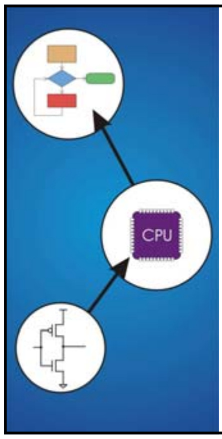
Introduction to Computer Engineering

ECE/CS 252, Fall 2010
Prof. Mikko Lipasti
Department of Electrical and Computer Engineering
University of Wisconsin – Madison



Chapter 5
The LC-3

Instruction Set Architecture

ISA = All of the ***programmer-visible*** components and operations of the computer

- memory organization
  - address space -- how may locations can be addressed?
  - addressibility -- how many bits per location?
- register set
  - how many? what size? how are they used?
- instruction set
  - opcodes
  - data types
  - addressing modes

ISA provides all information needed for someone that wants to write a program in machine language
(or translate from a high-level language to machine language).

LC-3 Overview: Memory and Registers

Memory
- address space: 2^{16} locations (16-bit addresses)
- addressability: 16 bits

Registers
- temporary storage, accessed in a single machine cycle
  - accessing memory generally takes longer than a single cycle
- eight general-purpose registers: R0 - R7
  - each 16 bits wide
  - how many bits to uniquely identify a register?
- other registers
  - not directly addressable, but used by (and affected by) instructions
  - PC (program counter), condition codes

LC-3 Overview: Instruction Set

Opcodes
- 15 opcodes
- *Operate* instructions: ADD, AND, NOT
- *Data movement* instructions: LD, LDI, LDR, LEA, ST, STR, STI
- *Control* instructions: BR, JSR/JSRR, JMP, RTI, TRAP
- some opcodes set/clear *condition codes*, based on result:
  - N = negative, Z = zero, P = positive (> 0)

Data Types
- 16-bit 2's complement integer

Addressing Modes
- How is the location of an operand specified?
- non-memory addresses: ***immediate***, ***register***
- memory addresses: ***PC-relative***, ***indirect***, ***base+offset***

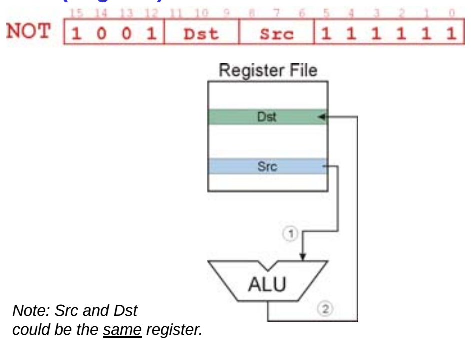
Operate Instructions

Only three operations: ADD, AND, NOT

Source and destination operands are registers
- These instructions ***do not*** reference memory.
- ADD and AND can use "immediate" mode, where one operand is hard-wired into the instruction.

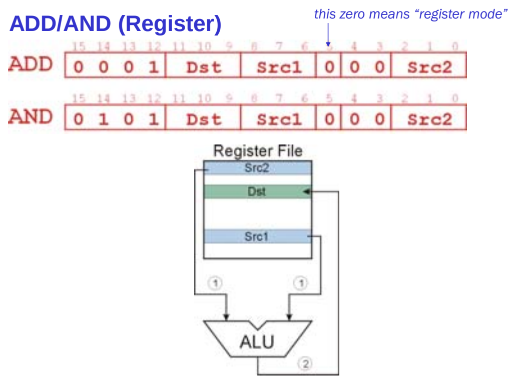
Will show dataflow diagram with each instruction.
- illustrates ***when*** and ***where*** data moves to accomplish the desired operation

NOT (Register)

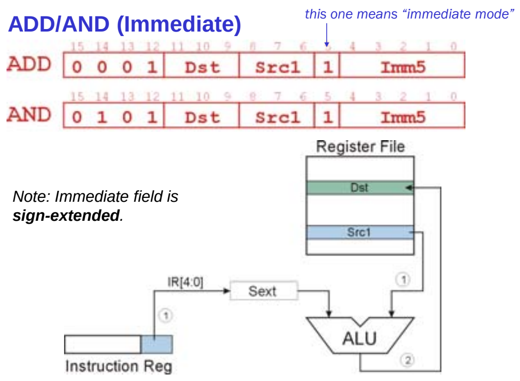


Note: Src and Dst could be the same register.

ADD/AND (Register)



ADD/AND (Immediate)



*Note: Immediate field is **sign-extended**.*

Data Movement Instructions

Load -- read data from memory to register

- **LD: PC-relative mode**
- **LDR: base+offset mode**
- **LDI: indirect mode (watch online lecture)**

Store -- write data from register to memory

- **ST: PC-relative mode**
- **STR: base+offset mode**
- **STI: indirect mode (watch online lecture)**

Load effective address -- compute address, save in register

- **LEA: immediate mode**
- ***does not access memory***

PC-Relative Addressing Mode

Want to specify address directly in the instruction

- **But an address is 16 bits, and so is an instruction!**
- **After subtracting 4 bits for opcode and 3 bits for register, we have <u>9 bits</u> available for address.**

Solution:

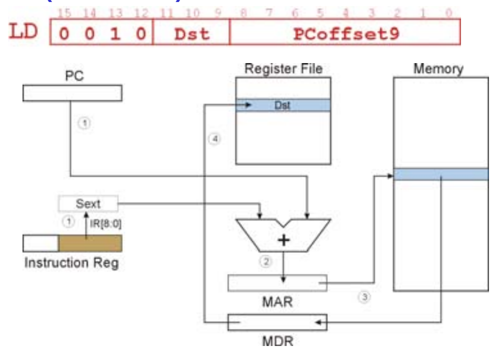
- **Use the 9 bits as a *<u>signed offset</u>* from the current PC.**

9 bits: $-256 \leq \text{offset} \leq +255$

Can form any address X, such that: $PC - 256 \leq X \leq PC + 255$

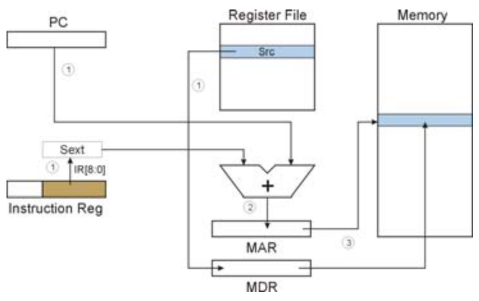
Remember that PC is incremented as part of the FETCH phase;
This is done <u>before</u> the EVALUATE ADDRESS stage.

LD (PC-Relative)



ST (PC-Relative)

	15	14	13	12	11–9	8–0
ST	0	0	1	1	Src	PCoffset9



Base + Offset Addressing Mode

With PC-relative mode, can only address data within 256 words of the instruction.

- **What about the rest of memory?**

Solution #2:

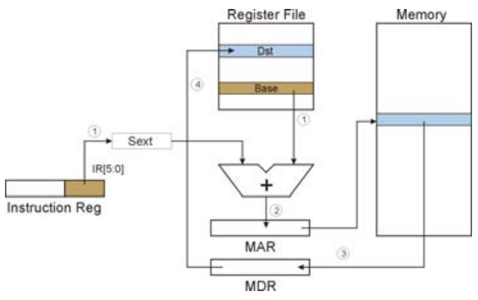
- **Use a register to generate a full 16-bit address.**

4 bits for opcode, 3 for src/dest register, 3 bits for *base* register -- remaining 6 bits are used as a *signed offset*.

- Offset is *sign-extended* before adding to base register.

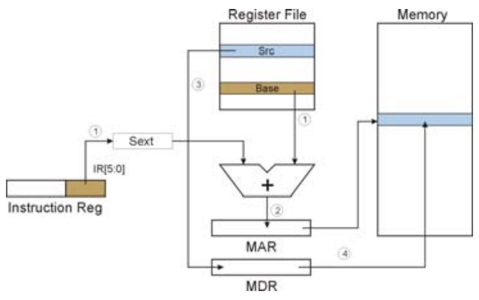
LDR (Base+Offset)

	15	14	13	12	11–9	8–6	5–0
LDR	0	1	1	0	Dst	Base	offset6



STR (Base+Offset)

	15	14	13	12	11–9	8–6	5–0
STR	0	1	1	1	Src	Base	offset6



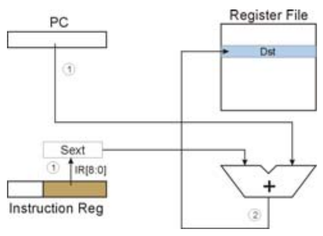
Load Effective Address

Computes address like PC-relative (PC plus signed offset) and stores the result into a register.

Note: The *address* is stored in the register, not the contents of the memory location.

LEA (Immediate)

	15	14	13	12	11–9	8–0
LEA	1	1	1	0	Dst	PCoffset9



Example

Address	Instruction	Comments
x30F6	1 1 1 0 0 0 1 1 1 1 1 1 1 1 0 1	R1 ← PC – 3 = x30F4
x30F7	0 0 0 1 0 1 0 0 0 1 1 0 1 1 1 0	R2 ← R1 + 14 = x3102
x30F8	0 0 1 1 0 1 0 1 1 1 1 1 1 0 1 1	M[PC - 5] ← R2 M[x30F4] ← x3102
x30F9	0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0	R2 ← 0
x30FA	0 0 0 1 0 1 0 0 1 0 1 0 0 1 0 1	R2 ← R2 + 5 = 5
x30FB	0 1 1 1 0 1 0 0 0 1 0 0 1 1 1 0	M[R1+14] ← R2 M[x3102] ← 5

opcode

Control Instructions

Used to alter the sequence of instructions
(by changing the Program Counter)

Conditional Branch

- **branch is *taken* if a specified condition is true**
  - **signed offset is added to PC to yield new PC**
- **else, the branch is *not taken***
  - **PC is not changed, points to the next sequential instruction**

Unconditional Branch (or Jump)

- **always changes the PC**

TRAP

- **changes PC to the address of an OS “service routine”**
- **routine will return control to the next instruction (after TRAP)**

Condition Codes

LC-3 has three condition code registers:

N -- negative
Z -- zero
P -- positive (greater than zero)

Set by any instruction that writes a value to a register
(ADD, AND, NOT, LD, LDR, LDI, LEA)

Exactly <u>one</u> will be set at all times

- **Based on the last instruction that altered a register**

Branch Instruction

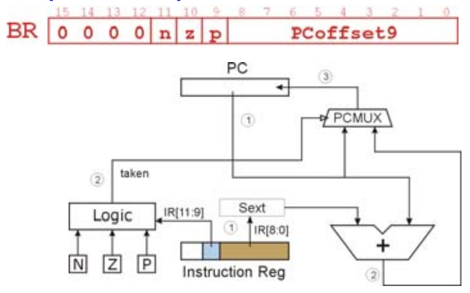
Branch specifies one or more condition codes.

If the specified bit is set, the branch is taken.

- **PC-relative addressing:**
  target address is made by adding signed offset (IR[8:0])
  to current PC.
- **Note: PC has already been incremented by FETCH stage.**
- **Note: Target must be within 256 words of BR instruction.**

If the branch is not taken,
the next sequential instruction is executed.

BR (PC-Relative)

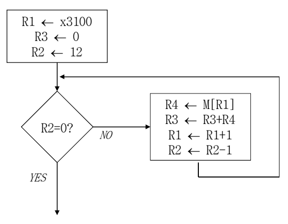


What happens if bits [11:9] are all zero? All one?

Using Branch Instructions

Compute sum of 12 integers.

Numbers start at location x3100. Program starts at location x3000.



Sample Program

Address	Instruction	Comments
x3000	1 1 1 0 0 0 1 0 1 1 1 1 1 1 1 1	*R1 ← x3100 (PC+0xFF)*
x3001	0 1 0 1 0 1 1 0 1 1 1 0 0 0 0 0	*R3 ← 0*
x3002	0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0	*R2 ← 0*
x3003	0 0 0 1 0 1 0 0 1 0 1 0 1 1 0 0	*R2 ← 12*
x3004	0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1	*If Z, goto x300A (PC+5)*
x3005	0 1 1 0 1 0 0 0 0 1 0 0 0 0 0 0	*Load next value to R4*
x3006	0 0 0 1 0 1 1 0 1 1 0 0 0 1 0 0	*Add R4 to R3*
x3007	0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 1	*Increment R1 (pointer)*
X3008	0 0 0 1 0 1 0 0 1 0 1 1 1 1 1 1	*Decrement R2 (counter)*
x3009	0 0 0 0 1 1 1 1 1 1 1 1 1 0 1 0	*Goto x3004 (PC-6)*

Summary

LC-3 Memory and Registers

Operate instructions: ADD/AND/NOT

LC-3 Addressing modes: PC-relative, Base+Offset

Data Movement Instructions: LD/ST/LDR/STR/LEA

Control Instructions: Branch

Online: additional instructions, examples, LC-3 datapath