**ELEN 695: Lab 2**

# Working with the LC-2 Instruction Set

Learning Objectives
This lab will familiarize you with some of the LC-2 instructions: what they do and how they are put together. There are only 16 instructions in the entire LC-2 ISA, and everything we compute must be expressed in terms of this fixed set of instructions. In this lab, we will concentrate on logical and arithmetic operations and on branches.

## Prerequisites

You should be familiar with the following concepts:

- Two's complement representation of numbers
- Sign extension of numbers
- Basic logical operations (AND, OR, NOT)
- Laws of the algebra of logical operations
- Decoding LC-2 instructions (what are it's parts?)
- Drawing flowcharts

## Prelab Assignment

### Section 1: Arithmetic and Logic Operations

The LC-2's logical and arithmetic instructions are shown below. Only three operations are provided: ADD, AND, and NOT. Starting at the most significant bits, each instruction begins with a unique four-bit opcode, followed by a destination register and one or two source operands. The first operand is always a register, and the second operand (for ADD and AND) is either a register or an "immediate" value, encoded into the instruction itself.

Recall that there are eight registers in the LC-2, used for temporary storage. Register number zero is called R0, register one is R1, and so on, through R7. In an instruction, a register is specified by its three-bit binary number (000, 001, ..., 111).

The *destination register* is where the result of the operation will be stored. The *source registers* are where the operands come from. The NOT operation only requires one operand, so it only includes one source register. Here are a few examples of instructions and their interpretations:

| *Instruction* | *Operation* | *Destination* | *Source 1* | *Source 2* |
|---|---|---|---|---|
| 0001010000000000 | ADD | R2 | R0 | R0 |
| 0101011111000010 | AND | R3 | R7 | R2 |
| 1001110110111111 | NOT | R6 | R6 | --- |

Note that both source operands can be taken from the same register, as in the ADD example. Also, the destination can be the same as one (or both) of the source registers, as in the NOT example.

For small operands, we can put the value directly into the instruction itself, rather than using a register. This is called "immediate" addressing. For ADD and AND, immediate addressing may be used for the second source operand; it is designated by a one in bit position 5. The lower five bits are used for a two's complement constant integer, which is sign-extended before adding to or anding with the other operand.

| *Instruction* | *Operation* | *Destination* | *Source 1* | *Source 2* |
|---|---|---|---|---|
| 0001010000100001 | ADD | R2 | R0 | 1 |
| 0001010000111111 | ADD | R2 | R0 | -1 |
| 0101110110101100 | AND | R6 | R6 | xC |

**Section 2: Branch Instructions**

Branch instructions allow us to make decisions - to execute a group of instructions only under certain conditions. We will first learn about condition codes and then the branch instructions that rely on those codes. The LC-2 sets one of three condition codes after every arithmetic/logical operation (and after memory load instructions), based on the results of that operation. The N flag is set if the result is negative, Z if zero, and P if positive. The branch (BR) instruction specifies one or more of the N, Z, or P condition codes to be checked. If one of the selected codes was set by the previous operation, the branch is taken, meaning that control passes to the address specified in the instruction. If none of the selected condition codes is set, the branch is not taken, meaning that the next sequential instruction is executed. To compute the branch target - the address of the next instruction in the case of a taken branch - we concatenate the high-order seven bits of the current instruction's address (also known as the program counter, or PC) with the low-order nine bits of the branch instruction. Here are some examples:

| Address | Instruction | Conditions | Offset | Target Address |
|---|---|---|---|---|
| x3000 | 0000001100100011 | P | x123 | x3123 |
| x5FFF | 0000011111000010 | Z, P | x1C2 | x5FC2 |

Make sure you understand how the target address is computed from the instruction address and the instruction offset. Note that the target address may come either before or after the branch instruction.

## Exercises (30 pts)

1. (20 pts) Do the exercises on the last sheet of this write-up, and turn the sheet into your lab instructor at the beginning of your lab period.

2. Provide two flow charts, one for each of the programming assignments below. The charts should show the step-by-step operations needed for the programs, including any loops and branches.

## Laboratory Assignments

The following two programming exercises are each worth 35% of your Lab 2 grade. Write each program in machine language, and verify that each works using the simulator.

Submit the programs electronically within one week of the lab period - precise directions on how to do this will be provided during the lab.

Copying someone else's solution is a violation of the NCSU Code of Student Conduct, and it deprives you of the chance to learn from this exercise.

**Program 1: Overflow Detection using Logical Operations (35 pts)**
Exercise 2.34 in the text asks you to detect an overflow using only logical operations. In particular, *n* and *m* are four-bit two's complement numbers, and s is their sum. You are asked to produce the binary result 1000 if an overflow has occurred, or 0000 if it has not.

The solution to this problem is to realize that only the sign bit of each operand matters, and the result has a one or zero in that same position. Since we can't perform a logical operation on the sign bit alone, we perform the operations on the entire number, ignoring the results in the upper bits. Finally, once we've produced the correct bit in the result, we use the AND operation to set all the other bits to zero.

We need to compute the following result:
O = (((not N) and (not M) and S) or (N and M and (not S))) and 0x8

For your program, N is R0, M is R1, S is R2, and O is R3. To test your program, use the "Simulate => Set Value..." command to initialize R0, R1, and R2. Then run the program and look at the contents of R3. Make sure the program works whether the overflow is caused by adding two positive or two negative numbers, and that it also works when there is no overflow at all.

Reminders:

- Your program must begin with the line "0011000000000000", which tells the simulator to load the instructions beginning at address x3000.
- Each instruction must be written on a separate line.
- End your program with the "TRAP x25" instruction: "1111000000100101".

**Program 2: Division (35 pts)**
Write a program to divide the 15-bit positive number in R0 by the 15-bit positive number in R1. After execution, the quotient should be in R4, and the remainder in R5.

Perform the operation by repeatedly subtracting R1 from R0 until R0 is less than R1. The number of times you subtract is the quotient, and the value left over (smaller than R1) is the remainder.

This program will require the use of a branch instruction. As above, test your program by setting R0 and R1.

## Prelab Exercises

Fill in the missing parts of the instructions and/or interpretations below. Turn this in to your lab instructor at the beginning of the lab period.

1. Arithmetic/Logical Operations (4pts)

| Instruction | Operation | Destination | Source 1 | Source 2 |
|---|---|---|---|---|
| ____010001000___ | ADD | R2 | _____ | R7 |
| 0101011101100011 | _____ | _____ | _____ | _____ |
| 1001___110_____ | _____ | R0 | _____ | |
| ____000000100000 | AND | _____ | _____ | _____ |
| _____ | ADD | R2 | R2 | -6 |

2. Branches (4 pts)

| Address | Instruction | Conditions | Offset | Target Address |
|---|---|---|---|---|
| x3100 | ____101010101111 | _____ | _____ | _____ |
| _____ | 0000_____ | Z, P | x7 | x4807 |
| x3210 | 0000100101111001 | _____ | _____ | _____ |
| x4503 | 0000_____ | Z | _____ | x4517 |

3. Instruction Sequences (12 pts)

For each of the following, write a short sequence of LC-2 instructions. Provide both the machine instructions and their assembly language equivalent (e.g., ADD R2, R1, x3).

(a) Subtract R1 from R0, and place the result in R2.

(b) Compute R5 OR R7, and place the result in R7.

(c) Using a single instruction put the value zero into R5.

4. Flow charts (10 pts)

_____ Approach to the Overflow Detection program (checked by TA)

_____ Approach to the Division program (checked by TA)