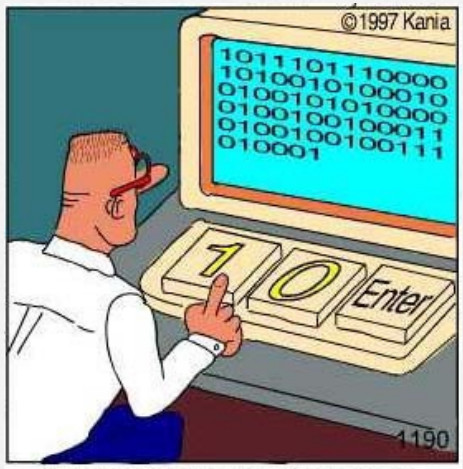
COMP1003 Computer Organization

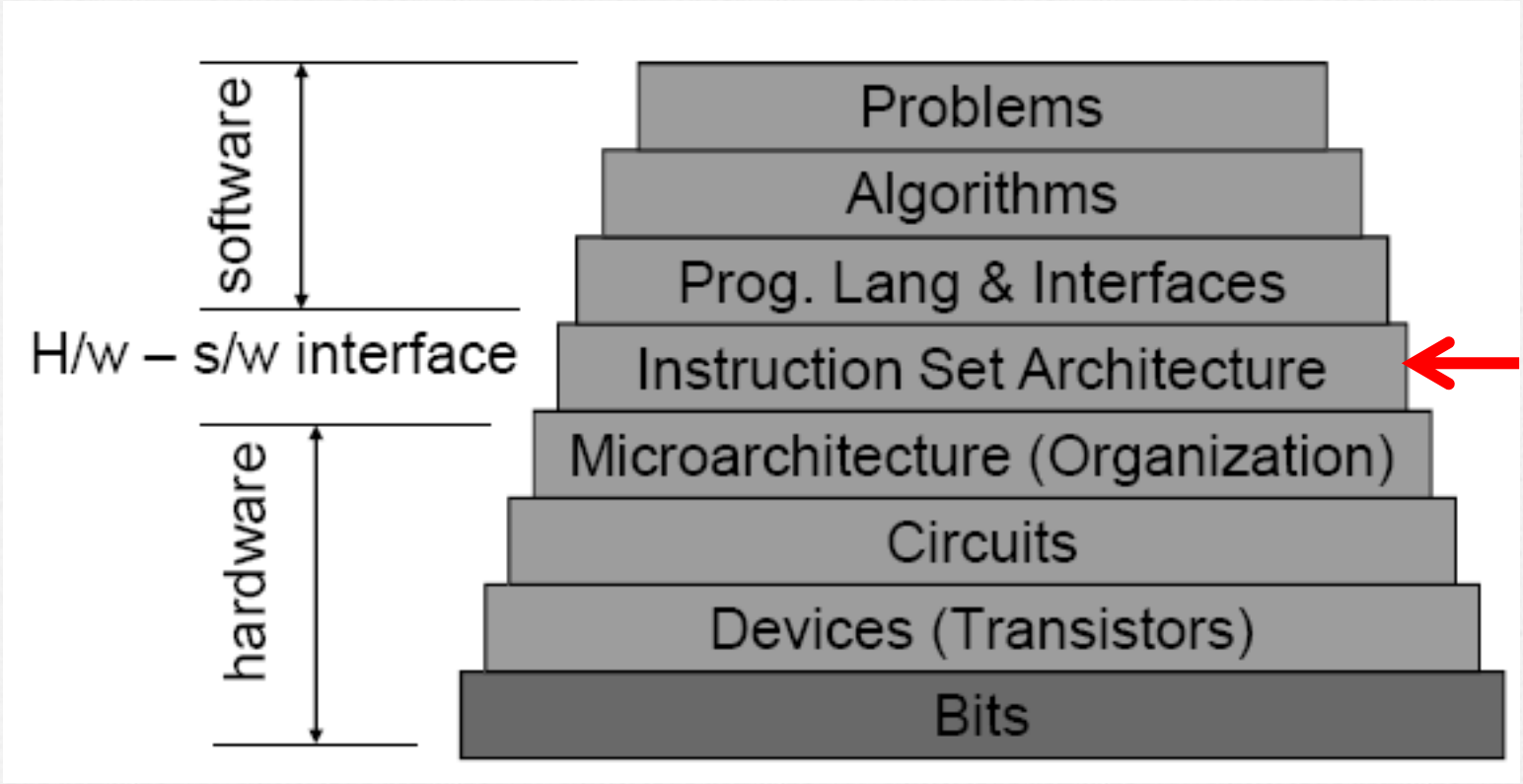
Lecture 9 The LC-3 ISA



Real programmers code in binary.

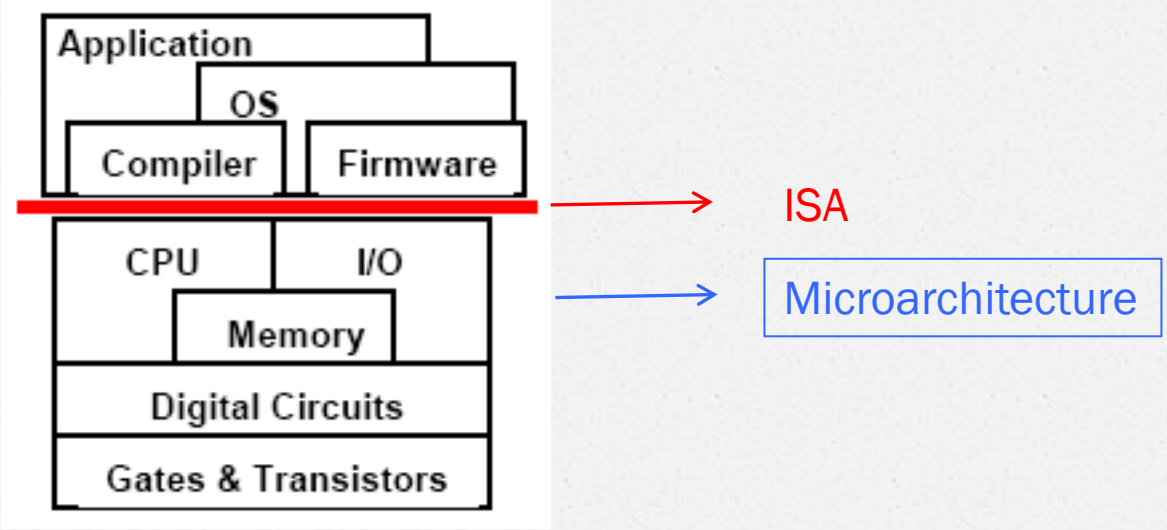
United International College

ISA


software
H/w – s/w interface
hardware
Problems
Algorithms
Prog. Lang & Interfaces
Instruction Set Architecture
Microarchitecture (Organization)
Circuits
Devices (Transistors)
Bits

ISA and Microarchitecture

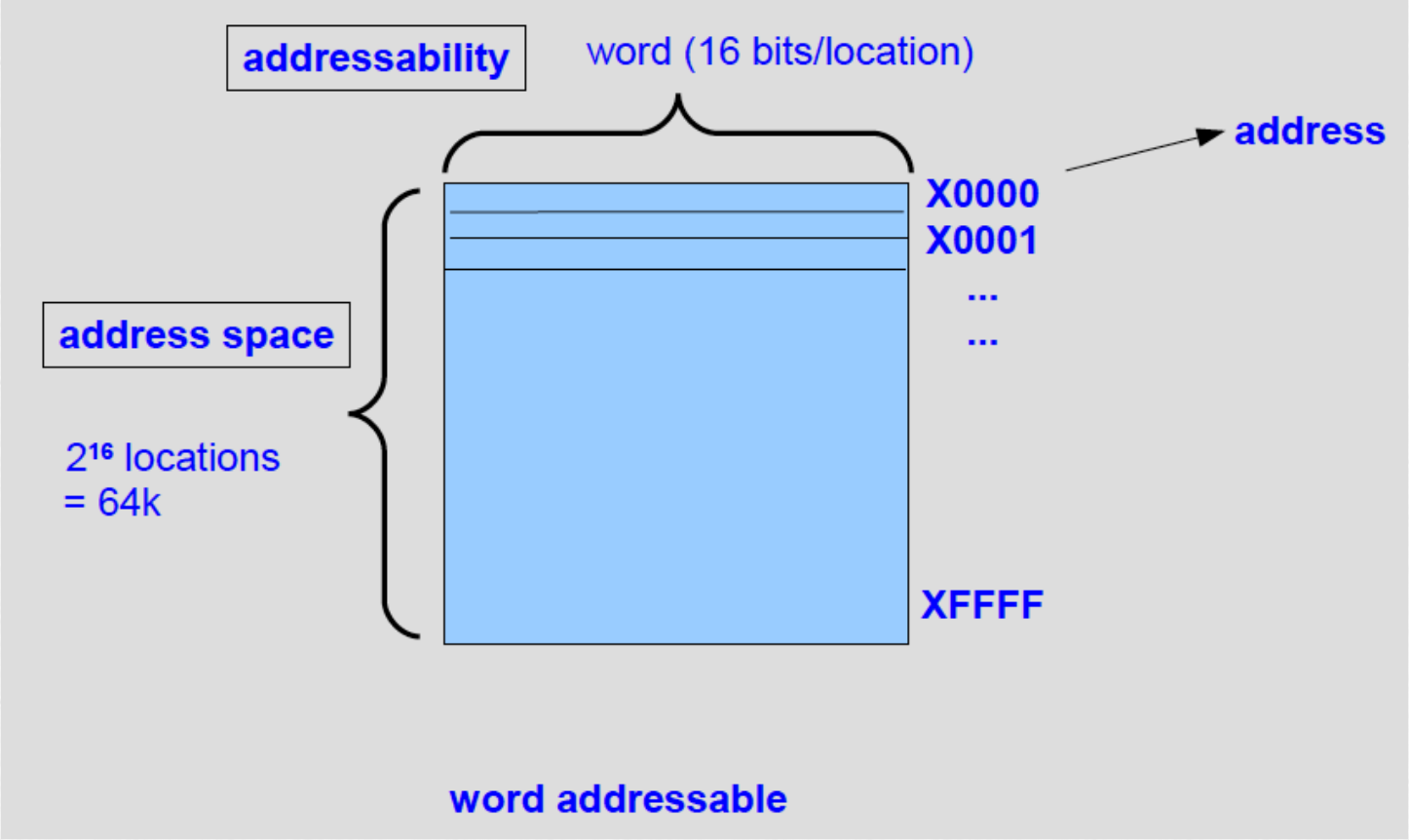
- ISA specifies what hardware does, but not how
- Microarchitecture specifies how it does it

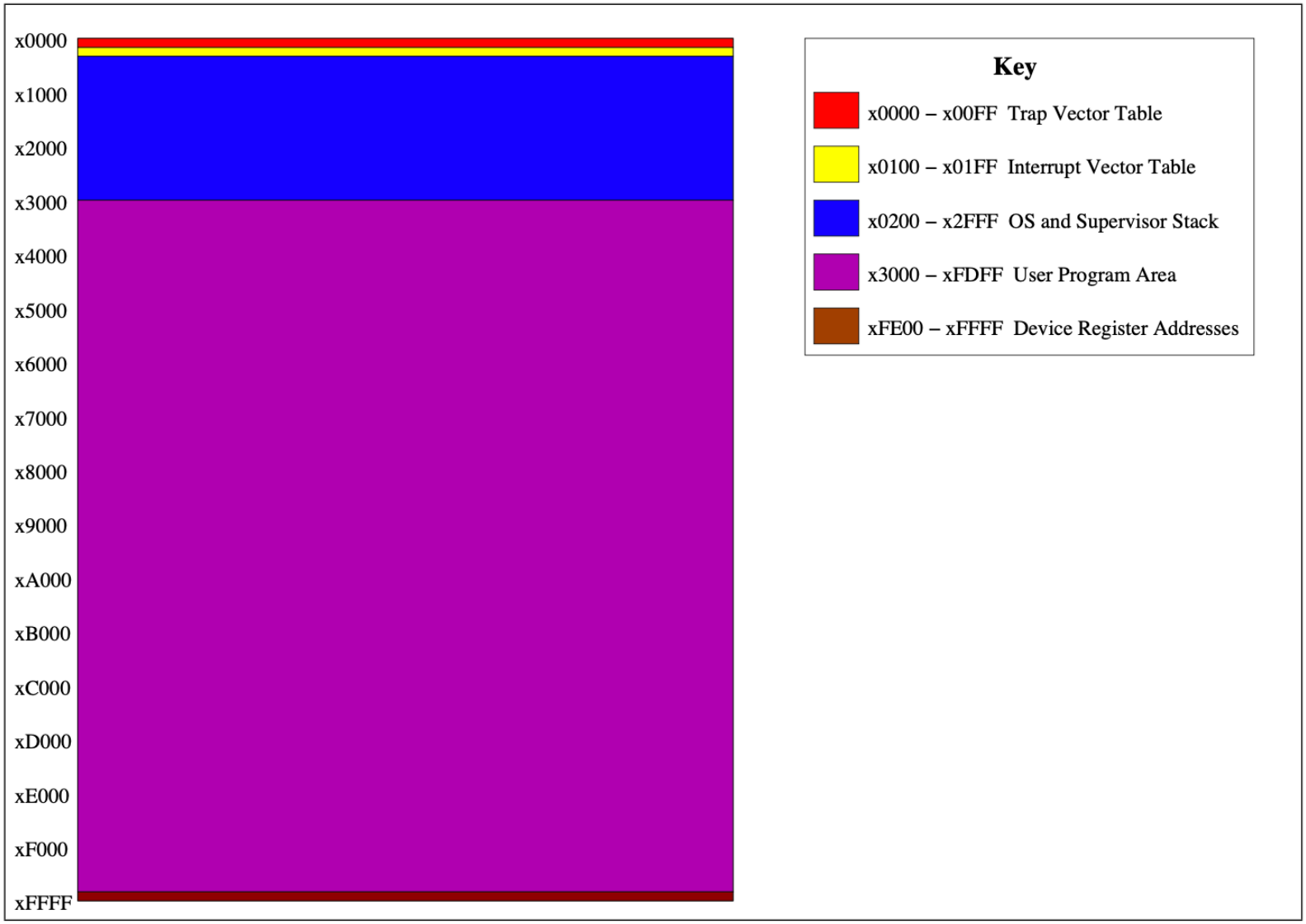


The ISA Overview

- The ISA specifies all the information about the computer that the software has to know.
  - Memory
    - Address space (how many locations?)
    - Addressability (word or byte, word size)
  - Registers
    - Number (how many?)
    - Type
  - Instructions
    - Operations
    - Data types
    - Addressing modes

LC-3 Memory Organization



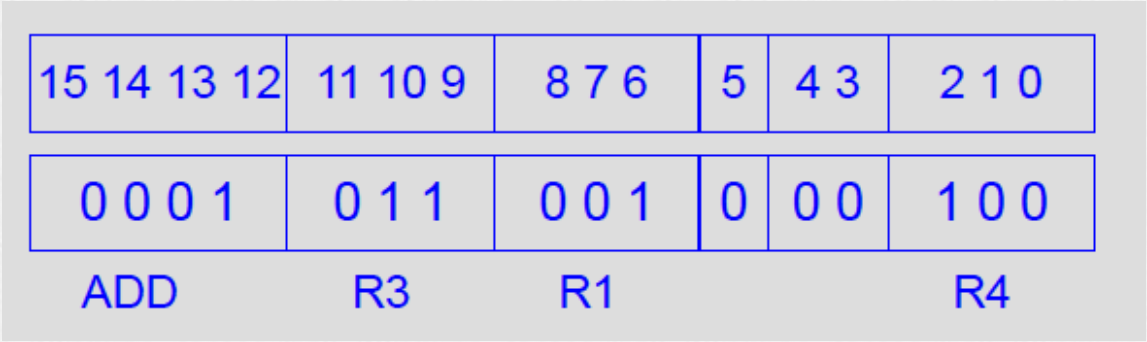

x0000
x1000
x2000
x3000
x4000
x5000
x6000
x7000
x8000
x9000
xA000
xB000
xC000
xD000
xE000
xF000
xFFFF
Key
x0000 – x00FF Trap Vector Table
x0100 – x01FF Interrupt Vector Table
x0200 – x2FFF OS and Supervisor Stack
x3000 – xFDFF User Program Area
xFE00 – xFFFF Device Register Addresses

Registers

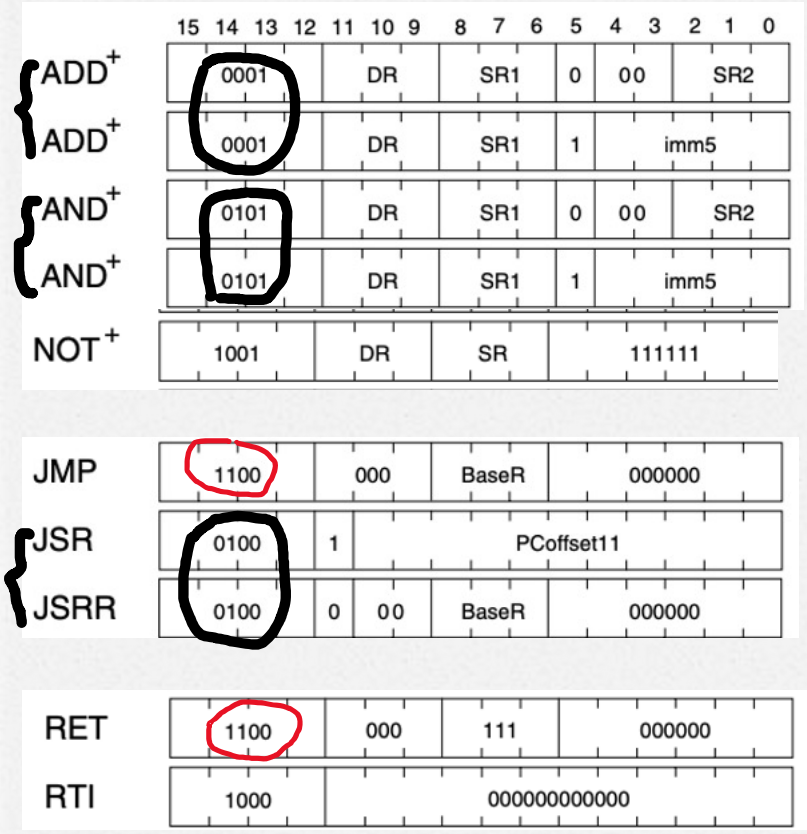
- Registers
  - Special storage devices that are inside the CPU
  - Very fast to access: 1 clock cycle
  - General Purpose Registers (GPR): accessible by instructions
  - Other registers may not be accessible
- LC-3
  - 8 GPRs: R0, R1, ... R7, each 16 bits
  - Other special purpose registers
    - MAR, MDR
    - PC, IR
    - Condition Codes: P,Z,N
    - KBDR, KBSR, DDR, DSR

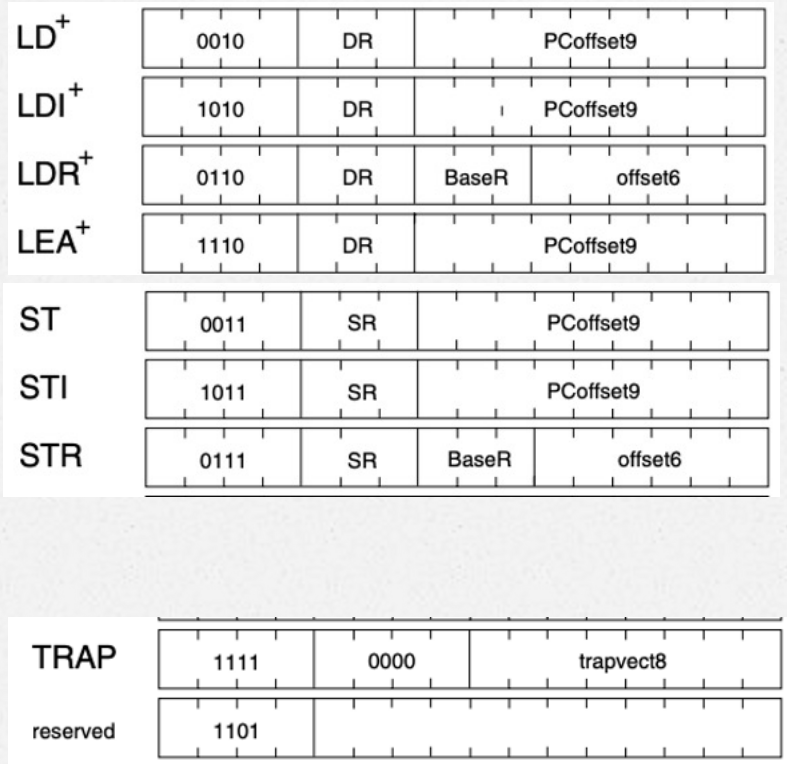
Instructions

- An instruction is made up of two things
  - Opcode (what the instruction does)
  - Operands (what the operation acts on)
- LC-3
  - 4 bits opcode (how many possible instructions?)
  - Up to 2 sources and one destination



Opcodes & Operations





Data Types

- What data types are supported?

- LC-3
  - 2's complement integers

Condition Codes

- 3 single bit registers (set to 1 or cleared to 0)
  - N: when value written was negative
  - Z: when value written was zero
  - P: when value written was positive
- Affected each time any register is written
- Condition codes are read by conditional branch instructions

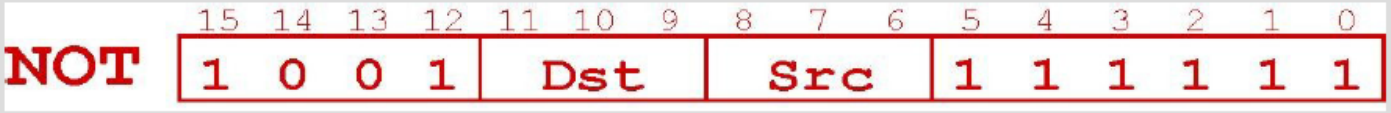
Addressing Modes

- Where to find the operands?
  - The addressing modes provide several ways for the instructions to specify the location of an operand
- 5 addressing modes
  - explicitly in the instruction itself (immediate)
  - in a register
  - in memory,
    - PC-relative mode: addr. = PC + offset
    - Base+offset mode: addr. = a base register + offset
    - Indirect mode: the address of a location that **contains the address** of the operand (**indirect**) Think! What is this?

3 Operate Instructions

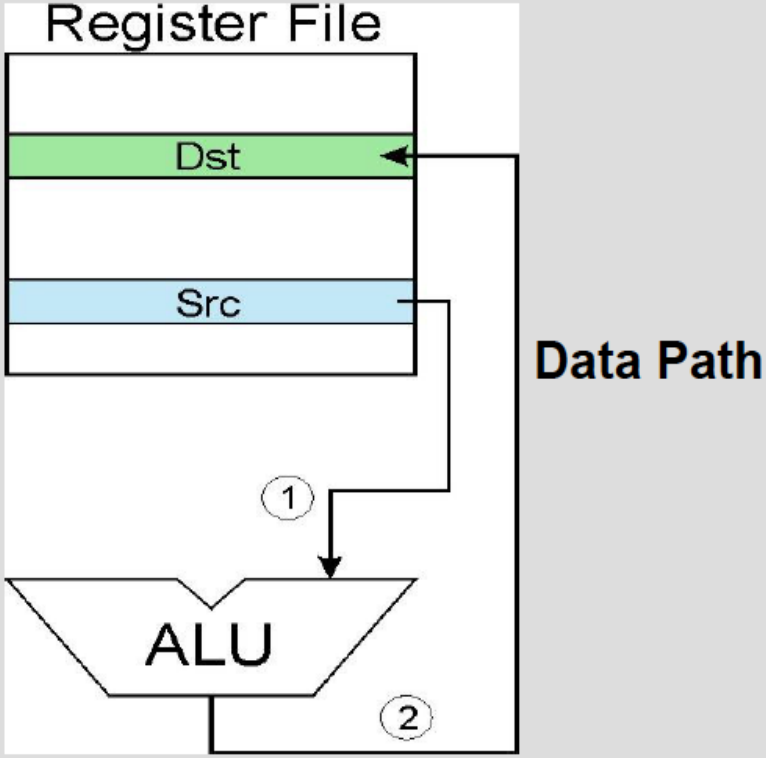
- Arithmetic: LC-3 has only ADD
- Logic: LC-3 has NOT, AND
- Source and destination operands are registers
  - These instructions *do not* reference memory.
  - ADD and AND can use "immediate" mode, where one operand is hard-wired into the instruction.

NOT

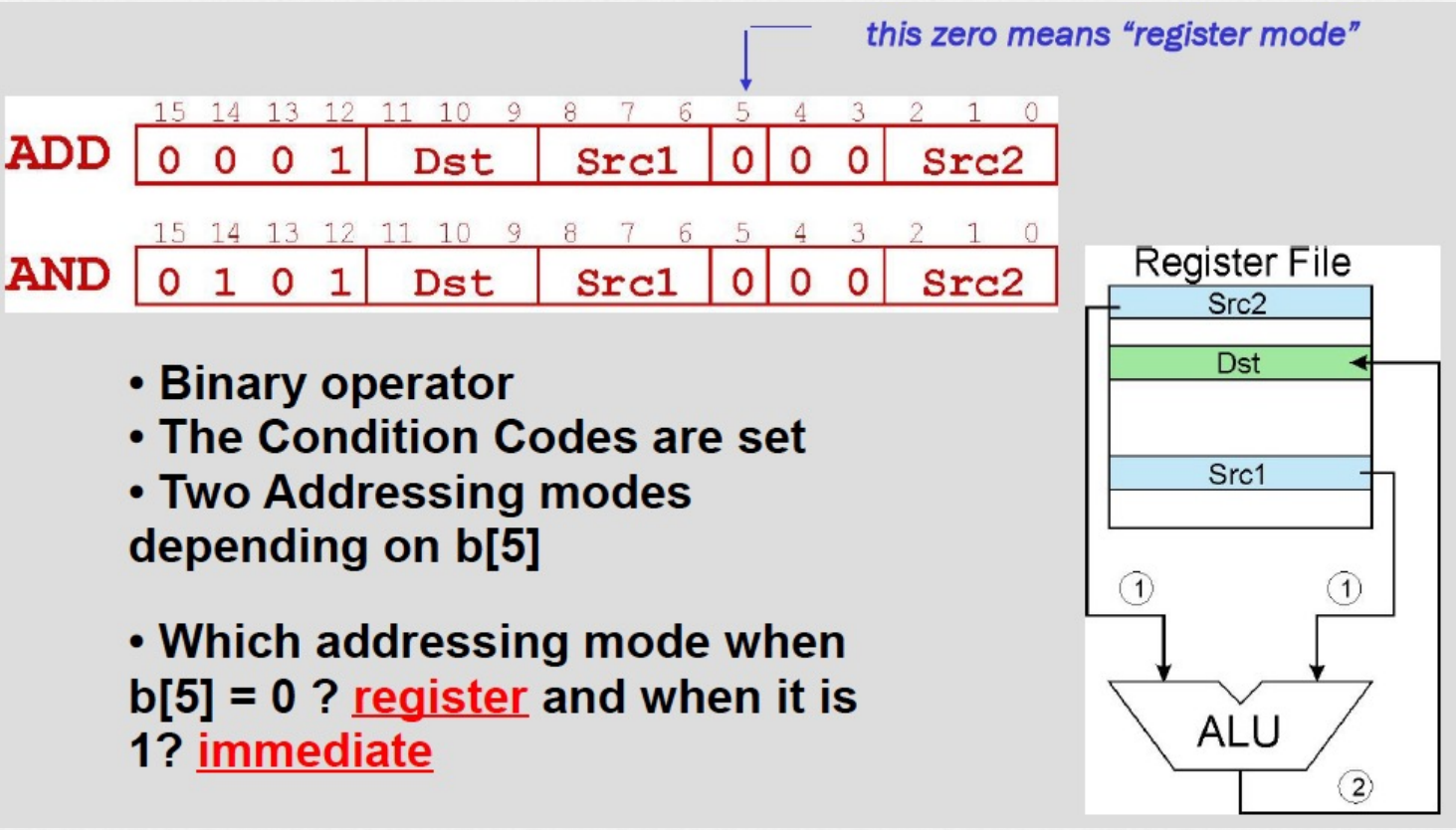


- **The bit-wise complement of Src is stored in Dst.**
- **Unary operator**
- **The Condition Codes are set**
- **Which Addressing mode?**
  - **Register Addressing**

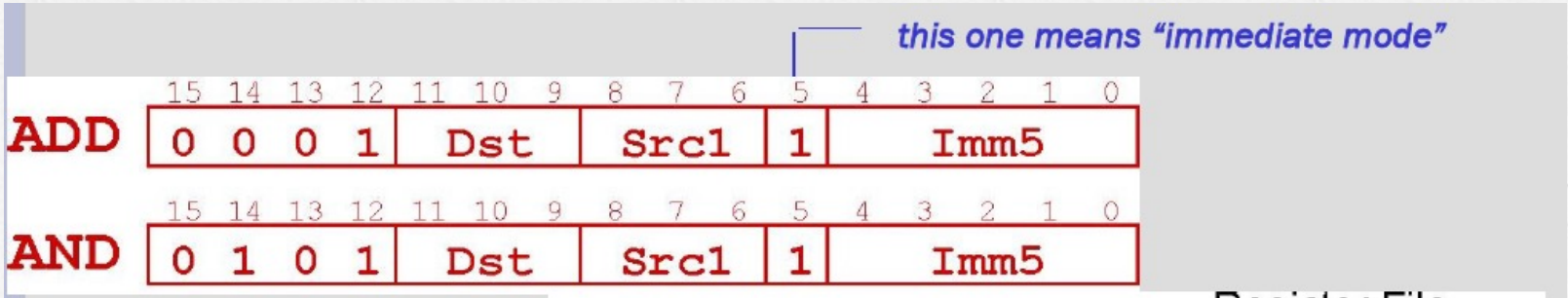
Note: Src and Dst could be the same register.



AND/ADD (Register)



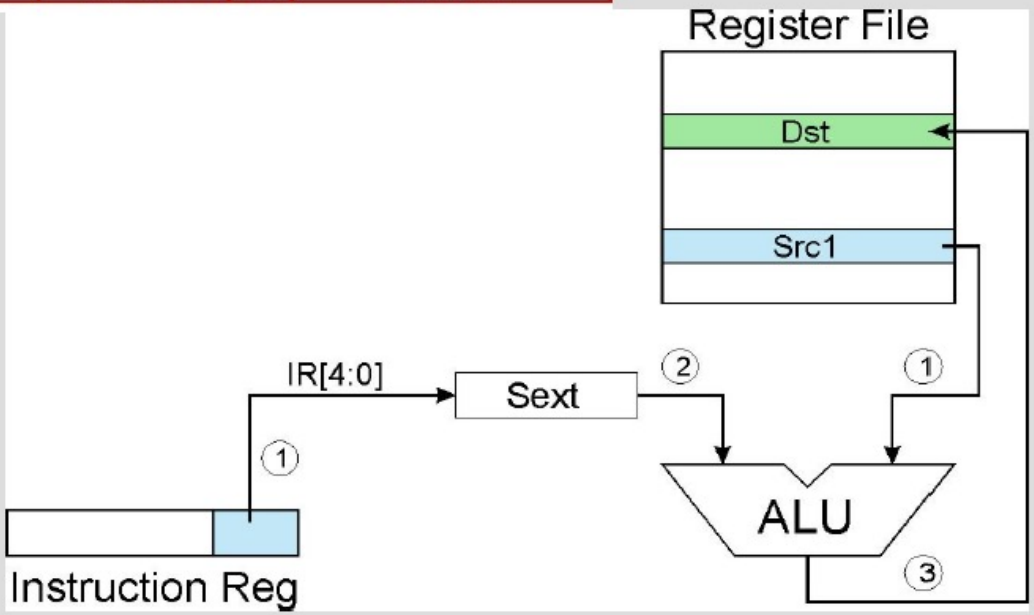
AND/ADD (Immediate)



- Immediate field Is sign-extended
- Example:

11110 will be sign-extended to

11111111111 11110 = - 2



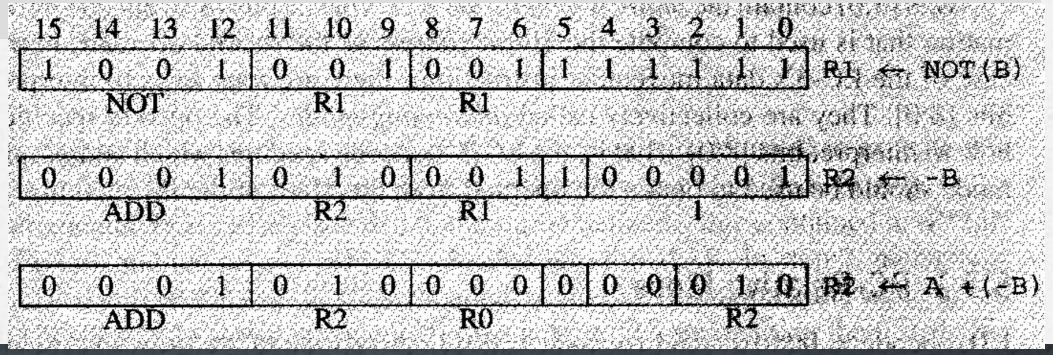
Using Operate Instructions

With only ADD, AND, NOT…

- How do we subtract? (Example 5.3)
- How do we OR?
- How do we copy from one register to another?
- How do we initialize a register to zero?

- what do the following instructions do?
  - 0001001100000101 or 0x1305
  - 0x1DA1

Example 5.3 →



Data Movement Instructions

Load -- read data from memory to register

- LD: direct mode
- LDR: base+offset mode
- LDI: indirect mode

Store -- write data from register to memory

- ST: direct mode
- STR: base+offset mode
- STI: indirect mode

Load effective address -- compute address, save in register

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

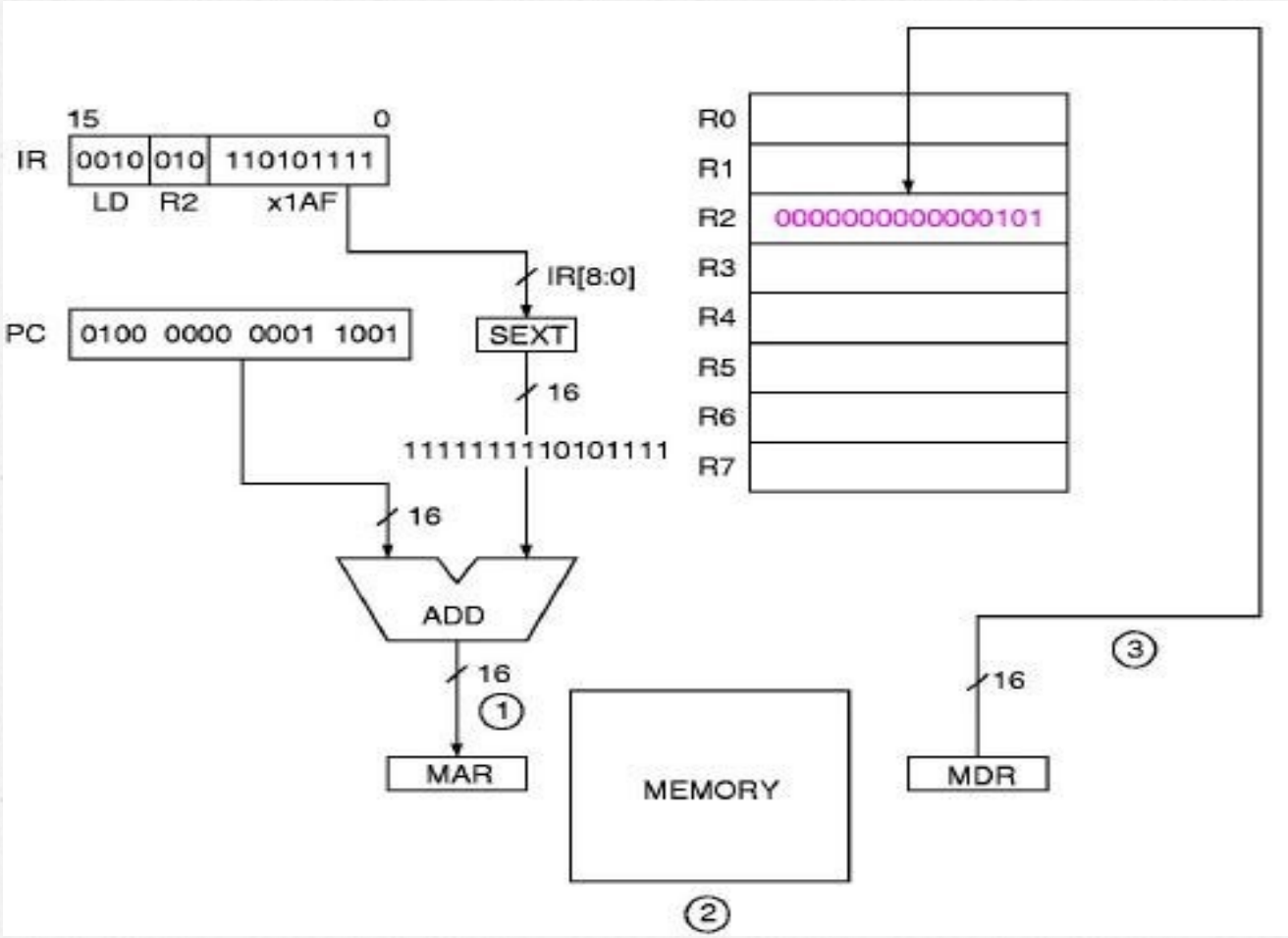
PC-Relative Mode

- Also called Direct Mode
- Specify address directly in the instruction
- Bits[8:0]: an offset relative to the PC (after fetch)
- Used by both LD and ST instructions

LD (Load)

- 0010 010 1101011111
- Opocode
  - Bits[15:12]: 0010 → LD
  - load data from memory into register
- Oprand
  - Destination register: Bits[11:9] → R2
  - Source data in the memory:
    - **effective address = (PC) + SEXT(IR[8:0])**
    - **operand location must be within approx. 256 locations of the instruction**

Data Path of LD



ST (Store)

- ST also uses PC-relative addressing mode
- Opocode
  - Bits[15:12]: 0011
  - store data from register into memory
- Oprand
  - source register: Bits[11:9]
  - Destination in the memory:
    - **effective address = (PC) + SEXT(IR[8:0])**
    - **operand location must be within approx. 256 locations of the instruction**

Base + Offset Mode

With direct mode, **operand location must be within approx. 256 locations of the instruction**

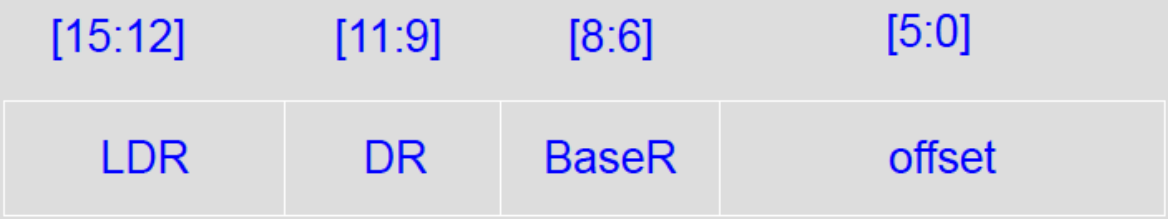
- What about the rest of memory?

Solution:

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

effective address = (BaseRegister) + offset

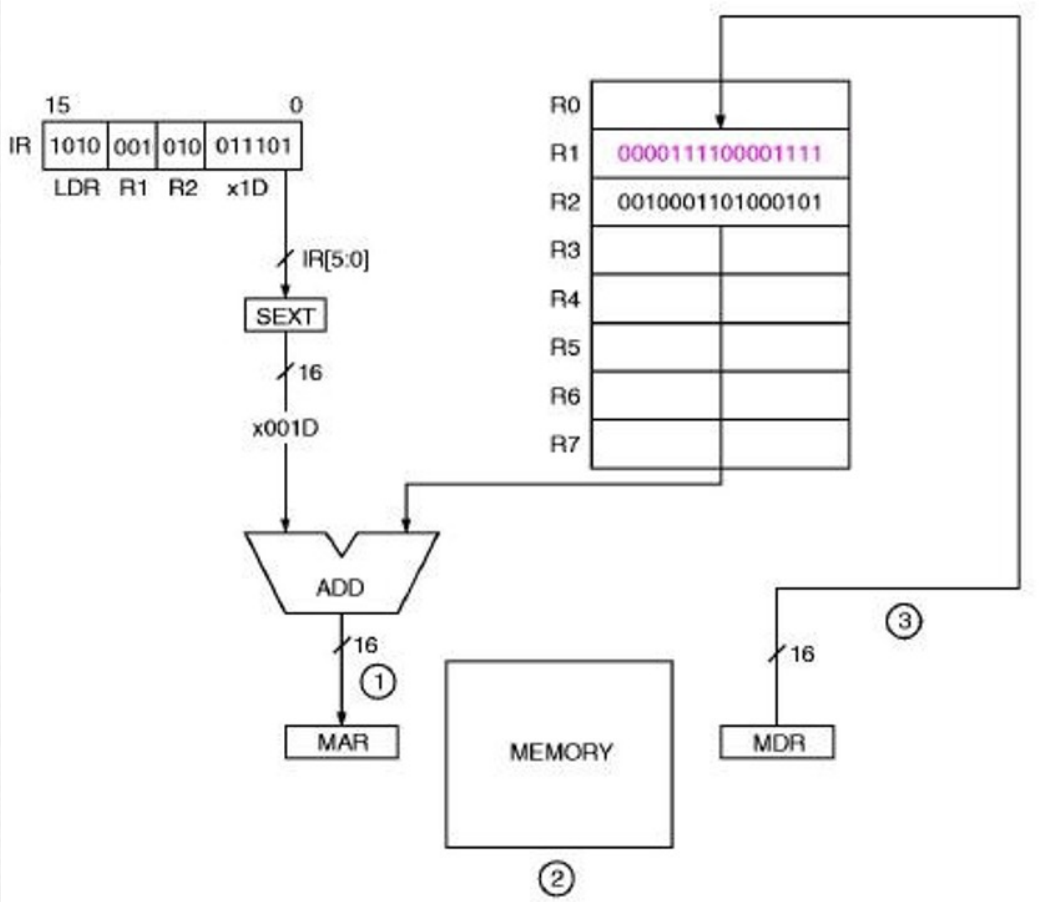
LDR (Base + Offset Mode)



- Opcode: 0110
- effective address = (BaseRegister) + offset
  - sign extend (SEXT) the 6 bit offset ([5:0]) to 16 bits
  - add it to the contents of the Base Register ([8:6])

- differences from Direct addressing (PC-Relative):
  - In base+offset, offset field is 6 bits; PC-Relative offset field is 9 bits
  - base+offset can address any location in memory, PC-Relative offset only within +/- 256 locations of the instruction.

LDR Data Path



STR (Base + Offset Mode)

- ST also uses Base+offset addressing mode
- Opcode
  - Bits[15:12]: 0111
- Operand
  - source register: Bits[11:9]
  - Destination in the memory:
    - **effective address = (BaseRegister) + offset**

Indirect Addressing Mode

- An address is first formed exactly the same way as with LD and ST
- This address contains the address of the operand.
- Just like base+offset, indirect mode can also address any location in the memory.
- Memory has to be accessed twice in order to get the address of the operand

LDI (Indirect Mode)

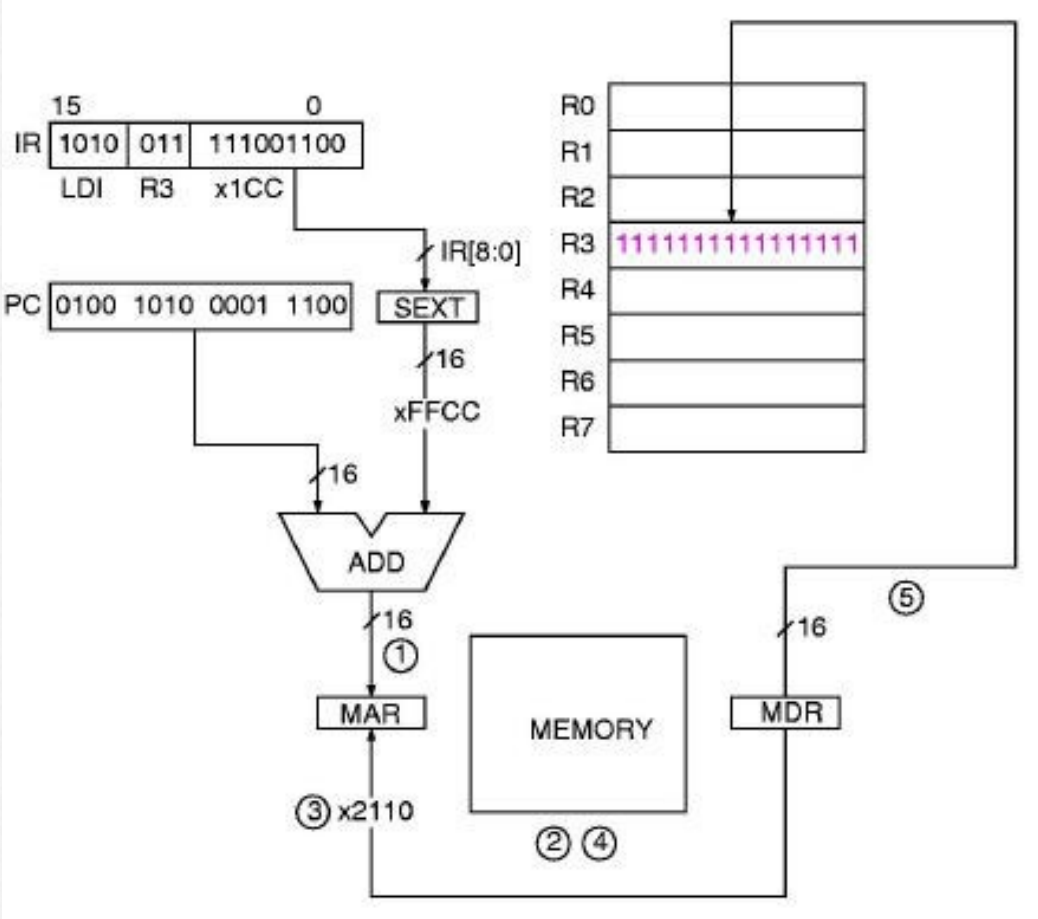
[15:12]	[11:9]	[8:0]
LDI	DR	Addr. Gen. bits

- **Opocode 1010**
- **Same initial mechanism as direct mode (i.e. PC-Relative), but the calculated memory location now contains the *address of the operand,* (i.e. the effective address is indirect):**

pointer address = (PC) + SEXT(IR[8:0])

effective address = Mem[(PC) + SEXT(IR[8:0])]

LDI Data Path



STI

STI also uses indirect addressing mode

Opocode 1 01 1

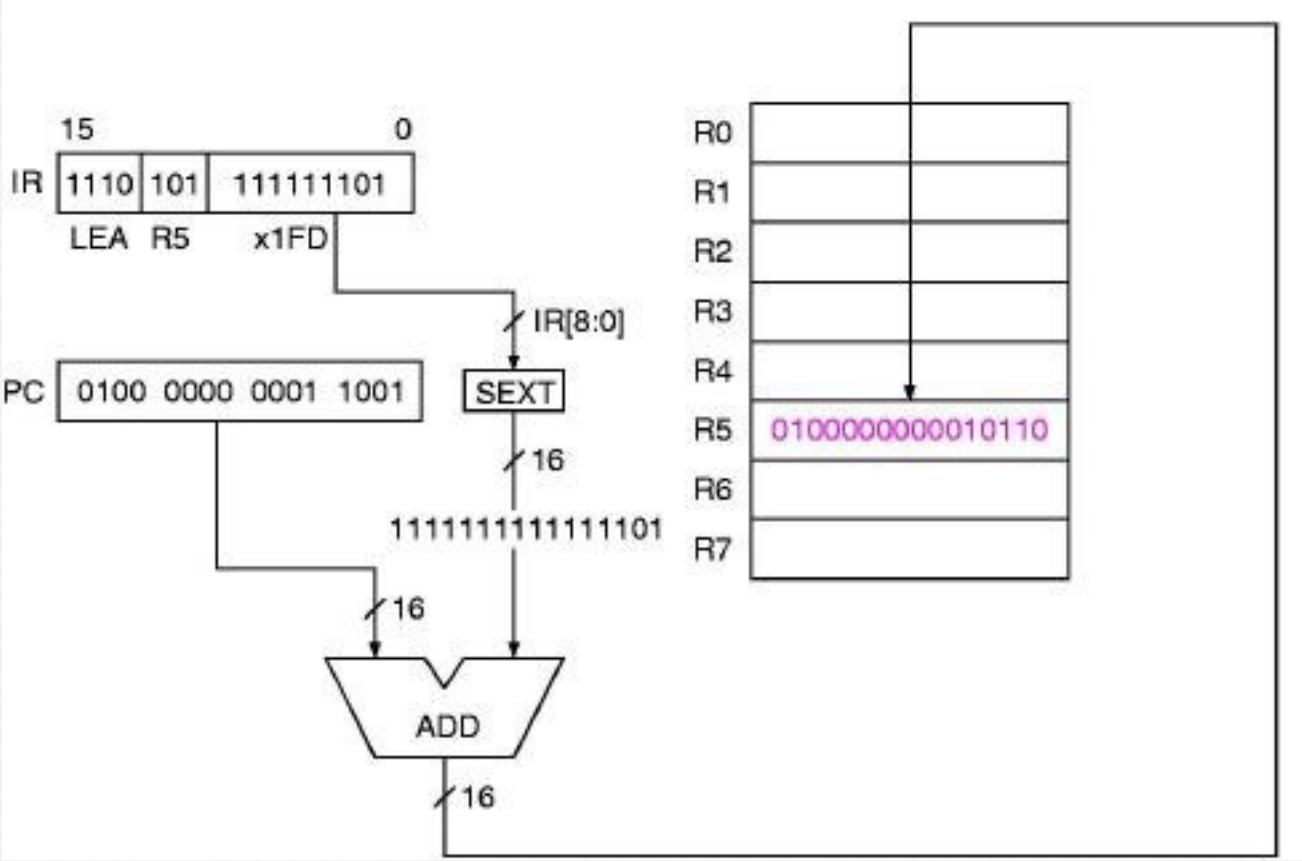
pointer address = (PC) + SEXT(IR[8:0])

effective address = Mem[(PC) + SEXT(IR[8:0])]

LEA (Immediate)

- LEA: Load Effective Address
- Opocode 1110
- Operand is obtained immediately, without accessing the memory

LEA Data Path



Addressing Mode Example

Address	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
x30F6	1	1	1	0	0	0	1	1	1	1	1	1	1	1	0	1	R1<- PC-3
x30F7	0	0	0	1	0	1	0	0	0	1	1	0	1	1	1	0	R2<- R1+14
x30F8	0	0	1	1	0	1	0	1	1	1	1	1	1	0	1	1	M[x30F4]<- R2
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
x30FB	0	1	1	1	0	1	0	0	0	1	0	0	1	1	1	0	M[R1+14]<- R2
x30FC	1	0	1	0	0	1	1	1	1	1	1	1	0	1	1	1	R3<- M[M[x3F04]]

Trace the code and figure out
the addressing mode
used by each instruction.

Control Instructions

- LC-3 has 5 control instructions
  - Conditional branch
  - Unconditional branch
  - The Trap instruction
- Change the Program Counter
  - BRx, JMP/RET, JSR/JSRR, TRAP, RTI
    - BRx uses PC-Relative addressing with 9-bit offset
    - JSR uses PC-Relative addressing with 11-bit offset
    - JMP/RET & JSRR use base+offset addressing with zero offset
    - we'll deal with TRAP & RTI later

Conditional Codes

- LC-3 has three condition code registers:
  - N -- negative
  - Z -- zero
  - P -- positive (greater than zero)

- Set by any instruction that stores 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

BR: Conditional Branch (if-then)

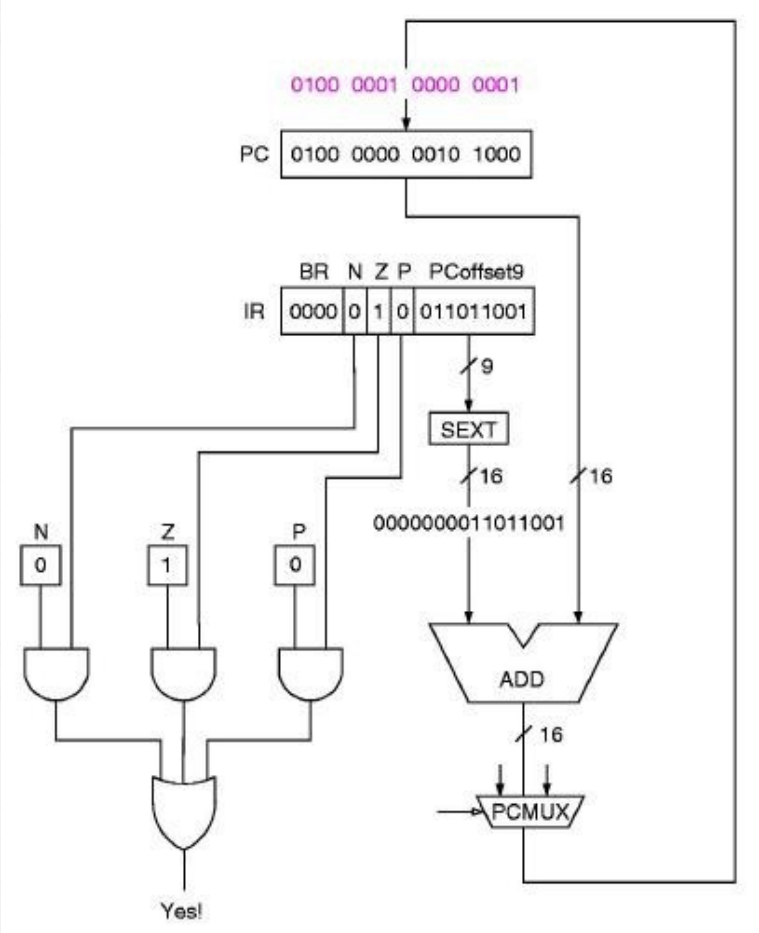
- 0000 010 011011001

- Bit[15:12] Opocode: 0000
- Bit[11]: N; bit[10]: Z; bit[9]: P
- Bit[8:0]: offset

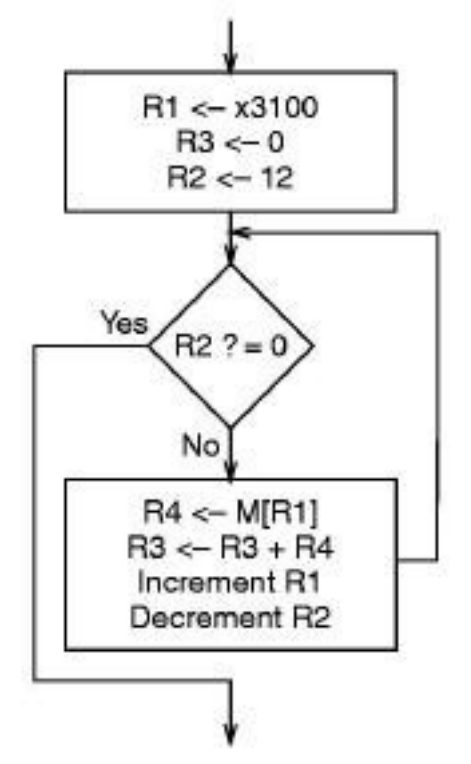
- When condition is satisfied, PC = PC + offset

- If none of the condition codes are examined, PC remains unchanged

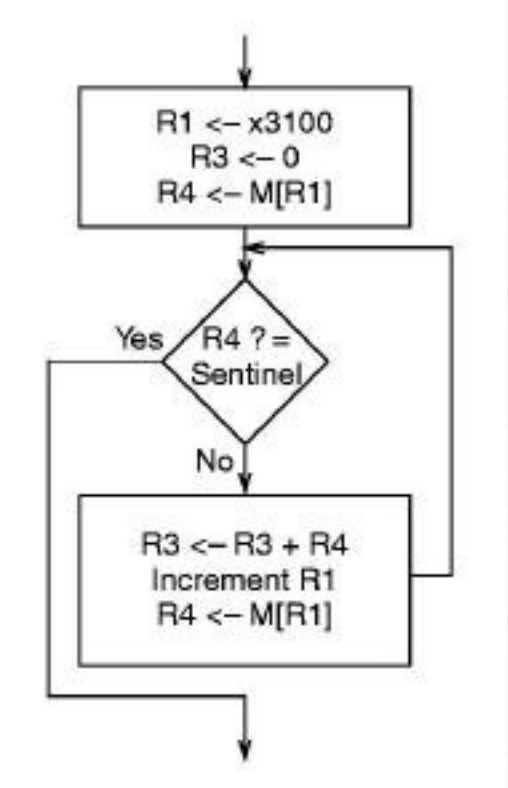
BR Data Path


0100 0001 0000 0001
PC
0100 0000 0010 1000
BR N Z P PCoffset9
IR
0000 0 1 0 011011001
9
SEXT
16
16
0000000011011001
N
0
Z
1
P
0
ADD
16
PCMUX
Yes!

Building Loops Using BR: Adding 12 integers



Counter control



Sentinel control

JMP (**jump** or **goto**)

1100 000 BaseR 00 0000

- Take the next instruction from the address stored in BaseR

- Example: 1100 000 101 000000

  if (R5) = x3500, the address x3500 is written to the PC

TRAP: Invoke a system routine

Calls a service routine, identified by 8-bit "trap vector."

vector	*routine*
x23	input a character from the keyboard
x21	output a character to the monitor
x25	halt the program

When routine is done,
PC is set to the instruction following TRAP.

(We'll talk about how this works later.)

Example 1

A program for adding 12 integers

Address	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
x3000	1	1	1	0	0	0	1	0	1	1	1	1	1	1	1	1	R1<- 3100
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	BRz x300A
x3005	0	1	1	0	1	0	0	0	0	1	0	0	0	0	0	0	R4 <- M[R1]
x3006	0	0	0	1	0	1	1	0	1	1	0	0	0	1	0	0	R3 <- R3+R4
x3007	0	0	0	1	0	0	1	0	0	1	1	0	0	0	0	1	R1 <- R1+1
x3008	0	0	0	1	0	1	0	0	1	0	1	1	1	1	1	1	R2 <- R2-1
x3009	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	1	BRnzp x3004

3 A program that implements the algorithm of Figure 5.12

Example 2

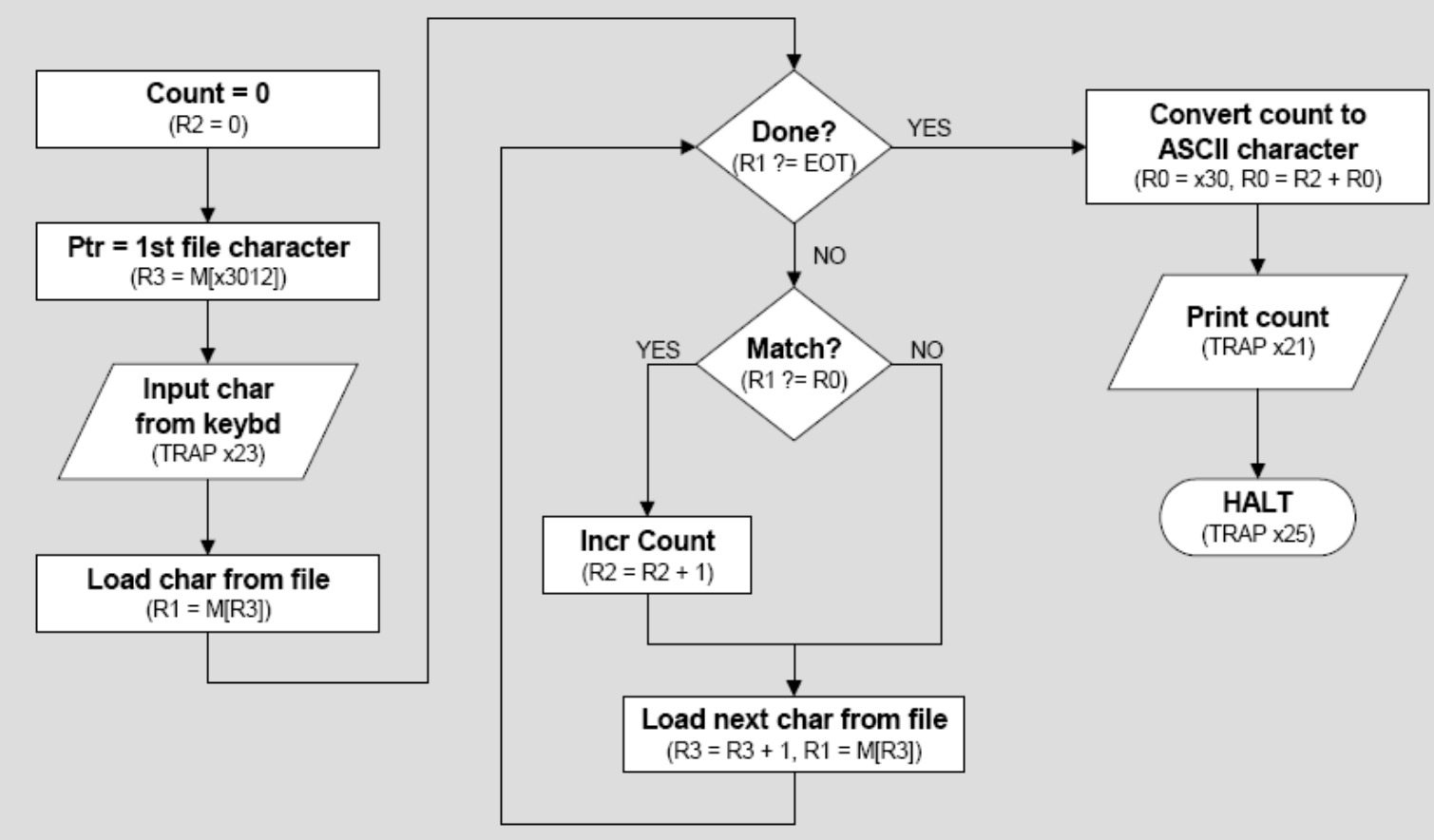
Count the occurrences of a character in a file

- Program begins at location x3000
- Read character from keyboard
- Load each character from a "file"
  - File is a sequence of memory locations
  - Starting address of file is stored in the memory location immediately after the program
- If file character equals input character, increment counter
- End of file is indicated by a special ASCII value: EOT (x04)
- At the end, print the number of characters and halt (assume there will be less than 10 occurrences of the character)

A special character used to indicate the end of a sequence is often called a sentinel.

- Useful when you don't know ahead of time how many times to execute a loop.

Flow Chart


Count = 0
(R2 = 0)
Ptr = 1st file character
(R3 = M[x3012])
Input char
from keybd
(TRAP x23)
Load char from file
(R1 = M[R3])
Done?
(R1 ?= EOT)
YES
NO
Match?
(R1 ?= R0)
YES
NO
Incr Count
(R2 = R2 + 1)
Load next char from file
(R3 = R3 + 1, R1 = M[R3])
Convert count to
ASCII character
(R0 = x30, R0 = R2 + R0)
Print count
(TRAP x21)
HALT
(TRAP x25)

Address	*Instruction*	*Comments*
x3000	0101 010 010 1 00000	*R2 ← 0 (counter)*
x3001	0010 011 0000010000	*R3 ← M[x3012] (ptr)*
x3002	1111 0000 00100011	*Input to R0 (TRAP x23)*
x3003	0110 001 011 000000	*R1 ← M[R3]*
x3004	0001 100 001 1 11100	*R4 ← R1 – 4 (EOT)*
x3005	0000 010 0000001000	*If Z, goto x300E*
x3006	1001 001 001 1 11111	*R1 ← NOT R1*
x3007	0001 001 001 1 00001	*R1 ← R1 + 1*
X3008	0001 001 001 0 00000	*R1 ← R1 + R0*
x3009	0000 101 0000000001	*If N or P, goto x300B*

Address	*Instruction*	*Comments*
x300A	0 0 0 1 0 1 0 0 1 0 1 0 0 0 0 1	*R2 ← R2 + 1*
x300B	0 0 0 1 0 1 1 0 1 1 1 0 0 0 0 1	*R3 ← R3 + 1*
x300C	0 1 1 0 0 0 1 0 1 1 0 0 0 0 0 0	*R1 ← M[R3]*
x300D	0 0 0 0 1 1 1 1 1 1 1 1 0 1 1 0	*Goto x3004*
x300E	0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0	*R0 ← M[x3013]*
x300F	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0	*R0 ← R0 + R2*
x3010	1 1 1 1 0 0 0 0 0 0 0 1 0 0 0 0 1	*Print R0 (TRAP x21)*
x3011	1 1 1 1 0 0 0 0 0 0 0 1 0 0 1 0 1	*HALT (TRAP x25)*
X3012	Starting Address of File	
x3013	0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0	*ASCII x30 ('0')*

What's Next: Assembly Language