Lab 1: Conditional Move on LC-3

September 22, 2025

1 Overview

This lab familiarizes you with the LC-3 instruction set — particularly arithmetic/logical and branching instructions — by implementing a simple conditional expression in **machine code only**. Your program should accept two integer inputs and a boolean input and, based on the the boolean's value, produce one of the two integers as the result. You must encode your program as a sequence of 16-bit binary instructions that can be executed by an LC-3 simulator.

2 Problem Description

As in the C programming language, the conditional expression `cond ? val1 : val2` selects either `val1` or `val2`. Specifically, given 16-bit two's-complement integers `val1` and `val2` and a boolean `cond`, the expression evaluates to

$$\text{result} = \begin{cases} \texttt{val1}, & \text{if } \texttt{cond} = 1, \\ \texttt{val2}, & \text{if } \texttt{cond} = 0. \end{cases}$$

Your task is to write an LC-3 program that implements this conditional expression, where

- `cond` is guaranteed to be either **0** or **1** and is initially in **register `R0`**.
- `val1` is an arbitrary 16-bit two's-complement integer in **`R1`**.
- `val2` is in **`R2`**.
- The program should store the result in **`R0`**.

2.1 Examples

`cond` (`R0`)	`val1` (`R1`)	`val2` (`R2`)	Expected result (`R0`)
0	5	9	9
1	5	9	5
1	−3	4	−3
0	−7	−2	−2

2.2 Requirements and Tips

- The program must start at address `x3000` (binary `0011 0000 0000 0000`).
- Use the `HALT` trap service routine to terminate your program (`TRAP x25`, binary `1111 0000 0010 0101`).
- You may use any LC-3 instructions as needed.
- Inputs are already placed in the specified registers when your program starts — do not read from memory or the keyboard.
- Submit a plain-text listing with one 16-bit binary word per line.
- Include brief inline comments explaining each instruction's purpose. For example,

```
0001 001 010 1 00000 ; R2 -> R1.
```

3 Reference Approaches

Below are two possible approaches to implement the conditional expression. You may use either of them or devise your own solution.

3.1 Approach A — Branching Solution

Basic Idea

This approach treats the expression as an *if-else* statement. You should test `cond` via condition codes and branch accordingly:

1. Set the condition codes based on `R0`.
2. If `cond` is `0`, branch to the *else* block that copies `R2` (`val2`) to `R0`.
3. Otherwise (i.e., `cond` is `1`), fall through and copy `R1` (`val1`) to `R0`, then use an unconditional branch to skip the *else* block.
4. Finally, halt the program.

Tips

- Use the `ADD` instruction to set condition codes based on `R0`.
- Use an unconditional branch (`BRnzp`) to jump around the *else* block.
- Keep labels straight on your scratch paper even though you submit binary only.
- Review how to encode PC-relative branch targets; this is often the most error-prone part.
- Review the textbook to determine which condition code represents `cond == 0`.

3.2 Approach B — Branchless Bit-Masking Solution

Basic Idea

This approach turns `cond` into an **all-ones or all-zeros mask** using two's complement:

$$\texttt{mask} = -\texttt{cond} = \begin{cases} \texttt{0xFFFF}, & \text{if } \texttt{cond} = 1, \\ \texttt{0x0000}, & \text{if } \texttt{cond} = 0. \end{cases}$$

Then select the desired value via

```
result = (val1 & mask) | (val2 & ~mask).
```

Tips

- Use `NOT` and `ADD` to compute the two's complement.
- Since there is no bitwise OR instruction in LC-3, you may need to use De Morgan's law to convert it into operations supported by LC-3:

  $$A \vee B = \neg(\neg A \wedge \neg B).$$

  However, in this particular case, the bitwise OR operation can be replaced by another arithmetic operation. Think carefully.
- Alternatively, you can define the mask as

  $$\texttt{mask} = \texttt{?} = \begin{cases} \texttt{0x0000}, & \text{if } \texttt{cond} = 1, \\ \texttt{0xFFFF}, & \text{if } \texttt{cond} = 0. \end{cases}$$

  Note that `0xFFFF` is the 16-bit two's complement representation of `-1`.

4 Testing and Debugging

You can use LC3Tools to run and debug your program. Refer to the official Installation Guide and User Guide to learn how to use this tool.

5 Submission

Organize your submission as follows:

```
PB********_Name_lab1.zip
|-- PB********_Name_report.pdf
`-- lab1.bin.txt
```

Your report should include:

- Your name and student ID.
- A brief description of your solution.
- A few test cases you used to verify your program. Include initial values in registers `R0`, `R1`, and `R2` as well as the final value in `R0`.
- (Optional) Any challenges you encountered and how you addressed them.