Lab1 Report

Shihao luo

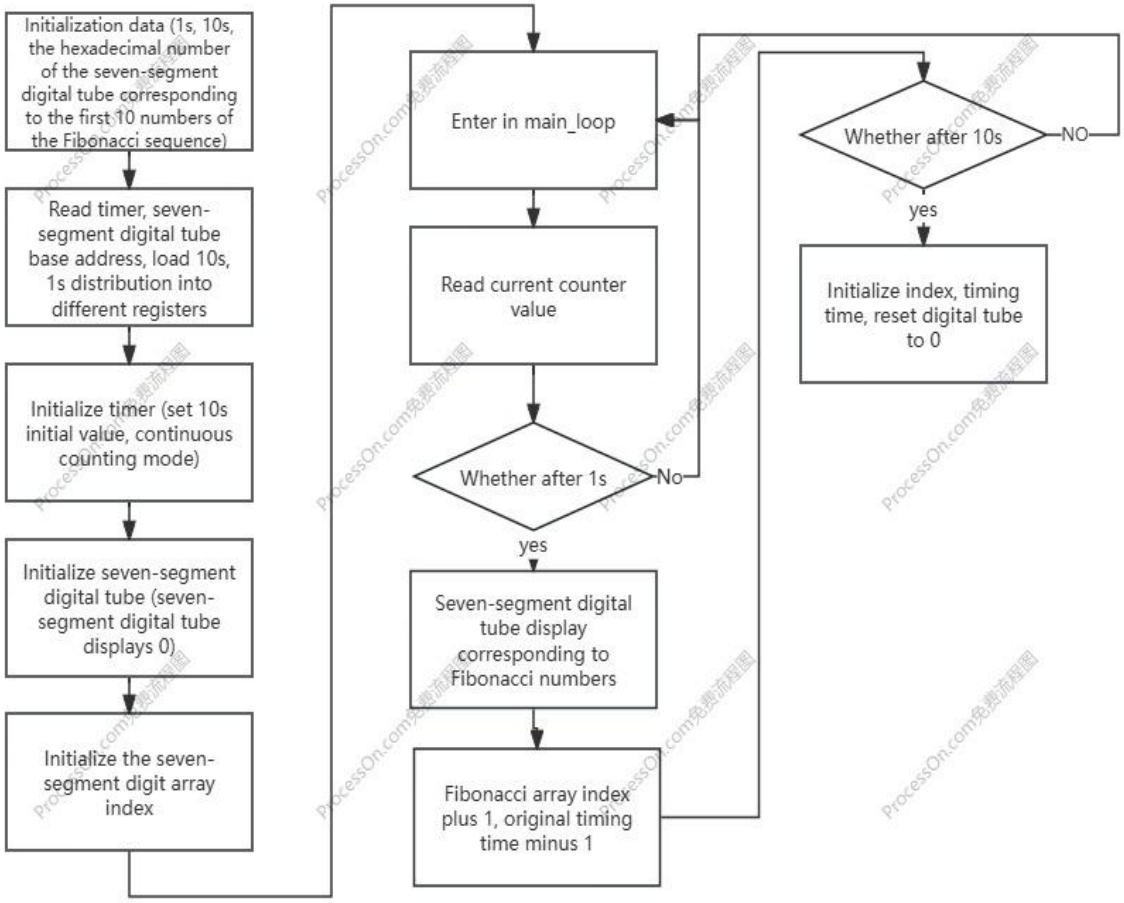
Statement

Write the first 10 Fibonacci numbers to a seven segment display. Show each number on the display for one second (1 s) before showing the next. Your program should endlessly loop, cycling through displaying these 10 numbers.

The Fibonacci numbers are deffned as Fn:

$$F0 = 0, F1 = 1, \text{and } Fn = Fn-1 + Fn-2 \text{ for } n > 1.$$

Explanation



Approached and implemented the problem

In this project, I followed a structured approach to implement the required functionality in four main steps: calculating Fibonacci numbers, implementing the timer for 1-second pauses, activating the seven-segment display, and indexing the displayed Fibonacci digits.

Step 1: Calculating the Fibonacci Numbers
The first step was to calculate the first 10 Fibonacci numbers. This part was relatively straightforward. I implemented a simple iterative function using the Fibonacci formula to generate the sequence and stored the results in an array. I tested this function independently by printing the values to ensure the correct sequence was generated before moving on to the next step.

Step 2: Setting the Timer for 1-Second Pauses
Next, I needed to create a 10-second interval with 1-second pauses between displaying each Fibonacci digit. I referred to the example code from Lesson 5, which demonstrated using a timer to measure a 5-second interval and sequentially light up LEDs every second. I adapted this code by extending the interval to 10 seconds. During testing, I used LEDs to visualize each 1-second interval before integrating this timing mechanism with the seven-segment display.

Step 3: Activating the Seven-Segment Display
To display the Fibonacci numbers on the seven-segment display, I first mapped the Fibonacci digits to their corresponding hexadecimal codes needed by the display. I consulted the lesson notes to understand how to properly address the seven-segment display. After storing the digits in the display's base address one by one, I tested this functionality by manually writing a few test values to confirm they were displayed correctly.

Step 4: Indexing the Fibonacci Digits for Display
The final step was to automate the display of each Fibonacci digit in sequence. I stored all the hexadecimal digits in an array and created an index variable to loop through the data. Initially, I struggled with this step, as I wasn't sure how to correctly update the index with each 1-second timer interrupt. After experimenting with different approaches, I successfully implemented a counter within the timer interrupt service routine, allowing each digit to display sequentially.

Debugging and Challenges
One of the main challenges I faced was ensuring that the Fibonacci digits looped correctly on the seven-segment display. I initially tested the index logic in isolation by outputting the index values to the console. Once I confirmed the index was updating correctly, I integrated this logic with the display code and observed the

actual display output. Debugging tools such as breakpoints and step-by-step execution helped me identify and resolve logic errors.

In conclusion, by breaking down the project into smaller, testable components, I was able to systematically implement and verify each part. This approach helped me maintain clarity throughout the development process and effectively troubleshoot any issues that arose.

TIME CONSUMPTION AND FUTHER INPROVEMENT

For this project, I spent the first hour reviewing the relevant concepts, as I had forgotten most of the details about timers and the seven-segment display. During this time, I also analyzed my approach and outlined my solution. Initially, I planned to complete the coding within an hour. However, I encountered challenges with managing the array index, which extended the total development time to three hours.

Although I successfully completed the project, I recognize a significant limitation in my current implementation. As it stands, if I wanted to display the first 100 (or more) Fibonacci digits, I would need to manually set 100 hexadecimal values beforehand. This approach is not only impractical but also difficult to maintain.

I believe it would be worthwhile to improve this aspect of my code. Developing a more dynamic and scalable solution would not only make my code more efficient but also deepen my understanding of assembly language and hardware interactions. Investing time in this enhancement would ultimately benefit my long-term learning and coding practices.

Code

```
.global _start


.data

timer_T:        .word 1000000000        @ 10s

timer_sT:       .word 100000000         @ 1s
```

```
fib_7seg:
.word 0x3F3F      @ 0    -> "00"
.word 0x3F06      @ 1    -> "01"
.word 0x3F06      @ 1    -> "01"
.word 0x3F5B      @ 2    -> "02"
.word 0x3F4F      @ 3    -> "03
.word 0x3F6D      @ 5    -> "05"
.word 0x3F7F      @ 8    -> "08"
.word 0x064F      @ 13 -> "13"
.word 0x5B06      @ 21 -> "21"
.word 0x4F66      @ 34 -> "34"



.text
_start:
@ 1) set the base address
ldr         r4, =0xff202000             @ timer's base address
ldr         r5, =0xff200020             @ seven-segement displays' base address
@ 2) ldr 10s and 1s' address
ldr         r6,    adr_T
ldr         r7,    adr_sT
@ ----------------------
@ initialize timer
@ ----------------------
```

```
ldr         r0, [r6]

str         r0, [r4, #8]

mov          r1, r0, lsr #16

str         r1, [r4, #12]

@ set the mode of timer

mov          r1, #6

str         r1, [r4, #4]

@ ldr 1s

ldr         r1, [r7]

@initialize seven_segement

ldr         r8, =0x3f3f

str         r8, [r5]

@ initialize fib_index

mov          r9, #0

main_loop:

@ ----------------------

@ load current value of counter

@ ----------------------

str         r1, [r4, #16]

ldr         r2, [r4, #16]

ldr         r3, [r4, #20]

add          r2, r2, r3, lsl #16

@ 1s

cmp          r2, r0
```

```
bhi         main_loop
@ show corresponding fibonacci nums in seven-segement displays
ldr         r8, =fib_7seg
add         r8, r8, r9, lsl #2
ldr         r8, [r8]
str         r8, [r5]

@ fib_index++
add         r9, r9, #1
cmp         r9, #11
@ sub 1s
sublo         r0, r1
@ check if it has shown 10 digits
blo         main_loop
@ when it has looped more than 10 times, it should be initialized
ldr         r0, [r6]
mov         r9, #0
ldr         r8, =0x3f3f
str         r8, [r5]
b           main_loop

adr_T:          .word timer_T
adr_sT:         .word timer_sT
```