LAB 4

Fibonacci Numbers

4.1 Problem Statement

1. Write a program in LC-3 assembly language that computes F_n, the $n-$th Fibonacci number.
2. Find the largest F_n such that no overflow occurs, i.e. find $n = N$ such that F_N is the largest Fibonacci number to be correctly represented with 16 bits in two's complement format.

4.1.1 Inputs

The integer n is in memory location **x3100**:

x3100	n

4.1.2 Outputs

x3101	F_n
x3102	N
x3103	F_N

4.2 Example

x3100	6
x3101	8
x3102	N
x3103	F_N

Starting with 6 in location **x3100** means that we intend to compute F_6 and place that result in location **x3101**. Indeed, $F_6 = 8$. (See below.) The actual values of N and F_N should be found by your program, and be placed in their corresponding locations.

4.3 Fibonacci Numbers

The Fibonacci F_i numbers are the members of the Fibonacci sequence: $1, 1, 2, 3, 5, 8, \ldots$. The first two are explicitly defined: $F_1 = F_2 = 1$. The rest are defined according to this recursive formula: $F_n = F_{n-1} + F_{n-2}$. In words, each Fibonacci number is the sum of the two previous ones in the Fibonacci sequence. From the sequence above we see that $F_6 = 8$.

4.4 Pseudo-code

Quite often algorithms are described using *pseudo-code*. Pseudo-code is not real computer language code in the sense that it is not intended to be compiled or run. Instead, it is intended to describe the steps of algorithms at a high level so that they are easily understood. Following the steps in the pseudo-code, an algorithm can be implemented to programs in a straight forward way. We will use pseudo-code[1] in some of the labs that is reminiscent of high level languages such as C/C++, Java, and Pascal. As opposed to C/C++, where group of statements are enclosed the curly brackets "{" and "}" to make up a compound statement, in the pseudo-code the same is indicated via the use of indentation. Consecutive statements that begin at the same level of indentation are understood to make up a compound statement.

4.5 Notes

- Figure 4.1 is a schematic of the contents of memory.

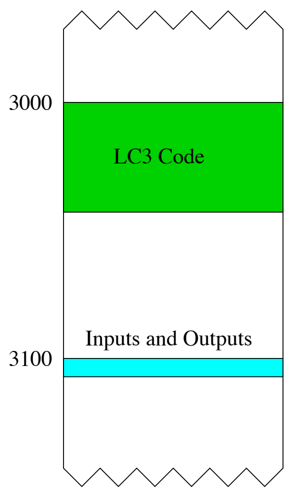

Figure 4.1: Contents of memory

- The problem should be solved by iteration using loops as opposed to using recursion.
- The pseudo-code for the algorithm to compute F_n is in listing 4.1. It is assumed that $n > 0$.

```
if n ≤ 2 then
    F ← 1
else
    a ← 1  // F_{n-2}
    b ← 1  // F_{n-1}
    for i ← 3 to n do
        F ← b + a  // F_n = F_{n-1} + F_{n-2}
        a ← b
        b ← F
```

Listing 4.1: Pseudo-code for computing the Fibonacci number F_n iteratively

[1]The pseudo-code is close to the one used in *Fundamentals of Algorithmics* by G. Brassard and P. Bratley, Prentice Hall, 1996.

- The way to detect overflow is to use a similar for-loop to the one in listing 4.1 on page 4–2 which checks when F first becomes negative, i.e. bit 16 becomes 1. See listing 4.2.
  Caution: upon exit from the loop, F does not have the value of F_N. To obtain F_N you have to slightly modify the algorithm in listing 4.2.

```
a ← 1  //  F_{n-2}
b ← 1  //  F_{n-1}
i ← 2  //loop index
repeat
    F ← b + a  //  F_n = F_{n-1} + F_{n-2}
    if F < 0 then
        N = i
        exit
    a ← b
    b ← F
    i ← i + 1
```

Listing 4.2: Pseudo-code for computing the largest $n = N$ such that F_N can be held in 16 bits

4.6 Testing

The table in figure 4.2 on page 4–4 will help you in testing your program.

4.7 What to turn in

- A hardcopy of the assembly source code.
- Electronic version of the assembly code.
- For each of $n = 15$ and $n = 19$, screen shots that show the contents of locations **x3100, x3101, x3102** and **x3103**, which show the values for F_{15} and F_{19}, respectively, and the values of N and F_N.

n	F_n	F_n in binary
1	1	0000000000000001
2	1	0000000000000001
3	2	0000000000000010
4	3	0000000000000011
5	5	0000000000000101
6	8	0000000000001000
7	13	0000000000001101
8	21	0000000000010101
9	34	0000000000100010
10	55	0000000000110111
11	89	0000000001011001
12	144	0000000010010000
13	233	0000000011101001
14	377	0000000101111001
15	610	0000001001100010
16	987	0000001111011011
17	1597	0000011000111101
18	2584	0000101000011000
19	4181	0001000001010101
20	6765	0001101001101101
21	10946	0010101011000010
22	17711	0100010100101111
23	28657	0110111111110001
24	46368	1011010100100000
25	75025	0010010100010001

Figure 4.2: Fibonacci numbers table