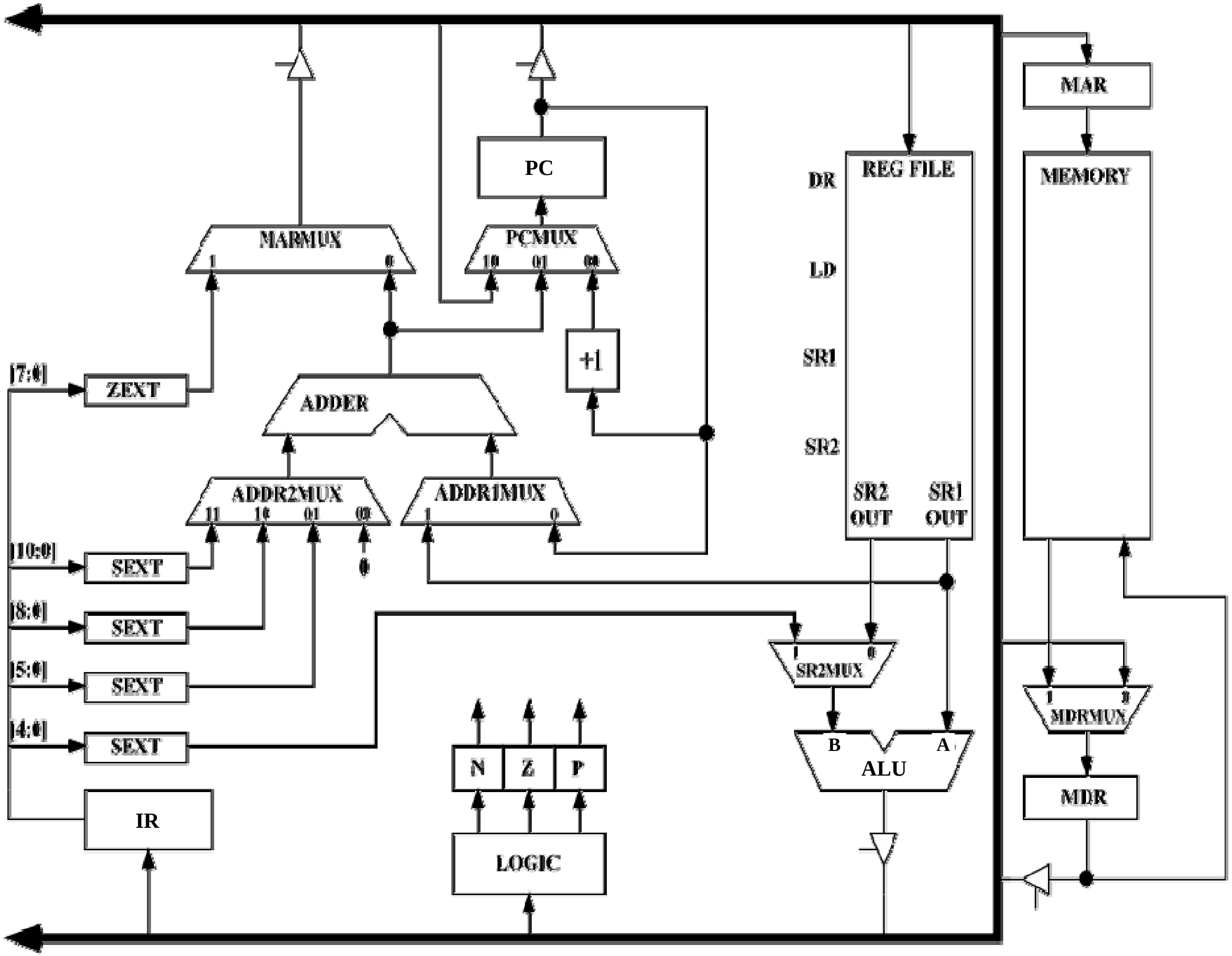
LC3-1

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

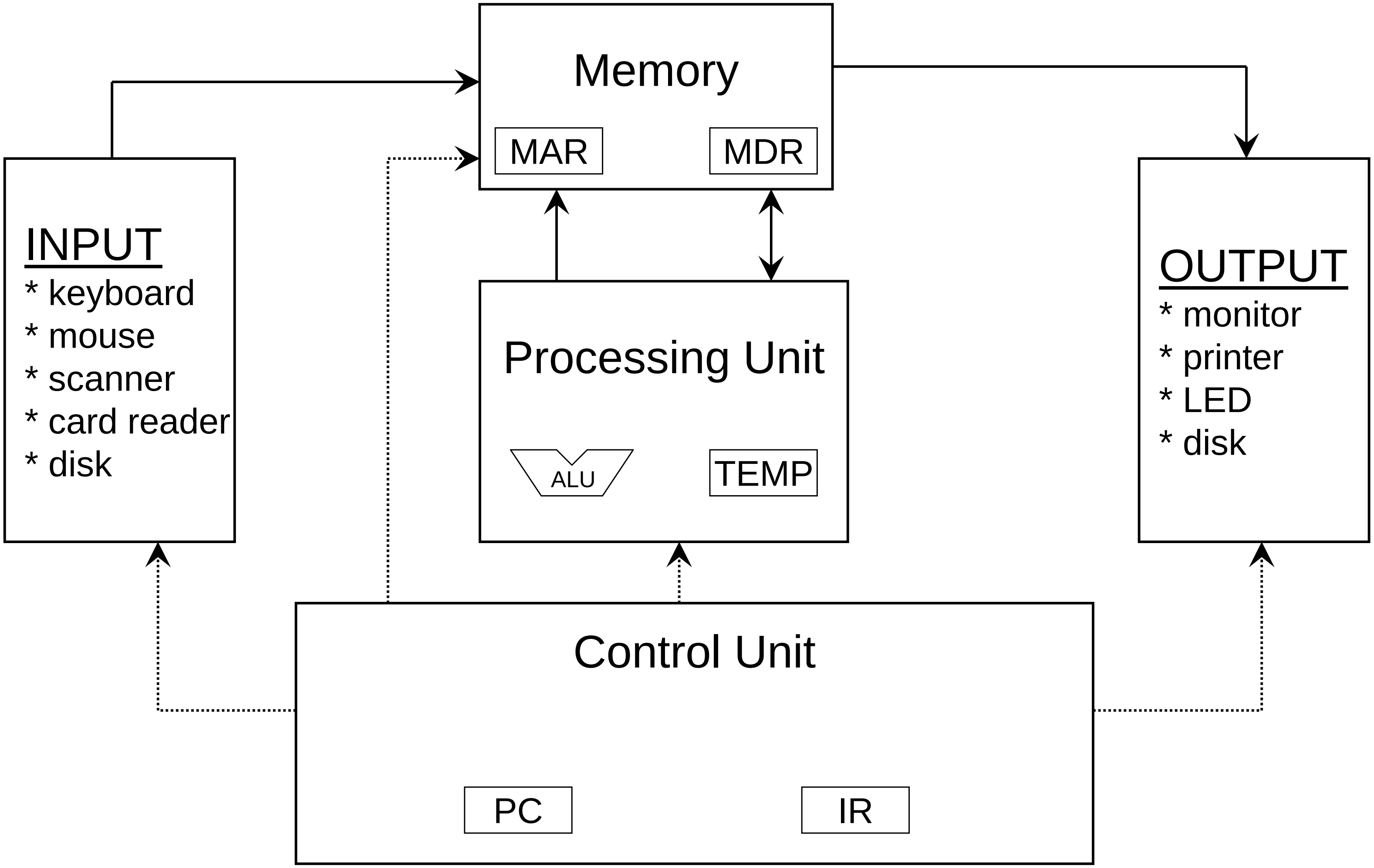
Reference Information

- You will *need* the LC-3 Description
- ECEn 124 / CS 124 Textbook:

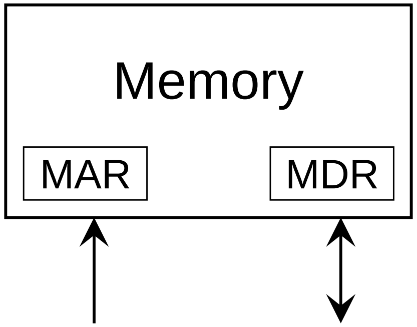
  "Introduction to Computing Systems" (second edition)
  Yale N. Patt & Sanjay J. Patel
  McGraw-Hill Higher Education 2004

- Useful Sections (in order of importance):
  - Appendix A (available on class webpage)
  - Chapter 5
  - Chapter 4
- Reference material available on the class webpage

The Von Neumann Model

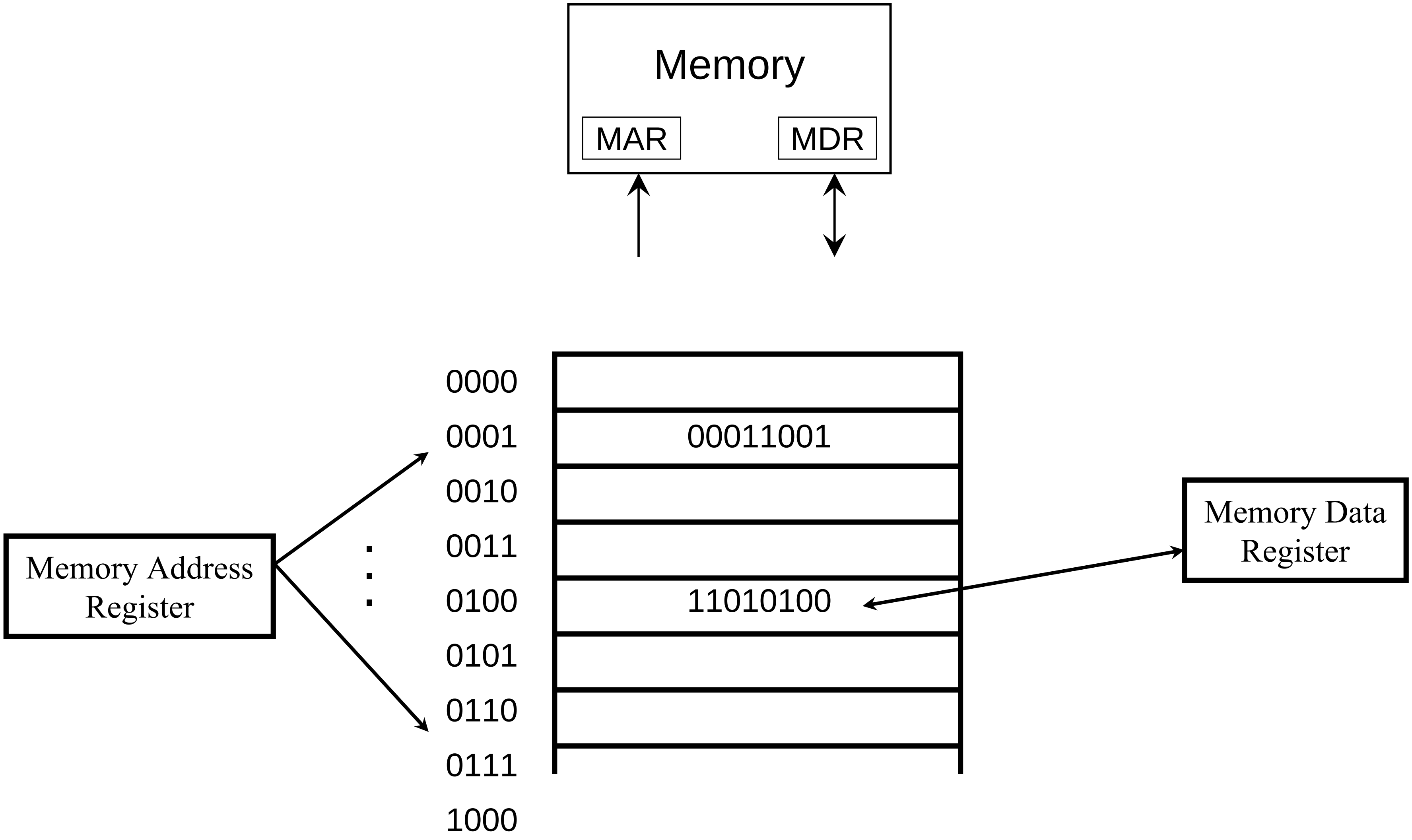


The Von Neumann Model



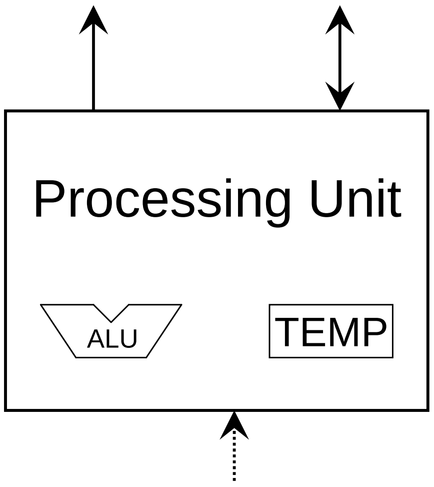
- Memory is used to store a sequence of instructions
- Memory is also used to store data
- Memory Address Register (MAR) selects which location in memory will be read or written
- Memory Data Register (MDR) contains the data read or to be written

The Von Neumann Model



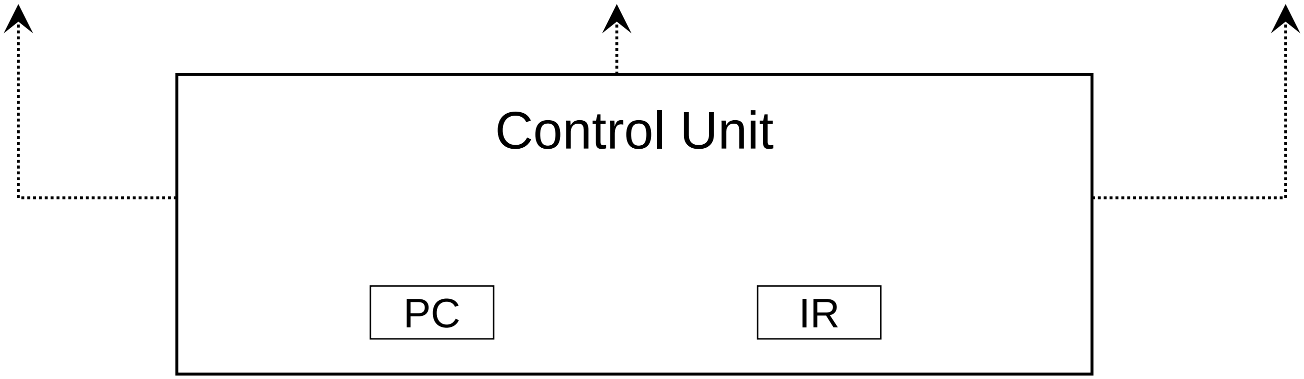
The Von Neumann Model

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

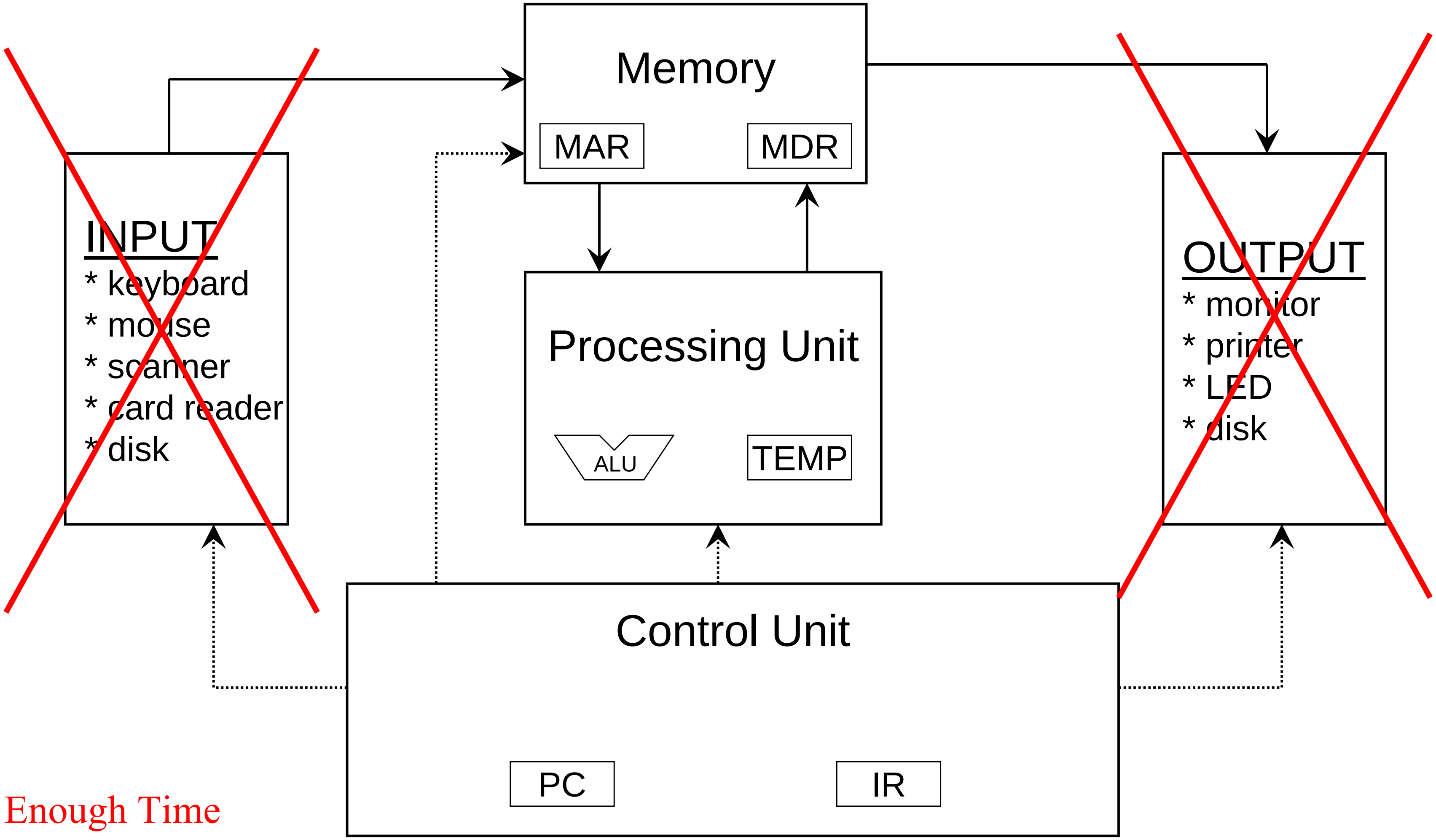


The Von Neumann Model

- 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

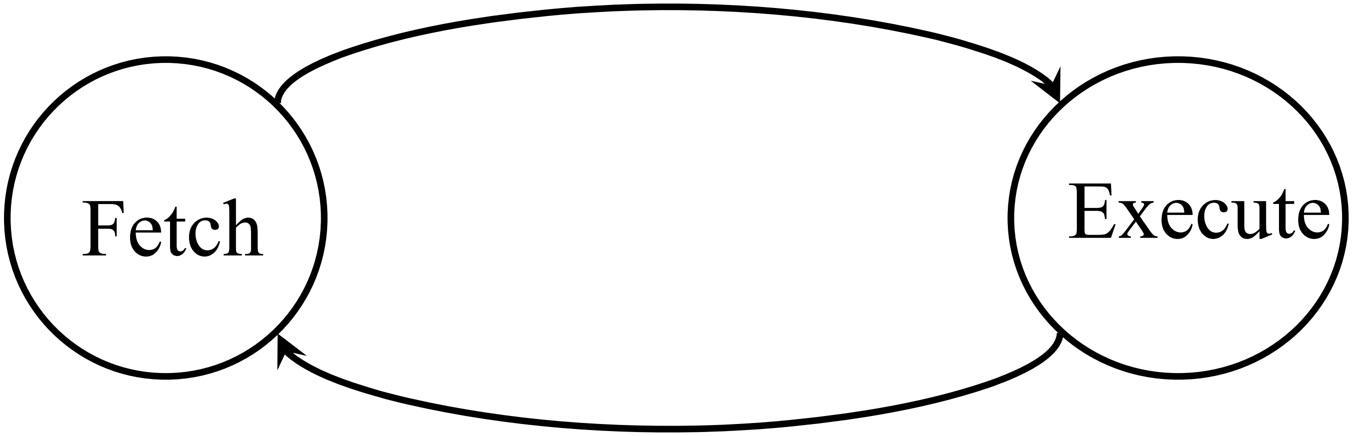


Not Enough Time
to Study Everything

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 a computer
    - Details of how to implement the ISA in hardware are left up to the designer's imagination

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
    - The LC-3 is not byte addressable, unlike most machines

Register Set

- Memory access is relatively slow
  - It is outside the processing unit
  - It requires completion of an instruction to access (LDR)
- Registers are *inside* the processing unit
  - They can be accessed *during* an instruction (ADD)
- Nearly all computers have a *register set*
  - LC-3 has 8 general purpose registers
  - Named R0, R1, …, R7
  - They are addressed with a 3-bit field in an instruction

Data Types

- LC-3 has only one data type
  - 16-bit <u>two's complement</u> integer

- Other computers have others
  - 32-bit floating point (*float*)
  - 64-bit floating point (*double*)
  - 32-bit signed/unsigned (*int*)
  - 16-bit signed/unsigned (*short*)
  - 8-bit signed/unsigned (*char*)
  - Possibly more…

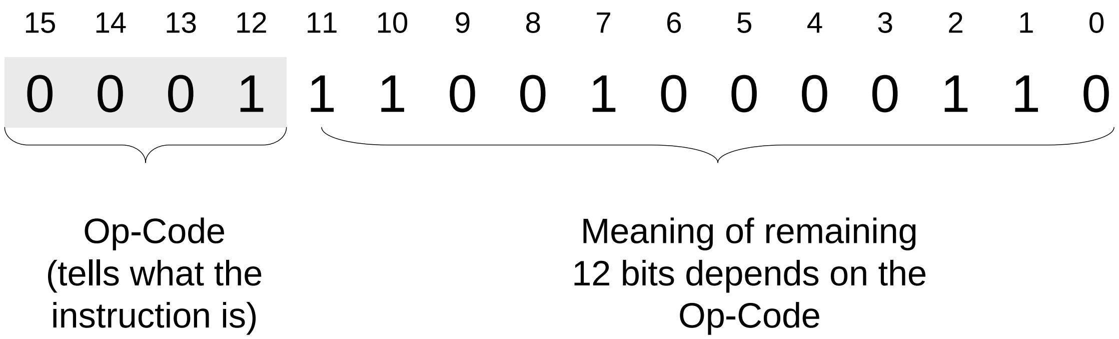
↑ These names are system dependent

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



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

Op-Code
(tells what the
instruction is)

Meaning of remaining
12 bits depends on the
Op-Code

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

ADD	0001	DR	SR1	0	00	SR2	DR = SR1 + SR2
ADD	0001	DR	SR1	1	imm5		DR = SR1 + SEXT(imm5)
AND	0101	DR	SR1	0	00	SR2	DR = SR1 AND SR2
AND	0101	DR	SR1	1	imm5		DR = SR1 AND SEXT(imm5)
NOT	1001	DR	SR	111111			DR = NOT(SR)

An Operate Instruction

ADD R6, R2, R5

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

Op-Code	**DR**	**SR1**		**SR2**
Tells what the instruction is	Where the result Is stored	Where the 1st operand comes from	unused in this instruction	Where the 2nd operand comes from
ADD	**R6**	**R2**		**R5**

R6 = R2 + R5

The Data Movement Instructions

LD	0010	DR	PCoffset9		DR = mem [PC + SEXT(PCoffset9)]
LDI	1010	DR	PCoffset9		DR = mem [mem [PC + SEXT(PCoffset9)]]
LDR	0110	DR	BaseR	offset6	DR = mem [BaseR + SEXT(offset6)]
LEA	1110	DR	PCoffset9		DR = PC + SEXT(PCoffset9)
ST	0011	SR	PCoffset9		mem [PC + SEXT(PCoffset9)] = SR
STI	1011	SR	PCoffset9		mem [mem [PC + SEXT(PCoffset9)]] = SR
STR	0111	SR	BaseR	offset6	mem [BaseR + SEXT(offset6)] = SR

An LDR Instruction

LDR R2, R3, 6

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

Op-Code	**DR**	**BaseR**	**Offset6**
Tells what the instruction is	Where the value fetched from memory will be placed	Where the base address comes from	Added to contents of BaseR to generate fetch memory address
LDR	**R2**	**R3**	**6**

Offset is sign-extended before being added to base

EffectiveMemoryAddress <= R3 + 6

R2 = MEM[EffectiveMemoryAddress]

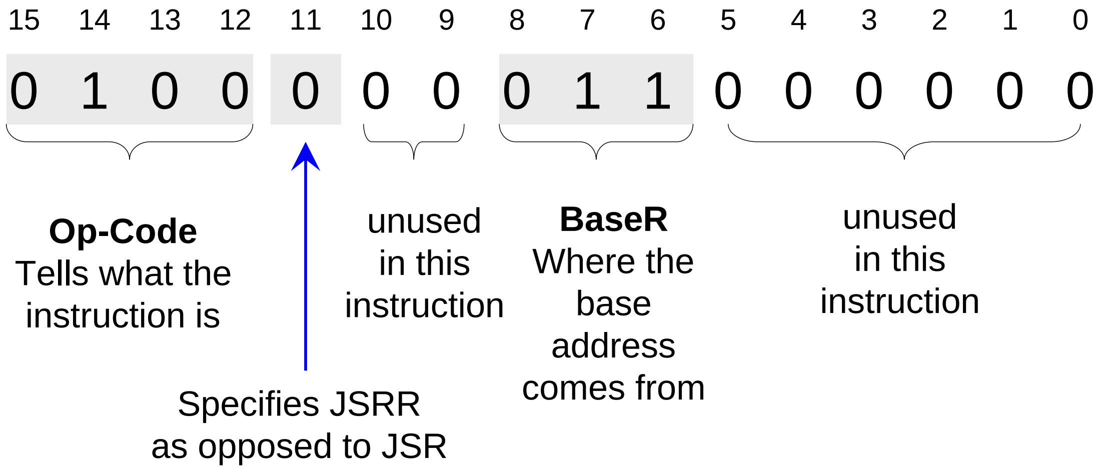
This requires the computation of an effective memory address. It is base + offset. The contents of R3 are the base address and 6 is the offset.

Control Instructions

<table>
<tr><td>BR</td><td colspan="4">0000</td><td>n</td><td>z</td><td>p</td><td colspan="9">PCoffset9</td><td colspan="2">PC = PC + SEXT(PCoffset9)
depending on condition(s)</td></tr>
<tr><td>JSR</td><td colspan="4">0100</td><td>1</td><td colspan="11">PCoffset11</td><td colspan="2">R7 = PC
PC = PC + SEXT(PCoffset11)</td></tr>
<tr><td>JSRR</td><td colspan="4">0100</td><td>0</td><td colspan="2">00</td><td colspan="3">BaseR</td><td colspan="6">000000</td><td colspan="2">R7 = PC
PC = BaseR</td></tr>
<tr><td>RET</td><td colspan="4">1100</td><td colspan="3">000</td><td colspan="3">111</td><td colspan="6">000000</td><td>PC = R7</td><td rowspan="2">Different name, same instruction</td></tr>
<tr><td>JMP</td><td colspan="4">1100</td><td colspan="3">000</td><td colspan="3">BaseR</td><td colspan="6">000000</td><td>PC = BaseR</td></tr>
<tr><td>RTI</td><td colspan="4">1000</td><td colspan="12">000000000000</td><td colspan="2" rowspan="2">Probably won’t have time to implement these</td></tr>
<tr><td>TRAP</td><td colspan="4">1111</td><td colspan="4">0000</td><td colspan="8">trapvect8</td></tr>
</table>

A JSRR Instruction

JSRR R3



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

Op-Code
Tells what the instruction is

Specifies JSRR as opposed to JSR

unused in this instruction

BaseR
Where the base address comes from

unused in this instruction

JSRR R3

R7 <= PC
PC <= R3

This is how a subroutine call would be executed.

A JMP Instruction

JMP R3

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0
Op-Code Tells what the instruction is				unused in this instruction			**BaseR** Where the base address comes from			unused in this instruction					

JMP **R3**

PC <= R3

This is how a GOTO statement would be executed.

The LC-3 Architecture

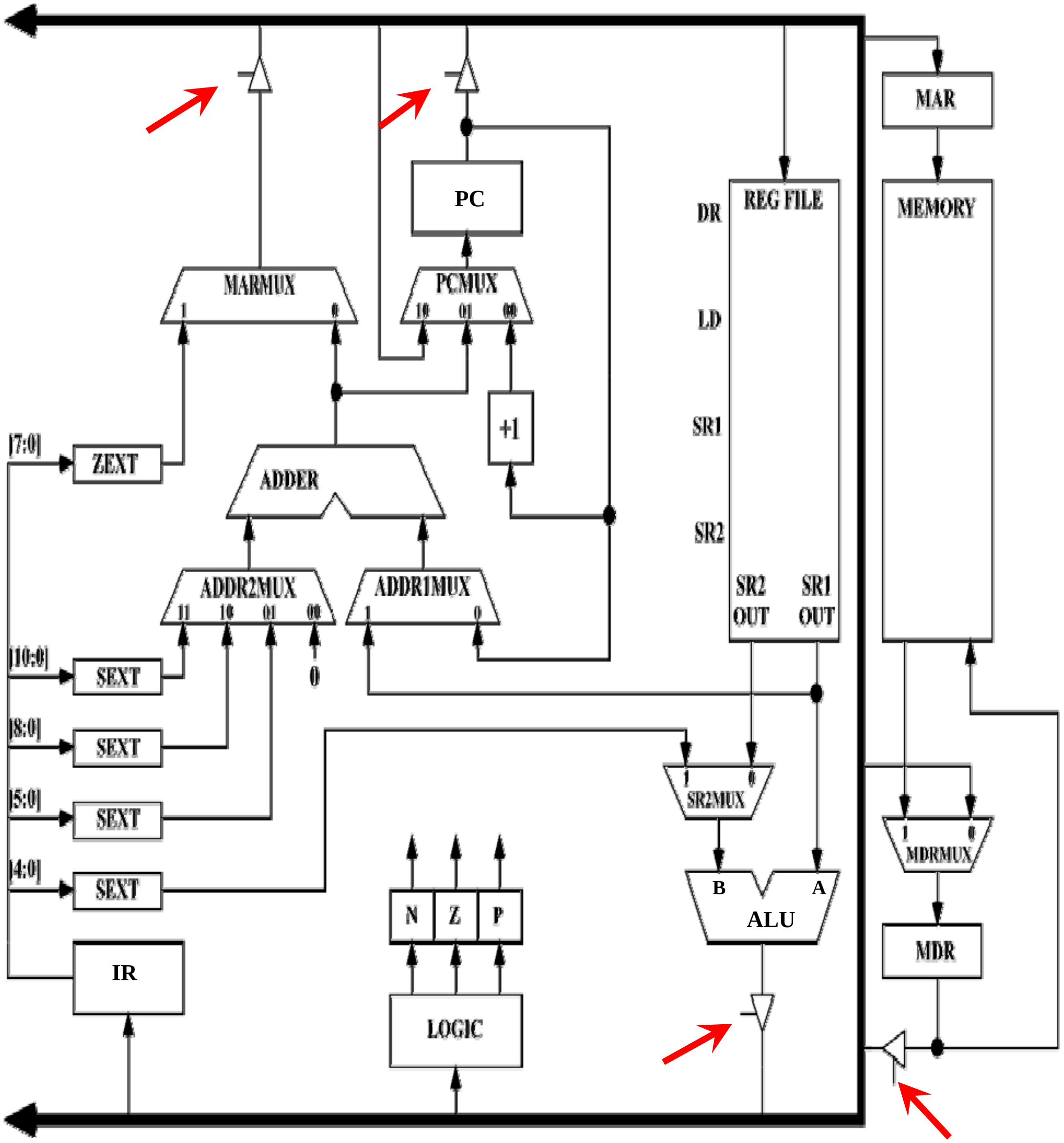
A More Detailed Look

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

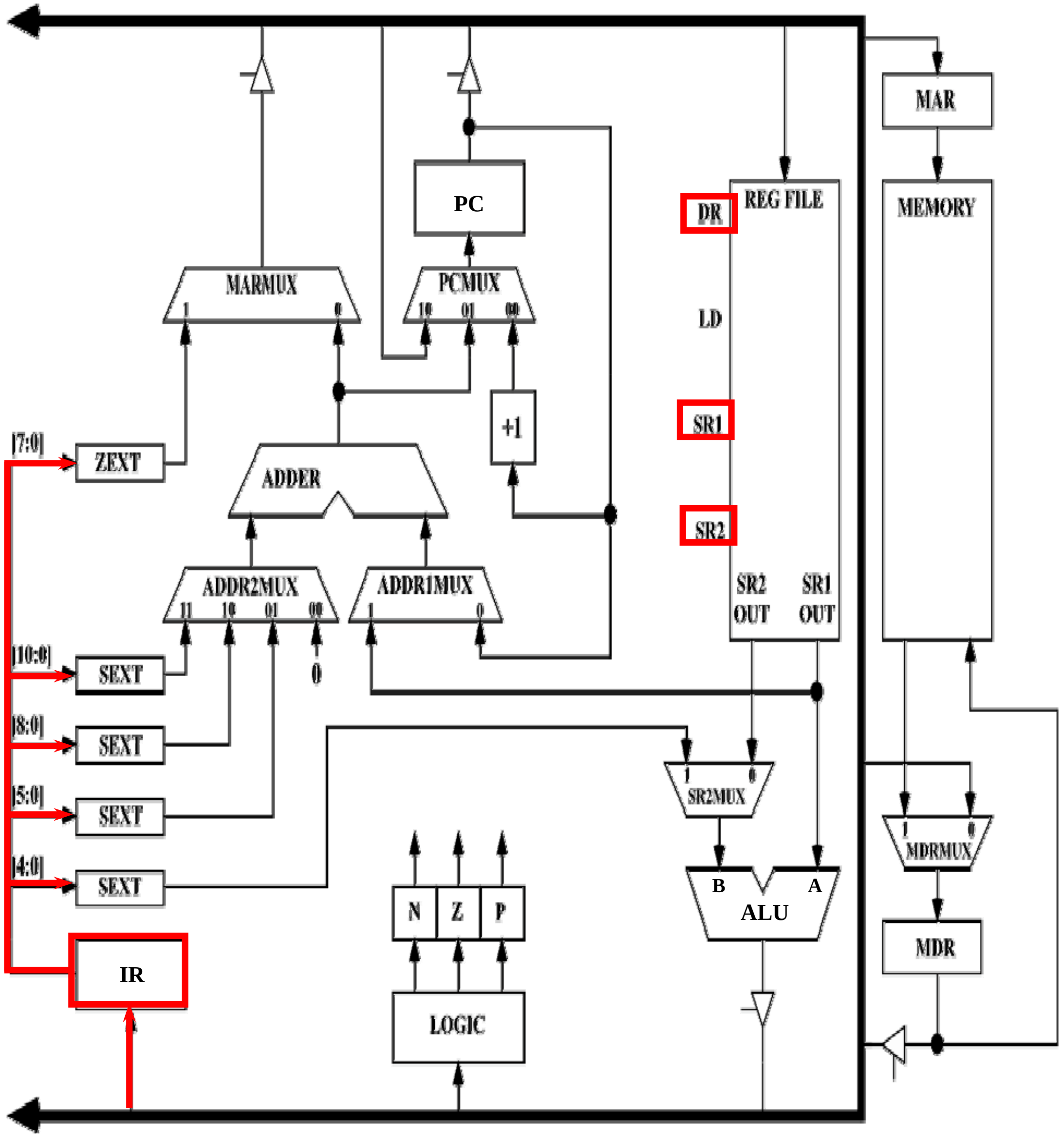
- A *tri-state* driver
  - Can drive 1's and 0's on the bus
  - Can disconnect from the bus

- Control unit turns them on and off



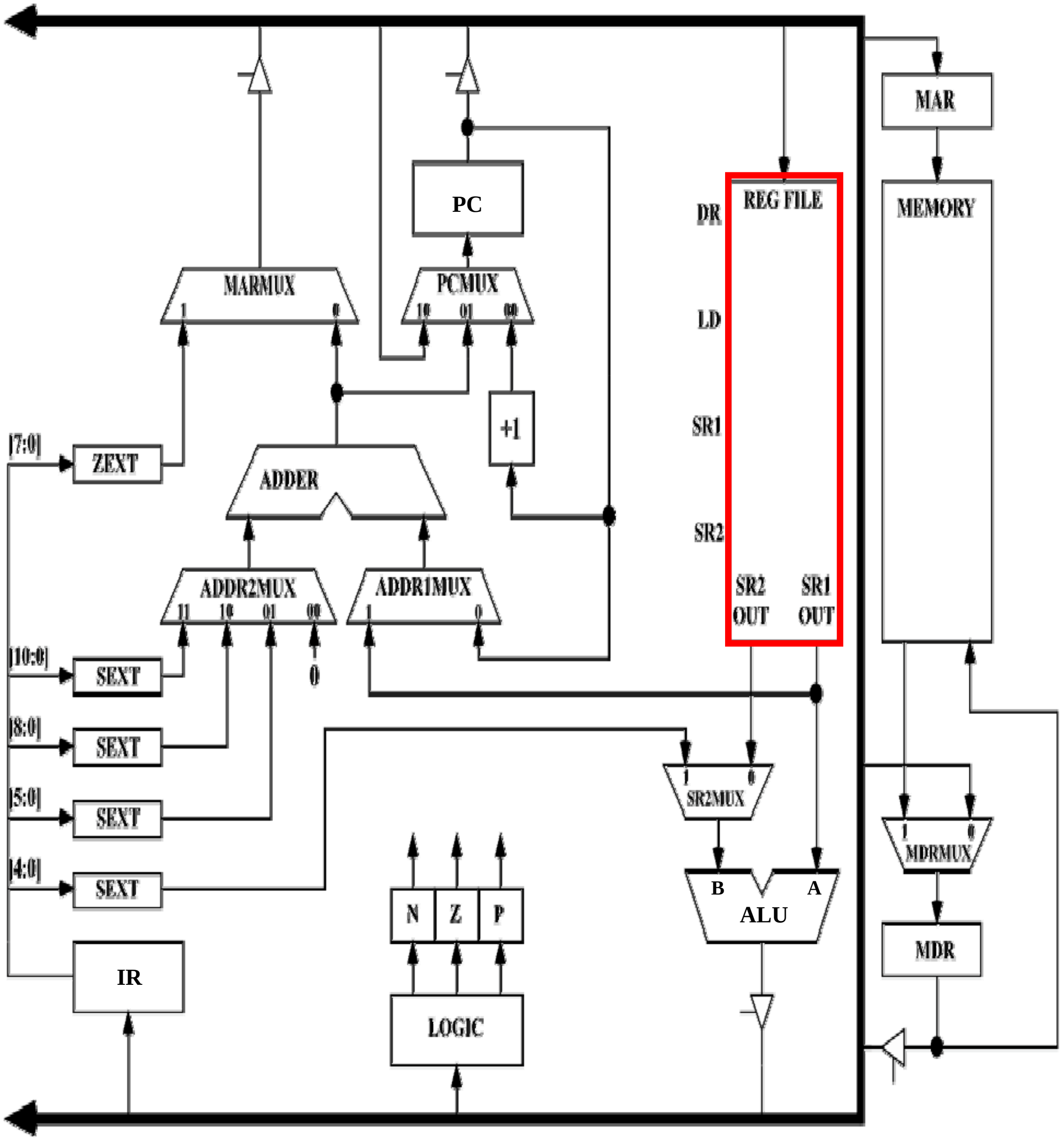
The LC-3 – Instruction Register (IR)

- 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



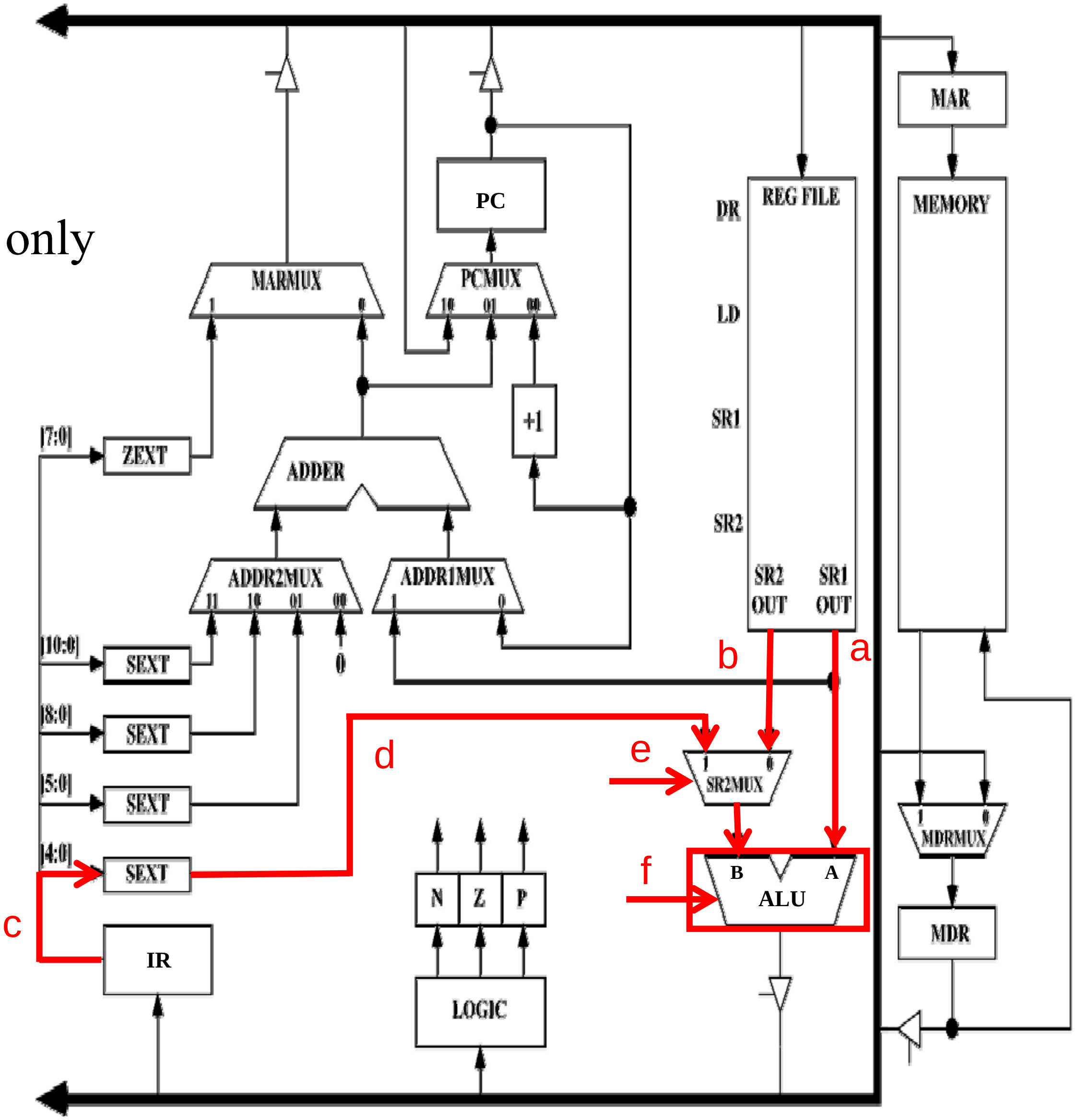
The LC-3 – Registers

- 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

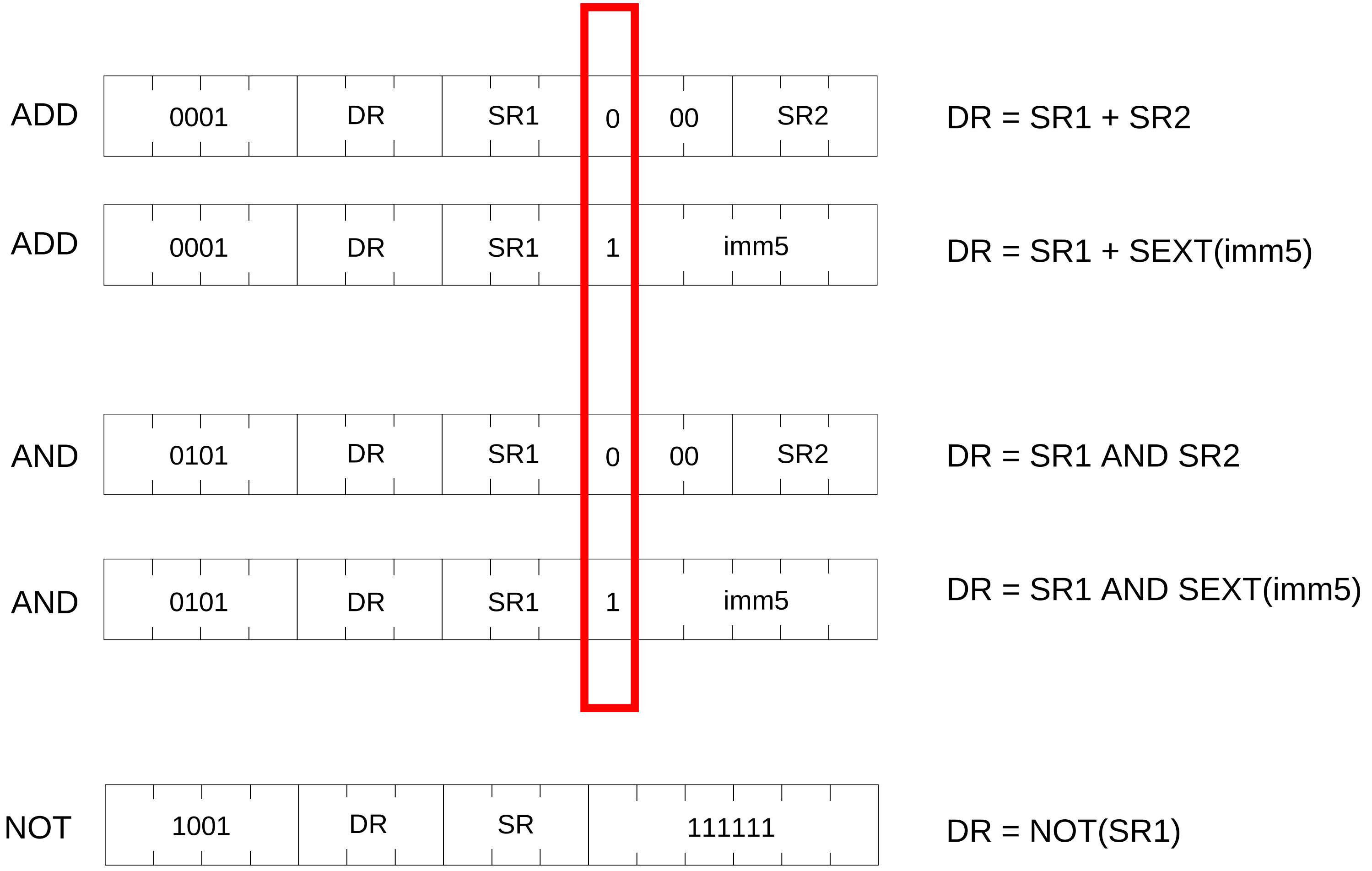


The LC-3 – ALU

- 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)



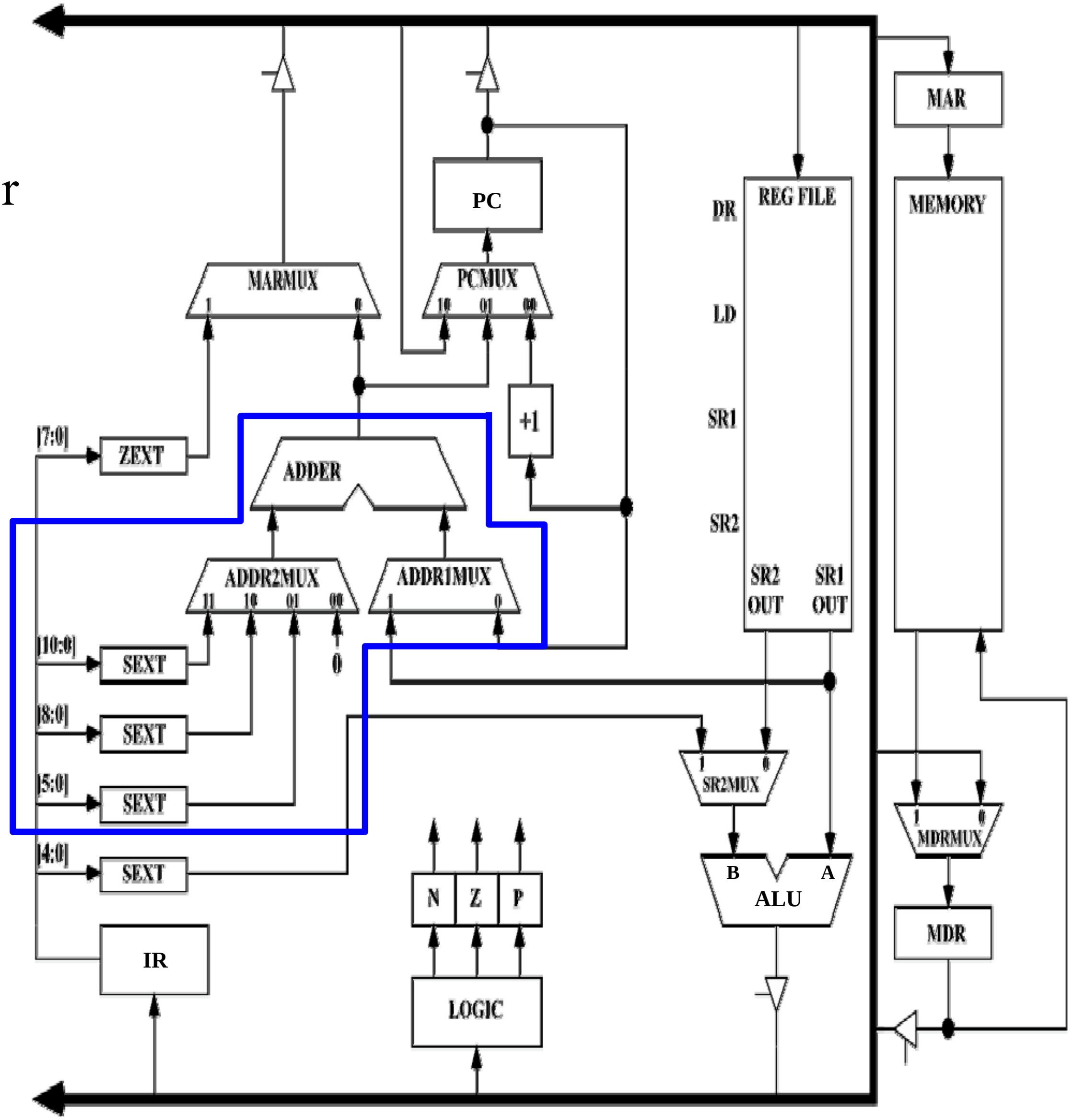
The Operate Instructions



ADD	0001	DR	SR1	0	00	SR2	DR = SR1 + SR2
ADD	0001	DR	SR1	1	imm5		DR = SR1 + SEXT(imm5)
AND	0101	DR	SR1	0	00	SR2	DR = SR1 AND SR2
AND	0101	DR	SR1	1	imm5		DR = SR1 AND SEXT(imm5)
NOT	1001	DR	SR	111111			DR = NOT(SR1)

The LC-3 – Effective Address Block (EAB)

- The *EAB* (Effective Address Block)
  - Calculates effective addresses for the MAR and the PC



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

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

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			

BYU
Computer Engineering
Electrical Engineering

The LC-3 – EAB

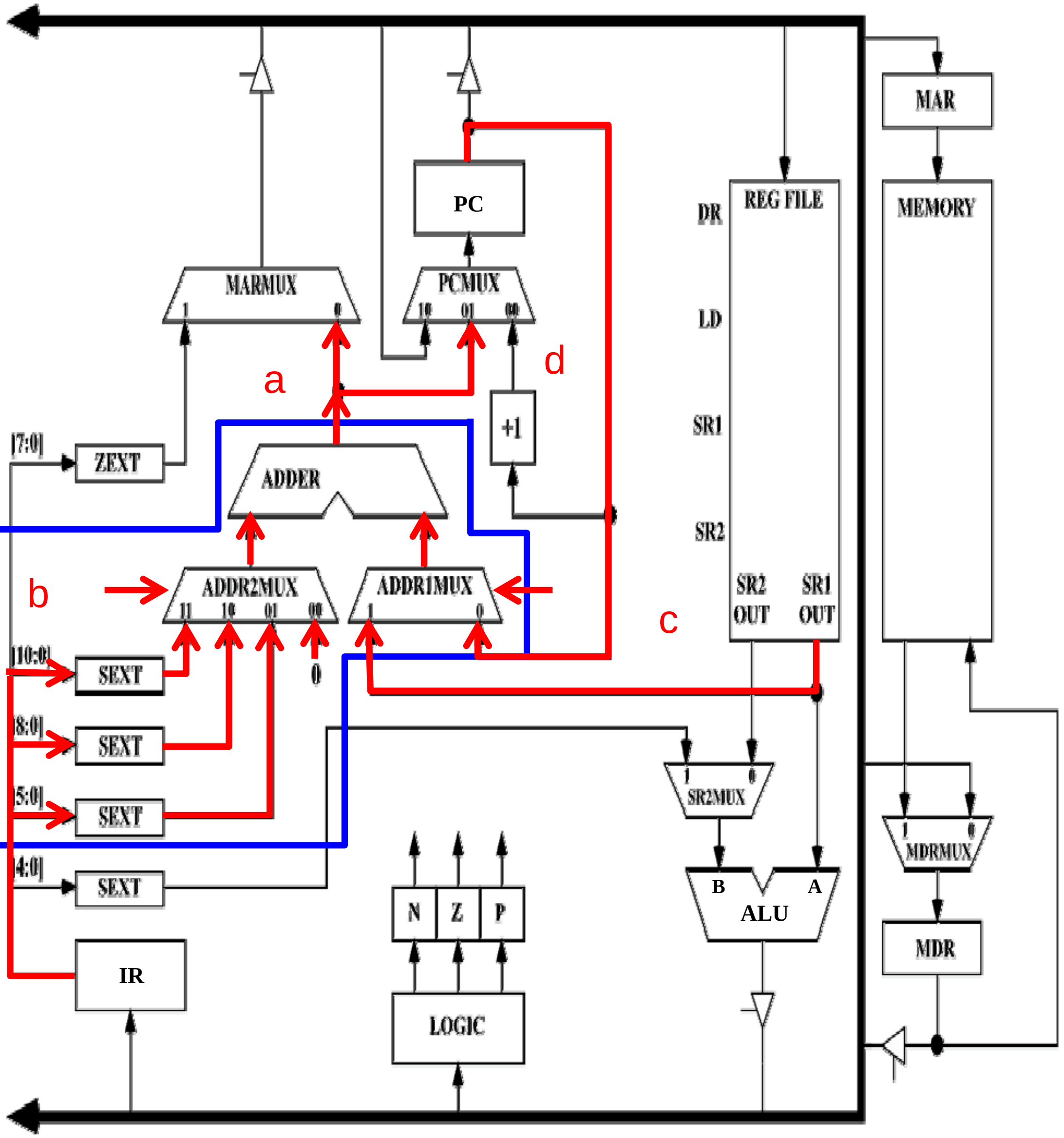
ADD	0001	DR	SR1	0	00	SR2
ADD	0001	DR	SR1	1	imm5	
AND	0101	DR	SR1	0	00	SR2
AND	0101	DR	SR1	1	imm5	
NOT	1001	DR	SR	111111		
BR	0000	n	z	p	PCoffset9	
JMP	1100	0	00	BaseR	000000	
JSR	0100	1	PCoffset11			
JSRR	0100	0	00	BaseR	000000	
RET	1100	0	00	111	000000	

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* (Effective Address Block)
  - Calculates effective addresses for the MAR and the PC

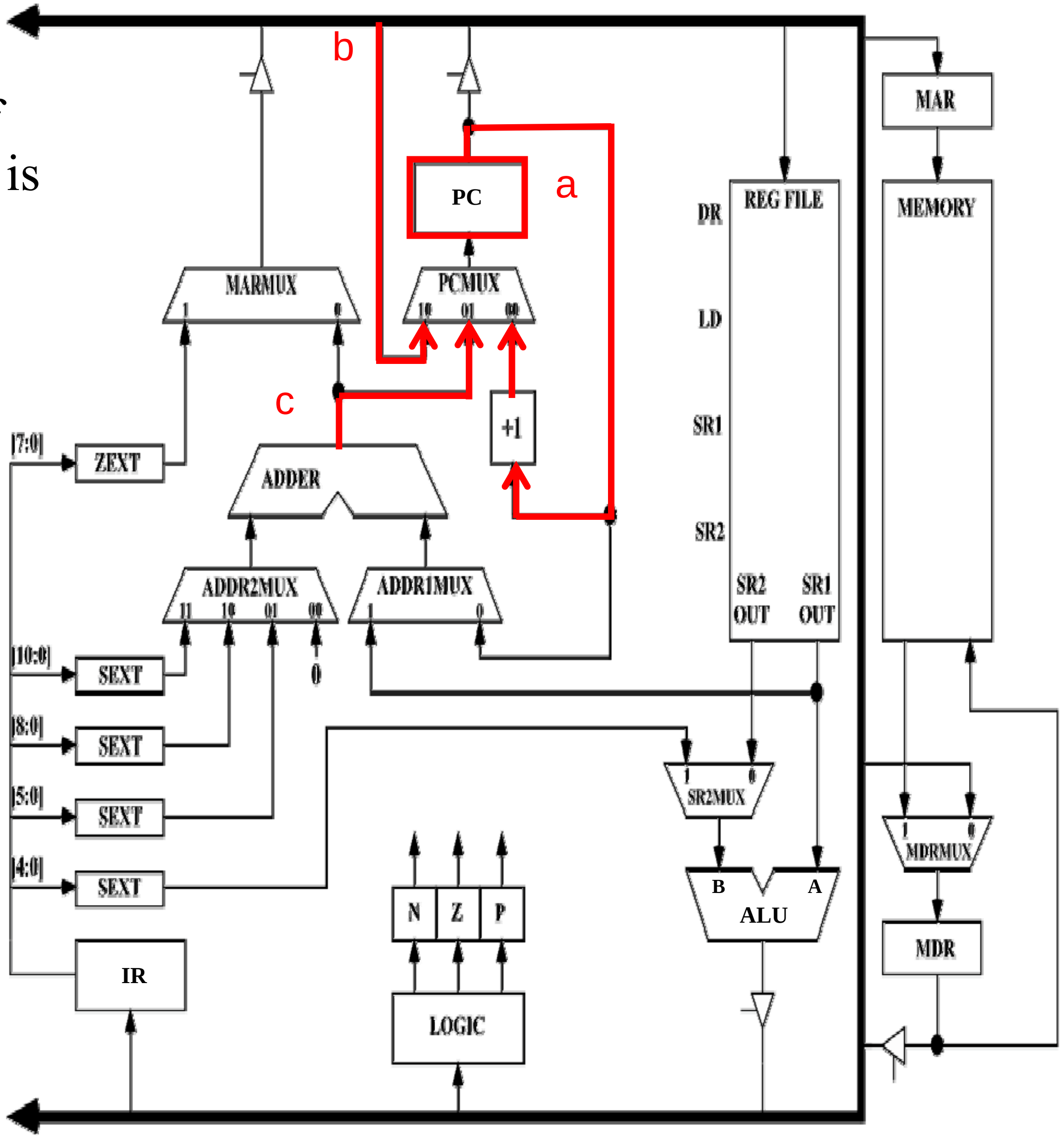
- It adds two operands that are selected by the control unit (a)
- One operand is zero or a sign extended field from the IR (10:0, 8:0, or 5:0) (b)
- The other operand is the current value of the PC or the contents of a register from the register file (c)
- The sum is passed to both the PCMUX and the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

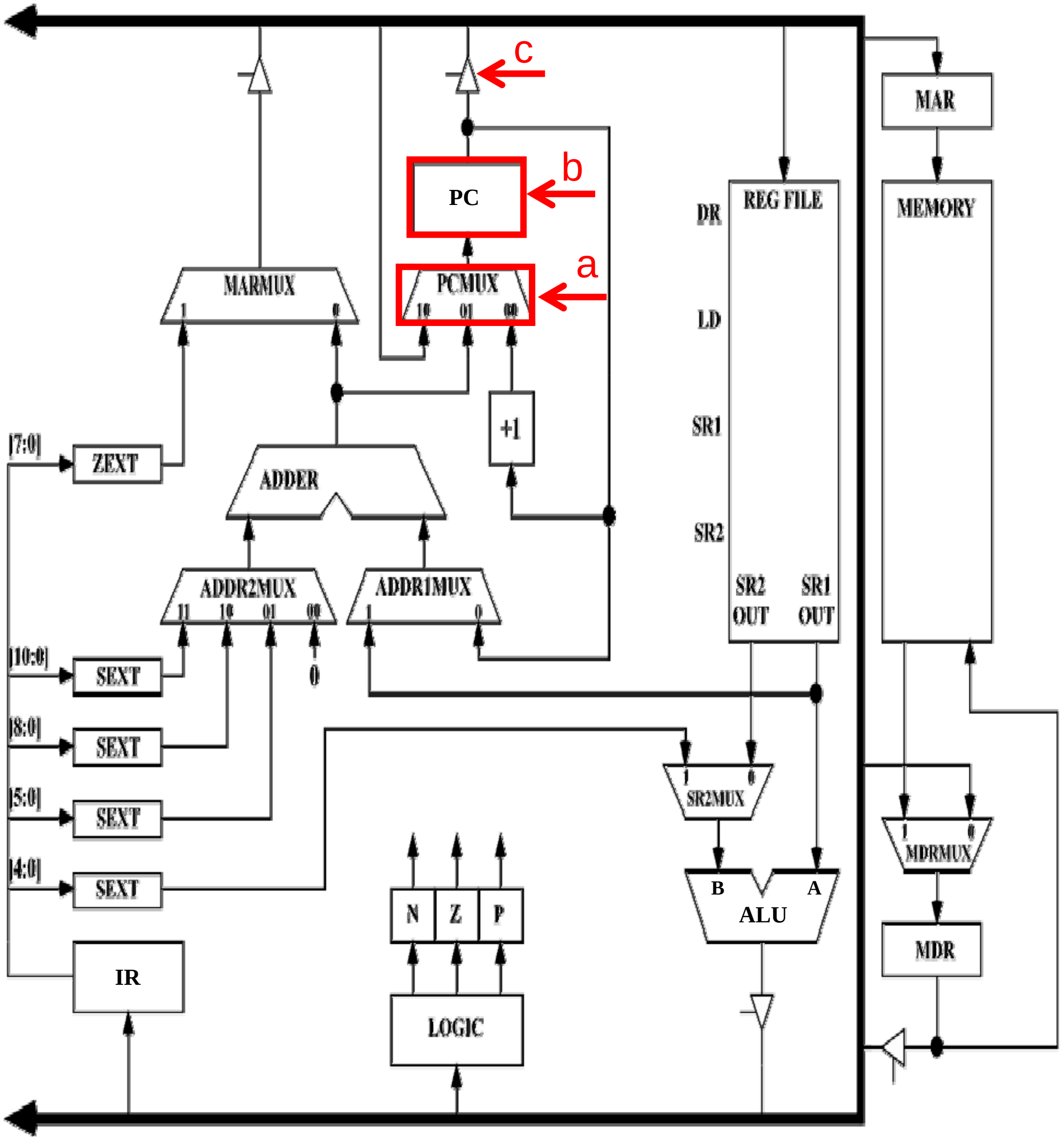
- New PC Computation
  - Can be PC+1 (a)
  - Can come from global bus (b)
  - Can come EAB (c)



BYU
Computer Engineering
Electrical Engineering

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)

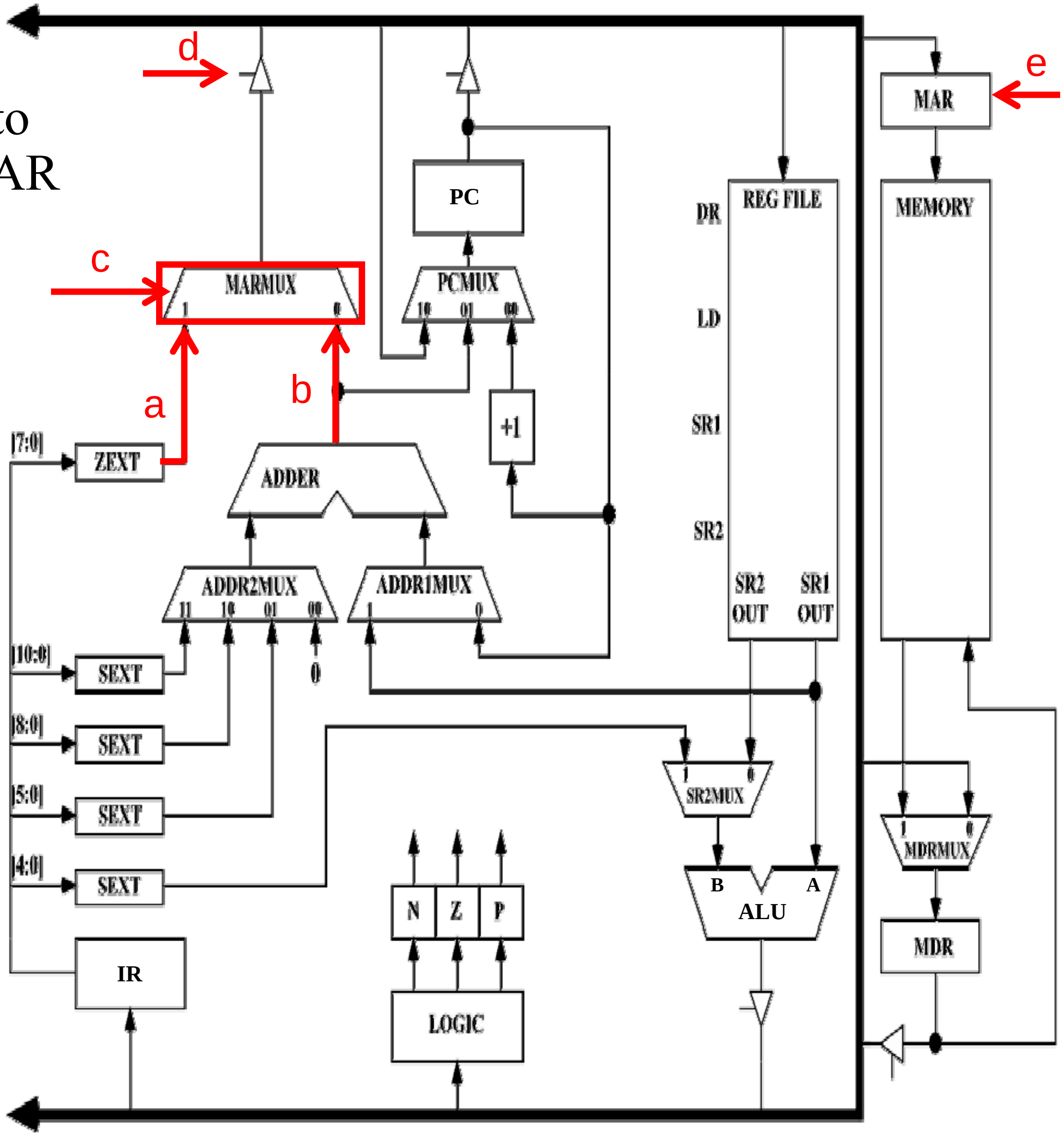


The LC-3 – MARMUX

- The *MARMUX*
  - Selects what address is driven onto global bus for loading into the MAR

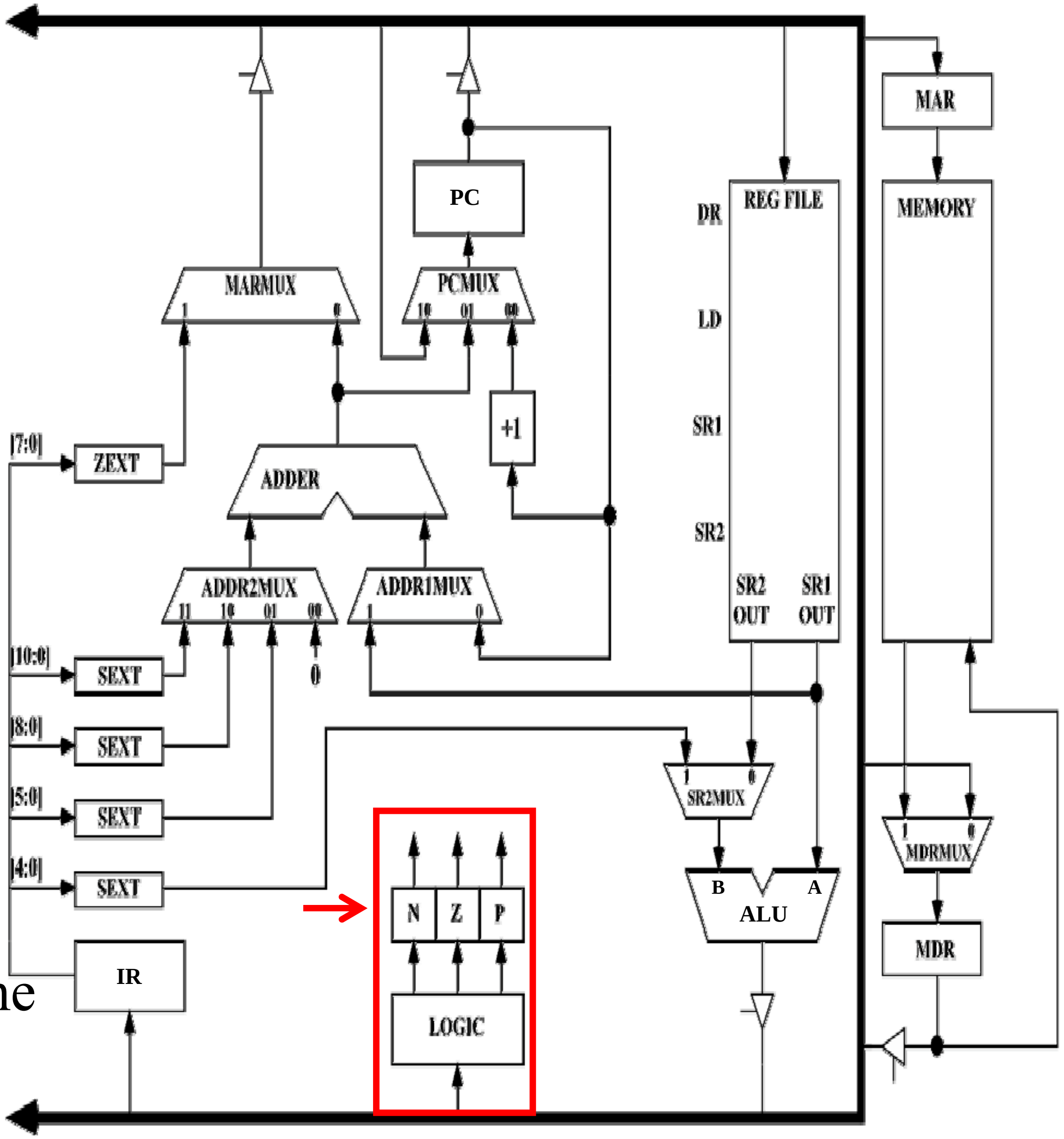
- *MARMUX* Sources
  - Can be $IR_{7:0}$ zero extended (a)
    - For TRAP instructions
  - Can be output of EAB (b)
    - For load instructions

- Control unit selects source (c), controls driving it out onto global bus (d), and when MAR is loaded (e)



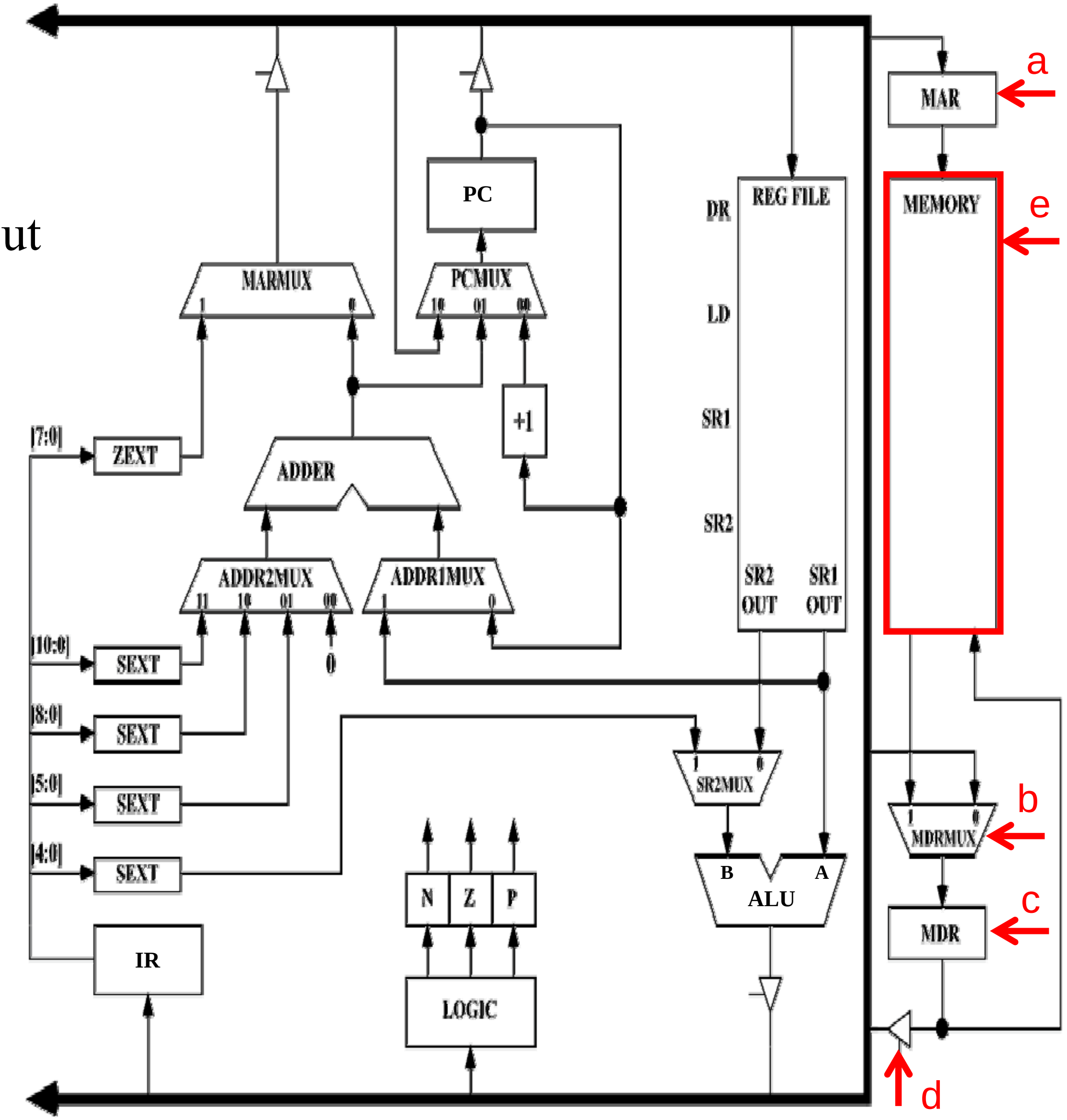
The LC-3 – N, Z, P Condition Codes

- 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 on arithmetic and load instructions
- 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).

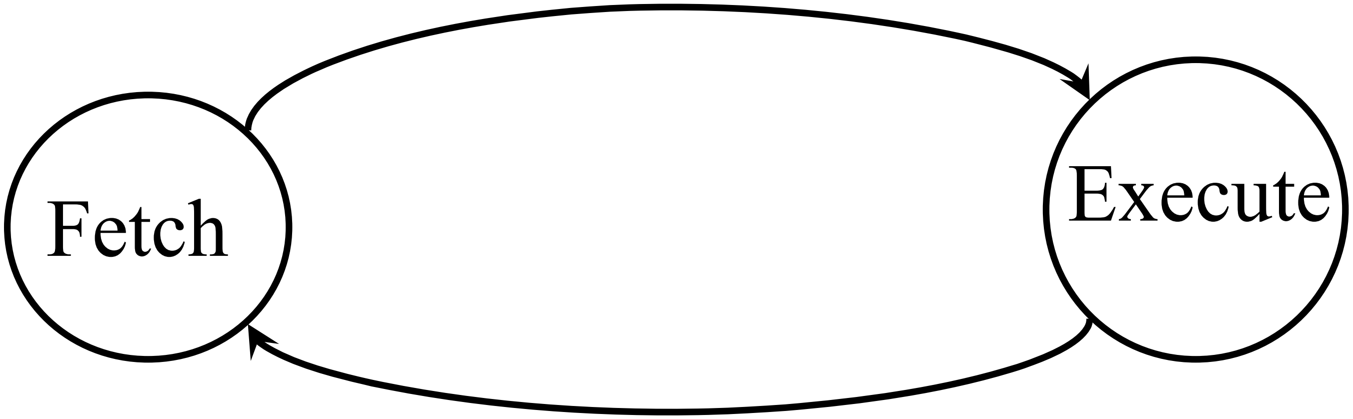


Data Flow

Tracing Data And The Execution of Instructions Through LC-3

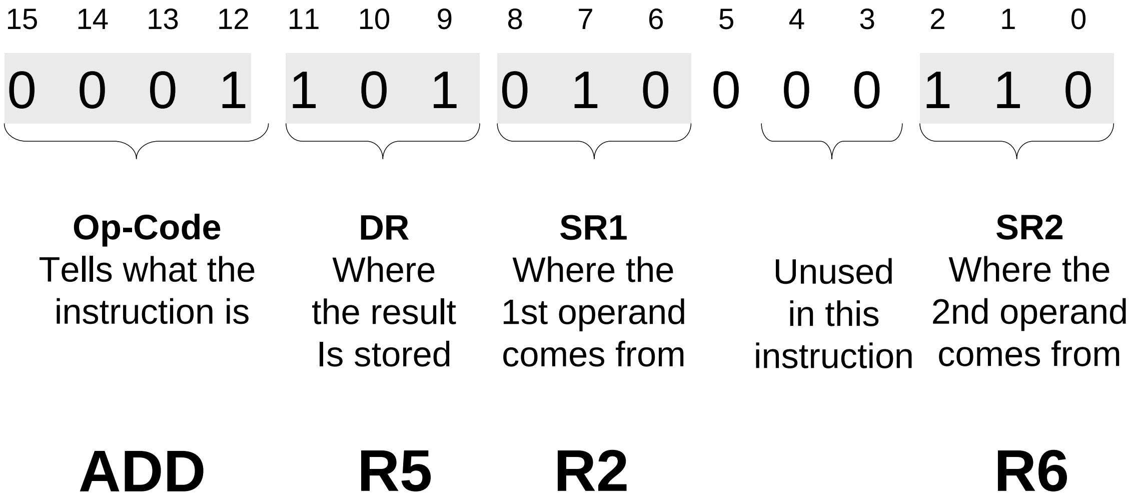
The Von Neumann Model

- Fetch an instruction
- Execute it
- Repeat



Example Instruction

- **ADD R5, R2, R6**
- Operands must already be in registers



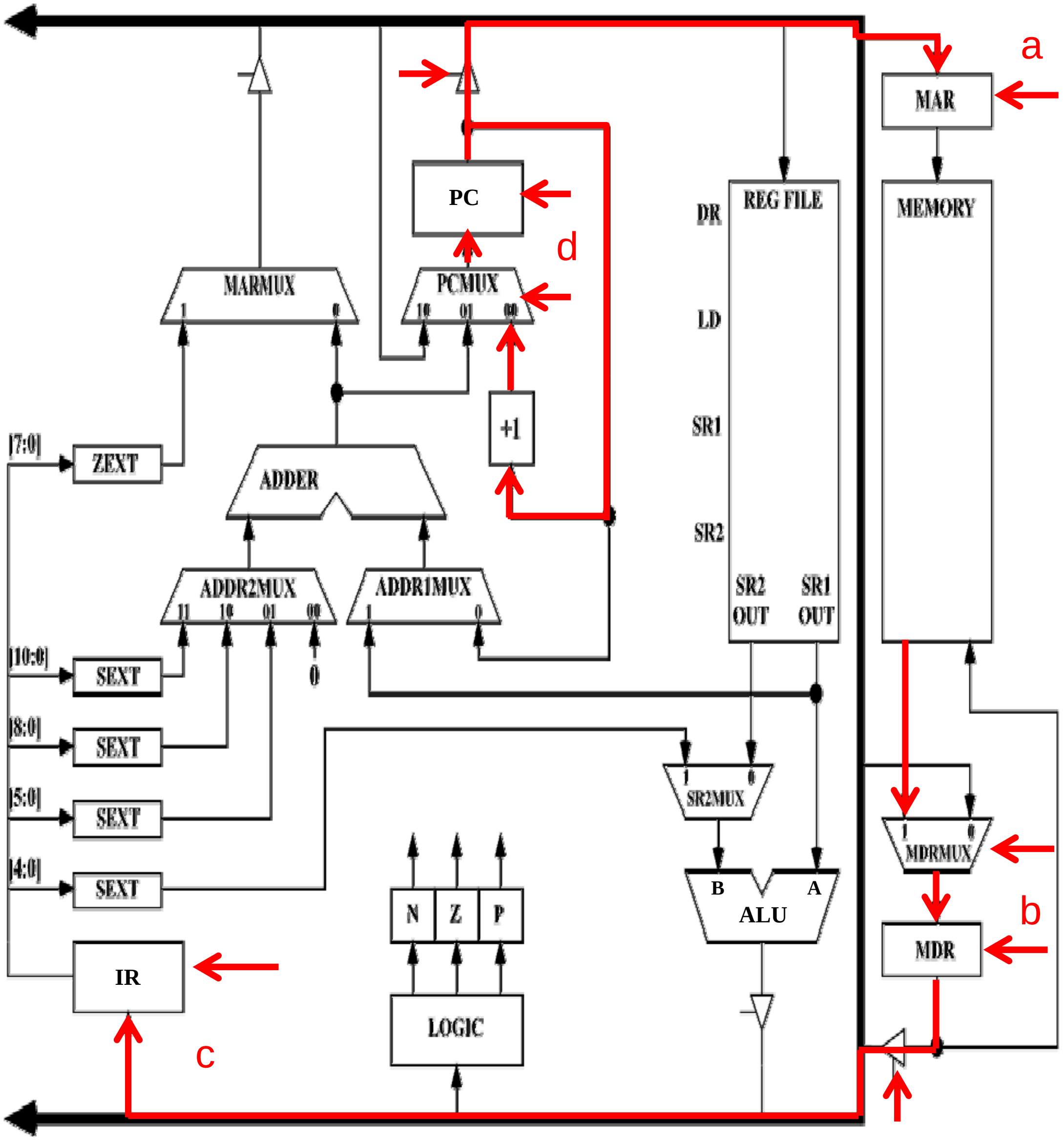
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	1	0	1	0	0	0	0	1	1	0

Op-Code	**DR**	**SR1**		**SR2**
Tells what the instruction is	Where the result Is stored	Where the 1st operand comes from	Unused in this instruction	Where the 2nd operand comes from
ADD	**R5**	**R2**		**R6**

R5 = R2 + R6

Instruction Fetch

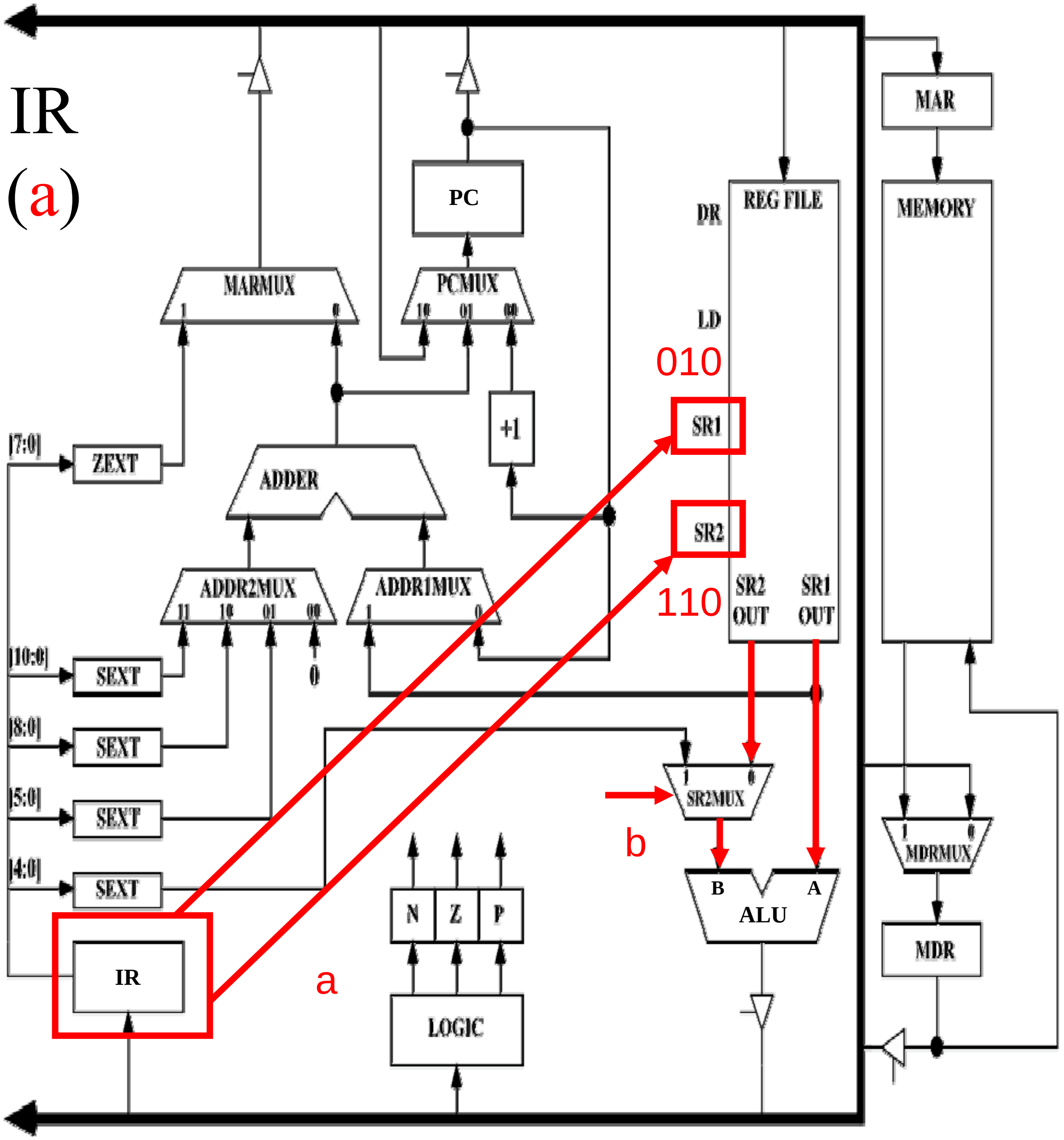
- Copy the PC into the MAR (a)
- Load Memory Output into MDR (b)
- Load Output of MDR into IR (c)
- Increment PC (d)



Operand Selection

0001	101	010	0	00	110
ADD	DR	SR1			SR2

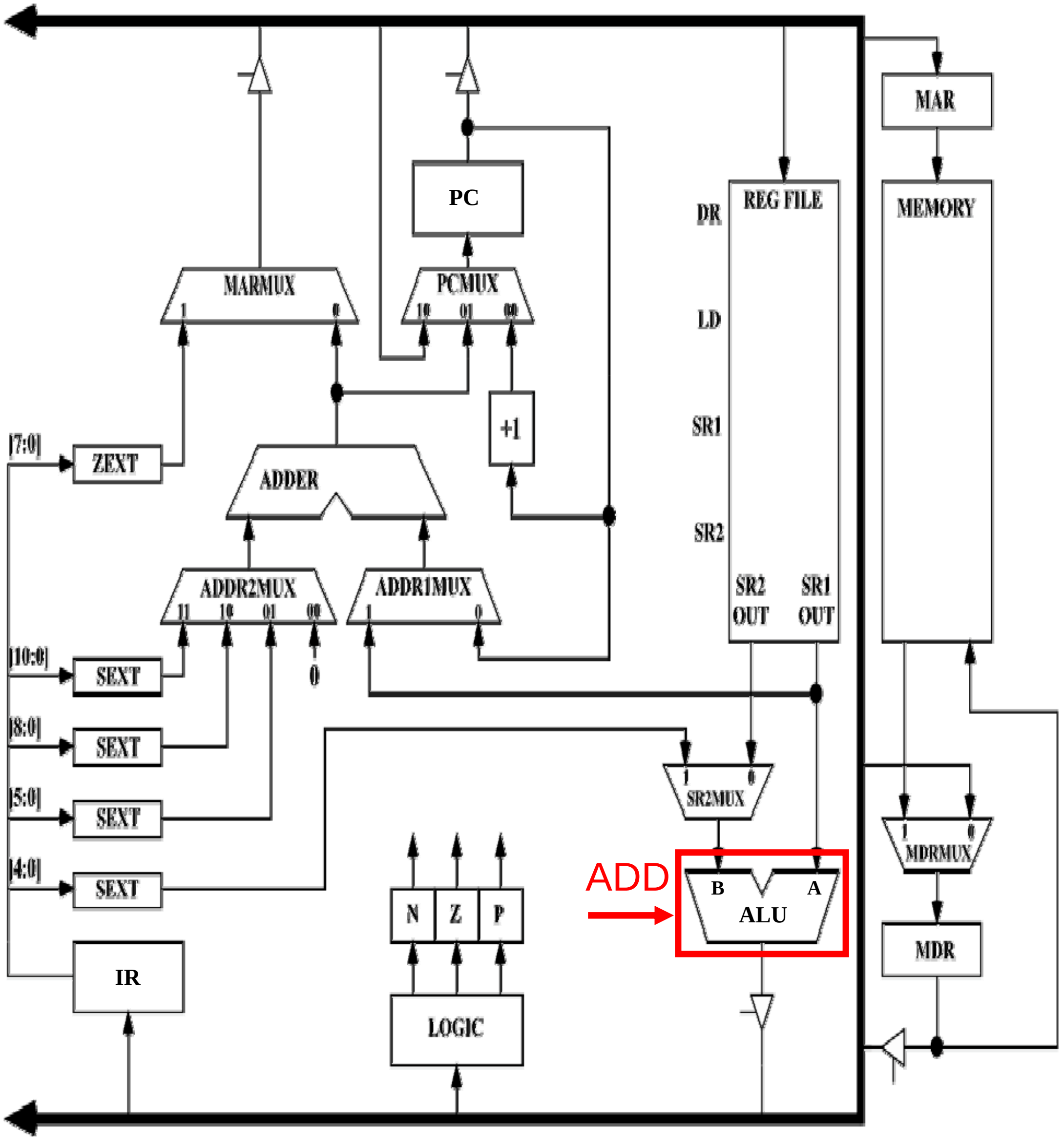
- Send SR1 and SR2 fields from IR as addresses to the register file (a)
- Retrieve values addressed by SR1 and SR2 and send to ALU for execution (b)



Execute

0001	101	010	0	00	110
ADD	DR	SR1			SR2

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



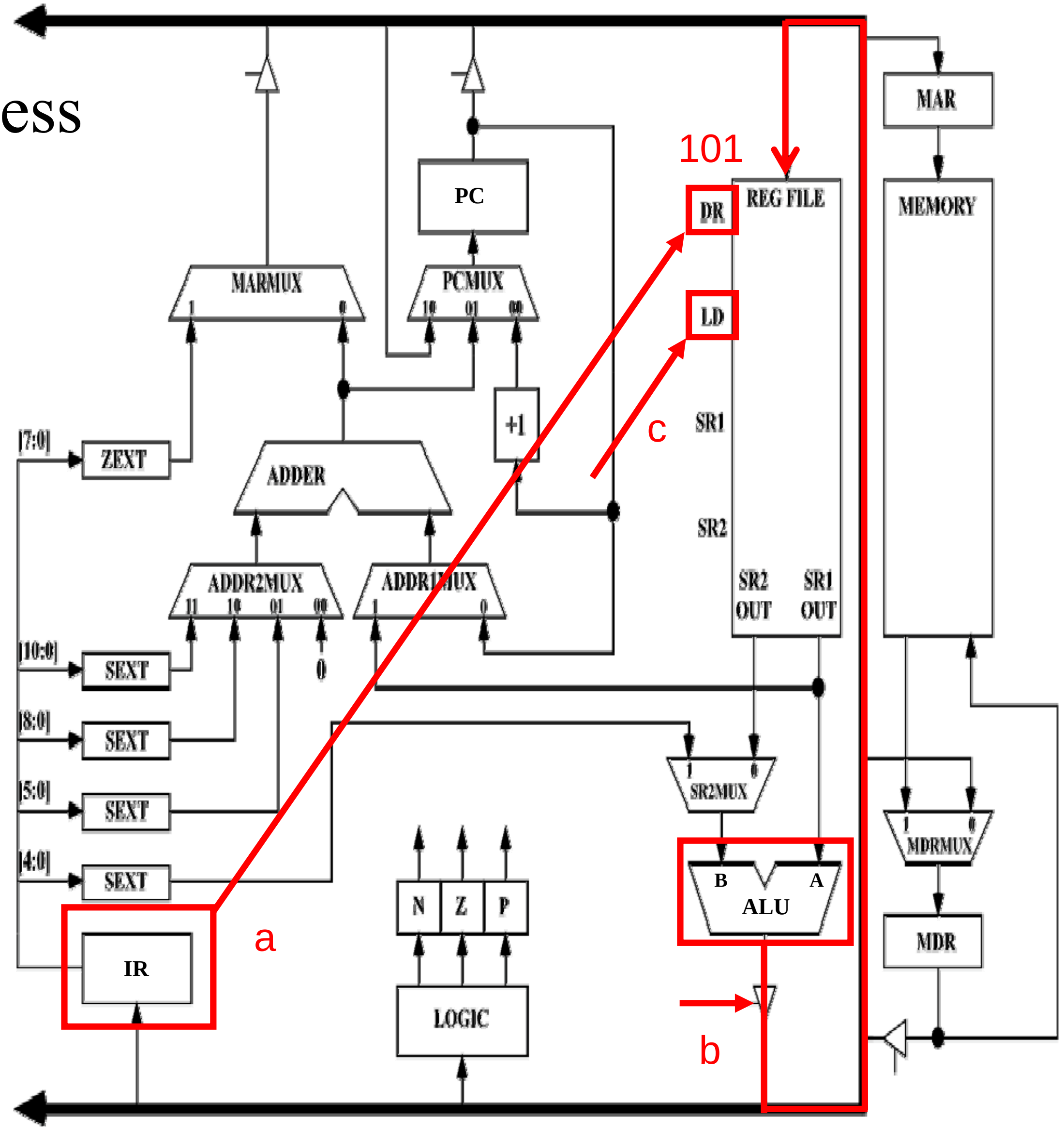
Store Result

0001	101	010	0	00	110
ADD	DR	SR1			SR2

- 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)



BYU
Computer Engineering
Electrical Engineering

Another Example Instruction

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

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

mem [BaseR + SEXT(offset6)] = SR

Op-Code
STR

SR
(**R2**)

BaseR
(**R3**)

offset6 (**9**)
is added to contents
of BaseR to get the
memory location where
the contents of SR will
be stored

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

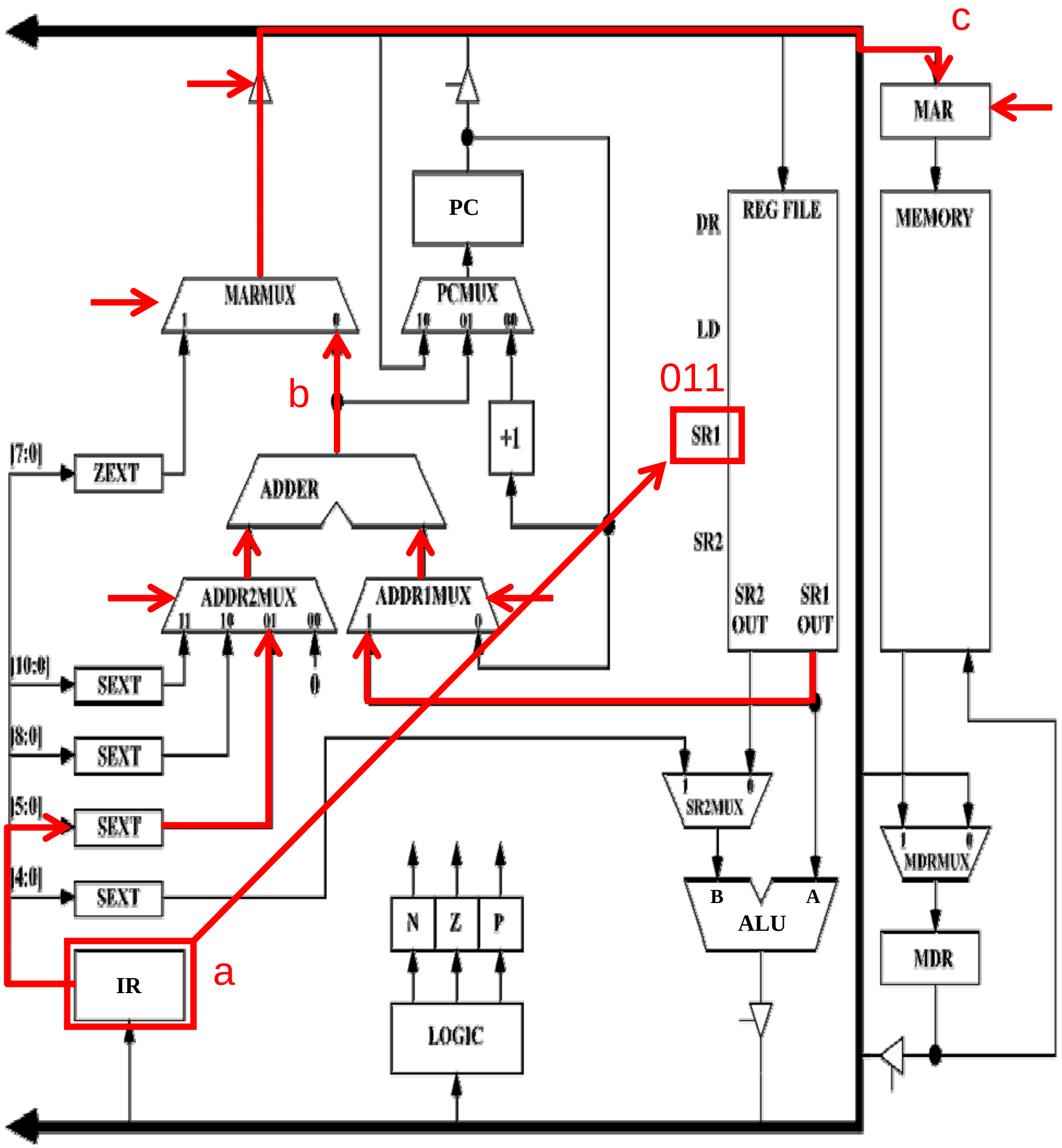
STR – Instruction Fetch

Same as ADD Instruction

STR

0111	010	011	001001
STR	SR	BaseR	offset6

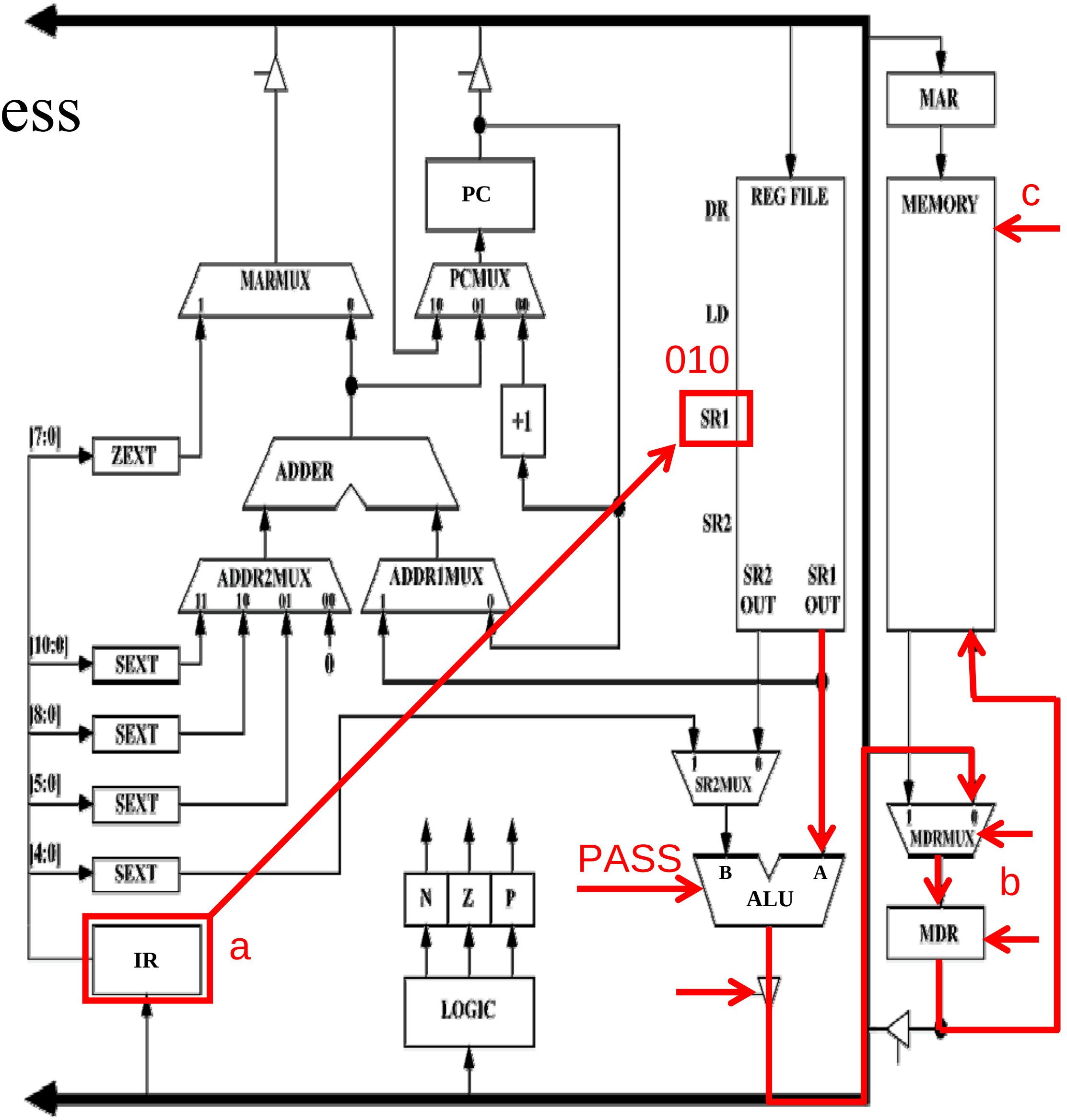
- Send BaseR field from IR as address to the register file (a)
- Add the contents of BaseR to the sign extended offset6 from the IR to form the destination memory address for the STR (b)
- Store the generated address into the MAR (c)



STR

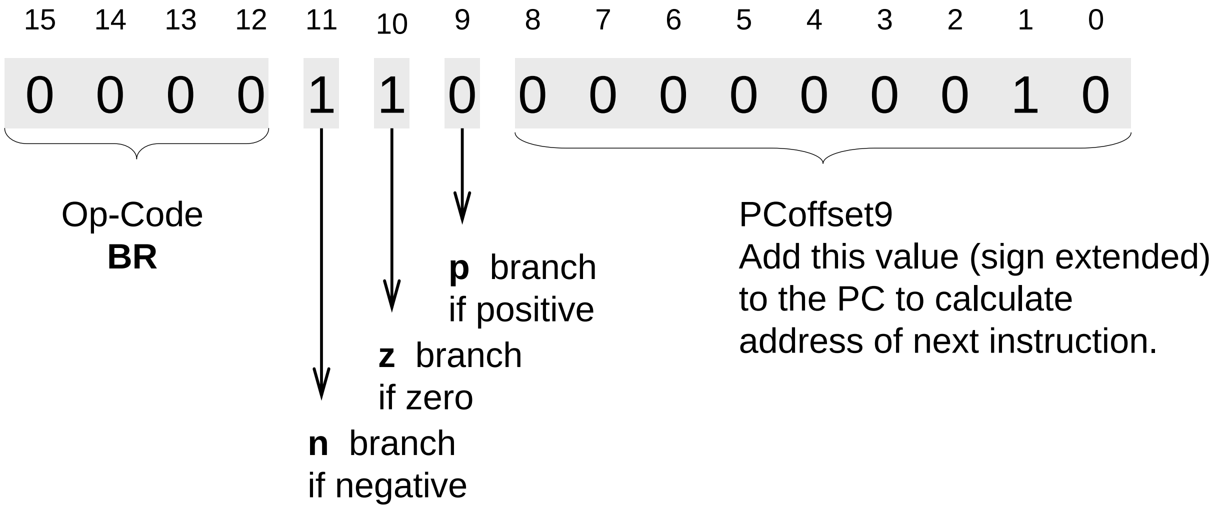
0111	010	011	001001
STR	SR	BaseR	index6

- 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



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

Op-Code
BR

p branch
if positive

z branch
if zero

n branch
if negative

PCoffset9
Add this value (sign extended)
to the PC to calculate
address of next instruction.

```
if (n AND N) OR (z AND Z) OR (p AND P)
    PC = PC + PCoffset9
```

What are Condition Codes ?

- LC-3 contains 3 special registers
  - 1-bit wide each
  - Named N, Z, P   (negative, zero, positive)
- For most instructions, when a register is written with a new value then N, Z, and P are updated to reflect the value written
- Only specific instructions modify the condition codes
  - See back cover or appendix of ECEn/CS 124 book to be sure

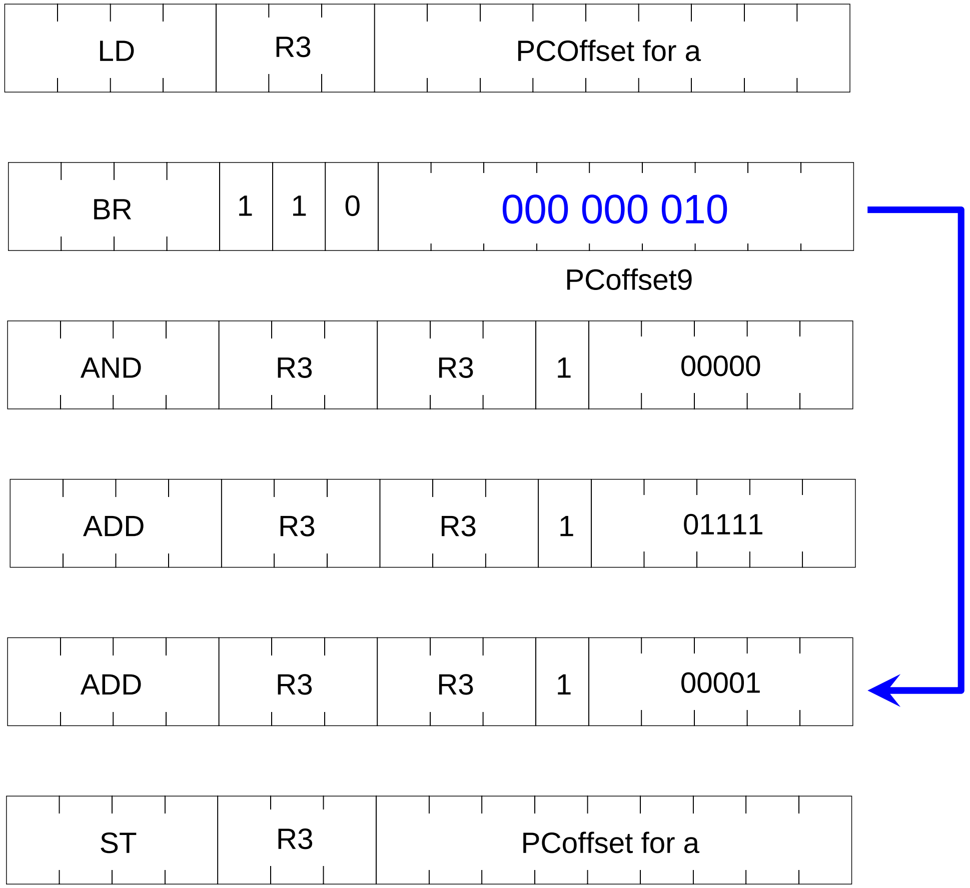
All Instructions

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															

🟨 ⇨ Instruction sets condition codes N Z P

An if Statement Using BR

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



Address					
x3000	LD	R3	PCOffset for a		
x3001	BR	1	1	0	000 000 010
					PCoffset9
x3002	AND	R3	R3	1	00000
x3003	ADD	R3	R3	1	01111
x3004	ADD	R3	R3	1	00001
x3005	ST	R3	PCoffset for a		

BRnz Instruction Fetch

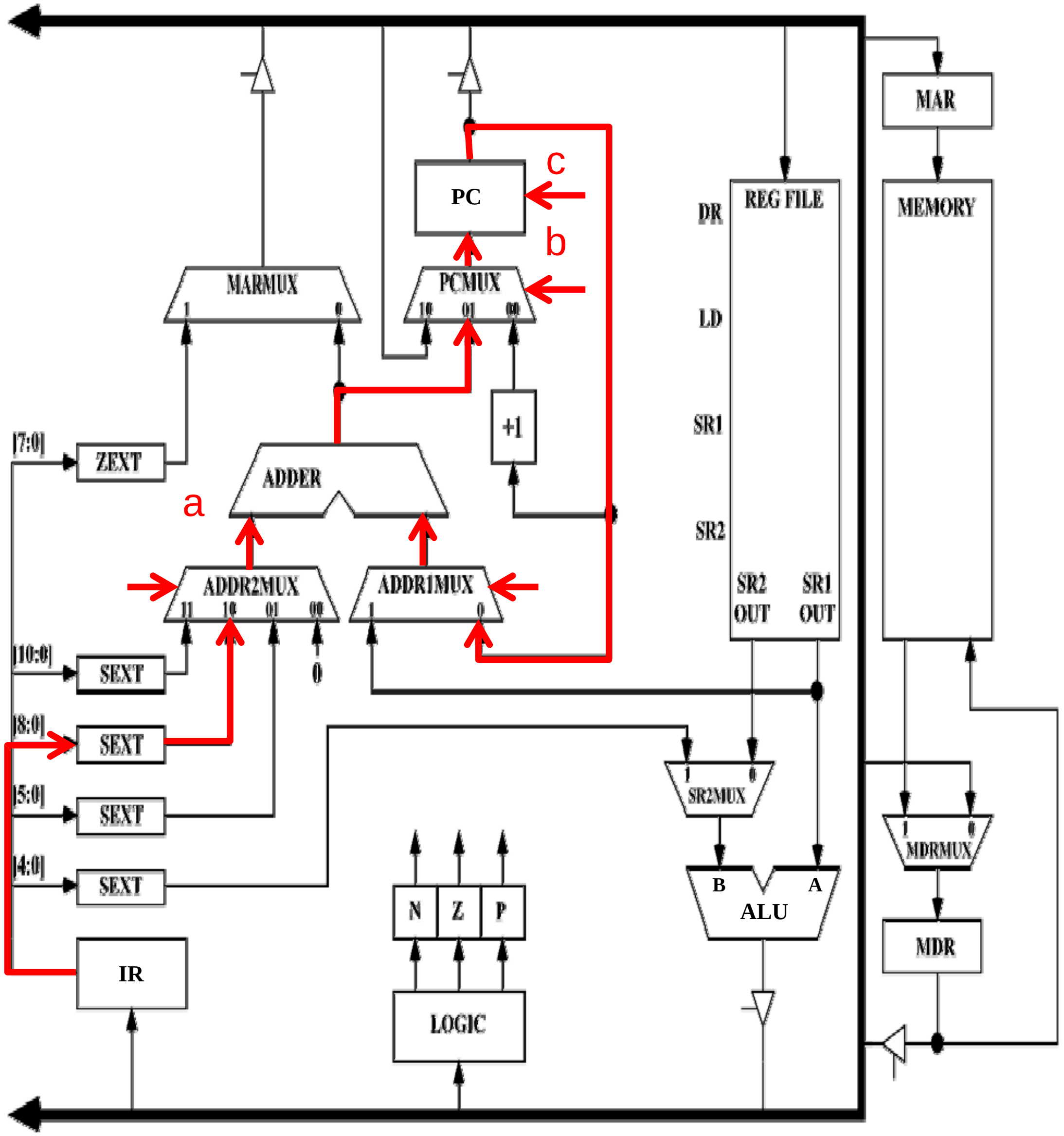
Same as ADD Instruction

BRnz – Execution

BR	n	z	p	PCoffset9
0000	1	1	0	000 000 010

- Compare n and z in IR to N and Z registers
- Generate branch address

  PC + SEXT(PCoffset9) (a)
- Pass new address through the PCMUX (b)
- Load branch address into PC iff the condition codes match (c)



c
PC
b
MARMUX
PCMUX
ADDER
+1
a
ZEXT
ADDR2MUX
ADDR1MUX
[7:0]
[10:0]
SEXT
[8:0]
SEXT
[5:0]
SEXT
[4:0]
SEXT
IR
0
N
Z
P
LOGIC
DR
LD
SR1
SR2
REG FILE
SR2 OUT
SR1 OUT
SR2MUX
B
A
ALU
MAR
MEMORY
MDRMUX
MDR