed character
    jal display_char
```

```
8
9       # Check for Enter key (end condition)
10      li t0, 10          # ASCII for newline
11      beq s5, t0, end_typing_test
12
13      # Compare with sample text
14      lb t1, (s3)
15      beq s5, t1, correct_char
16      j check_word
17
18  correct_char:
19      addi s0, s0, 1     # Increment correct characters
20      jal display_correct_chars
21
22  check_word:
23      # Check if current char is space
24      li t0, 32          # ASCII for space
25      bne s5, t0, next_char
26
27      # Check if previous char wasn't space
28      li t0, 32
29      beq s4, t0, next_char
30
31      # Increment word count
32      lw t1, word_count
33      addi t1, t1, 1
34      sw t1, word_count, t0
35
36  next_char:
37      mv s4, s5          # Save current char as previous
38      addi s3, s3, 1     # Move to next sample text char
39      addi s1, s1, 1     # Increment total chars
40      j typing_loop
```

1.3.4 Finalization

Chức năng:

- Xác nhận từ cuối và tính elapsed time.
- In số từ, số ký tự đúng, thời gian (s), và tốc độ gõ (words/min).
- Hỏi người dùng retry hay exit.

Chi tiết:

- Khi kết thúc, chương trình kiểm tra xem ký tự cuối có phải khoảng trắng không để đảm bảo từ cuối cùng được tính.
- Sau đó giá trị thời gian hiện tại được lấy và so với thời điểm bắt đầu để ra kết quả elapsed.

- Các kết quả gồm số từ, số ký tự đúng, thời gian và tốc độ gõ được trình bày theo thứ tự, trong đó tốc độ sẽ là 0 nếu elapsed quá nhỏ, ngược lại sẽ tính tỷ lệ words/min.

```
end_typing_test:
    # Check last word (if doesn't end with space)
    li t0, 32        # ASCII for space
    beq s4, t0, skip_last_word

    # Count last word
    lw t1, word_count
    addi t1, t1, 1
    sw t1, word_count, t0

skip_last_word:
    # Get end time
    li a7, 30
    ecall
    mv s3, a0        # s3 = end time

    # Calculate elapsed time
    sub s4, s3, s2   # s4 = elapsed time in milliseconds

    # Display results
    # 1. Word count
    li a7, 4
    la a0, word_count_msg
    ecall

    li a7, 1
    lw a0, word_count
    ecall

    li a7, 4
    la a0, newline
    ecall

    # 2. Correct characters
    li a7, 4
    la a0, correct_chars_msg
    ecall

    li a7, 1
    mv a0, s0
    ecall

    li a7, 4
    la a0, newline
    ecall

    # 3. Elapsed time
    li a7, 4
```

```
    la a0, elapsed_time_msg
    ecall

    # Convert ms to seconds
    fcvt.s.w ft0, s4         # Convert ms to float
    li t0, 1000
    fcvt.s.w ft1, t0         # Convert 1000 to float
    fdiv.s ft0, ft0, ft1     # Divide by 1000 for seconds

    fmv.s fa0, ft0
    li a7, 2                 # Print float
    ecall

    li a7, 4
    la a0, newline
    ecall

    # 4. Typing speed
    li a7, 4
    la a0, typing_speed_msg
    ecall

    # Check if elapsed time <= 1 ms
    li t0, 10
    bgt s4, t0, calculate_typing_speed  # If elapsed time > 1 ms, go to typing
     speed calculation

    # If elapsed time <= 1 ms, typing speed is 0
    li a0, 0
    li a7, 1                 # syscall print integer
    ecall

    j typing_speed_done      # Jump to done after printing 0

calculate_typing_speed:
    # Calculate words per minute
    lw t0, word_count
    fcvt.s.w ft1, t0         # Convert word count to float
    li t0, 60
    fcvt.s.w ft2, t0         # ft2 = 60
    fmul.s ft1, ft1, ft2     # ft1 = ft1 * 60
    fdiv.s ft0, ft1, ft0     # words/minute

    fmv.s fa0, ft0
    li a7, 2                 # syscall print float
    ecall

    li a7, 4
    la a0, newline
    ecall

```

```
typing_speed_done:
    li a7, 4
    la a0, newline
    ecall

    # Retry prompt
    la a0, retry_prompt
    li a7, 50
    ecall

    beqz a0, retry_start    # If input = 0 (Yes), retry

    # Else, exit program
    li a7, 10
    ecall
```

1.3.5 I/O Utils

Các chương trình con này đảm nhiệm việc đồng bộ hoá và trao đổi dữ liệu với phần cứng bàn phím và màn hình:

- `wait_for_key` và `get_keyboard_input` cùng sử dụng cơ chế polling: liên tục kiểm tra thanh ghi điều khiển bàn phím cho đến khi đặt cờ ready, sau đó `get_keyboard_input` đọc ký tự từ thanh ghi dữ liệu vào a0 còn `wait_for_key` chỉ dừng lại mà không trả giá trị.

```
wait_for_key:
    li t0, KEYBOARD_CONTROL
wait_key_loop:
    lw t1, (t0)
    andi t1, t1, 1
    beqz t1, wait_key_loop
    ret

get_keyboard_input:
    li t0, KEYBOARD_CONTROL
    li t1, KEYBOARD_DATA
wait_input_loop:
    lw t2, (t0)
    andi t2, t2, 1
    beqz t2, wait_input_loop
    lw a0, (t1)
    ret
```

- `display_char` đợi thanh ghi điều khiển hiển thị sẵn sàng rồi ghi mã ký tự từ a0 vào thanh ghi dữ liệu để xuất lên màn hình.

```
display_char:
    li t0, DISPLAY_CONTROL
```

```
wait_display:
    lw t1, (t0)
    andi t1, t1, 1
    beqz t1, wait_display
    li t0, DISPLAY_DATA
    sw a0, (t0)
    ret
```

- `display_correct_chars` cập nhật hai màn hình bảy đoạn để hiển thị số ký tự đúng (s0): tính chữ số hàng chục và hàng đơn vị, tra bảng `seven_seg_patterns` lấy mã hiển thị tương ứng, rồi ghi vào hai thanh ghi `DISPLAY_7SEG_LEFT` và `DISPLAY_7SEG_RIGHT`.

```
display_correct_chars:
    # Get patterns for digits
    la t0, seven_seg_patterns

    # Calculate tens digit
    li t1, 10
    div t2, s0, t1       # t2 = tens
    add t3, t0, t2
    lb t4, (t3)

    # Display left digit
    li t5, DISPLAY_7SEG_LEFT
    sb t4, (t5)

    # Calculate ones digit
    rem t2, s0, t1       # t2 = ones
    add t3, t0, t2
    lb t4, (t3)

    # Display right digit
    li t5, DISPLAY_7SEG_RIGHT
    sb t4, (t5)
    ret
```

- `reset_display_chars` khởi tạo hai màn hình bảy đoạn về số 0 bằng cách lấy mẫu pattern đầu tiên trong `seven_seg_patterns` và ghi ra cả hai thanh ghi.

```
reset_display_chars:
    la t0, seven_seg_patterns
    lb t1, (t0)          # Get pattern for '0'

    # Reset both digits to 0
    li t2, DISPLAY_7SEG_LEFT
    sb t1, (t2)
    li t2, DISPLAY_7SEG_RIGHT
    sb t1, (t2)
    ret
```

Tất cả các chương trình con đều kết thúc bằng lệnh ret để trả về luồng điều khiển cho hàm gọi.

1.4 Output:

Chương trình đánh giá đầu vào của người dùng theo ba tình huống: đầu vào đúng, đầu vào sai và đầu vào trống (nhấn Enter mà không nhập bất kỳ ký tự nào). Mỗi trường hợp mang lại kết quả khác nhau, như được mô tả dưới đây.

1.4.1 Trường hợp Input đúng:

Trong trường hợp nhập liệu đúng, người dùng đã gõ chính xác văn bản mẫu đã cho. Chương trình đã đếm số ký tự đúng, tính toán tốc độ gõ, và hiển thị kết quả tương ứng.

1.4.2 Trường hợp Input không đúng:

Trong trường hợp nhập sai (người dùng đã mắc một số lỗi khi gõ), chương trình chỉ đếm các ký tự đúng đã gõ, và tốc độ gõ sẽ bị ảnh hưởng bởi các lỗi đó.

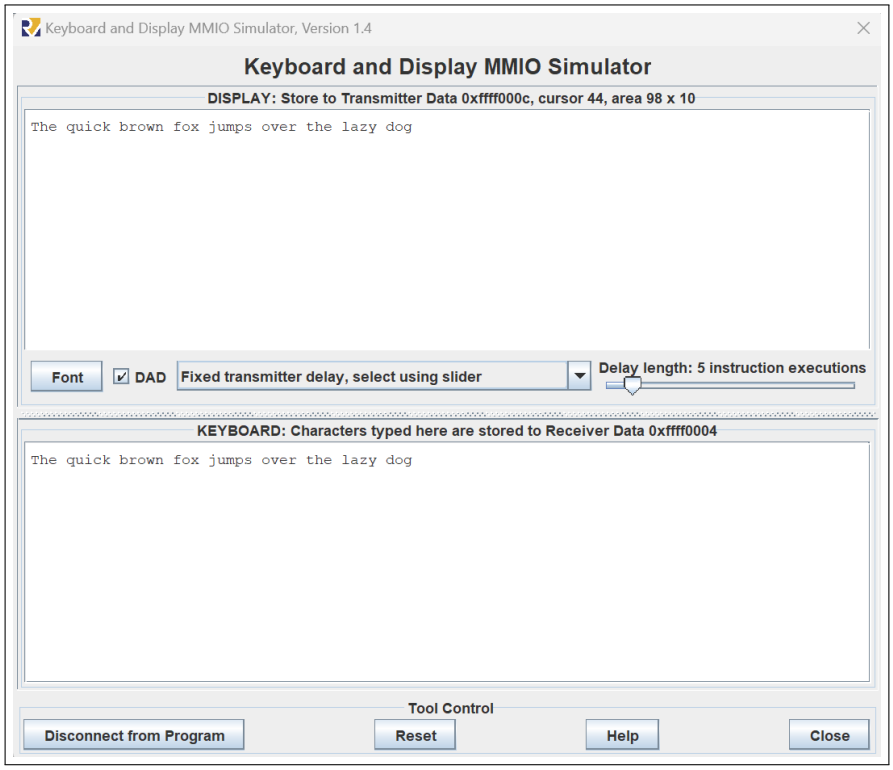
1.4.3 Trường hợp Input rỗng:

Trong trường hợp đầu vào trống, người dùng nhấn Enter mà không gõ bất kỳ ký tự nào. Chương trình phát hiện tình huống này và xử lý nó một cách thích hợp bằng cách hiển thị kết quả là không có ký tự đúng và tốc độ gõ bằng không.

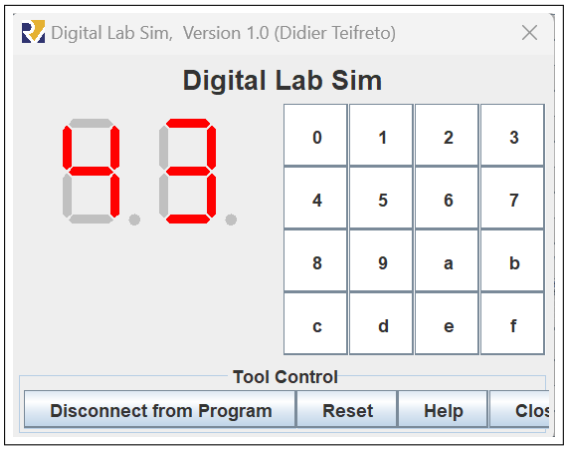
1.4.4 Thông báo thử lại:

Cuối mỗi bài kiểm tra, chương trình hiển thị một thông báo thử lại, như được trình bày trong Hình 4. Điều này cho phép người dùng chọn xem có muốn thử lại bài kiểm tra đánh máy hay thoát khỏi chương trình.
Trong thông báo này, người dùng có ba lựa chọn, mỗi lựa chọn có hành vi như sau:

1. Nếu người dùng chọn **Có**:
    - Bắt đầu lại bài kiểm tra gõ phím.
    - Đặt lại bộ đếm thời gian, thanh 7 LED.
2. Nếu người dùng chọn **Không**:
    - Hiển thị thông điệp tạm biệt "`Exiting...`".
    - Thoát khỏi bài kiểm tra.
3. Nếu người dùng chọn **Hủy**:
    - Đóng hộp thoại xác nhận.
    - Quay lại màn hình hiện tại mà không thực hiện thay đổi nào.



(a) Correct input

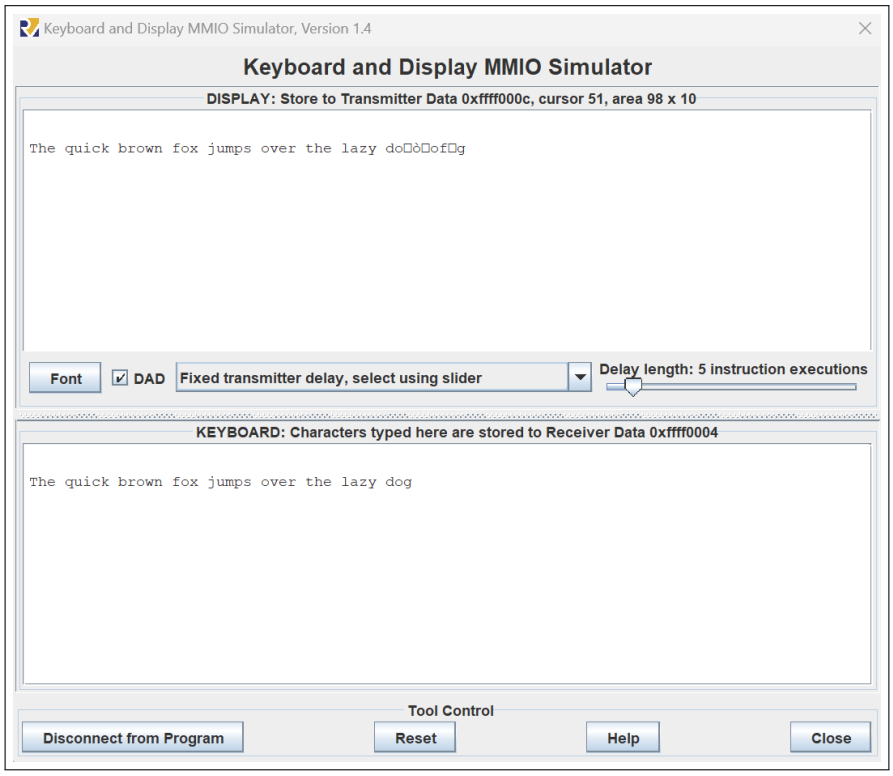


(b) Kết quả Digital Lab Sim

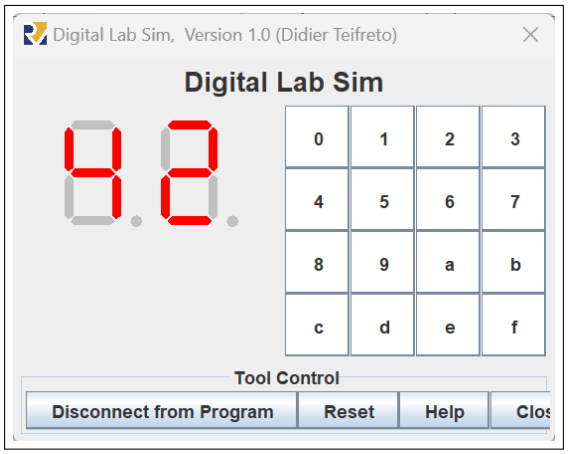
```
Press any key to start typing test:
Number of words: 9
Number of correct characters: 43
Elapsed time (seconds): 5.848
Typing Speed (words/min): 92.339264
```

(c) Kết quả Console

Hình 1: Input và kết quả đếm WPM, Word Count và Time Elapsed



(a) Incorrect input

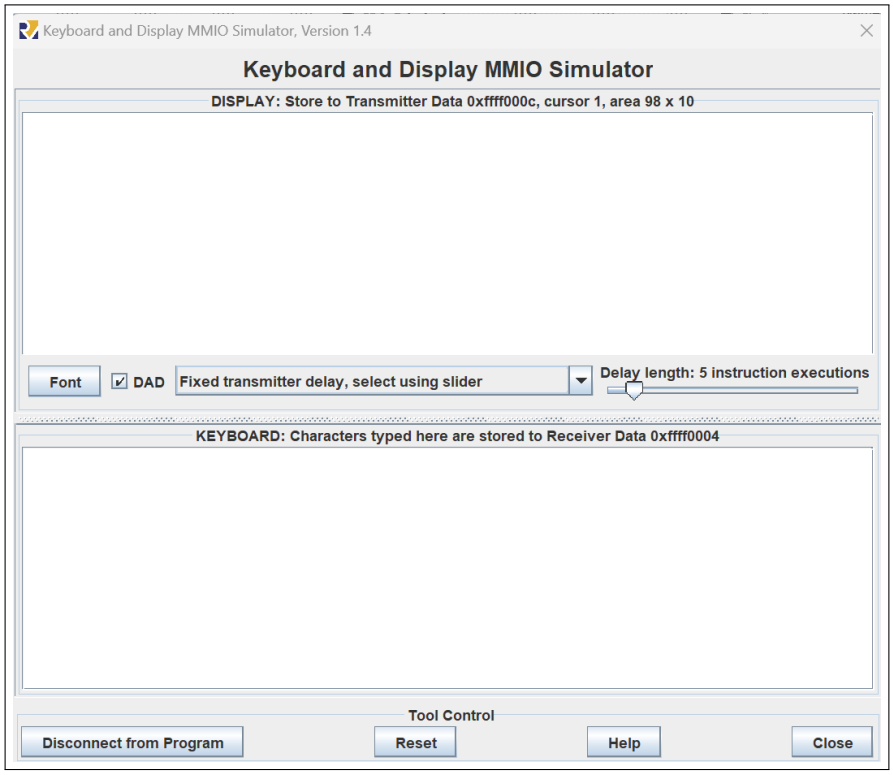


(b) Kết quả Digital Lab Sim

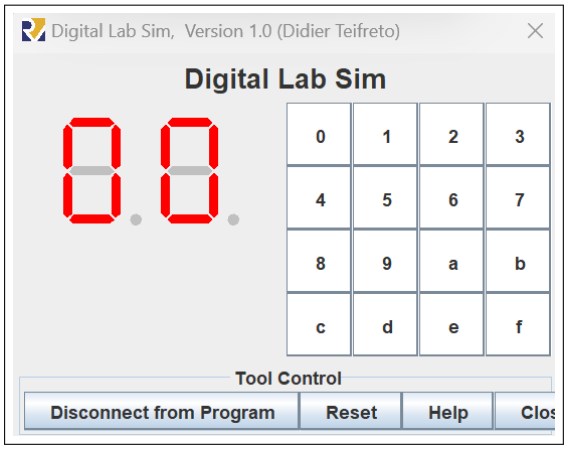
```
Press any key to start typing test:
Number of words: 9
Number of correct characters: 42
Elapsed time (seconds): 9.463
Typing Speed (words/min): 57.064354
```

(c) Kết quả Console

Hình 2: Input và kết quả đếm WPM, Word Count và Time Elapsed



(a) Empty input



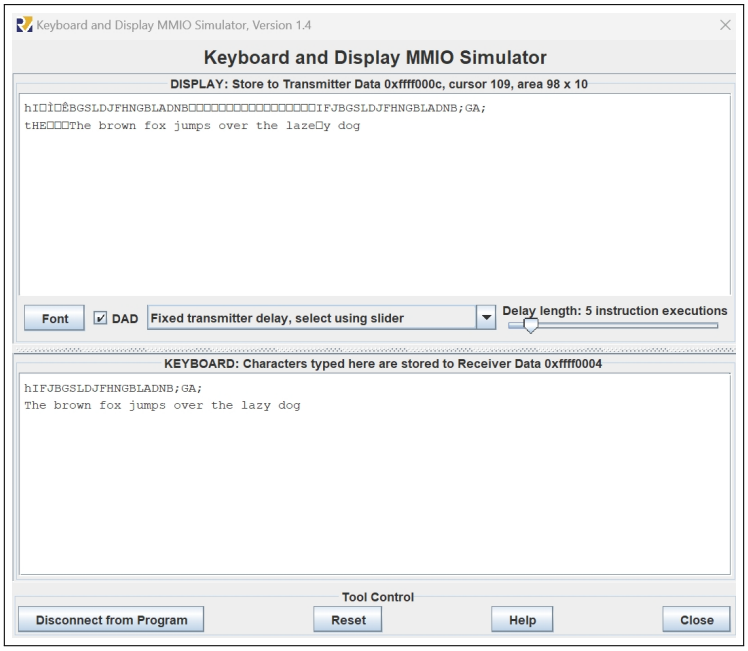
(b) Kết quả Digital Lab Sim

```
Press any key to start typing test:
Number of words: 1
Number of correct characters: 0
Elapsed time (seconds): 0.0
Typing Speed (words/min): 0
```

(c) Kết quả Console

Hình 3: Input và kết quả đếm WPM, Word Count và Time Elapsed

Hình 4: Hộp thoại thông báo



Hình 5: Người dùng nhập input mới

2 Khóa điện tử

2.1 Mô tả

Xây dựng một chương trình "ổ khóa điện tử" trên RISC-V với các yêu cầu:

- Xây dựng một hệ thống khóa điện tử sử dụng bàn phím ma trận (hex keypad) và hiển thị thông tin qua LED 7 đoạn
- Mật khẩu mặc định gồm 4 chữ số (1, 2, 3, 4) được lưu trữ trong bộ nhớ. Người dùng được yêu cầu nhập mật khẩu thông qua bàn phím:
  - Nếu đúng, hệ thống hiển thị "**ON**" trên LED.
  - Nếu sai, hiển thị "**OF**".
- Sau 3 lần nhập sai liên tiếp, hệ thống sẽ bị khóa trong 1 phút.
- Người dùng có thể thay đổi mật khẩu bằng cách nhấn nút 'a', nhập mật khẩu cũ và sau đó nhập mật khẩu mới.

2.2 Ý tưởng

Trong bài toán này, chúng ta cần xây dựng một chương trình cho phép người dùng nhập vào mật khẩu và kiểm tra xem nó có khớp với mật khẩu đã lưu hay không. Ý tưởng của bài toán có thể được chia làm ba bước chính:

1. **Lưu trữ mật khẩu ban đầu:** Chương trình cần có khả năng ghi nhận và lưu giá trị của mật khẩu mới được người dùng nhập vào. Mật khẩu này sẽ được đưa vào vùng nhớ tạm thời (tạm như thanh ghi hoặc mảng ký tự) để lưu trữ.

2. **Yêu cầu người dùng nhập lại mật khẩu:** Sau khi đã có mật khẩu ban đầu, chương trình sẽ yêu cầu người dùng nhập lại mật khẩu để xác nhận. Giá trị này được lưu trữ ở vùng nhớ khác để so sánh sau.

3. **So sánh hai mật khẩu:** Cuối cùng, chương trình sẽ duyệt qua từng ký tự trong hai lần nhập mật khẩu và kiểm tra xem chúng có giống nhau không. Nếu có ký tự nào không trùng, chương trình sẽ thông báo lệnh nhập sai. Ngược lại, nếu toàn bộ các ký tự đều trùng khớp, chương trình sẽ thông báo xác nhận mật khẩu thành công.

2.3 Chương trình

2.3.1 Data

Khai báo dữ liệu:

- `msg1, msg2,..., msg6`: Các thông báo in ra màn hình.
- `entered_password`: Mật khẩu do người dùng nhập từ bàn phím.
- `default_password`: Mật khẩu mặc định ban đầu.

Các `.eqv` gán tên cho địa chỉ các thanh ghi điều khiển và hiển thị LED 7 đoạn.

```
.eqv IN_ADDRESS_HEXA_KEYBOARD 0xFFFF0012    # Command row selection
.eqv OUT_ADDRESS_HEXA_KEYBOARD 0xFFFF0014   # Read key pressed (row + column)
.eqv SEVENSEG_LEFT 0xFFFF0011               # Dia chi cua den led 7 doan trai
.eqv SEVENSEG_RIGHT 0xFFFF0010              # Dia chi cua den led 7 doan phai

.data
    msg1: .asciz "\nEnter password:\n"
    msg2: .asciz "\nPassword wrong\n"
    msg3: .asciz "\nEnter new password\n"
    msg4: .asciz "\nYou want to change password. Enter your password:\n"
    msg5: .asciz "\nPassword true\n"
    msg6: .asciz "\nPassword must be at least 4 digits\n"
    entered_password: .word 0:100
    default_password: .word 1, 2, 3, 4
```

2.3.2 Initialization

Chức năng: Khởi tạo các giá trị ban đầu về 0 (mặc định) hoặc 3 (`max_try`)

```
main:
    la t4, entered_password     # Address of password
    la t5, default_password     # Address of default password
    li t6, 0                    # Check if need change password
    li s0, 0                    # Loop index
    li s10, 3                   # Max try
    li s9, 0                    # number of incorrect try
    li s8, 16                   # length of default
```

```
    li s7, 0                    # length of entered
    li a6, 0                    # Store the nearest != 0
    li a5, 0                    # Store the nearest != 0
```

2.3.3 Polling

Chức năng:

- Là vòng lặp chính của chương trình.
- Liên tục quét bàn phím để phát hiện phím được nhấn.
- Thực hiện việc kích hoạt từng hàng một của bàn phím ma trận và kiểm tra xem có phím nào được nhấn ở hàng đó hay không.

Chi tiết:

- Chương trình lần lượt kích hoạt các hàng (row) của bàn phím bằng cách ghi giá trị tương ứng vào thanh ghi `a4`.
- Sau mỗi lần kích hoạt hàng, chương trình gọi hàm `check_key` để xác định xem có phím nào ở hàng đó đang được nhấn không.
- Nếu không có phím nào được nhấn, chương trình tiếp tục chuyển sang hàng tiếp theo.

```
polling:
    li a3, SEVENSEG_LEFT
    sb a2, 0(a3)
    li a2, 0x0
    li a3, SEVENSEG_RIGHT
    sb a2, 0(a3)
    li t1, 0xffff0012           # Address to send row command
    li t2, 0xffff0014           # Address to read scan code
    li t3, 0x1                  # check 1st row
    sb t3, 0(t1)
    jal check_key
    li t3, 0x2                  # check 2nd row
    sb t3, 0(t1)
    jal check_key
    li t3, 0x4                  # check 3rd row
    sb t3, 0(t1)
    jal check_key
    li t3, 0x8                  # check 4th row
    sb t3, 0(t1)
    jal check_key
    j polling
```

2.3.4 check_key : Kiểm tra phím nhấn

Chức năng:

- Là cầu nối giữa việc quét phần cứng và xử lý logic phần mềm.
- Đọc giá trị scan code từ bàn phím ma trận và chuyển tiếp cho các hàm xử lý tiếp theo

Chi tiết:

- Hàm thực hiện lệnh load byte từ địa chỉ `OUT_ADDRESS_HEXA_KEYBOARD` để lấy scan code của phím được nhấn, giá trị này sẽ được lưu vào thanh ghi `a0`.
- Sau khi có được scan code, hàm không thực hiện bất kỳ xử lý logic nào mà chuyển ngay đến hàm `preprocess_key` để thực hiện các bước tiền xử lý cần thiết.

```
check_key:
    lbu a0, 0(t2)
```

2.3.5 preprocess_key - Tiền xử lý phím

Chức năng:

- Hàm thực hiện cơ chế debouncing để loại bỏ tín hiệu nhiễu và đảm bảo tính ổn định của đầu vào.
- Sử dụng thuật toán so sánh trạng thái để xác định tính hợp lệ của tín hiệu phím.

Chi tiết:

- Hàm so sánh giá trị hàng hiện tại (trong thanh ghi t3) với giá trị hàng trước đó (lưu trong `a6`) để phát hiện việc chuyển đổi giữa các hàng quét, nếu khác nhau thì chuyển đến `process_key` để xử lý mới.
- Trong trường hợp cùng một hàng, hàm kiểm tra xem có scan code hay không và so sánh với giá trị trước đó (lưu trong `a5`) để đảm bảo tín hiệu ổn định, tránh việc xử lý nhiều lần cùng một phím.

```
preprocess_key:
    bne a6, t3, process_key

continue:
    beqz a0, process_key_2
    beqz a5, process_key
    beq a5, a0, back
    j process_key

back:
    jr ra
```

2.3.6 process_key và process_key_2 - Xử lý phím chính

Chức năng:

- Thực hiện việc giải mã scan code và phân luồng xử lý theo chức năng của từng phím.

Chi tiết:

- Hàm `process_key` cập nhật các biến a5 và a6 để lưu trữ trạng thái hiện tại, phục vụ cho việc debouncing ở các lần quét tiếp theo, đồng thời reset LED 7 đoạn về trạng thái tắt.
- Hàm `process_key_2` kiểm tra các scan code đặc biệt như 0x44 (phím A - đổi mật khẩu) và 0x88 (phím F - xác nhận), nếu phát hiện sẽ chuyển hướng đến các hàm xử lý tương ứng.
- Đối với các scan code từ 0x11 đến 0x24, hàm sẽ so sánh tuần tự để xác định phím số tương ứng (0-9) và chuyển đến hàm `press_x` phù hợp để xử lý lưu trữ.

```
process_key:
    beqz a0, back
    xor a6, a6, a6
    add a6, a6, t3

process_key_2:
    xor a5, a5, a5
    add a5, a5, a0
    li a2, 0x0
    li a3, SEVENSEG_LEFT
    sb a2, 0(a3)
    li a2, 0x0
    li a3, SEVENSEG_RIGHT
    sb a2, 0(a3)
    li t3, 0x44
    beq a0, t3, new_password
    li t3, 0x88
    beq a0, t3, compare_password

    # Check for '0'
    li t3, 0x11                 # Load the scan code for key '0' (0x11)
    beq a0, t3, press_0         # If the pressed key is '0', jump to press_0

    # Check for '1'
    li t3, 0x21                 # Load the scan code for key '1' (0x21)
    beq a0, t3, press_1         # If the pressed key is '1', jump to press_1

    # Check for '2'
    li t3, 0x41                 # Load the scan code for key '2' (0x41)
    beq a0, t3, press_2         # If the pressed key is '2', jump to press_2

    # Check for '3'
```

```
    li t3, 0x81                 # Load the scan code for key '3' (0x81)
    beq a0, t3, press_3         # If the pressed key is '3', jump to press_3

    # Check for '4'
    li t3, 0x12                 # Load the scan code for key '4' (0x12)
    beq a0, t3, press_4         # If the pressed key is '4', jump to press_4

    # Check for '5'
    li t3, 0x22                 # Load the scan code for key '5' (0x22)
    beq a0, t3, press_5         # If the pressed key is '5', jump to press_5

    # Check for '6'
    li t3, 0x42                 # Load the scan code for key '6' (0x42)
    beq a0, t3, press_6         # If the pressed key is '6', jump to press_6

    # Check for '7'
    li t3, 0x82                 # Load the scan code for key '7' (0x82)
    beq a0, t3, press_7         # If the pressed key is '7', jump to press_7

    # Check for '8'
    li t3, 0x14                 # Load the scan code for key '8' (0x14)
    beq a0, t3, press_8         # If the pressed key is '8', jump to press_8

    # Check for '9'
    li t3, 0x24                 # Load the scan code for key '9' (0x24)
    beq a0, t3, press_9         # If the pressed key is '9', jump to press_9

    jr ra
```

2.3.7 press_0 đến press_9 - Xử lý phím số

Chức năng:

- Nhóm 10 hàm con chuyên biệt xử lý các phím số từ 0 đến 9.
- Mỗi hàm có nhiệm vụ chuyển đổi scan code thành giá trị số thực tế và chuẩn bị cho việc lưu trữ.

Chi tiết:

- Mỗi hàm `press_x` sẽ load giá trị số tương ứng (0-9) vào thanh ghi t3, thực hiện việc mapping từ scan code phức tạp sang giá trị số đơn giản.
- Tất cả các hàm đều sử dụng cùng một cơ chế là jump đến `store_digit`, đảm bảo tính nhất quán trong việc xử lý và lưu trữ dữ liệu.

```
press_0:
    li t3, 0                    # Store digit '0'
    j store_digit

press_1:
    li t3, 1                    # Store digit '1'
```

```
7       j store_digit
8
9   press_2:
10      li t3, 2                    # Store digit '2'
11      j store_digit
12
13  press_3:
14      li t3, 3                    # Store digit '3'
15      j store_digit
16
17  press_4:
18      li t3, 4                    # Store digit '4'
19      j store_digit
20
21  press_5:
22      li t3, 5                    # Store digit '5'
23      j store_digit
24
25  press_6:
26      li t3, 6                    # Store digit '6'
27      j store_digit
28
29  press_7:
30      li t3, 7                    # Store digit '7'
31      j store_digit
32
33  press_8:
34      li t3, 8                    # Store digit '8'
35      j store_digit
36
37  press_9:
38      li t3, 9                    # Store digit '9'
39      j store_digit
```

2.3.8 store_digit - Lưu trữ chữ số

Chức năng:

- Thực hiện nhiệm vụ lưu trữ chữ số vào mảng mật khẩu và cập nhật các biến trạng thái liên quan.
- Đây là điểm cuối của chuỗi xử lý đầu vào số.

Chi tiết:

- Hàm lưu giá trị từ thanh ghi t3 vào vị trí hiện tại của mảng `entered_password`, đồng thời tăng độ dài mật khẩu (`s7`) và chỉ số vòng lặp (`s0`) lên 4 byte.
- Để hỗ trợ debug và tạo phản hồi cho người dùng, hàm sử dụng system call số 1 để in giá trị vừa nhập ra console, giúp xác nhận việc nhập liệu thành công.
- Sau khi hoàn tất, hàm cập nhật con trỏ mảng t4 để trỏ đến vị trí tiếp theo và quay về vòng lặp chính `polling` để tiếp tục nhận đầu vào.

```
# Function to store digit into password array
store_digit:
    addi a0, t3, 0
    addi s7, s7, 4
    sw a0, 0(t4)
    li a7, 1
    ecall
    addi s0, s0, 4
    addi t4, t4, 4
    j polling
```

2.3.9 new_password - Kích hoạt đổi mật khẩu

Chức năng:

- Khởi tạo quy trình thay đổi mật khẩu khi người dùng nhấn phím A.
- Đóng vai trò như một công tắc chuyển đổi chế độ hoạt động của hệ thống.

Chi tiết:

- Hàm tăng giá trị của biến flag t6 lên 1, đánh dấu rằng hệ thống đang ở chế độ thay đổi mật khẩu và các xử lý tiếp theo sẽ tuân theo logic đặc biệt.
- Hàm sử dụng system call số 4 để hiển thị thông báo `msg4` yêu cầu người dùng nhập mật khẩu hiện tại để xác thực trước khi cho phép thay đổi.
- Sau khi hiển thị thông báo, hàm quay về vòng lặp `polling` để chờ người dùng nhập mật khẩu cũ, tuy nhiên lần này với flag t6 đã được set.

```
new_password:
    addi t6, t6, 1
    li a7, 4
    la a0, msg4
    ecall
    j polling
```

2.3.10 compare_password - So sánh mật khẩu

Chức năng:

- Hàm xử lý trung tâm cho việc xác thực mật khẩu trong cả hai chế độ: mở khóa thường và xác thực trong quá trình đổi mật khẩu.
- Hàm này quyết định luồng xử lý tiếp theo dựa trên kết quả so sánh.

Chi tiết:

- Hàm kiểm tra giá trị của biến a1 để xác định đây là lần xác thực thứ hai trong quá trình đổi mật khẩu hay là xác thực thường, từ đó chuyển hướng đến `change_password` nếu cần.

- Trước khi so sánh chi tiết, hàm so sánh độ dài mật khẩu đã nhập (`s7`) với độ dài mật khẩu mặc định (`s8`), nếu khác nhau thì ngay lập tức kết luận mật khẩu sai.
- Nếu độ dài khớp, hàm reset các con trỏ mảng và chuyển đến vòng lặp here để thực hiện so sánh từng ký tự một cách tuần tự.

```
compare_password:
    la t4, entered_password     # Address of password
    la t5, default_password     # Address of default password
    bgtz a1, change_password
    bne s7, s8, false_password
    li s0, 0
```

2.3.11 here - Vòng lặp so sánh chi tiết

Chức năng:

- Thực hiện việc so sánh từng phần tử của mảng mật khẩu một cách tuần tự và chi tiết.

Chi tiết:

- Vòng lặp load giá trị từ cả hai mảng `entered_password` và `default_password` tại cùng một vị trí, sau đó thực hiện so sánh, nếu khác nhau thì ngay lập tức jump đến `false_password`.
- Sau mỗi lần so sánh thành công, vòng lặp tăng bộ đếm s0 và kiểm tra xem đã so sánh hết độ dài mật khẩu chưa, nếu đã hoàn tất thì chuyển đến `true_password`.
- Nếu chưa hoàn tất, vòng lặp tăng con trỏ của cả hai mảng và tiếp tục so sánh phần tử tiếp theo, đảm bảo toàn bộ mật khẩu được kiểm tra.

```
here:
    lw s4, 0(t4)
    lw s5, 0(t5)
    bne s4, s5, false_password
    addi s0, s0, 4
    beq s0, s7, true_password
    addi t4, t4, 4
    addi t5, t5, 4
    j here
```

2.3.12 true_password - Xử lý mật khẩu đúng

Chức năng:

- Xử lý khi mật khẩu được nhập đúng, thực hiện các hành động tương ứng như hiển thị trạng thái thành công và reset các biến hệ thống.
- Nó cũng xử lý logic đặc biệt cho chế độ đổi mật khẩu.

Chi tiết:

- Hàm ghi các giá trị `0x3F` và `0x37` vào LED 7 đoạn để hiển thị chữ "ON", thông báo cho người dùng biết khóa đã được mở thành công.
- Hàm khôi phục các biến về trạng thái ban đầu bao gồm reset độ dài mật khẩu nhập vào (`s7`), số lần thử sai (`s9`), và con trỏ mảng, đồng thời hiển thị thông báo thành công.
- Nếu hệ thống đang ở chế độ đổi mật khẩu (`t6` > 0), hàm sẽ chuyển đến `if_true_for_change` để tiếp tục quy trình, ngược lại sẽ quay về `polling` để hoạt động bình thường.

```
true_password:
    li a2, 0x3F
    li a3, SEVENSEG_LEFT
    sb a2, 0(a3)
    li a2, 0x37
    li a3, SEVENSEG_RIGHT
    sb a2, 0(a3)
    la t4, entered_password     # Address of password
    la t5, default_password     # Address of default password
    la a0, msg5
    li s7, 0
    li a7, 4
    li s9, 0
    ecall
    bgtz t6, if_true_for_change
    j polling
```

2.3.13 if_true_for_change - Xử lý đổi mật khẩu

Chức năng:

- Xử lý giai đoạn trung gian trong quá trình đổi mật khẩu, khi mật khẩu cũ đã được xác thực thành công và hệ thống chuẩn bị chuyển sang giai đoạn nhập mật khẩu mới.

Chi tiết:

- Hàm tăng giá trị biến a1 lên 1 để đánh dấu rằng quá trình xác thực mật khẩu cũ đã hoàn tất và hệ thống sẵn sàng cho giai đoạn nhập mật khẩu mới.
- Sau khi cập nhật trạng thái, hàm quay về vòng lặp `polling` để cho phép người dùng nhập mật khẩu mới, lần này hệ thống sẽ hoạt động với logic khác nhờ flag a1 đã được set.
- Giai đoạn này đảm bảo rằng khi người dùng nhấn phím F lần tiếp theo, hệ thống sẽ biết đây là lần xác nhận mật khẩu mới và sẽ thực hiện việc cập nhật mật khẩu.

```
if_true_for_change:
    addi a1, a1, 1
    j polling
```

2.3.14 change_password: Thay đổi mật khẩu

Chức năng:

- Kiểm tra mật khẩu mới nhập có ít hơn 4 chữ số hay không.
- Nếu ít hơn 4 chữ số, nhảy tới nhãn too_short để xử lý tiếp

```
change_password:
    li   t0, 16           # t0 = 16 bytes
    blt  s7, t0, too_short
    j    do_copy
```

2.3.15 too_short: Xử lý mật khẩu dưới 4 ký tự

Chức năng:

- Xử lý trường hợp mật khẩu mới được nhập vào có ít hơn 4 ký tự.
- Yêu cầu người dùng nhập lại mật khẩu khác.

```
too_short:
    la   a0, msg6
    li   a7, 4
    ecall
    li   s7, 0
    la   t4, entered_password
    j    new_password
```

2.3.16 do_copy: Thay đổi mật khẩu

Chức năng:

- Cập nhật mật khẩu mặc định bằng cách sao chép mật khẩu vừa nhập thành mật khẩu mới của hệ thống.
- Hoạt động theo cơ chế sao chép tuần tự từng khối dữ liệu cho đến khi hoàn thành toàn bộ chuỗi mật khẩu.

Chi tiết:

- Hoạt động theo cơ chế sao chép tuần tự từng khối dữ liệu cho đến khi hoàn thành toàn bộ chuỗi mật khẩu
- Đọc từng khối dữ liệu từ vùng mật khẩu nguồn và ghi vào vùng mật khẩu đích, sau mỗi lần sao chép sẽ di chuyển đến vị trí tiếp theo.
- Lặp lại việc sao chép cho đến khi đủ độ dài mật khẩu, sau đó chuyển sang giai đoạn thông báo thành công.

```
do_copy:
    la t4, entered_password     # Address of password
    la t5, default_password     # Address of default password
    li s8, 0
    li s0, 0

here3:
    lw s4, 0(t4)
    sw s4, 0(t5)
    addi s0, s0, 4
    addi s8, s8, 4
    beq s0, s7, return
    addi t4, t4, 4
    addi t5, t5, 4
    j here3
```

2.3.17 false_password: Xử lý nhập sai mật khẩu

Chức năng:

- Xử lý tình huống khi người dùng nhập sai mật khẩu, cung cấp phản hồi và áp dụng biện pháp bảo mật.
- Hiển thị cảnh báo trực quan, thông báo lỗi và theo dõi số lần thử để kích hoạt cơ chế bảo vệ

Chi tiết:

- Hiển thị thông báo "FA" (False - Sai) trên màn hình LED để người dùng nhận biết ngay lập tức rằng mật khẩu không chính xác.
- In thông báo chi tiết về việc nhập sai mật khẩu, đồng thời ghi nhận thêm một lần thử không thành công vào bộ đếm.
- Đánh giá xem số lần nhập sai có vượt quá giới hạn cho phép hay không, nếu có thì kích hoạt chế độ khóa, ngược lại cho phép thử lại.

```
false_password:
    li a2, 0x3F
    li a3, SEVENSEG_LEFT
    sb a2, 0(a3)
    li a2, 0x88
    li a3, SEVENSEG_RIGHT
    sb a2, 0(a3)
    la t4, entered_password     # Address of password
    la a0, msg2
    li a7, 4
    li s7, 0
    ecall
    addi s9, s9, 1
    bge s9, s10, frozen
    j polling
```

2.3.18 frozen : Cơ chế khóa bàn phím tự động

Chức năng:

- Cơ chế bảo vệ an ninh khi phát hiện dấu hiệu tấn công brute force thông qua việc nhập sai quá nhiều lần
- Tạm thời vô hiệu hóa hệ thống trong khoảng thời gian nhất định, sau đó cho phép hoạt động bình thường trở lại

Chi tiết:

- Đặt lại số lần thử về zero để sau khi mở khóa, người dùng có cơ hội bắt đầu lại từ đầu thay vì bị ảnh hưởng bởi lịch sử trước đó.
- Đình chỉ hoàn toàn mọi hoạt động của hệ thống trong vòng 60 giây để ngăn chặn các cuộc tấn công tự động.
- Sau khi hết thời gian khóa, tự động đưa hệ thống trở về trạng thái chờ nhập liệu bình thường.

```
frozen:
    li s9, 0
    li a0, 60000
    li a7, 32
    ecall
    j polling
```

2.3.19 return - Hàm trả về

Chức năng:

- Xử lý khi quá trình xác thực thành công hoặc thay đổi mật khẩu hoàn tất.
- Dọn dẹp môi trường, thông báo kết quả và chuẩn bị hệ thống cho lần sử dụng tiếp theo.

Chi tiết:

- Xóa sạch các dữ liệu tạm thời và đưa hệ thống về trạng thái ban đầu để đảm bảo an toàn thông tin.
- Hiển thị thông điệp xác nhận rằng thao tác đã được thực hiện thành công để người dùng biết kết quả.
- Đưa hệ thống trở về trạng thái chờ để sẵn sàng phục vụ các yêu cầu xác thực hoặc thay đổi mật khẩu trong tương lai.

```
return:
    li t6, 0
    li a1, 0
    la a0, msg1
    li a7, 4
```

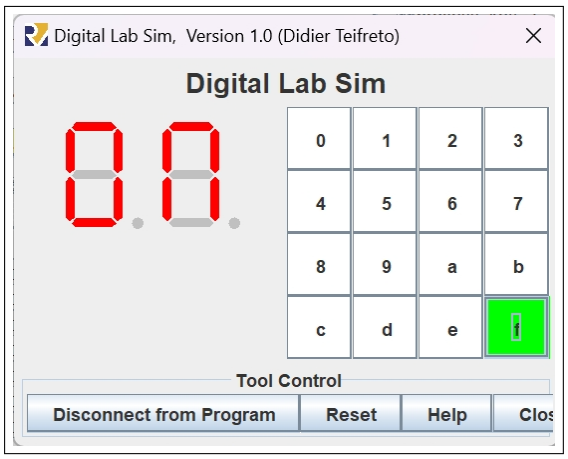
```
    li s7, 0
    ecall
    la t4, entered_password     # Address of password
    la t5, default_password     # Address of default password
    j polling
```

2.4 Output:

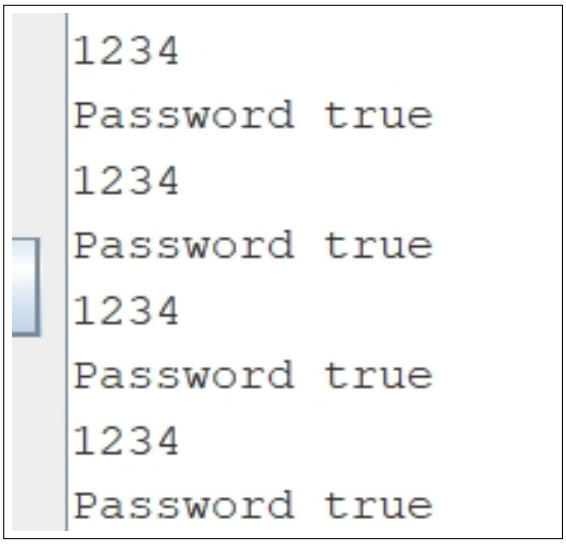
Chương trình đánh giá đầu vào của người dùng theo hai tình huống: nhập mật khẩu Đúng/Sai, thay đổi mật khẩu. Mỗi trường hợp mang lại kết quả khác nhau, như được mô tả dưới đây.

2.4.1 Trường hợp nhập mật khẩu Đúng/Sai:

1. Trường hợp người dùng nhập đúng mật khẩu, khóa sẽ hiện "ON" và ở console sẽ có thông báo "Password true".



(a) Kết quả Digital Lab Sim



(b) Kết quả Console

Hình 6: Khi nhập mật khẩu đúng

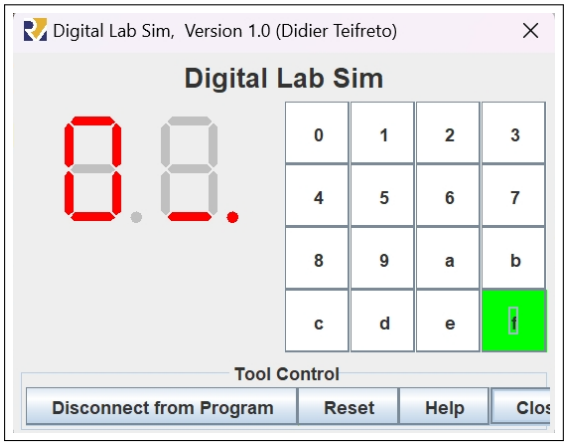
2. Trường hợp người dùng nhập sai mật khẩu, khóa sẽ hiện "OF" và ở console sẽ có thông báo "Password wrong".

3. Trường hợp người dùng nhập sai mật khẩu 3 lần, khóa sẽ hiện "OF" và chương trình sẽ bị treo, không thể nhập ký tự trong vòng 1 phút".

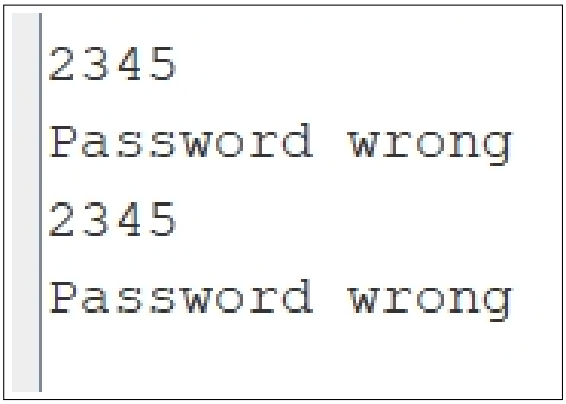
2.4.2 Trường hợp thay đổi mật khẩu:

Trường hợp người dùng muốn thay đổi mật khẩu cũng sẽ có 3 khả năng xảy ra để có thể thay đổi mật khẩu: Nhập mật khẩu đúng, nhập mật khẩu sai và nhập sai 3 lần, mật khẩu mới có ít hơn 4 chữ số.

1. Trường hợp nhập sai mật khẩu lần đầu tiên khi muốn thay đổi mật khẩu:

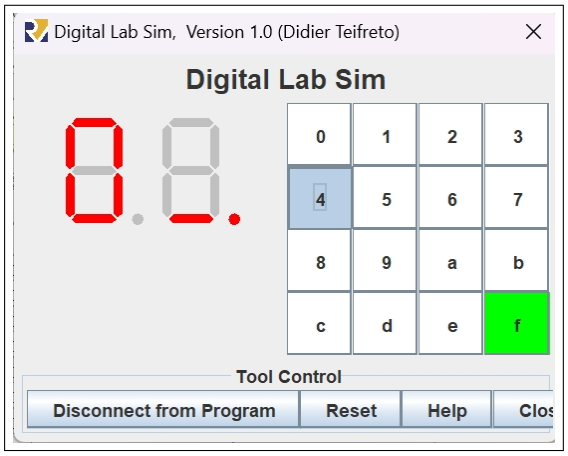


(a) Kết quả Digital Lab Sim

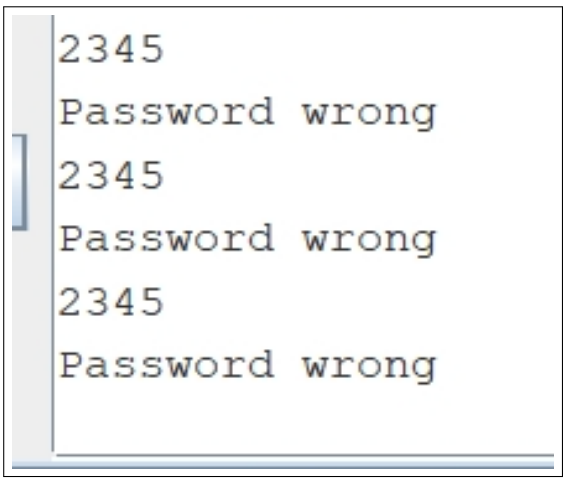


(b) Kết quả Console

Hình 7: Khi nhập sai 1 lần

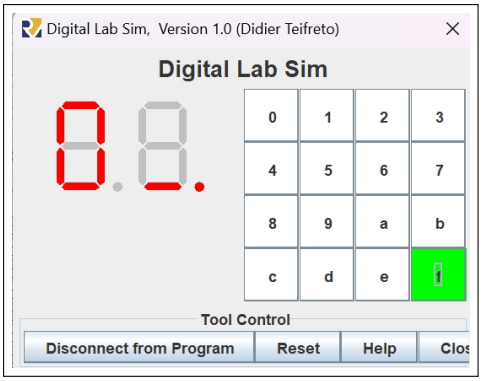


(a) Kết quả Digital Lab Sim



(b) Kết quả Console

Hình 8: Khi nhập sai mật khẩu 3 lần liên tiếp



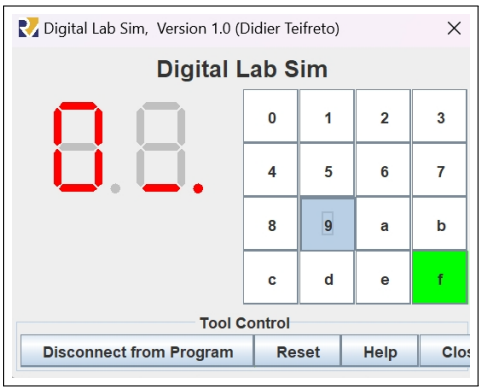
(a) Kết quả Digital Lab Sim

```
You want to change password. Enter your password:
2345
Password wrong
```

(b) Kết quả Console

Hình 9: Nhập sai mật khẩu 1 lần khi thay đổi password mới

2. Trường hợp người dùng nhập sai mật khẩu 3 lần, khóa sẽ hiện ”OF” và chương trình sẽ bị treo, không thể nhập ký tự trong vòng 1 phút:



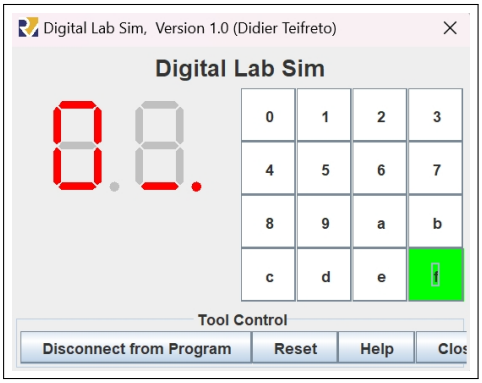
(a) Kết quả Digital Lab Sim

```
You want to change password. Enter your password:
2345
Password wrong
2345
Password wrong
2345
Password wrong
```

(b) Kết quả Console

Hình 10: Nhập sai mật khẩu 3 lần liên tiếp khi muốn thay đổi mật khẩu

3. Trường hợp người dùng nhập mật khẩu mới có ít hơn 4 chữ số, chương trình sẽ yêu cầu nhập lại:



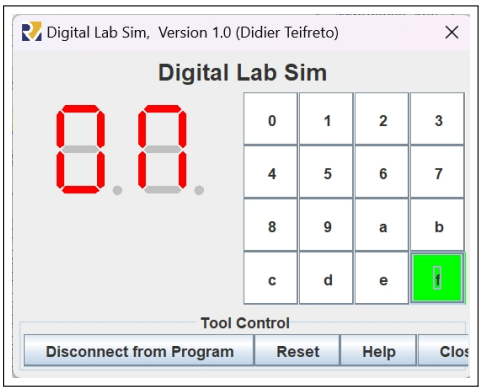
(a) Kết quả Digital Lab Sim

```
You want to change password. Enter your password:
1234
Password true
123
Password must be at least 4 digits
```

(b) Kết quả Console

Hình 11: Mật khẩu mới có ít hơn 4 ký tự

4. Nhập mật khẩu đúng, thay đổi mật khẩu thành công:



(a) Kết quả Digital Lab Sim

```
You want to change password. Enter your password:
1234
Password true
2345
Enter password:
2345
Password true
```

(b) Kết quả Console

Hình 12: Thay đổi mật khẩu thành công