LC-3 ISA - II

Lecture Topics

LC-3 data movement instructions

LC-3 control instructions

LC-3 data path review

Lecture materials

Textbook § 5.3 - 5.6

Textbook Appendix A.3

Homework

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

Machine problem

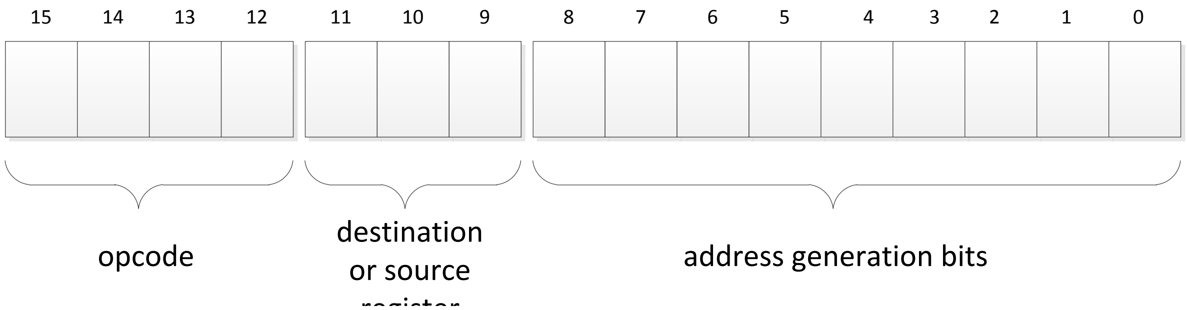
MP2 due March 2, 2011 at 5pm submitted electronically.

Announcements

LC-3 data movement instructions

Overview

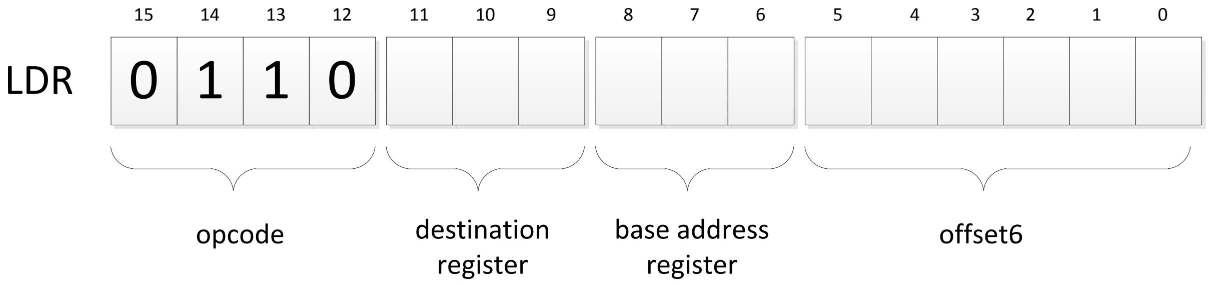
- Load: move data from memory to register
    - LD, LDI, LDR
    - LEA – immediate mode load instruction
- Store:
    - ST, STR, STI
- Load/store instruction format



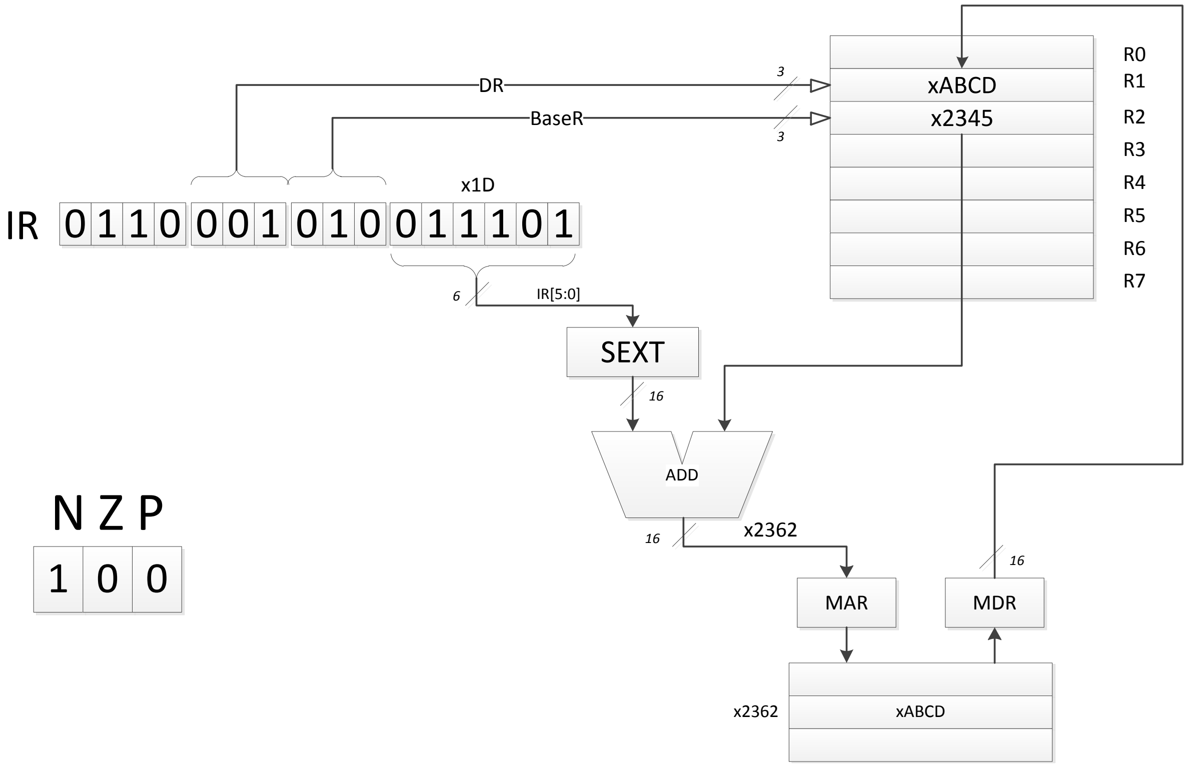
- 4 ways to interpret address generation bits (4 addressing modes)
    - PC-relevant mode (LD and ST instructions)
    - Indirect mode (LDI and STI instructions)
    - Base + offset mode (LDR and STR instructions)
    - Immediate mode (LEA instruction)

Load instruction using *base + offset* addressing mode (LDR)

- Operation: the content of memory at the address computed as the sum of the address stored in the base address register (BaseR) and the sign-extended last 6 bits of the instruction (offset6) is loaded into the destination register (DR)
    - DR ← mem[BaseR+SEXT(offset6)]
    - setcc
- Encoding:



- Datapath relevant to the execution of this instruction:
    - Example: **0110 001 010 011101; LDR R1, R2, *offset***



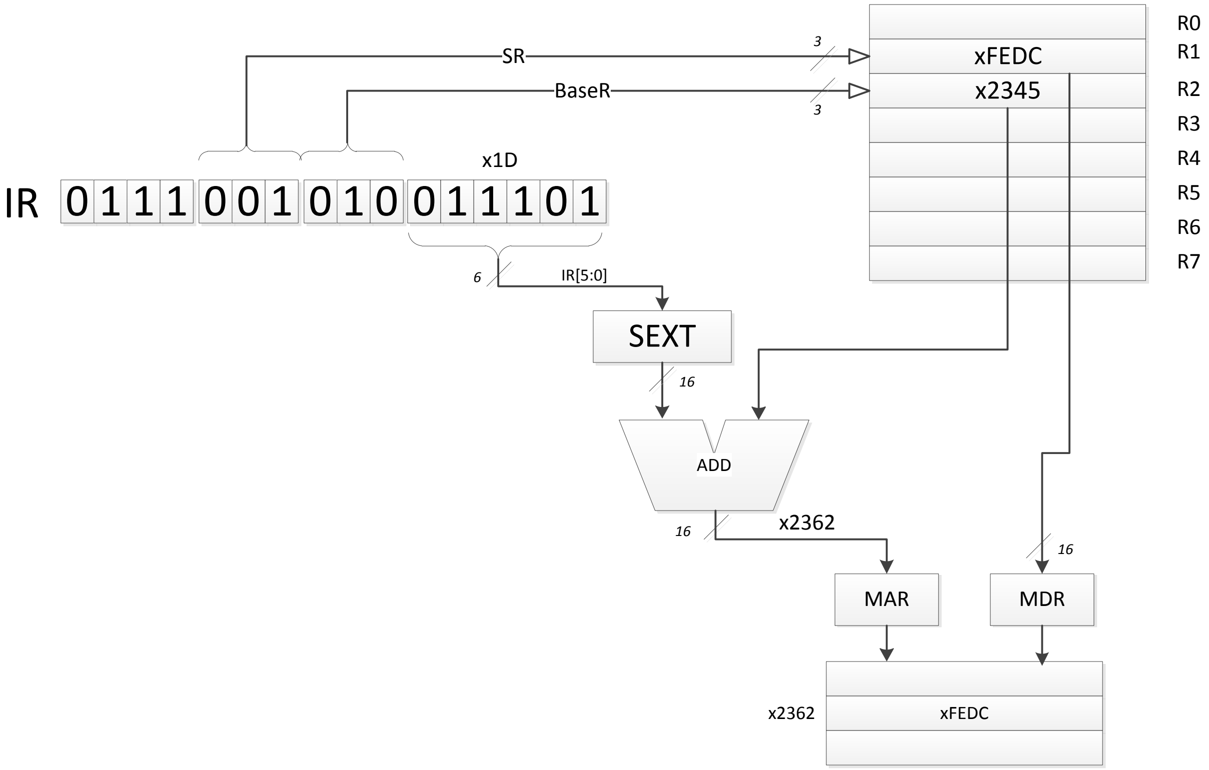
  - MAR ← R2 + SEXT(IR[5:0])
  - MDR ← mem[MAR]
  - R1 ← MDR
- offset6 field is 6-bit wide, thus the offset can be from -32 to +31

Store instruction using *base + offset* addressing mode (STR)

- Operation: value stored in the source register (SR) is transferred to the memory at the address computed as the sum of the address stored in the base address register (BaseR) and the sign-extended last 6 bits of the instruction (offset6)
  - mem[BaseR+SEXT(offset9)] ← SR
- Encoding:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STR	0	1	1	1												
	opcode				source register			base address register			offset6					

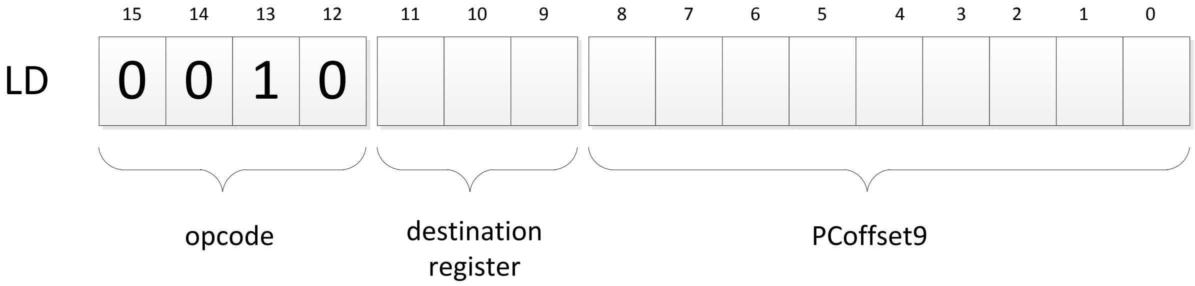
- Datapath relevant to the execution of this instruction:
  - Example: **0111 001 010 011101; STR R1, R2, *offset***



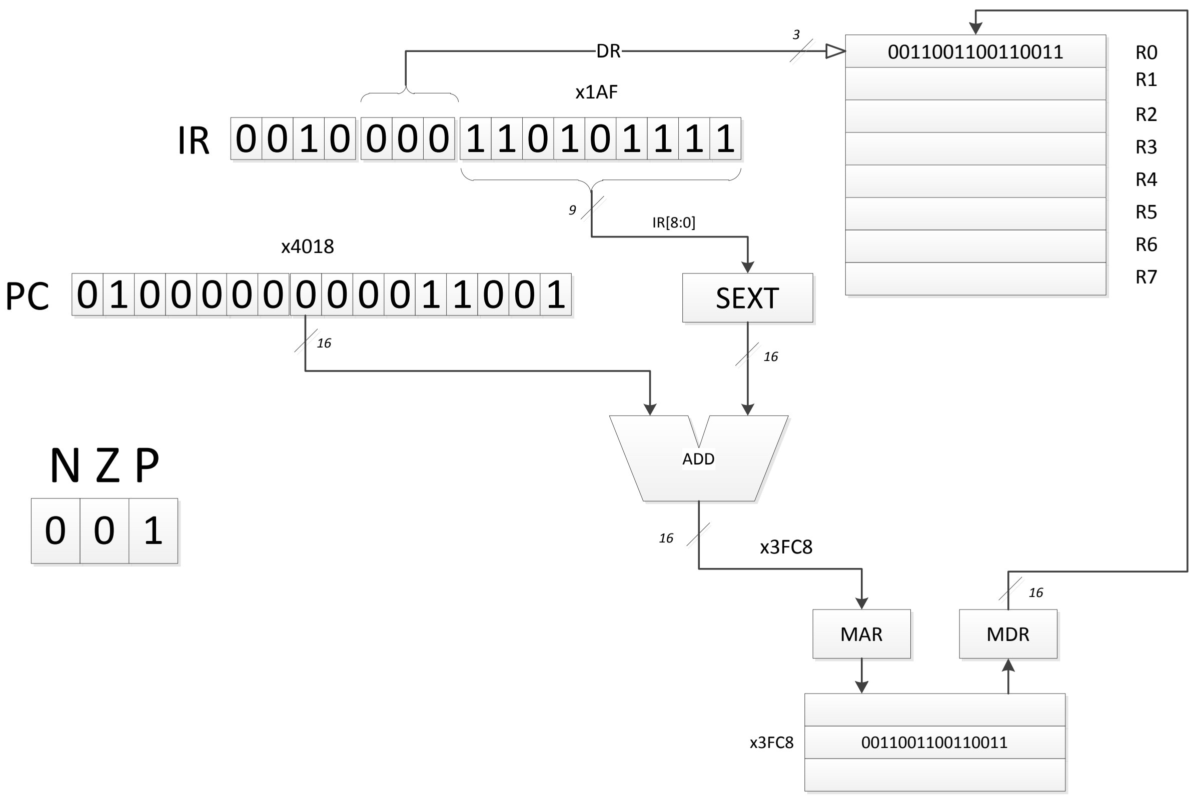
- MAR ← R2 + SEXT(IR[5:0])
- MDR ← R1
- mem[MAR] ← MDR

Load instruction using PC relative addressing mode (LD)

- Operation: the content of memory at the address computed as the sum of the address stored in PC register and the sign-extended last 9 bits of the instruction (PCoffset9) is loaded into the destination register (DR)
  - DR ← mem[PC+SEXT(PCoffset9)]
  - setcc
- Encoding:



- Datapath relevant to the execution of this instruction:
  - Example: **0010 000 110101111; LD R0, *offset***



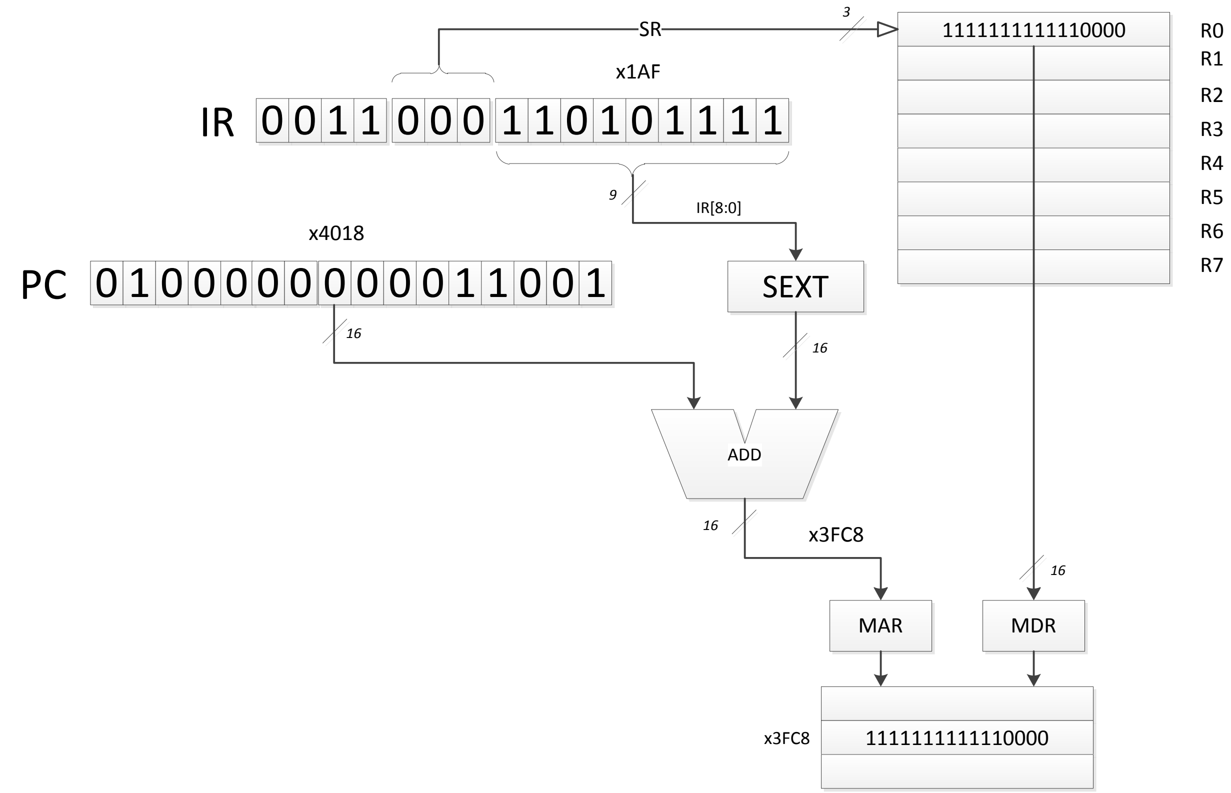
- MAR ← PC + SEXT(IR[8:0])
- MDR ← mem[MAR]
- R0 ← MDR

Store instruction using PC relative addressing mode (ST)

- Operation: value stored in the source register (SR) is transferred to the memory at the address computed as the sum of the address stored in PC register and the sign-extended last 9 bits of the instruction (PCoffset9)
  - mem[PC+SEXT(PCoffset9)] ← SR
- Encoding:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ST	0	0	1	1												
	opcode				source register			PCoffset9								

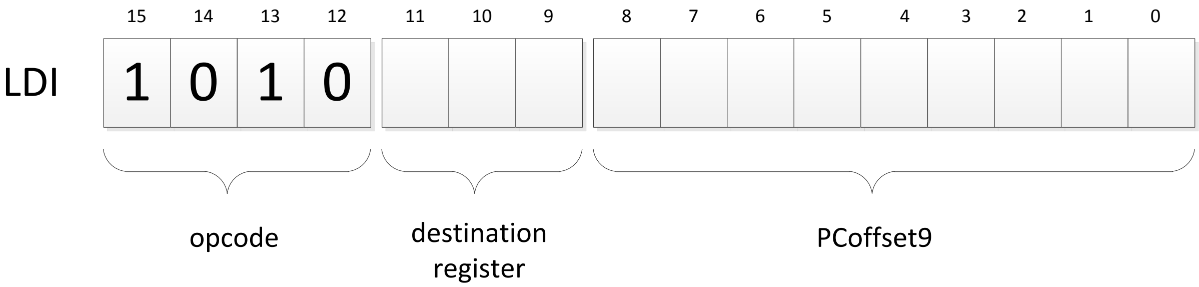
- Datapath relevant to the execution of this instruction:
  - Example: **0011 000 110101111; ST R0, *offset***



  - MAR ← PC + SEXT(IR[8:0])
  - MDR ← R0
  - mem[MAR] ← MDR
- PCoffset9 field is 9-bit wide, thus the offset can be from -256 to +255

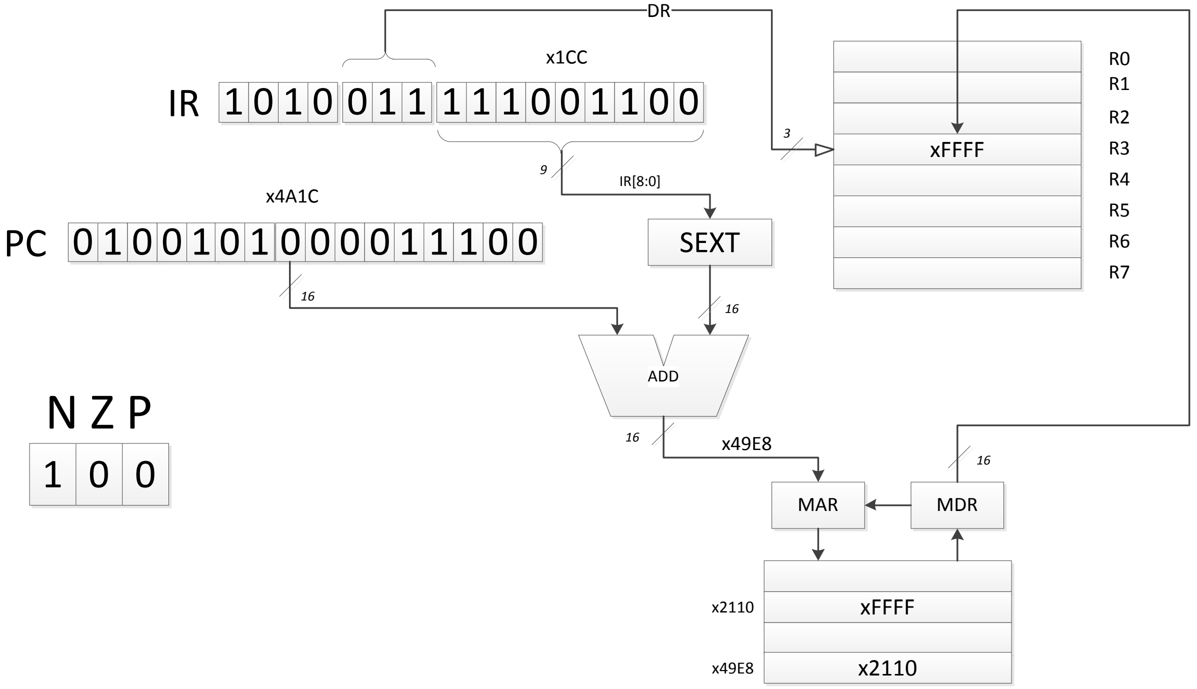
Load instruction using indirect addressing mode (LDI)

- Operation: the content of memory at the address stored in memory at the address computed as the sum of the address stored in PC register and the sign-extended last 9 bits of the instruction (PCoffset9) is loaded into the destination register (DR)
  - DR ← mem[mem[PC+SEXT(PCoffset9)]]
  - setcc
- Encoding:



- Datapath relevant to the execution of this instruction:

- Example: **1010 011 111001100; LDI R3, *offset***



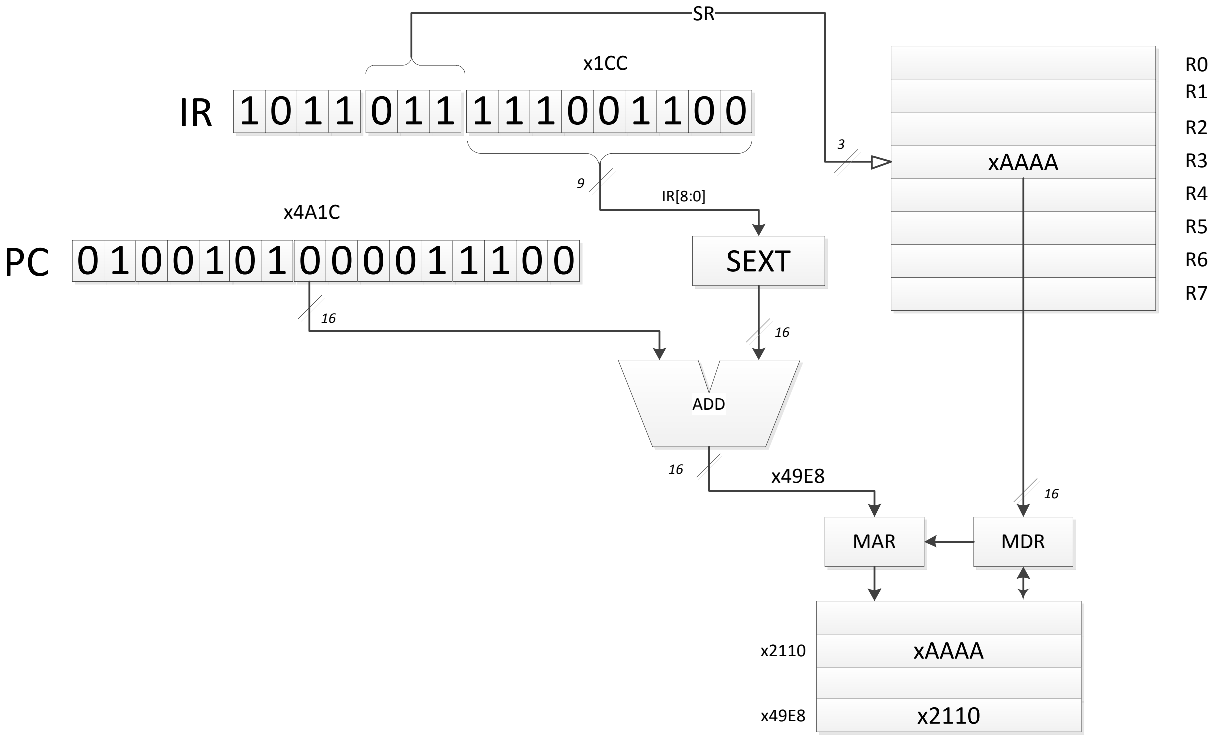
  - MAR ← PC + SEXT(IR[8:0])
  - MDR ← mem[MAR]
  - MAR ← MDR
  - MDR ← mem[MAR]
  - R3 ← MDR

Store instruction using indirect addressing mode (STI)

- Operation: value from the source register (SR) is transferred to the memory at the address stored in memory at the address computed as the sum of the address stored in PC register and the sign-extended last 9 bits of the instruction (PCoffset9)
  - mem[mem[PC+SEXT(PCoffset9)]] ← SR
- Encoding:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STI	1	0	1	1												
	opcode				source register			PCoffset9								

- Datapath relevant to the execution of this instruction:
  - Example: **1011 011 111001100; STI R3, *offset***



- MAR ← PC + SEXT(IR[8:0])
- MDR ← mem[MAR]
- MAR ← MDR
- MDR ← R3
- mem[MAR] ← MDR

LC-3 control instructions

- Control instructions change the sequence of the instructions that are executed. The achieve this by directly manipulating the value of PC register.
- Control instructions supported by LC-3
  - **JMP - unconditional jump**
  - **BR – conditional branch**
  - **TRAP – invokes an OS service call**
  - JSR/JSRR – subroutine call
  - RET – return from subroutine
  - RTI – return from interrupt
- For now we will study only JMP, BR, and TRAP instructions

Conditional branches (BR)

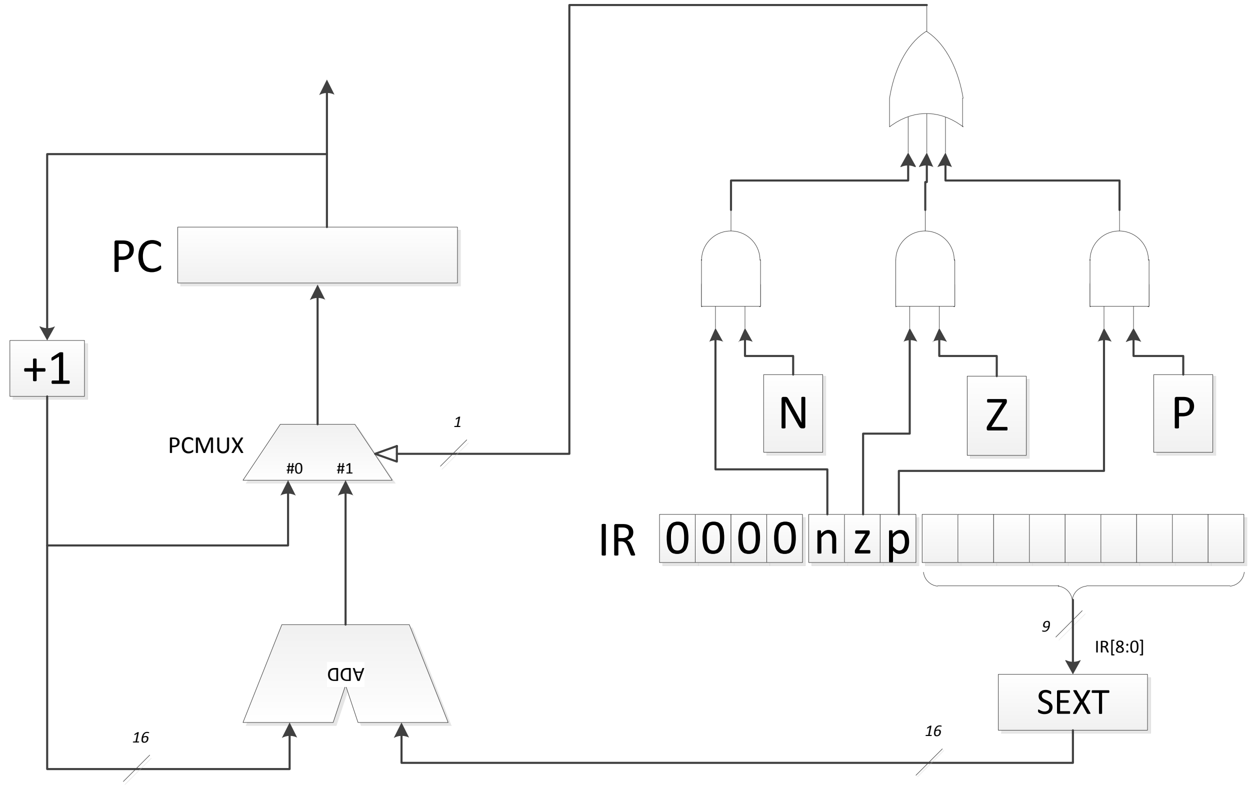
- Operation: if any of the condition codes tested is set, increment the PC with the sign-extended PCoffset9 (bits 0-8 of the instruction). In other words, branch is taken if specified condition is true.

- If (n AND N) OR (z AND Z) OR (p AND P)) then PC ← PC + SEXT(PCoffset9)

- Encoding:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BR	0	0	0	0	n	z	p									
	opcode				condition codes to test			PCoffset9								

- PCoffset range is from $-(2)^8$ to 2^8-1
- Datapath relevant to the execution of this instruction:



- Instruction cycle for BR:
  - FETCH AND DECODE phases are the same as for any other instruction
    - Note that PC ← PC + 1 during the FETCH phase
  - EVALUATE ADDRESS phase: compute PC + SEXT(PCoffset9)
  - EXECUTE phase
    - Condition codes N,Z,P for which corresponding bits n,z,p are set in the instruction are examined, .e.g.,
      - If n=1, N condition code ins checked

        - If the condition code that was examined is set, the branch is taken, that is PC ← PC + SEXT(PCoffset9)
        - Otherwise, no branch is taken, that is, PC ← PC
        - If all 3 bits n,z,p of the instruction are set, then all 3 condition codes N,Z,P are examined, and since at least one of them must be set, the condition is always met, and the branch is always taken, essentially unconditionally.

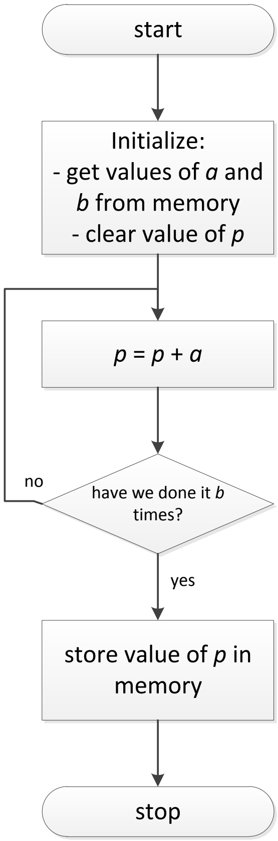
- Example1:
    - Z=1, N=P=0
    - PC: x4027
    - IR: 0000 010 011011001; BRz, x0D9
    - Outcome: PC ← x4027 + x1 + x0D9 (PC ← x4101)
- Example2:
    - For any N,Z, P
    - PC: x507B
    - IR: 0000 111 110000101; BRnzp, x185
    - Outcome: PC ← x507B + x1 + x185 (PC ← x5001)
- Example3:
    - N=1, Z=P=0
    - PC: x3000
    - IR: 0000 100 111111111; BRz. #-1
    - Outcome: PC ← x3000 + x1 - #1 (PC ← x3000)

Multiplication example

Write a program that computes a product of two integers: *a* x *b* = *p*

- *a* and *b* are 2's complement numbers
- we will impose a restriction that *b* > 0
- algorithm: replace product with the sum: *p* = *a* + *a* + .. + *a* (*b* times)
    - we will use R0 to hold value of *a*
    - we will use R1 to hold value of *b*
    - we will use R2 to hold value of *p*
    - our program will be located in memory starting at the address x3000
    - before the program starts
        - value of *a* is located in memory at the address x3010
        - value of *b* is located in memory at the address x3011
    - when the program is done, value of p should be located in memory at the address x3012
- First, we represent the algorithm for solving our problem as a flowchart
- Next, we translate the flowchart, step by step, into a program in machine language
    - We save the program (binary instruction column and optionally comments column only!) into a fine with extension .bin, e.g., **mult.bin**
    - Also need to add a line of code that specifies the starting address of the program
    - Convert it to an LC-3 executable: **lc3convert mult.bin**

- ○ Use either command line or graphical LC-3 simulator to run it
    - ▪ command line LC-3 simulator: **lc3sim mult.obj**
    - ▪ graphical LC-3 simulator: **lc3sim-tk mult.obj**


start
Initialize:
- get values of *a* and *b* from memory
- clear value of *p*
p = *p* + *a*
have we done it *b* times?
no
yes
store value of *p* in memory
stop

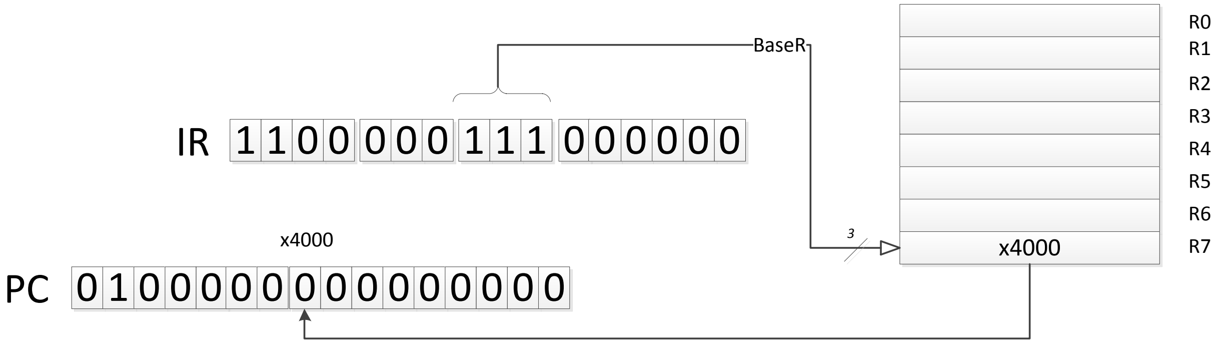
addr	binary instruction	comments
x3000	0101 010 010 000000	R2←0 (AND, R0, R0, #0)
x3001	0010 000 000001110	R0←*a* (LD, R0, #14)
x3002	0010 001 000001110	R1←*b* (LD, R1, #14)
x3003	0001 010 010 0 00 000	R2←R2+R0 (ADD R2, R2, R0)
x3004	0001 001 001 1 11111	R1←R1-1 (ADD R1, R1, #-1)
x3005	0000 001 111111101	PC←x3003 (BRp, #-3)
x3006	0011 010 000001011	*p*←R2 (ST R2, #11)
x3007	1111 0000 00100101	HALT
x3008		
x3009		
x300A		
x300B		
x300C		
x300D		
x300E		
x300F		
x3010		*a* is stored here
x3011		*b* is stored here
x3012		*p* is to be stored here

Unconditional jump (JMP)

- Operation: load the PC with the contents of the register specified by bits 6-8
    - ○ PC ← BaseR
- Encoding:

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

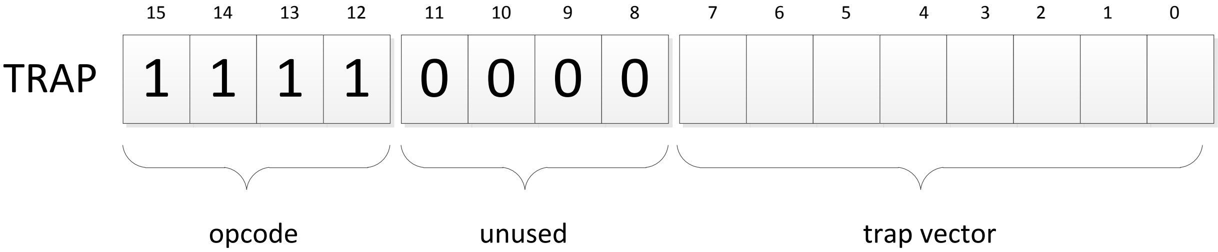
- Datapath relevant to the execution of this instruction:
    - ○ Example: **1100 000 111 000000; JMP R7**



- JMP instruction has no limitations on where the next instruction to be executed is located since the full 16-bit memory address can be stored in the register.

TRAP

- Operation: changes PC to a memory address that is a part of the OS. As the result, OS gets control an executes some task on behalf of our program
  - PC ← mem[trap vector]
- Encoding:



- TRAP instruction invokes an operating system *service call* identified by bits 7:0 of the instruction
- For now, remember the following 3 trap vectors:
  - x23 allows to input a character from the keyboard, its value will be placed in R0
  - x21 allows to output a character to the display (R0[7:0] are written out)
  - x25 halts the program
- we will look at the implementation details of TRAP instruction later (Ch 9)

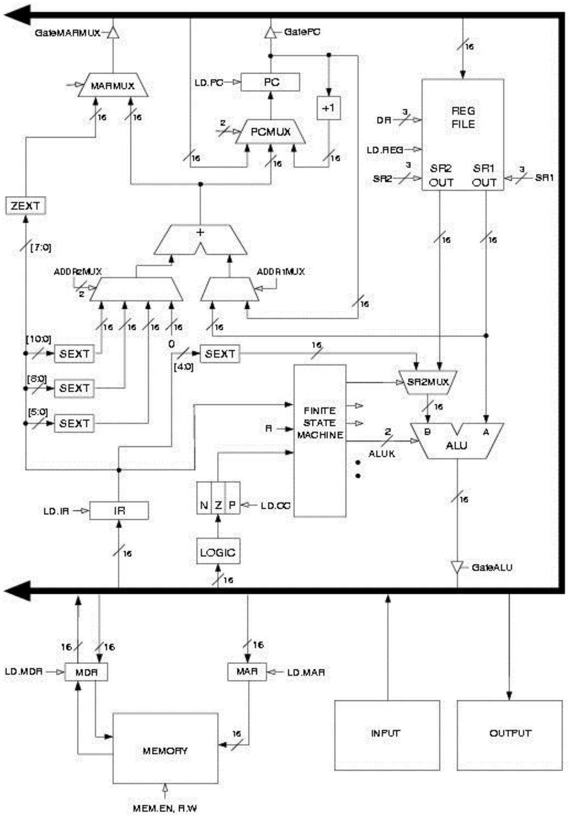
LC-3 data path review

- The LC-3 data path consist of all the components that process the information during an instruction cycle
  - The functional unit that operates on the information
  - The registers that store the information
  - Buses and wires that carry information from one point to another
- **Basic components of the data path:**

- The global bus
    - LC-3 global bus consists of 16 wires and associated electronics
    - It allows one structure to transfer up to 16 bits of information to another structure by making the necessary electronic connections on the bus
    - Exactly one value can be translated on the bus at a time
    - Each structure that supplies values to the bus connects to it via a *tri-state device* that allows the computer's control logic to enable exactly one supplier of information to the bus at a time
    - The structure wishing to obtain information from the bus can do so by asserting (setting to 1) its LD.x (load enable) signal.
- Memory
    - Memory in LC-3 is accessed by loading the memory address value into MAR
    - The result of memory read is stored in MDR
    - The computer's control logic supplies the necessary control signals to enable the read/write of memory
- The ALU and the register file
    - The ALU is the processing element
        - Has two inputs and one output
        - Inputs come directly from the register file
        - One of the inputs can also be supplied directly from the IR, controlled by the SR2MUX
        - Output goes to the bus
            - It then is stores in the register file and
            - Processed by some additional circuitry to generate the condition codes N,Z,P
    - The register file consists of 8 16-bit registers
        - It can supply up to two values via its two output ports
        - It can store one value, coming from the bus
        - Read/write access to the register file is controlled by the 3-bit resister signals, DR, SR1, SR2
- The PC and PCMUX
    - The PC supplies via the global bus to the MAR the address of the instruction to be fetched at the start of the instruction cycle. The PC itself is supplied via the three-to-one PCMUX, depending on the instruction being executed
    - During the fetch cycle, the PC is incremented by 1 and written to the PC register
    - If the executed instruction is a control instruction, then the relevant source of the PCMUX depends on the type of the control instruction and is computed using a special ADD unit
- The MARMUX
    - Controls which of two sources will supply memory address to MAR

  - It is either the value coming form the instruction register needed to implement TRAP instruction, or the value computed based on the PC or a value stored in one of the general-purpose registers

- LC-3 instruction cycle revisited
  - FETC
    - IR contains the fetched instruction
    - PC is incremented by 1
  - DECODE
    - Instruction is decoded resulting in control logic providing various control signals to the computer
  - EVALUATE ADDRESS
    - Address of memory to be accessed during the execution of the instruction is computed
    - Memory access instructions require this phase
  - OPERAND FETCH
    - The data from memory is loaded into MDR
  - EXECUTE
    - ALU is directed to perform the operation
    - Memory access instructions do not have this phase
  - STORE RESULTS
    - Results are stored to memory or registers, depending on the instruction


GateMARMUX
GatePC
16
MARMUX
LD.PC
PC
+1
REG FILE
DR
3
LD.REG
SR2
SR2 OUT
SR1 OUT
SR1
2
PCMUX
16
ZEXT
[7:0]
+
ADDR2MUX
ADDR1MUX
[10:0]
SEXT
[8:0]
SEXT
[5:0]
SEXT
0
[4:0]
SEXT
FINITE STATE MACHINE
R
SR2MUX
B
A
ALU
ALUK
LD.IR
IR
N
Z
P
LD.CC
LOGIC
GateALU
LD.MDR
MDR
MAR
LD.MAR
MEMORY
MEM.EN, R.W
INPUT
OUTPUT