Lab2: Hofstadter Q

September 23, 2025

1 Overview

The purpose of this lab is to practice the use of LC-3 assembly **data movement instructions** (e.g., `LD` and `ST`) and **memory addressing modes** by implementing a recurrence relation. You will gain experience with:

- Loading and storing values between registers and memory.
- Using PC-relative addressing and base+offset addressing.
- Organizing memory to simulate arrays.
- Stepwise debugging in LC-3Tools.

2 Problem Description

Many number sequences are defined recursively. For example, the well-known **Fibonacci sequence** is defined as

$$F(0) = 0, \quad F(1) = 1, \quad F(n) = F(n-1) + F(n-2),\ n \geq 2.$$

Here, each term depends on the two immediately preceding terms, leading to a simple linear recurrence.

In this experiment, we focus on a more unusual recursive sequence: the **Hofstadter Q-sequence**. It is defined as

$$Q(1) = Q(2) = 1, \qquad Q(n) = Q(n - Q(n-1)) + Q(n - Q(n-2)), \quad n > 2.$$

Unlike the Fibonacci sequence, where the indices decrease by fixed amounts, in the Hofstadter Q-sequence the indices themselves depend on previously computed values of the sequence.

Your task is to write an LC-3 assembly program to compute the Q-sequence. The input value N is stored in memory location x3100, and your program must store the result Q(N) in x3101. Note that $1 \leq N \leq 100$

2.1 Examples

Input n @x3100	Expected result @ x3101
1	Q(1) = 1
5	Q(5) = 3
10	Q(10) = 6

3 submission

Compress and submit:

```
PB********_Name_lab2.zip
|-- PB********_Name_report.pdf
`-- lab2.asm
```

3.1 Requirements for Source Code

- Your program should start with .ORIG x3000.
- Your program should end with .END.
- Your last instruction should be TRAP x25 (HALT).
- Capitalized keywords(also labels) are recommended (For example, use "ADD" instead of "add", use "NUMBER" instead of "number").
- Spaces after commas (ADD R0, R0, #1 rather than ADD R0,R0,#1).
- Decimal constants start with #, hexadecimal with lowercase x.
- Write comments when necessary.

3.2 Requirement for Report

Your report should include

- Your name and student ID.
- A brief description of your solution.(e.g.,how do you store and load the array Q[n]).
- Results: Provide evidence of correct execution.(you can set N=100 and take some screenshots of your Q[n]).
- (Optional) Any challenges you encountered and how you addressed them.

4 Reference Approach

4.1 Implementation Steps

1. Allocate/define storage for Q[1..100]
2. Read n from memory.
3. If $n \leq 2$, write 1 to `x3101`and halt.
4. Set Q[1]=1, Q[2]=1.
5. For i = 3..n:
    - load q1 = Q(i-1)
    - t1 = i - q1
    - v1 = Q(t1)
    - load q2 = Q(i-2)
    - t2 = i - q2

- v2 = Q(t2)
- Q(i) = v1 + v2 (store)

6. Write Q(n) to `x3101` and halt.

4.2 Observations

When computing the Fibonacci sequence, we typically do not store the entire array in memory. Instead, since each new value only depends on the two most recent results, it is sufficient to keep $F(n-1)$ and $F(n-2)$ in registers.

A similar idea can be applied to the Hofstadter Q-sequence. Although its recurrence relation appears to involve multiple memory lookups for each $Q(n)$, two of these values — namely $Q(n-1)$ and $Q(n-2)$ — are always the most recently computed ones. Thus, they can be maintained directly in registers rather than reloaded from memory.

By doing so, the program not only becomes simpler but also more efficient: register access is much faster than memory access, and reducing the number of memory operations can significantly improve overall performance.

4.3 Overflow Considerations(Optional in your report)

If you compute the Fibonacci sequence, you will notice that its 100th term is $F(99) = 218922995834555169026$, which is far beyond the range that a 16-bit integer can represent. This naturally raises the question: does the Hofstadter Q-sequence exhibit a similar overflow problem? Should we consider applying a modulus operation to keep the results within range?

Plotting the Hofstadter Q-sequence can provide insight into its growth rate and overall behavior.