9. LC-3 Instructions, Examples (Chapter 5)

October 1, 2018

- Review
  - Instruction processing
  - Example instructions
- LC-3 instructions
  - Sequence of operations
  - Data path and control sequences
- Example program(s)

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

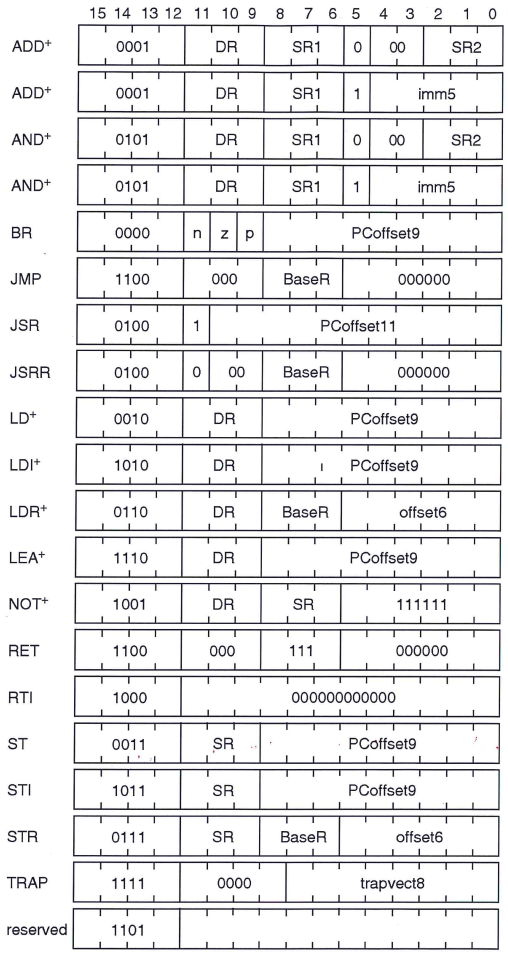
- 15 opcodes
- *Operate* instructions: ADD, AND, NOT
- *Data movement* instructions: LD, LDI, LDR, LEA, ST, STR, STI
- *Control* instructions: BR, JSR/JSRR, JMP, RTI, TRAP
- some opcodes set/clear *condition codes*, based on result:
  - ➢ N = negative, Z = zero, P = positive (> 0) — SET ON REGISTER OPERATIONS

Data Types

- 16-bit 2's complement integer

Addressing Modes

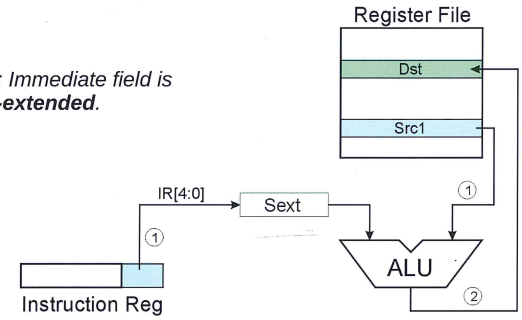
- How is the location of an operand specified?
- non-memory addresses: *immediate*, *register*
- memory addresses: *PC-relative*, *indirect*, *base+offset*



ADD/AND (Immediate)

this one means "immediate mode"

*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

Store -- write data from register to memory

- **ST:** PC-relative mode
- **STR:** base+offset mode
- **STI:** indirect mode

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 9 bits available for address.**

Solution:

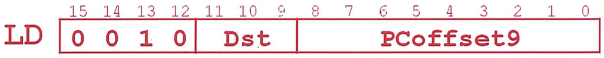
- **Use the 9 bits as a *signed offset* from the current PC.**

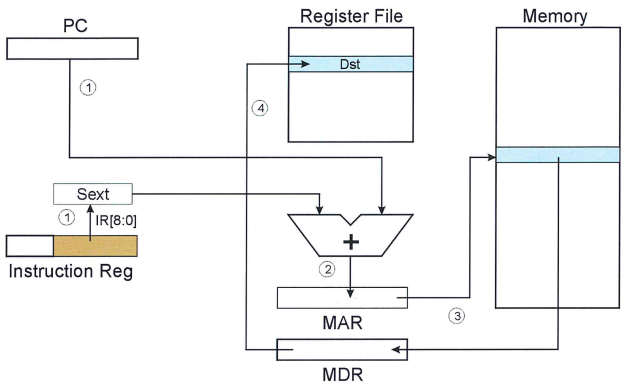
9 bits: $-256 \le \text{offset} \le +255$

Can form any address X, such that: $\text{PC}-256 \le X \le \text{PC}+255$

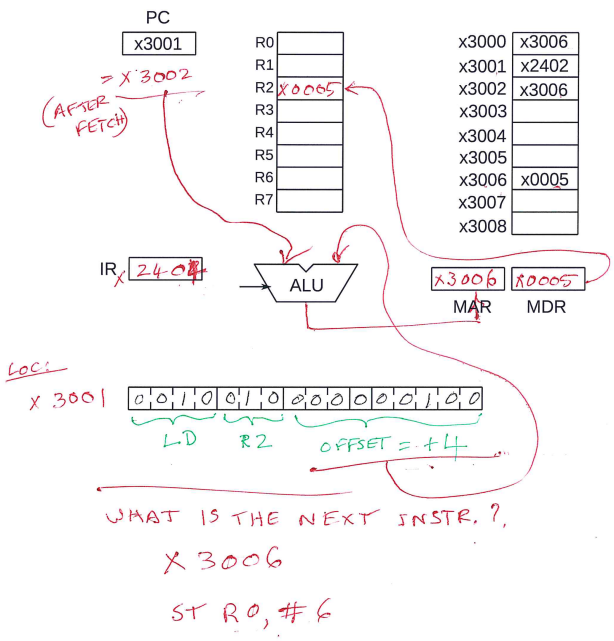
Remember that PC is incremented as part of the FETCH phase; This is done before the EVALUATE ADDRESS stage.

LD (PC-Relative)

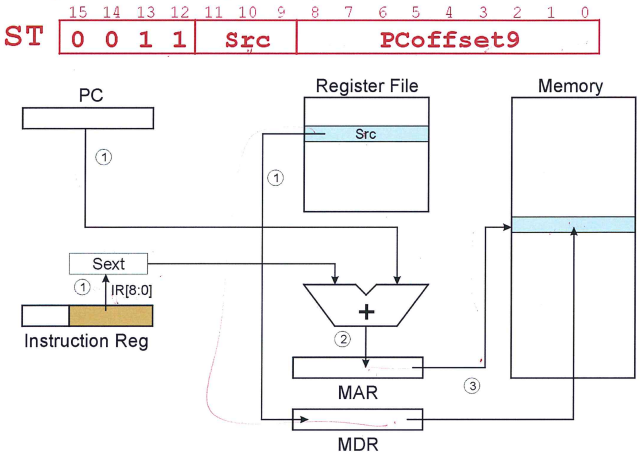




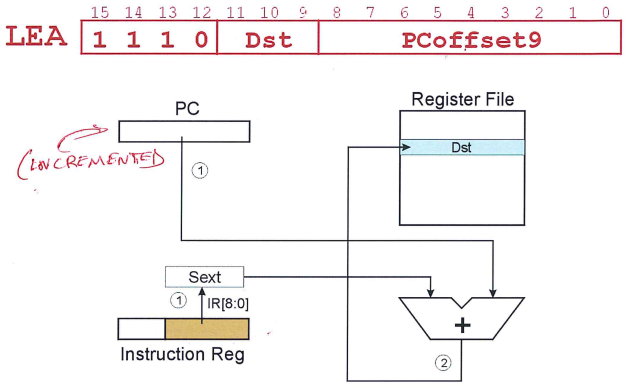
Example: LD (PC-relative addressing)



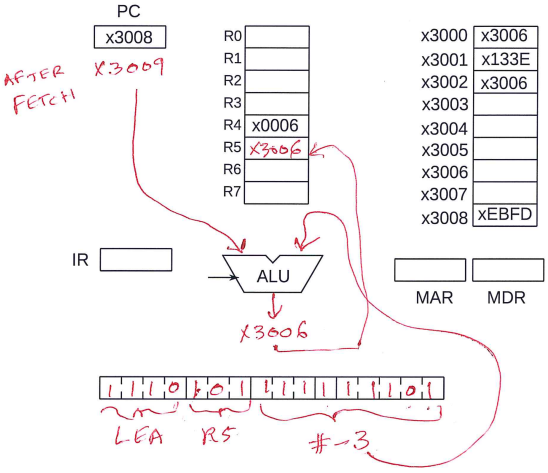
ST (PC-Relative)



LEA (Immediate)



Example: LEA (Immediate Mode)



Indirect Addressing Mode

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

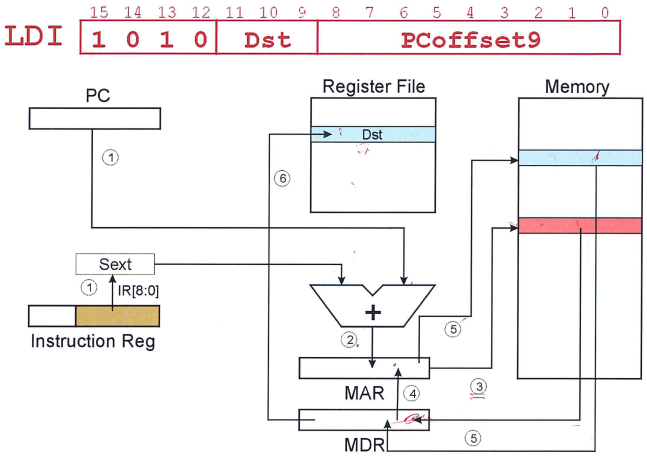
- **What about the rest of memory?**

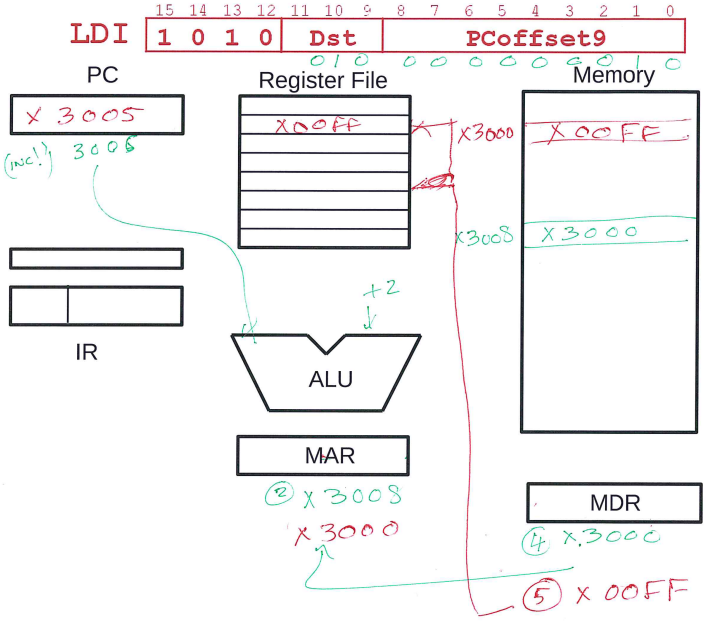
Solution #1:

- **Read address from memory location, then load/store to that address.**

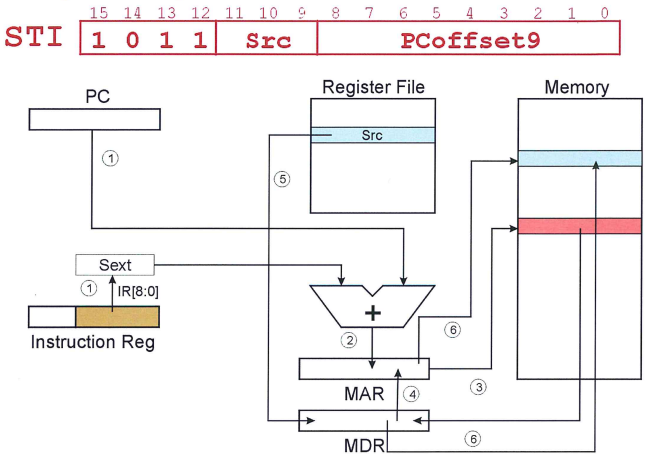
First address is generated from PC and IR (just like PC-relative addressing), then content of that address is used as target for load/store.

LDI (Indirect)





STI (Indirect)



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:

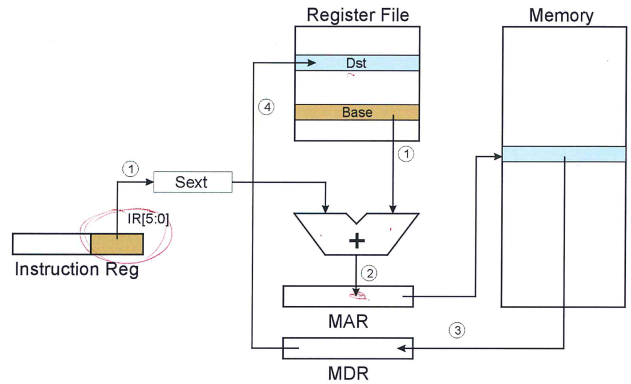
- **Use a register to generate a full 16-bit address.**

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	10	9	8	7	6	5	4	3	2	1	0
LDR	0	1	1	0	Dst			Base			offset6					



Example

Address	Instruction	Comments
LEA x30F6	1 1 1 0 0 0 1 1 1 1 1 1 1 1 0 1	R1 ← PC − 3 = x30F4
ADD x30F7	0 0 0 1 0 1 0 0 0 1 1 0 1 1 1 0	R2 ← R1 + 14 = x3102
ST x30F8	0 0 1 1 0 1 0 1 1 1 1 1 1 0 1 1	M[PC−5] ← R2; M[30F4] ← x3102
AND x30F9	0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0	R2 ← 0
ADD x30FA	0 0 0 1 0 1 0 0 1 0 1 0 0 1 0 1	R2 ← R2 + 5 = 5
STR x30FB	0 1 1 1 0 1 0 0 0 1 0 0 1 1 1 0	M[R1+14] ← R2; M[x3102] ← 5
LDI x30FC	1 0 1 0 0 1 1 1 1 1 1 1 0 1 1 1	R3 ← M[M[x30F4]]; R3 ← M[x3102]; R3 ← 5

opcode

= 1001
= −9
x30FD − 9 = x30F4

Example: LC-3 JMP Instruction

Set the PC to the value contained in a register. This becomes the address of the next instruction to fetch.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JMP				0	0	0	Base			0	0	0	0	0	0

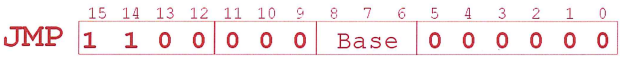
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0

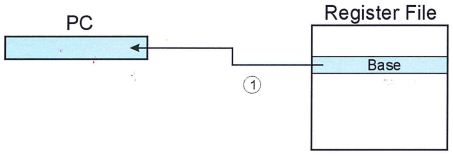
"Load the contents of R3 into the PC."

JMP (Register)

Jump is an unconditional branch -- *always* taken.

- **Target address is the contents of a register.**
- **Allows any target address.**





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 one will be set at all times

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

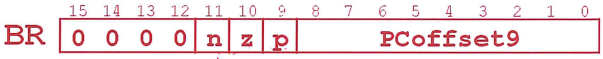
Branch Instruction

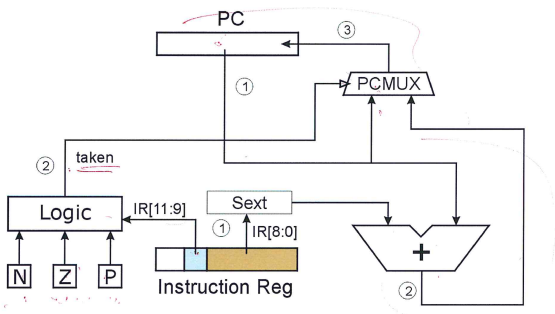
Branch specifies one or more condition codes.
If the set bit is specified, the branch is taken.

- **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?

BR n -9 0000 1001 1111 1011