LC-3 ISA
Overview & ALU Instructions

LC-3 Overview: Registers

Temporary storage

- Eight general-purpose registers: R0 - R7 (each 16 bits wide)
- Modern machines have more
  - Register access is faster than memory
- How many bits to uniquely identify a register

Others - not directly addressable, but used by (and affected by) instructions

- PC (Program Counter)
- IR (Instruction Register)
- PSR (Processor Status Word)
  - For Book Keeping (more on it later)
- CC (condition codes)
  - Codes are N, Z, P– Neg, Zero, Pos)
- MAR(Memory Address Register), MDR(Memory Data Register)

LC3 Overview: Memory Overview

Address space: 2^{16} locations

- 16-bit addresses -> 16-bit machine

Addressibility: 16 bits

- Each location can store 16-bits
- Therefore word size is 16-bit

Instructions vs. Data

- LC3 interleaves instructions and data in one 2^{16} word memory

More realistic

- Separate instructions and data memory
- Provides more memory
- We will work with modified version of the LC3 to incorporate real world features (more on it later!)

LC-3 ISA: Overview

Opcodes

- 16 opcodes ([15:12] of instruction = 2^4 = 16 possible values)
- Types of instructions:
  - *Operate* instructions: E.g. ADD
  - *Control* instructions: E.g. BR
  - *Data movement* instructions: E.g. LD
- Some opcodes set/clear CC (*condition codes*), based on result
- Adding more instructions for convenience (more on it later)
  - E.g. OR, CMP

Addressing Modes

- How is the location of an operand (data to acted upon) specified?
  - Non-memory addresses: *register*, *immediate (literal)*
  - Memory addresses: *base+offset*, *PC-relative*, *indirect*

Data Types

- 16-bit 2's complement integer and ASCII

Operate Instructions

Only three Arithmetic & Logical (ALU) operations

- **ADD, AND, NOT**

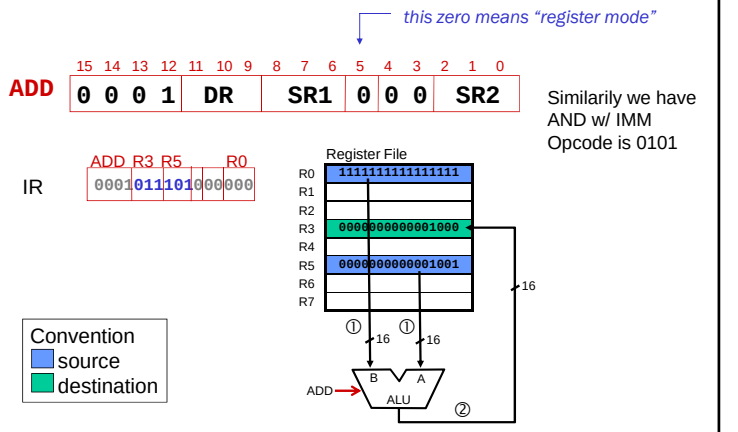
Source and destination operands are registers

- ***Do not*** **reference memory**
- **ADD and AND can use "immediate" mode, (*i.e.*, one operand is hard-wired into instruction)**

Will show abstracted datapath with each instruction

- **Illustrate *when* and *where* data moves to accomplish desired op.**

ADD (Register format)

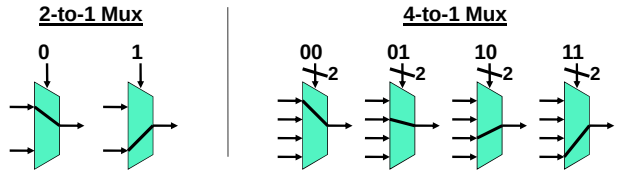


Similarily we have AND w/ IMM Opcode is 0101

Multiplexer (MUX)

Selector/Chooser of value (electrical voltages)

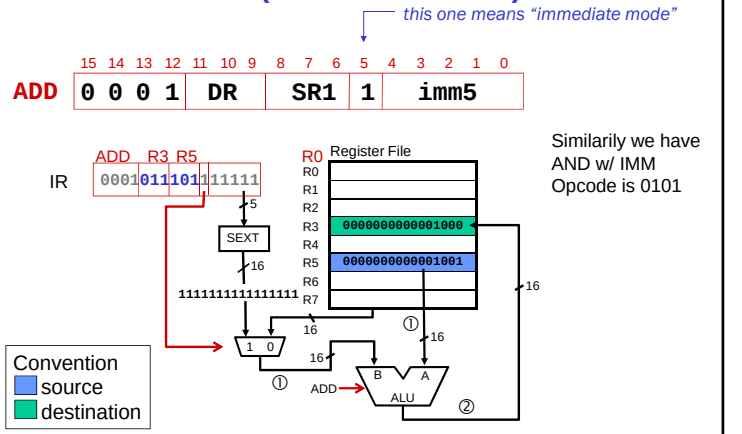
- **Multi-way switch**



In general

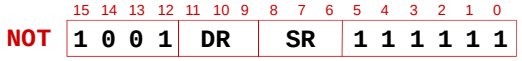
- **N select bits chooses from 2^N inputs**
- **An incredibly useful building block**

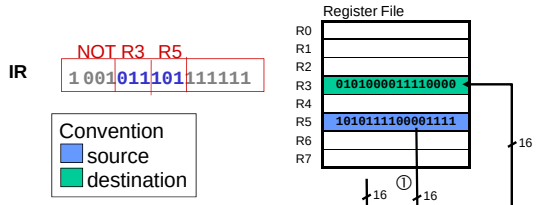
ADD (Immediate format)



Similarily we have AND w/ IMM Opcode is 0101

NOT (Register format)





Note: DR and SR could be the same register

Using Operate Instructions: Subtraction

How do we subtract two numbers?

Goal

- **R1 <- R2 - R3 (no such instruction!)**

Idea (Use 2's complement)

1. **R1 <- NOT R3**
2. **R1 <- R1 + 1**
3. **R1 <- R2 + R1**

If 2nd operand is known and small, easy

- **R1 <- R2 + -3**

LC-3 Instruction Encoding: Example 1

R1 <- R2 - R3															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R1 <- R2 - 3															

Using Operate Instructions: OR

How do we OR two values ?

Goal

- **R1 <- R2 OR R3 (No such instruction)**

Idea (Use DeMorgan's Law)

- **A OR B = NOT(NOT(A) AND NOT(B))**

1. **R4 <- NOT R2**
2. **R5 <- NOT R3**
3. **R1 <- R4 AND R5 (AND is just like ADD instruction)**
4. **R5 <- NOT R1**

Using Operate Instructions: Copying

How do we copy a number from register to register?

Goal

- **R1 <- R2 (no such instruction!)**
- **Most ISAs provide a MOV operation but we can still do without one**

Idea (Use immediate)

- **R1 <- R2 + 0**

Could we use AND?

R1 <- R2 AND ?															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0