The LC-3

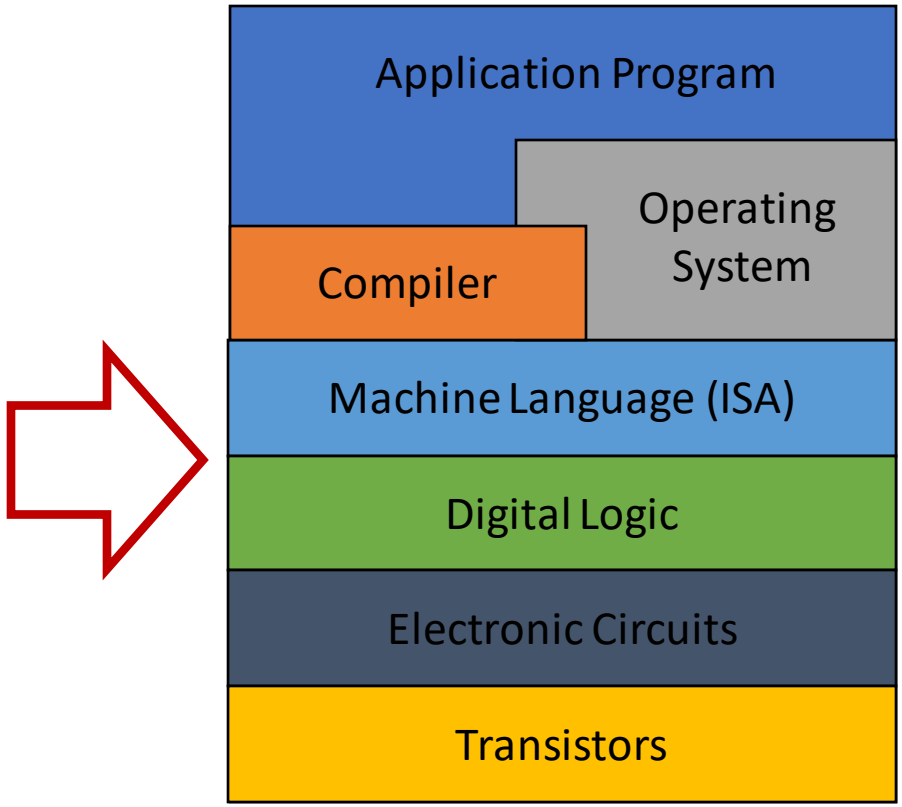
EECS388 Fall 2022

© Prof. Mohammad Alian

Context

- Recommended reading

Chapter 5 of “Introduction to Computing,” Patt, Patel



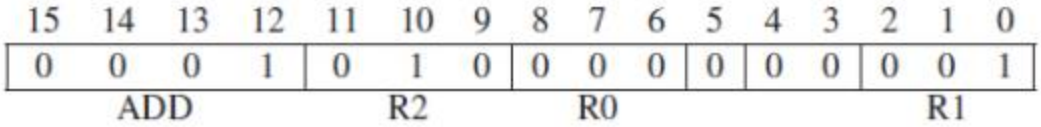
** Picture in the slides are in part taken from “Introduction to Computing From bits & gates to C and beyond” Patt, Patel

Instruction Set Architecture (ISA)

- Memory organization
  - Address space: 2^16
  - Addressability: 16 bits
- Genera Purpose Registers (GPR)
  - 8 GPR x 16 bits
- Available instruction
  - Defined by opcode, data types, and addressing mode
  - LC-3: 15 opcodes, 1 opcode reserved

General Purpose Registers (GPR)

- Can be accessed in one clock cycle



Register 0	(R0)	0000000000000001
Register 1	(R1)	0000000000000011
Register 2	(R2)	0000000000000101
Register 3	(R3)	0000000000000111
Register 4	(R4)	1111111111111110
Register 5	(R5)	1111111111111100
Register 6	(R6)	1111111111111010
Register 7	(R7)	1111111111111000

Register file snapshot before ADD instruction execution

Register 0	(R0)	0000000000000001
Register 1	(R1)	0000000000000011
Register 2	(R2)	0000000000000100
Register 3	(R3)	0000000000000111
Register 4	(R4)	1111111111111110
Register 5	(R5)	1111111111111100
Register 6	(R6)	1111111111111010
Register 7	(R7)	1111111111111000

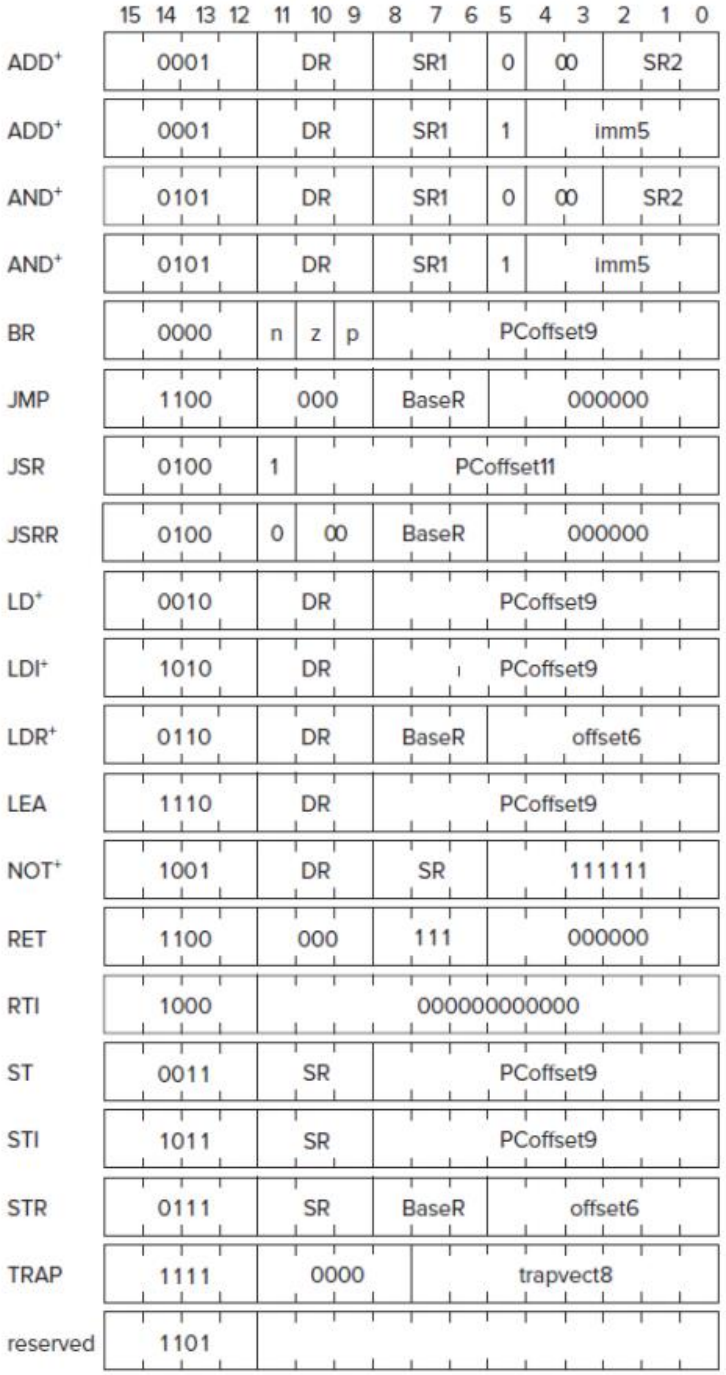
Register file snapshot after ADD instruction execution

LC-3 Instruction Set

- Opcodes: bits [15:12]
  - ADD & ADDi, AND & ANDi, BR & RET share opcodes => total 19 instructions
- Data type:
  - 2's complement
- Addressing Mode: 5 modes
  - Immediate, register, PC-relative, indirect, Base+offset

Memory

- Condition Codes:
  - 3 single bit registers: N, Z, P



Condition Codes

- N (negative), Z (zero), P (Positive) registers
- Each time a GPR is written by an *operate* or *load* instruction the condition codes are individually set
- E.g., after the following code: N = 0, Z = 0, P = 1

```
R1 <- 0
R2 <- 1
R3 <- R1 + R2
```

Instruction Types

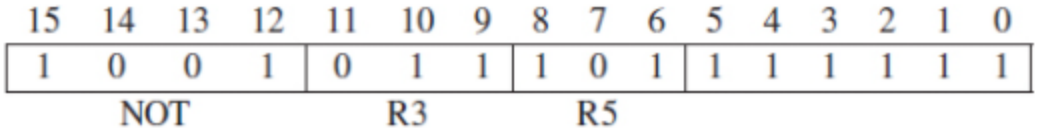
- Operate
- Data Movement
- Control

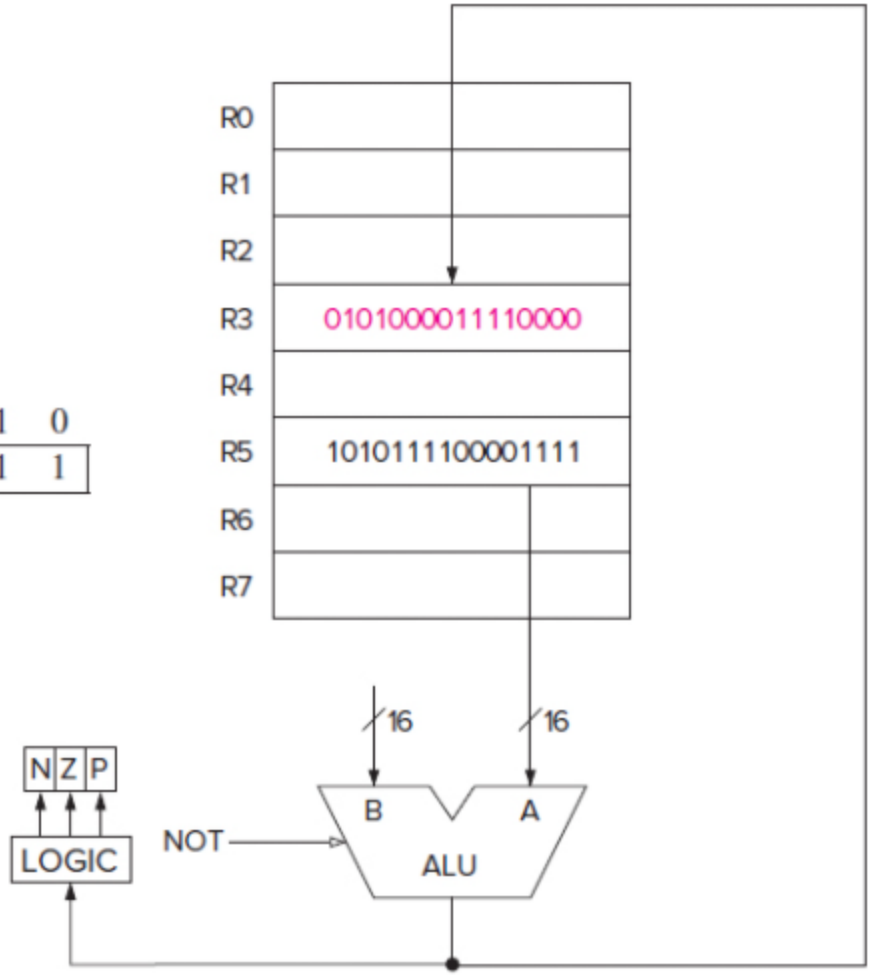
Instruction Types

- Operate
- Data Movement
- Control

Operate Instructions

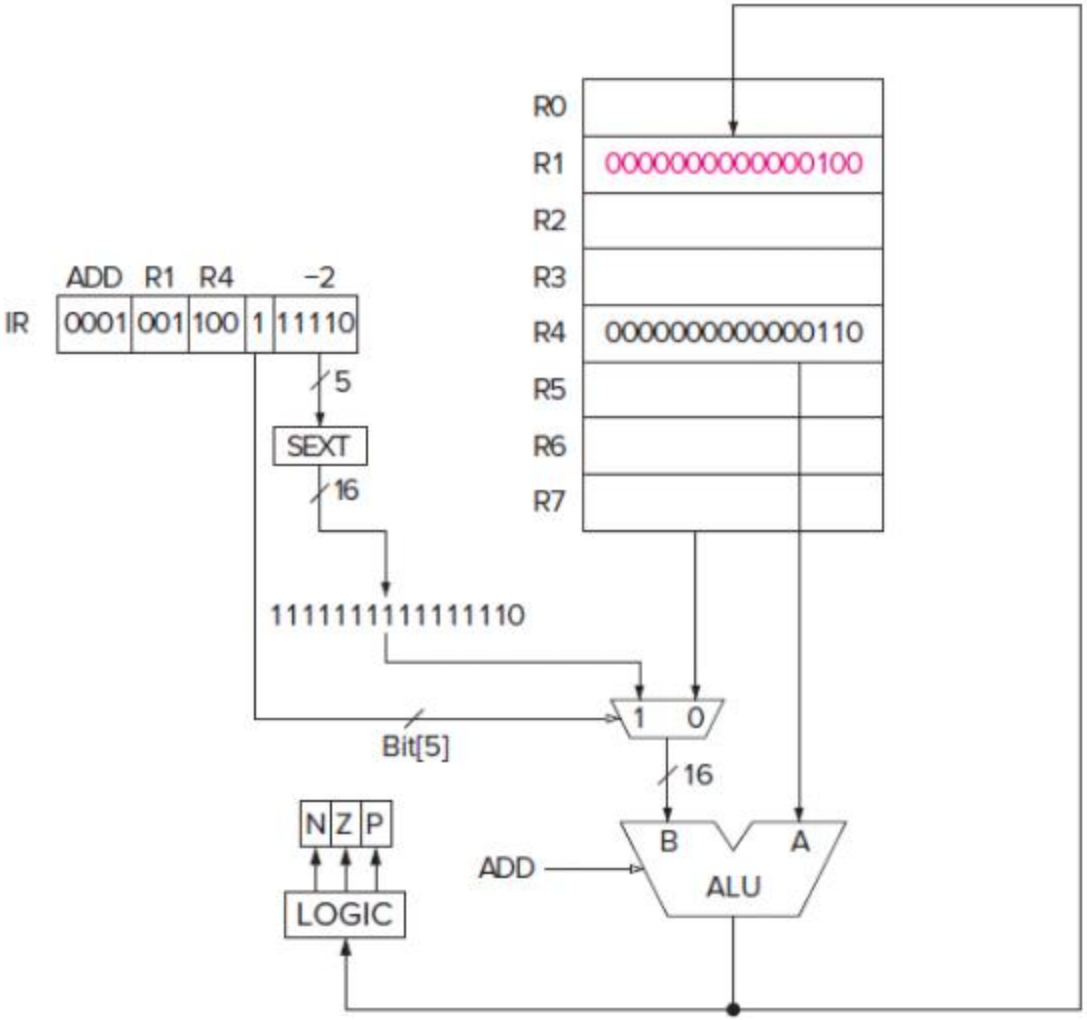
- ADD, AND, NOT
- NOT only operate on one operand



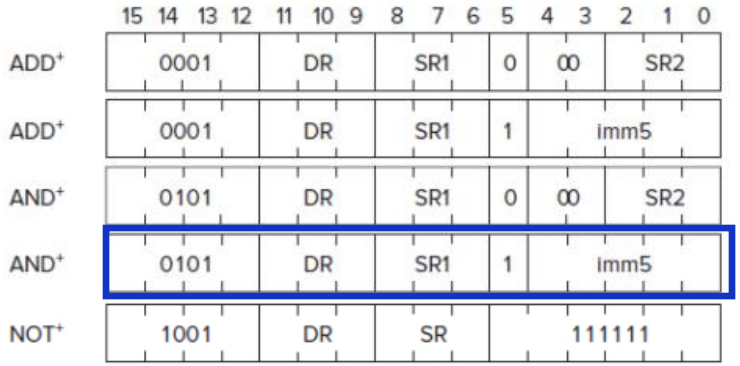


Operate Instructions (Cont.)

- Immediates



Example



- **What does this instruction do?**

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

AND R2 R2 Imm

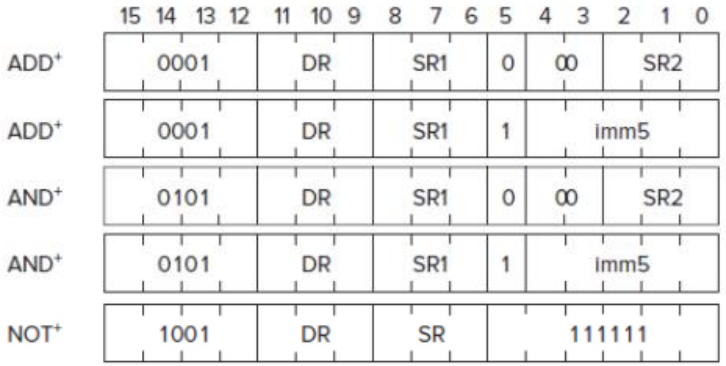
```
R2 <- R2 & SEXT(0b00000)
```

=> Clear R2

LC-3 Program: Negate a 2's Compl.

- **Write a program that negate a 2's complement number.**
- **Hint:** you can negate a 2's complement by complementing the number and add 1
- Examples:
  - 0b0101 (5) → 0b1010 + 0b0001 = 0b1011 (-5)
  - 0b1001 (-7) → 0b0110 + 0b0001 = 0b0111 (7)

Negate a 2's Compl.



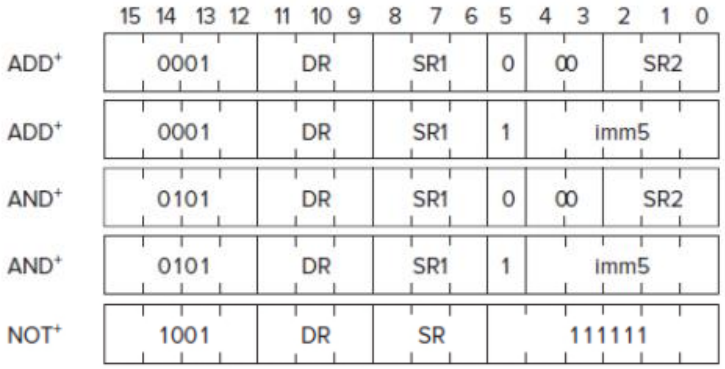
- **Write a program that negate a 2's complement number. (assume its in R1)**

```
R1 <- NOT(R1)    1001 001 001 111111
R1 <- R1 + 1     0001 001 001 1 00001
```

LC-3 Program: A minus B

- Implement “A = A minus B”, assume A is in R2 and B is in R1.
- Steps
  - Negate B
  - A = A + B

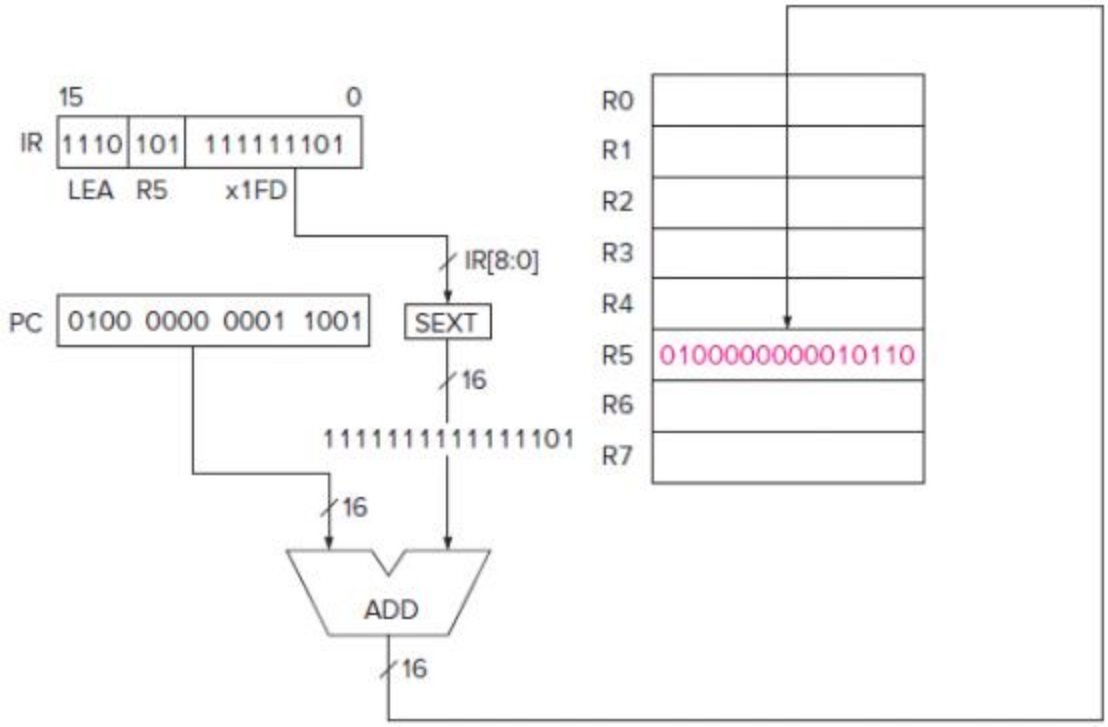
```
1001 001 001 111111
0001 001 001 1 00001

0001 010 001 0 00 010
```



Load Effective Address (LEA) Instruction

- Load DR register with

PC + SEXT[8:0]



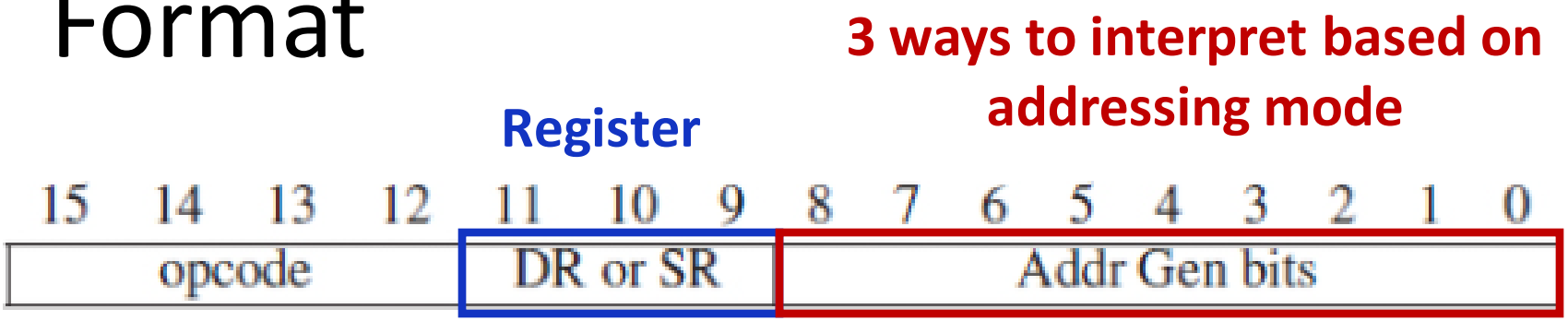
Instruction Types

- Operate
- Data Movement
- Control

Data Movement Instructions

- Move data between
  - Registers and memory
  - Registers and I/O device registers
- Load: moving data from memory to a register
- Store: moving data from register to memory

Data Movement Instruction Format



- Two operands:
  - 1 source: data to be moved
  - 1 destination: location where data is moved to

  One of the operands is a **register**, and other one is a **memory location** or an **I/O device**

LC-3 Addressing Modes for Data movement Instructions

- PC-Relative Mode
- Indirect Mode
- Base+Offset Mode

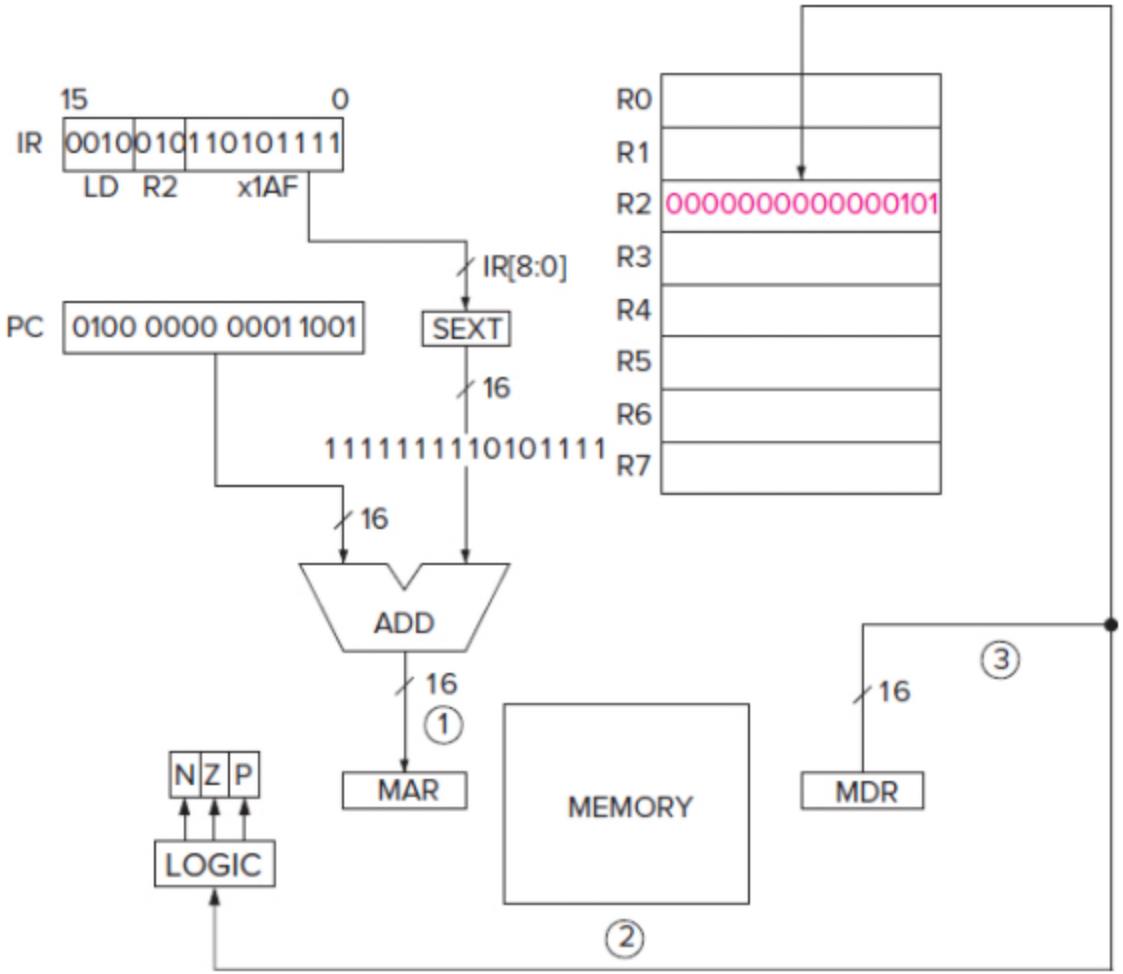
PC-Relative Mode

- LD and ST
- Address:

SEXT(bits[8:0]) + PC

Limited range:

[+255, -256] of PC

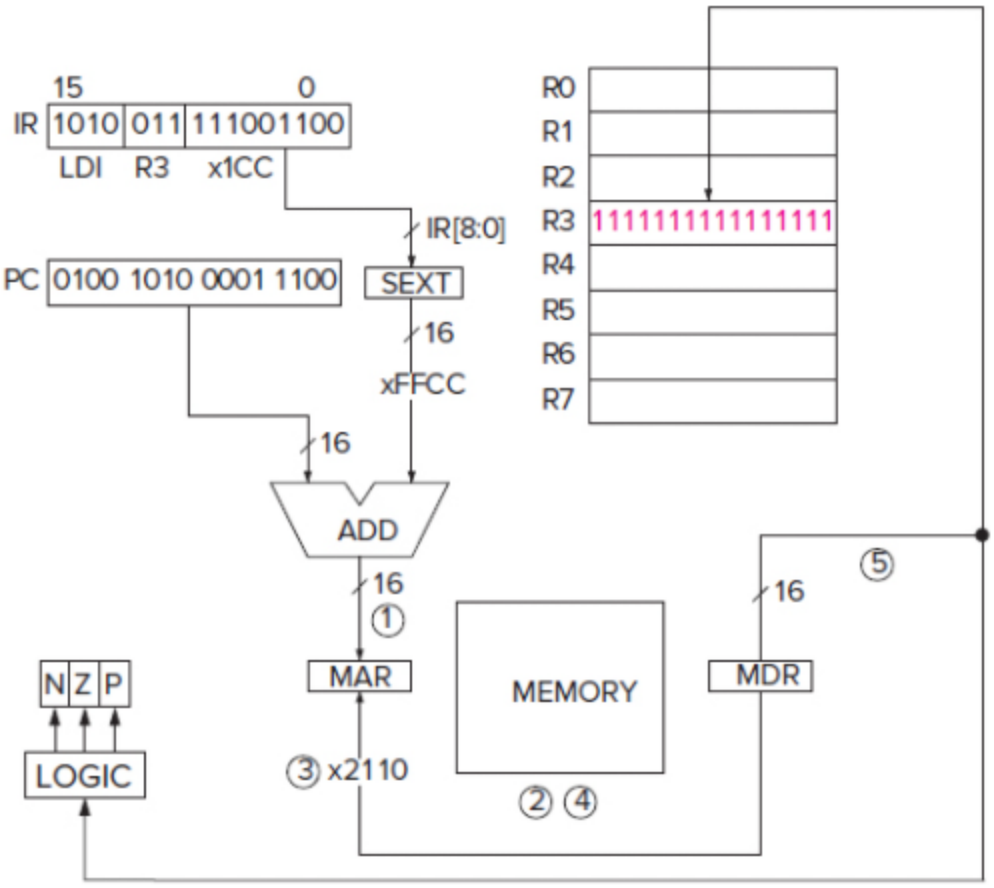


Indirect Mode

- LDI and STI
- Address:

Mem[Mem[SEXT(bits[8:0]) + PC]]

Address is not limited to the range provide by bits[8:0]

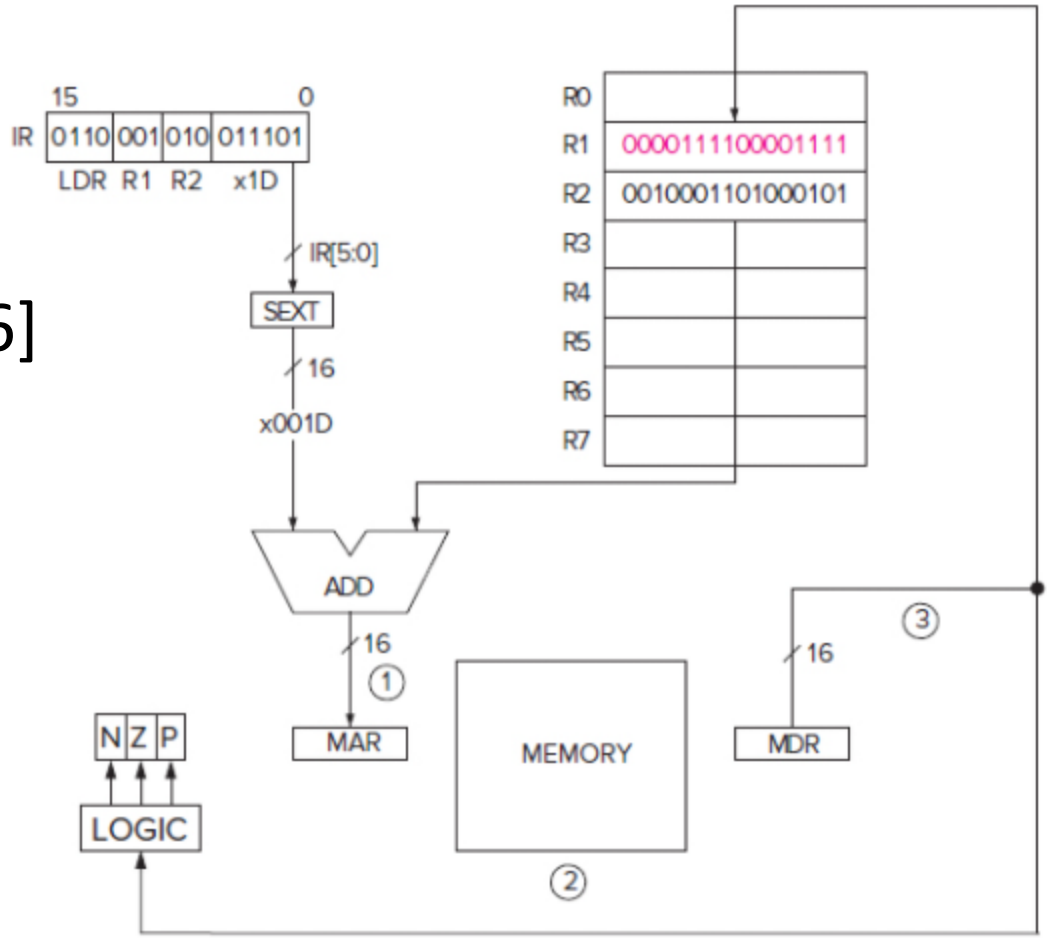


Base+Offset Mode

- LDR and STR
- Address:

SEXT(bits[5:0]) + Reg#[8:6]

Address is not limited to the range provide by bits[5:0]



Instruction Types

- Operate
- Data Movement
- Control

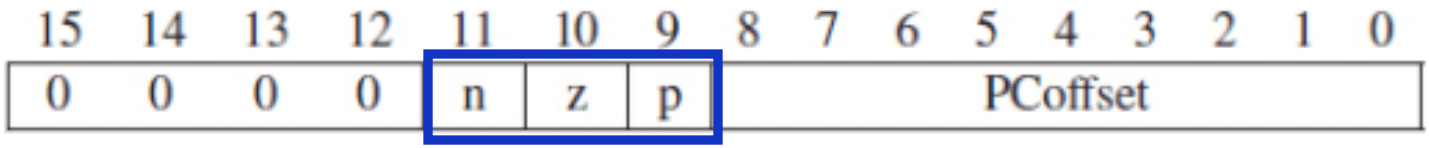
LC-3 Control Instructions: *alter the sequential execution of instructions*

- Conditional branch
- Unconditional Jump
- Subroutine (function) call
- TRAP (or *service call*)
- RTI (Return from Trap or Interrupt)

Conditional Branches

- BR (opcode=0000): Execute next instruction *in sequence* or execute one *out of sequence*?
  - In another word, leave the incremented PC *unchanged* or *change* it.
- Decision is made based on the previously executed instructions
  - Reflected in the ***condition codes***

BR Format

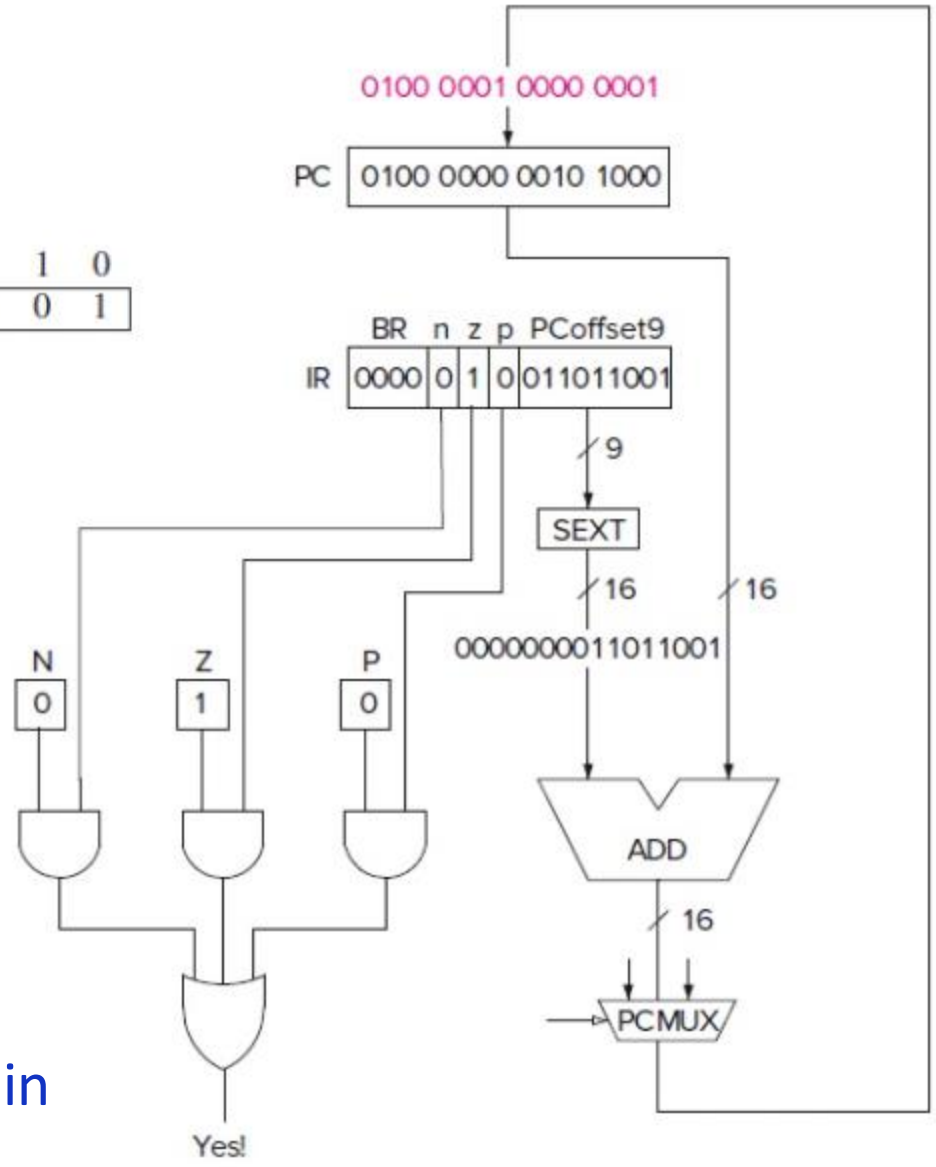


Condition Codes (CC) set by: ADD, AND, NOT, LD, LDI, LDR

- If CC.N=bit[11], CC.Z=bit[10], CC.P=bit[9]
  - Next instruction gets executed from *Mem[PC + SEXT(PCoffset)]*
- Otherwise
  - Next instruction gets executed from *Mem[PC]*

BR Example

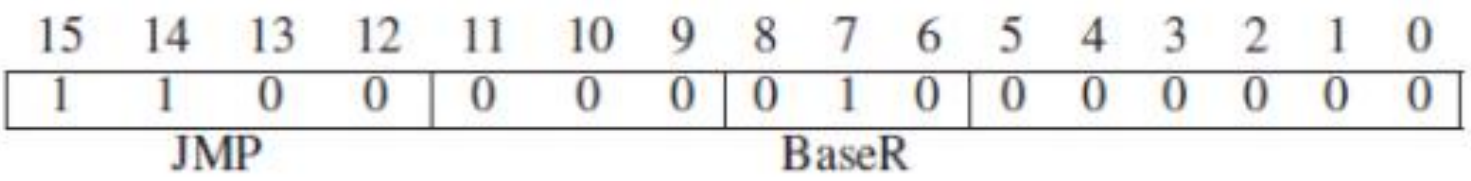
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	1	1	0	1	1	0	0	1
BR				n	z	p	x0D9								



Limitation of BR: only jumps within [+255, -256] range of the PC

Jump (JMP) Instruction

- Unconditional jump
- JMP R_{base}
  - Execute next instruction from $Mem[R_{base}]$
- *E.g., if R2 = 0x00FA*



Next instruction will execute from 0x00FA

TRAP Instruction

- Invokes an OS service call
- Bits[7:0] or *trapvector* identifies the service that the program wishes OS to do
- Once OS is done performing the service call, it will set PC to the next instruction following TRAP
- Examples of LC-3 *trapvectors*:

```
Read a char from keyboard (0x23)
Write a char to monitor (0x21)
Halt (0x25)
```

Recap

- LC-3 ISA
  - Operate instructions
  - Data movement instructions
  - Control instructions
- We are now ready to write our first LC-3 Program!