

## 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 ( $> 0$ ) — <sup>SET ON</sup> REGISTER OPERATIONS

## Data Types

- 16-bit 2's complement integer

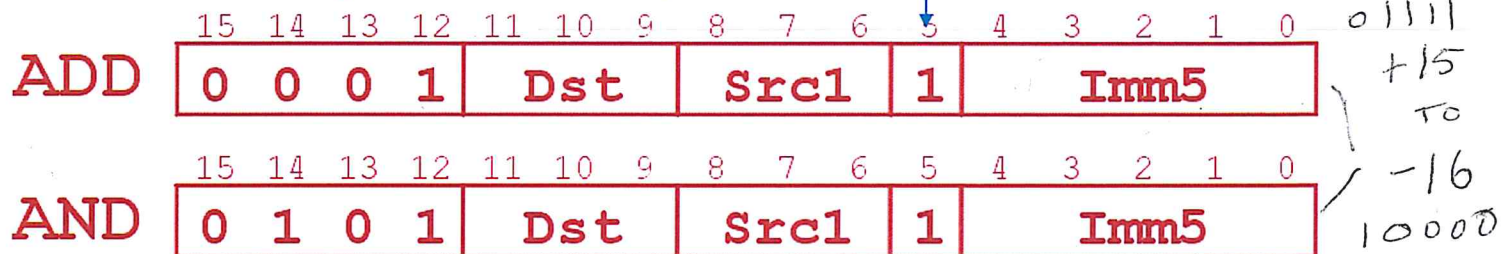
## Addressing Modes

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

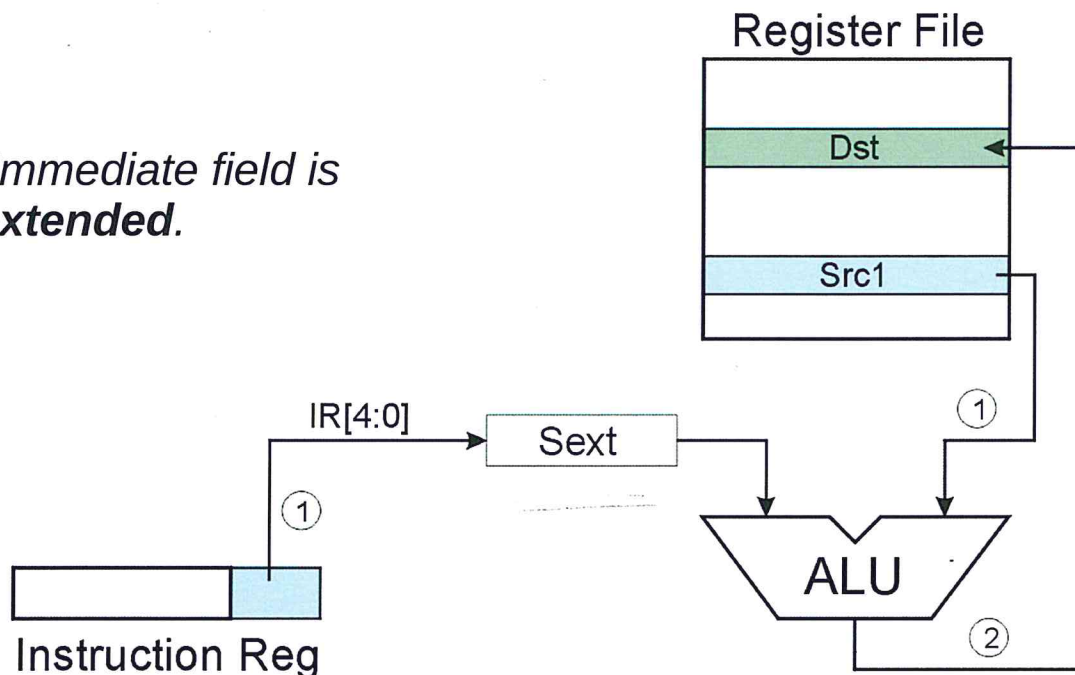
|                  | 15   | 14 | 13 | 12 | 11           | 10         | 9 | 8         | 7 | 6 | 5       | 4    | 3 | 2   | 1 | 0 |
|------------------|------|----|----|----|--------------|------------|---|-----------|---|---|---------|------|---|-----|---|---|
| ADD <sup>+</sup> | 0001 |    |    |    | DR           |            |   | SR1       |   |   | 0       | 00   |   | SR2 |   |   |
| ADD <sup>+</sup> | 0001 |    |    |    | DR           |            |   | SR1       |   |   | 1       | imm5 |   |     |   |   |
| AND <sup>+</sup> | 0101 |    |    |    | DR           |            |   | SR1       |   |   | 0       | 00   |   | SR2 |   |   |
| AND <sup>+</sup> | 0101 |    |    |    | DR           |            |   | SR1       |   |   | 1       | imm5 |   |     |   |   |
| BR               | 0000 |    |    |    | n            | z          | p | PCoffset9 |   |   |         |      |   |     |   |   |
| JMP              | 1100 |    |    |    | 000          |            |   | BaseR     |   |   | 000000  |      |   |     |   |   |
| JSR              | 0100 |    |    |    | 1            | PCoffset11 |   |           |   |   |         |      |   |     |   |   |
| JSRR             | 0100 |    |    |    | 0            | 00         |   | BaseR     |   |   | 000000  |      |   |     |   |   |
| LD <sup>+</sup>  | 0010 |    |    |    | DR           |            |   | PCoffset9 |   |   |         |      |   |     |   |   |
| LDI <sup>+</sup> | 1010 |    |    |    | DR           |            |   | PCoffset9 |   |   |         |      |   |     |   |   |
| LDR <sup>+</sup> | 0110 |    |    |    | DR           |            |   | BaseR     |   |   | offset6 |      |   |     |   |   |
| LEA <sup>+</sup> | 1110 |    |    |    | DR           |            |   | PCoffset9 |   |   |         |      |   |     |   |   |
| NOT <sup>+</sup> | 1001 |    |    |    | DR           |            |   | SR        |   |   | 111111  |      |   |     |   |   |
| RET              | 1100 |    |    |    | 000          |            |   | 111       |   |   | 000000  |      |   |     |   |   |
| RTI              | 1000 |    |    |    | 000000000000 |            |   |           |   |   |         |      |   |     |   |   |
| ST               | 0011 |    |    |    | SR           |            |   | PCoffset9 |   |   |         |      |   |     |   |   |
| STI              | 1011 |    |    |    | SR           |            |   | PCoffset9 |   |   |         |      |   |     |   |   |
| STR              | 0111 |    |    |    | SR           |            |   | BaseR     |   |   | offset6 |      |   |     |   |   |
| TRAP             | 1111 |    |    |    | 0000         |            |   | trapvect8 |   |   |         |      |   |     |   |   |
| reserved         | 1101 |    |    |    |              |            |   |           |   |   |         |      |   |     |   |   |

# 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.

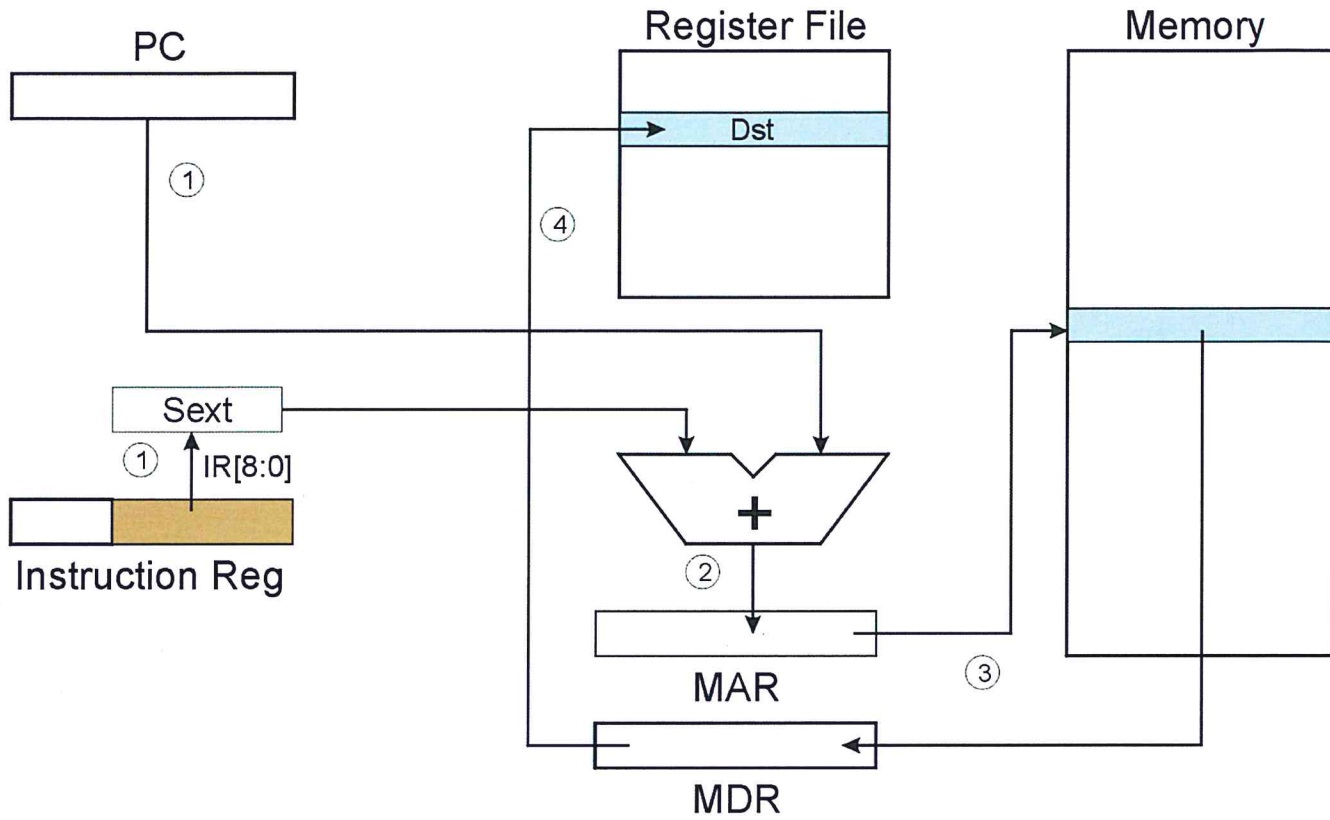
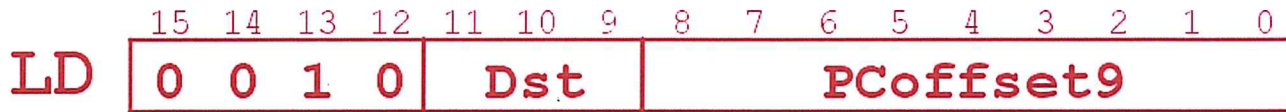
9 bits:  $-256 \leq \text{offset} \leq +255$

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

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

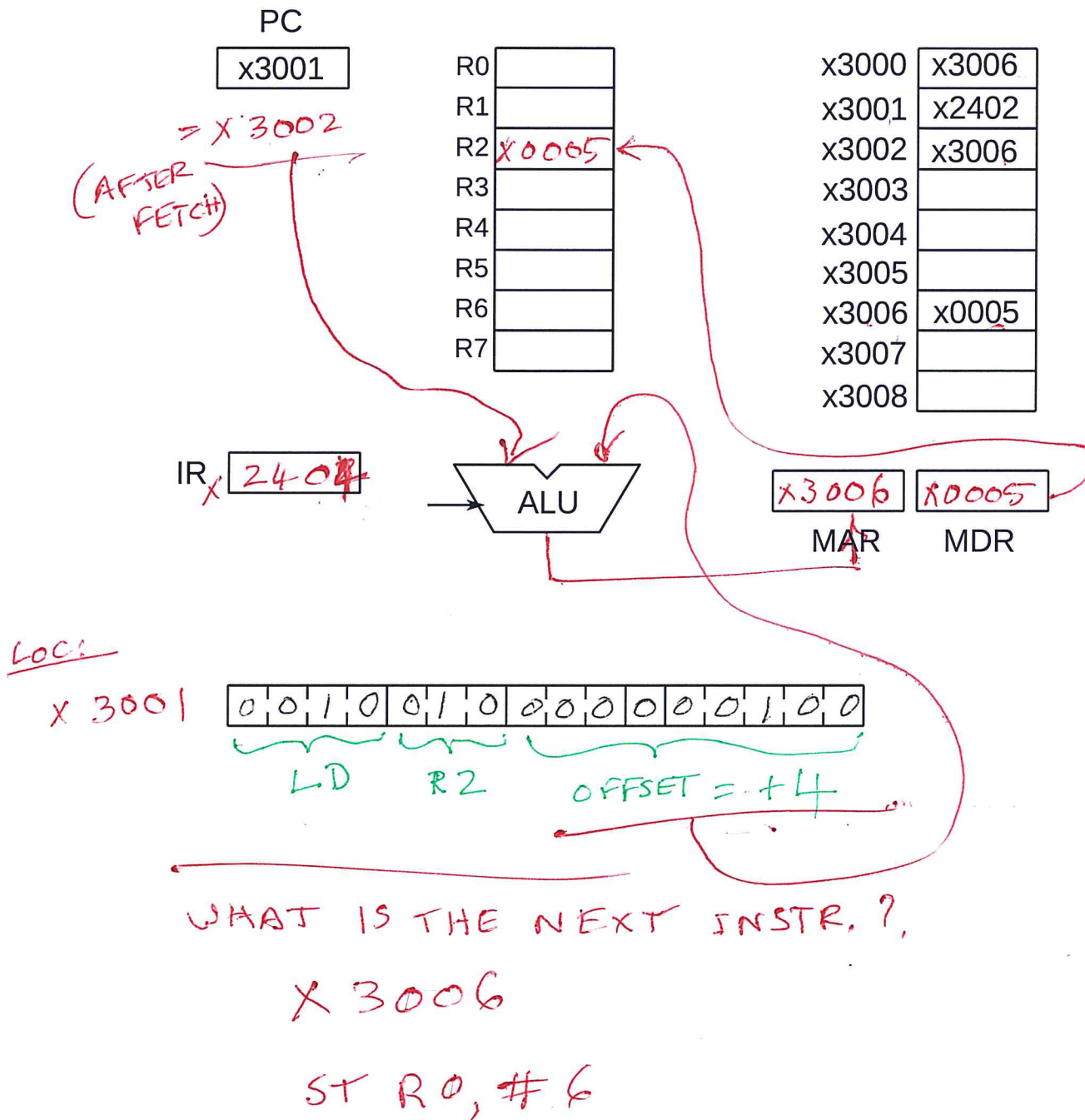


# LD (PC-Relative)

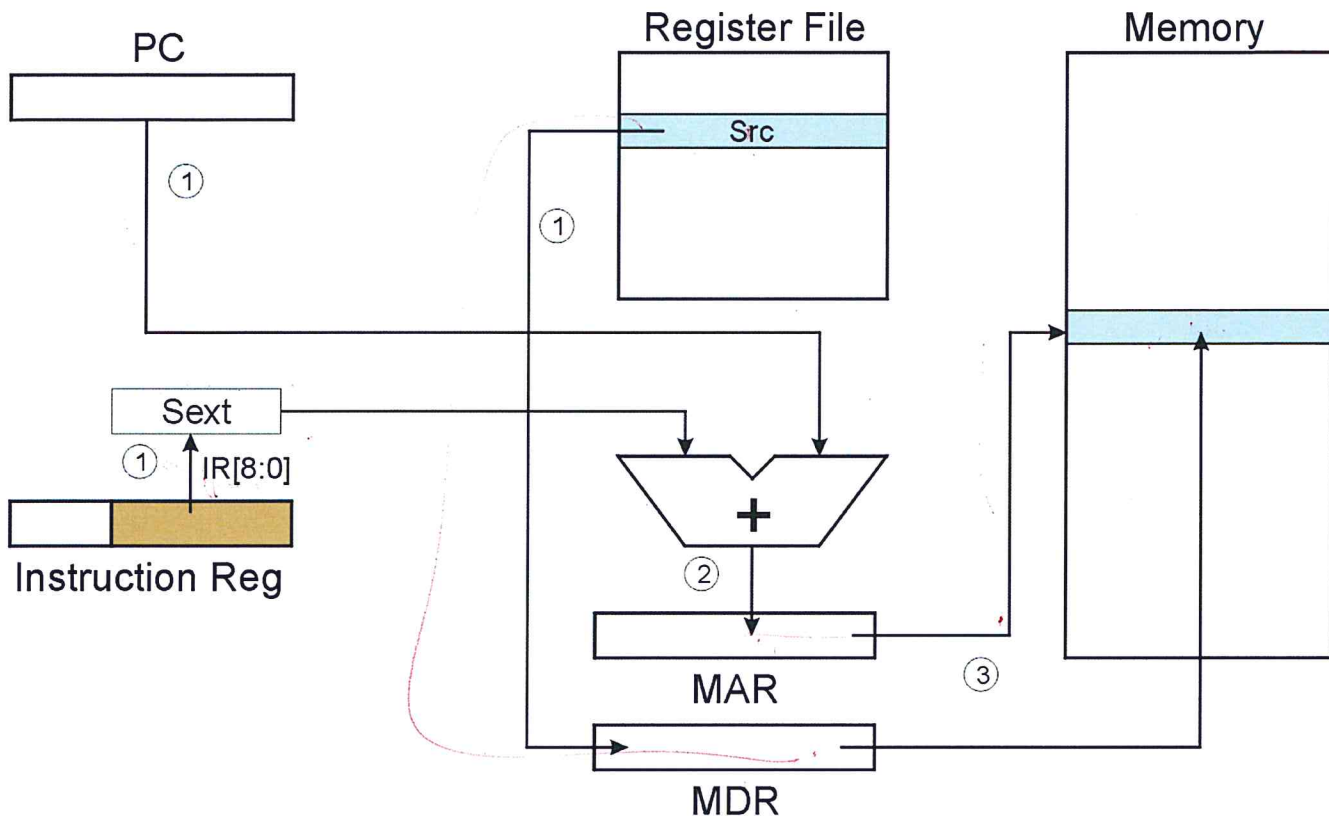
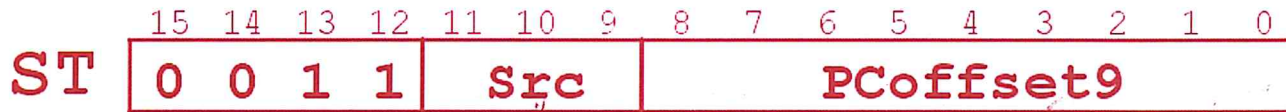




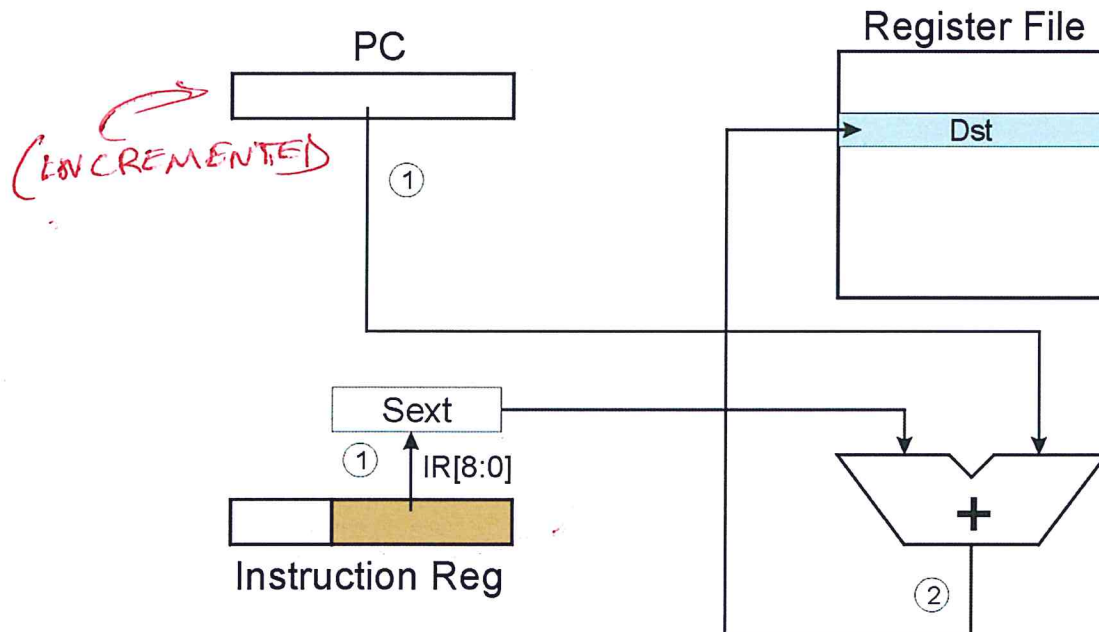
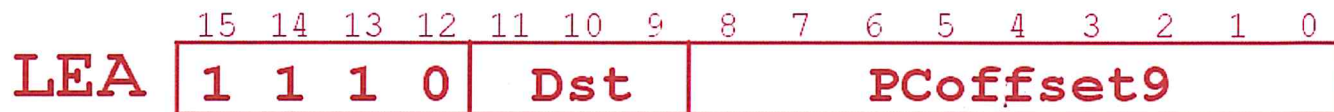
# Example: LD (PC-relative addressing)



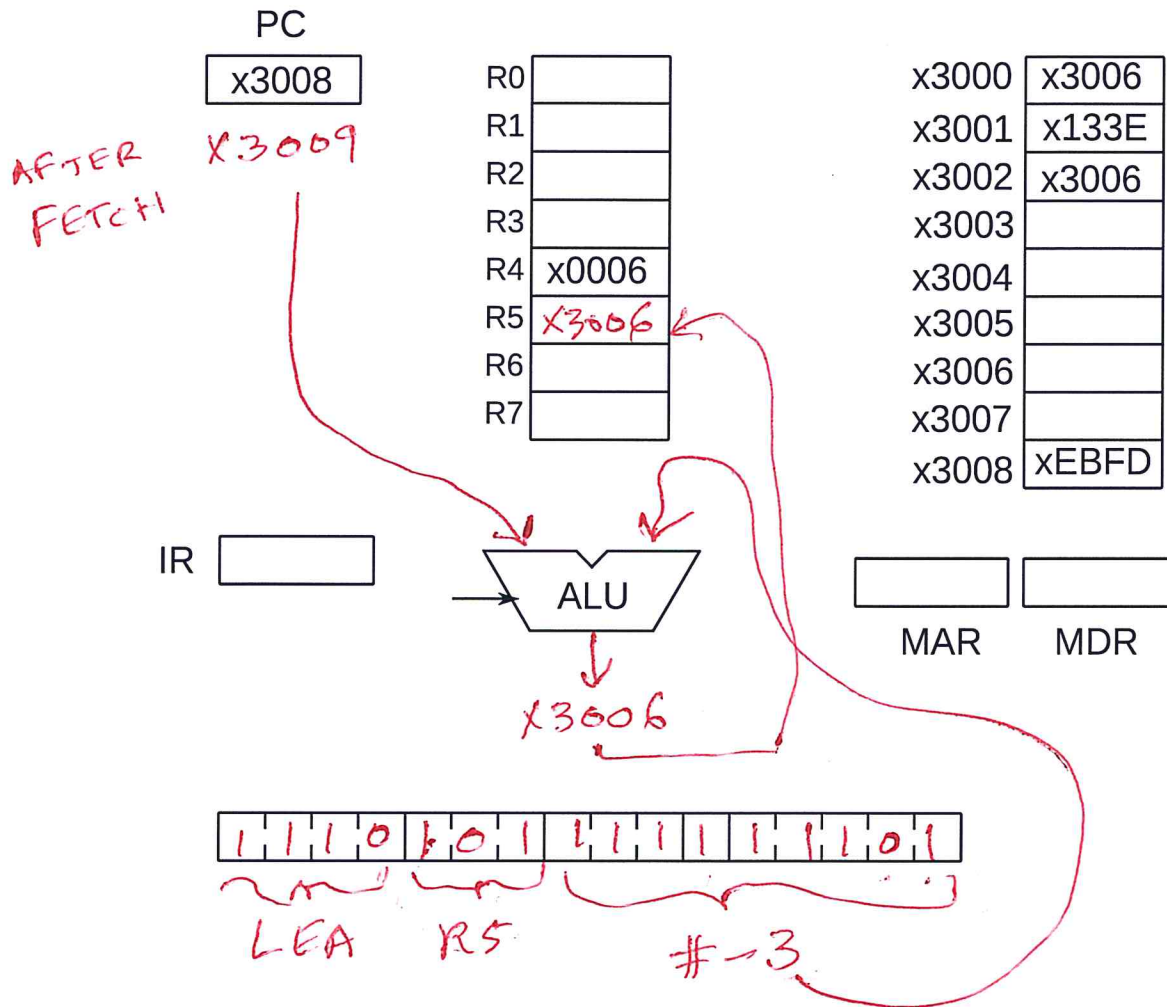
# ST (PC-Relative)



# LEA (Immediate)



## Example: LEA (Immediate Mode)



## Indirect Addressing Mode

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

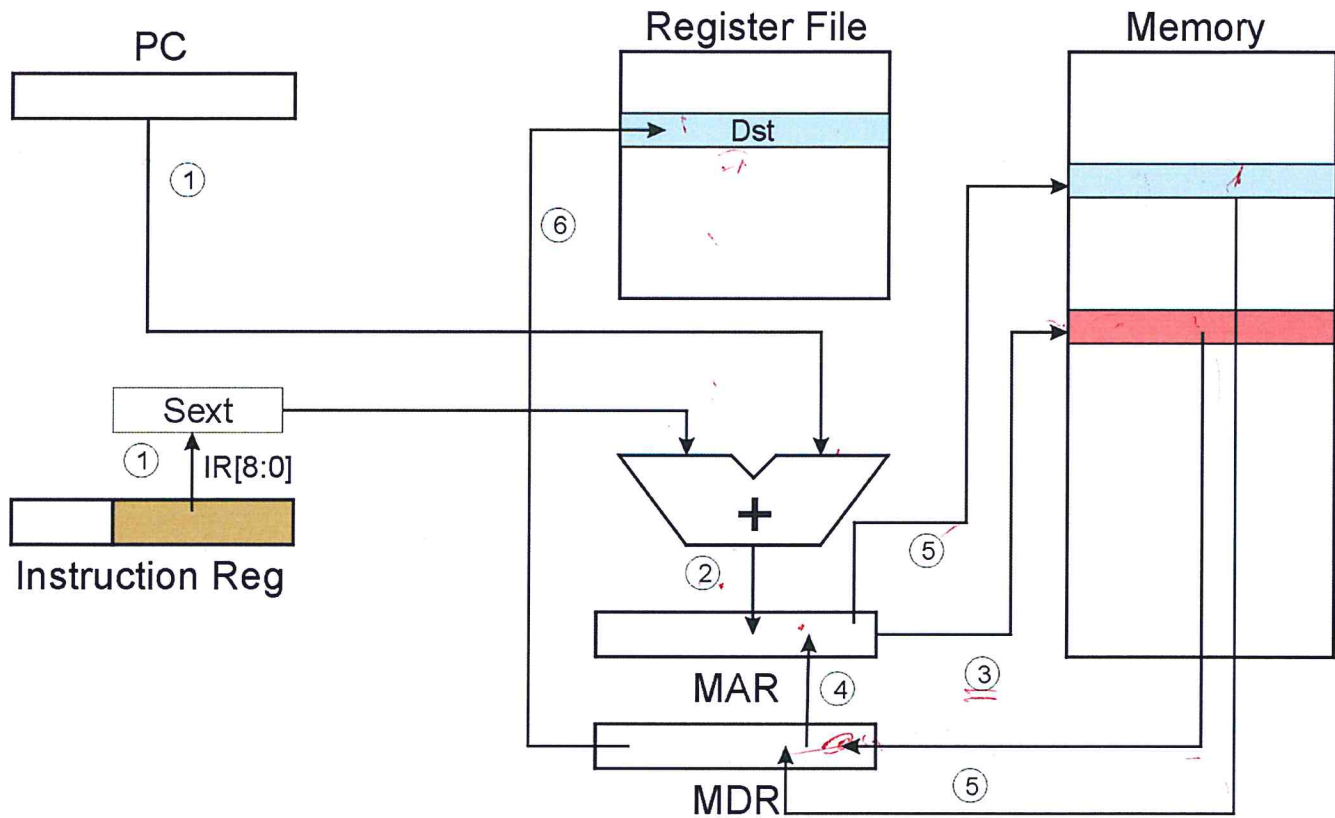
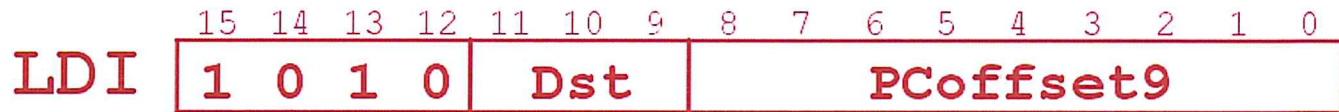
- What about the rest of memory?

### **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)





LDI 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0  
1 0 1 0 Dst PCoffset9

PC

X 3005

(inc!) 3006

IR

Register File

X00FF

+2

ALU

MAR

② X 3008

X 3000

Memory

X00FF

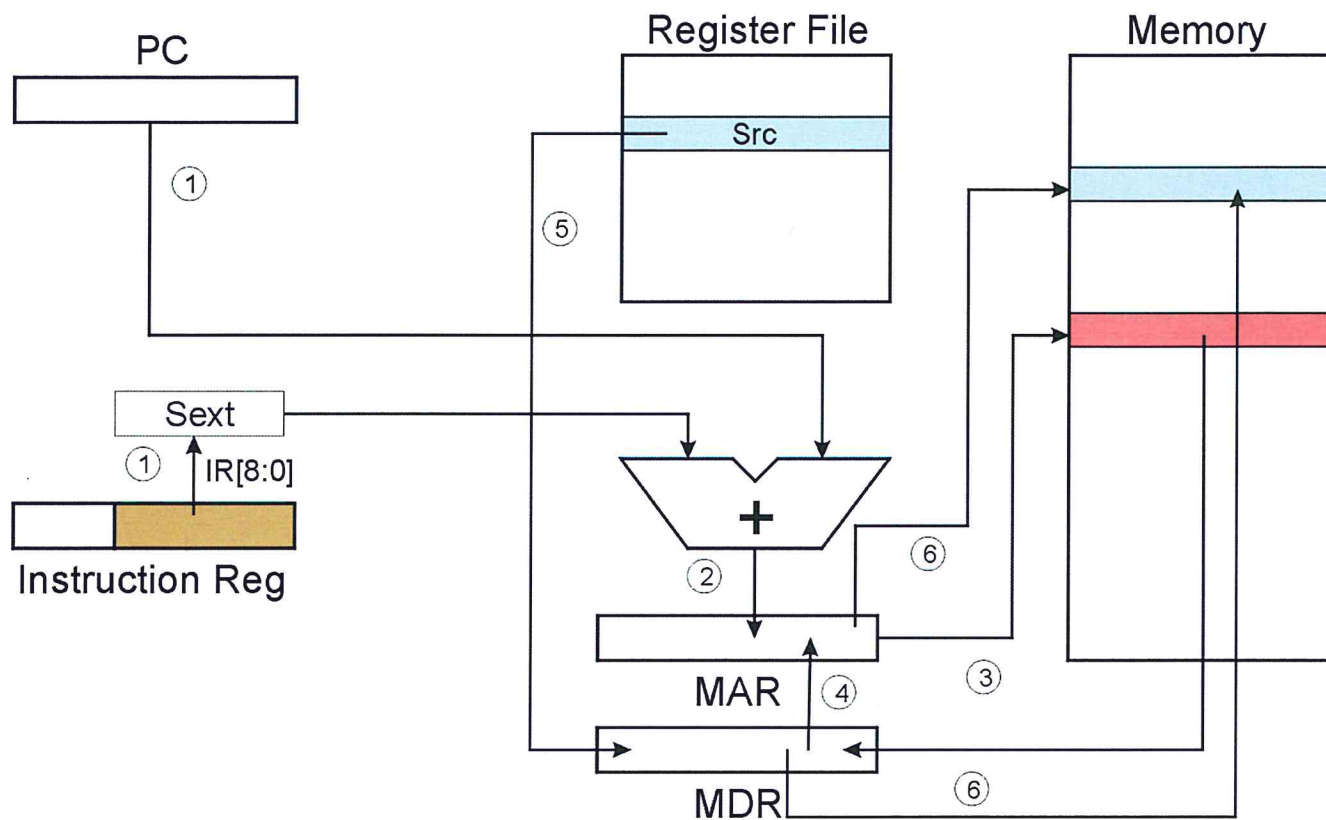
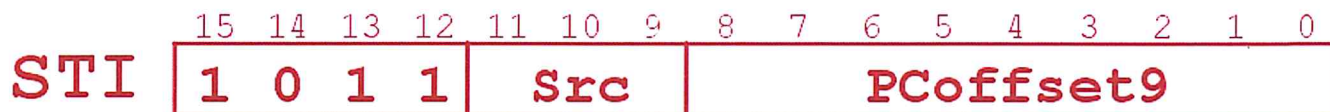
X 3000

MDR

④ X 3000

⑤ X 00FF

# 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?

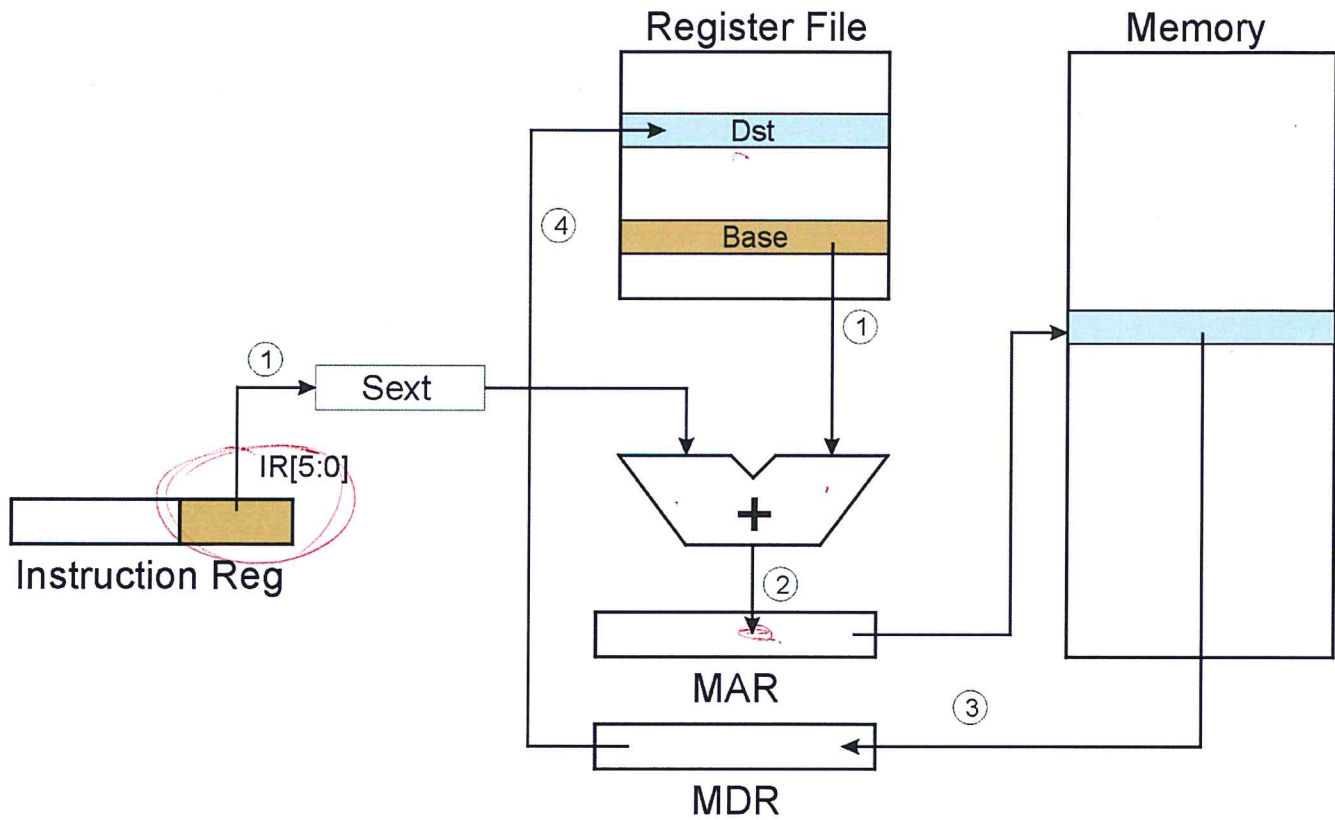
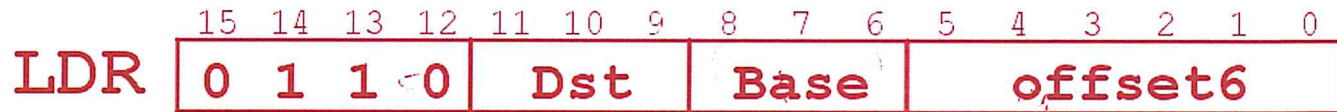
### Solution #2:

- 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)



# Example

| Address      | Instruction |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   | Comments                                                                     |
|--------------|-------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|------------------------------------------------------------------------------|
| LEA<br>x30F6 | 1           | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |   | $R1 \leftarrow PC - 3 = x30F4$                                               |
| ADD<br>x30F7 | 0           | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | $R2 \leftarrow R1 + 14_{10} = x3102$                                         |
| ST<br>x30F8  | 0           | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |   | $M[PC-5] \leftarrow R2$<br>$M[x30F4] \leftarrow x3102$                       |
| AND<br>x30F9 | 0           | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | $R2 \leftarrow 0$                                                            |
| ADD<br>x30FA | 0           | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | $R2 \leftarrow R2 + 5 = 5$                                                   |
| STR<br>x30FB | 0           | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | $M[R1+14] \leftarrow R2$<br>$M[x3102] \leftarrow 5$                          |
| LDI<br>x30FC | 1           | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | $R3 \leftarrow M[M[x30F4]]$<br>$R3 \leftarrow M[x3102]$<br>$R3 \leftarrow 5$ |

opcode

$= 1001$

$= -9$

$x30FD - 9 =$

$x30FD - 9 = x30F4$

# 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.”*

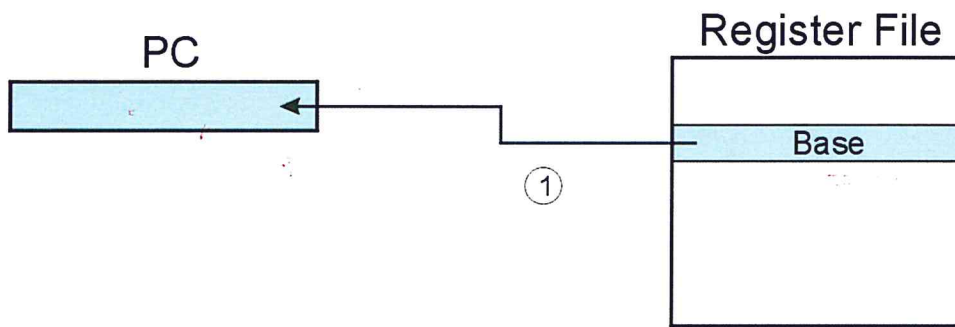


## JMP (Register)

Jump is an unconditional branch -- always taken.

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

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



# 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

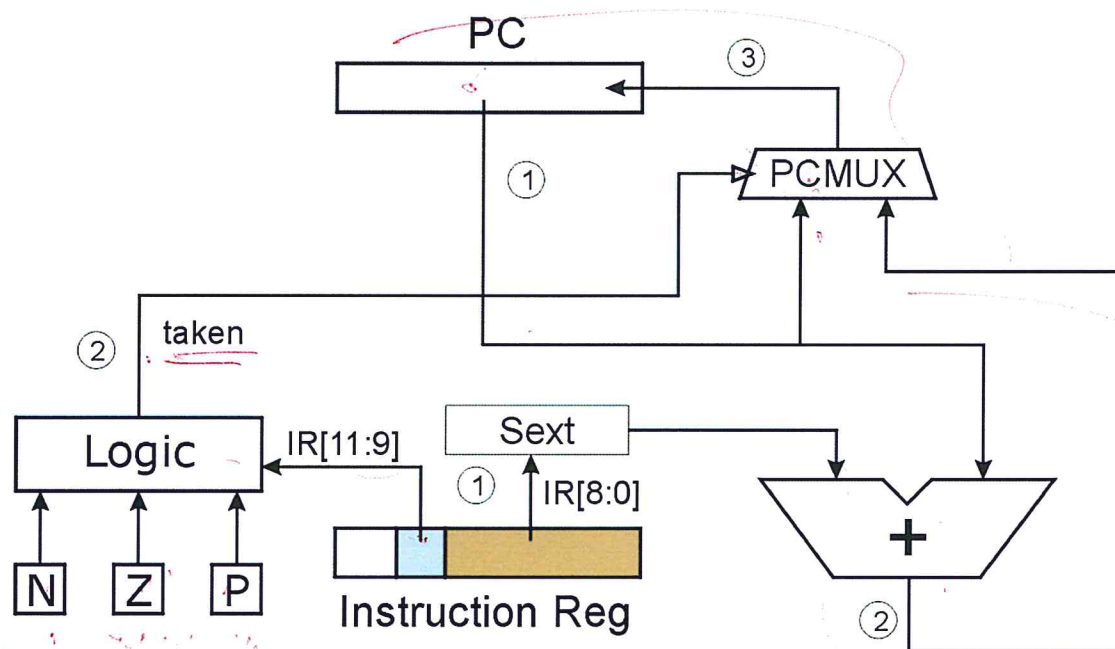
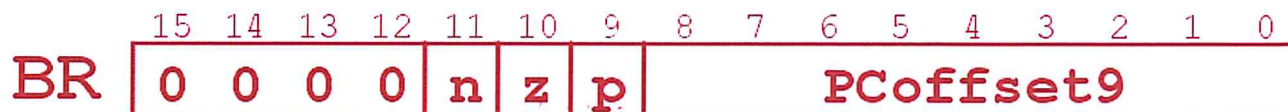
Branch specifies one or more condition codes.

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?

The diagram illustrates the internal architecture of the 68000 microprocessor, divided into two main horizontal sections. The top section contains the core processing units, while the bottom section handles memory and I/O operations.

**Top Section (Core Processing):**

- Control and Instruction Flow:** The **CONTROL** unit is central, receiving a **16-bit** **LD.CC** signal and outputting **N**, **Z**, and **P** status flags to the **LOGIC** block. It also manages the **IR** (Instruction Register) via **LD.IR** and the **PC** (Program Counter) via **LD.PC**. The **PC** is incremented by **+1** and its next value is selected by the **PCMUX** (Program Counter Multiplexer). A **TAKEN** signal (handwritten in red) is shown as an input to the **PCMUX**. The **PCMUX** also receives a **2-bit** **LD.PC** signal and a **16-bit** **PC** signal. The **PCMUX** output is selected by the **PCMUX** and the **PC** signal.
- Addressing:** The **ADDR1MUX** and **ADDR2MUX** (Address Multiplexers) receive **16-bit** **PC** and **PCMUX** signals. The **ADDR1MUX** also receives a **16-bit** **PC** signal. The **ADDR2MUX** receives a **2-bit** **LD.PC** signal and a **16-bit** **PC** signal. The **ADDR1MUX** and **ADDR2MUX** outputs are selected by the **ADDR1MUX** and **ADDR2MUX** signals. The **ADDR1MUX** output is selected by the **ADDR1MUX** and the **ADDR2MUX** output is selected by the **ADDR2MUX**.
- Register File:** The **REG FILE** (Register File) receives **3-bit** **LD.REG** and **SR2** signals. It outputs **16-bit** **SR2 OUT** and **16-bit** **SR1 OUT** signals. The **SR1 OUT** is selected by the **SR1** signal.
- ALU and Status:** The **ALU** (Arithmetic Logic Unit) receives **16-bit** **SR2 OUT** and **16-bit** **SR1 OUT** signals. It also receives a **2-bit** **ALUK** signal. The **ALU** output is selected by the **ALU** signal. The **ALU** output is selected by the **ALU** signal.
- Instruction Register (IR):** The **IR** receives a **16-bit** **LD.IR** signal and outputs **16-bit** **IR** signals. The **IR** output is selected by the **IR** signal.
- Instruction Multiplexer (MARMUX):** The **MARMUX** (Instruction Multiplexer) receives **16-bit** **IR** and **16-bit** **PC** signals. It outputs **16-bit** **MARMUX** signals. The **MARMUX** output is selected by the **MARMUX** signal.
- Instruction Register (IR):** The **IR** receives a **16-bit** **LD.IR** signal and outputs **16-bit** **IR** signals. The **IR** output is selected by the **IR** signal.
- Instruction Register (IR):** The **IR** receives a **16-bit** **LD.IR** signal and outputs **16-bit** **IR** signals. The **IR** output is selected by the **IR** signal.

**Bottom Section (Memory and I/O):**

- Memory and I/O:** The **MEMORY** block receives **16-bit** **ADDR1MUX** and **16-bit** **ADDR2MUX** signals. It outputs **16-bit** **MEMORY** signals. The **MEMORY** output is selected by the **MEMORY** signal.
- Address Control Logic (ADDR. CTL. LOGIC):** The **ADDR. CTL. LOGIC** (Address Control Logic) receives **16-bit** **MEMORY** and **16-bit** **PC** signals. It outputs **16-bit** **ADDR. CTL. LOGIC** signals. The **ADDR. CTL. LOGIC** output is selected by the **ADDR. CTL. LOGIC** signal.
- Input and Output:** The **INPUT** block receives **16-bit** **ADDR. CTL. LOGIC** and **16-bit** **PC** signals. It outputs **16-bit** **INPUT** signals. The **INPUT** output is selected by the **INPUT** signal.
- Output:** The **OUTPUT** block receives **16-bit** **ADDR. CTL. LOGIC** and **16-bit** **PC** signals. It outputs **16-bit** **OUTPUT** signals. The **OUTPUT** output is selected by the **OUTPUT** signal.
- Input Multiplexer (INMUX):** The **INMUX** (Input Multiplexer) receives **16-bit** **INPUT** and **16-bit** **PC** signals. It outputs **16-bit** **INMUX** signals. The **INMUX** output is selected by the **INMUX** signal.
- Output Multiplexer (OUTMUX):** The **OUTMUX** (Output Multiplexer) receives **16-bit** **OUTPUT** and **16-bit** **PC** signals. It outputs **16-bit** **OUTMUX** signals. The **OUTMUX** output is selected by the **OUTMUX** signal.

**Handwritten Annotations:**

- TAKEN:** A red handwritten word above the **PCMUX** input.
- 1:** A red circled number next to the **ADDR1MUX** input.
- 2:** A red circled number next to the **PCMUX** input.

