University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 120: Introduction to Computing

The LC-3 Instruction Set Architecture

The LC-3 ISA Has Three Kinds of Opcodes

The **LC-3 ISA** has **three kinds of opcodes**:

1. **operations**
   (with the ALU)
2. **data movement**
   (registers to/from memory)
3. **control flow**
   (conditionally change the PC)

Let's look at each kind in turn.

The LC-3 Supports Three Operate Instructions

The LC-3 ALU is capable of **three operations**.

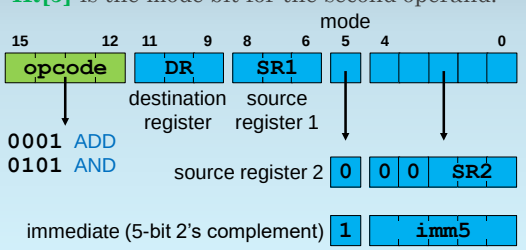
The ISA includes **one opcode for each** operation: **ADD**, **AND**, and **NOT**.

Each operation uses
- **one source register** and
- **one destination register**.

The second operand (for **ADD** and **AND** only) allows a choice of **addressing modes**:
- **register** (another register) or
- **immediate** (a number stored in the instruction).

ADD and AND Have Two Addressing Modes

IR[5] is the mode bit for the second operand.



What Can We Do with Immediate Mode?

Add small numbers
- +1
- -1
- (anything from -16 to +15).

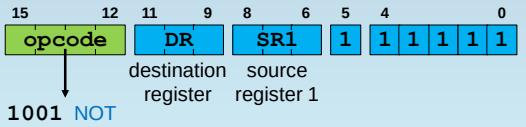
Mask out high bits
- AND 1
- AND 3

Mask out low bits
- AND -2 (xFFFE)
- AND -4 (xFFFC)

Put 0 in a register! (AND 0)

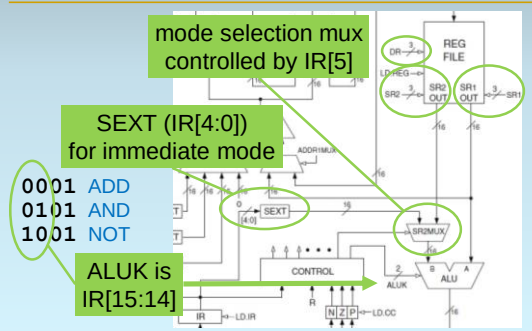
Second Operand Bits are All 1 for the NOT Instruction

For **NOT**, **IR[5:0] must be 1**.



(NOT only has one input operand.)

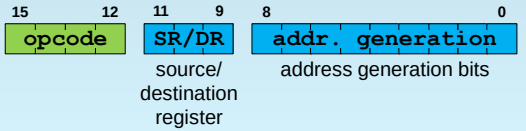
Execution of Operate Instructions in the Datapath



Loads and Stores Have Four Addressing Modes

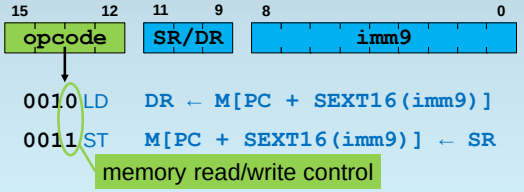
Data movement instructions are of two types:
- **loads** (from memory to register)
- **stores** (from register to memory)

The LC-3 ISA supports **four addressing modes** for data movement instructions.



PC-Relative Addressing Adds an Offset to the PC

The first addressing mode is **PC-relative**.



Important: **PC** is the **value after FETCH**
(the address of the LD/ST + 1).

Example of PC-Relative Addressing

What does this instruction (at memory address x1480) do?

`x1480 LD R3,x09`

In **RTL**: `R3 ← M[PC + SEXT16(x09)]`

What is PC?

Again, since execution occurs after **FETCH**, **PC in this case is x1481**. So…

`R3 ← M[x148A]` **(NOT x1489!)**

Time for a Quiz!

What's stored at a memory address?

Bits. (16 of them, yes.)

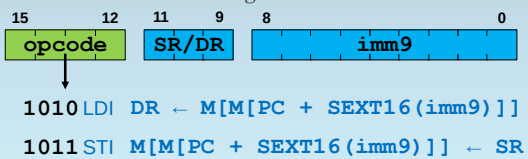
How do we name a memory address?

Bits. (Again 16 of them.)

So we can
use the bits at a memory address to name another memory address!

Indirect Addressing Accesses Memory Twice

The second addressing mode is **indirect**.



Indirect mode reads the address for the load/store from a PC-relative address.

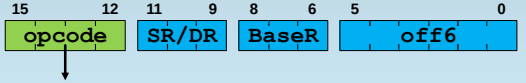
What Purpose Does Indirect Mode Serve?

Primarily to make you realize that
- the bits at a memory address
- can **point to** a memory address.

The concept of a **pointer**
- (just another word for a memory address)
- is **critical for understanding more complex representations** in many programming languages (such as C).

Base + Offset Mode Uses Another Register

The third addressing mode is **base + offset**.




```
0110 LDR DR ← M[BaseR + SEXT16(off6)]
0111 STR M[BaseR + SEXT16(off6)] ← SR
```

Base + offset mode **uses another register** to generate the address for the memory access.

Base + Offset Mode Enables Wider Memory Access

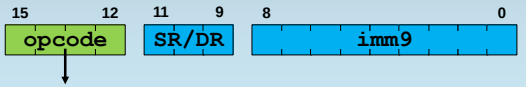
PC-relative and **indirect** addressing
- can only generate
- addresses within a 9-bit offset
- of the instruction address (-256 to +255)
- (Indirect addressing can access any location, but the address has to be stored near the **LDI**/**STI**.)

Base+Offset enables access to any address by using another register.

But how do we get an address into a register?

Immediate Addressing Loads a Nearby Address

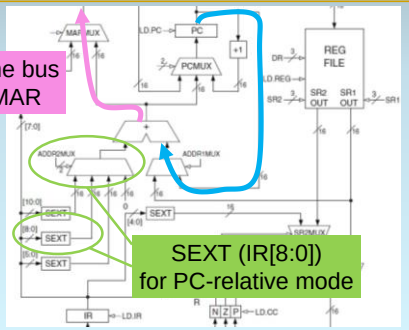
The fourth addressing mode is **immediate**.



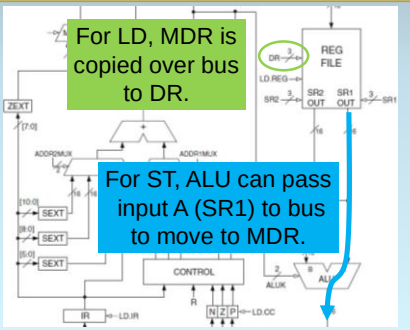

```
1110 LEA    DR ← PC + SEXT16(imm9)
```

LEA stands for "load effective address":
- **address generation is PC-relative**, but
- **memory is not accessed** (so not really data movement).

Datapath Generation of PC-Relative Addressing



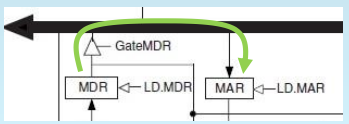
Handling the Data for Stores and Loads



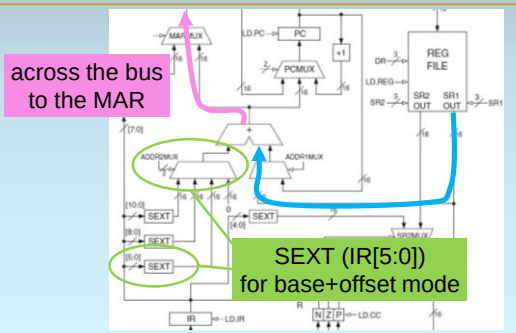
LDI and STI Move MDR into MAR

For **LDI**/**STI**,

- the **PC-relative** address read from memory into the **MDR**
- can be copied across the bus into the **MAR**,
- then used as the second access' address.



Datapath Generation of Base+Offset Addressing



Control Flow Instructions Conditionally Change PC

After executing an instruction at address **A**, the **LC-3** next executes the instruction at address **A + 1**, then **A + 2**, and so forth.

So far, we have seen **operations** and **data movement** instructions.

But how can we do things like `if` statements and loops?

We need another kind of instruction to manage **control flow**.

Control flow instructions **conditionally change the PC**.

LC-3 ISA Provides Three Condition Codes: N, Z, and P

The **LC-3** maintains **three 1-bit registers** called **condition codes**.

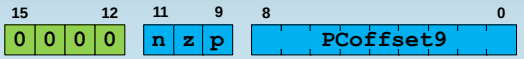
These are based on the **last value written to the register file** (by operations or by loads).

- **N**: the last value was **negative**
- **Z**: the last value was **zero**
- **P**: the last value was **positive**

Obviously, **exactly one of these three bits is 1** in any cycle.

Conditional Branch BR* Conditionally Changes PC

Let's start with conditional branch, BR.

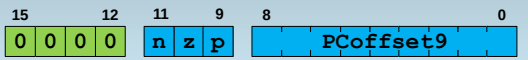


`BEN: PC ← PC + SEXT16(PCoffset9)`

The calculation of **BEN**, the **branch enable condition**, is specified in the opcode's name.

For example, **BRnp** has the **n** and **p** bits set in the instruction, while the **z** bit is zero.

Calculation of the Branch Enable Condition



`BEN: PC ← PC + SEXT16(PCoffset9)`

The **BEN** condition is calculated

- in the **DECODE** state
- based on instruction bits **n**, **z**, **p** and
- condition codes **N**, **Z**, and **P**:

`BEN ← nN + zZ + pP`

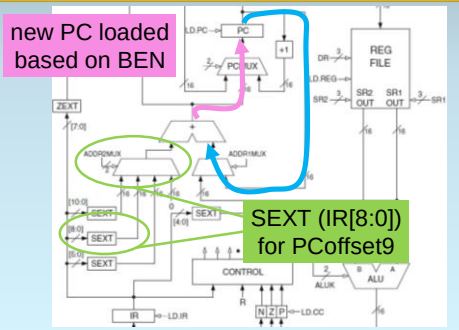
Examples of Branch Condition Names

Let's consider a few examples…

BRnz **branch if not positive**
BRnzp **branch always**
BRnp **branch if not zero**

Note: by convention, **BR** means **BRnzp** (unconditional branch), not "do nothing."

Datapath Generation of Branch Targets (New PC Values)



JMP Instruction Copies Any Register to the PC

Branch target addresses are limited.

The **BR** instruction only has a **9-bit offset**:
- **PC** is the **BR address + 1**, and
- add another -256 to +255.

What if we want to change **PC** to something farther from the current instruction?

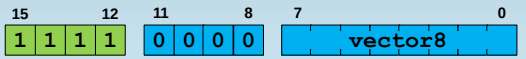
Use a JMP (jump) instruction!* The **RTL** is

`PC ← BaseR`

*Look up the encoding if you need it.

Use TRAP to Invoke Operating System Services

One more instruction: TRAP.



The **TRAP** instruction **invokes operating system services**.*

The specific service depends on **vector8**.

*For a detailed explanation of the mechanism, read Ch. 8 & 9 of Patt and Patel, or wait for ECE220.

TRAP Functions Available in the LC-3 OS

TRAP vec #	mnemonic	effect
x20	GETC	read one ASCII character from keyboard into R0
x21	OUT	write one ASCII character from R0 to display
x25	HALT	end program (return control to the "operating system")

For more detail, see p. 543 of Patt and Patel.

Do NOT Use R7 in Your ECE120 LC-3 Programs

Obviously,
- if you invoke the **GETC** trap,
- whatever bits were in **R0** are lost.

Not so obviously,
- any **TRAP** will change **R7**.
- **Do NOT use R7 for our class**.

(Again, see Ch. 8 and 9 of Patt and Patel if you want to know why before ECE220.)

