LC-3 ISA - I

Lecture Topics

LC-3 Instruction Set Architecture

LC-3 operate instructions

LC-3 data movement instructions

Lecture materials

Textbook § 5.1 - 5.3

Textbook Appendix A.1 - A.3

Homework

HW2 due Thursday February 17 at 5pm in the ECE 190 drop-off box

HW3 due Wednesday February 23 at 5pm in the ECE 190 drop-off box

Machine problem

MP1.2 due Thursday February 17 at 5pm submitted electronically

Announcements

LC-3 ISA

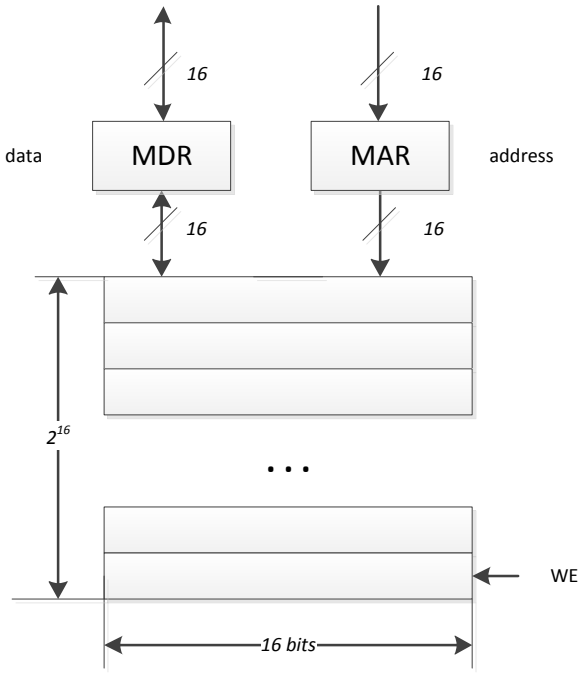
ISA's role

- ISA specifies all the information about the computer that the software has to be aware of
- ISA defines an interface for the programmer that he needs to know when writing programs in the computer's own machine language

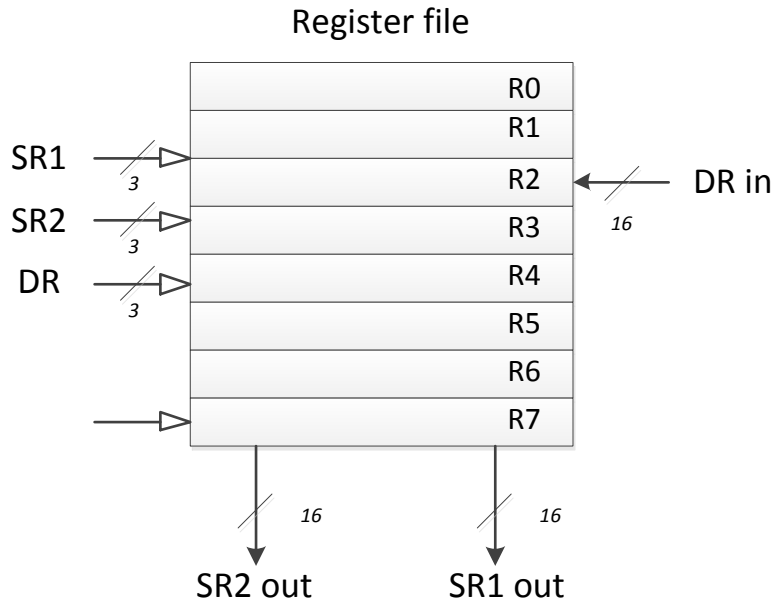
- ISA specifies the following
  - Memory organization
  - Resister set
  - Instruction set
  - Data types
  - Addressing models

LC-3 ISA overview

- Memory organization
  - 16-bit addressable
  - 2^{16} address space



- Register set
  - 8 16-bit general-purpose registers, named R0-R7

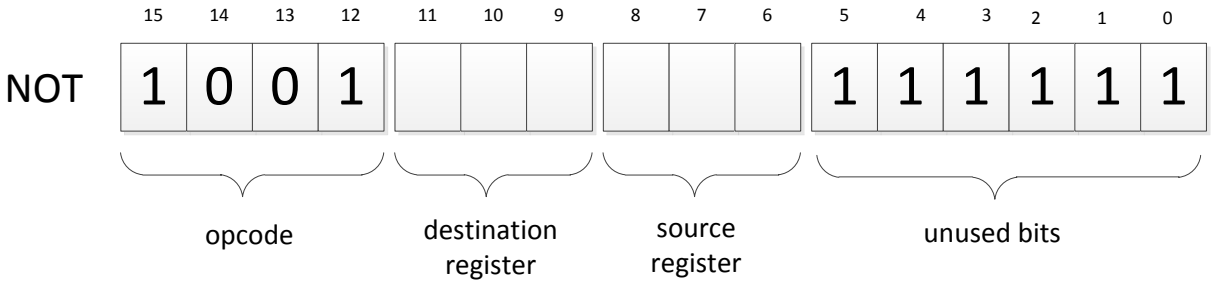


  - Other special-purpose registers, such as PC and IR
- Data types
  - 16-bit 2's complement integers
- Addressing models
  - A mechanism for specifying where the operands are located
  - Non-memory addresses
    - immediate (part of the instruction)
    - register
  - Memory addresses
    - PC-relative, base+offset, indirect
- The instruction set
  - 16-bit instructions
  - Bits 12-15 of the 16-bit instruction are used to specify the opcode
  - Operate instructions: ADD, AND, NOT
  - Data movement instructions: LD, LDI, LDR, LEA, ST, STR, STI
  - Control instructions: BR, JSR/JSSR, JMP, RTI, TRAP
- Condition codes
  - LC-3 has 3 single-bit registers that are set to 0 or 1 each time one of the general-purpose registers (R0-R7) is **written to**
    - N, Z, P – condition codes
    - N=1 (Z=P=0) when the value stored in one of the registers is negative
    - P=1 (Z=N=0) when the value stored in one of the registers is positive
    - Z=1 (N=P=0) when the value stored in one of the registers is zero

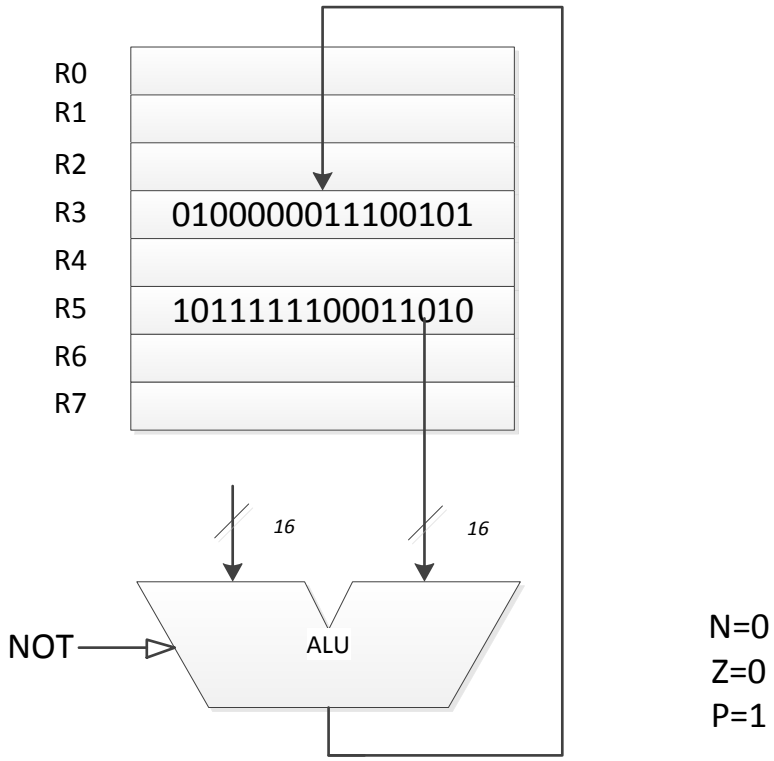
LC-3 operate instructions

NOT

- Operation: bitwise complement of the value from the source register (SR) is stored in the destination register (DR)
  - DR ← NOT(SR)
  - setcc (modifies condition codes)
- Encoding:

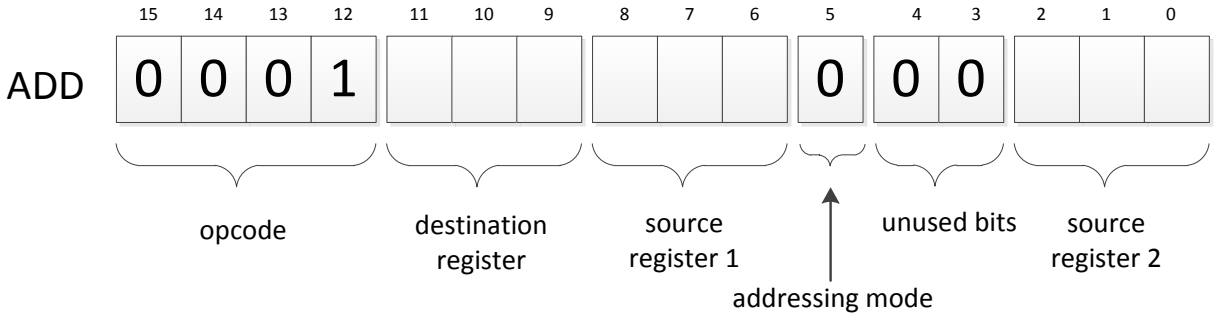


- Datapath relevant to the execution of this instruction:
  - Example: **1001 011 101 111111; NOT R3, R5**

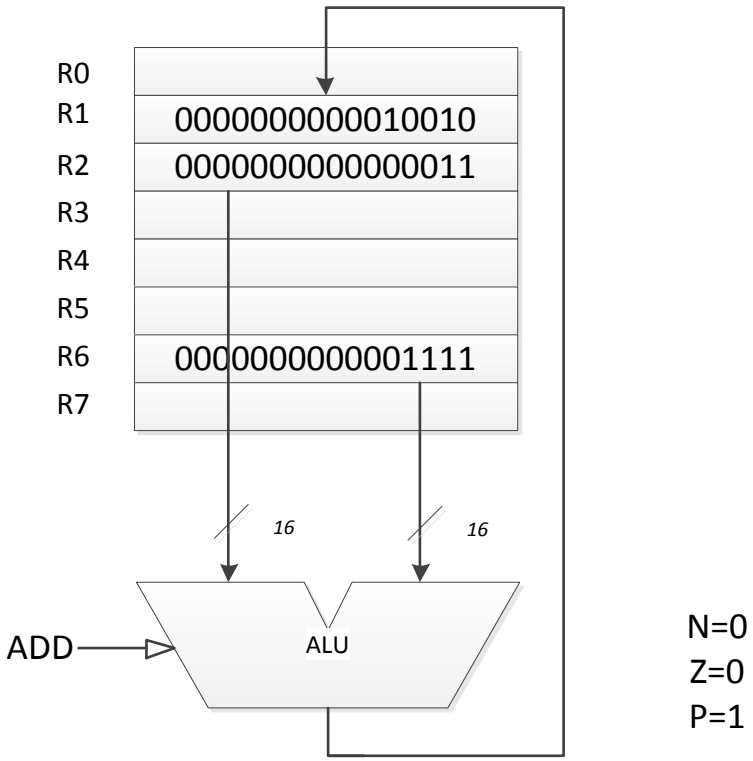


ADD (register mode)

- Operation: values from two source registers (SR1 and SR2) are added together (using 2's complement addition) and the resulting value is stored in the destination register (DR)
  - DR ← SR1 + SR2
  - setcc (modifies condition codes)
- Encoding:

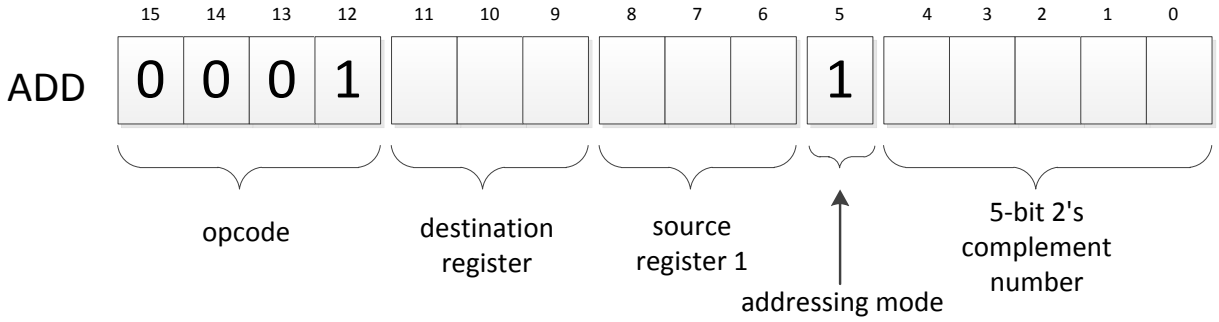


- Datapath relevant to the execution of this instruction:
  - Example: **0001 001 010 0 00 110; ADD R1, R2, R6**

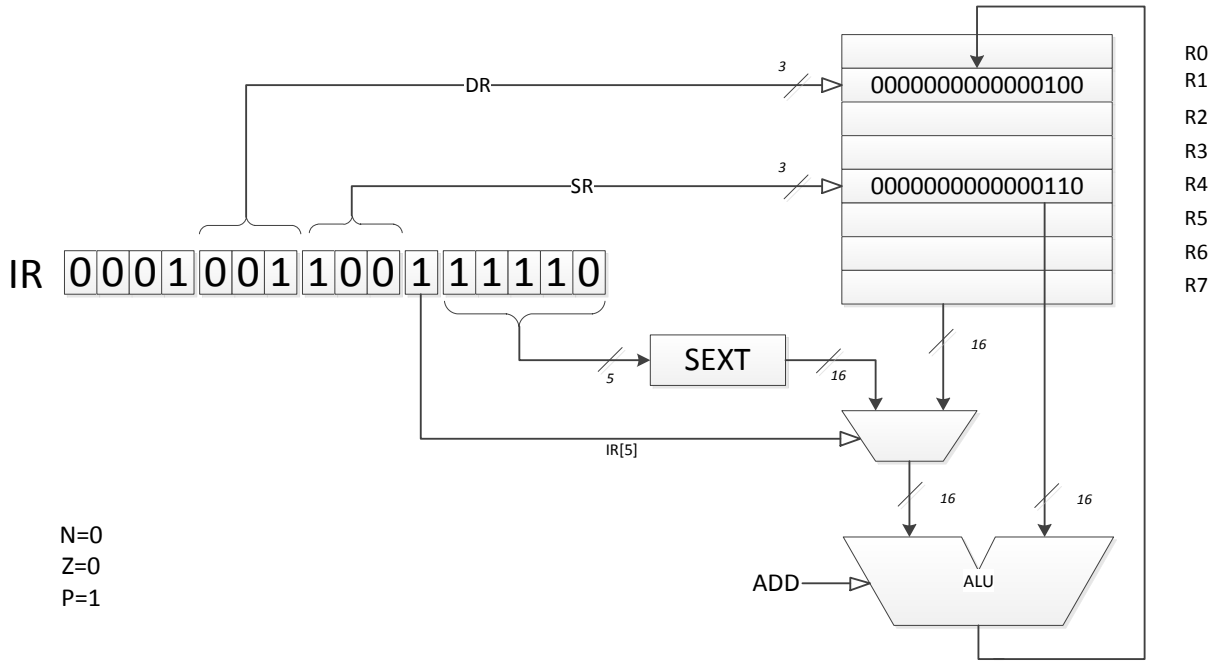


ADD (immediate mode)

- Operation: value from one source register (SR1) is added to the sign-extended imm5 filed (last 5 bits of the instruction) and the resulting value is stored in the destination register (DR)
  - DR ← SR1 + SEXT(imm5)
  - setcc (modifies condition codes)
- Encoding:



- Datapath relevant to the execution of this instruction:
  - Example: **0001 001 100 1 11110; ADD R1, R4, #-2**



- imm5 field is 5-bit long
- in 2's complement, it is sufficient to represent numbers in the interval from -2^4 to 2^4-1 (-16 .. 15)

Few examples with ADD instruction

- Increment value by 1
  - 0001 110 110 **1 00001**; ADD R6, R6, #1
- Decrement value by 1:
  - 0001 001 001 **1 11111**; ADD R6, R6, #-1
- Copy value from one register to another
  - 0001 110 001 1 00000; ADD R6, R1, #0
- Subtract two numbers using 2's complement representation
  - We want to compute A= B – C
  - But there is no subtraction instruction in LC-3 ISA!
  - We can replace subtraction with addition: A = B + (-C)
    - Additive inverse of C can be computed using NOT and ADD instructions
  - The subtraction procedure is
    - *R1 ← B (load value of B to register 1)*
    - *R2 ← C (load value of C to register 2)*
    - R2 ← NOT(R2) (compute bitwise complement of 2)
    - R2 ← R2 + 1 (add one to get -C)
    - R3← R1 + R2 [A = B + (-C)]
  - The program (in LC-3 machine language!):

```
1001 010 010 111111          ; R2 ← NOT(R2)
0001 010 010 1 00001         ; R2 ← R2 + 1
0001 011 001 0 00 010        ; R3← R1 + R2
```

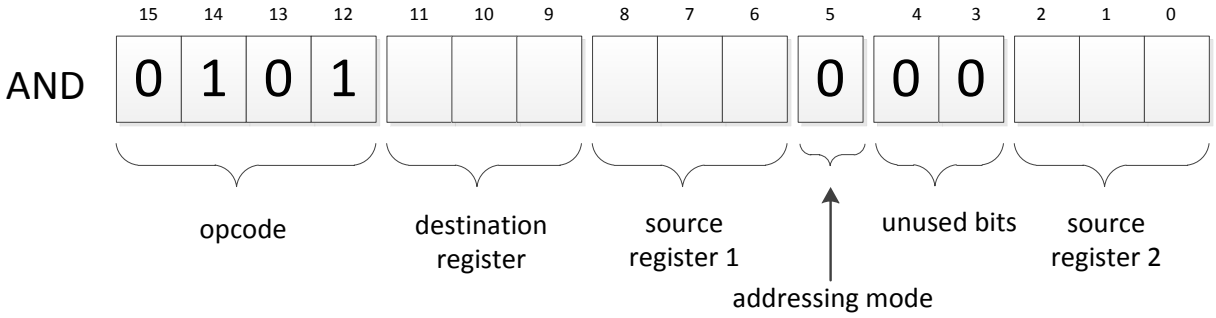
  - There is one fundamental flaw with this program: value stored in register 2 will be modified in the process of program execution (what if we need value of C later on?)
    - A better way is to store intermediate value in another (unused) register

About notation

- Machine code looks like this: 1001010010111111
  - For our convenience, ON PAPER we can separate the 16 bits that make up an instruction into smaller groups based on their meaning, e.g.: 1001 010 010 111111
- We can also add a comment to a line of binary code using semicolon to separate it from the actual instruction, e.g., 1001010010111111; *some comment goes here*
- We can use three styles of comments
  - Words, e.g.: *add numbers from R1 and R2 and place the results to R3*
  - Register transfer style, e.g.: R3← R1 + R2
  - Assembly code style, e.g.: ADD R3, R1, R2
- Whichever style you decide to use, just be consistent!

AND (register mode)

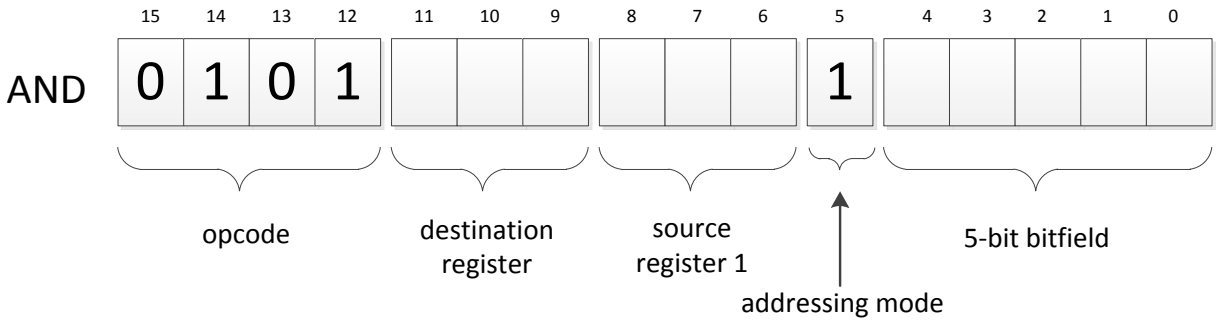
- Operation: values from two source registers (SR1 and SR2) are bitwise AND'ed together and the resulting value is stored in the destination register (DR)
  - DR ← SR1 AND SR2
  - setcc (modifies condition codes)
- Encoding:



- Datapath relevant to the execution of this instruction is identical to the register mode ADD instruction, the only difference is the select signal to the ALU
- Example: 0101 001 010 0 00 111; AND R1, R2, R7

AND (immediate mode)

- Operation: value from one source register (SR1) is AND'ed with the sign-extended imm5 filed (last 5 bits of the instruction) and the resulting value is stored in the destination register (DR)
  - DR ← SR1 AND SEXT(imm5)
  - setcc (modifies condition codes)
- Encoding:



- Datapath relevant to the execution of this instruction is identical to the register mode ADD instruction, the only difference is the select signal to the ALU
- Example: 0101 001 010 1 00001; AND R1, R2, #1

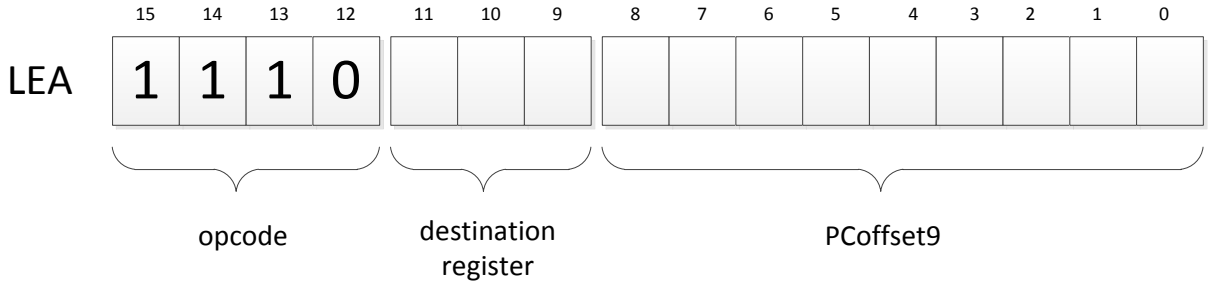
Few examples with AND instruction

- Clear register
  - 0101 110 110 **1 00000**; R6 ← R6 AND SEXT(00000)
- Copy value from one register to another
  - 0101 110 000 1 11111; R6 ← R1 AND SEXT(11111)
- Implement bitwise OR operation
  - We want to compute A= B OR C
  - But there is no OR instruction in LC-3 ISA!
  - Remember DeMorgan's Law: B OR C = NOT(NOT(B) AND NOT(C))
  - The procedure is
    - *R1 ← B (load value of B to register 1)*
    - *R2 ← C (load value of C to register 2)*
    - R1 ← NOT(R1)
    - R2 ← NOT(R2)
    - R3 ← R1 AND R2
    - R3← NOT(R3)

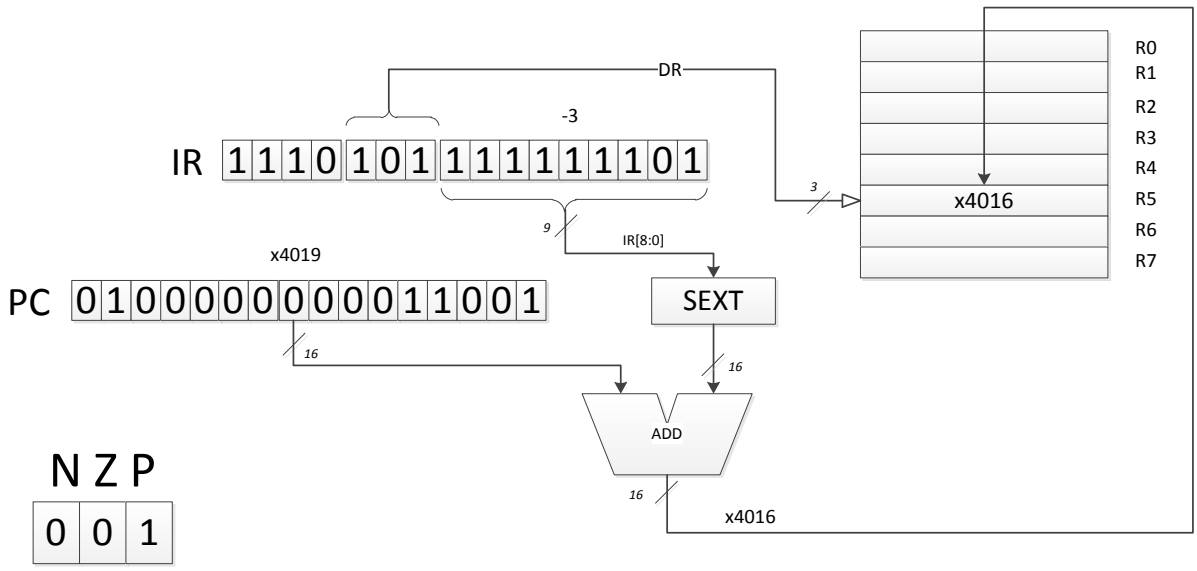
LC-3 data movement instructions

Load effective address instruction (LEA)

- Operation: load the destination register (DR) with the value formed by adding values from PC register and the sign-extended last 9 bits of the instruction (PCoffset9)
  - DR ← PC+SEXT(PCoffset9)
  - setcc
- Implements *immediate* addressing mode
  - Operand to be loaded to destination register is obtained *immediately*, without any access to memory
- Encoding:



- Datapath relevant to the execution of this instruction:
  - Example: **1110 101 111111101; LEA R5, *offset***



- R5 ← PC + SEXT(IR[8:0])

- Useful for loading an address of memory to be later on used with LDR/STR instructions