Design of Digital Circuits
Lecture 9: Von Neumann Model, ISA, LC-3 and MIPS

Dr. Juan Gómez Luna
Prof. Onur Mutlu

ETH Zurich
Spring 2018
22 March 2018

Basic Elements of a Computer

- In past lectures we learned
  - Combinational circuits
  - Sequential circuits
- With them, we can build
  - Decision elements
  - Storage elements
- Basic elements of a computer

- To get a task done by a computer we need
  - Computer
  - Data
  - Program: A set of instructions
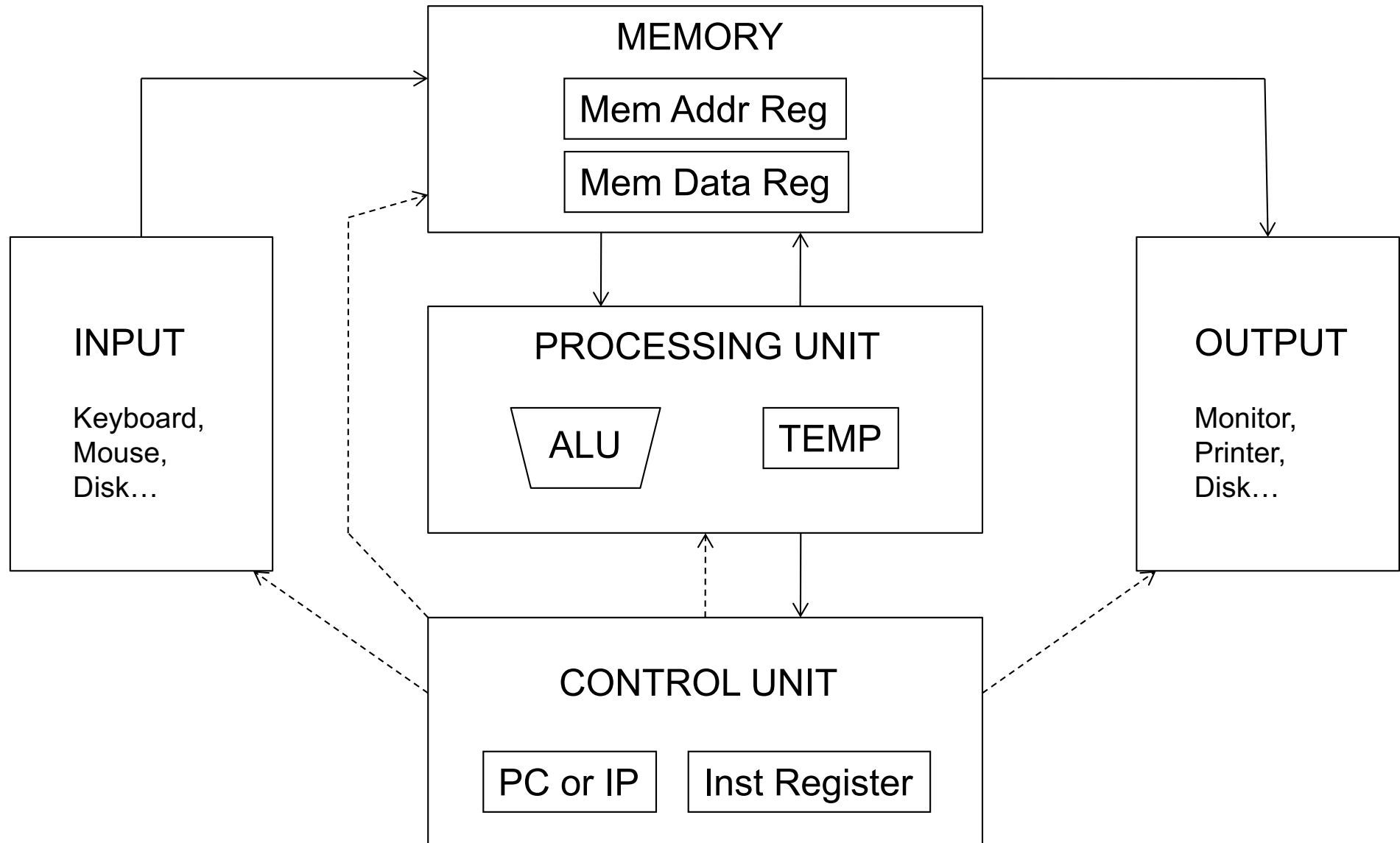   - Instruction: the smallest piece of work in a computer

The Von Neumann Model

- Let's start building the computer

- In order to build a computer we need a model

- John von Neumann proposed a fundamental model in 1946
- It consists of 5 parts
  - Memory
  - Processing unit
  - Input
  - Output
  - Control unit

- Throughout this lecture, we consider two examples of the von Neumann model
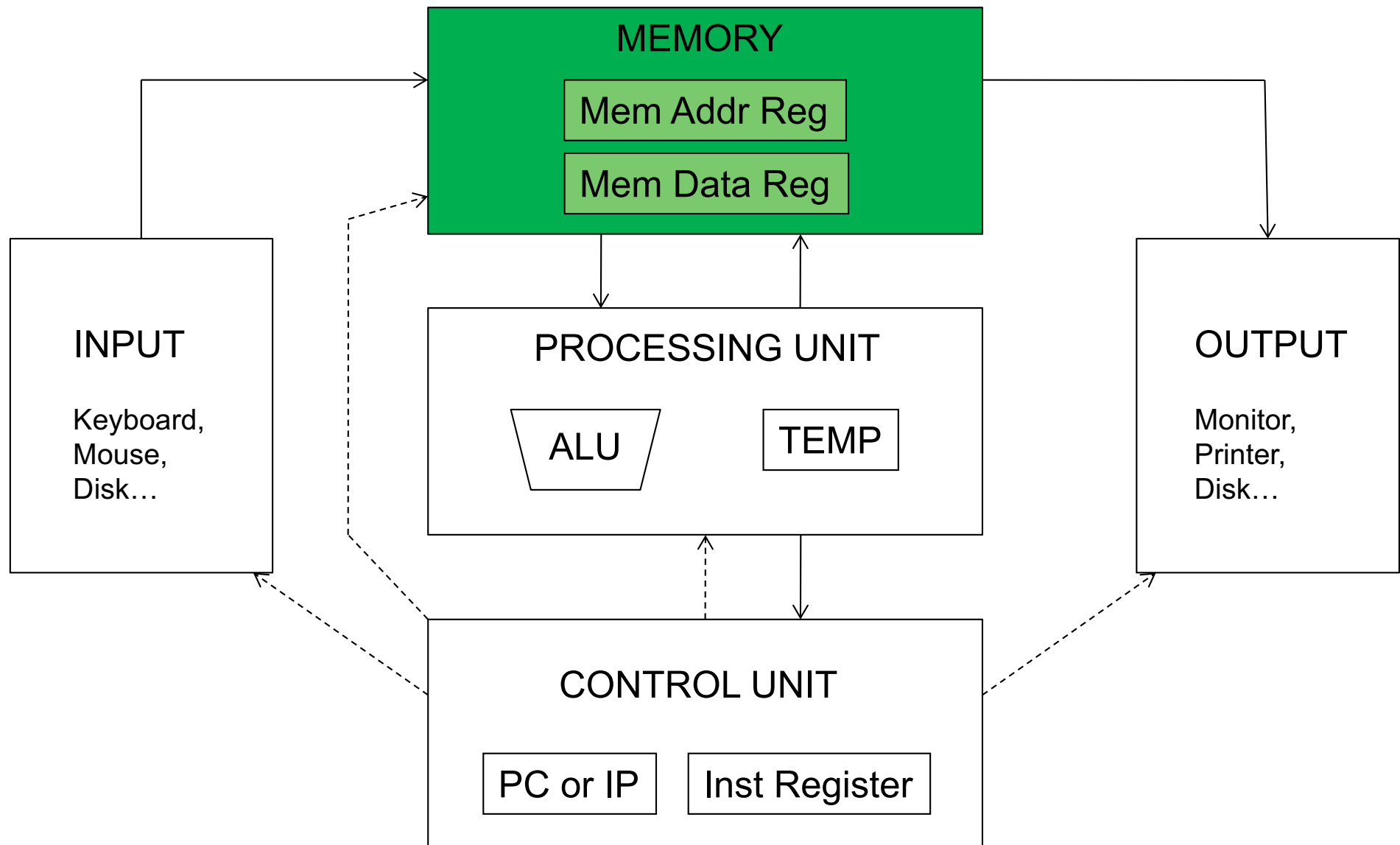 - LC-3
  - MIPS

Burks, Goldstein, von Neumann,
"Preliminary discussion of the logical design
of an electronic computing instrument," 1946.

The Von Neumann Model

The Von Neumann Model

MEMORY

Mem Addr Reg

Mem Data Reg

INPUT

Keyboard,
Mouse,
Disk…

PROCESSING UNIT

ALU

TEMP

OUTPUT

Monitor,
Printer,
Disk…

CONTROL UNIT

PC or IP

Inst Register

Memory

- The memory stores
  - Data
  - Programs

- The memory contains bits
  - Bits are grouped into bytes (8 bits) and words (e.g., 8, 16, 32 bits)

- How the bits are accessed determines the addressability
  - E.g., word-addressable
  - E.g., 8-bit addressable (or byte-addressable)

- The total number of addresses is the address space
  - In LC-3, the address space is 2^{16}
    - 16-bit addresses
  - In MIPS, the address space is 2^{32}
    - 32-bit addresses

Word-Addressable Memory

- Each data word has a unique address
    - In MIPS, a unique address for each 32-bit data word
    - In LC-3, a unique address for each 16-bit data word

Word Address	Data	MIPS memory
.	.	.
.	.	.
.	.	.
00000003	D 1 6 1 7 A 1 C	Word 3
00000002	1 3 C 8 1 7 5 5	Word 2
00000001	F 2 F 1 F 0 F 7	Word 1
00000000	8 9 A B C D E F	Word 0

Byte-Addressable Memory

- Each byte has a unique address
  - Actually, MIPS is byte-addressable
  - LC-3b is byte-addressable, too

MIPS memory

Word Address	Data				MIPS memory
⋮	⋮				⋮
0000000C	D 1	6 1	7 A	1 C	Word 3
00000008	1 3	C 8	1 7	5 5	Word 2
00000004	F 2	F 1	F 0	F 7	Word 1
00000000	How are these four bytes addressed?				Word 0

Accessing Memory: MAR and MDR

- There are two ways of accessing memory
  - Reading or loading
  - Writing or storing

- Two registers are necessary to access memory
  - Memory Address Register (MAR)
  - Memory Data Register (MDR)

- To read
  - Step 1: Load the MAR with the address
  - Step 2: Data is placed in MDR

- To write
  - Step 1: Load the MAR with the address and the MDR with the data
  - Step 2: Activate Write Enable signal
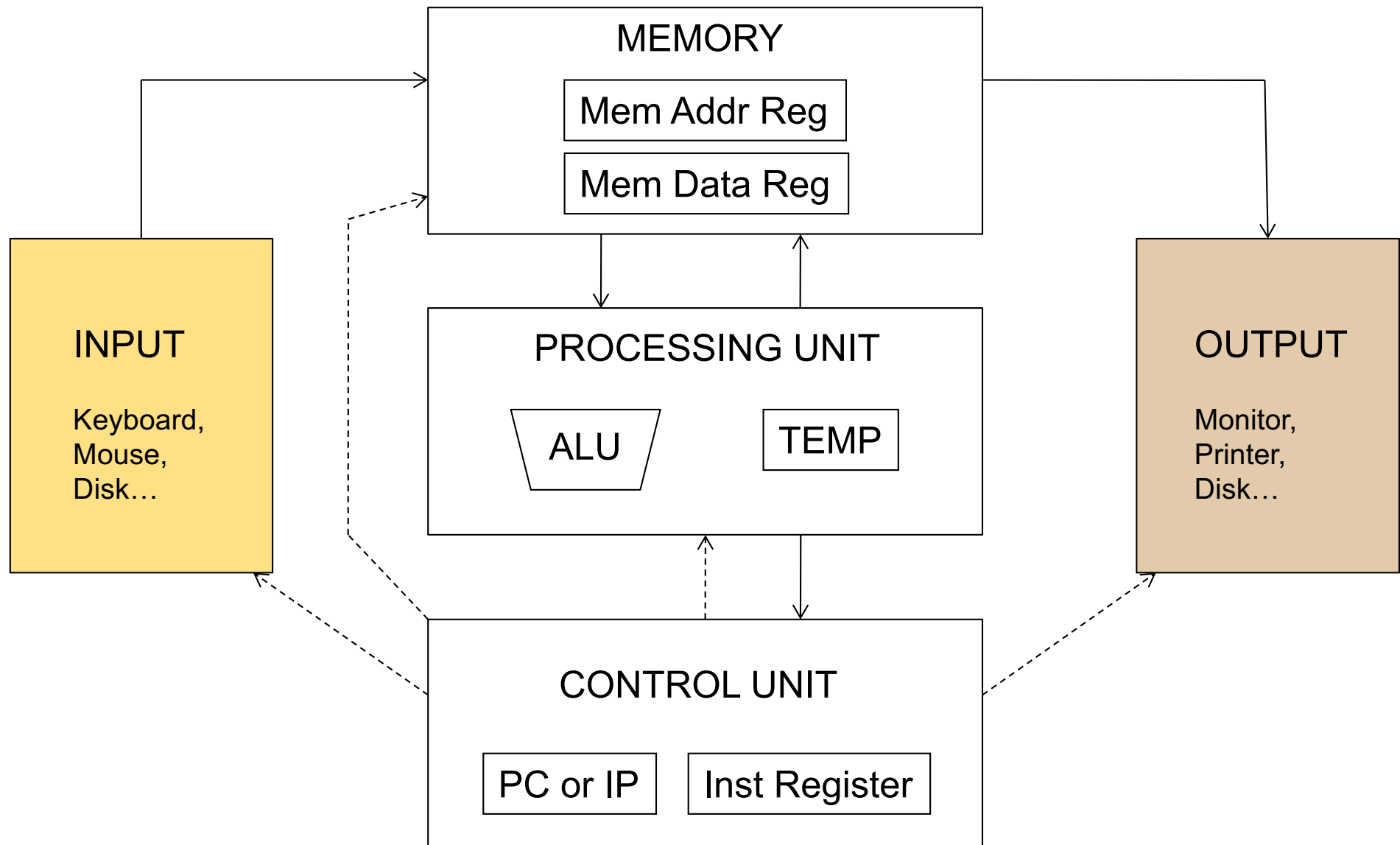
The Von Neumann Model
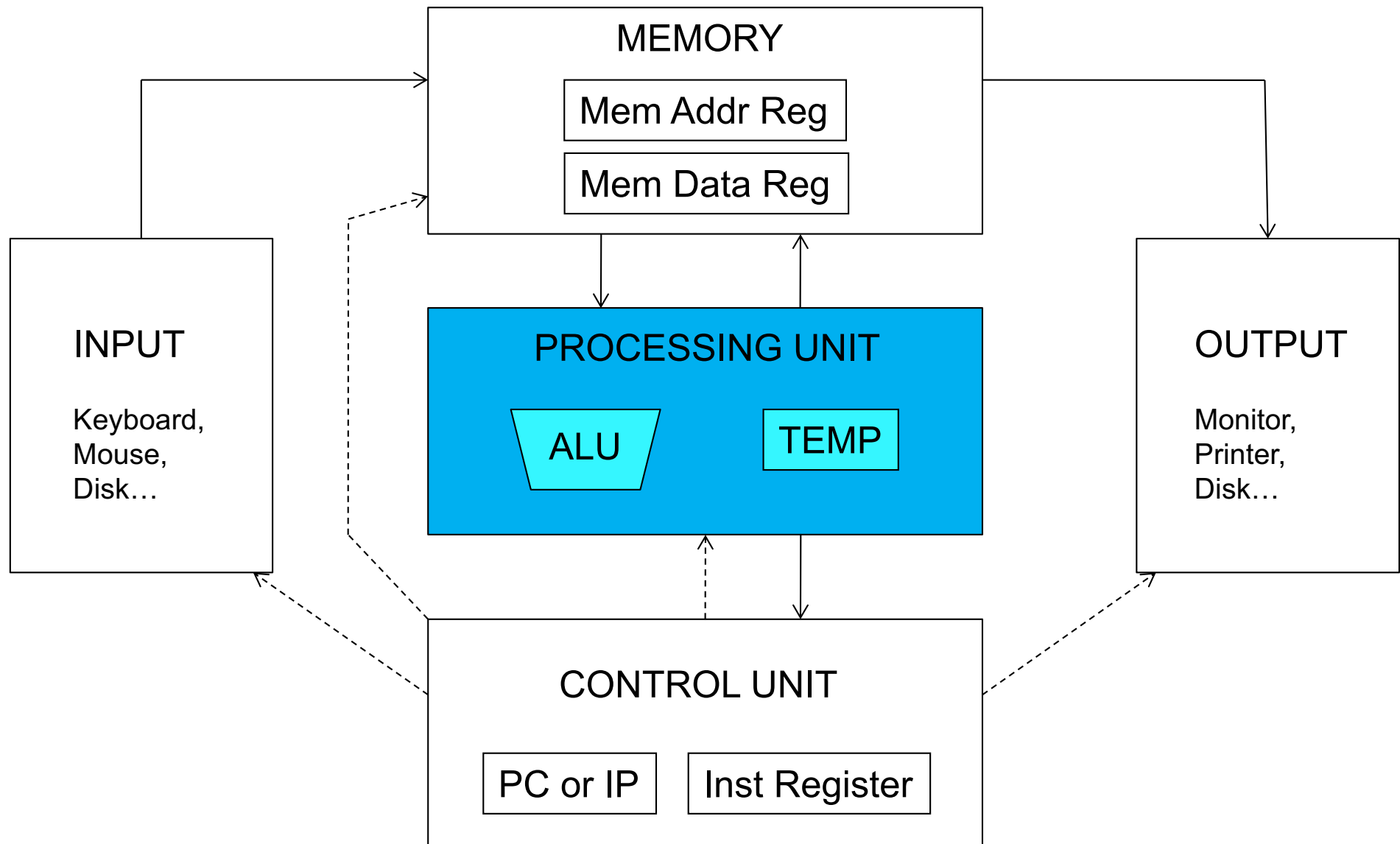
Processing Unit

- The processing unit can consist of many functional units

- We start with a simple Arithmetic and Logic Unit (ALU)
    - LC-3: ADD, AND, NOT (XOR in LC-3b)
    - MIPS: add, sub, mult, and, nor, sll, slr, slt…

- The ALU processes quantities that are referred to as words
    - Word length in LC-3 is 16 bits
    - In MIPS it is 32 bits

- Temporary storage: Registers
    - E.g., to calculate (A+B)*C, the intermediate result of A+B is stored in a register

Registers

- **Memory** is big but slow

- **Registers**
  - Ensure fast access to operands
  - Typically one register contains one word

- **Register set or file**
  - LC-3 has 8 general purpose registers (GPR)
    - R0 to R7: 3-bit register number
    - Register size = Word length = 16 bits
  - MIPS has 32 registers
    - Register size = Word length = 32 bits
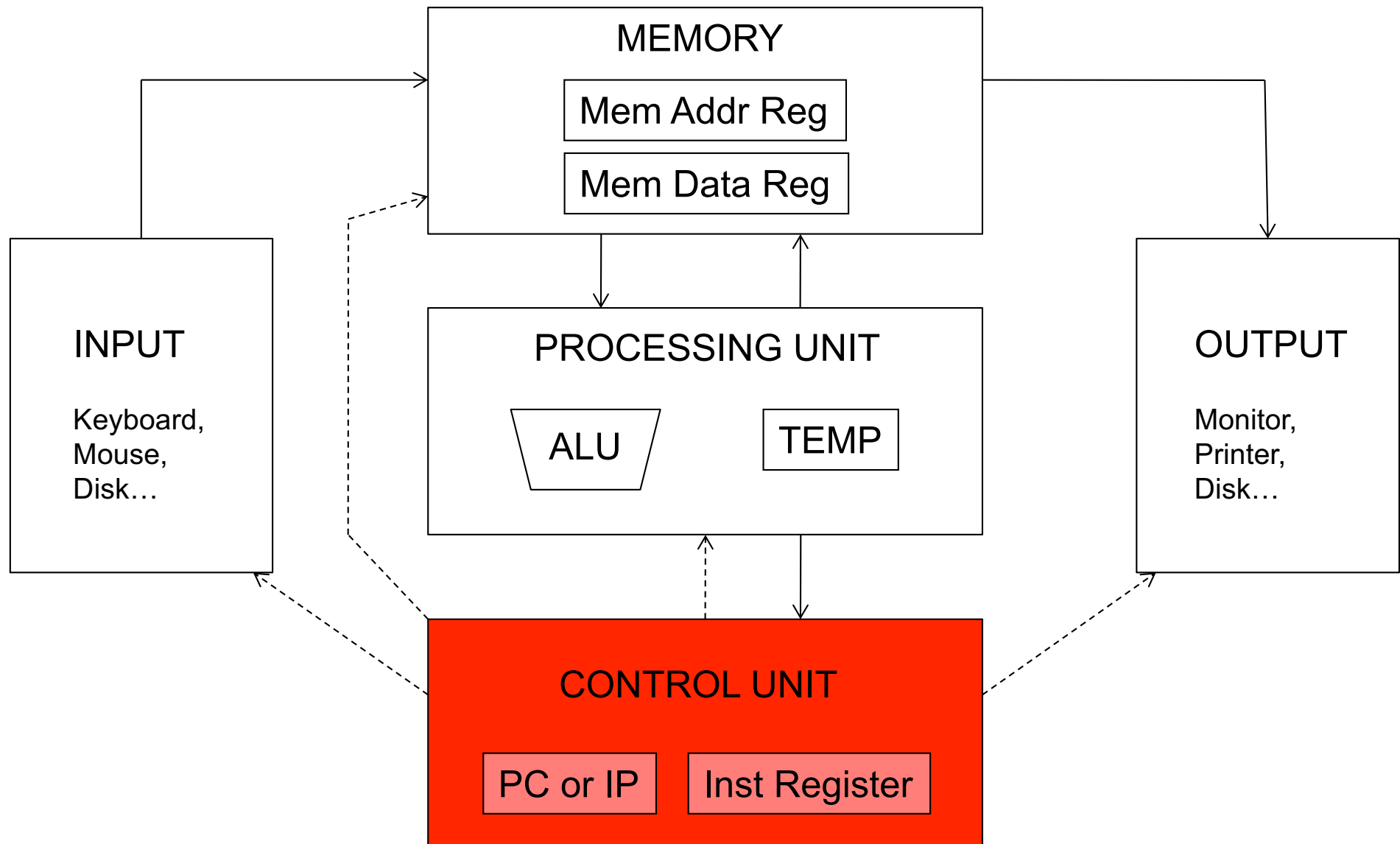
The Von Neumann Model

Input and Output

- Many devices can be used for input and output

- They are called peripherals
  - Input
    - Keyboard
    - Mouse
    - Scanner
    - Disks
    - Etc.
  - Output
    - Monitor
    - Printer
    - Disks
    - Etc.

 - In LC-3, we consider keyboard and monitor

The Von Neumann Model

Control Unit

- The control unit is the conductor of the orchestra

- It conducts the step-by-step process of executing a program

- It keeps track of the instruction being executed with an instruction register (IR), which contains the instruction

- Another register contains the address of the next instruction to execute. It is called program counter (PC) or instruction pointer (IP)

Programmer Visible (Architectural) State

M[0]
M[1]
M[2]
M[3]
M[4]

M[N-1]
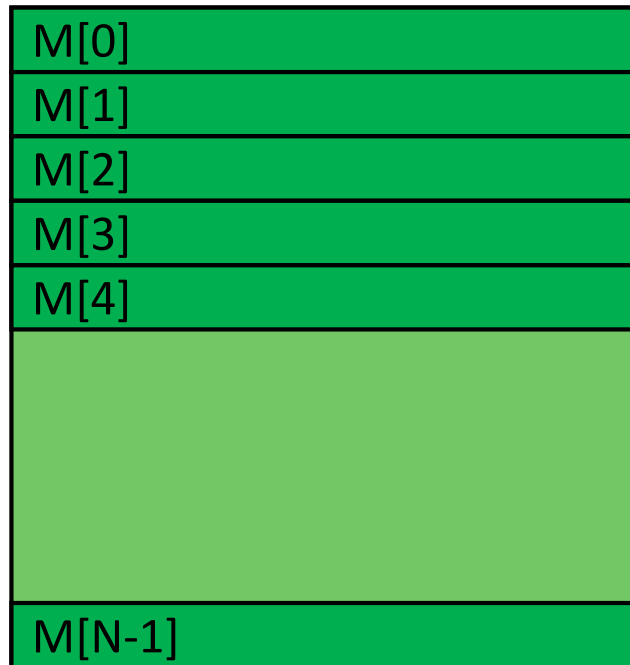
Memory
array of storage locations
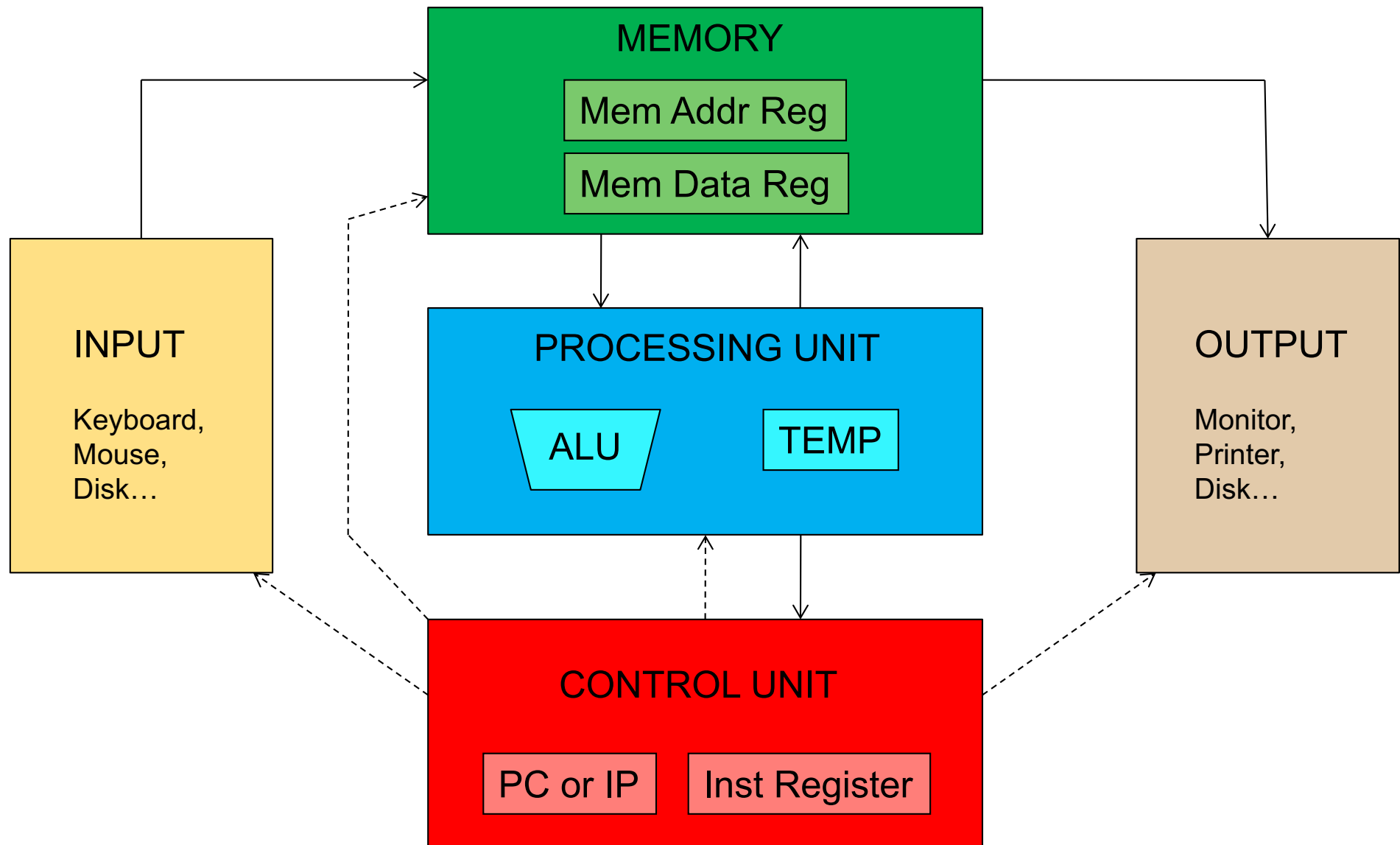indexed by an address

Registers
- given special names in the ISA
  (as opposed to addresses)
- general vs. special purpose

Program Counter
memory address
of the current instruction

Instructions (and programs) specify how to transform
the values of programmer visible state

The Von Neumann Model

LC-3: A Von Neumann Machine

LC-3: A Von Neumann Machine

Figure 4.3   The LC-3 as an example of the von Neumann model

Program Counter

Control signals

Data

Finite State Machine (for Generating Control Signals)

Instruction Register

Memory Data Register

Memory Address Register

8 General Purpose Registers (GPR)

ALU: 2 inputs, 1 output

ALU operation

GateALU

Keyboard KBDR (data), KBSR (status)

Monitor DDR (data), DSR (status)
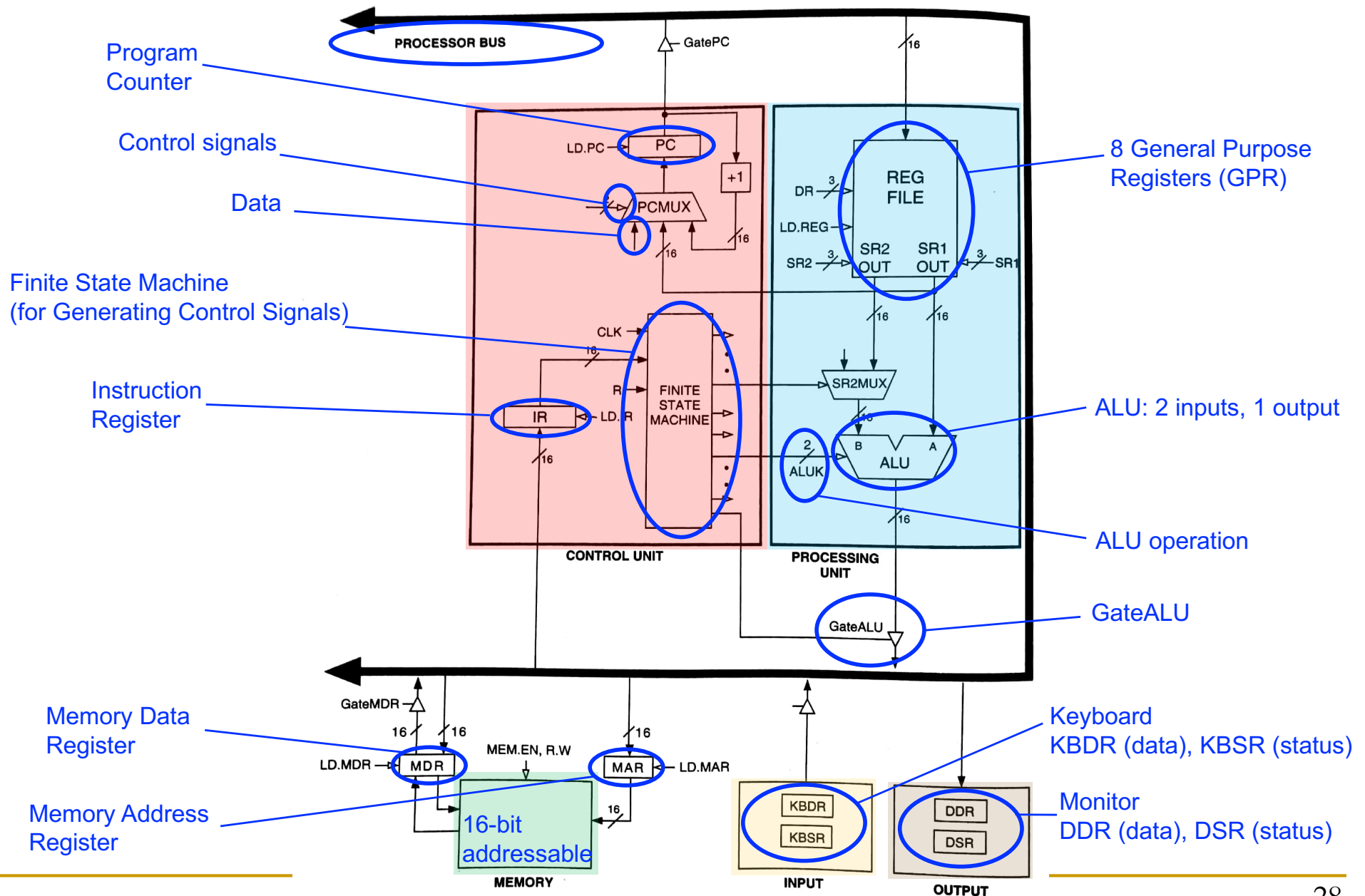
Stored Program & Sequential Execution

- Instructions and data are stored in memory
  - Typically the instruction length is the word length

- The processor fetches instructions from memory sequentially
  - Fetches one instruction
  - Decodes and executes the instruction
  - Continues with the next instruction

- The address of the current instruction is stored in the program counter (PC)

  - If word-addressable memory, the processor increments the PC by 1 (in LC-3)

  - If byte-addressable memory, the processor increments the PC by the word length (4 in MIPS)
    - In MIPS the OS typically sets the PC to 0x00400000

The Instruction

- An instruction the most basic unit of computer processing
  - Instructions are words in the language of a computer
  - Instruction Set Architecture (ISA) is the vocabulary

- The language of the computer can be written as

  - Machine language: Computer-readable representation (that is, 0's and 1's)

  - Assembly language: Human-readable representation

- We learn LC-3 instructions and MIPS instructions

- Let us start with some examples of instructions

Instruction Types

- There are three main types of instructions

- Operate instructions
  - Execute instructions in the ALU

- Data movement instructions
  - Read from or write to memory

- Control flow instructions
  - Change the sequence of execution

An Example of Operate Instruction

- **Addition**

High-level code

```
a = b + c;
```

Assembly

```
add a, b, c
```

- ❏ add: mnemonic to indicate the operation to perform

- ❏ b, c: source operands

- ❏ a: destination operand

- ❏ a ← b + c

Registers

- **We map variables to registers**

Assembly

```
add a, b, c
```

LC-3 registers

```
b = R1

c = R2

a = R0
```

MIPS registers

```
b = $s1

c = $s2

a = $s0
```

From Assembly to Machine Code in LC-3

- Addition

LC-3 assembly

```
ADD  R0, R1, R2
```

Field Values

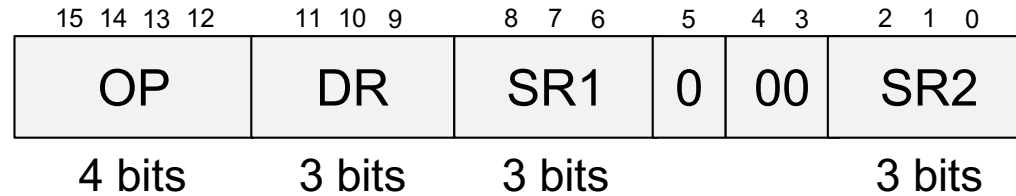| OP	DR	SR1			SR2
1	0	1	0	00	2

Machine Code

OP	DR	SR1			SR2
0 0 0 1	0 0 0	0 0 1	0	0 0	0 1 0
15 14 13 12	11 10 9	8 7 6	5	4 3	2 1 0

0x1042

Instruction Format or Encoding

- **LC-3**

15 14 13 12	11 10 9	8 7 6	5	4 3	2 1 0
OP	DR	SR1	0	00	SR2

4 bits   3 bits   3 bits   3 bits

- ❑ OP = opcode (what the instruction does)
  - E.g., ADD = 0001
    - ❑ DR ← SR1 + SR2
  - E.g., AND = 0101
    - ❑ DR ← SR1 AND SR2

- ❑ SR1, SR2 = source registers

- ❑ DR = destination register

Reading Operands from Memory

- With the operate instructions, such as addition, we tell the computer to execute arithmetic (or logic) computations in the ALU

- We also need instructions to access the operands from memory

- Next, we see how to read (or load) from memory

- Writing (or storing) is performed in a similar way, but we will talk about that later

Reading Word-Addressable Memory

- Load word

High-level code

`a = A[i];`

Assembly

`load a, A, i`

- ❑ load: mnemonic to indicate the load word operation

- ❑ A: base address

- ❑ i: offset
  - ▪ E.g., immediate or literal (a constant)

- ❑ a: destination operand

- ❑ a ← Memory[A + i]

Load Word in LC-3 and MIPS

- ## LC-3 assembly

High-level code

```
a = A[2];
```

LC-3 assembly

```
LDR   R3, R0, #2
```

R3 ← Memory[R0 + 2]

- ## MIPS assembly

High-level code

```
a = A[2];
```

MIPS assembly

```
lw    $s3, 2($s0)
```

$s3 ← Memory[$s0 + 2]

These instructions use a particular addressing mode
(i.e., the way the address is calculated), called base+offset

Instruction Format With Immediate

- ## LC-3

LC-3 assembly

```
LDR   R3, R0, #4
```

Field Values

OP	DR	BaseR	offset6
6	3	0	4

15   12 11   9 8   6 5   0

- ## MIPS

MIPS assembly

```
lw  $s3, 8($s0)
```

Field Values

op	rs	rt	imm
35	16	19	8

31   26 25   21 20   16 15   0

I-Type

How are these Instructions Executed?

- By using instructions we can speak the language of the computer

- Thus, we now know how to tell the computer to

  - Execute computations in the ALU by using, for instance, an addition

  - Access operands from memory by using the load word instruction

- But, how are these instructions executed on the computer?

  - The process of executing an instruction is called is the instruction cycle

The Instruction Cycle

- The instruction cycle is a sequence of steps or phases, that an instruction goes through to be executed
  - FETCH
  - DECODE
  - EVALUATE ADDRESS
  - FETCH OPERANDS
  - EXECUTE
  - STORE RESULT

- Not all instructions have the six phases
  - LDR does not require EXECUTE

  - ADD does not require EVALUATE ADDRESS

  - Intel x86 instruction ADD [eax], edx is an example of instruction with six phases

After STORE RESULT, a New FETCH

- FETCH
- DECODE
- EVALUATE ADDRESS
- FETCH OPERANDS
- EXECUTE
- STORE RESULT
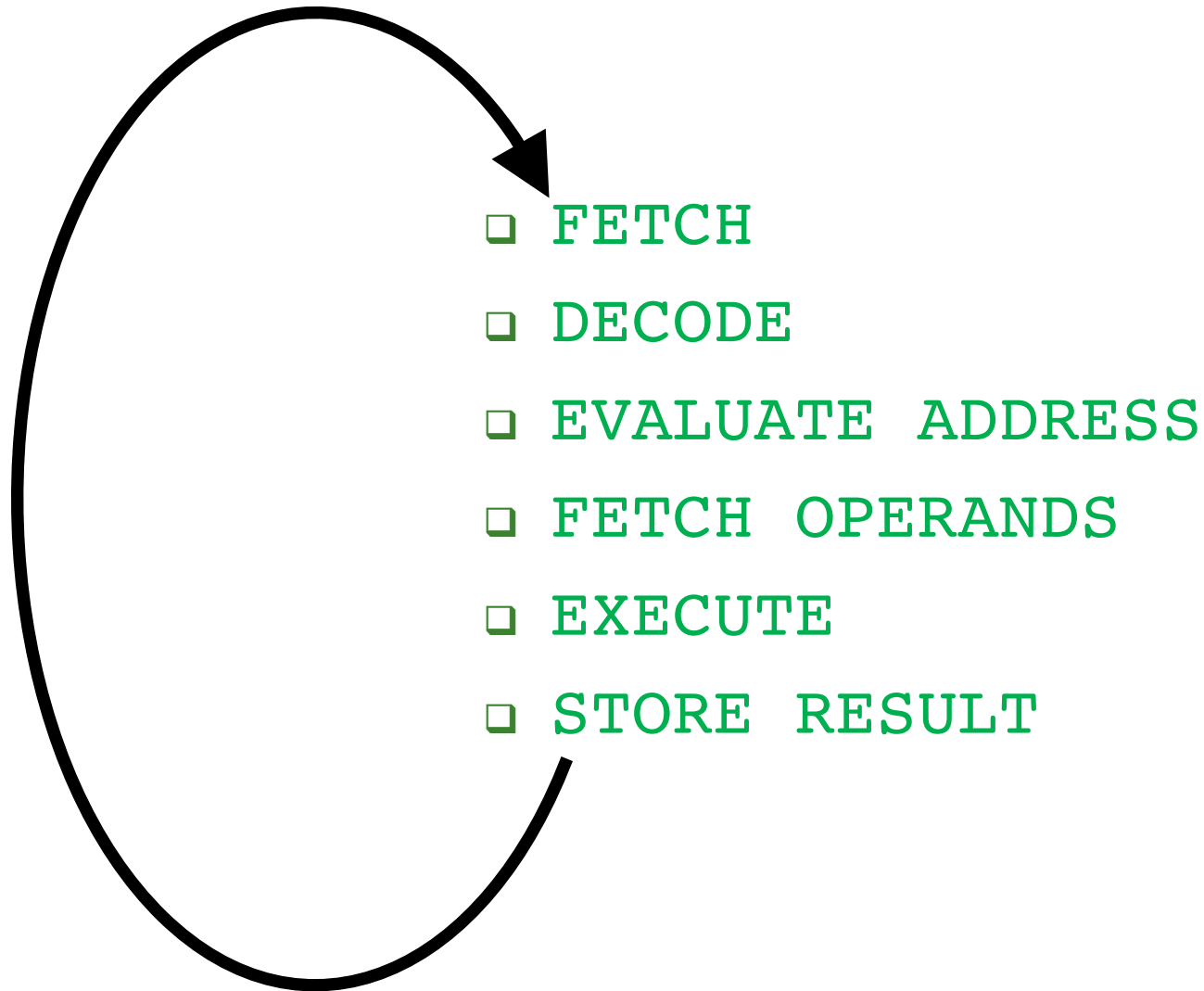
FETCH

- The FETCH phase obtains the instruction from memory and loads it into the instruction register

- This phase is common to every instruction type

- Complete description
  - Step 1: Load the MAR with the contents of the PC, and simultaneously increment the PC

  - Step 2: Interrogate memory. This results the instruction to be placed in the MDR

  - Step 3: Load the IR with the contents of the MDR

FETCH in LC-3

Step 1: Load MAR and increment PC

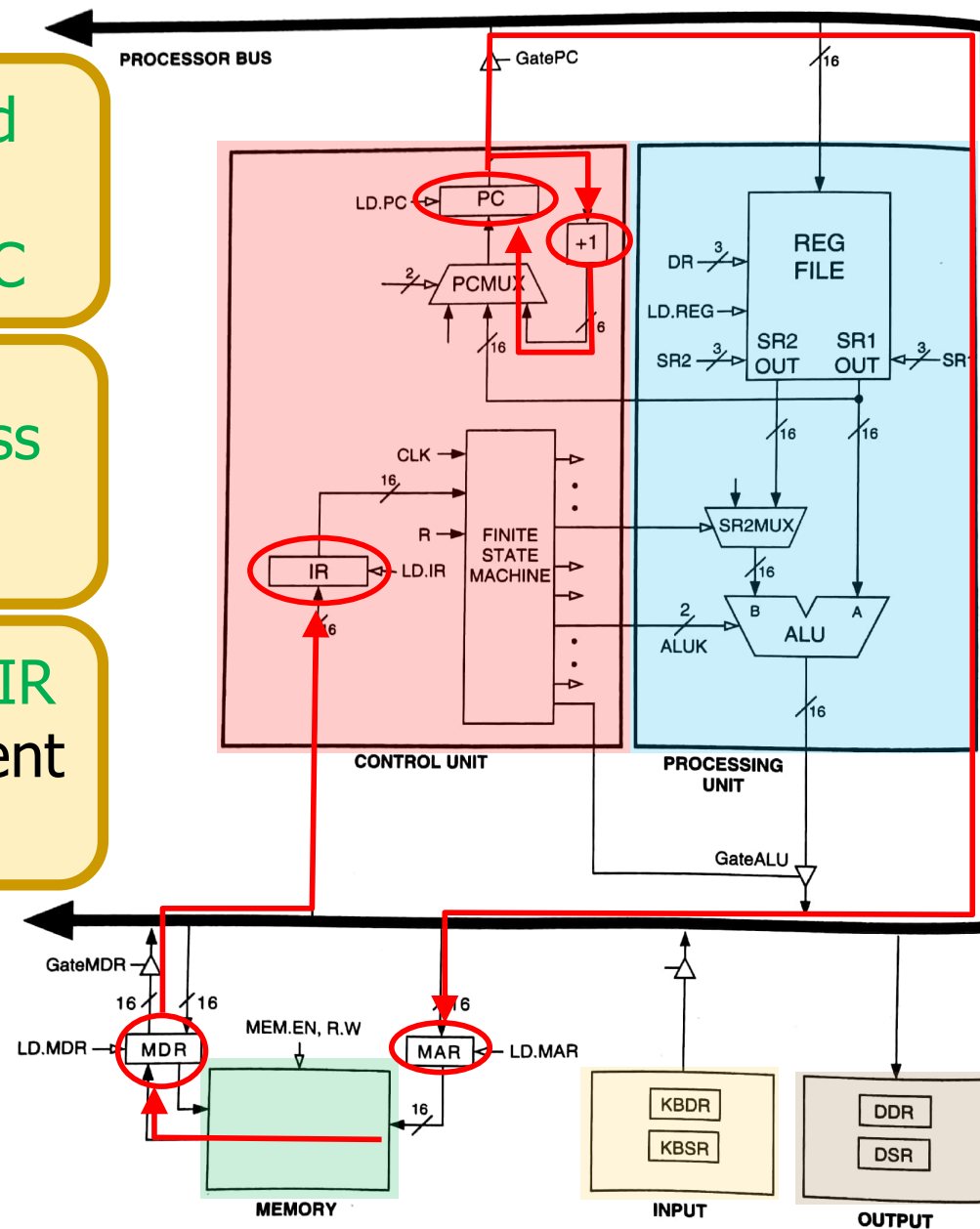Step 2: Access memory

Step 3: Load IR with the content of MDR

Figure 4.3   The LC-3 as an example of the von Neumann model

48

DECODE

- The DECODE phase identifies the instruction

- Recall the decoder (Lecture 6, Slides 26-27)

  - A 4-to-16 decoder identifies which of the 16 opcodes is going to be processed

- The input is the four bits IR[15:12]

- The remaining 12 bits identify what else is needed to process the instruction

DECODE in LC-3

DECODE *identifies the* **instruction** *to be processed*
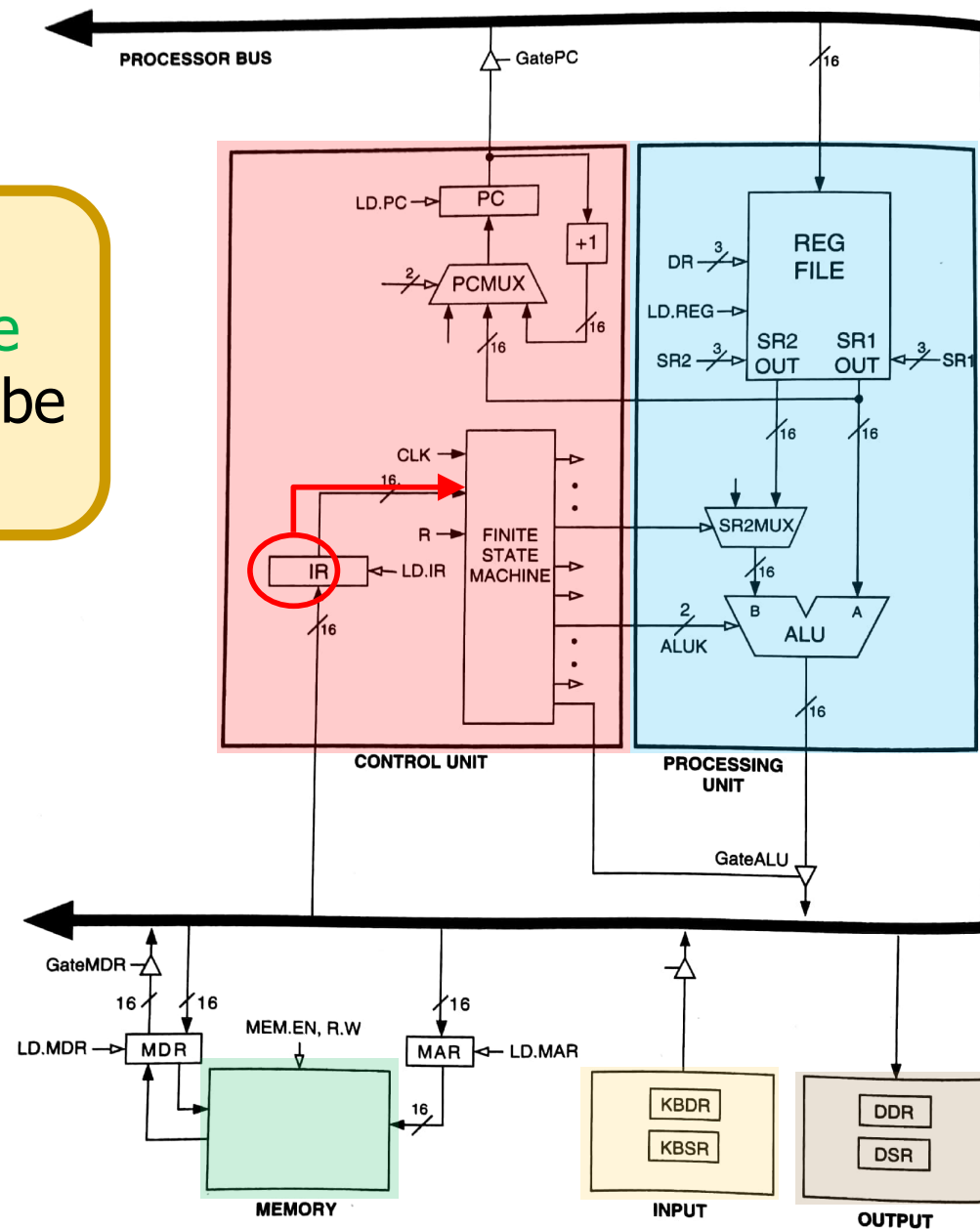
Figure 4.3   The LC-3 as an example of the von Neumann model

EVALUATE ADDRESS

- The EVALUATE ADDRESS phase computes the address of the memory location that is needed to process the instruction

- This phase is necessary in LDR

  - It computes the address of the data word that is to be read from memory

  - By adding an offset to the content of a register

- But not necessary in ADD

EVALUATE ADDRESS in LC-3

LDR calculates the address by adding a register and an immediate

Figure 4.3   The LC-3 as an example of the von Neumann model

52
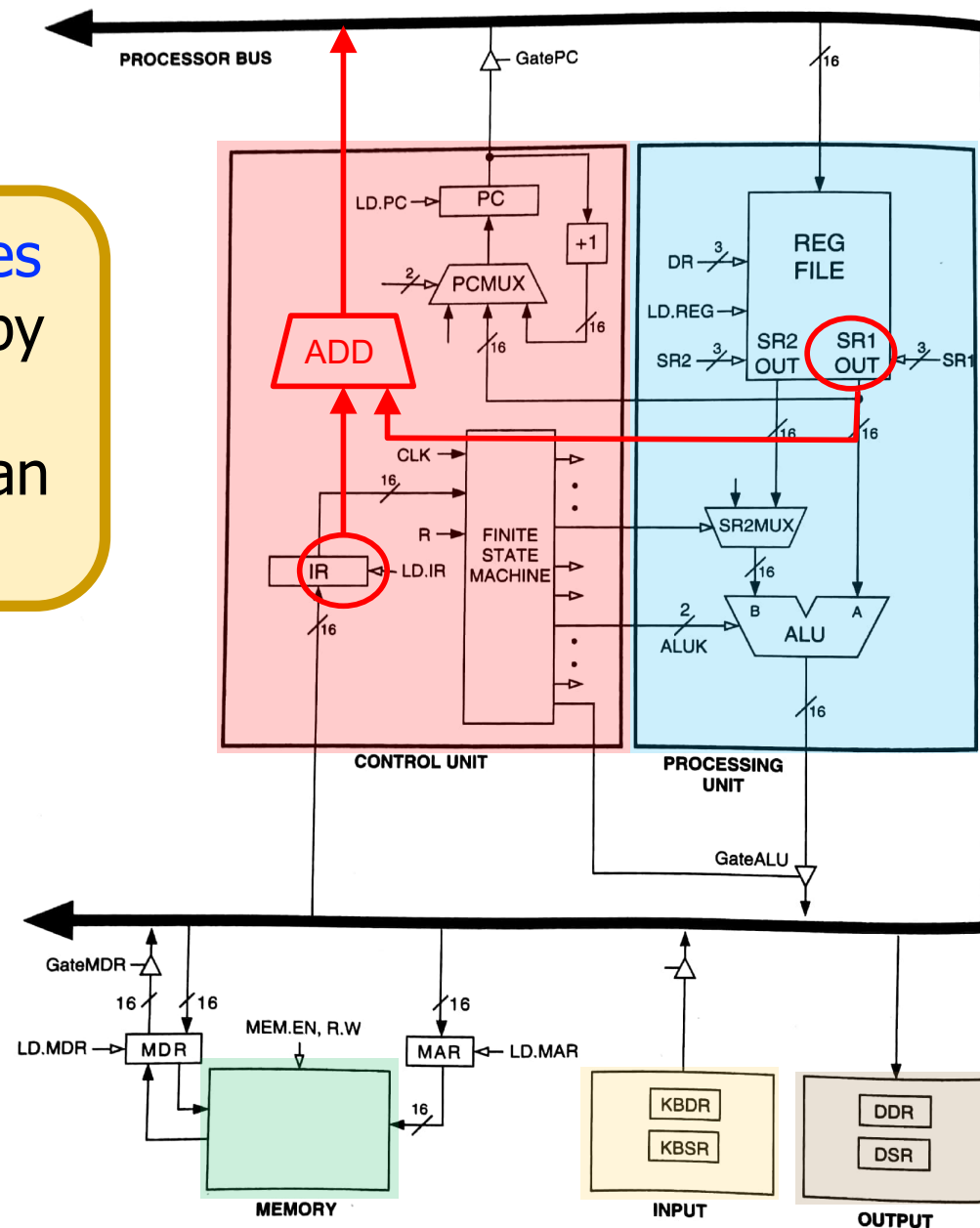
FETCH OPERANDS

- The FETCH OPERANDS phase obtains the source operands needed to process the instruction

- In LDR
  - Step 1: Load MAR with the address calculated in EVALUATE ADDRESS

  - Step 2: Read memory, placing source operand in MDR

- In ADD
  - Obtain the source operands from the register file

  - In most current microprocessors, this phase can be done at the same time the instruction is being decoded

FETCH OPERANDS in LC-3

LDR loads MAR (step 1), and places the results in MDR (step 2)
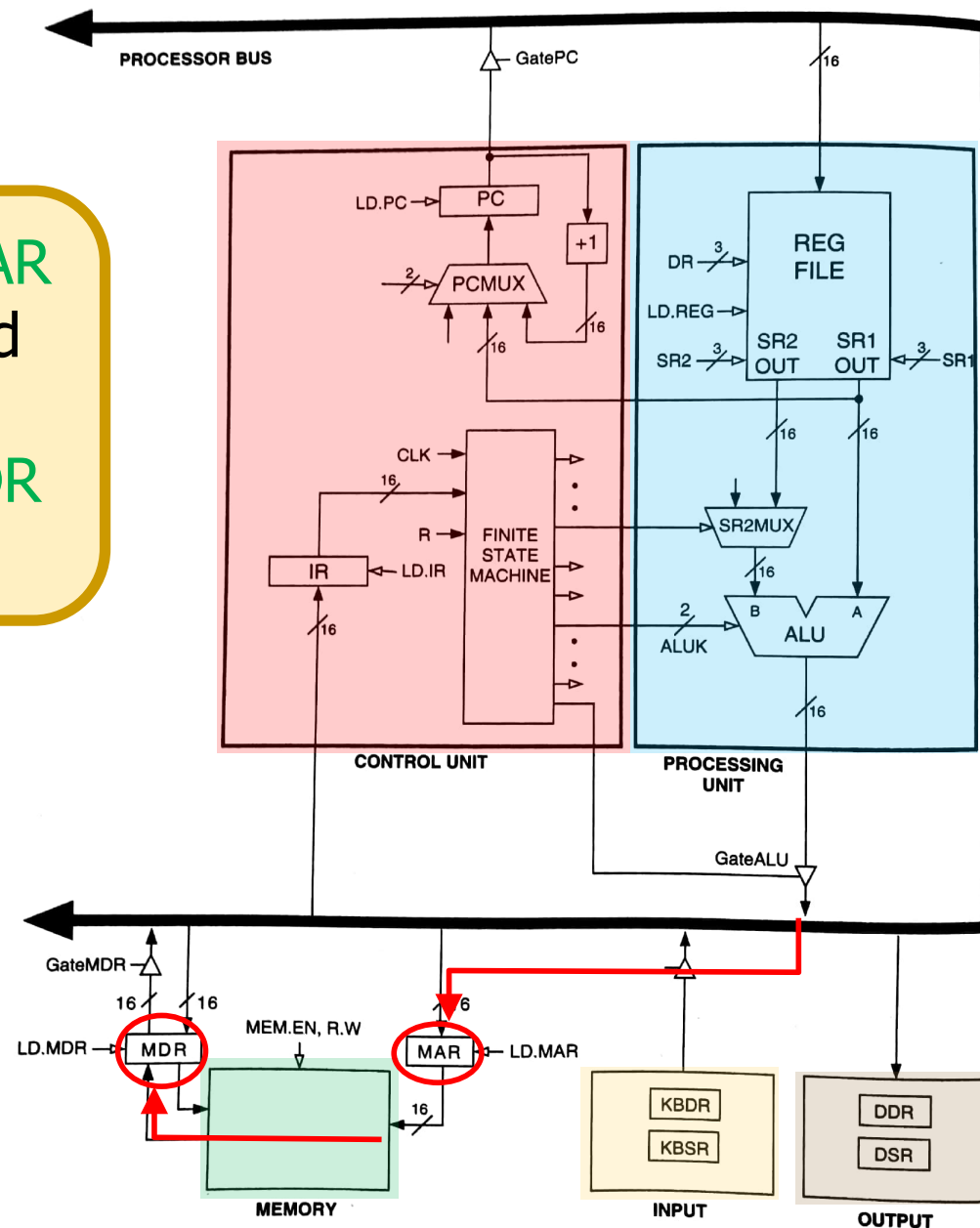
Figure 4.3   The LC-3 as an example of the von Neumann model

EXECUTE

- The EXECUTE phase executes the instruction

  - In ADD, it performs addition in the ALU

EXECUTE in LC-3

ADD adds SR1 and SR2
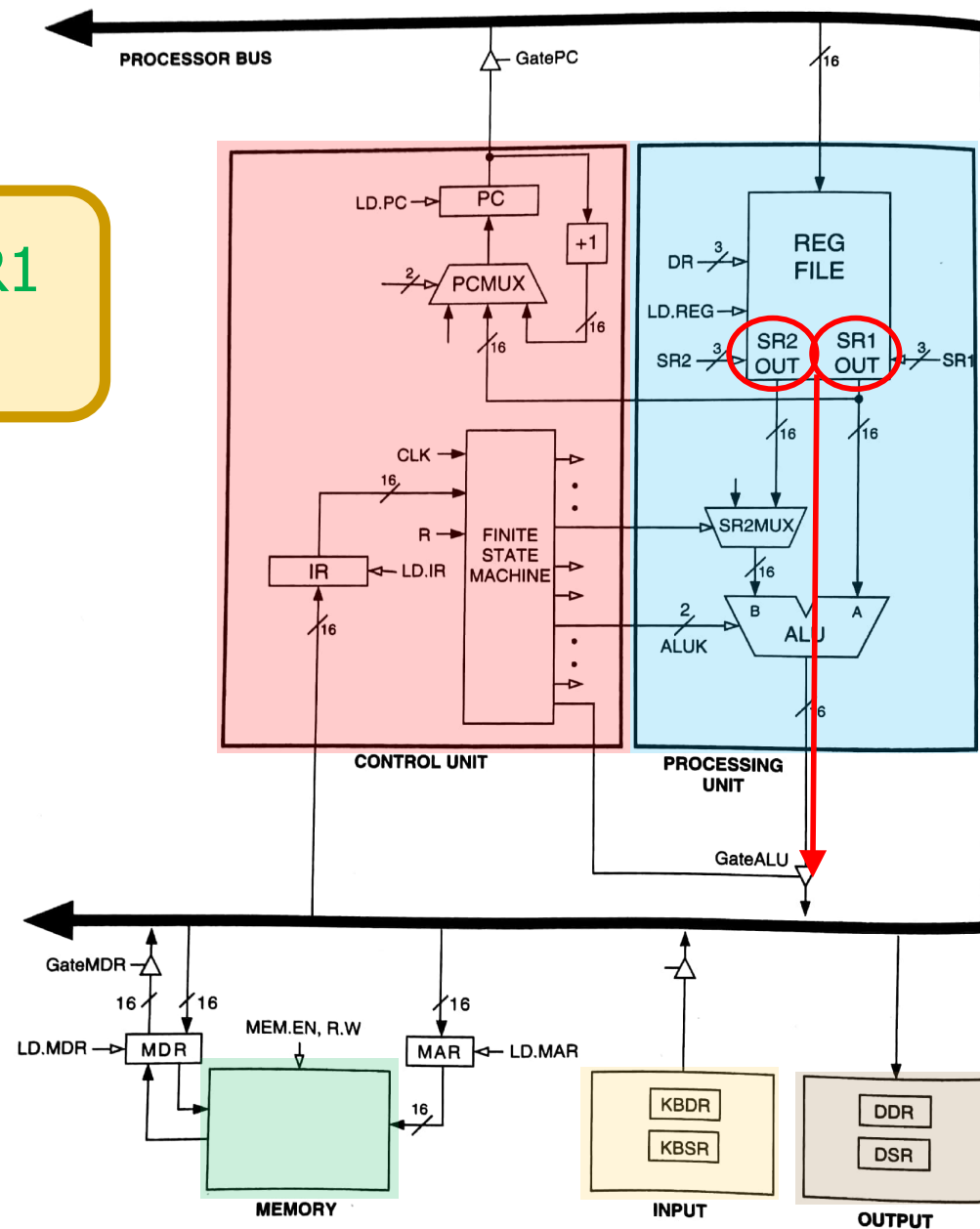
Figure 4.3   The LC-3 as an example of the von Neumann model

STORE RESULT

- The STORE RESULT phase writes to the designated destination

- Once STORE RESULT is completed, a new instruction cycle starts (with the FETCH phase)

STORE RESULT in LC-3

LDR loads MDR into DR
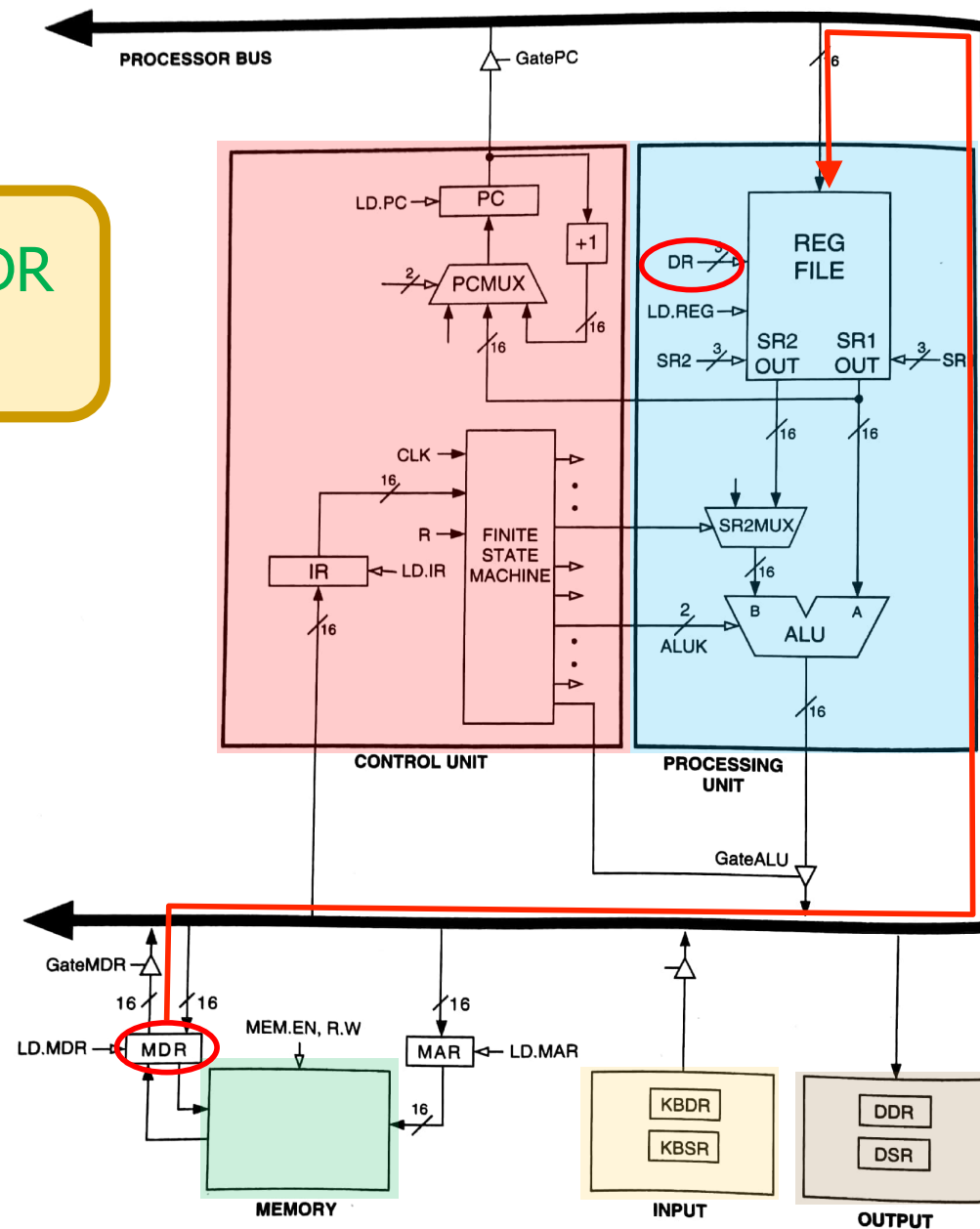
Figure 4.3   The LC-3 as an example of the von Neumann model

The Instruction Cycle

- FETCH
- DECODE
- EVALUATE ADDRESS
- FETCH OPERANDS
- EXECUTE
- STORE RESULT
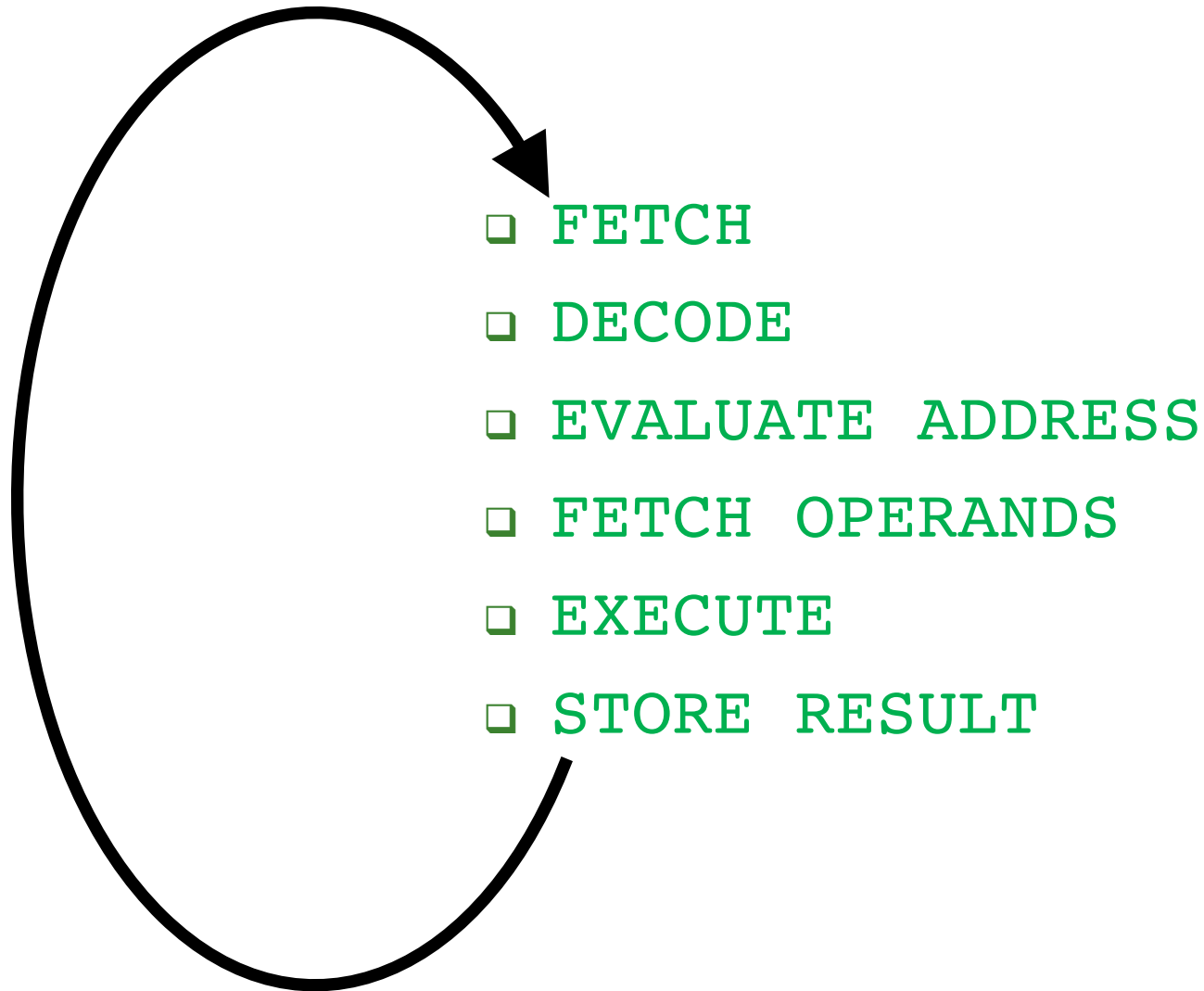
Changing the Sequence of Execution

- A computer program executes in sequence (i.e., in program order)
  - First instruction, second instruction, third instruction and so on

- Unless we change the sequence of execution

- Control instructions allow a program to execute out of sequence
  - They can change the PC by loading it during the EXECUTE phase

  - That wipes out the incremented PC (loaded during the FETCH phase)

Jump in LC-3

- **Unconditional branch or jump**

- **LC-3** `JMP   R2`

1100	000	BaseR	000000

    4 bits   3 bits
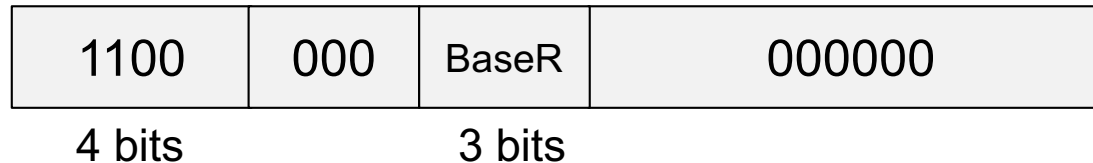
- ❑ BaseR = Base register
- ❑ PC ← R2 (Register identified by BaseR)

This is register addressing mode

- ❑ Variations
  - ■ RET: special case of JMP where BaseR = R7
  - ■ JSR, JSRR: jump to subroutine

LC-3 Data Path

Figure 4.3 The LC-3 as an example of the von Neumann model

63
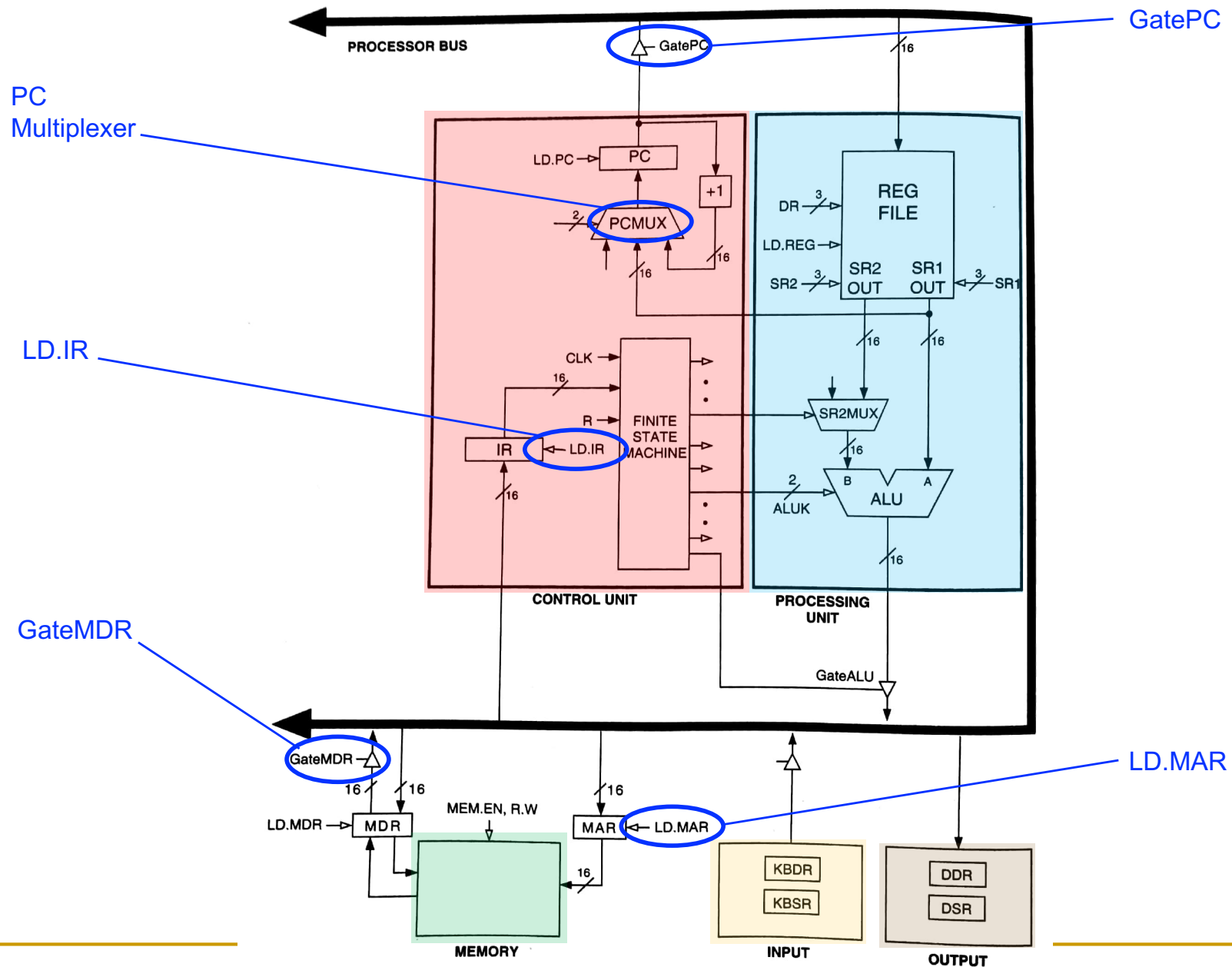
Control of the Instruction Cycle

Figure 4.4   An abbreviated state diagram of the LC-3
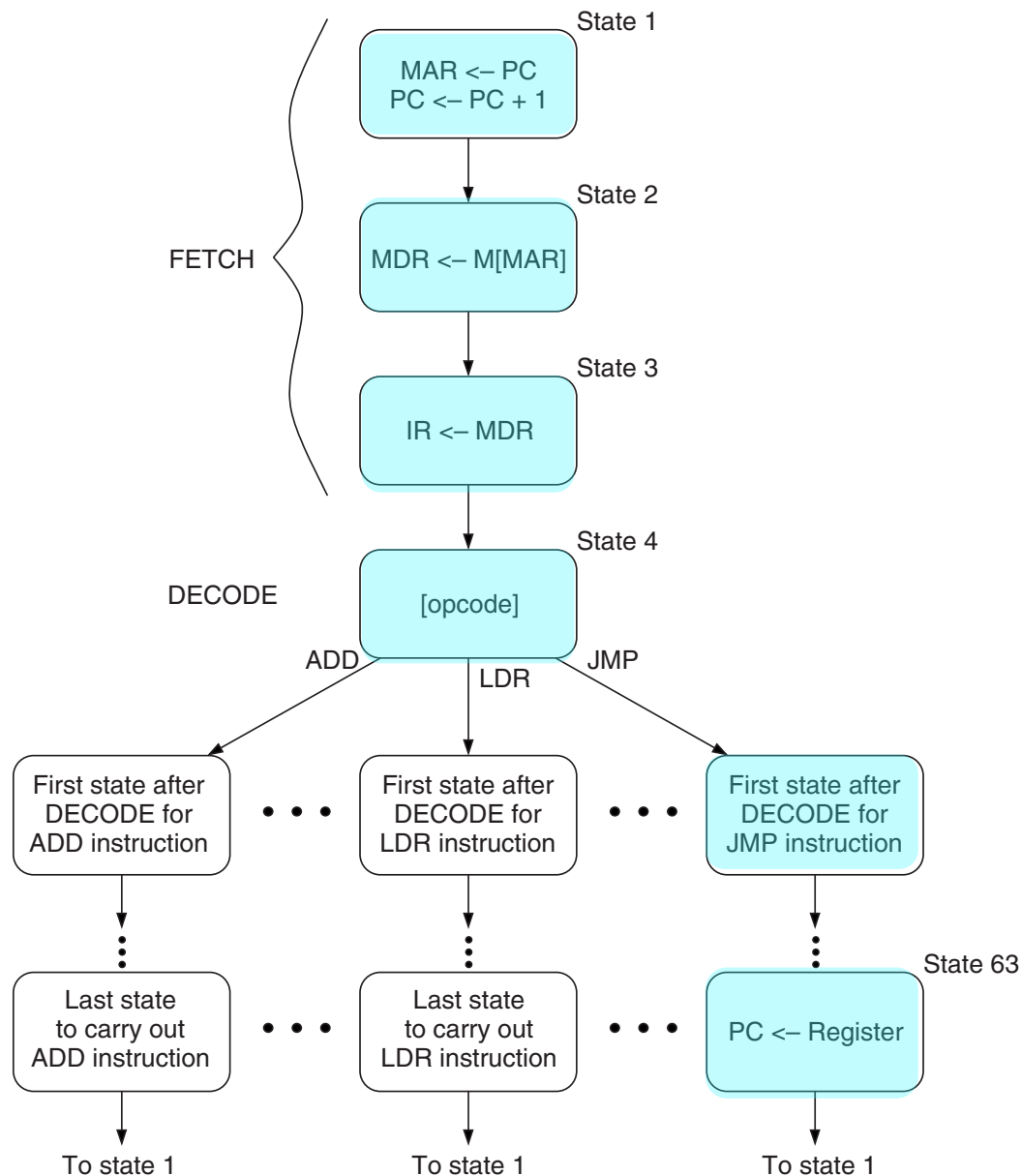
- State 1
  - The FSM asserts GatePC and LD.MAR
  - It selects input (+1) in PCMUX and asserts LD.PC

- State 2
  - MDR is loaded with the instruction

- State 3
  - The FSM asserts GateMDR and LD.IR

- State 4
  - The FSM goes to next state depending on opcode

- State 63
  - JMP loads register into PC

- Full state diagram in Patt&Pattel, Appendix C

The Instruction Cycle

- FETCH
- DECODE
- EVALUATE ADDRESS
- FETCH OPERANDS
- EXECUTE
- STORE RESULT
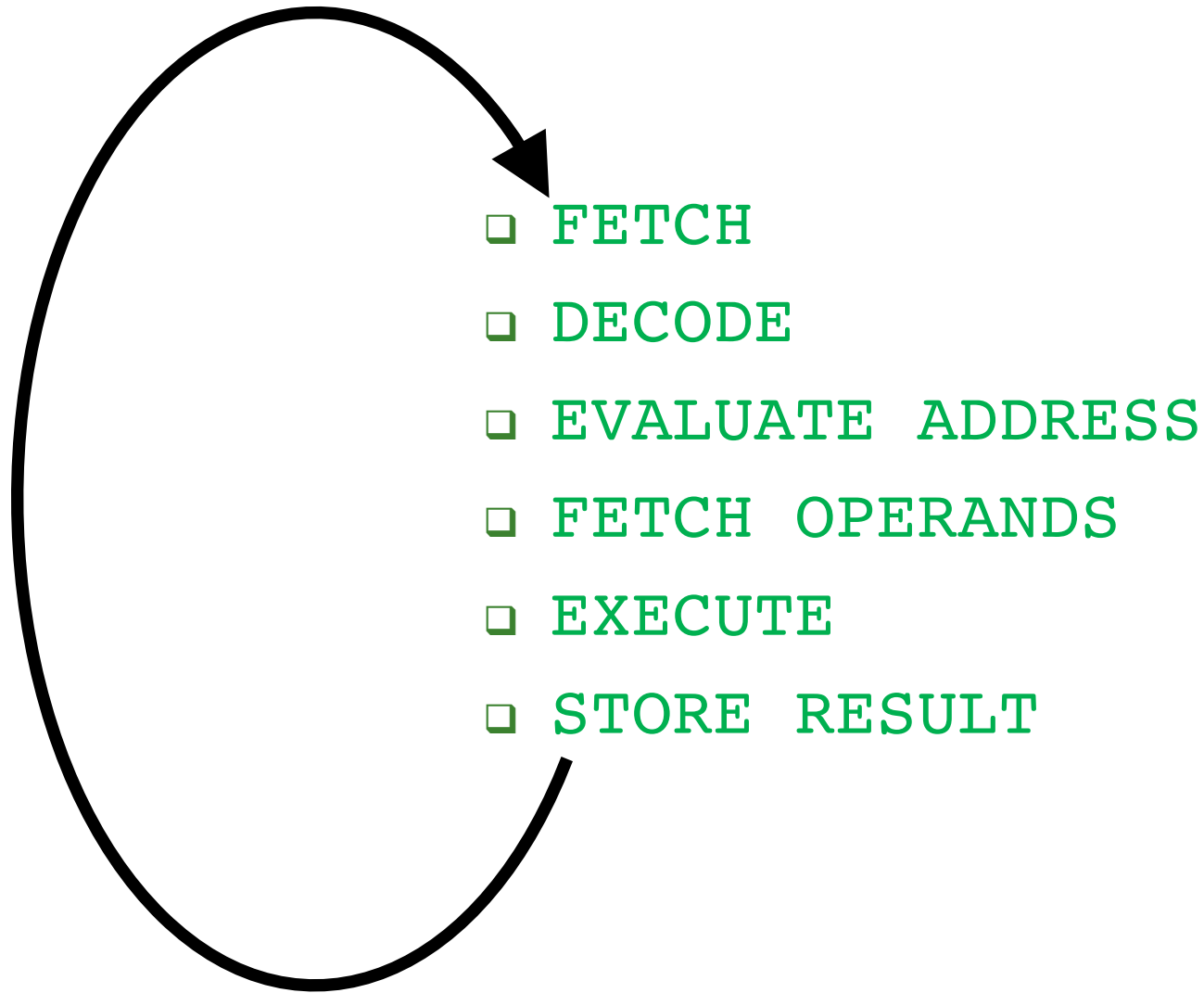
LC-3 and MIPS Instruction Set Architectures

The Instruction Set Architecture

- The ISA is the interface between what the software commands and what the hardware carries out

- The ISA specifies
  - The memory organization
    - Address space (LC-3: 2^{16}, MIPS: 2^{32})
    - Addressability (LC-3: 16 bits, MIPS: 32 bits)
    - Word- or Byte-addressable

  - The register set
    - R0 to R7 in LC-3
    - 32 registers in MIPS

  - The instruction set
    - Opcodes
    - Data types
    - Addressing modes

| Problem |
| Algorithm |
| Program |
| ISA |
| Microarchitecture |
| Circuits |
| Electrons |

The Instruction Set

- It defines opcodes, data types, and addressing modes
- ADD and LDR have been our first examples

ADD

OP	DR	SR1			SR2
1	0	1	0	00	2

Register mode

LDR

OP	DR	BaseR	offset6
6	3	0	4

Base+offset mode

Opcodes

- Large or small sets of opcodes could be defined
  - E.g, HP Precision Architecture: an instruction for A*B+C
  - E.g, x86: multimedia extensions
  - E.g, VAX: opcode to save all information of one program prior to switching to another program

- Tradeoffs are involved
  - Hardware complexity vs. software complexity

- In LC-3 and in MIPS there are three types of opcodes
  - Operate
  - Data movement
  - Control

Opcodes in LC-3

	15 14 13 12	11	10	9	8	7	6	5	4	3	2	1	0
ADD+	0001	DR			SR1			0	0	0	SR2		
ADD+	0001	DR			SR1			1	imm5				
AND+	0101	DR			SR1			0	0	0	SR2		
AND+	0101	DR			SR1			1	imm5				
BR	0000	n	z	p	PCoffset9								
JMP	1100	0	0	0	BaseR			000000					
JSR	0100	1	PCoffset11										
JSRR	0100	0	0	0	BaseR			000000					
LD+	0010	DR			PCoffset9								
LDI+	1010	DR			PCoffset9								
LDR+	0110	DR			BaseR			offset6					
LEA+	1110	DR			PCoffset9								
NOT+	1001	DR			SR			111111					
RET	1100	0	0	0	111			000000					
RTI	1000	000000000000											
ST	0011	SR			PCoffset9								
STI	1011	SR			PCoffset9								
STR	0111	SR			BaseR			offset6					
TRAP	1111	0000			trapvect8								
reserved	1101												

Figure 5.3   Formats of the entire LC-3 instruction set. NOTE: + indicates instructions that modify condition codes

70

Data Types

- An ISA supports one or several data types

- LC-3 only supports 2's complement integers

- MIPS supports
  - 2's complement integers
  - Unsigned integers
  - Floating point

- Again, tradeoffs are involved

Data Type Tradeoffs

- What is the benefit of having more or high-level data types in the ISA?

- What is the disadvantage?

- Think compiler/programmer vs. microarchitect

- Concept of semantic gap
  - Data types coupled tightly to the semantic level, or complexity of instructions

- Example: Early RISC architectures vs. Intel 432
  - Early RISC (e.g., MIPS): Only integer data type
  - Intel 432: Object data type, capability based machine

Addressing Modes

- An addressing mode is a mechanism for specifying where an operand is located

- There five addressing modes in LC-3
    - Immediate or literal (constant)
        - The operand is in some bits of the instruction
    - Register
        - The operand is in one of R0 to R7 registers
    - Three of them are memory addressing modes
        - PC-relative
        - Indirect
        - Base+offset

- In addition, MIPS has pseudo-direct addressing (for j and jal), but does not have indirect addressing

Operate Instructions

Operate Instructions

- In LC-3, there are three operate instructions
  - NOT is a unary operation (one source operand)
    - It executes bitwise NOT

  - ADD and AND are binary operations (two source operands)
    - ADD is 2's complement addition
    - AND is bitwise SR1 & SR2

- In MIPS, there are many more
  - Most of R-type instructions (they are binary operations)
    - E.g., add, and, nor, xor…
  - I-type versions of the R-type operate instructions
  - F-type operations, i.e., floating-point operations

NOT in LC-3

- NOT assembly and machine code

LC-3 assembly

```
NOT   R3, R5
```

Field Values

OP	DR	SR	
9	3	5	1 1 1 1 1 1

Machine Code

OP	DR	SR	
1 0 0 1	0 1 1	0 0 1	1 1 1 1 1 1
15 12	11 9	8 6	5 0

Register file

R0	
R1	
R2	
R3	0101000011110000	DR
R4	
R5	1010111100001111	SR
R6	
R7	

16   16

B   A

NOT ───► ALU

From FSM
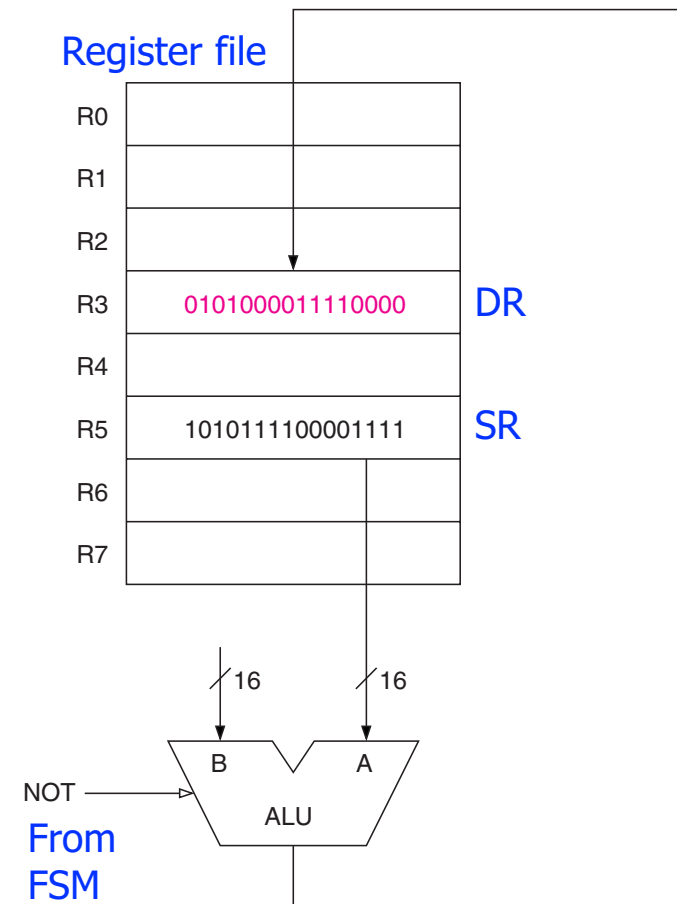
There is no NOT in MIPS. How is it implemented?

Operate Instructions

- We are already familiar with LC-3's ADD and AND with register mode (R-type in MIPS)

- Now let us see the versions with one literal (i.e., immediate) operand

- Subtraction is another necessary operation
  - How is it implemented in LC-3 and MIPS?

Operate Instr. with one Literal in LC-3

- ADD and AND

OP	DR	SR1	1	imm5
4 bits	3 bits	3 bits		5 bits
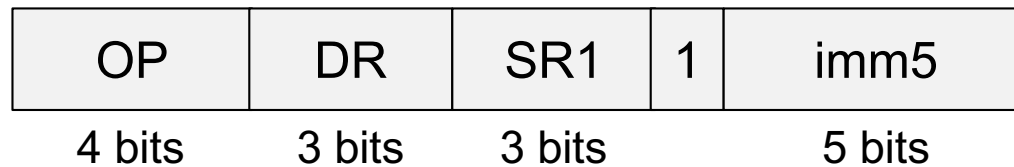
- ❑ OP = operation
  - ■ E.g., ADD = 0001 (same OP as the register-mode ADD)
    - ❑ DR ← SR1 + sign-extend(imm5)

  - ■ E.g., AND = 0101 (same OP as the register-mode AND)
    - ❑ DR ← SR1 AND sign-extend(imm5)

- ❑ SR1 = source register

- ❑ DR = destination register

- ❑ imm5 = Literal or immediate (sign-extend to 16 bits)

ADD with one Literal in LC-3

- ADD assembly and machine code

LC-3 assembly

```
ADD R1, R4, #-2
```

Field Values

OP	DR	SR		imm5
1	1	4	1	-2

Machine Code

OP	DR	SR		imm5
0 0 0 1	0 0 1	1 0 0	1	1 1 1 1 0

15   12   11   9   8   6   5   4   0

Register file

R0	
R1	0000000000000100	**DR**
R2	
R3	
R4	0000000000000110	**SR**
R5	
R6	
R7	

Instruction register

ADD R1 R4 −2

IR: 0001 001 100 1 11110

/5

SEXT **Sign-extend**

/16

1111111111111110

Bit[5]

1 0

/16

ADD → ALU

From FSM

B A
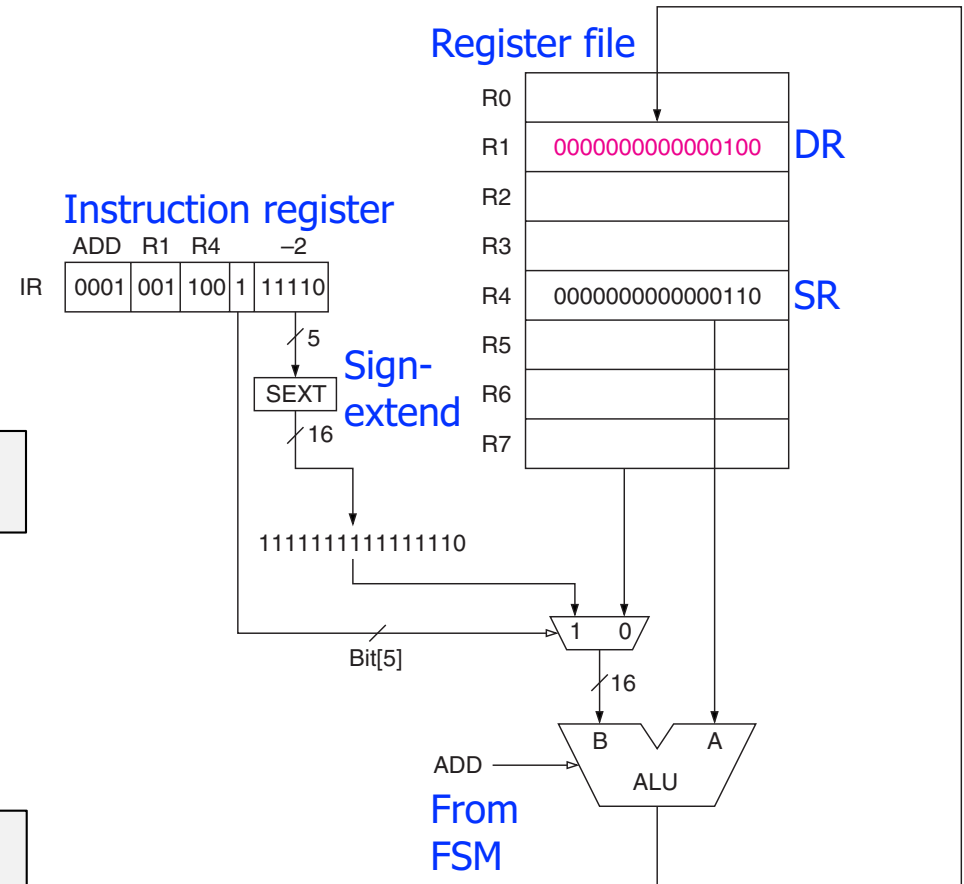
Subtract in LC-3

- **MIPS assembly**

High-level code

```
a = b + c – d;
```

MIPS assembly

```
add   $t0, $s0, $s1
sub   $s3, $t0, $s2
```

- **LC-3 assembly**

High-level code

```
a = b + c – d;
```

LC-3 assembly

```
ADD   R2, R0, R1
NOT   R4, R3
ADD   R5, R4, #1
ADD   R6, R2, R5
```

2's complement of R4

- **Tradeoff in LC-3**
  - More instructions
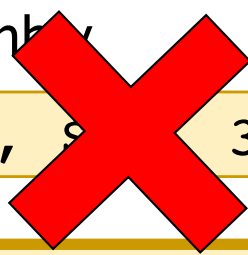 - But, simpler control logic

Subtract Immediate

- ## MIPS assembly

High-level code

```
a = b - 3;
```

MIPS assembly

```
subi $s1, $s0, 3
```

Is subi necessary in MIPS?

MIPS assembly

```
addi $s1, $s0, -3
```

- ## LC-3

High-level code

```
a = b - 3;
```

LC-3 assembly

```
ADD  R1, R0, #-3
```

Data Movement Instructions and Addressing Modes

Data Movement Instructions

- In LC-3, there are seven data movement instructions
  - LD, LDR, LDI, LEA, ST, STR, STI

- Format of load and store instructions
  - Opcode (bits [15:12])
  - DR or SR (bits [11:9])
  - Address generation bits (bits [8:0])
  - Four ways to interpret bits, called addressing modes
    - PC-Relative Mode
    - Indirect Mode
    - Base+offset Mode
    - Immediate Mode

- In MIPS, there are only Base+offset and immediate modes for load and store instructions

PC-Relative Addressing Mode

- LD (Load) and ST (Store)

	15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
	OP	DR/SR	PCoffset9
	4 bits	3 bits	9 bits
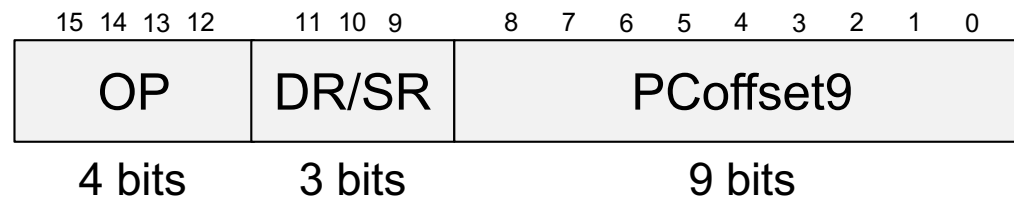
- OP = opcode
  - E.g., LD = 0010
  - E.g., ST = 0011

- DR = destination register in LD
- SR = source register in ST

- LD: DR ← Memory[PC† + sign-extend(PCoffset9)]

- ST: Memory[PC† + sign-extend(PCoffset9)] ← SR

$\dagger$ This is the incremented PC

LD in LC-3

- **LD assembly and machine code**

LC-3 assembly

```
LD R2, 0x1AF
```

Field Values

OP	DR	PCoffset9
2	2	0x1AF

Machine Code

OP	DR	PCoffset9
0 0 1 0	0 1 0	1 1 0 1 0 1 1 1 1

15   12 11   9   8   0

Instruction register

15   0

IR | 0010 | 010 | 110101111

LD   R2   x1AF

Incremented PC

PC | 0100 0000 0001 1001

IR[8:0]

SEXT Sign-extend

16

1111111110101111

16

ADD

16

1. Address calculation ①

MAR

MEMORY

② 2. Memory read

Register file

R0
R1
R2 | 0000000000000101 DR
R3
R4
R5
R6
R7

3. DR is loaded

③

16

MDR

The memory address is only +256 to -255 locations away of the LD or ST instruction
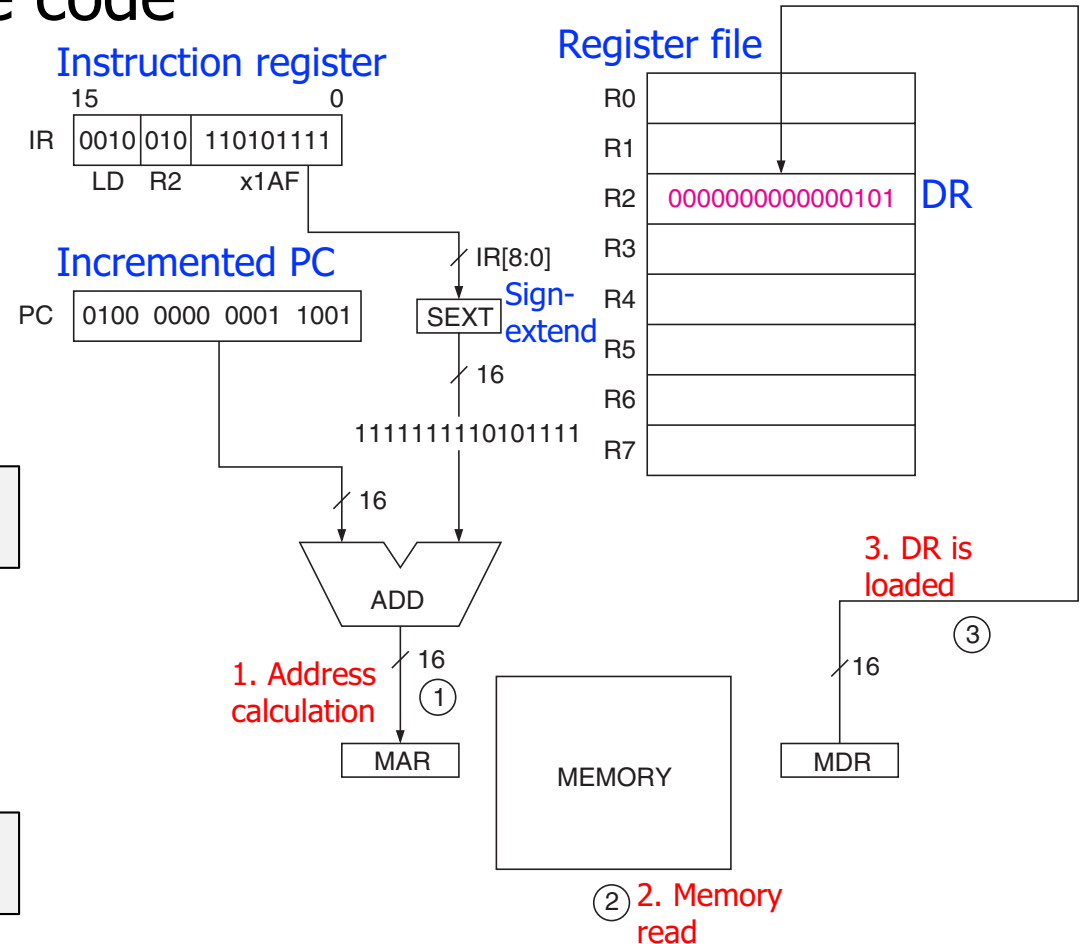
Limitation: The PC-relative addressing mode cannot address far away from the instruction

Indirect Addressing Mode

- LDI (Load Indirect) and STI (Store Indirect)

	15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
	OP	DR/SR	PCoffset9
	4 bits	3 bits	9 bits

- ❑ OP = opcode
  - E.g., LDI = 1010
  - E.g., STI = 1011

- ❑ DR = destination register in LDI
- ❑ SR = source register in STI

- ❑ LDI: DR ← Memory[Memory[PC† + sign-extend(PCoffset9)]]

- ❑ STI: Memory[Memory[PC† + sign-extend(PCoffset9)]] ← SR

† This is the incremented PC

LDI in LC-3

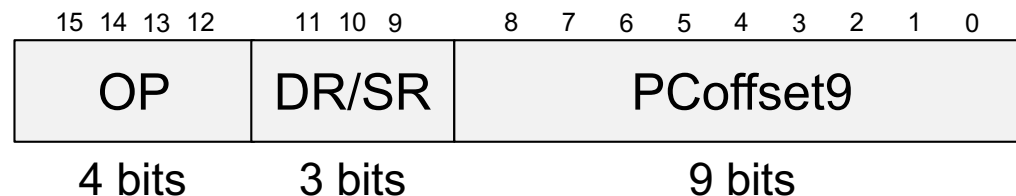- LDI assembly and machine code

LC-3 assembly

LDI R3, 0x1CC

Field Values

OP	DR	PCoffset9
A	3	0x1CC

Machine Code

OP	DR	PCoffset9
1 0 1 0	0 1 1	1 1 1 0 0 1 1 0 0

15   12   11 9   8   0

Instruction register

	15		0
IR	1010	011	111001100
	LDI	R3	x1CC

Incremented PC

PC | 0100 1010 0001 1100

IR[8:0]

SEXT

Sign-extend

16

xFFCC

16

ADD

1. Address calculation ①

16

MAR

3. Loaded address ③ x2110 from MDR to MAR

MEMORY

2. Memory read ② 4. Memory read ④

Register file

R0
R1
R2
R3 1111111111111111 DR
R4
R5
R6
R7

5. DR is loaded ⑤

16

MDR
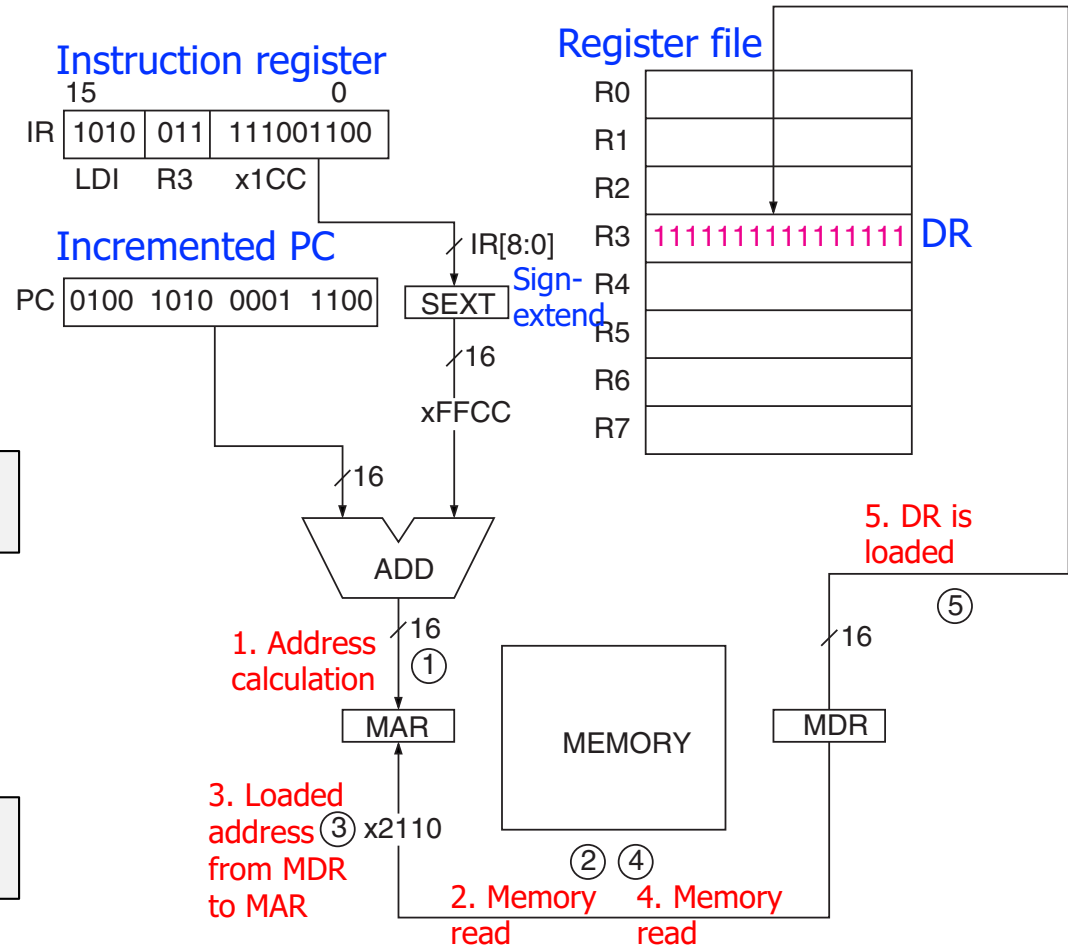
Now the address of the operand can be anywhere in the memory

93

Base+Offset Addressing Mode

- LDR (Load Register) and STR (Store Register)

15 14 13 12	11 10 9	8 7 6	5 4 3 2 1 0
OP	DR/SR	BaseR	offset6
4 bits	3 bits	3 bits	6 bits
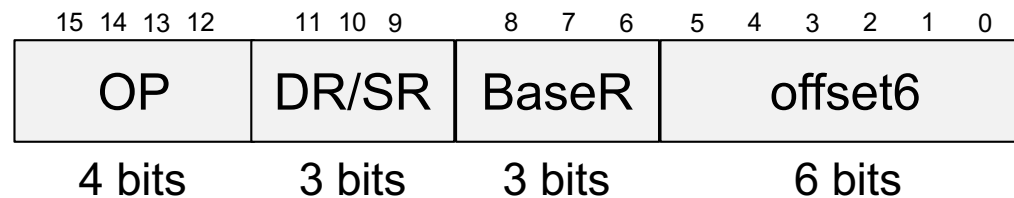
- OP = opcode
  - E.g., LDR = 0110
  - E.g., STR = 0111

- DR = destination register in LDR
- SR = source register in STR

- LDR: DR ← Memory[BaseR + sign-extend(offset6)]

- STR: Memory[BaseR + sign-extend(offset6)] ← SR

LDR in LC-3

- LDR assembly and machine code

LC-3 assembly

LDR R1, R2, 0x1D

Field Values

OP	DR	BaseR	offset6
6	1	2	0x1D

Machine Code

OP	DR	BaseR	offset6
0 1 1 0	0 0 1	0 1 0	0 1 1 1 0 1

15   12   11 9   8   6   5   0

Instruction register

15   0
IR | 0110 | 001 | 010 | 011101 |
    LDR   R1   R2   x1D

IR[5:0]

SEXT — Sign-extend

16

x001D

ADD

1. Address calculation ①

MAR

16

MEMORY

② 2. Memory read

Register file

| R0 | |
| R1 | 0000111100001111 | DR
| R2 | 0010001101000101 | BaseR
R3	
R4	
R5	
R6	
R7	

3. DR is loaded

③

16

MDR
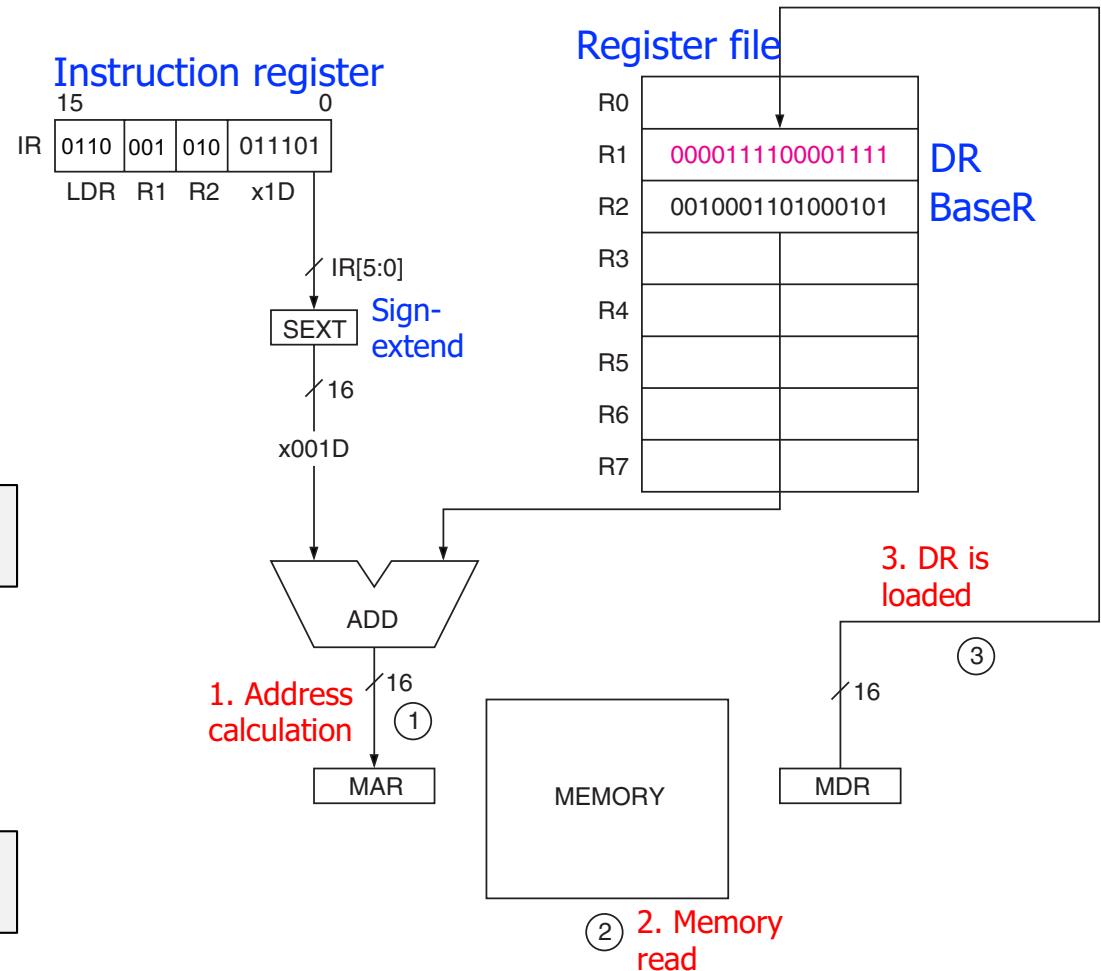
The address of the operand can also be anywhere in the memory

95

An Example Program in MIPS and LC-3

High-level code

```
a    = A[0];
c    = a + b - 5;
B[0] = c;
```

MIPS registers

```
A = $s0
b = $s2
B = $s1
```

LC-3 registers

```
A = R0
b = R2
B = R1
```
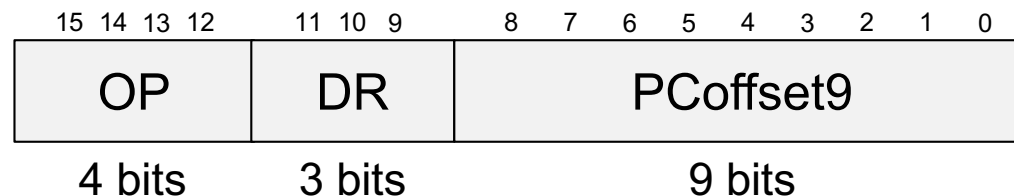
MIPS assembly

```
lw   $t0, 0($s0)
add  $t1, $t0, $s2
addi $t2, $t1, -5
sw   $t2, 0($s1)
```

LC-3 assembly

```
LDR  R5, R0, #0
ADD  R6, R5, R2
ADD  R7, R6, #-5
STR  R7, R1, #0
```

Immediate Addressing Mode

- **LEA (Load Effective Address)**

15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
OP	DR	PCoffset9
4 bits	3 bits	9 bits

- ❏ OP = 1110

- ❏ DR = destination register

- ❏ LEA: DR ← PC† + sign-extend(PCoffset9)

> What is the difference from PC-Relative addressing mode?

> Answer: Instructions with PC-Relative mode access memory, but LEA does not

† This is the incremented PC

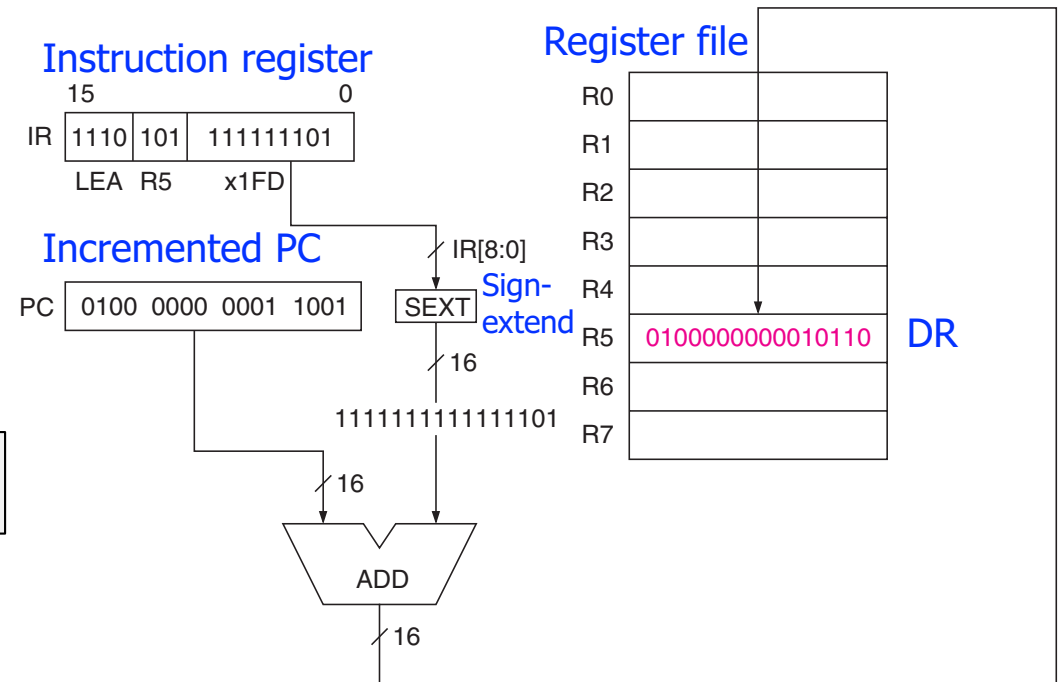# LEA in LC-3

- LEA assembly and machine code

LC-3 assembly

```
LEA R5, #-3
```

Field Values

OP	DR	PCoffset9
E	5	0x1FD

Machine Code

OP	DR	PCoffset9
1 1 1 0	1 0 1	1 1 1 1 1 1 1 0 1

15   12 11   9 8   0

Instruction register

15   0

IR | 1110 | 101 | 111111101 |

LEA R5   x1FD

Incremented PC

IR[8:0]

PC | 0100 0000 0001 1001 | SEXT   Sign-extend

16

1111111111111101

16   16

ADD

16

Register file

R0
R1
R2
R3
R4
R5 0100000000010110   DR
R6
R7

Control Flow Instructions

Control Flow Instructions

- Allow a program to execute out of sequence

- Conditional branches and jumps

  - Conditional branches are used to make decisions
    - E.g., if-else statement
  - In LC-3, three condition codes are used

  - Jumps are used to implement
    - Loops
    - Function calls
  - JMP in LC-3 and j in MIPS

Condition Codes in LC-3

- Each time one GPR (R0-R7) is written, three single-bit registers are updated

- Each of these condition codes are either set (set to 1) or cleared (set to 0)

  - If the written value is negative
    - N is set, Z and P are cleared

  - If the written value is zero
    - Z is set, N and P are cleared

  - If the written value is positive
    - P is set, N and P are cleared

- SPARC and x86 are examples of ISAs that use condition codes

Conditional Branches in LC-3

- **BRz (Branch if Zero)**

0000	n	z	p	PCoffset9

4 bits   9 bits
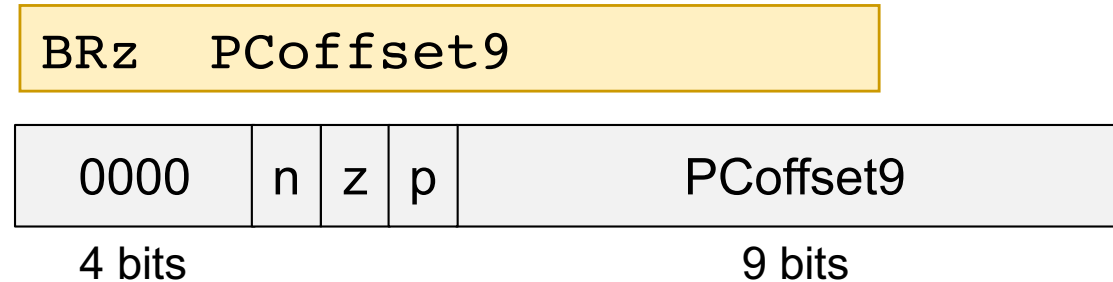
- □ n, z, p = which N, Z, and/or P is tested

- □ PCoffset9 = immediate or constant value

- □ if ((n AND N) OR (p AND P) OR (z AND Z))
  - then PC ← PC[†] + sign-extend(PCoffset9)

- □ Variations: BRn, BRz, BRp, BRzp, BRnp, BRnz, BRnzp

Conditional Branches in LC-3

- BRz

  BRz   0x0D9

What if n = z = p = 1?
(i.e., BRnzp)

And what if n = z = p = 0?

Program
Counter
0100 0001 0000 0001

PC 0100 0000 0010 1000

Instruction
register
BR n z p PCoffset9

IR 0000 0 1 0 011011001

9

SEXT

16   16

Condition
registers

N   Z   P
0   1   0

0000000011011001

ADD

16

PCMUX

Yes!
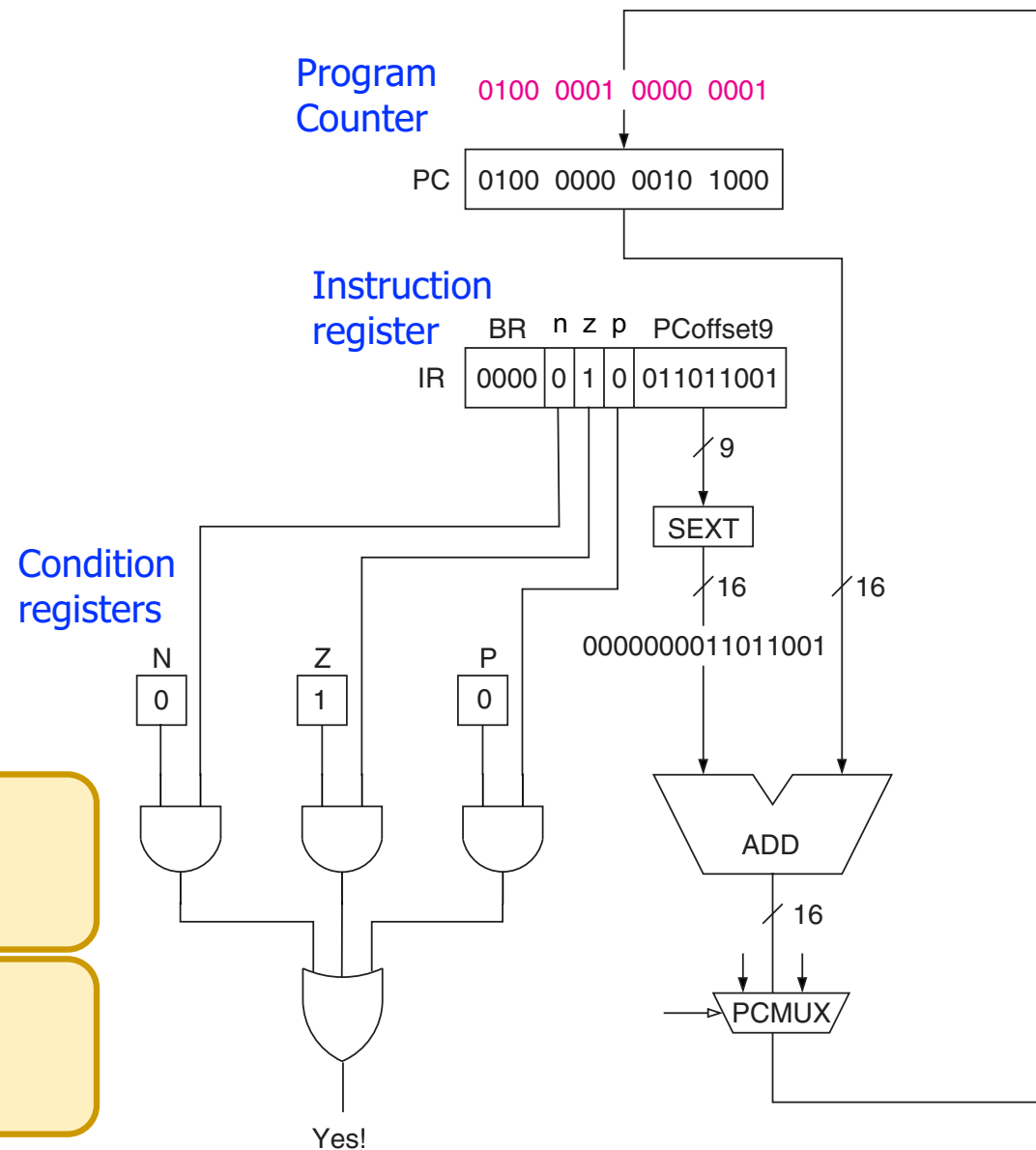
Branch If Equal in MIPS and LC-3

MIPS assembly

```
beq  $s0, $s1, offset
```

LC-3 assembly

```
NOT  R2, R1
ADD  R3, R2, #1
ADD  R4, R3, R0
BRz  offset
```

Subtract (R0 - R1)

- This is an example of tradeoff in the instruction set

  - The same functionality requires more instructions in LC-3

  - But, the control logic requires more complexity in MIPS

Lecture Summary

- **The von Neumann model**
  - LC-3: An example of von Neumann machine

- **Instruction Set Architectures: LC-3 and MIPS**
  - Operate instructions
  - Data movement instructions
  - Control instructions

- **Instruction formats**

- **Addressing modes**