ECE9047: Laboratory 1

Joshua Bainbridge

Problem Statement

The goal of this lab is to is to code an assembly language program to 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.

Solutions

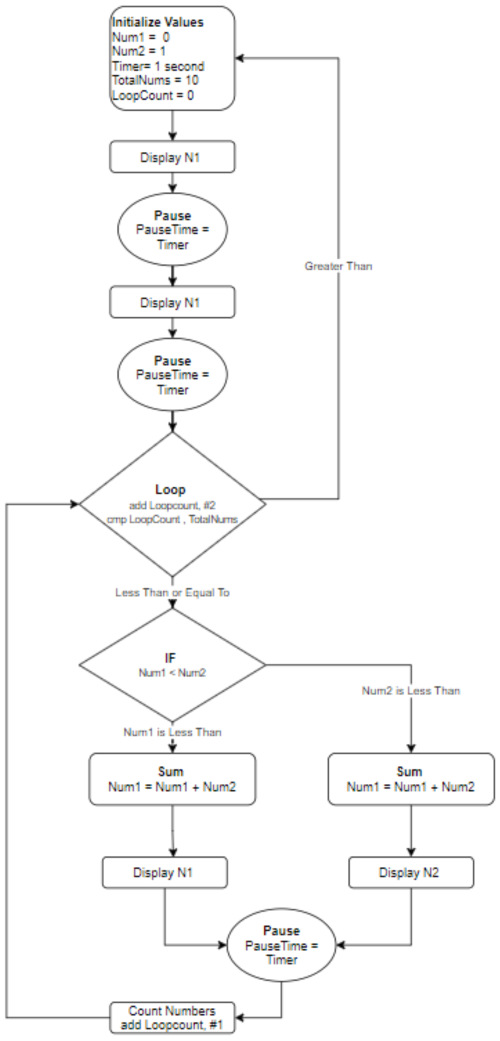
The flow chart on the right outlines the initial solution idea for this lab. Two registers will first be initialized with starting values for the first two numbers in the Fibonacci sequence as well as the total of numbers to be displayed and the time delay between each. One more register will be required to act as loop counter.

The program with then display the first two numbers with a pause in-between each. The program should now increment the loop counter by 2.

The program will then compare the loop counter to the total number of loops value. If it is greater than the count the program will loop back to the start. If there are still more numbers to display the program will continue to loop.

The program will then check to see if Num1 or Num2 is greater. If Num1 is less than Num2 then the Num1 will be updated to be Num1 + Num2. Example. The first loop when Nu1 = 0 and Num2 =1 . Num1 is less than Num2 therefore Num1 = 0 + 1. Then Num1 is displayed. The same procedure occurs if the opposite is true.

Finally after being displayed and paused the program will update the loop counter and branch back to the loop so the program will loop forever.

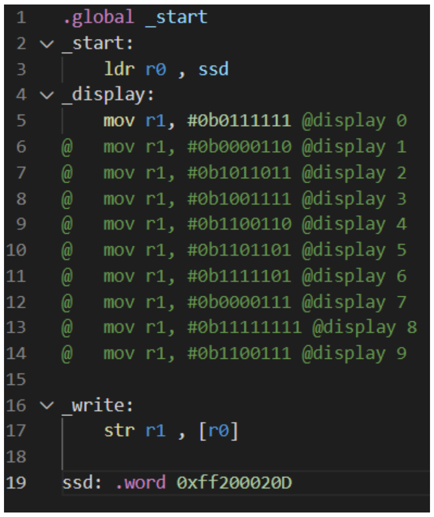


This solution to the problem presents a lot of additional steps in order to implement. Prior to the lab extension I coded a different solution. This solution simply had all 10 numbers hard coded into the program and it looped continuously. This code as also been added to the appendix as Solution 1.

Technical Approach

Seven Segment Display

The seven-segment display was the part of the lab that was coded first. This was done because it is the most straight forward part of the code and besides reading the memory mapping the display is the only feedback given to determine if the program is functioning properly. The code used to test the display is shown below.



```
.global _start
_start:
    ldr r0 , ssd
_display:
    mov r1, #0b0111111 @display 0
@   mov r1, #0b0000110 @display 1
@   mov r1, #0b1011011 @display 2
@   mov r1, #0b1001111 @display 3
@   mov r1, #0b1100110 @display 4
@   mov r1, #0b1101101 @display 5
@   mov r1, #0b1111101 @display 6
@   mov r1, #0b0000111 @display 7
@   mov r1, #0b11111111 @display 8
@   mov r1, #0b1100111 @display 9

_write:
    str r1 , [r0]

ssd: .word 0xff200020D
```

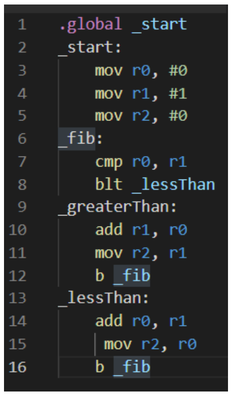
Every digit, 0 through 9 was tested to ensure all the value would display properly when implemented as a subroutine in the final program.

Timer

The next part of the program to be coded was the timer. The timer was coded next because it will be implemented as a subroutine later in the coding. The program was coded before the timer lectures. A one second pause was created by have the program count up from zero. Based on reading and texting with the emulator, the clock frequency was found to be roughly 1.2 MHz. This testing largely involved the black box technique to get the correct value. By altering the input: count value and timing the output as final clock speed and delay was found. The pause subroutine therefore counts up to 0x124f80. Not using the internal clock is not an ideal solution because counting uses processor resources that are not necessary. For this lab it has not impact because the process does not have to do any functions during the pause, but it would make if difficult to add functionality to the code.

Fibonacci Numbers

A small program was coded to test the formula being used to generate any number of Fibonacci numbers. The code can be seen below.

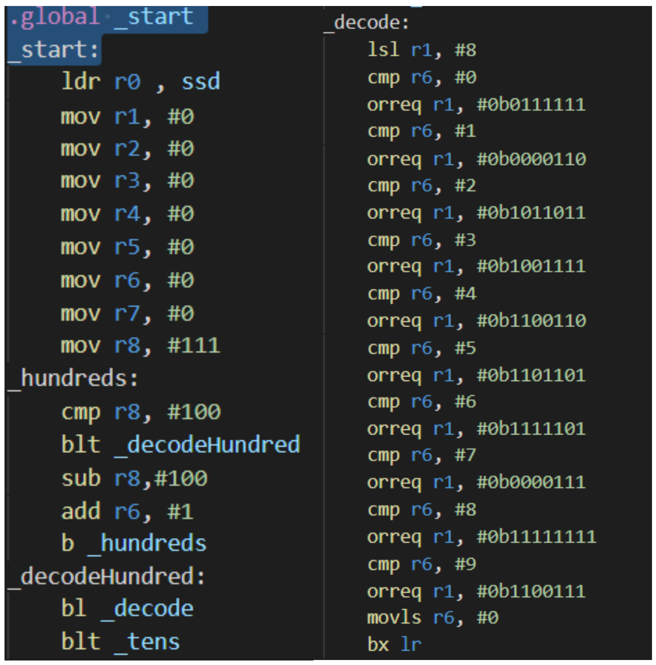


```
.global _start
_start:
    mov r0, #0
    mov r1, #1
    mov r2, #0
_fib:
    cmp r0, r1
    blt _lessThan
_greaterThan:
    add r1, r0
    mov r2, r1
    b _fib
_lessThan:
    add r0, r1
     mov r2, r0
    b _fib
```

The Fibonacci code was tested using the white box technique along with breakpoints and stepping through the program, while observing the change in register values.

<u>Decimal Decoder</u>

The final technical section of the code is a decimal decoder used to properly display the numbers of a seven-segment display. The subroutine created to decode the values can be seen below. To implement the decoder, the hundreds digit was isolated first and then subtracted from the total. The value of the hundreds digit was then translated into its seven-segment display value and then shifted into one register. The same process was repeated for the tens and one's digits. The code used to test this can be seen below.



```
.global _start
_start:
    ldr r0 , ssd
    mov r1, #0
    mov r2, #0
    mov r3, #0
    mov r4, #0
    mov r5, #0
    mov r6, #0
    mov r7, #0
    mov r8, #111
_hundreds:
    cmp r8, #100
    blt _decodeHundred
    sub r8,#100
    add r6, #1
    b _hundreds
_decodeHundred:
    bl _decode
    blt _tens
```

```
_decode:
    lsl r1, #8
    cmp r6, #0
    orreq r1, #0b0111111
    cmp r6, #1
    orreq r1, #0b0000110
    cmp r6, #2
    orreq r1, #0b1011011
    cmp r6, #3
    orreq r1, #0b1001111
    cmp r6, #4
    orreq r1, #0b1100110
    cmp r6, #5
    orreq r1, #0b1101101
    cmp r6, #6
    orreq r1, #0b1111101
    cmp r6, #7
    orreq r1, #0b0000111
    cmp r6, #8
    orreq r1, #0b11111111
    cmp r6, #9
    orreq r1, #0b1100111
    movls r6, #0
    bx lr
```

This process required both black and white box debugging. Additionally random numbers were put into the register to test the decoder outside of the required Fibonacci numbers. During the debugging the emulator required the use of additional registers. To ensure these new registers were added correctly and the program would work breakpoint were place to pause the program so the values of the new registers could be checked.

Cost

Given the extra time allotted to the finish the lab I was able to finish both the hard coded and alterable solution.

The hard-coded solution had a much lower time cost when comparted with the alterable solution. The is partially because the hard-coded solution only required the implementation of the first two subroutines discussed in the technical approach. On the other hand it will be extremely difficult to add more functionality to the code.

The alterable program took a significant amount more time to finish, to product the same results. It is also limited only three seven-segment display digits. This program has more flexibility and is more portable, but it also requires many more resources from the micro controller itself. The code is not full optimized and uses all twelve available registers. Compared to the three registers used in the hard-coded, the alterable solution requires more resources.

The best solution when considering the cost for this problem is the hardcoded solution.

Appendix

SOLUTION 1: Hardcoded Solution

```
.global _start
_start:
        ldr r0 , ssd                    @SEVEN SEGMENT DISPLAY
        mov r2, #0                                  @TIMER COMPARE
        ldr r3, =0x124f80    @TIMER VALUE
_loop:
        mov r1, #0b0111111              @SEVEN SEGMENT --> 0
        bl _write
        bl _pause
        mov r1, #0b0000110 @SEVEN SEGMENT --> 0
        bl _write
        bl _pause
        mov r1, #0b0000110              @SEVEN SEGMENT --> 0
        bl _write
        bl _pause
        mov r1, #0b1011011              @SEVEN SEGMENT --> 0
        bl _write
        bl _pause
        mov r1, #0b1001111              @SEVEN SEGMENT --> 0
        bl _write
        bl _pause
        mov r1, #0b1101101              @SEVEN SEGMENT --> 0
        bl _write
        bl _pause
        mov r1, #0b1111111              @SEVEN SEGMENT --> 0
        bl _write
        bl _pause
        ldr r1, =0b000011001001111 @SEVEN SEGMENT --> 0
        bl _write
        bl _pause
        ldr r1, =0b101101100000110 @SEVEN SEGMENT --> 0
        bl _write
        bl _pause
        ldr r1, =0b100111101100110 @SEVEN SEGMENT --> 0
        bl _write
        bl _pause
        b _loop
_pause:
        add r2, #1  @Increment clock counter
        cmp r2, r3  @Compare to clock limit
        ble _pause
        mov r2, #0 @reset clock counter
        bx lr
_write:
        str r1 , [r0]   @Display values
        bx lr
ssd: .word 0xff200020
```

SOLUTION 2: Alterable/Decoding Solution

```
.global _start
_start:
        ldr r0 , ssd                    @seven segment display address
        mov r1, #0                              @display value
        mov r2, #0                              @fibonacci number 1
        mov r3, #1                              @fibonacci number 2
        mov r4, #10                             @number of sequences
        mov r5, #0                              @timer count up value
        ldr r6, =0x124f80           @timer value
        mov r7, #0                              @save pc address
        mov r8, #0                              @to be decoded value
        mov r9, #0                              @decode value 100s
        mov r10, #0                             @decode value 10s
        mov r11, #0                             @decode value 1s
        mov r12, #0                             @DECODE VALUE
_opening:
        sub r4, #2                              @reduce number of sequence by 2 for display 0, 1
        orr r8, r2                      @store Fibonacci numbe to be decoded
        mov r7, pc                              @save PC pointing to orr r8, r3
        add r7, #4
        b _hundreds                             @begin to deocde Fibonacci numbers for display
        orr r8, r3
        mov r7, pc
        add r7, #4                              @save PC pointing to _sequence
        b _hundreds                             @begin to deocde Fibonacci numbers for display
        b _sequence                             @subroutine that completes remaining sequence
_sequence:
        cmp r4, #0
        beq _start                              @restart loop from 0
        add r2,r3                       @ Num1 = Num1 + Num2
        @display r0 --> 1 2 5 13 34
        orr r8, r2                      @store Fibonacci numbe to be decoded
        add r4, #-1
        mov r7, pc
        add r7, #4
        b _hundreds
        add r3,r2                       @ Num2 = Num1 + Num2
        @display r1 --> 1 3 8 21
        orr r8, r3                      @store Fibonacci numbe to be decoded
        add r4, #-1
        mov r7, pc
        add r7, #4
        b _hundreds
        @pause 1s
        cmp r4, #0
        beq _start                              @restart loop from 0
        b _sequence
_hundreds:
        mov r12,r9                              @store 100s digit
        cmp r8, #100
        blt _decodeHundred
        sub r8,#100                             @subtract 100 from Fibonacci number
        add r9, #1                          @increase 100s digit by 1
        b _hundreds
_decodeHundred:
        bl _decode
        b _tens
_tens:
        mov r12,r10                         @store 10s digit
        cmp r8, #10
        blt _decodeTen
        sub r8,#10                          @subtract 10 from Fibonacci number
        add r10, #1                         @increase 10s digit by 1
        b _tens
_decodeTen:
        bl _decode
        blt _ones
_ones:
        mov r12,r11                         @store 1s digit
        cmp r8, #1
        blt _decodeOne
        sub r8,#1                   @subtract 1 from Fibonacci number
        add r11, #1                         @increase 1s digit by 1
        b _ones
_decodeOne:
        bl _decode
        b _write
_decode:
        lsl r1, #8                  @logical shift left by 8 digits
        cmp r12, #0                 @decoder compares digit to 0 to 9
        orreq r1, #0b0111111    @OR seven-seg value with 00000000
        cmp r12, #1
        orreq r1, #0b0000110
        cmp r12, #2
        orreq r1, #0b1011011
        cmp r12, #3
        orreq r1, #0b1001111
        cmp r12, #4
        orreq r1, #0b1100110
        cmp r12, #5
        orreq r1, #0b1101101
        cmp r12, #6
        orreq r1, #0b1111101
        cmp r12, #7
        orreq r1, #0b0000111
        cmp r12, #8
        orreq r1, #0b11111111
        cmp r12, #9
        orreq r1, #0b1100111
        movls r12, #0
        bx lr                               @return to spot
_write:
        str r1 , [r0]
        b _pause
_pause:
        add r5, #1                  @counting to vaue to pause program
        cmp r5, r6
        ble _pause
        mov r1, #0                      @rest digit holding values
        mov r5, #0
        mov r9, #0
        mov r10, #0
        mov r11, #0
        bx r7
ssd: .word 0xff200020
```