

# Notes: LC-3 Data Path, Breakpoints, Assembler Language

## A. Why?

The data path of the LC-3 describes its internal organization (especially that of the CPU). Breakpoints let us stop the LC-3 simulator to inspect a program's status. Assembler language is easier to read/write than machine language.

## B. Outcomes

By the end of the class you should

- Know how to use breakpoints on the LC-3 simulator.
- Recognize the different LC-3 assembler instructions

## C. Midterm Exam Review

## D. The LC-3 Data Path

- See Figure 5.18. It's complicated and you don't need to understand all its details, but it's interesting to see the different parts of the LC-3 and how they interact. Easiest to concentrate on the global bus and how components are connected to it: the MARMUX, PC & PCMUX, registers, ALU, FSM control, condition codes, IR, memory, and I/O.

## E. Chapter 6: LC-3 Breakpoints

- Study Chapter 6 at home.
- Section 1 includes program decomposition and the basic control constructs (sequencing, conditionals, loops) — you saw this in CS 1xx.
- Section 2 covers debugging. At home, read through the LC-3 manual section on using breakpoints and single-step execution (we saw this in Lecture 14). Recommend you work through the examples.
- Hint: In low-level programming, it's especially helpful to (a) figure out what properties your data has and to (b) write out the properties as comments.
  - Take Example 1 (page 165): Set  $R2 = R4 * R5$  via repeated addition.
  - The instructions have comments
 

```

          ; R2 ← 0
          ; R2 ← R2 + R4 [top of loop]
          ; R5 ← R5 - 1
          ; BRzp next-3
          
```

    - The bug is that the loop runs too many times.

- It's better to write correct code than it is to write wrong code and debug it.
- What's the relationship between R2, R4, and R5?
  - $R2 = R4 * (V - R5)$  where  $V =$  original value of R5
- We need this to be true every time control hits the loop test so that it's still true when the test fails and we finish the code.
  - Also, the loop must stop with  $R5 = 0$ .
  - The loop test must be BRp, not BRzp.
- If we write out the properties involved as initial comments, it's easy to refer to them in the line comments. "Establish R2" means to set R2 so that it meets its specified property.

```

; Multiply: R2 = R4 * R5
; Require: R5 ≥ 0
; At end: R2 = R4 * V
;   where V = original value of R5. R5 now = 0.
; Internal properties:
;   R2 = R4 * (V-R5)
;   R5 ≥ 0

AND R2, R2, 0      ; R2 ← 0; Establish R2
ADD R2, R2, R4    ; R2 ← R2 + R4
ADD R5, #-1      ; R5 ← R5 - 1; reestablish R2
BRP #-3          ; loop while R5 > 0

```

## F. Lab 8 Review

## G. Chapter 7: LC-3 Assembler Language

- Machine code ugly; assembler language less ugly.
- Each line of assembly language is translated into a single machine language instruction by an Assembler program.
- An assembler provides
  - Mnemonic labels for opcodes
  - Labels: symbolic names for address locations
  - Automatic conversion of binary / hex / decimal
  - Pseudo-operations to reserve memory, mark beginning/end of program, etc.
- Instruction format:
  - Format

```

LABEL      OPCODE      OPERANDS ; COMMENTS

```

- Opcode - Symbolic name for the 4-bit opcode
- Label (optional) - Symbolic name for a memory location; can be used as PC+offset operand in load, store, add, and, etc.
- Operands - Separated by commas; can include R0 – R7 (registers), symbolic label, x with hex digits (e.g. x03AF) or # with decimal integer (possibly negative), e.g. #-16, #0, #15. (x and # useful for immediate operands)
  - E.g. ZEROR1 AND R1, R1, #0 ; R1 <- 0
- Comment (with preceding semicolon) is optional.
- Can have a whole line be a comment by starting it with a semicolon.
- Branch instructions
  - BRN, BRZ, BRP, BRNZ, BRZP, BRNP, BRNZP, BR followed by location to go to.