HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION AND COMMUNICATIONS TECHNOLOGY

Final Project Report

IT3280E- Assembly Language and Computer Architecture Lab

Instructor: Ph.D Pham Ngoc Hung

Teaching Assistant: Nguyen Dang Phuc Hung

Full name: Hoang Manh Cuong

Student ID: 20226018

Ha Noi, 2024

Contents

I. Problem Description 3
II. Method and Algorithms 3
1. Set up 3
2. Initial data 3
3. Main program 4
4. Subroutine: 5
III. Simulation Results 7
3.1 Sample Test: 7
3.2 Limitations: 8
3.3 Lesson learns: 8
IV. Source code: 8

I. Problem Description

Script-Based Music Playback – Electronic Keyboard

Task:

- Learn about system functions for playing music.
- A music track is represented as a string of parameters: pitch, duration, instrument, and volume. Example: "60, 1200,1, 120, 73, 220,1, 125, ...".
- Prepare four preset music tracks.
- During execution, the user selects a music track by pressing keys 1 to 4. Pressing 0 stops playback.

II. Method and Algorithms

1. **Set up**

   First we initiate the data:

```
.eqv IN_ADDRESS_HEXA_KEYBOARD  0xFFFF0012  #Input addresss for the
keyboard (Memory-mapped I/O)
.eqv OUT_ADDRESS_HEXA_KEYBOARD 0xFFFF0014  #Output address for the
keyboard (Memory-mapped I/O)
```

   - Input Address (IN_ADDRESS_HEXA_KEYBOARD): Used to send polling signals to the keyboard
   - Output Address (OUT_ADDRESS_HEXA_KEYBOARD): Reads the pressed key.

2. **Initial data**

```
key_map: # Key to map for select song (1-4) or pause (0)
    .word 0x21, song1       # Key 1 -> Song 1
    .word 0x41, song2       # Key 2 -> Song 2
    .word 0x81, song3       # Key 3 -> Song 3
    .word 0x12, song4       # Key 4 -> Song 4
    .word 0x11, pause_song  # Key 0 -> Pause
    .word 0x00               # End of the key mapMe
```

The key_map section maps specific keys to corresponding actions:

- Keys 1-4 → Play songs 1-4 correspond to 4 addresses of 4 button 1->4.
- Key 0 → Pause playback (0x11).
- If the pressed key does not match any in the map, an error message is displayed.

I prepared four music tracks include (Happy birthday, Jingle Belle, Little Star, Happy New Year (English folk song) with format of each note [pitch, duration, instrument, and volume]

```
error_msg: .asciz "Invalid key pressed. Please press 0-4 to select a song
or pause.\n" # Error message for invalid key

pause_msg: .asciz "Song da paused\n" # Message for pausing the song
```

Message that will print if press invalid button and message when pause (press button 0).

3. **Main program**

```
main:
    li s1, IN_ADDRESS_HEXA_KEYBOARD # Load the input address for the keyboard
    li s2, OUT_ADDRESS_HEXA_KEYBOARD # Load the output address for the
keyboard
    li t3, 0x01 #Set initial bit for key press checking (Start from row 1)

polling:
    sb t3, 0(s1) # Write the polling bit to the input address
    lbu a0, 0(s2) # Read the key press from the output address

    beq a0, zero, next_row # If no key is pressed, move to the next row

    li t1, 0x11
    beq a0, t1, pause_music # If key 0 is pressed, pause the song

    jal ra, find_song  # jump to find_song subroutine to find corresponding
song
    beq a0, zero, invalid_key # If the key is not found, print an error
message
    jal ra, play_song # Play the song

    li a0, 400 # Delay for 400ms
    li a7, 32
    ecall

next_row:
    slli t3, t3, 1 # Move to the next row (shift bit to check the next key)
    li t4, 0x10 # Check if all rows have been checked
    bne t3, t4, polling # If not, continue polling
    li t3, 0x01 # Reset bit polling to check from the start
    j polling # Continue polling
```

Initalization:

- S1 and s2: Load the keyboard I/O address
- T3: Initialize to 0x01 to start polling from the row 1.

Polling loop:

- Send Polling Signal (sb t3, 0(s1)): is to activate the current row
- Read keypress (lbu a0, 0(s2)): is to check if any key is pressed.

Handle Key event:

- If no key is pressed (a0 ==0) then go to next row
- If key a0 is press and is key 0 then jump to the pause_music
- For other key:
    - Jump to find song which matching in key_map and push current register to ra to return after jr ra.
    - If not find the song then jump to invalid key.

New row:

- Shift t3 left to poll the next row
- And reset the t3 to first row (0x01) after all rows are checked and continue polling.

4. Subroutine:

A, Pause_music

```
pause_music:

   li a7, 4     # Print the pause message
   la a0, pause_msg
   ecall

    li a0, 500 # Delay for 500ms
    li a7, 32
    ecall
   j polling
```

- Display Pause message: Output is the message that store in **pause_msg**
- After print this message will pause little time (500ms) before return polling.

B. invalid_key

```
invalid_key:

   li a7, 4 # Print the error message
   la a0, error_msg
   ecall
   j polling # REturn to polling for new input
```

- Display Error: Outputs **error-msg** when the key is valid

C. Find_song

```
find_song:

    la t0, key_map # Load the address of key map
find_song_loop:
    lw t1, 0(t0) # Load the key from the key map
    beq t1, zero, not_found # If the key is not found, return 0
    bne a0, t1, next_entry # If the key is not found, jump to next_entry
    lw a0, 4(t0) # Load the address of the song if match
    jr ra

next_entry:
    addi t0, t0, 8 # Move to the next entry in the key map
    j find_song_loop
not_found:
    li a0, 0 # Return 0 if the key is not found
    jr ra
```

- First load the address of key map to **t0**
- Make a loop to find song:
    - Load the key from the key map to **t1**
    - And check that if can not found the key (when come to the last address), then jump to **not_found** to reset a0 to 0 then return back the `jal ra, find_song` in the main program
    - Check if that key is not the key we want then jump to next_entry to find again,
    - Then if match them load the address of the song to a0 then return back the `jal ra, find_song`
- Next_entry: add t0 with 8 to move to the next entry in the key map and jump to back to the find_song_loop for finding.
- Not_found: assign a0 to 0 then return back .

D. Play_song

```
play_song:
    mv t0, a0 # Load song data address
play_song_loop:
    lbu t1, 0(s2) # Check if the key is still pressed (if not, stop)
    beq t1, zero, return # if no key is pressed, exit

    lw t1, 12(t0) # Get the volume of the current note
    beq t1, zero, return # If the volume is 0, the song has ended

    lw a0,0(t0) # Load the pitch
```

```
    lw a1,4(t0) # Load the duration
    lw a2,8(t0) # Load the instrument type
    lw a3,12(t0) # Load the volume

    li a7, 31 # System call for playing the note
    ecall

    addi t0, t0, 16 # Move to the next note
    j play_song_loop

return:
    jr ra
```

- First load song data address a0 to t0
- Play_song_loop:
    - Load byte unsigned t1 from 0(s2) to check if the key is still pressing or not, if t1 = 0x00 (mean that is not still press) then jump to return and not display the music.
    - Load t1 to get the volume of the current note (we define the last note that have volume 0 the song is end (jump to return).
    - Load the value the pitch, duration, instrument type, volume to a0, a1, a2, a3 and using system call MidiOut to play the note.
    - Addi t0, t0, 16 to move to the next note and jump back to play_song_loop to play another note.

III. Simulation Results

The program can do all the requirements of the task. To take the input, I use the Digital Lab Sim. When the system is running, it will wait for the input through this. The program plays a song based on the key that presses from (1-4) and button 0 is to pause music.

Additionally, this project also handles when pressing invalid key, an error message will show.

3.1 Sample Test:

1. When pressing key 1: Song 1 plays.
2. When pressing key 2: Song 1 plays
3. When pressing key 3: Song 1 plays
4. When pressing key 4: Song 1 plays
5. When pressing key 0: Pause the music and display the message “Song da paused”
6. When pressing invalid key: Displays an error message: “Invalid key pressed. Please press 0-4 to select a song or pause.”

I create a demo video to simulate in this link.

3.2 Limitations:

- Polling dependency:
  - Continuously scanning the keyboard wastes processing time
  - Could be replaced with an **interrupt-based approach** for better efficiency (reference to lab 11 Home Assignment 2 3 4) but I still can't implement this approach.
- Error: The program loops immediately after invalid input, potentially causing multiple error messages if not carefully managed.

3.3 Lesson learns:

- Know how to implement POLLING to project
- Know how to use the system called MidiOut.
- Have experience about music note, know to create a note need some components like (pitch, duration, instrument, volume.

IV. Source code:

```
.eqv IN_ADDRESS_HEXA_KEYBOARD  0xFFFF0012  #Input addresss for the keyboard
(Memory-mapped I/O)
.eqv OUT_ADDRESS_HEXA_KEYBOARD 0xFFFF0014  #Output address for the keyboard
(Memory-mapped I/O)

.data
key_map: # Key to map for select song (1-4) or pause (0)
    .word 0x21, song1       # Key 1 -> Song 1
    .word 0x41, song2       # Key 2 -> Song 2
    .word 0x81, song3       # Key 3 -> Song 3
    .word 0x12, song4       # Key 4 -> Song 4
    .word 0x11, pause_song  # Key 0 -> Pause
    .word 0x00               # End of the key map
pause_song:

song1: # Song data : [Pitch, Duration , Instrument Type, Volume]
    # Happy birthday
    .word 60, 500, 0, 100    # C4 (Middle C) - "Happy"
    .word 60, 500, 0, 100    # C4 - "Birthday"
    .word 62, 1000, 0, 100   # D4 - "To"
    .word 60, 1000, 0, 100   # C4 - "You"
    .word 65, 1000, 0, 100   # F4 - "Happy"
    .word 64, 2000, 0, 100   # E4 - "Birthday"

    .word 60, 500, 0, 100    # C4 - "Happy"
    .word 60, 500, 0, 100    # C4 - "Birthday"
```

```
    .word 62, 1000, 0, 100   # D4 - "To"
    .word 60, 1000, 0, 100   # C4 - "You"
    .word 67, 1000, 0, 100   # G4 - "Happy"
    .word 65, 2000, 0, 100   # F4 - "Birthday"

    .word 60, 500, 0, 100    # C4 - "Happy"
    .word 60, 500, 0, 100    # C4 - "Birthday"
    .word 72, 1000, 0, 100   # C5 - "Dear"
    .word 69, 1000, 0, 100   # D4
    .word 65, 1000, 0, 100   # F4 - "Happy"
    .word 64, 1000, 0, 100   # E4 - "Birthday"
    .word 62, 2000, 0, 100   # D4 - "To You"
    .word 0,0,0 ,0  # End of the song

song2: # Jingle Belles
    .word 76, 500, 0, 100    # E5
    .word 76, 500, 0, 100    # E5
    .word 76, 1000, 0, 100   # E5
    .word 76, 500, 0, 100    # E5
    .word 76, 500, 0, 100    # E5
    .word 76, 1000, 0, 100   # E5
    .word 76, 500, 0, 100    # E5
    .word 79, 500, 0, 100    # G5
    .word 72, 500, 0, 100    # C5
    .word 74, 500, 0, 100    # D5
    .word 76, 1000, 0, 100   # E5

    # Oh what fun it is to ride in a one-horse open sleigh
    .word 77, 500, 0, 100    # F5
    .word 77, 500, 0, 100    # F5
    .word 77, 500, 0, 100    # F5
    .word 77, 500, 0, 100    # F5
    .word 76, 500, 0, 100    # E5
    .word 76, 500, 0, 100    # E5
    .word 76, 1000, 0, 100   # E5

    # Refrain
    .word 76, 500, 0, 100    # E5
    .word 74, 500, 0, 100    # D5
    .word 74, 500, 0, 100    # D5
    .word 76, 500, 0, 100    # E5
    .word 74, 500, 0, 100    # D5
    .word 77, 1000, 0, 100   # F5
    .word 76, 500, 0, 100    # E5
    .word 72, 500, 0, 100    # C5
```

```
    .word 72, 500, 0, 100    # C5
    .word 74, 500, 0, 100    # D5
    .word 72, 500, 0, 100    # C5
    .word 76, 1000, 0, 100   # E5
    .word 0, 0, 0, 0           # End of song

song3: #Little Star
    # Twinkle, Twinkle, Little Star
    .word 72, 500, 0, 100    # C5
    .word 72, 500, 0, 100    # C5
    .word 79, 500, 0, 100    # G5
    .word 79, 500, 0, 100    # G5
    .word 81, 500, 0, 100    # A5
    .word 81, 500, 0, 100    # A5
    .word 79, 1000, 0, 100   # G5

    # How I wonder what you are
    .word 77, 500, 0, 100    # F5
    .word 77, 500, 0, 100    # F5
    .word 76, 500, 0, 100    # E5
    .word 76, 500, 0, 100    # E5
    .word 74, 500, 0, 100    # D5
    .word 74, 500, 0, 100    # D5
    .word 72, 1000, 0, 100   # C5

    # Up above the world so high
    .word 79, 500, 0, 100    # G5
    .word 79, 500, 0, 100    # G5
    .word 77, 500, 0, 100    # F5
    .word 77, 500, 0, 100    # F5
    .word 76, 500, 0, 100    # E5
    .word 76, 500, 0, 100    # E5
    .word 74, 1000, 0, 100   # D5

    # Like a diamond in the sky
    .word 79, 500, 0, 100    # G5
    .word 79, 500, 0, 100    # G5
    .word 77, 500, 0, 100    # F5
    .word 77, 500, 0, 100    # F5
    .word 76, 500, 0, 100    # E5
    .word 76, 500, 0, 100    # E5
    .word 74, 1000, 0, 100   # D5
    .word 0, 0, 0, 0           # End of song data

song4: # Happy New Year (English folk song)
```

```
    # Opening melody - "Happy New Year, Happy New Year"
    .word 72, 500, 0, 100    # C5
    .word 74, 500, 0, 100    # D5
    .word 76, 1000, 0, 100   # E5
    .word 74, 500, 0, 100    # D5
    .word 72, 1000, 0, 100   # C5

    # Second line - "May we all have a vision now and then"
    .word 74, 500, 0, 100    # D5
    .word 76, 500, 0, 100    # E5
    .word 77, 1000, 0, 100   # F5
    .word 76, 500, 0, 100    # E5
    .word 74, 1000, 0, 100   # D5

    # Chorus - "Of a world where every neighbor is a friend"
    .word 76, 500, 0, 100    # E5
    .word 79, 500, 0, 100    # G5
    .word 77, 1000, 0, 100   # F5
    .word 76, 500, 0, 100    # E5
    .word 74, 1000, 0, 100   # D5

    # Repeat melody
    .word 72, 500, 0, 100    # C5
    .word 74, 500, 0, 100    # D5
    .word 76, 1000, 0, 100   # E5
    .word 74, 500, 0, 100    # D5
    .word 72, 1000, 0, 100   # C5
    .word 0, 0, 0, 0           # End of song data

error_msg: .asciz "Invalid key pressed. Please press 0-4 to select a song or
pause.\n" # Error message for invalid key
pause_msg: .asciz "Song da paused\n" # Message for pausing the song

.text
.global main

main:
    li s1, IN_ADDRESS_HEXA_KEYBOARD # Load the input address for the keyboard
    li s2, OUT_ADDRESS_HEXA_KEYBOARD # Load the output address for the
keyboard
    li t3, 0x01 #Set initial bit for key press checking (Start from row 1)

polling:
    sb t3, 0(s1) # Write the polling bit to the input address
    lbu a0, 0(s2) # Read the key press from the output address
```

```
    beq a0, zero, next_row # If no key is pressed, move to the next row

    li t1, 0x11
    beq a0, t1, pause_music # If key 0 is pressed, pause the song

    jal ra, find_song  # jump to find_song subroutine to find corresponding
song
    beq a0, zero, invalid_key # If the key is not found, print an error
message
    jal ra, play_song # Play the song

    li a0, 400 # Delay for 400ms
    li a7, 32
    ecall

next_row:
    slli t3, t3, 1 # Move to the next row (shift bit to check the next key)
    li t4, 0x10 # Check if all rows have been checked
    bne t3, t4, polling # If not, continue polling
    li t3, 0x01 # Reset bit polling to check from the start
    j polling # Continue polling

pause_music:
   li a7, 4     # Print the pause message
   la a0, pause_msg
   ecall

    li a0, 500 # Delay for 500ms
    li a7, 32
    ecall
   j polling

invalid_key:
    li a7, 4 # Print the error message
    la a0, error_msg
    ecall
    j polling # REturn to polling for new input

find_song:
    la t0, key_map # Load the address of key map
find_song_loop:
    lw t1, 0(t0) # Load the key from the key map
    beq t1, zero, not_found # If the key is not found, return 0
    bne a0, t1, next_entry # If the key is not found, jump to next_entry
```

```
    lw a0, 4(t0) # Load the address of the song if match
    jr ra

next_entry:
    addi t0, t0, 8 # Move to the next entry in the key map
    j find_song_loop
not_found:
    li a0, 0 # Return 0 if the key is not found
    jr ra

play_song:
    mv t0, a0 # Load song data address
play_song_loop:
    lbu t1, 0(s2) # Check if the key is still pressed (if not, stop)
    beq t1, zero, return # if no key is pressed, exit

    lw t1, 0(t0) # Get the pitch of the current note
    beq t1, zero, return # If the pitch is 0, the song has ended

    lw a0,0(t0) # Load the pitch
    lw a1,4(t0) # Load the duration
    lw a2,8(t0) # Load the instrument type
    lw a3,12(t0) # Load the volume

    li a7, 31 # System call for playing the note
    ecall

    addi t0, t0, 16 # Move to the next note
    j play_song_loop

return:
    jr ra
```