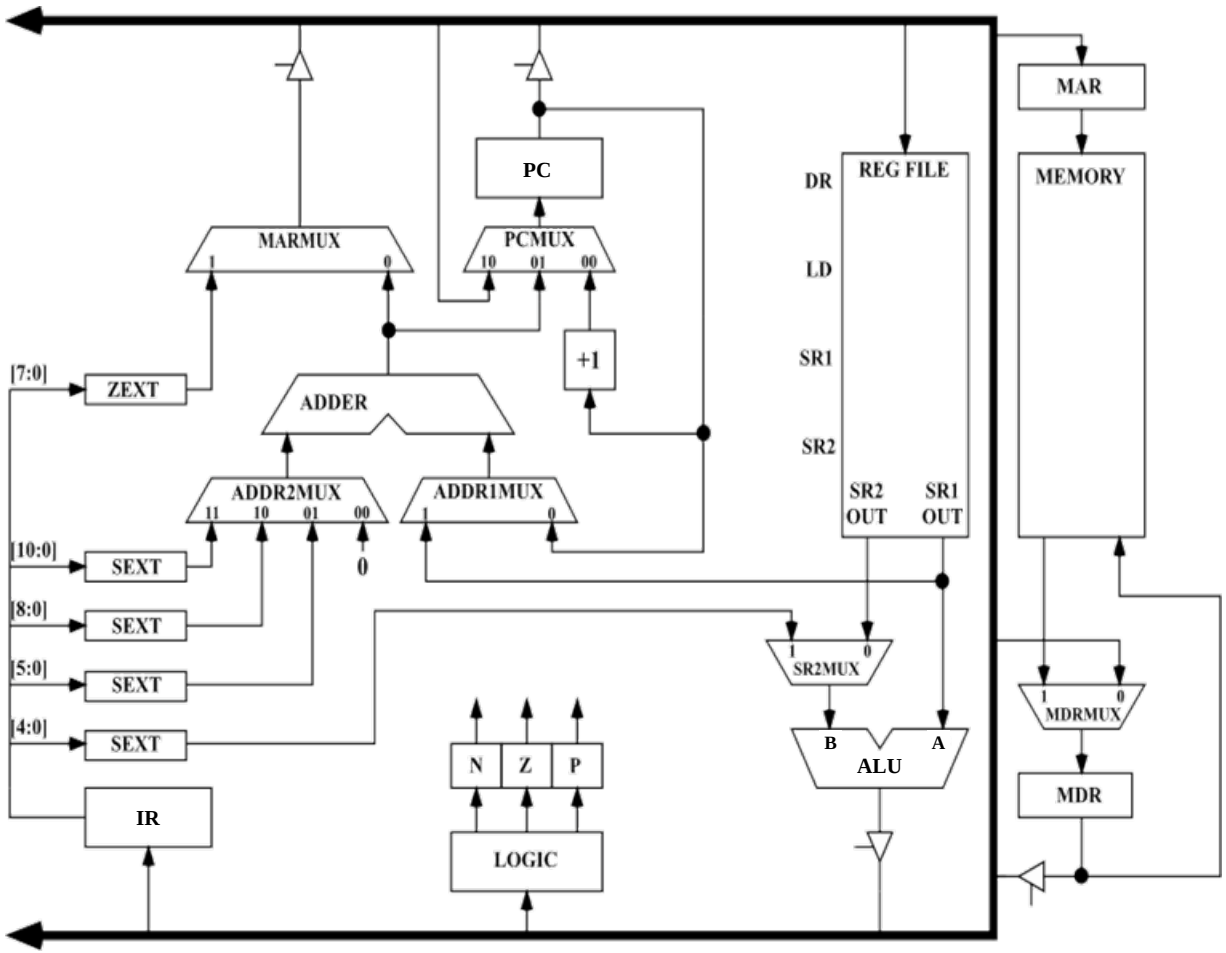
LC3-1

The LC-3
A Review



Introduction

- In this class we will:
  - Complete the hardware design of the LC-3
  - Simulate it
  - Run programs on it and hopefully download on the board.

Reference Information

- You will *need* the LC-3 Description
- Patt & Patel Textbook:

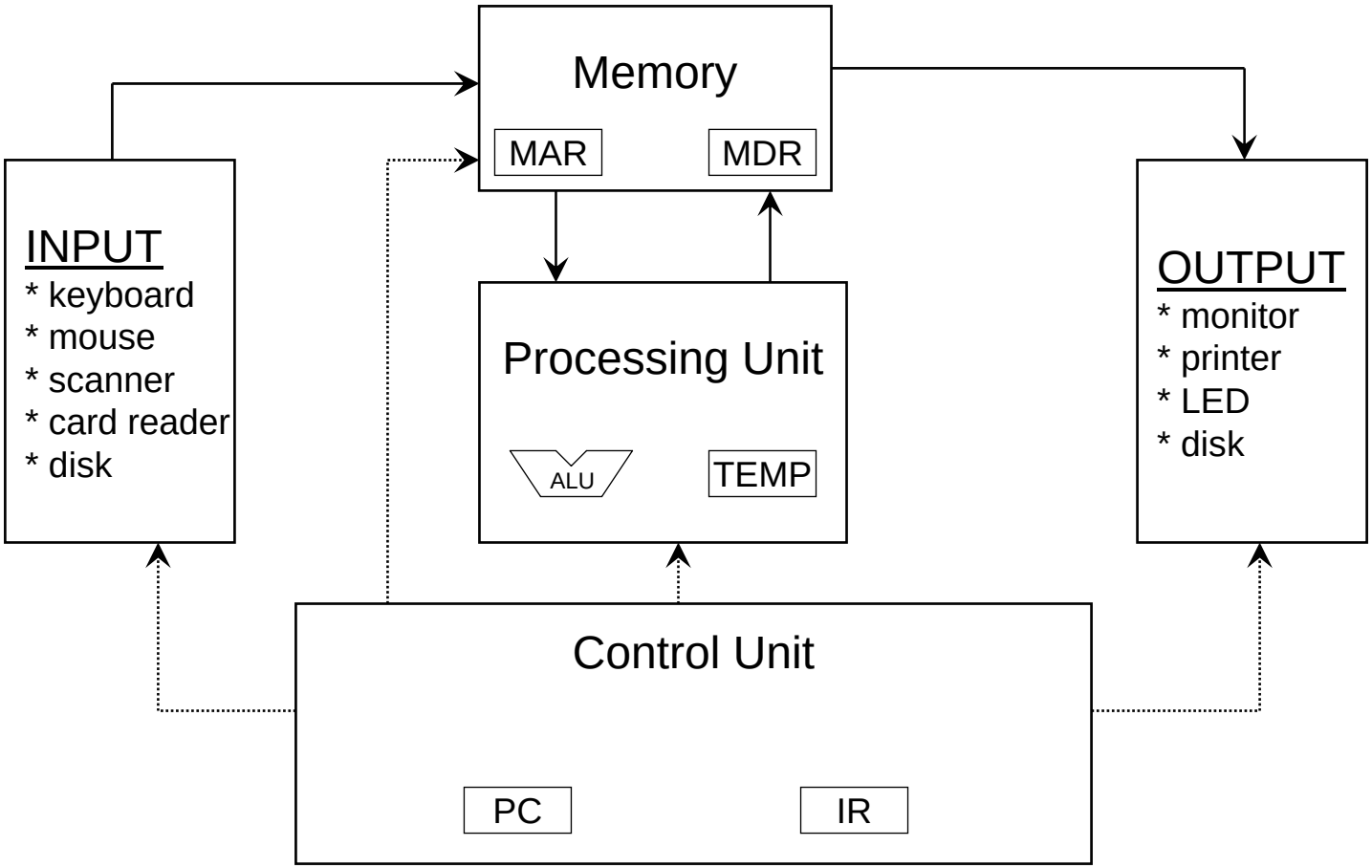
  "Introduction to Computing Systems" (second edition)

  Yale N. Patt & Sanjay J. Patel

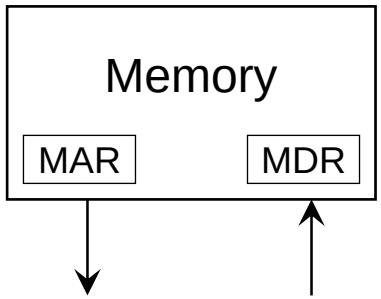
  McGraw-Hill Higher Education 2004

- Useful Sections (in order of importance):
  - Appendix A
  - Chapter 5
  - Chapter 4
- Or use the links supplied in the reference section of the class webpage

The Von Neumann Model

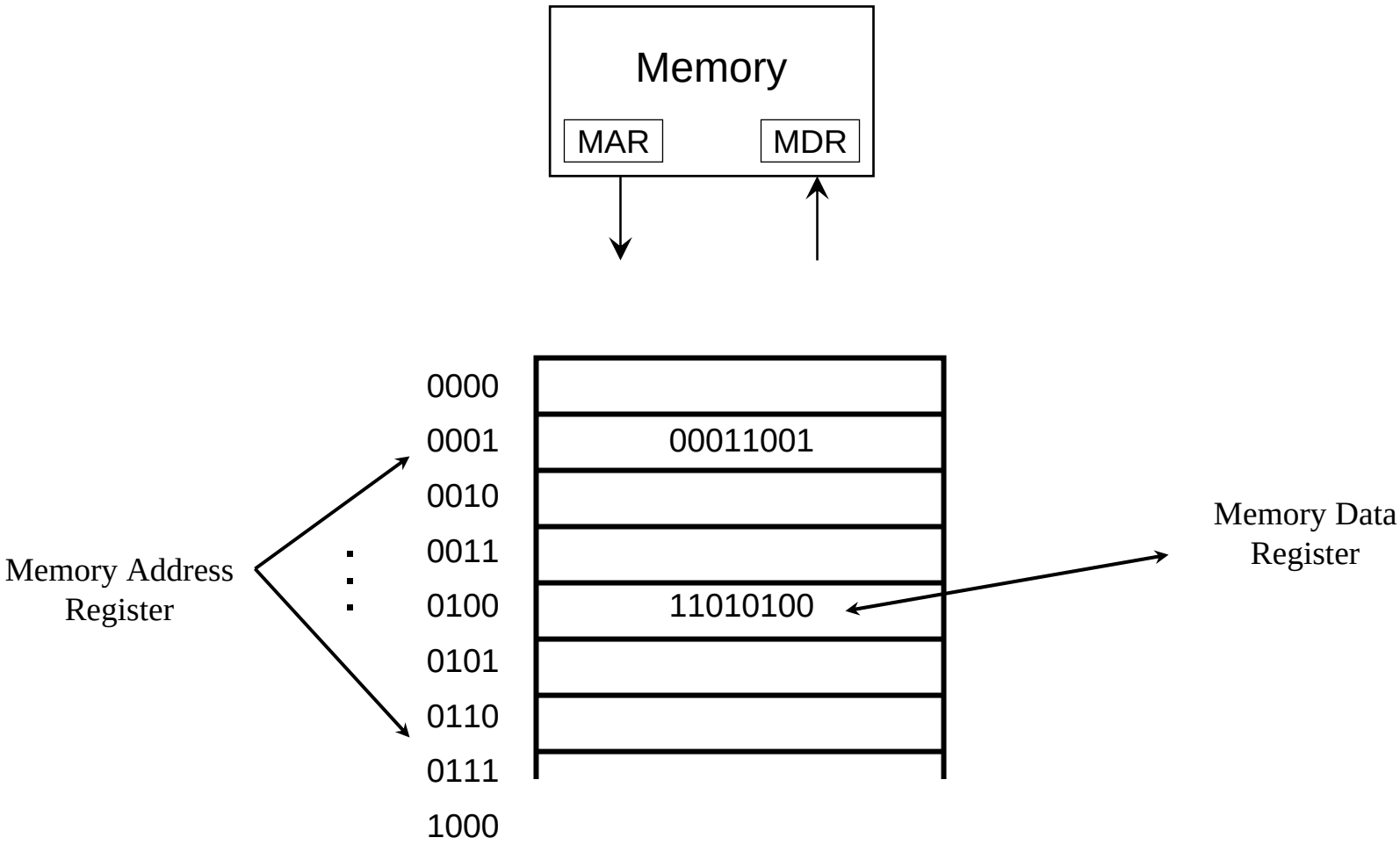


The Von Neumann Model



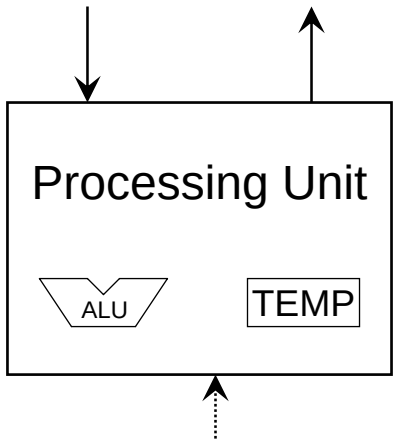
- Memory is used to store a sequence of instructions
- Memory is also used to store data
- Memory Address Register (MAR) selects which location in memory will be read or written
- Memory Data Register (MDR) contains the data read or to be written

The Von Neumann Model



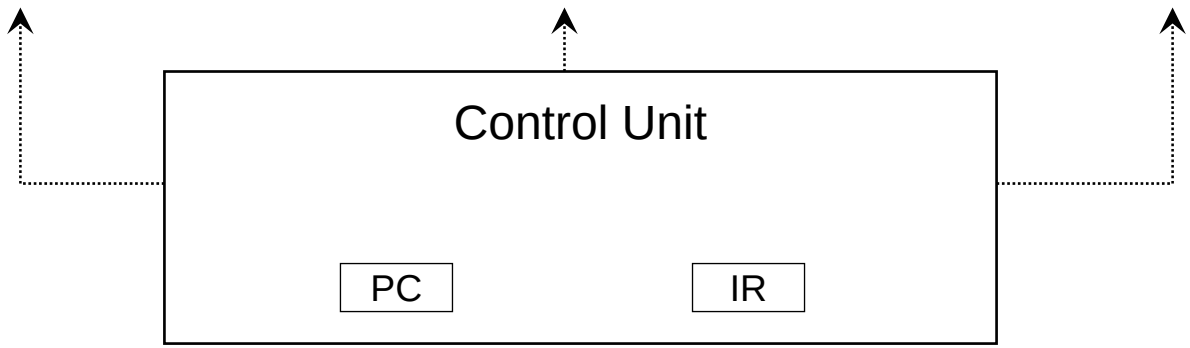
The Von Neumann Model

- Arithmetic Logic Unit (ALU) does computations and information processing (ADD, AND, NOT, etc.)
- Registers (TEMP) provide a small amount of high-speed temporary storage

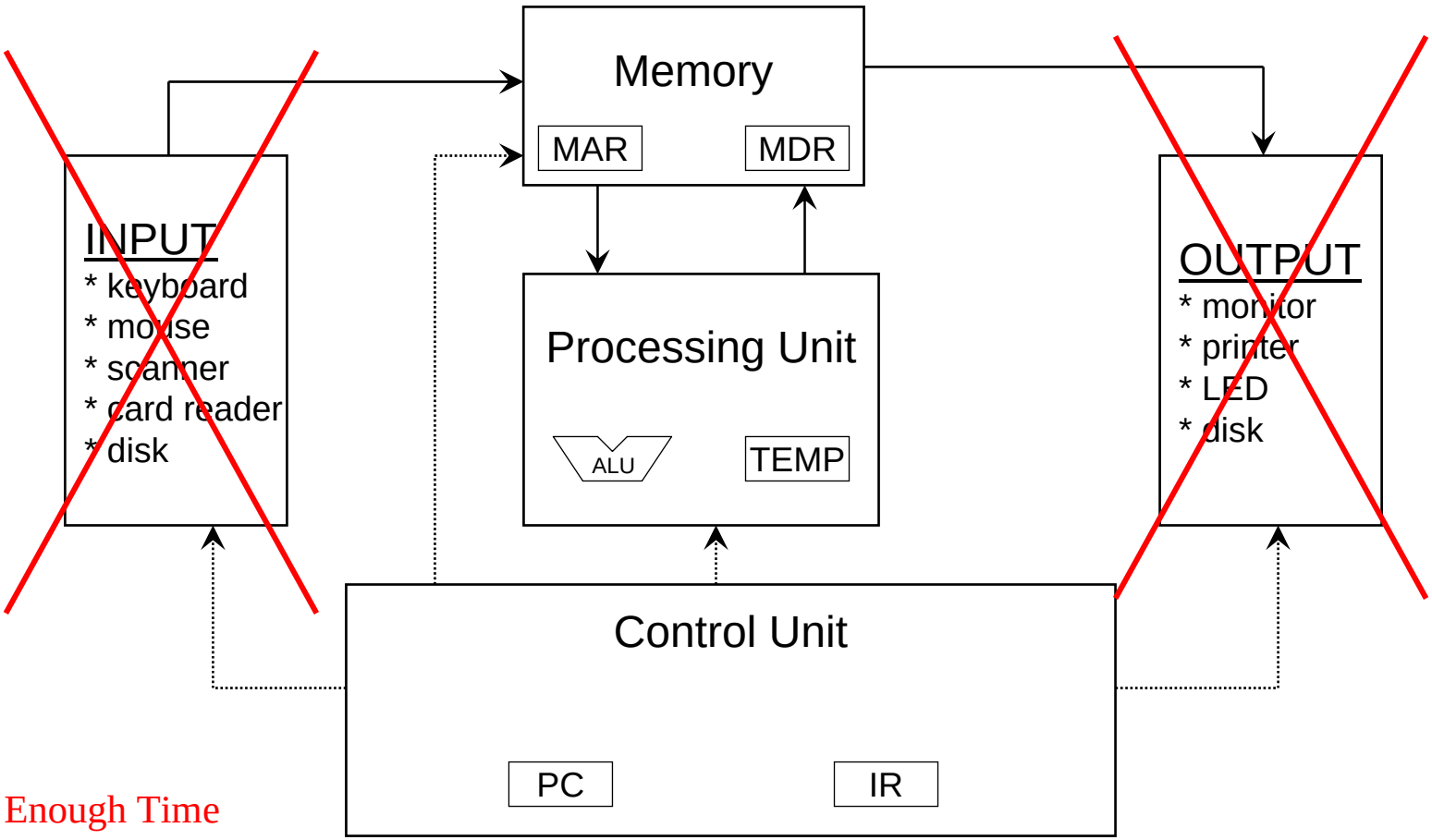


The Von Neumann Model

- Control Unit (CU) determines what to do next and controls the rest of the processor
- Program Counter (PC) contains the address of the next instruction to be executed
- Instruction Register (IR) contains the current instruction being executed



The Von Neumann Model

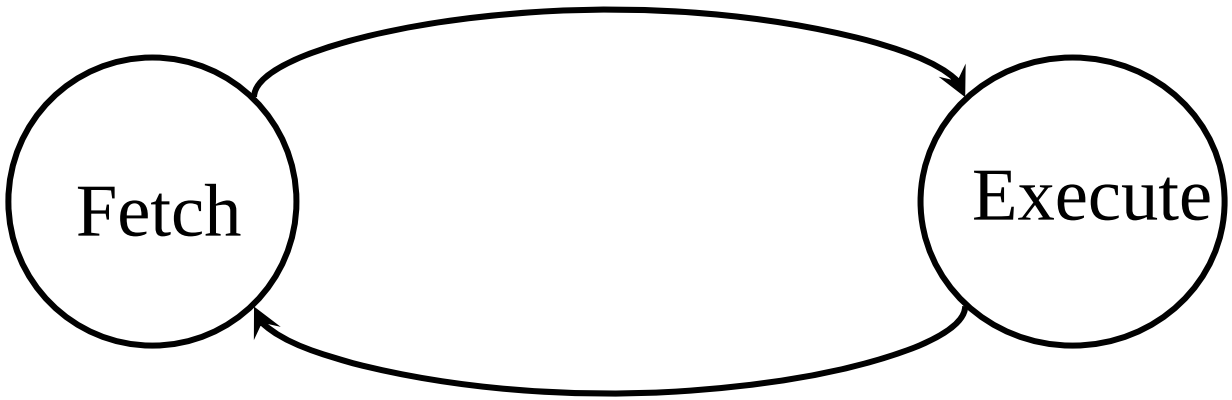


Not Enough Time
to Study Everything

The Von Neumann Model

- Fetch an instruction
- Execute it
- Repeat

(Looks a lot like a State Graph)



The Instruction Set Architecture (ISA)

- ISA for LC-3
  - Everything about the computer the software needs to know
    - memory organization
    - register set
    - instruction set
      - opcodes
      - data types
      - addressing modes
  - Everything the hardware designer needs to know in order to build the computer

Memory Organization

- The LC-3 is a 16-bit machine
  - all instructions fit into a 16-bit word
  - memory is accessed using a 16-bit address word
    - its address space is 2^{16} locations (65,536 locations)
  - memory is *word-addressable*
    - each location is 16-bits wide (2 bytes each)
    - total memory size is 131,072 bytes
    - in most other machines memory is byte-addressable
      - LC-3 is different in this respect
- Use a 16-bit word to address memory
- Get back a 16-bit value

Register Set

- Memory access is slow(er)
  - it is outside the processing unit
  - it takes a whole instruction cycle to access (LDR)
- Registers are *inside* the processing unit
  - they can be accessed *during* an instruction (ADD)
- All computers have a *register set*
  - LC-3 has 8 general purpose registers
  - Named R0, R1, R2, R3, R4, R5, R6, R7
  - They are addressed with a 3-bit field in an instruction

Data Types

- LC-3 has only one data type
  - a 16-bit 2's complement integer

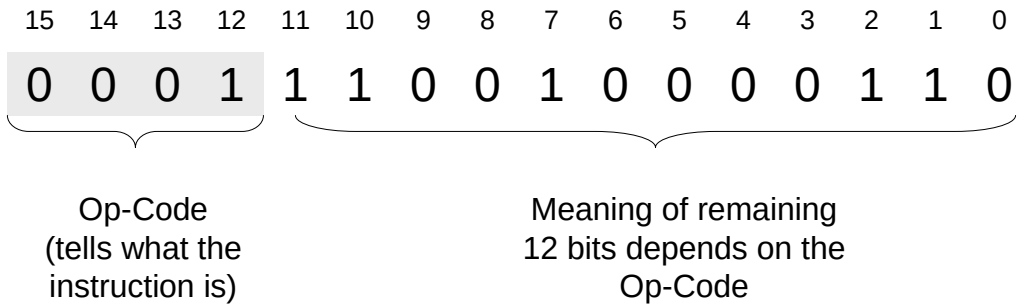
- Other computers have others
  - 32-bit floating point (*float*)
  - 64-bit floating point (*double*)
  - long
  - short
  - byte

LC-3 Instructions

ADD	0001	DR	SR1	0	00	SR2
ADD	0001	DR	SR1	1	imm5	
AND	0101	DR	SR1	0	00	SR2
AND	0101	DR	SR1	1	imm5	
NOT	1001	DR	SR	111111		
BR	0000	n	z	p	PCoffset9	
JMP	1100	0	00	BaseR	000000	
JSR	0100	1	PCoffset11			
JSRR	0100	0	00	BaseR	000000	
RET	1100	0	00	111	000000	

LD	0010	DR	PCoffset9	
LDI	1010	DR	PCoffset9	
LDR	0110	DR	BaseR	offset6
LEA	1110	DR	PCoffset9	
ST	0011	SR	PCoffset9	
STI	1011	SR	PCoffset9	
STR	0111	SR	BaseR	offset6
TRAP	1111	0000	trapvect8	
RTI	1000	000000000000		
reserved	1101			

Anatomy of an Instruction



This is a 16-bit instruction format.
The instruction always fills one 16-bit word.

A Note About Register Notation

- We will often write things like this:

  R6 = R5 + R3

- What we mean is:
  - the result of adding the *contents of* R5 to the *contents of* R3 is stored into R6
- What does this mean?

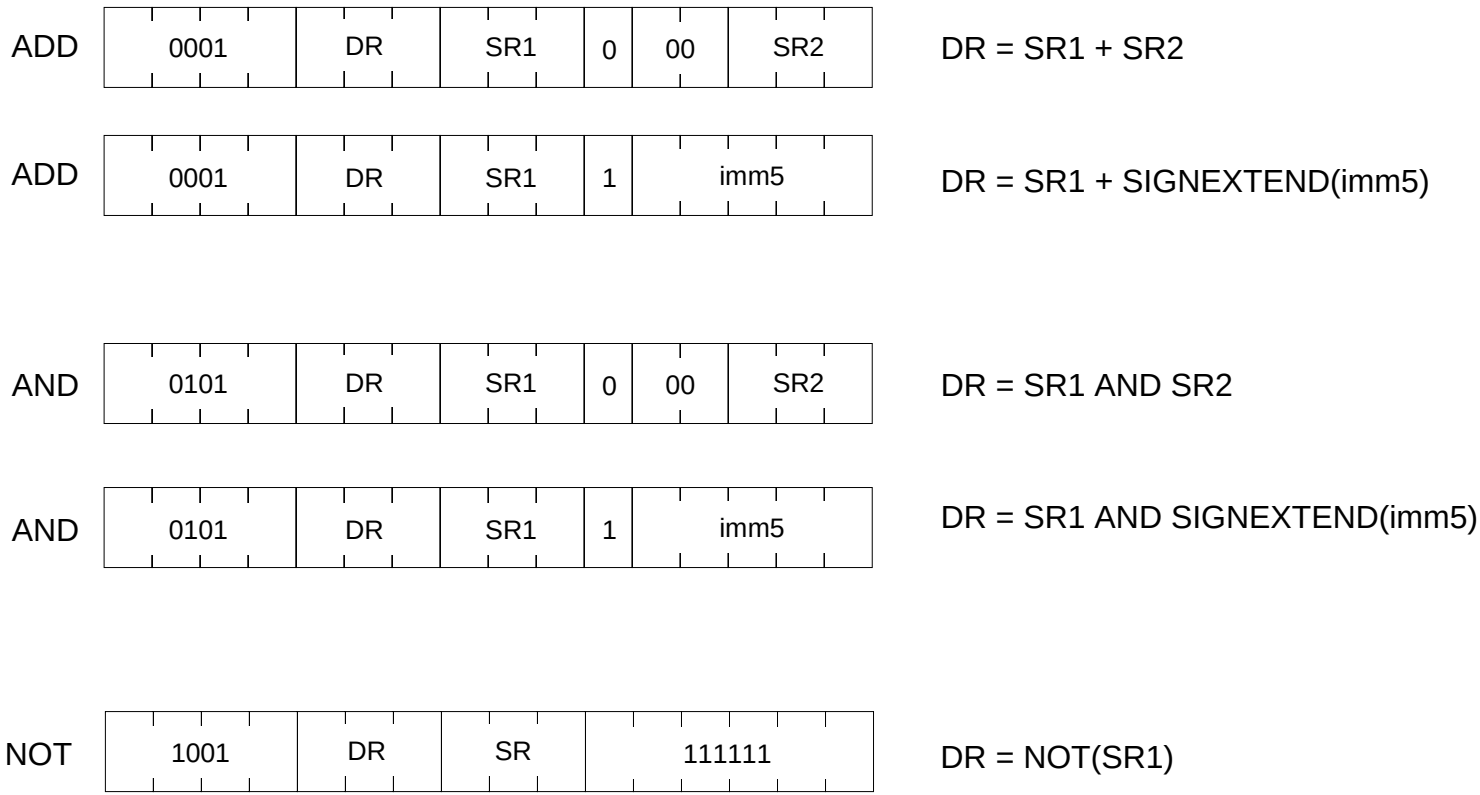
  R6 = R5 + 7

  - the result of adding the *contents of* R5 to the integer 7 is stored into R6

The Instruction Set

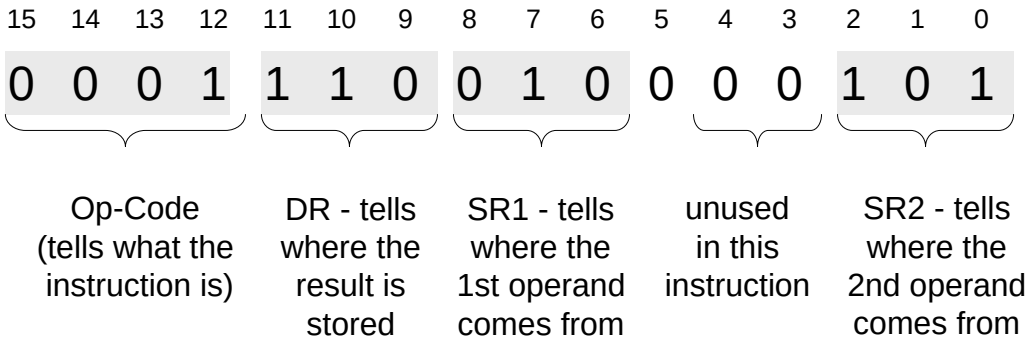
- LC-3 has 16 instructions
- Three *types* of instructions
  - Operate instructions
    - operate on data **(ADD R6, R2, R5)**
  - Data movement instructions
    - memory <=> registers **(LDR R2, R3, #6)**
    - memory/registers <=> input/output devices
  - Control instructions
    - change which instruction is executed next **(JMP R3)**

The Operate Instructions



An Operate Instruction

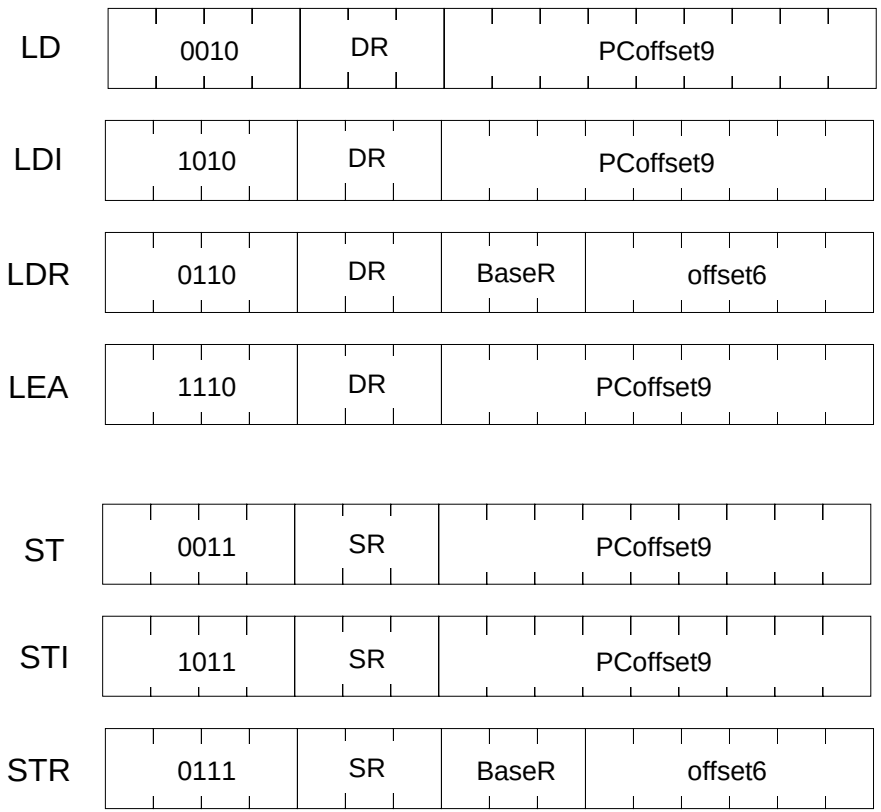
ADD R6 R2 R5



R6 = R2 + R5

This is a 16-bit instruction format -
the instruction fills a 16-bit word.
Not all bits have meaning in this particular instruction.

The Data Movement Instructions



DR = mem [PC + (sign extended) PCoffset9]

DR = mem [mem [PC + (sign extended) PCoffset9]]

DR = mem [BaseR + (sign extended) offset6]

DR = PC + (sign extended) PCoffset9

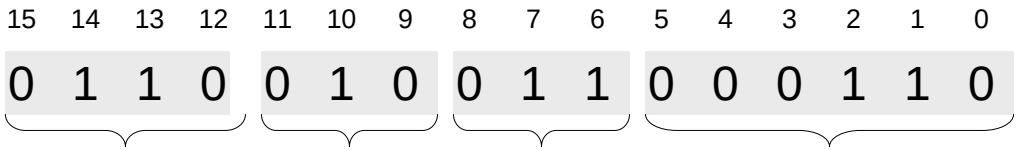
mem [PC + (sign extended) PCoffset9] = SR

mem [mem [PC + (sign extended) PCoffset9]] = SR

mem [BaseR + (sign extended) offset6] = SR

An LDR Instruction

LDR R2 R3 6



Op-Code (tells what the instruction is)

DR - tells where the value fetched from memory will be placed

BaseR - tells where the base address comes from

Offset6 - is added to contents of BaseR to get the memory location to fetch from

LDR R2 R3 6

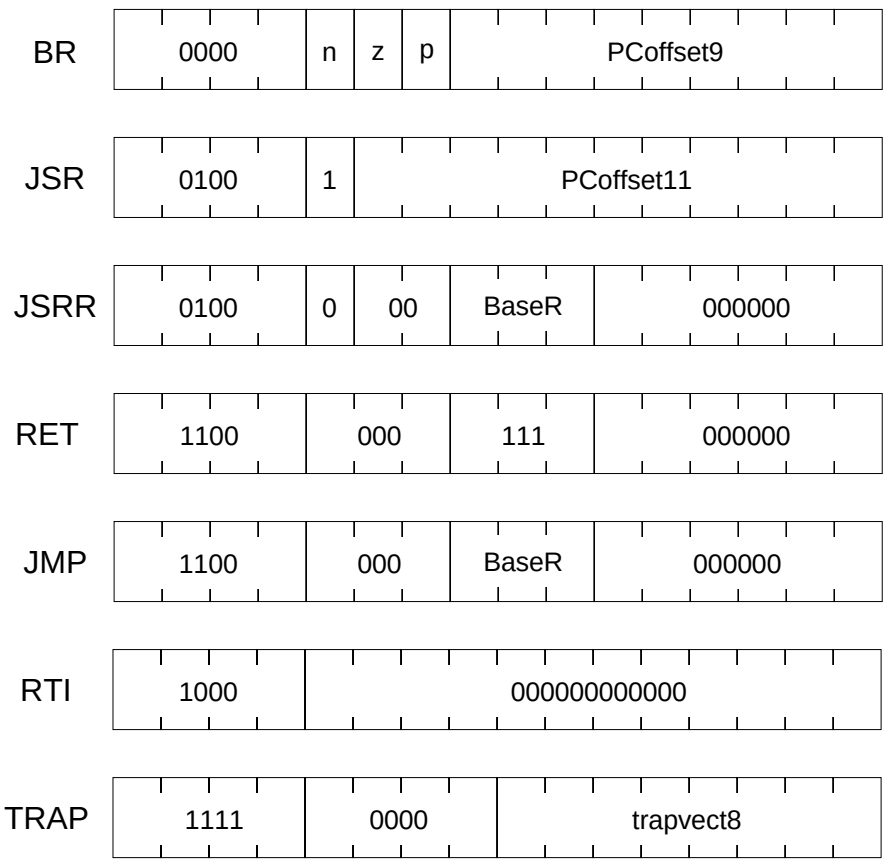
Offset is sign-extended before being added to base

EffectiveMemoryAddress <= R3 + 6

R2 = MEM[EffectiveMemoryAddress]

This requires the computation of an effective memory address. It is base + offset. The contents of R3 are the base address and 6 is the offset.

Control Instructions



PC = PC + (sign extended) PCoffset9
depending on condition(s)

R7 = PC
PC = PC + (sign extended) PCoffset11

R7 = PC
PC = BaseR

PC = R7

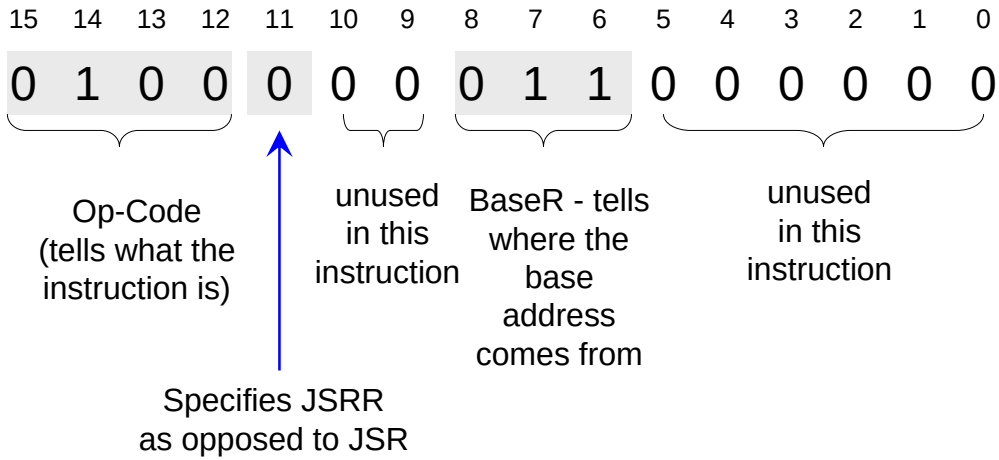
PC = BaseR

Different name, instructions…

Probably won't have time

A JSRR Instruction

JSRR R3



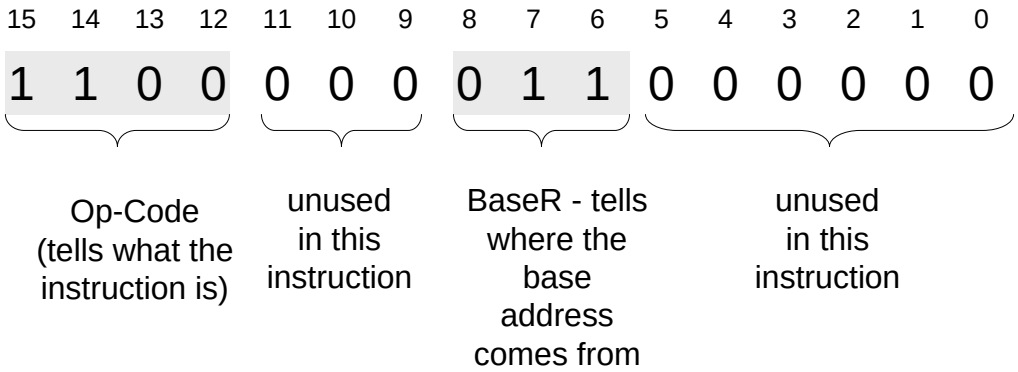
JSRR R3

R7 <= PC
PC <= R3

This is how a subroutine call would be executed.

A JMP Instruction

JMP R3



JMP **R3**

PC <= R3

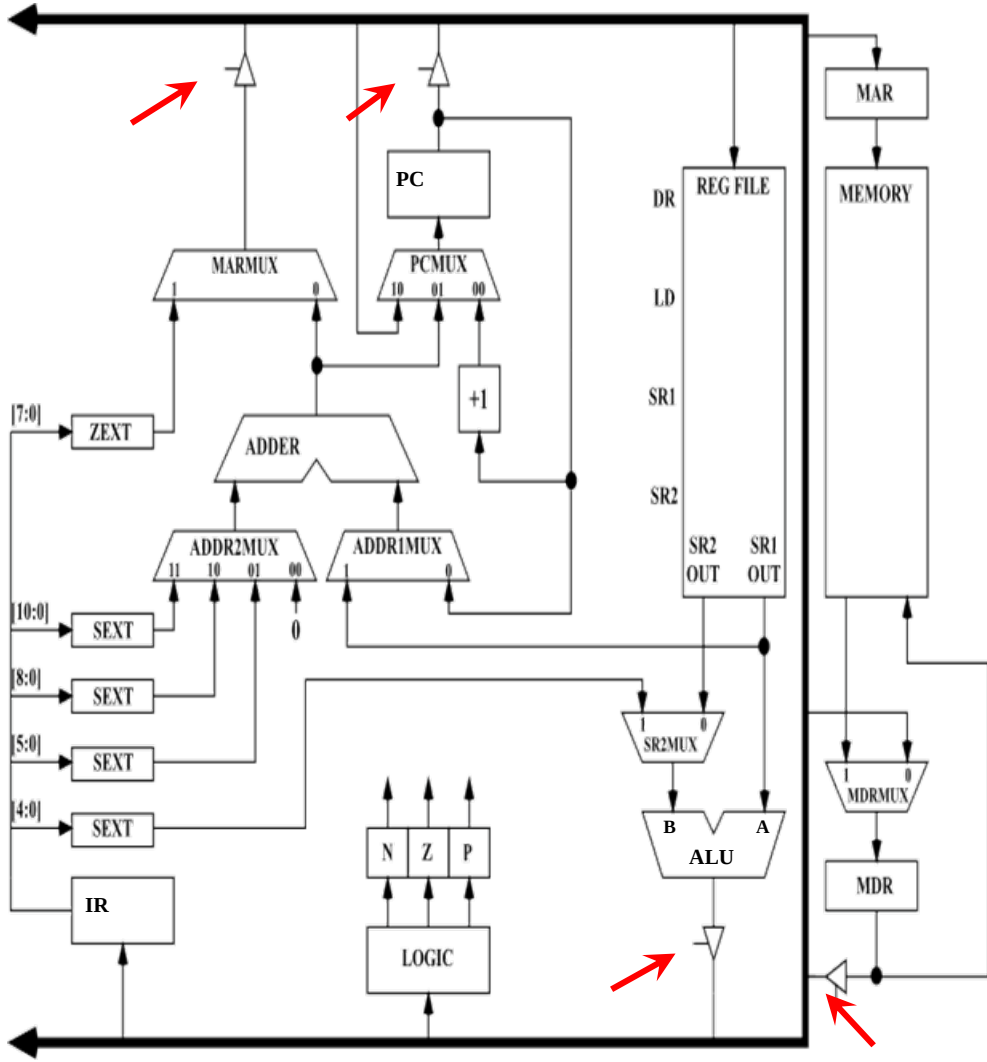
This is how a GOTO statement would be executed.

The LC-3 Architecture

A More Detailed Look

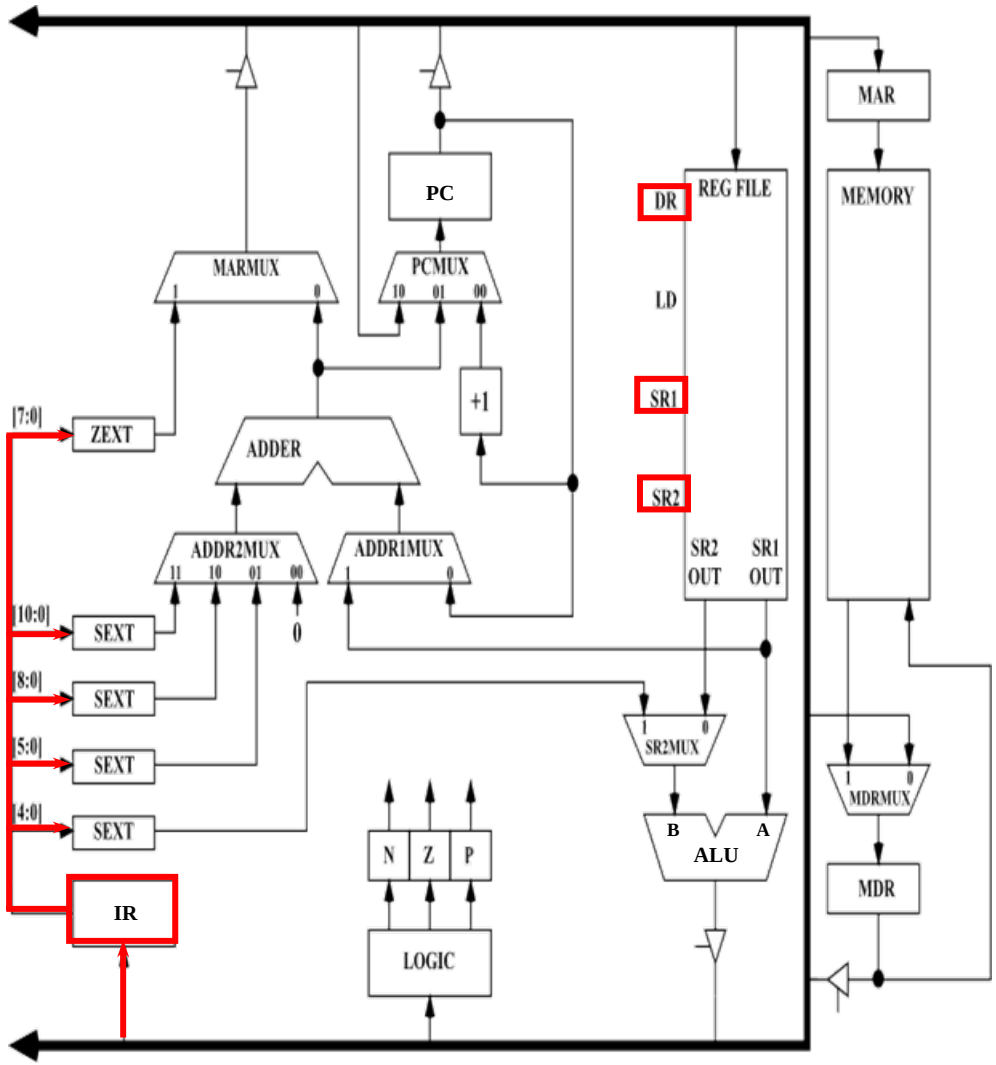
The LC-3 - Global Bus

- A *bus*
  - Common data highway
    - multiple on-ramps and off-ramps
  - Most data transfers between units go across the bus
    - Example: PC => MAR
    - Example: MDR => IR
- A *tri-state* driver
  - Can drive 1's and 0's on the bus
  - Can disconnect from the bus
- Control unit turns them on and off



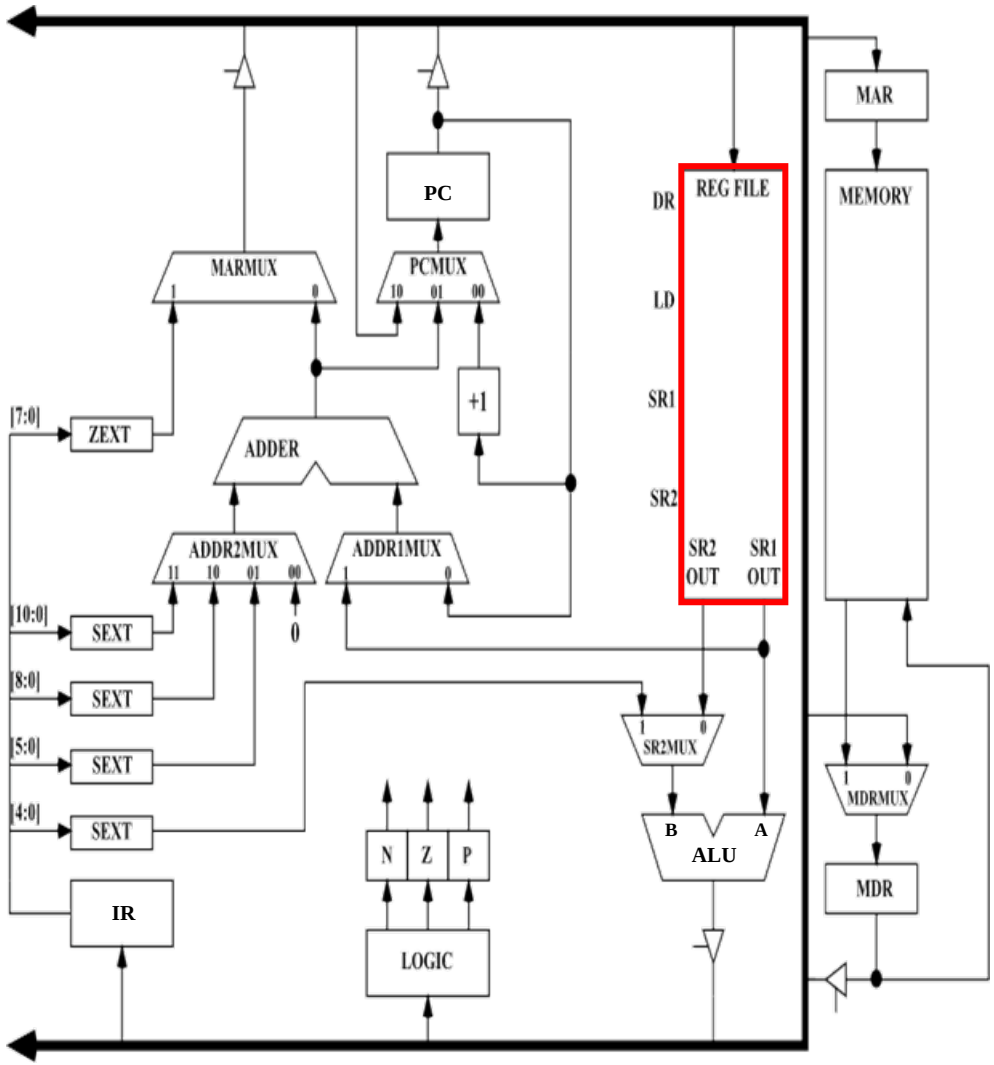
The LC-3 - Instruction Register (IR)

- The *IR*
  - During a fetch the IR is loaded from the bus
  - Control unit controls when it should be loaded
  - Its fields are pulled apart and fed to many places in the circuit
    - op code
    - source/destination registers
    - immediate data
    - offsets



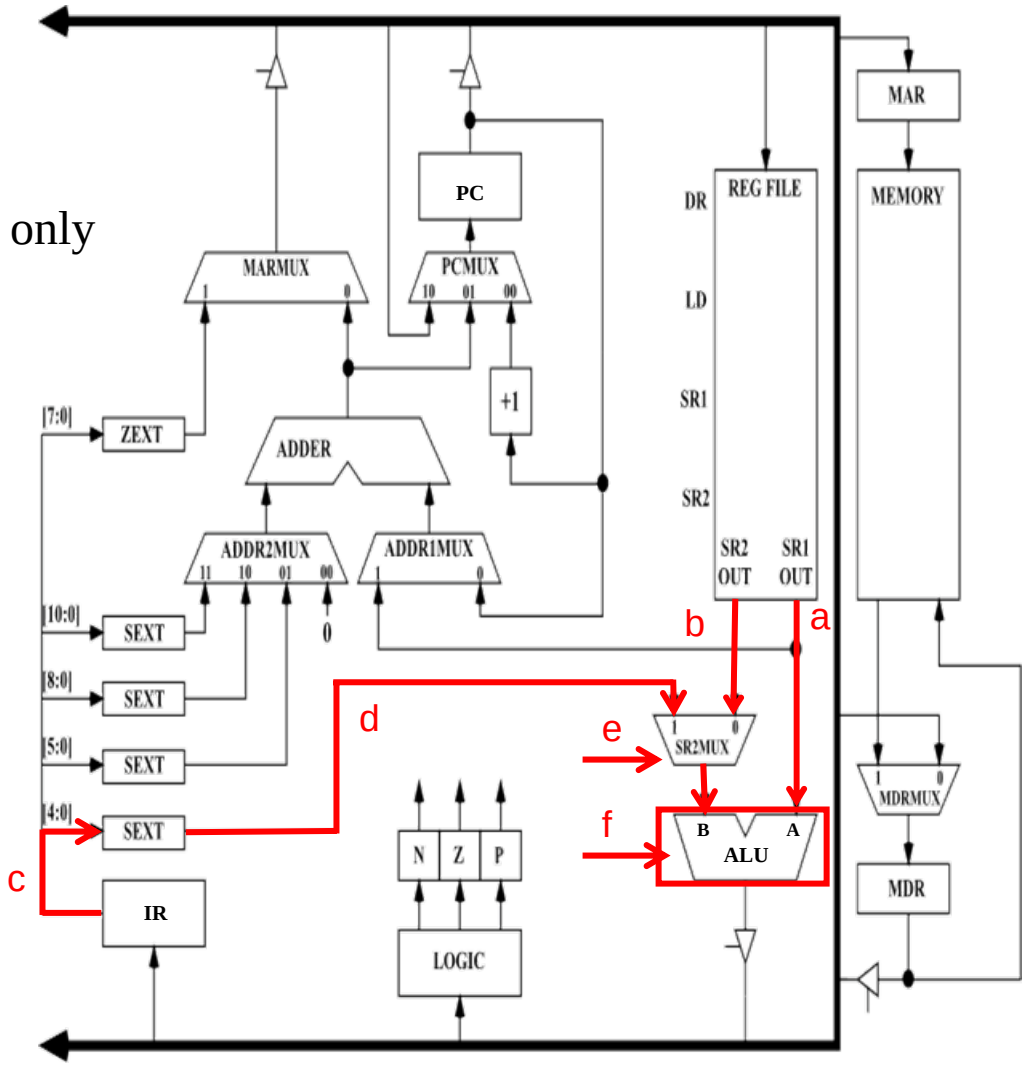
The LC-3 - Registers

- The *register file*
  - 8 words of 16-bits each
  - R0-R7
- Two read address ports
- One write address port
- Control unit generates control and address signals
  - to read register file
  - to write back into the register file

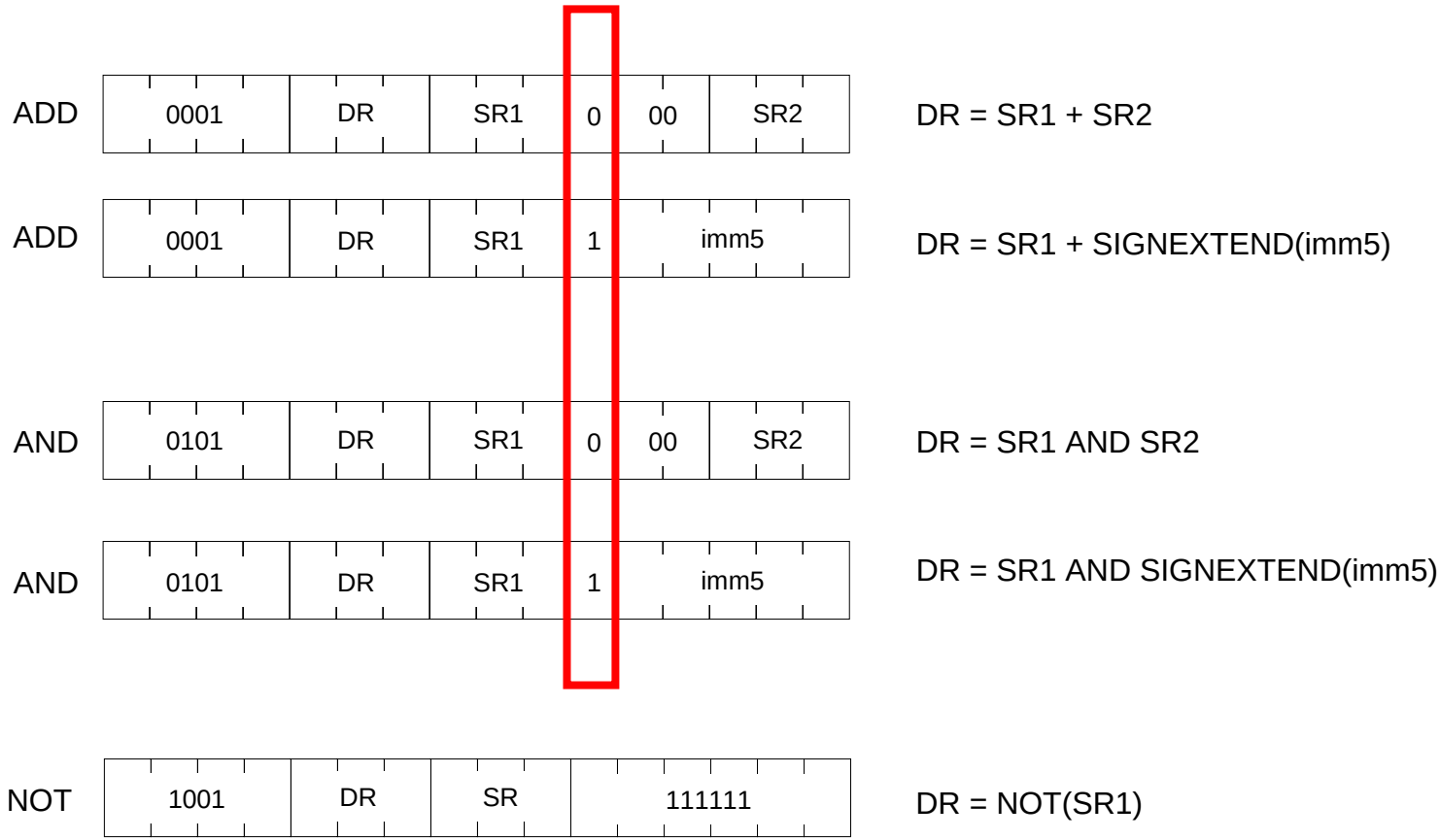


The LC-3 - ALU

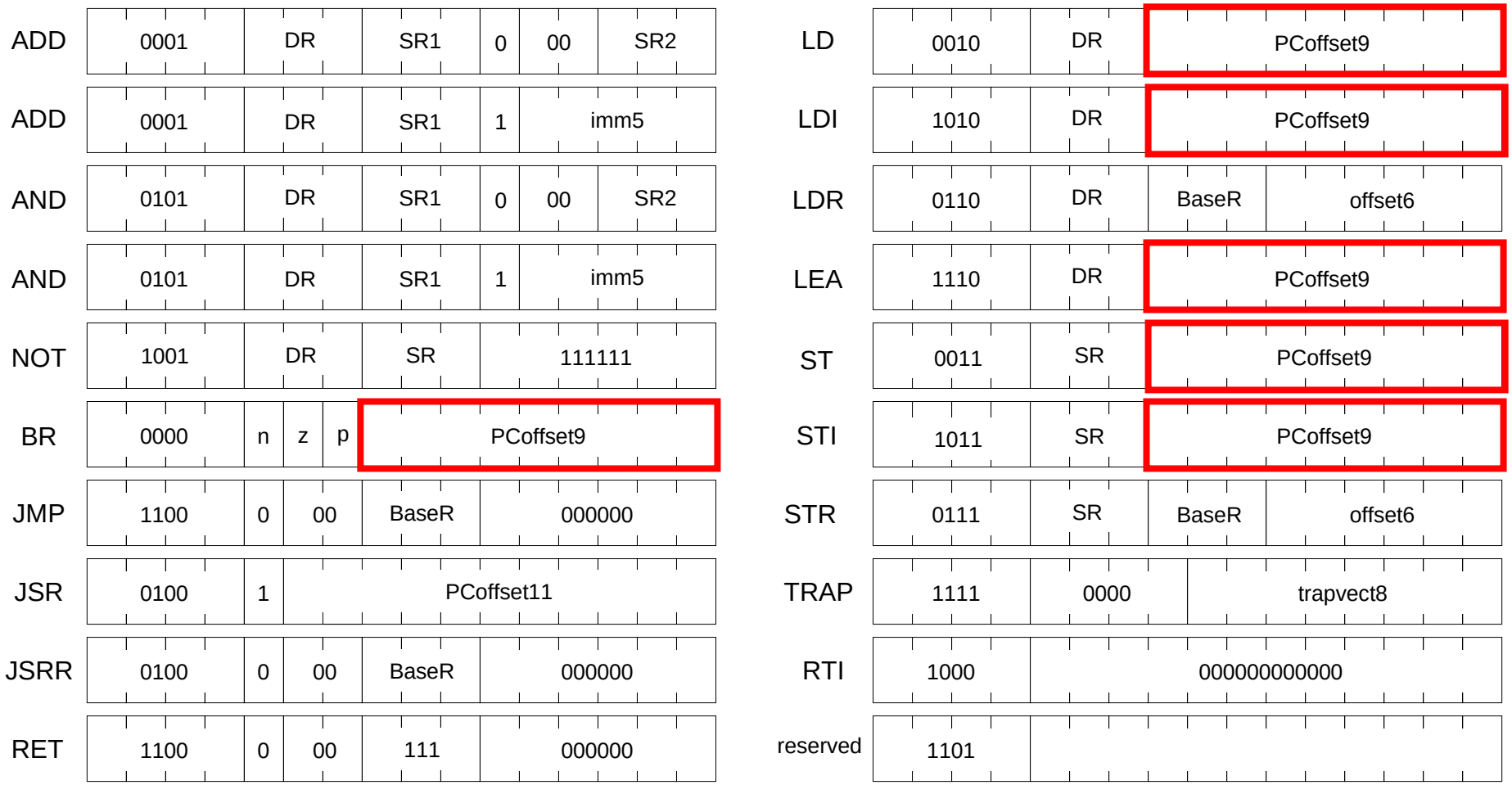
- The *ALU*
  - Does the arithmetic and logical operations on the data
  - It is *always* working, results are only stored away at the right time
- One input always comes from register file (a)
- Second input has two sources
  - register file (b)
  - imm5 from instruction (c)
    - always sign extended (d)
- Bit 5 of IR selects 2nd input (e)
- Control unit tells ALU which operation to perform (f)



The Operate Instructions



The LC-3 - EAB

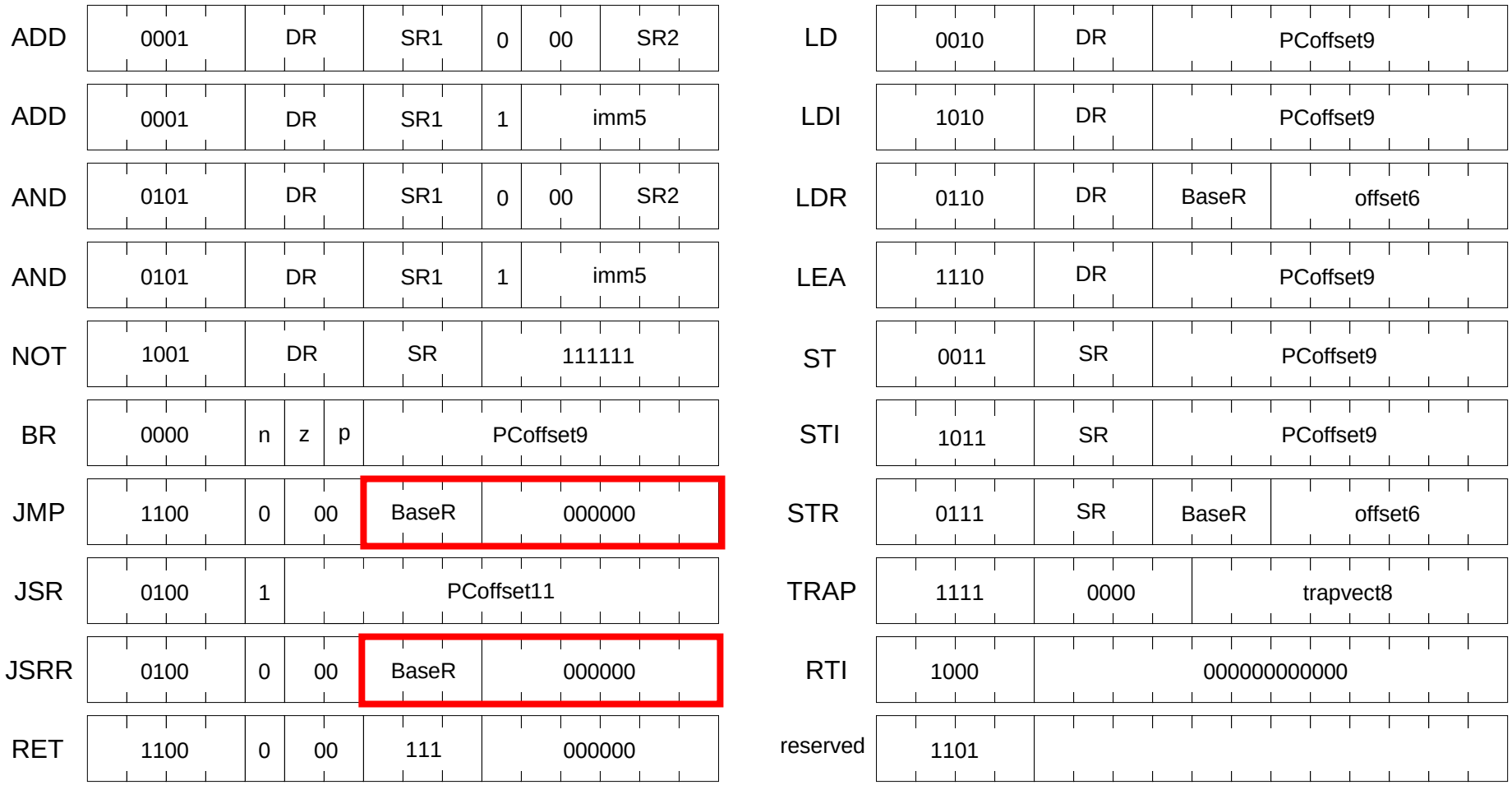


The LC-3 - EAB

ADD	0001	DR	SR1	0	00	SR2
ADD	0001	DR	SR1	1	imm5	
AND	0101	DR	SR1	0	00	SR2
AND	0101	DR	SR1	1	imm5	
NOT	1001	DR	SR	111111		
BR	0000	n	z	p	PCoffset9	
JMP	1100	0	00	BaseR	000000	
JSR	0100	1	PCoffset11			
JSRR	0100	0	00	BaseR	000000	
RET	1100	0	00	111	000000	

LD	0010	DR	PCoffset9	
LDI	1010	DR	PCoffset9	
LDR	0110	DR	BaseR	offset6
LEA	1110	DR	PCoffset9	
ST	0011	SR	PCoffset9	
STI	1011	SR	PCoffset9	
STR	0111	SR	BaseR	offset6
TRAP	1111	0000	trapvect8	
RTI	1000	000000000000		
reserved	1101			

The LC-3 - EAB



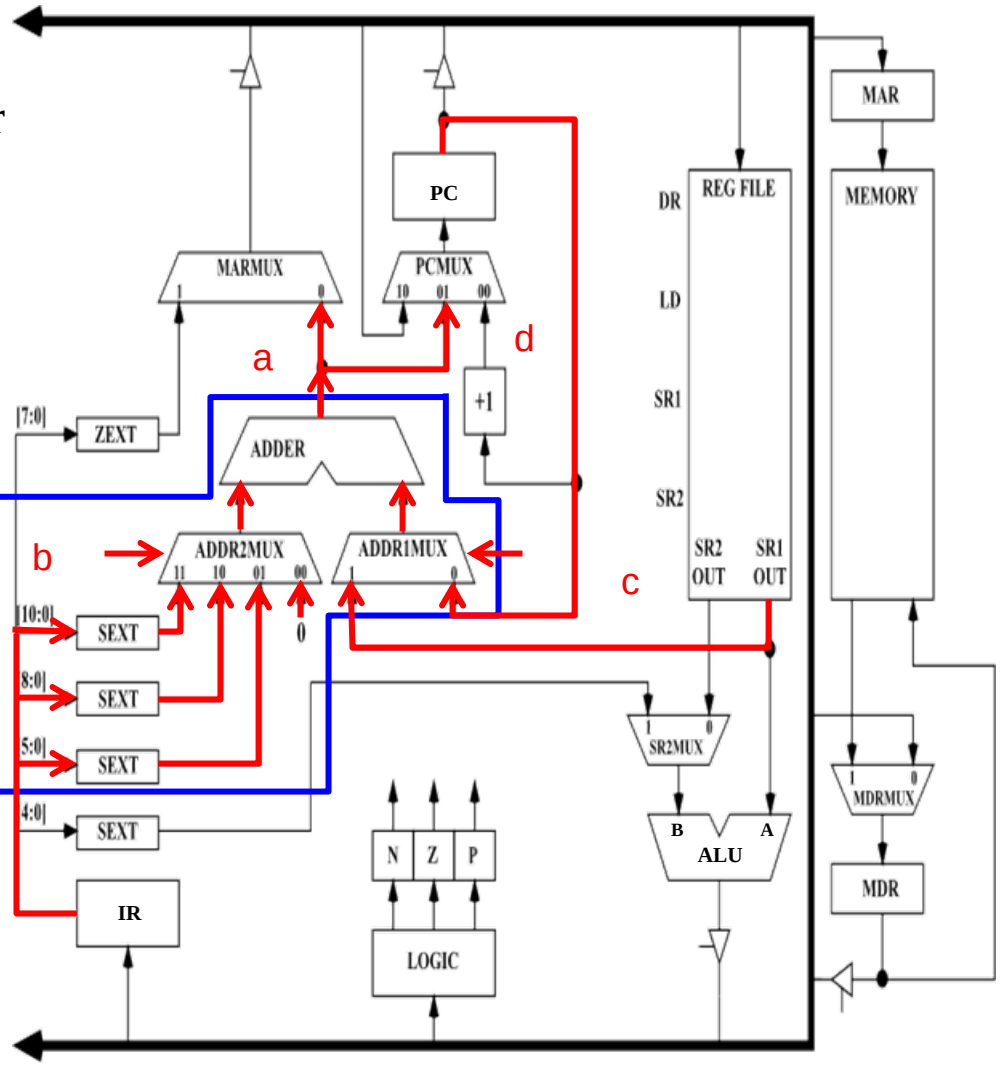
The LC-3 - EAB

ADD	0001	DR	SR1	0	00	SR2
ADD	0001	DR	SR1	1	imm5	
AND	0101	DR	SR1	0	00	SR2
AND	0101	DR	SR1	1	imm5	
NOT	1001	DR	SR	111111		
BR	0000	n	z	p	PCoffset9	
JMP	1100	0	00	BaseR	000000	
JSR	0100	1	PCoffset11			
JSRR	0100	0	00	BaseR	000000	
RET	1100	0	00	111	000000	

LD	0010	DR	PCoffset9	
LDI	1010	DR	PCoffset9	
LDR	0110	DR	BaseR	offset6
LEA	1110	DR	PCoffset9	
ST	0011	SR	PCoffset9	
STI	1011	SR	PCoffset9	
STR	0111	SR	BaseR	offset6
TRAP	1111	0000	trapvect8	
RTI	1000	000000000000		
reserved	1101			

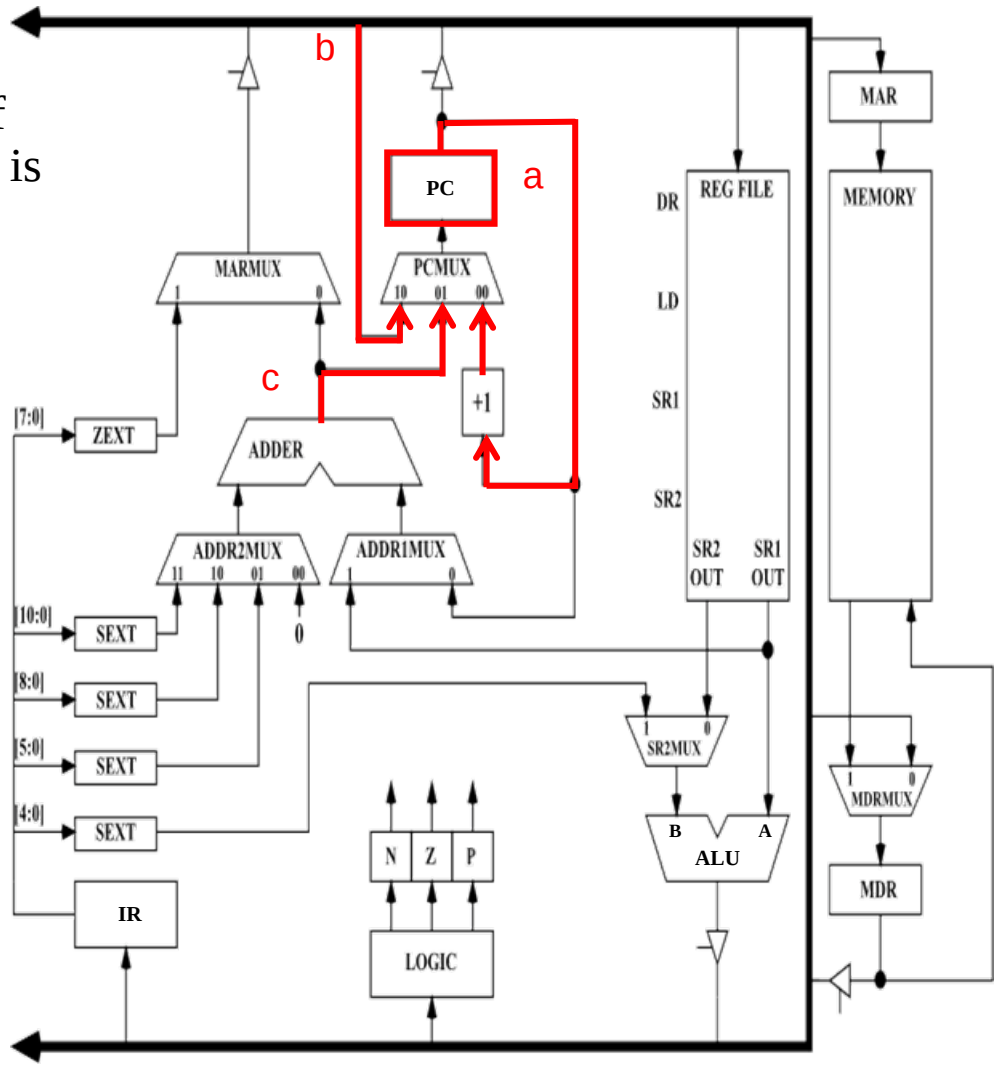
The LC-3 - EAB

- The *EAB*
  - Calculates effective addresses for the MAR and the PC
- It adds two operands that are selected by the control unit (a)
- One operand is zero or a sign extended field from the IR (10:0, 8:0, or 5:0) (b)
- The other operand is the current value of the PC or the contents of a register from the register file (c)
- The sum is passed to both the PCMUX and the MARMUX as an effective address (d)



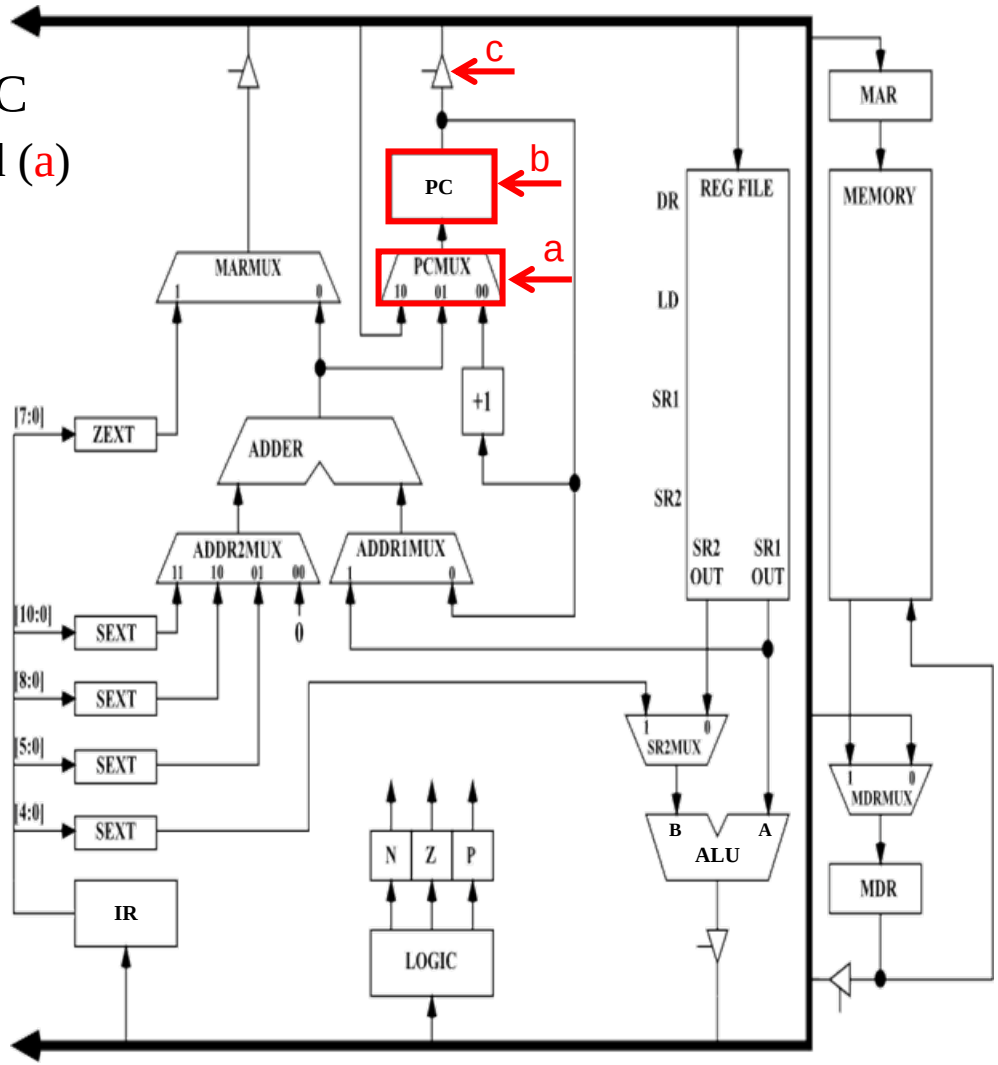
The LC-3 - PC and PCMUX

- The *Program Counter*
  - During the fetch and at the end of some control instructions, the PC is updated to point to the next instruction to be executed
- New PC Computation
  - Can be PC+1 (a)
  - Can come from global bus (b)
  - Can come EAB (c)



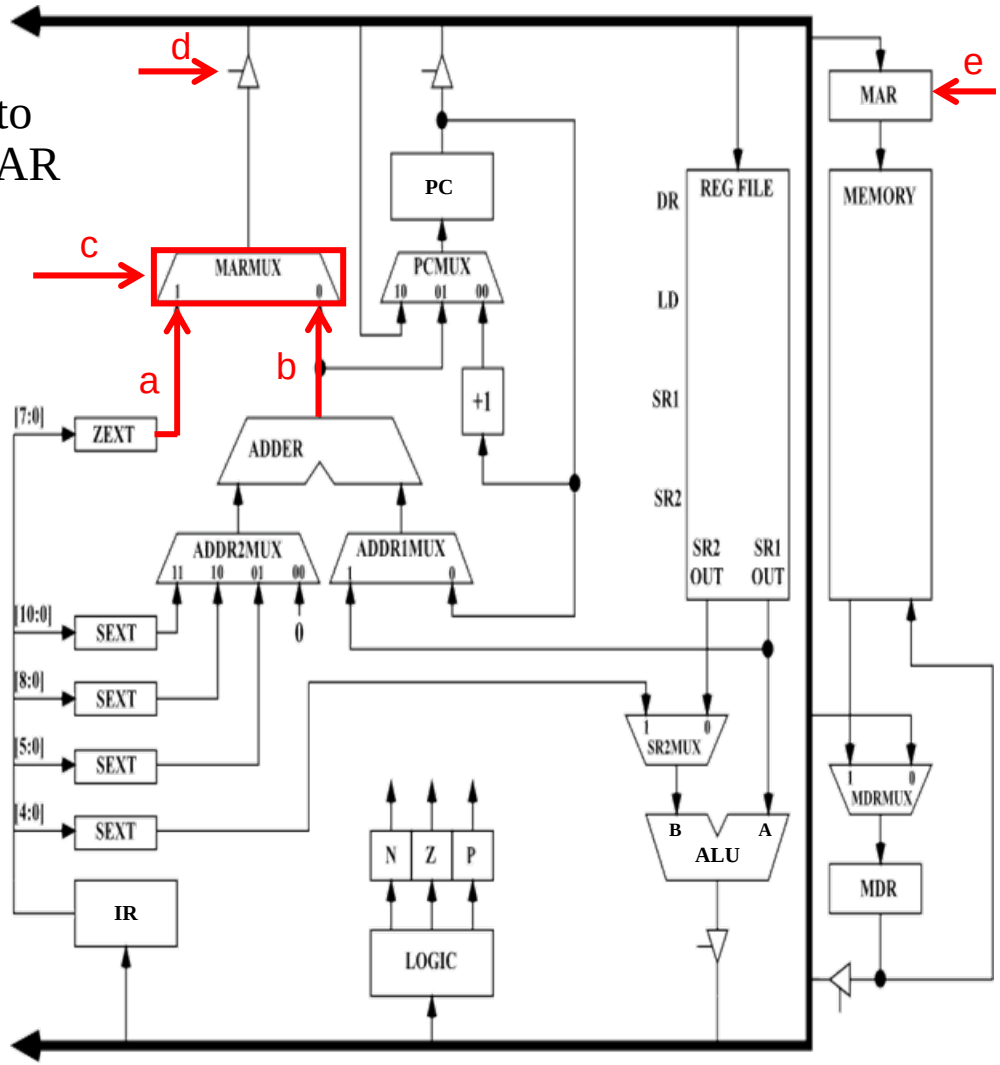
The LC-3 - PC and PCMUX

- Control unit controls loading of PC
  - selects which value it should load (a)
  - tells *when* PC should load a new value (b)
- Control unit tells PC when to drive onto global bus (c)



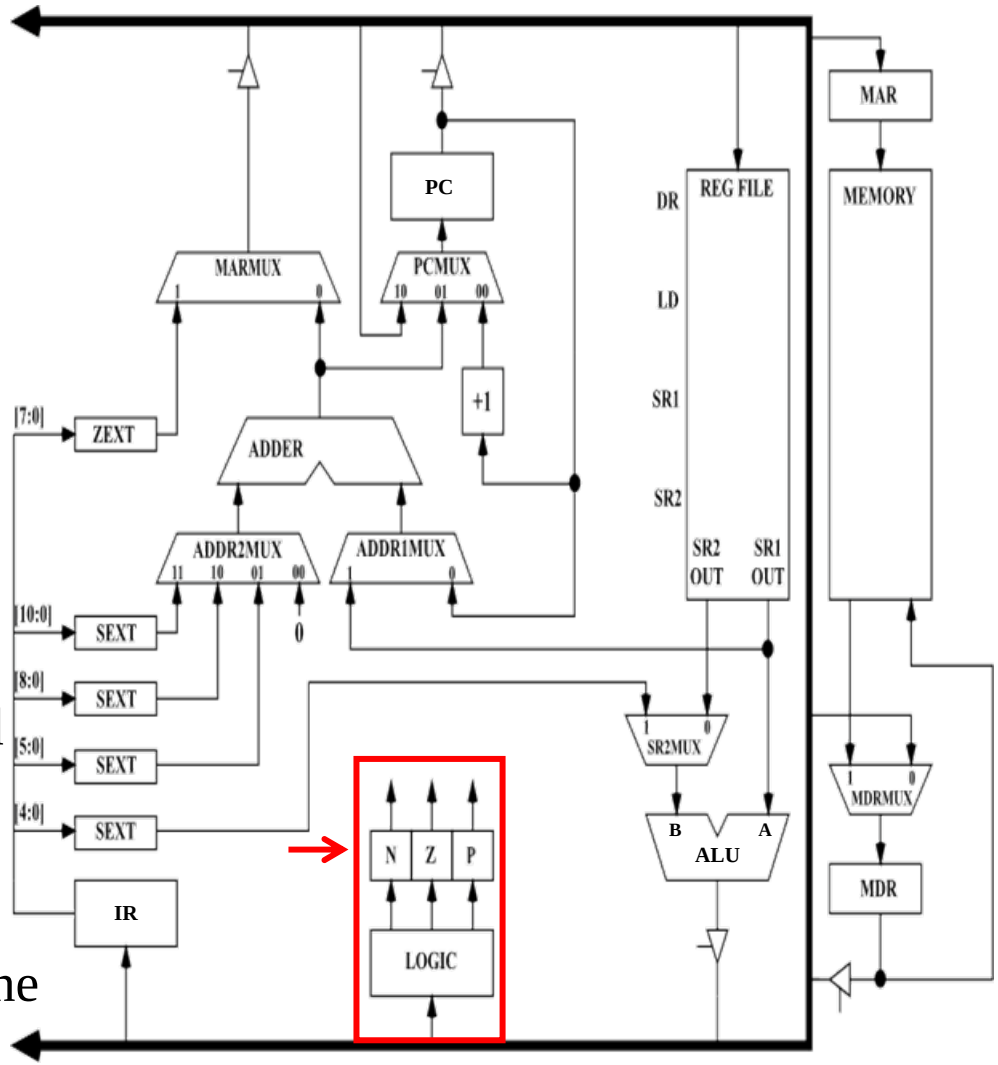
The LC-3 - MARMUX

- The *MARMUX*
  - Selects what address is driven onto global bus for loading into the MAR
- *MARMUX* Sources
  - Can be $IR_{7:0}$ zero extended (a)
    - for TRAP instructions
  - Can be output of EAB (b)
    - for base+offset
- Control unit selects source (c), controls driving it out onto global bus (d), and when MAR is loaded (e)



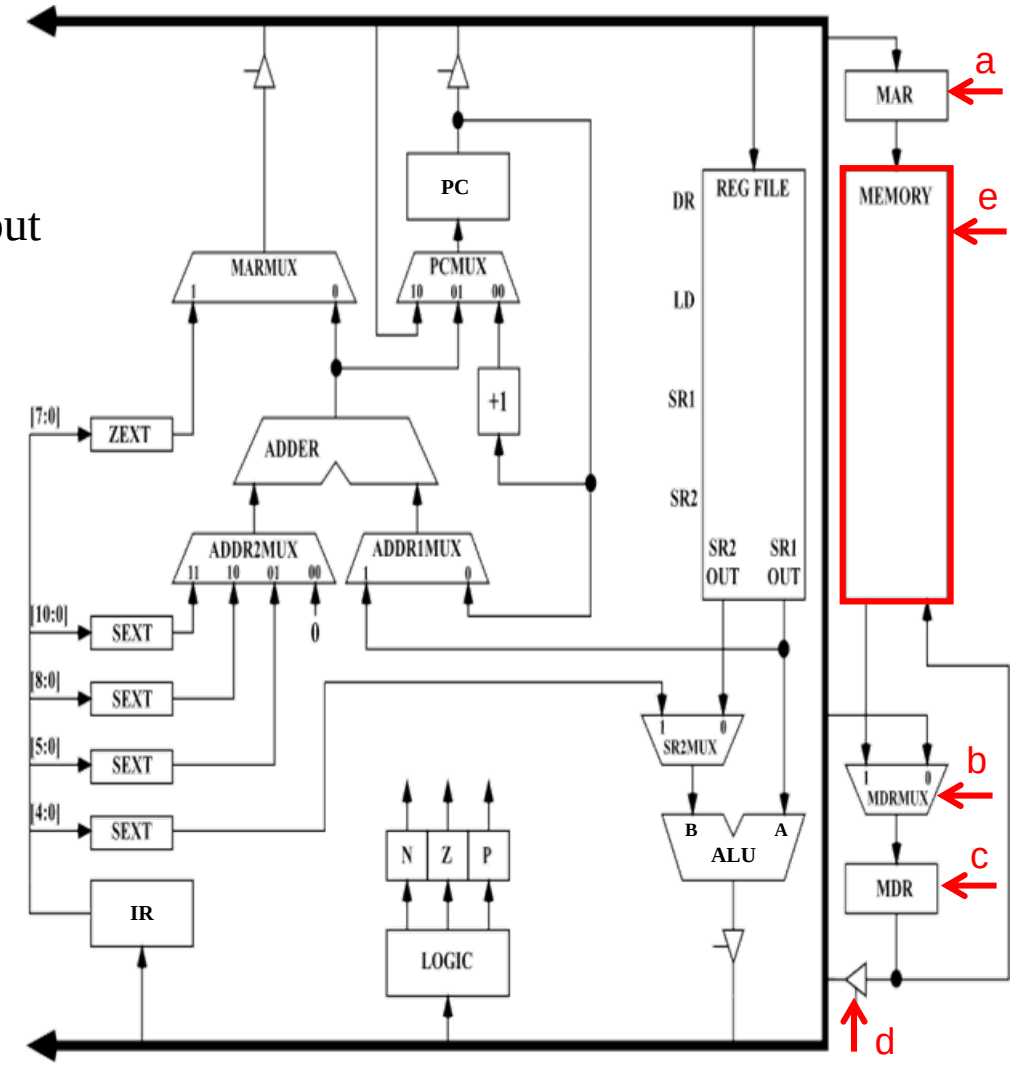
The LC-3 - N, Z, P Condition Codes

- The condition code registers
  - 1-bit each
- Logic block monitors global bus values
  - It continuously outputs whether bus value is negative, zero, or positive
- Control unit controls when N, Z, and P registers are actually loaded
  - They are loaded any time the register file is written to
- Control unit uses them to determine whether or not to branch on BR



The Memory

- On a read:
  - Address comes from MAR
  - Data is put into MDR and then out onto the bus
- On a write:
  - Address comes from MAR
  - Data to be written comes from MDR
- Control unit tells memory when to load MAR (a), what value to pass through the MDRMUX (b), when to load MDR (c), when to drive the value in the MDR onto global bus (d), and when to write to memory (e).

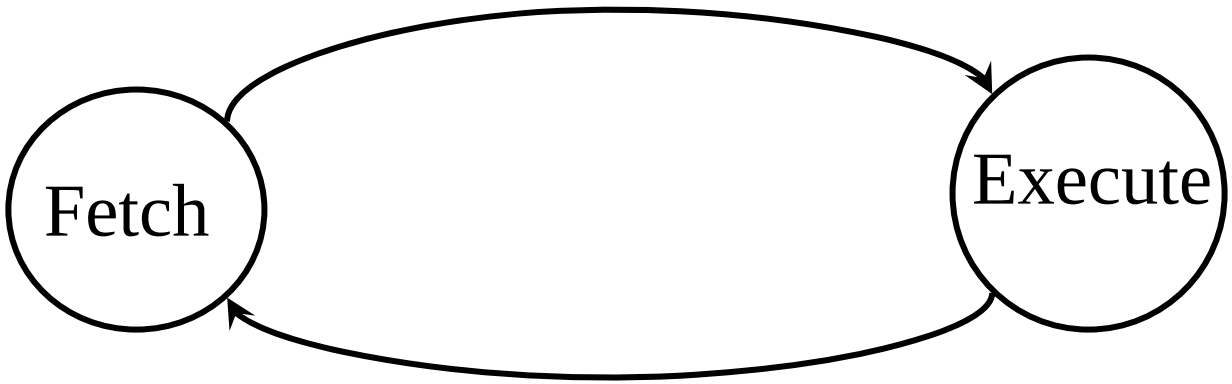


Data Flow

(Tracing Data And The Execution of Instructions Through LC-3)

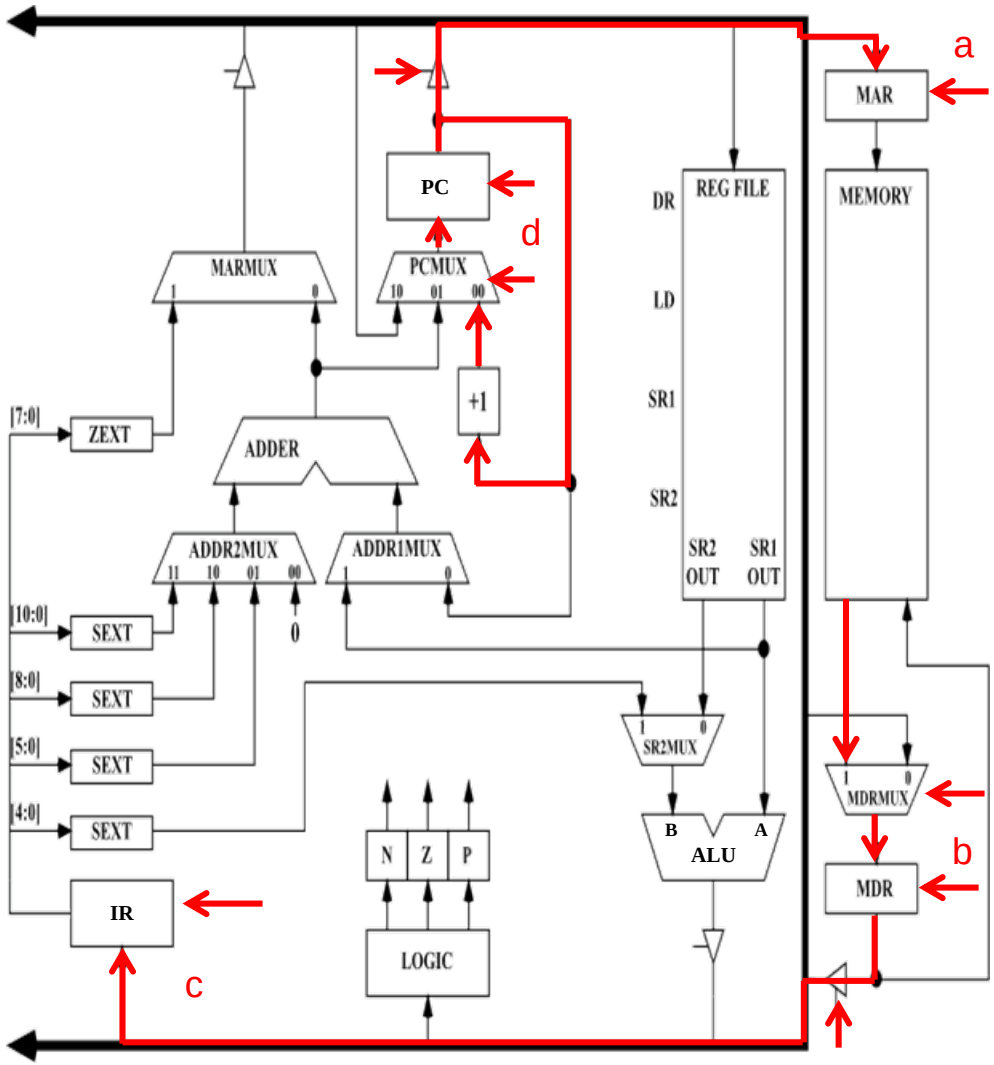
The Von Neumann Model

- Fetch an instruction
- Execute it
- Repeat

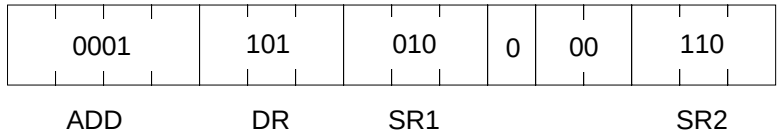


Instruction Fetch

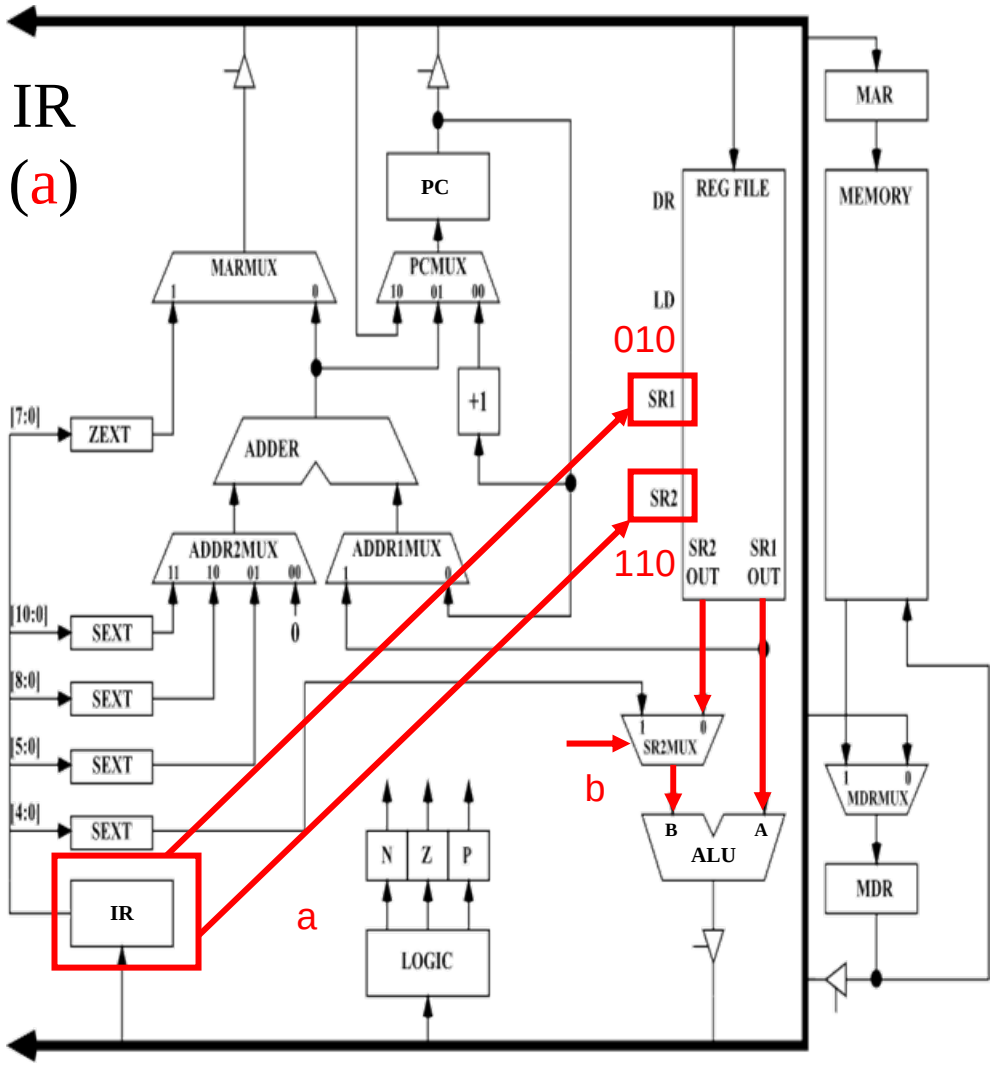
- Copy the PC into the MAR (a)
- Load Memory Output into MDR (b)
- Load Output of MDR into IR (c)
- Increment PC (d)



Operand Selection

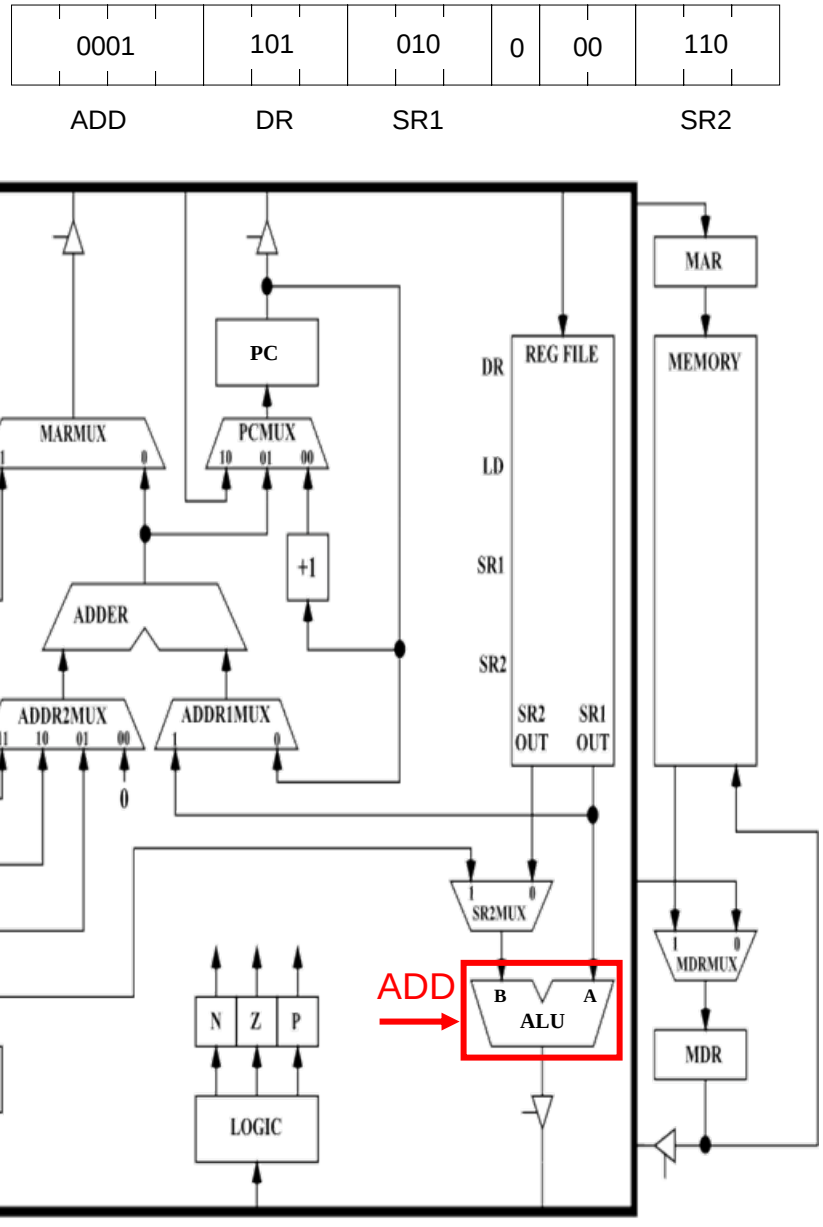


- Send SR1 and SR2 fields from IR as addresses to the register file (a)
- Retrieve values addressed by SR1 and SR2 and send to ALU for execution (b)



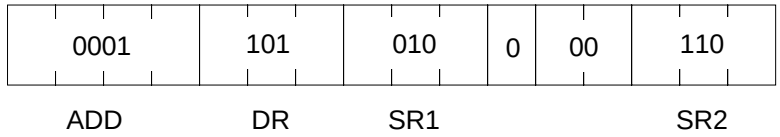
Execute

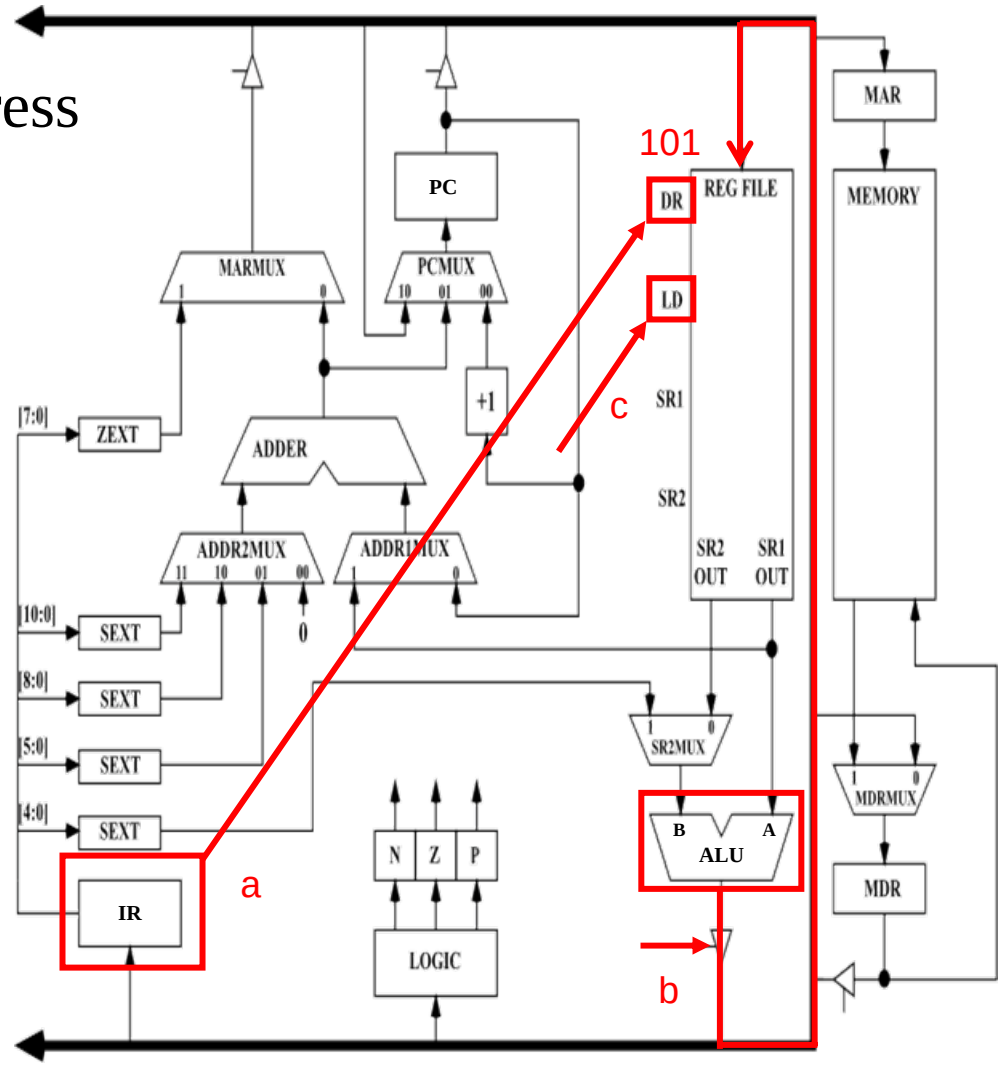
- The ALU does the addition
  - control unit tells it which operation to do (ADD)



Store Result

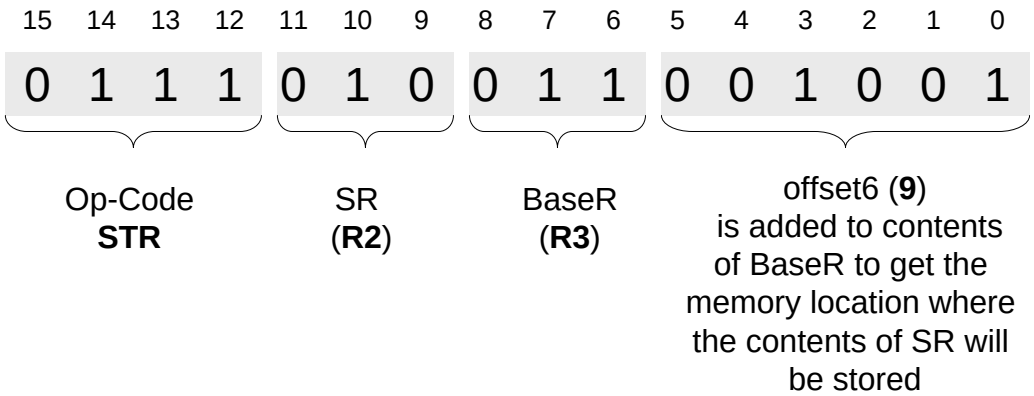
- Send DR field from IR as address to the register file (a)
- Enable ALU output to pass onto the bus (b)
- Store ALU output into DR by enabling register file load (c)





Another Example Instruction

- **STR R2 R3 9**
- Numbers must already be in registers



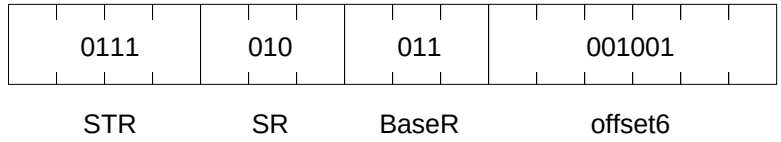
mem [R3 + 9] = R2

EffectiveMemoryAddress <= R3 + 9
MEM[EffectiveMemoryAddress] = R2

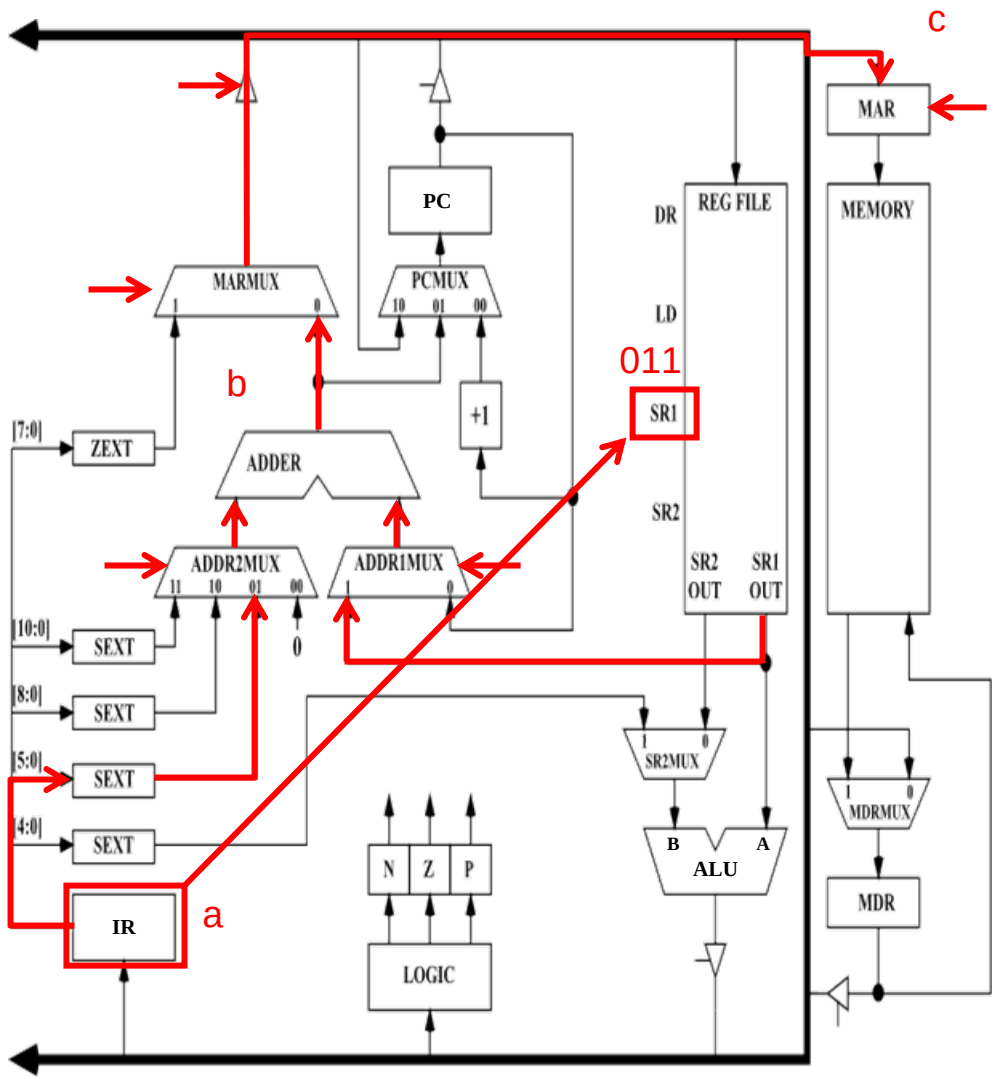
Instruction Fetch

Same as ADD Instruction

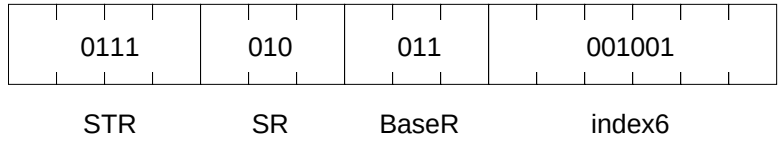
STR - key parts



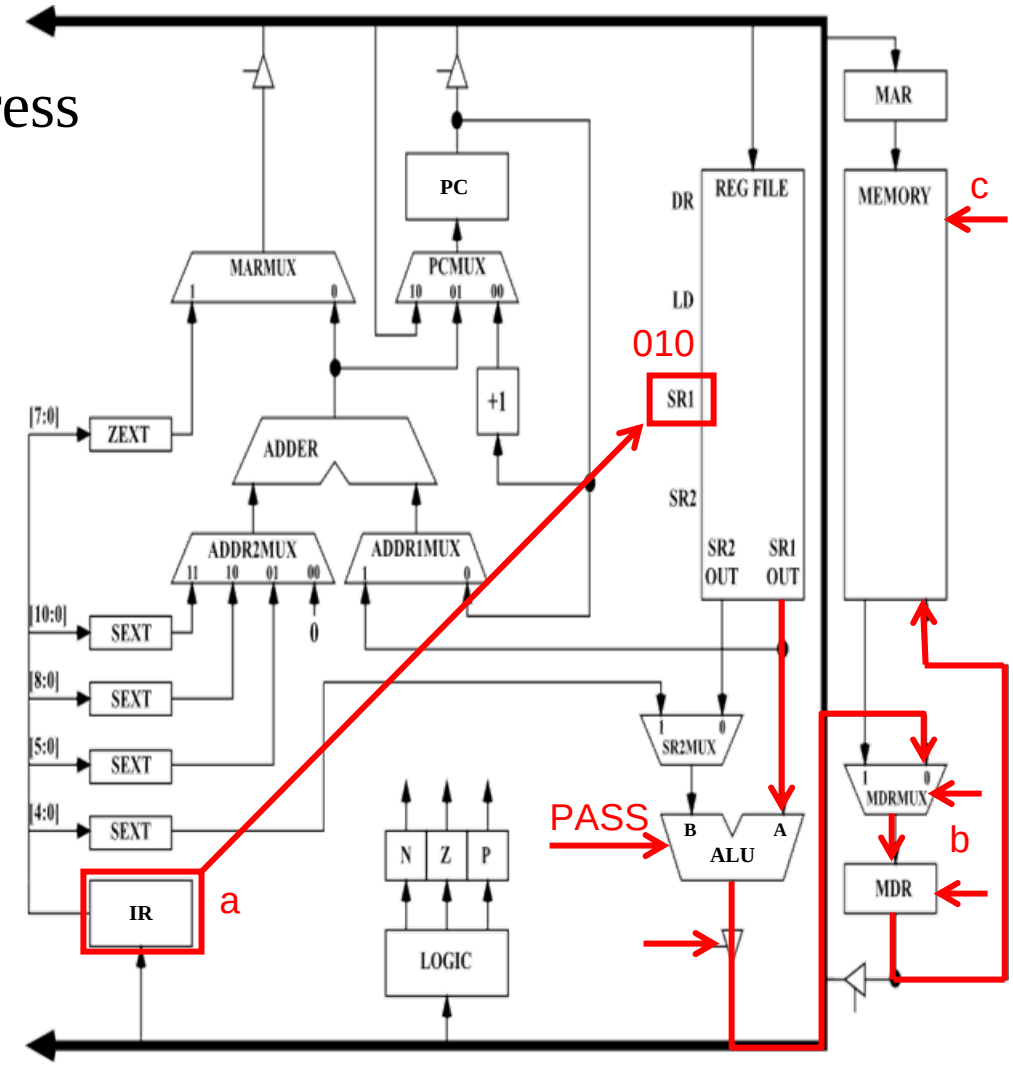
- Send BaseR field from IR as address to the register file (a)
- Add the contents of BaseR to the sign extended offset6 from the IR to form the destination memory address for the STR (b)
- Store the generated address into the MAR (c)



STR - key parts

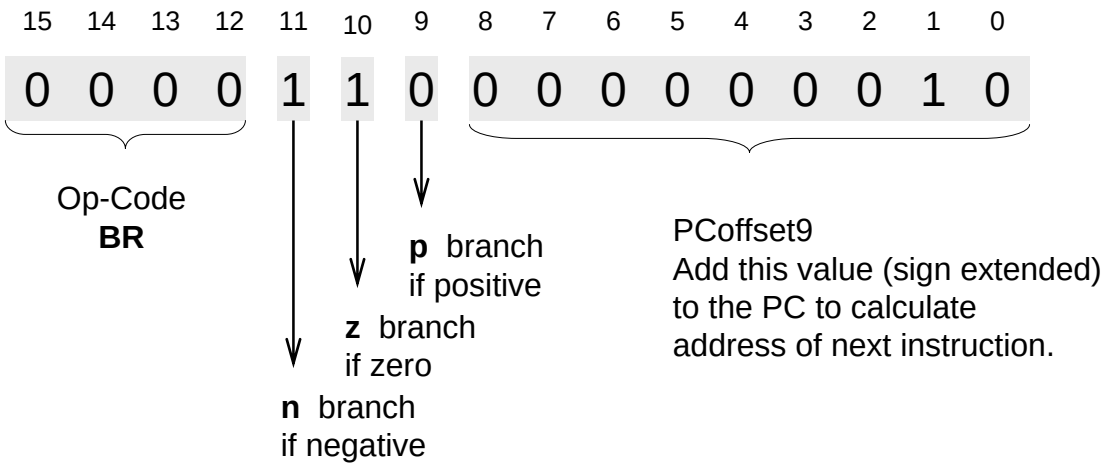


- Send SR field from IR as address to the register file (a)
- Store the contents of SR to the MDR (b)
- Perform the memory write (c)



Another Example Instruction

- **BRnz LABEL**
- Condition Codes loaded by previous instruction



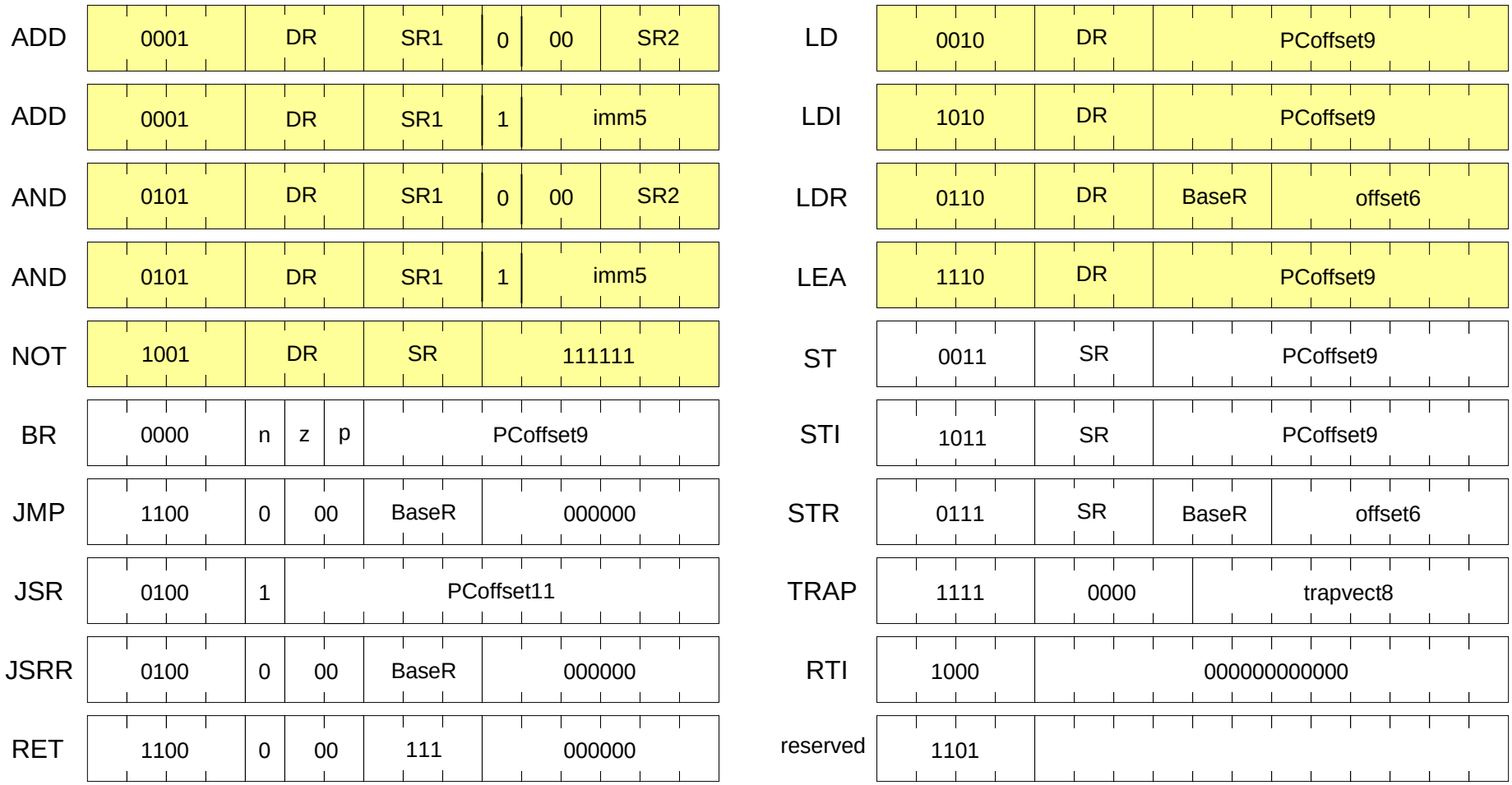
if (n AND N) OR (z AND Z)

PC = PC + PCoffset9

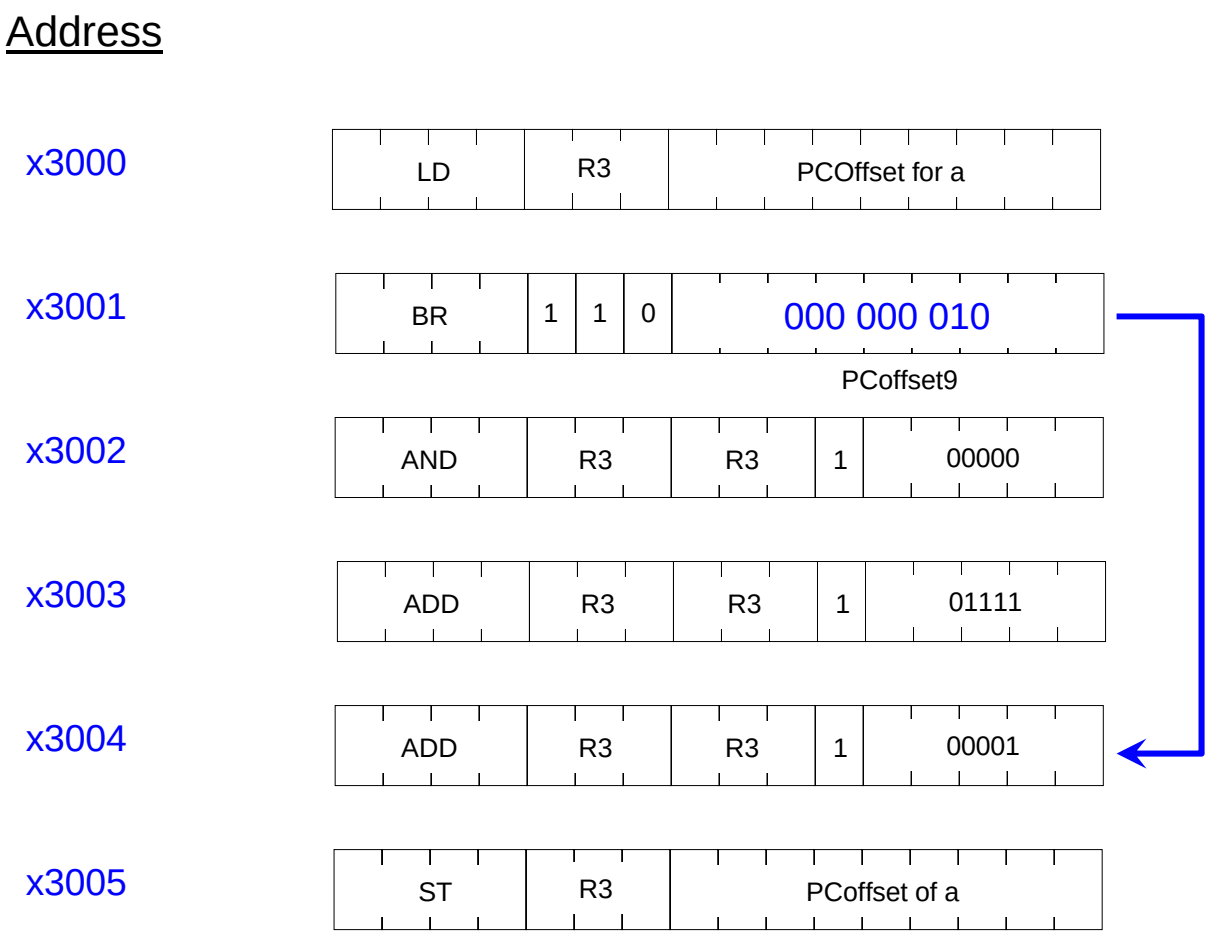
What are Condition Codes ?

- LC-3 contains 3 special registers
  - 1-bit wide each
  - named N, Z, P (negative, zero, positive)
- On some instructions, when a register is loaded with a new value
  - N, Z, and P are updated to reflect the value there
- Only specific instructions modify the condition codes
  - See back cover of 124 book to be sure

All Instructions



An if Statement Using BR

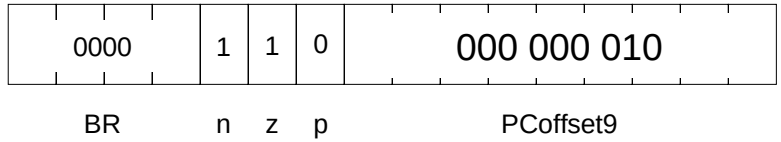



```
if (a > 0)
        a = 15 ;
a = a + 1 ;
```

BRnz Instruction Fetch

Same as ADD Instruction

BRnz - Execution



- Compare n and z in IR to N and Z registers
- Generate branch address
  PC + (sign extend) PCoffset9 (a)
- Pass new address through the PCMUX (b)
- Load branch address into PC iff the condition codes match (c)