

LC-3 Datapath



Chapter 4

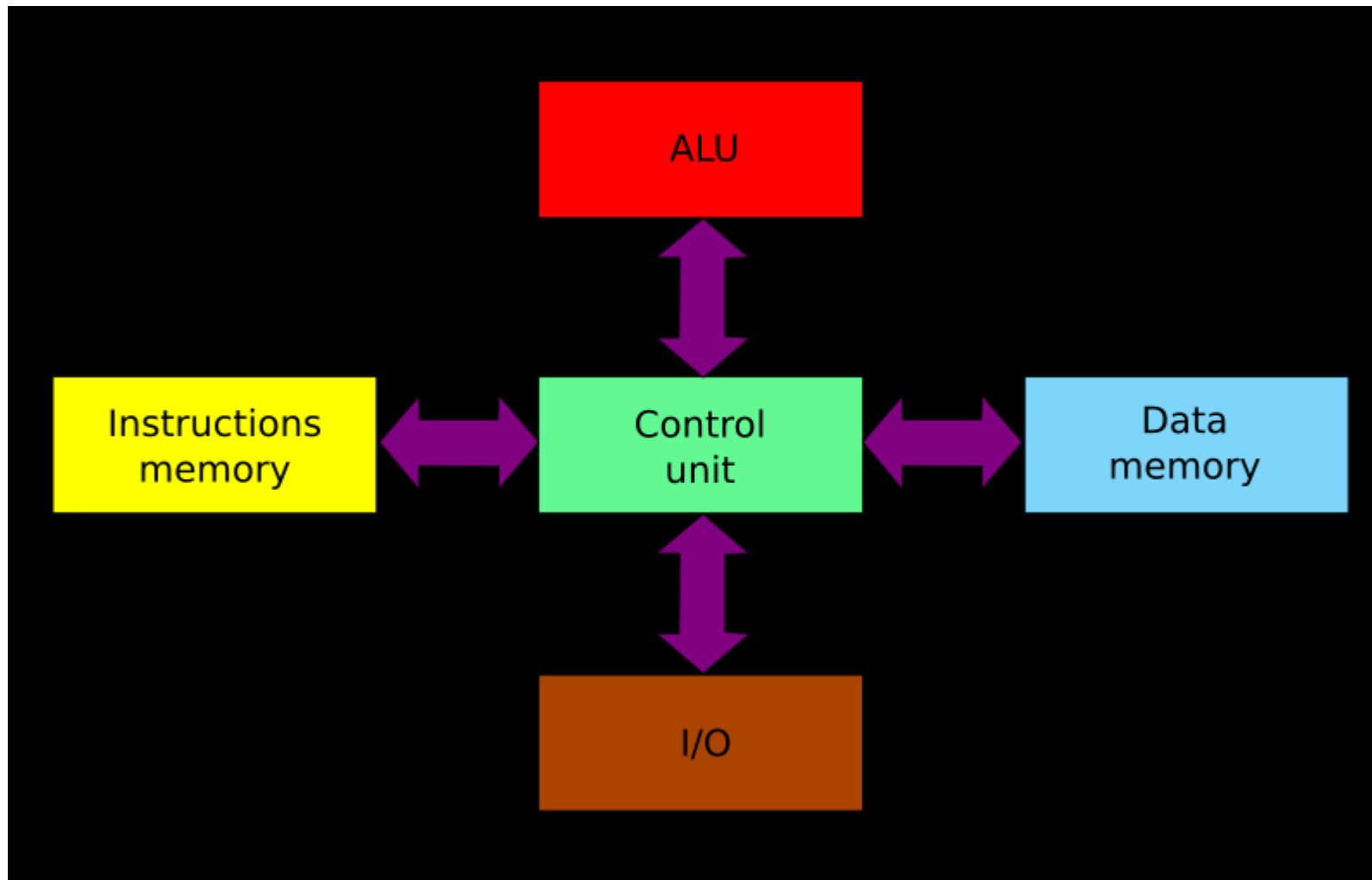


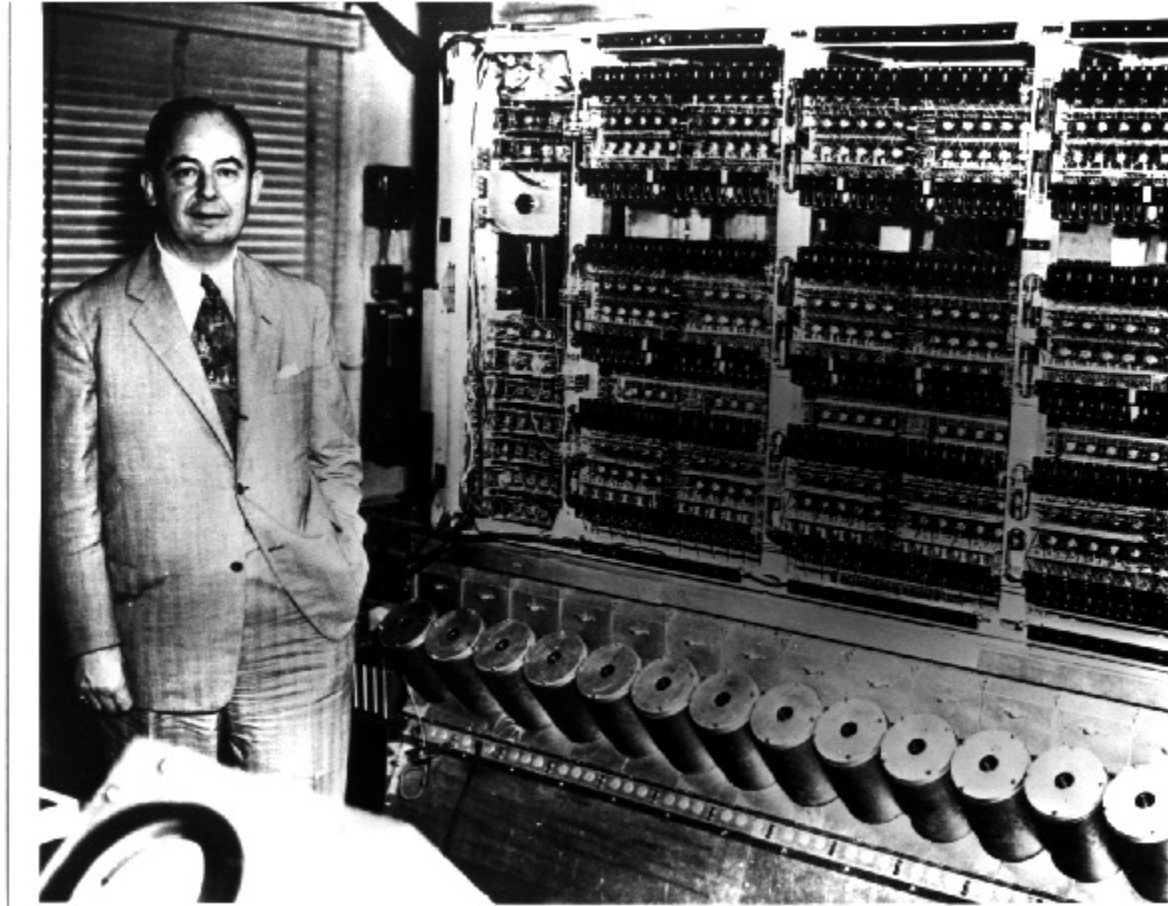
- Basic Components
 - Memory, Processing Unit, Input & Output, Control Unit
- Architectures
 - Harvard
 - Von Neumann
- LC-3: A von Neumann Architecture
- Tri-State Buffers
- Instruction Cycle
 - Instructions
 - Fetch, Decode, Evaluate Address, Fetch Operands, Execute, Store Result



Harvard Mark I

Harvard Model



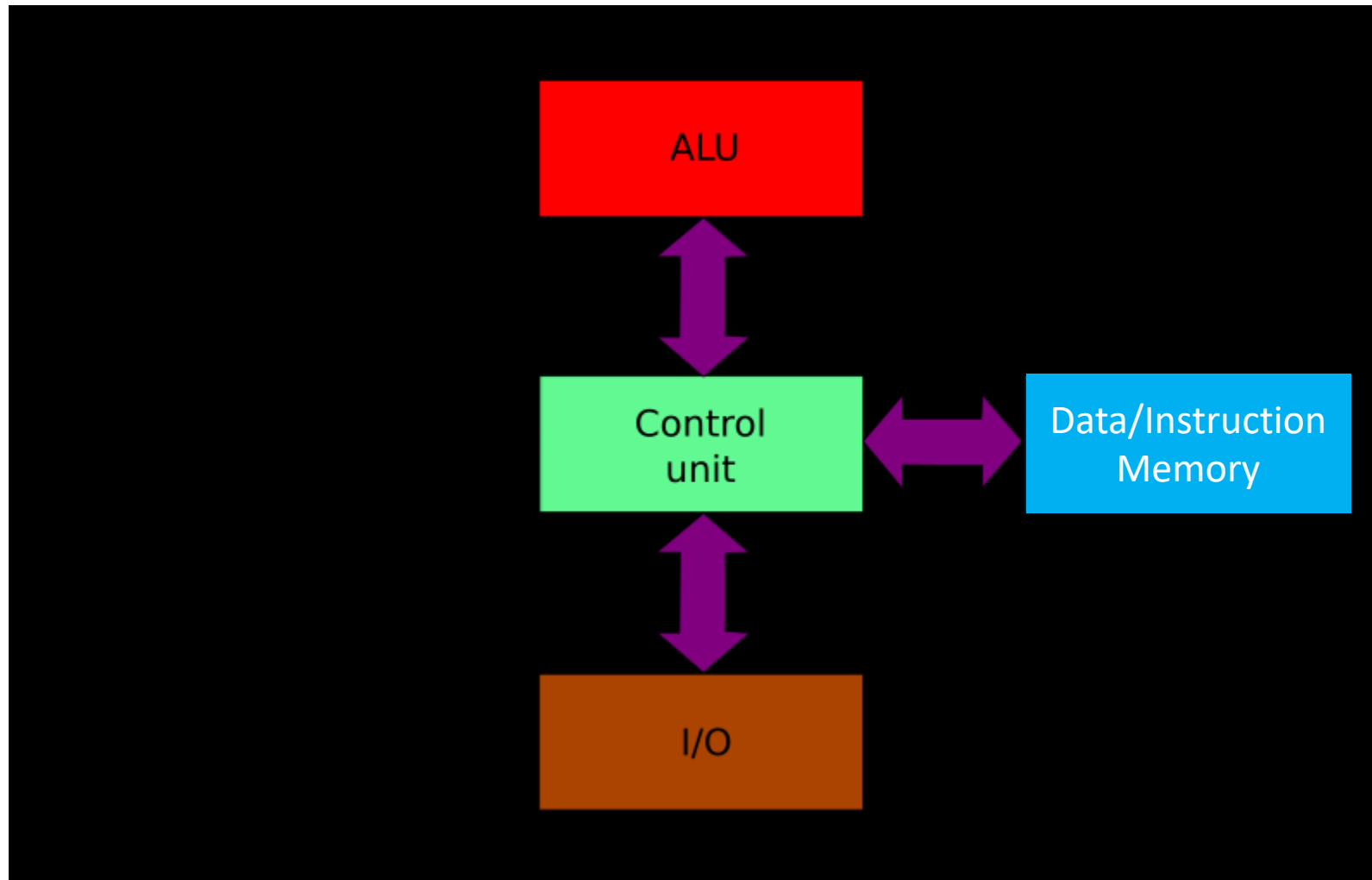


John von Neumann & EDVAC

Electronic Discrete Variable Automatic Computer

First Draft of a Report on the EDVAC
by John von Neumann,
Contract No. W-670-ORD-4926,
Between the United States Army Ordnance Department
and the University of Pennsylvania Moore School of
Electrical Engineering
University of Pennsylvania
June 30, 1945

Von Neumann Model



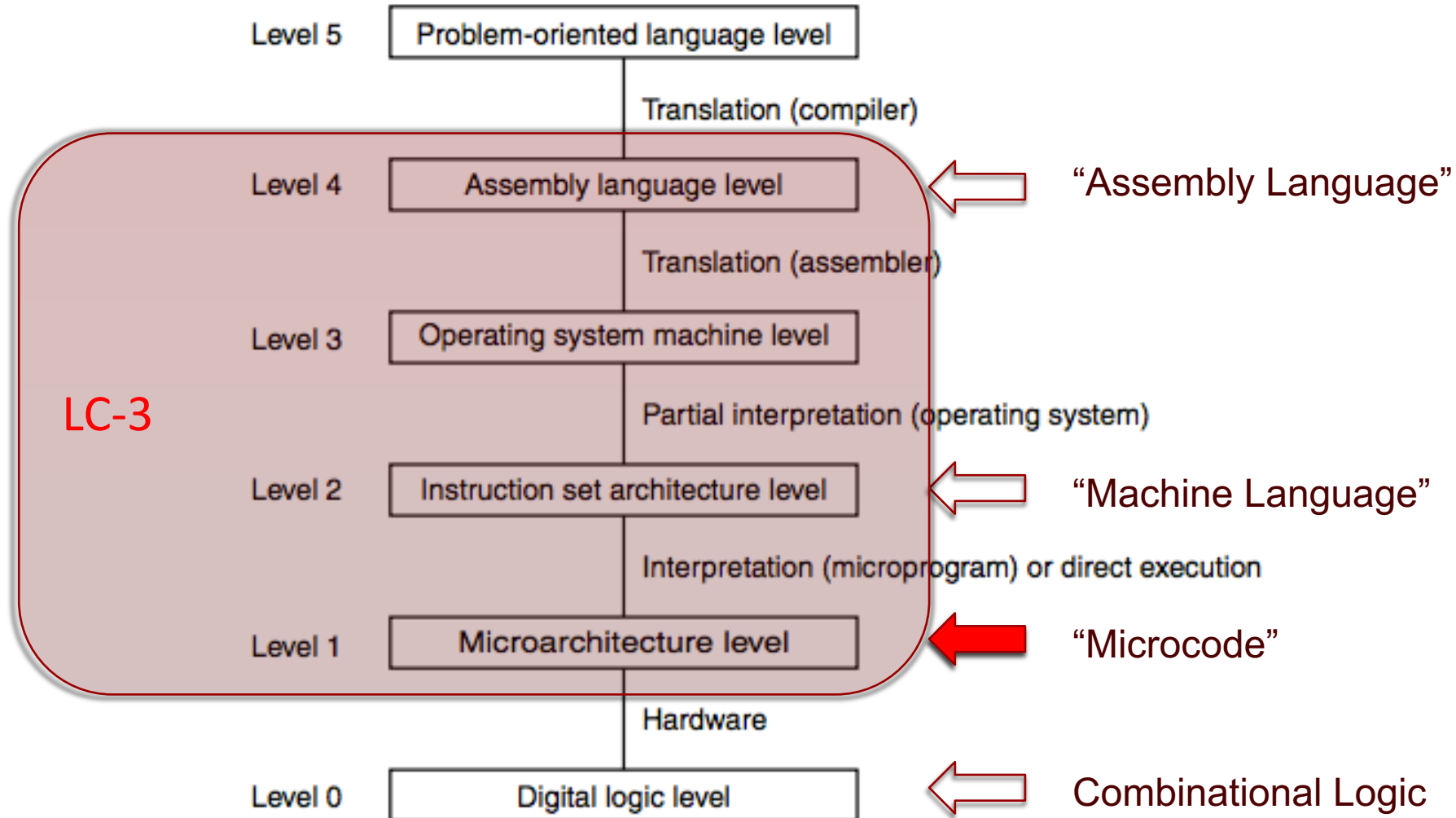
- The central idea in the von Neumann model of computer processing is that the ***program*** and ***data*** are both ***stored*** as sequences of bits in the ***computer's memory***, and the program is executed, one instruction at a time, under the direction of the control unit.

Machine Instructions as Data?

- The von Neumann model leads us to treat machine instructions as just another data representation!
 - *This is a big deal*
- Think of
 - unsigned integer
 - twos-complement integer
 - ASCII
 - IEEE-754 floating point
 - ...
 - **machine instructions**
- They are all data representations
 - *(Bits are just bits!)*

- What can be stored in memory?
 - Data
 - Instructions
- How many memories do we need?
 - Just one, according to von Neumann

Where are we in this course?

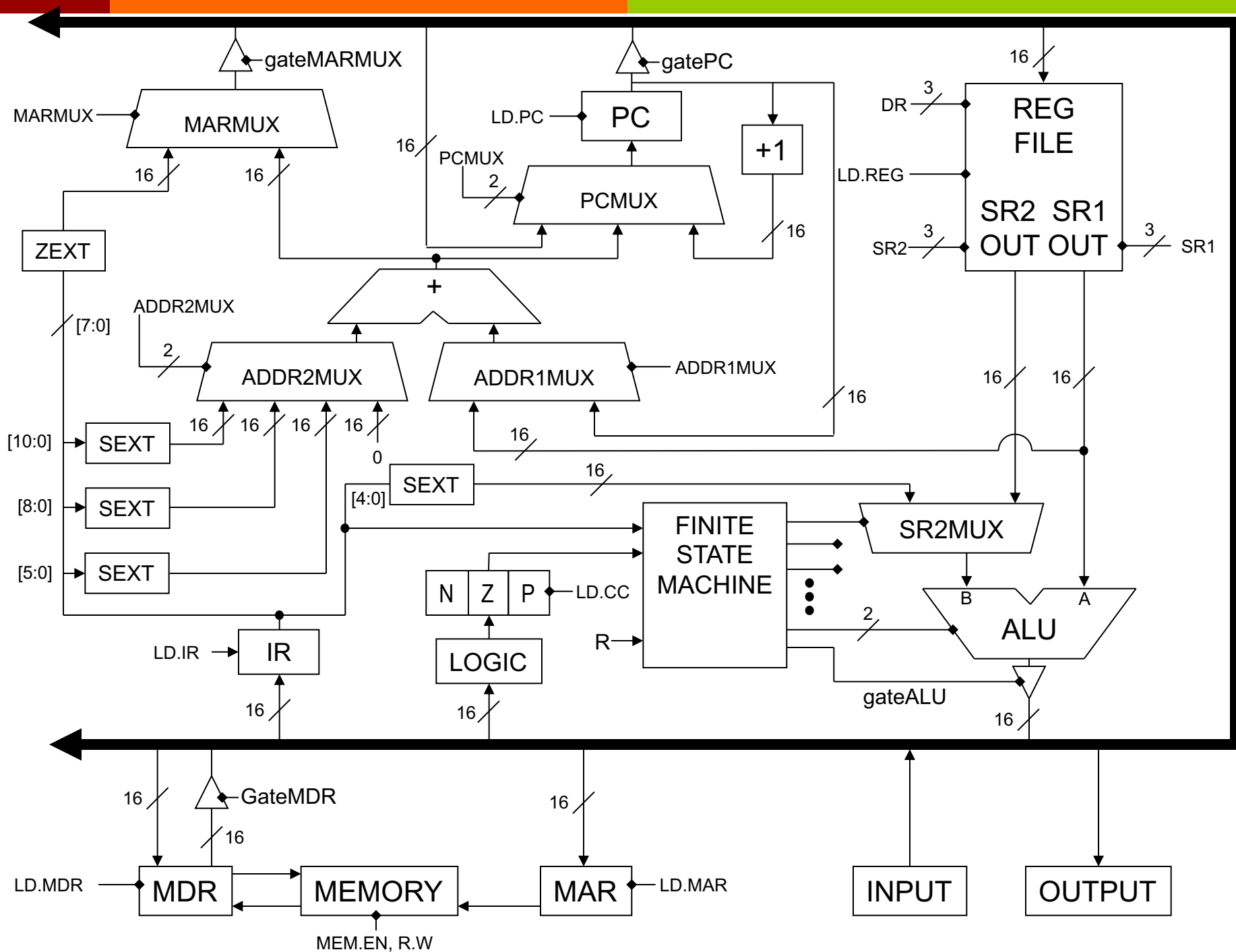


Introducing the LC-3 CPU!

- Address space
 - $2^{16} = 65536$
- Addressability
 - 16 bits
- Architecture type
 - Von Neumann
- General purpose registers
 - 8
- Instruction size
 - 16 bits

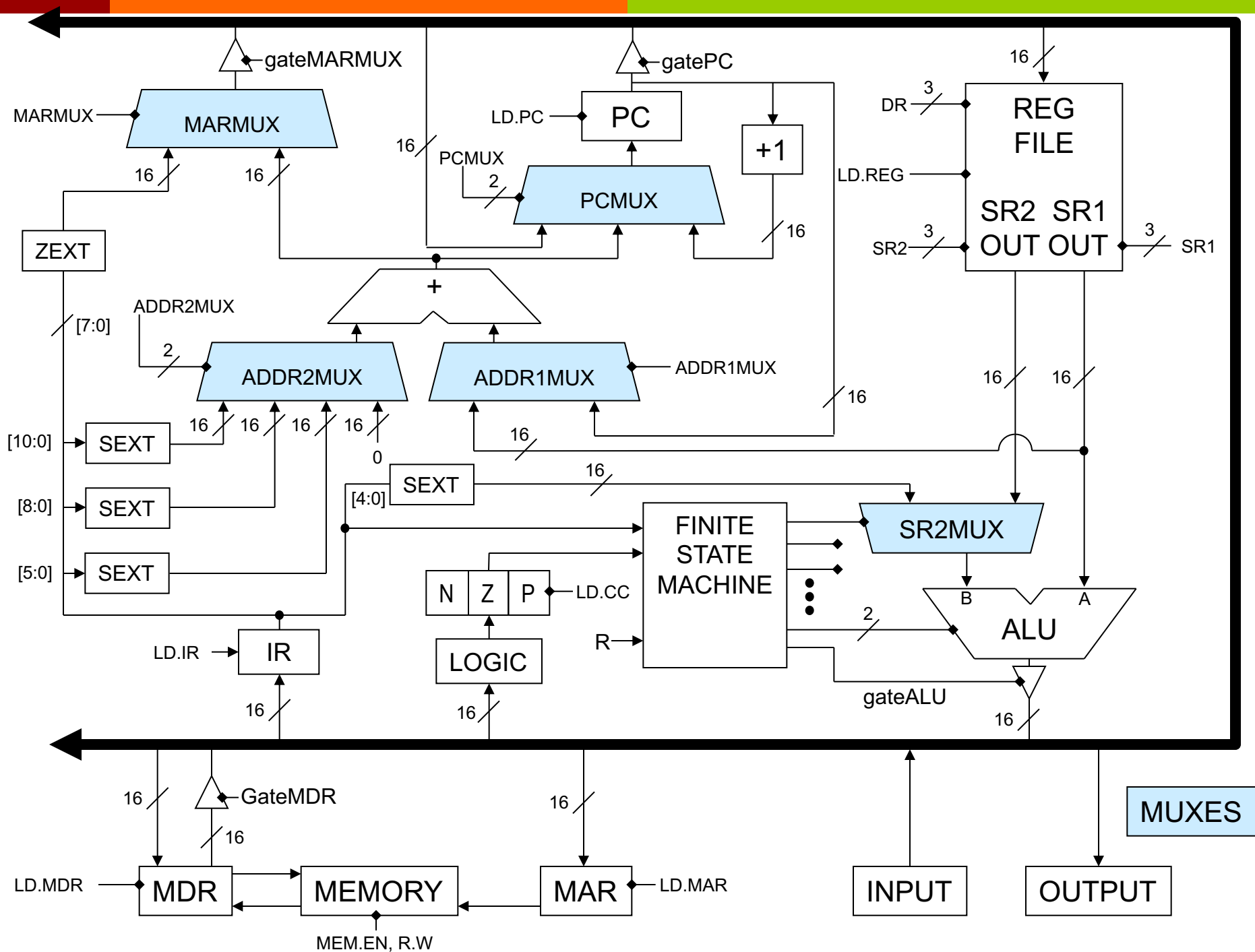
The LC-3 Datapath

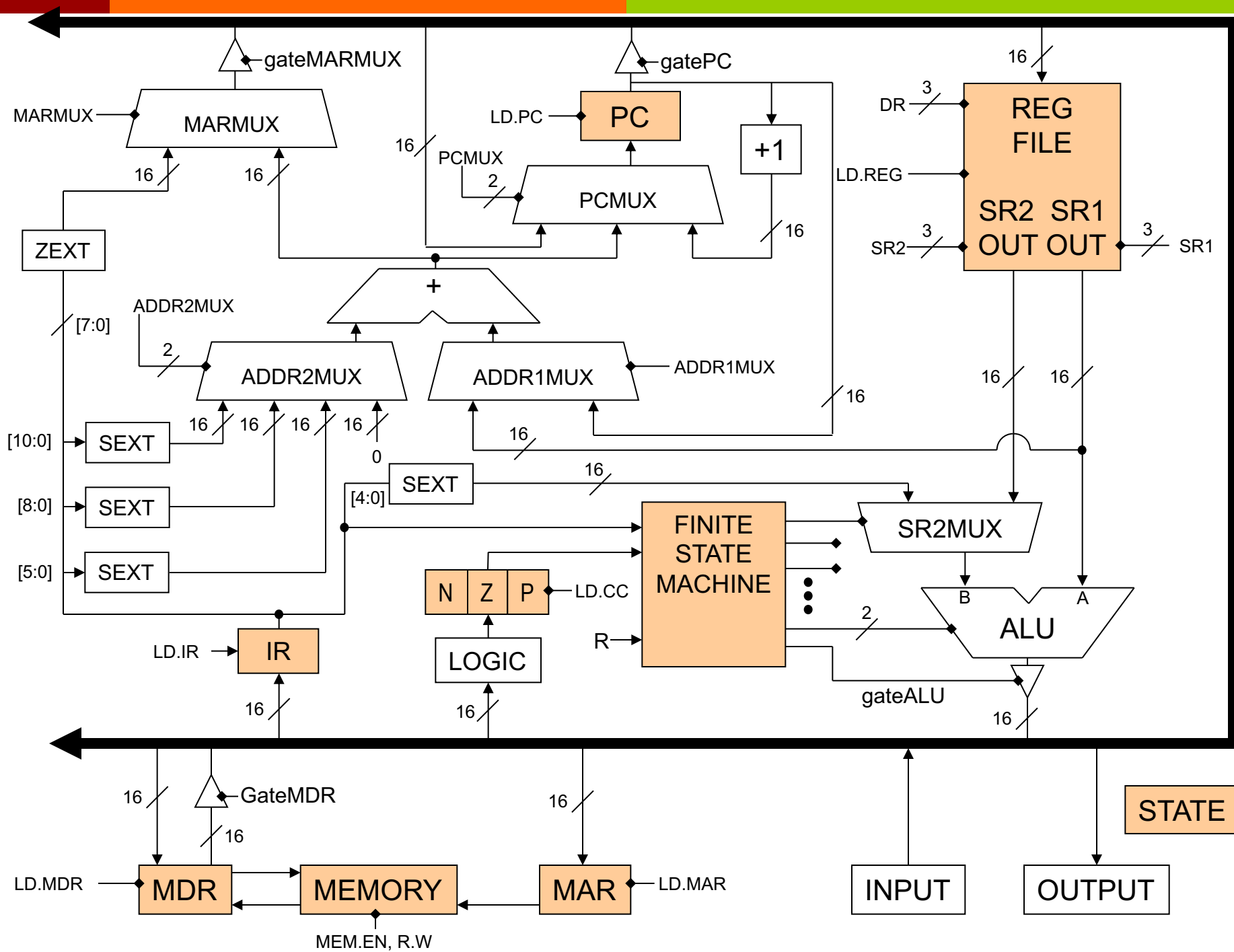
- Let's build a CPU
 - Using digital logic – that we already know!

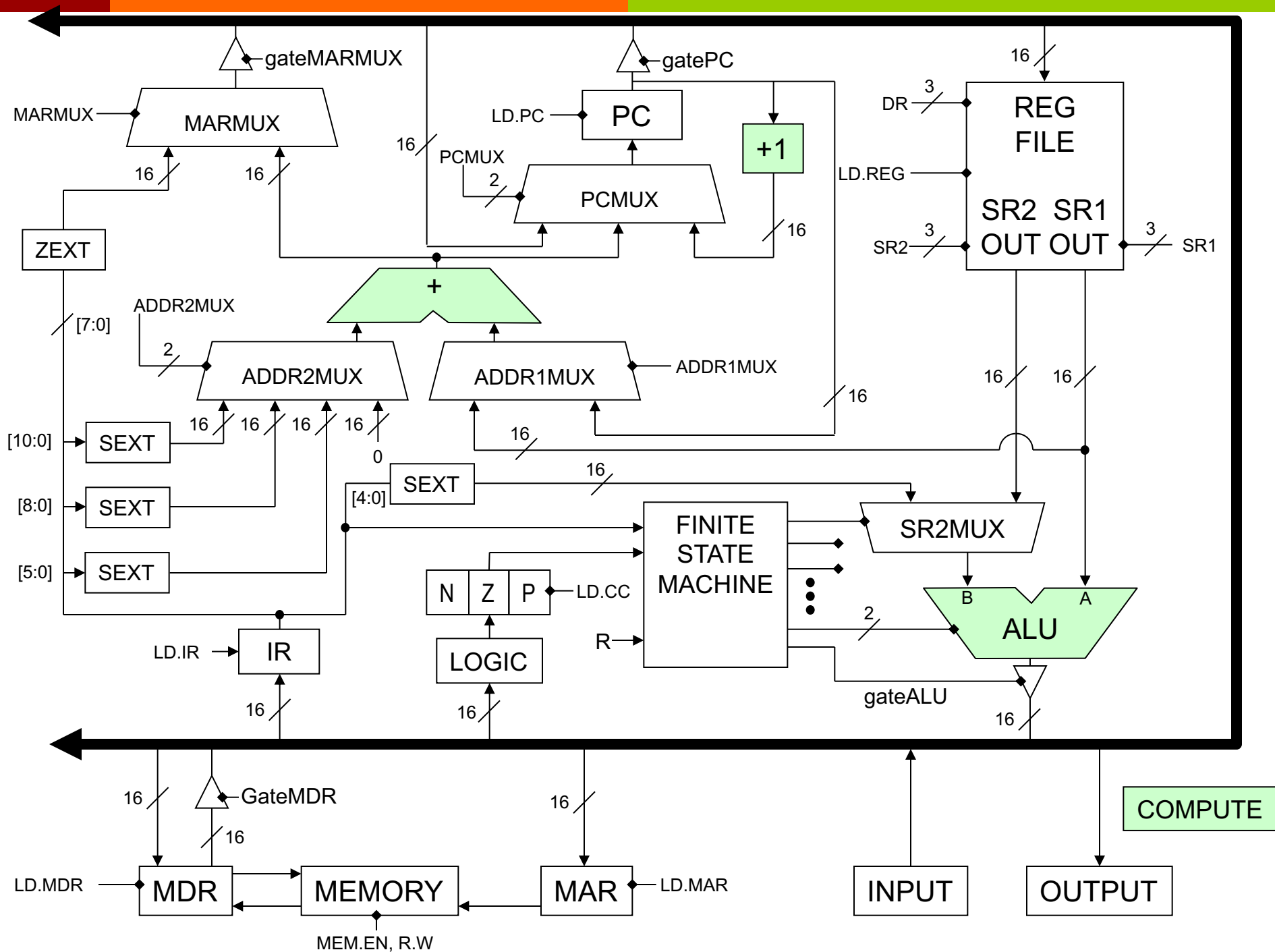


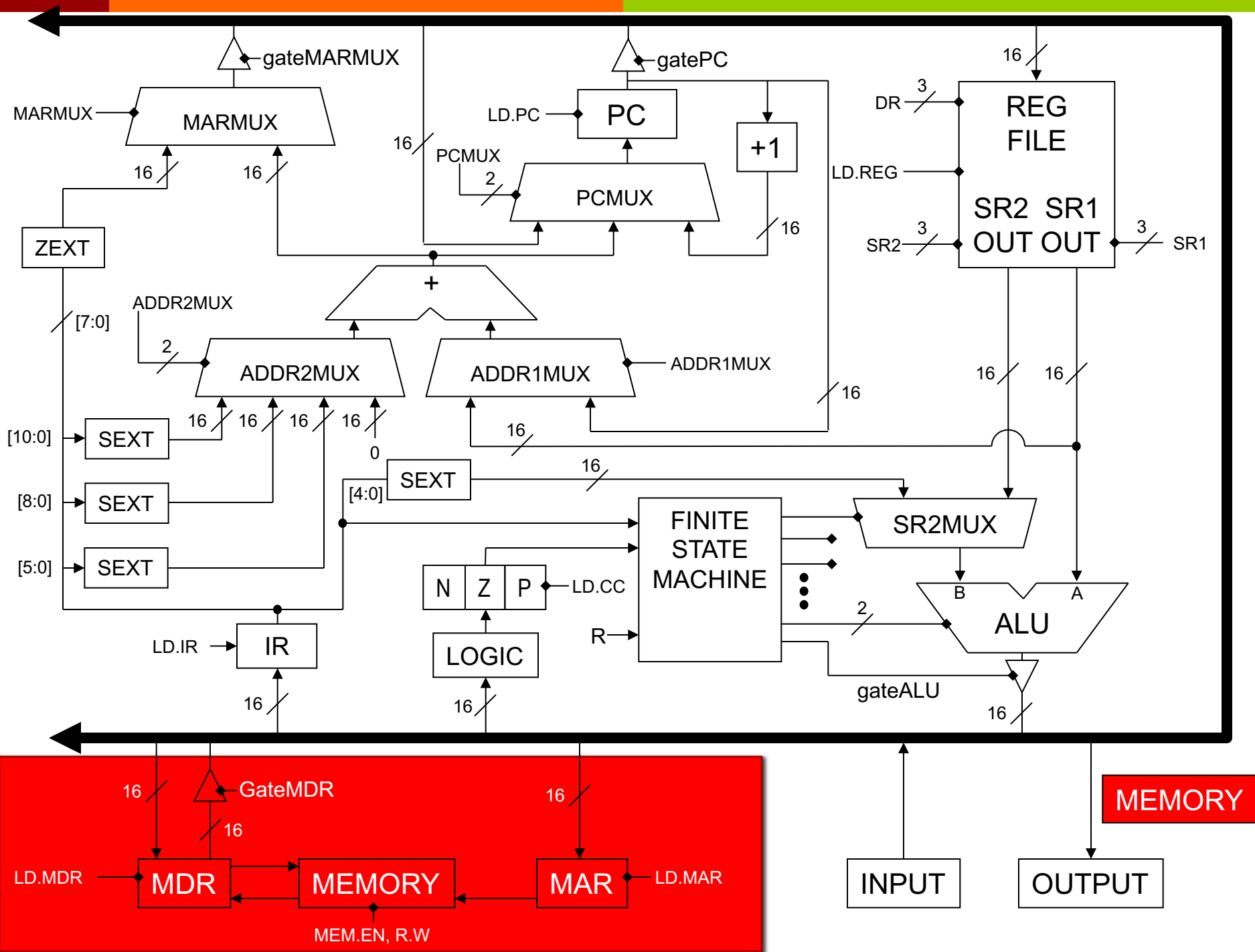
LC-3
Datapath

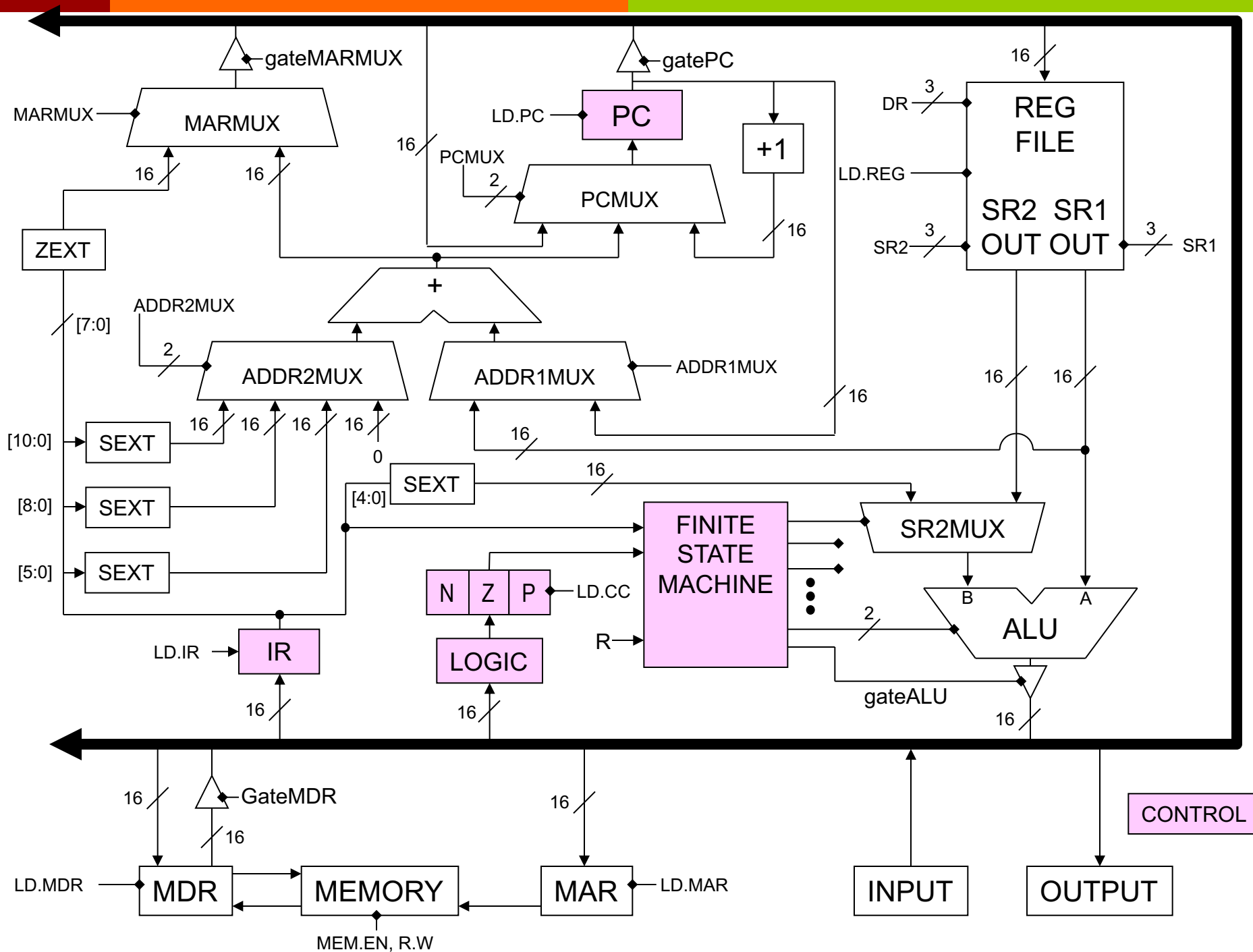
- That looks scary!
 - But you already know what nearly all of these components are.
- Let's break it down, piece by piece.











One more Digital Logic Component ...

- There is one more combinational logic circuit we have not yet introduced:

- **Tri-State Buffer**

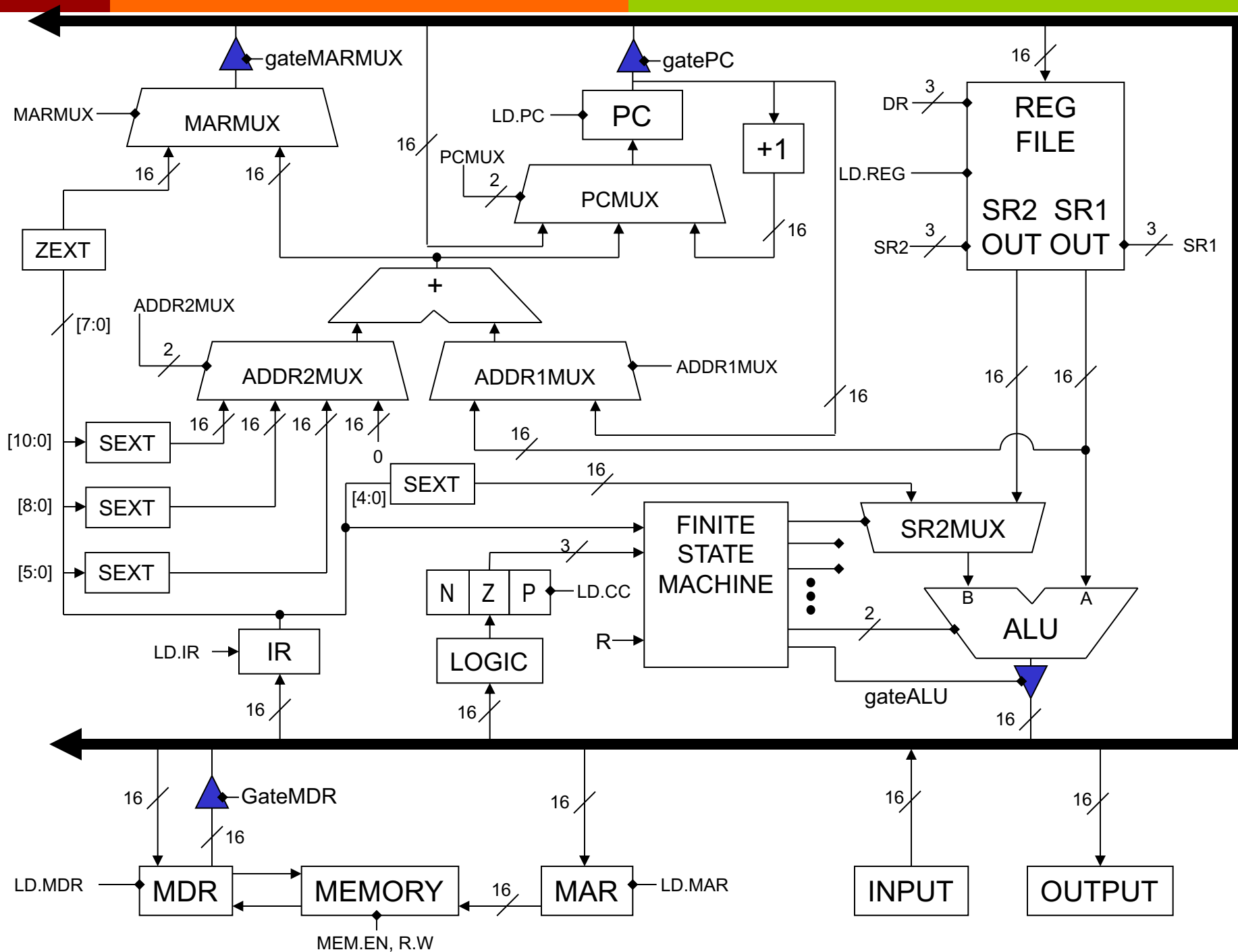
- The tri-state buffers are used to help with the **bus**

- The bus is 16 shared wires in the datapath

- (No transistors or logic, just wires to connect things)

- This is what is used to disconnect inputs so the bus can be shared among multiple components

- Patt names all his tri-state buffers with the prefix “gate”



the **Bus**
16 shared wires

Important Concept Approaching

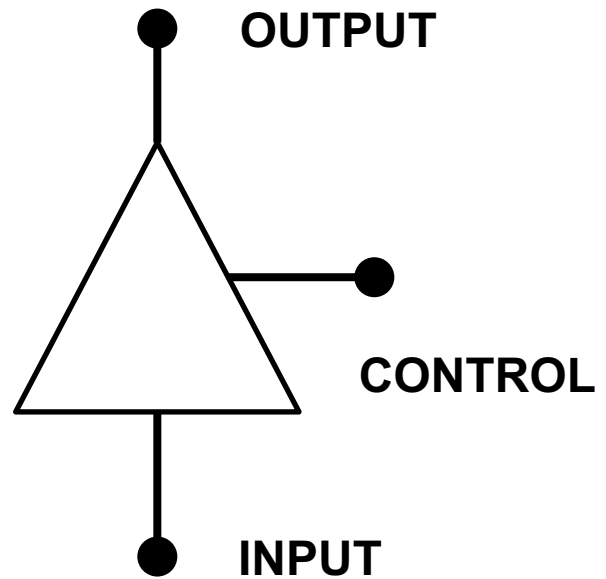
- Remember – our logic gates literally connect their output to $+V_{CC}$ for 1 and Ground for 0
- When sharing a wire with another circuit – this arrangement is called a **bus**
- What happens when you connect a 1 ($+V_{CC}$) to a 0 (ground)?
- Hint: It involves converting electrical energy into heat energy. A lot of it.

Melted Wires, Smoke, Flames...



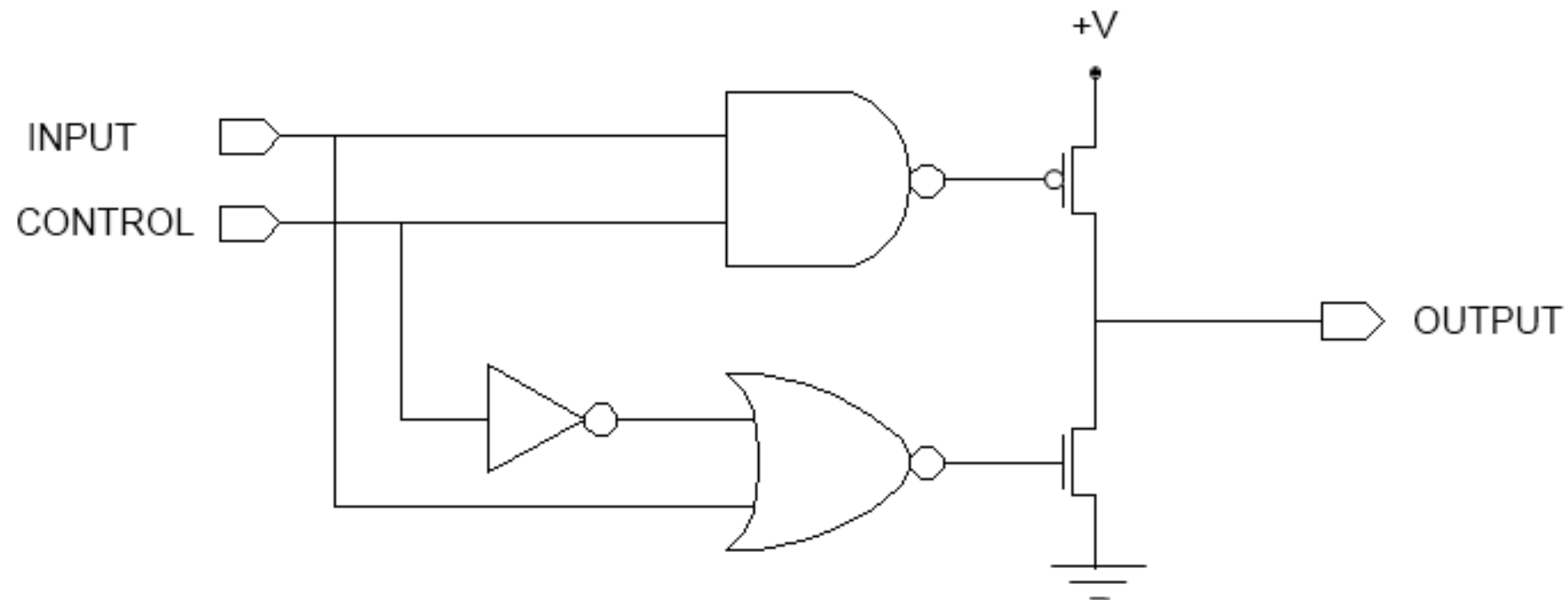
**Can Prevent Datapath Fires!
Only assert one Gate signal
per clock cycle!**

Tri State Buffer




CONTROL	INPUT	OUTPUT
0	0	Z
0	1	Z
1	0	0
1	1	1

Tri State Buffer





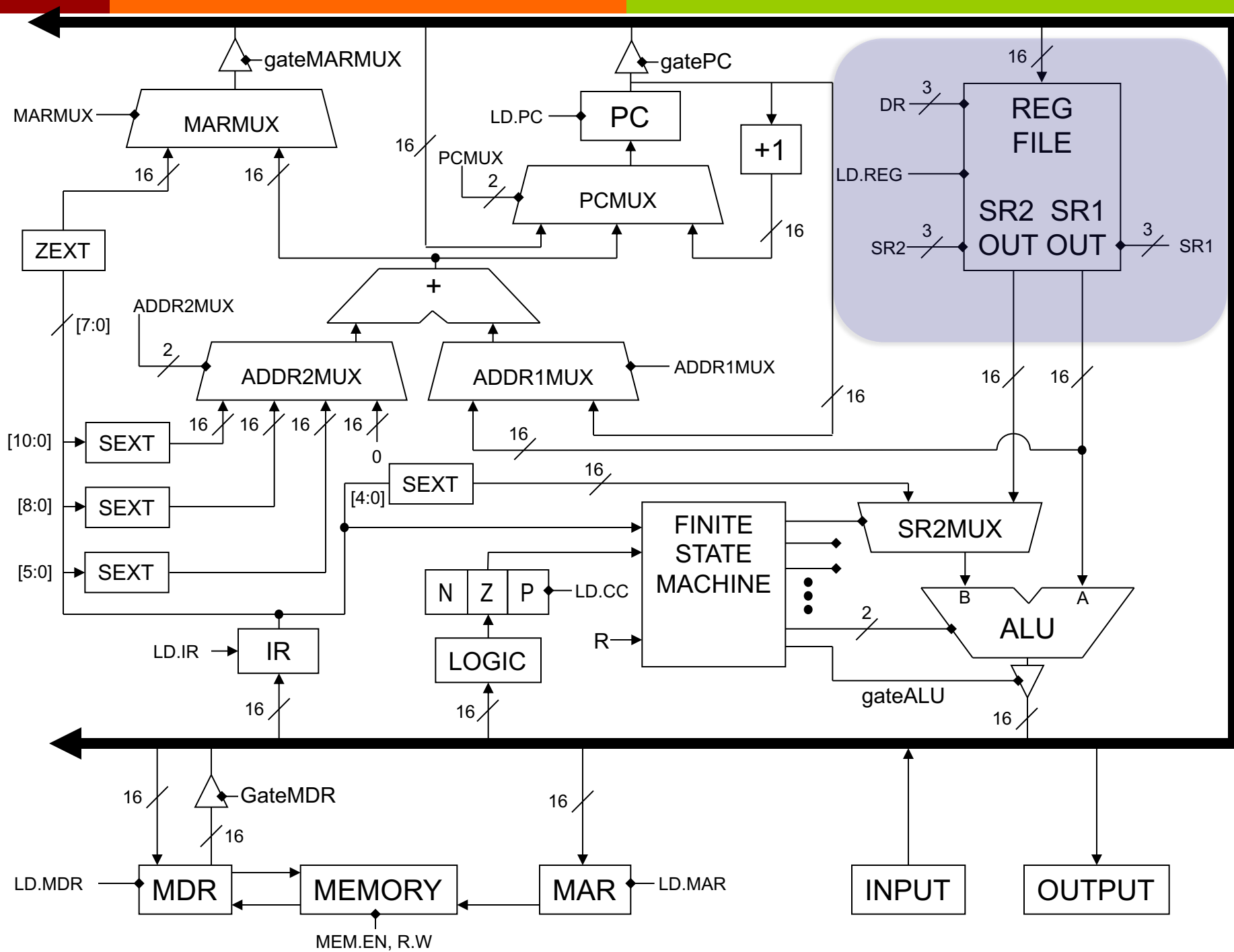
Tri-state buffers are used to

- A. Prevent data path fires
- B. Avoid short circuits
- C. Disconnect a circuit from a bus so that another circuit can assert a value on the same bus without interference
- D. Present a value that is neither 1 or 0 on an output
- E. All of the above 

Questions?

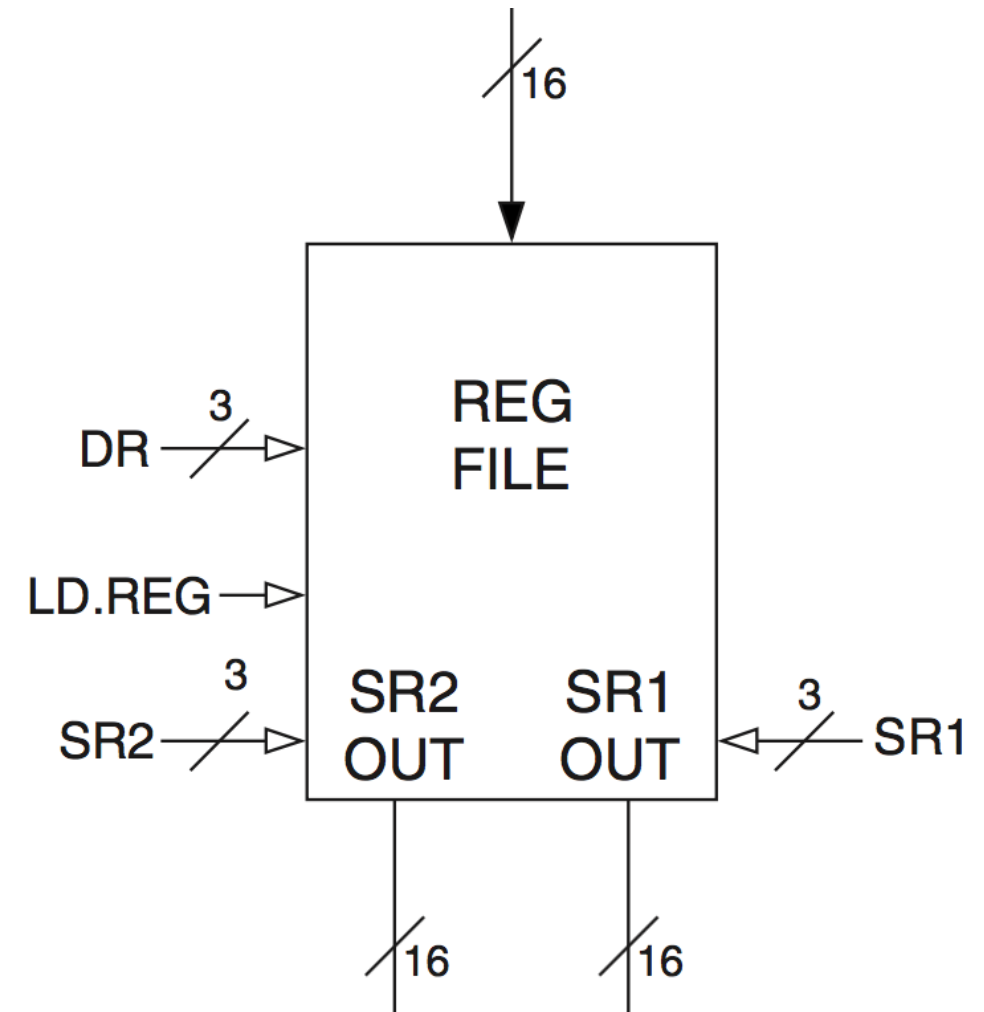
Register File





Register File Circuit

- A small, fast “Memory”
 - Address Space: 8 registers
 - 16-bit addressability per reg
- Two outputs
 - Dual-ported memory
 - Can read two regs at same time
 - SR1 (3 bit address)
 - SR2 (3 bit address)
- One input
 - DR (3 bit address)
 - LD.REG (write enable)
- Can read two registers and write one register in a single clock cycle(!)



What is a Machine Instruction?

- It's just another data representation!
- It tells the processor what to do next
- Is it a datatype?
- Yes. There is definitely hardware to interpret it.

Categories of Machine Instructions

Operate (ALU)

- ADD
- AND
- NOT

Data Movement (Memory)

Load

- LD
- LDR
- LDI
- LEA

Store

- ST
- STR
- STI

Control

- BR
- JMP
- JSR
- JSRR
- RET
- RTI
- TRAP

We'll explain what each of these instructions does

- Just briefly enough for now, so you know how to make them happen in the datapath
- With control signals, that come from the state machine

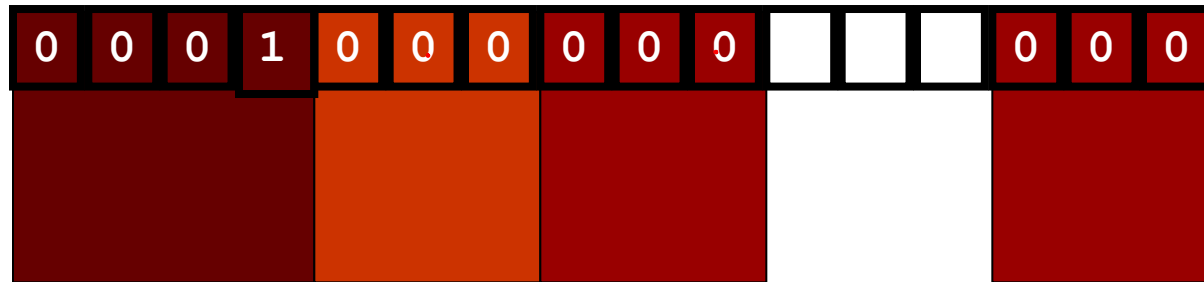
Some LC-3 Instructions

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	
BR	0000	n	z	p	PCOffset9	
JMP	1100	000	BaseR	000000		
JSR	0100	1	PCOffset11			
JSRR	0100	0	00	BaseR	000000	
LD ⁺	0010	DR	PCOffset9			
LDI ⁺	1010	DR	PCOffset9			

⁺ Indicates instructions that modify condition codes

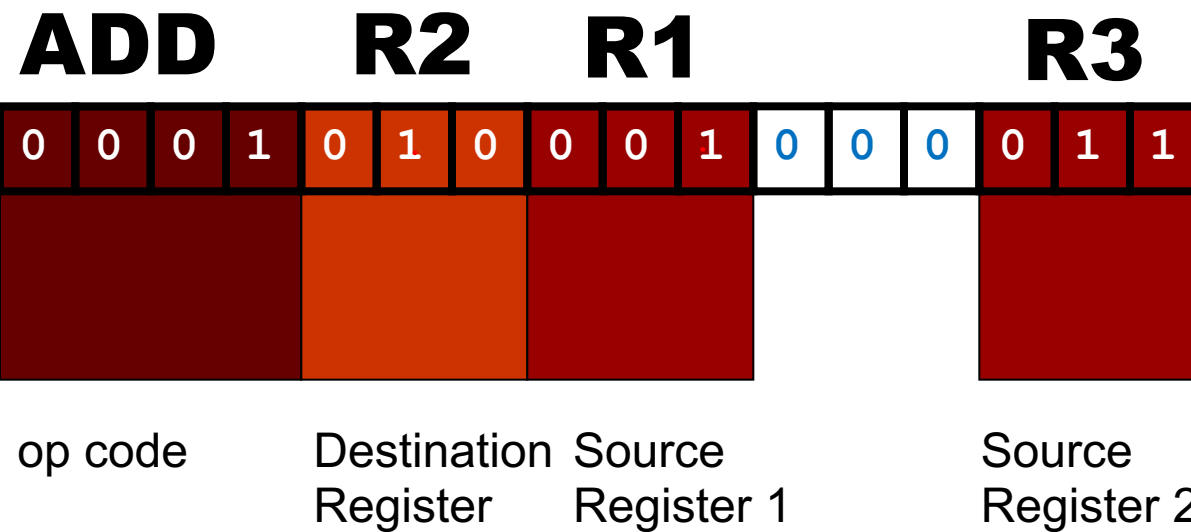
LC-3 Instructions

ADD



Instruction Example

ADD R2,R1,R3



0x1443
012103

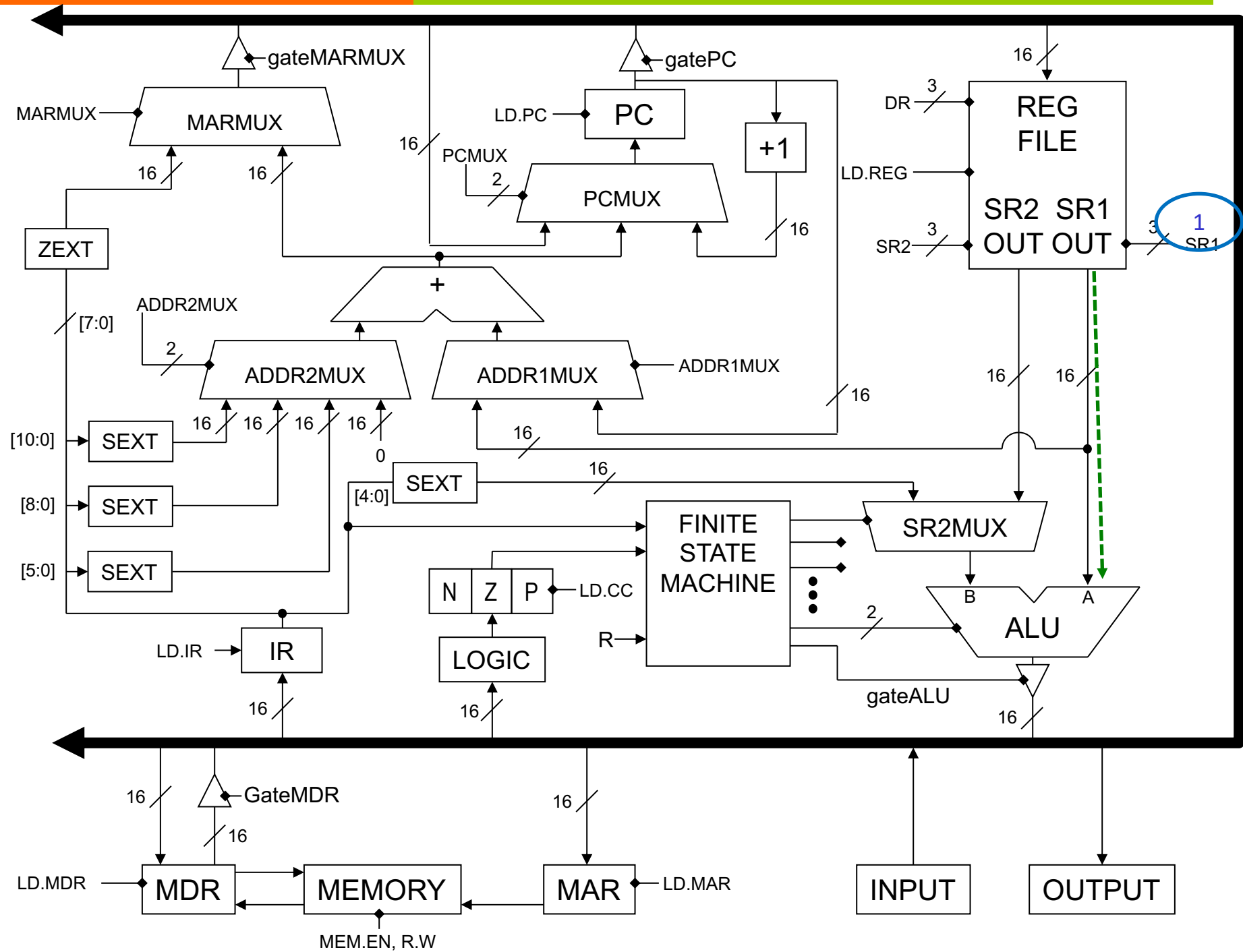
This instruction means:
 $R2 = R1 + R3$

How do we make the LC-3 do an “ADD”?

- The LC-3 is controlled by a large finite state machine.
- So it has output signals (**control signals**)
 - Like UP and DOWN signals in the garage door
- The FSM turns on specific control signals in the LC-3
 - This activates certain paths in the datapath.
 - These control signals can make the datapath do things, such as add two numbers (R1 and R3), and store the result in R2.
- Example:
 - ADD R2, R1, R3
 - This means, $R2 = R1 + R3$
 - *Assume the data is already in R1 and R3 in the register file*

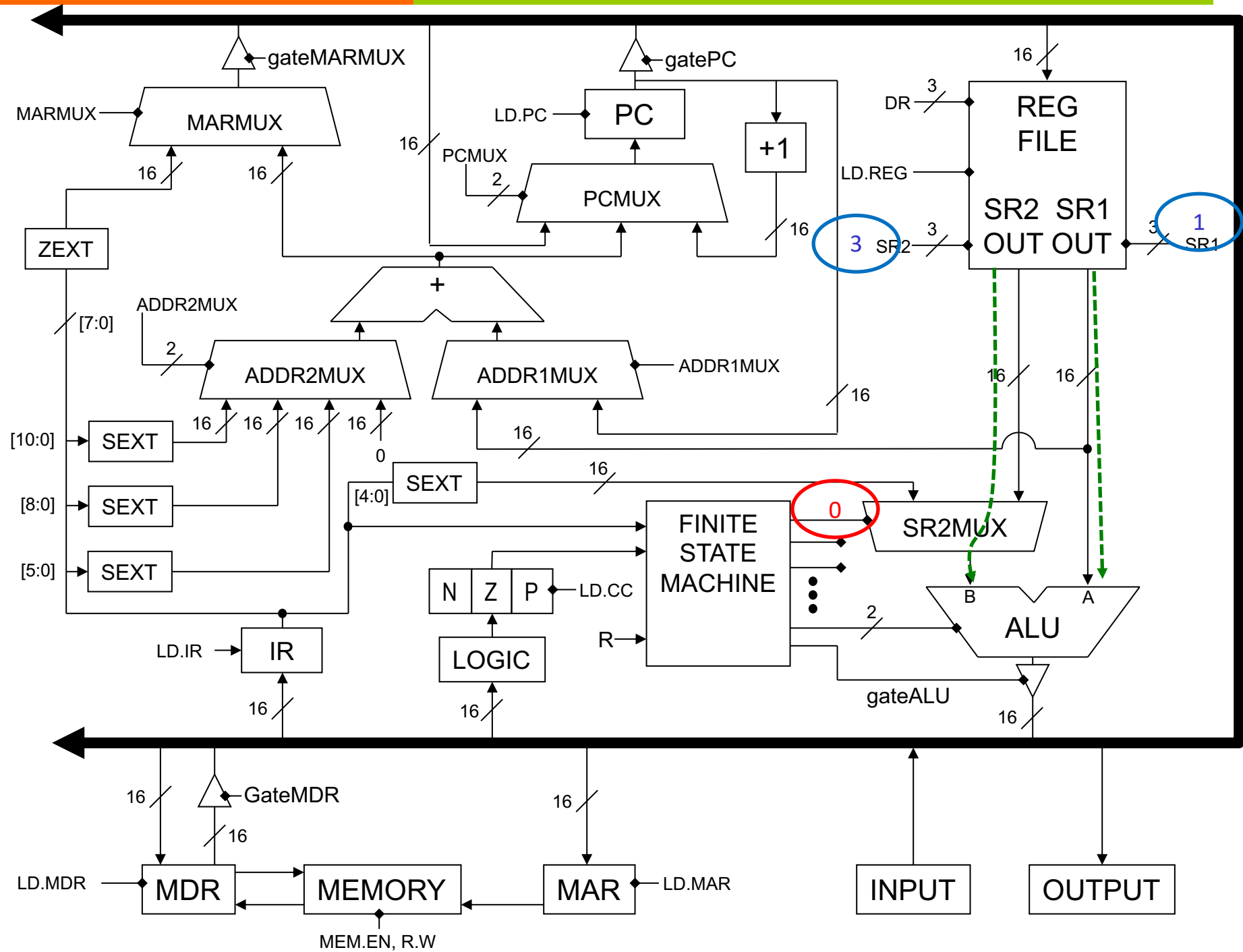
ADD R2, R1, R3 (Execute)

- Copy R1 to ALU Input A



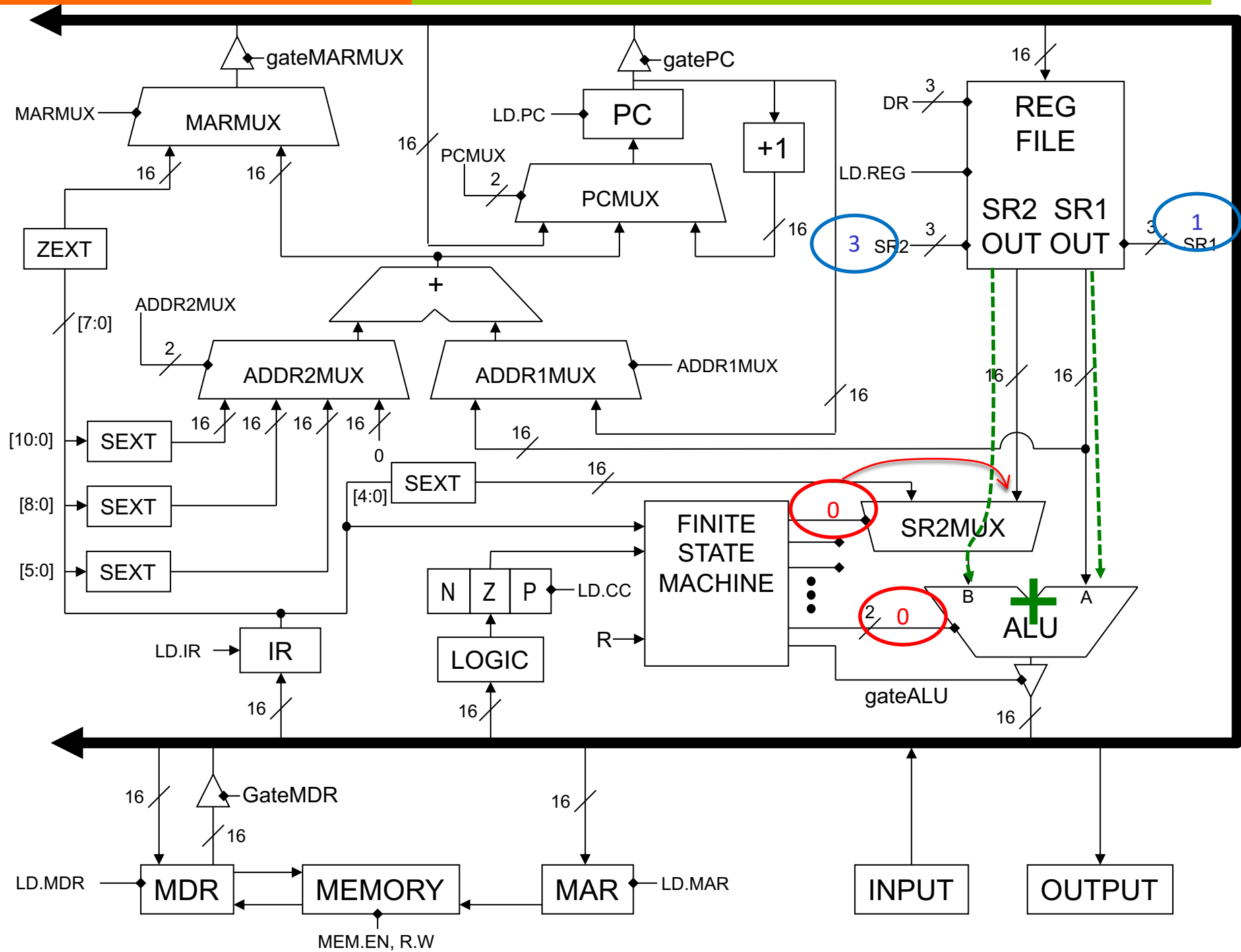
ADD R2, R1, R3 (Execute)

- Copy R1 to ALU Input A
- Copy R3 to ALU input B



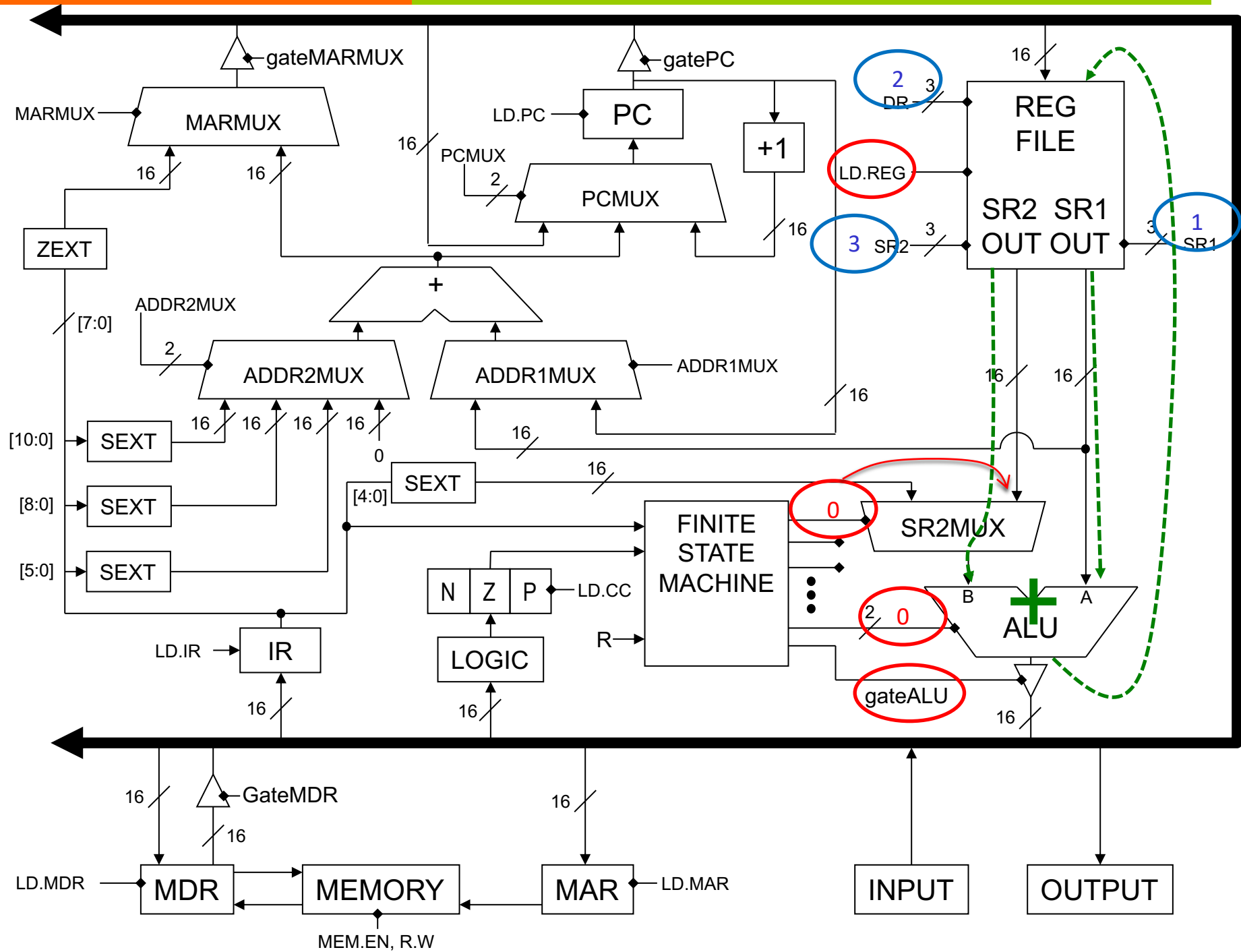
ADD R2, R1, R3 (Execute)

- Copy R1 to ALU Input A
- Copy R3 to ALU input B
- **Set ALU to ADD**



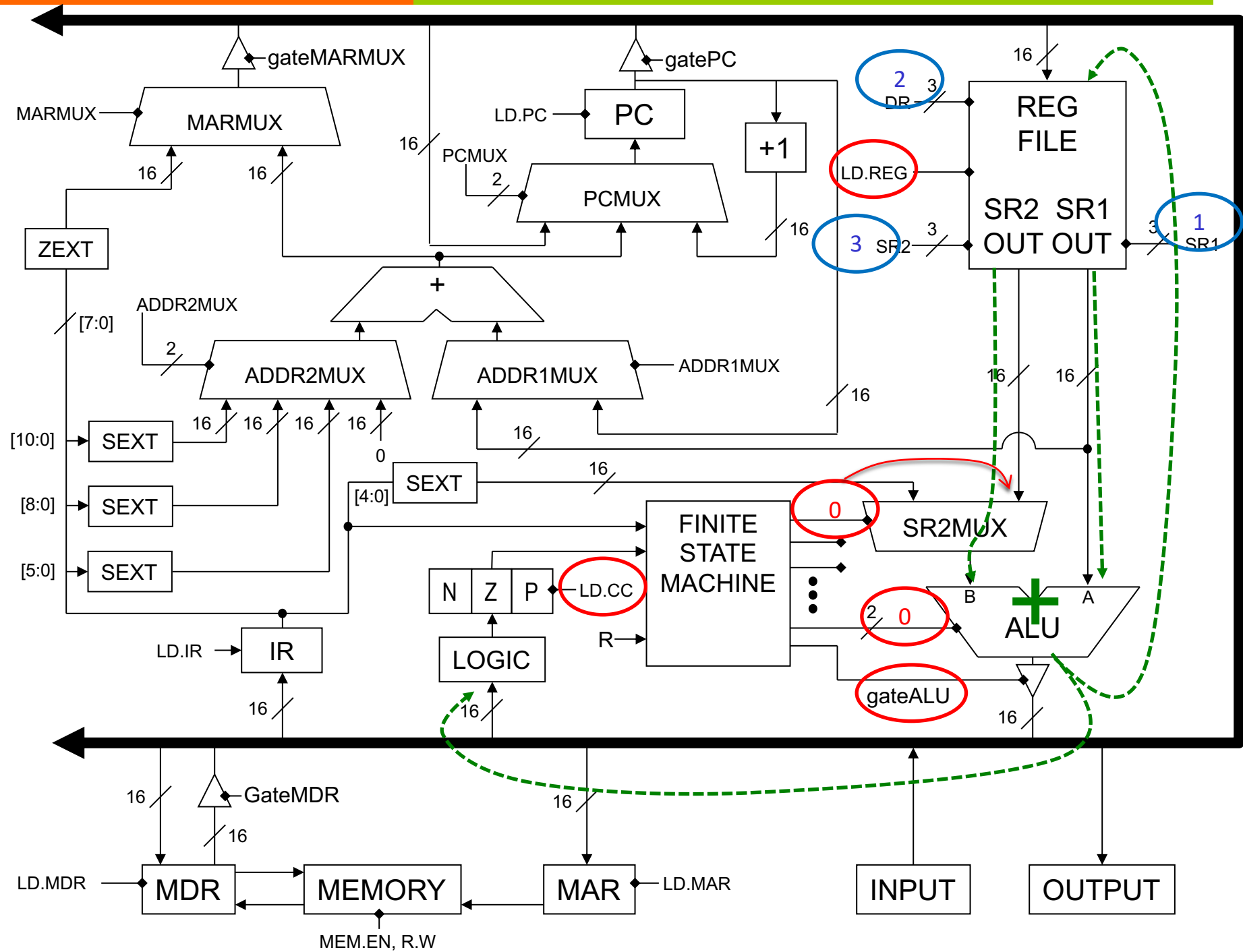
ADD R2, R1, R3 (Execute)

- Copy R1 to ALU Input A
- Copy R3 to ALU input B
- Set ALU to ADD
- Copy ALU output to R2



ADD R2, R1, R3 (Execute)

- Copy R1 to ALU Input A
- Copy R3 to ALU input B
- Set ALU to ADD
- Copy ALU output to R2
- And copy ALU output to CC



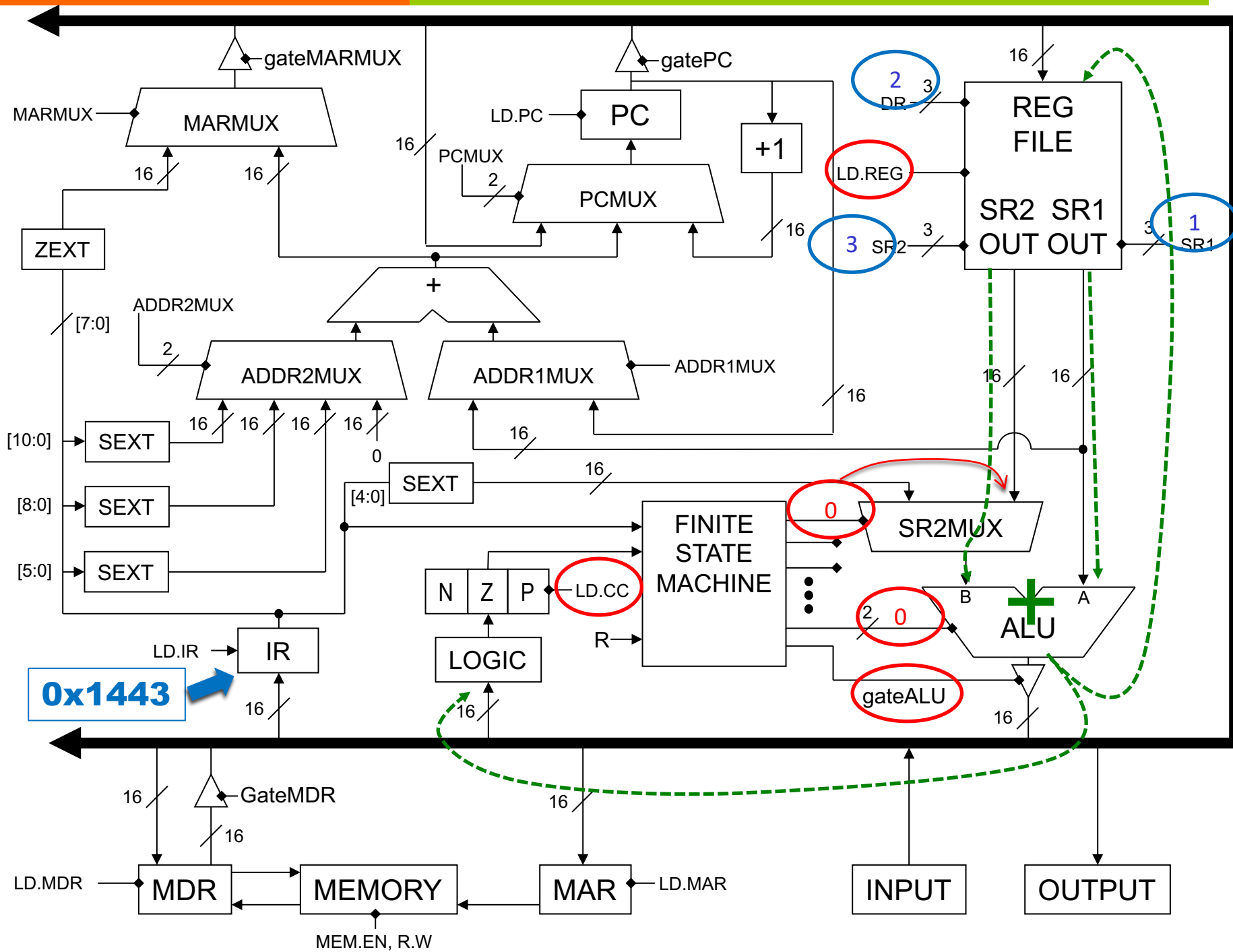
ADD R2, R1, R3 (Execute)

Control Signals (from FSM):

- LD.REG=1
- gateALU=1
- SR2MUX=1
- ALUK=00
- LD.CC =1

From Instruction (IR):

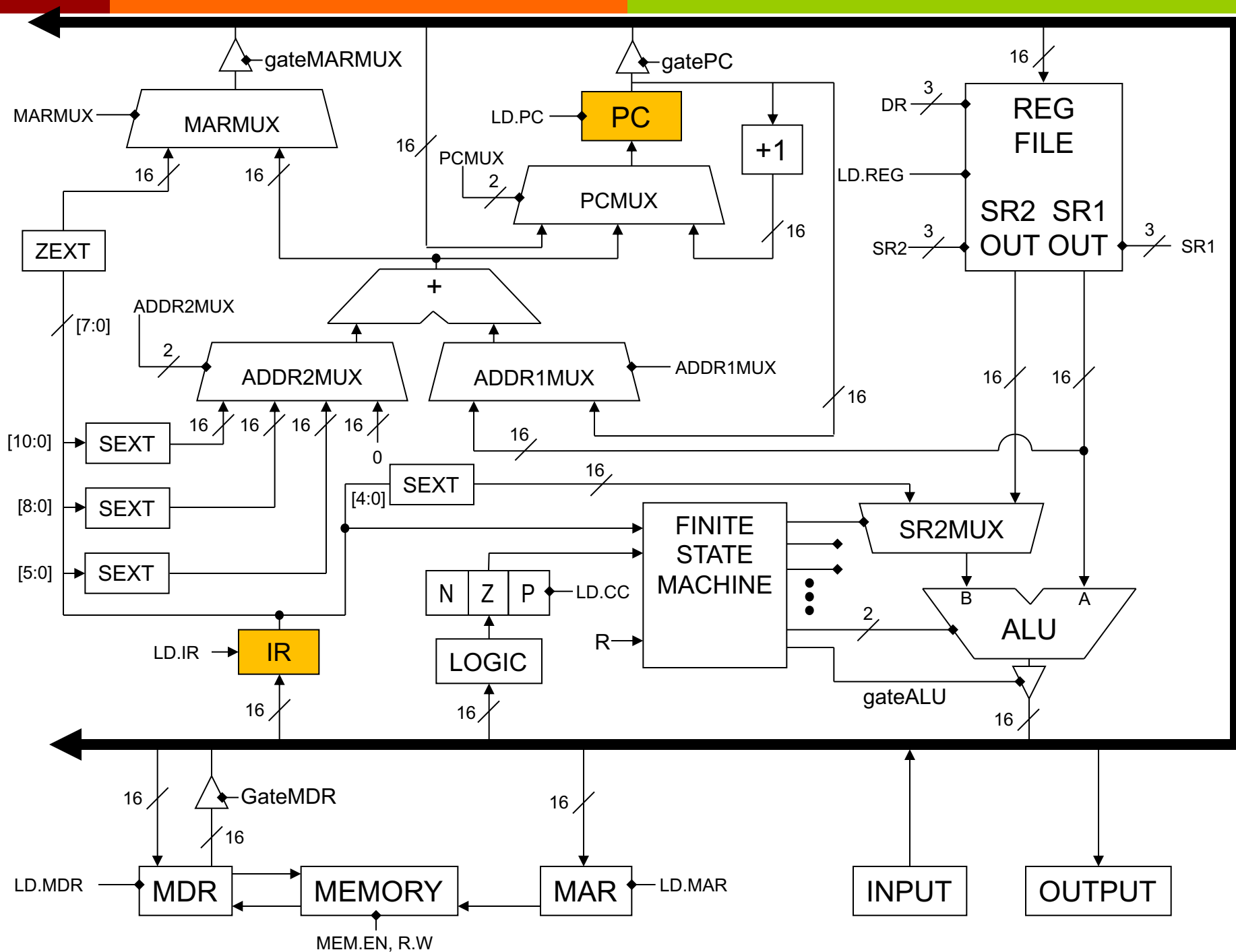
- SR1=001
- SR2=011
- DR=010



Exactly How Did the FSM Get That Add Instruction?

- Before the FSM could execute the ADD instruction in the previous slides, it had to fetch the instruction from memory
- The FSM has a procedure that it executes over-and-over to process instructions
 - FETCH
 - DECODE
 - EXECUTE
- The FSM fetches and decodes each instruction before it executes it
- We'll talk about the "fetch-execute" loop in detail in a bit

Back to the full LC-3 Datapath




What is a machine code program?

- A series of machine code instructions.
- Each instruction is a 16-bit data value.
- The instructions are stored in memory.
 - Usually in sequence
 - Consecutive memory locations
- Each memory location has an address
 - The address is a 16-bit unsigned integer.
- **Do you know what the PC and IR registers do, from the Patt book?**

Program Counter (PC)


The Program Counter (PC) register holds

- A. The count of instructions executed
- B. The address of the next instruction to be executed
-  C. The instruction currently being executed
- D. Number of programs completed since the last boot

E. Today's number is 48,360

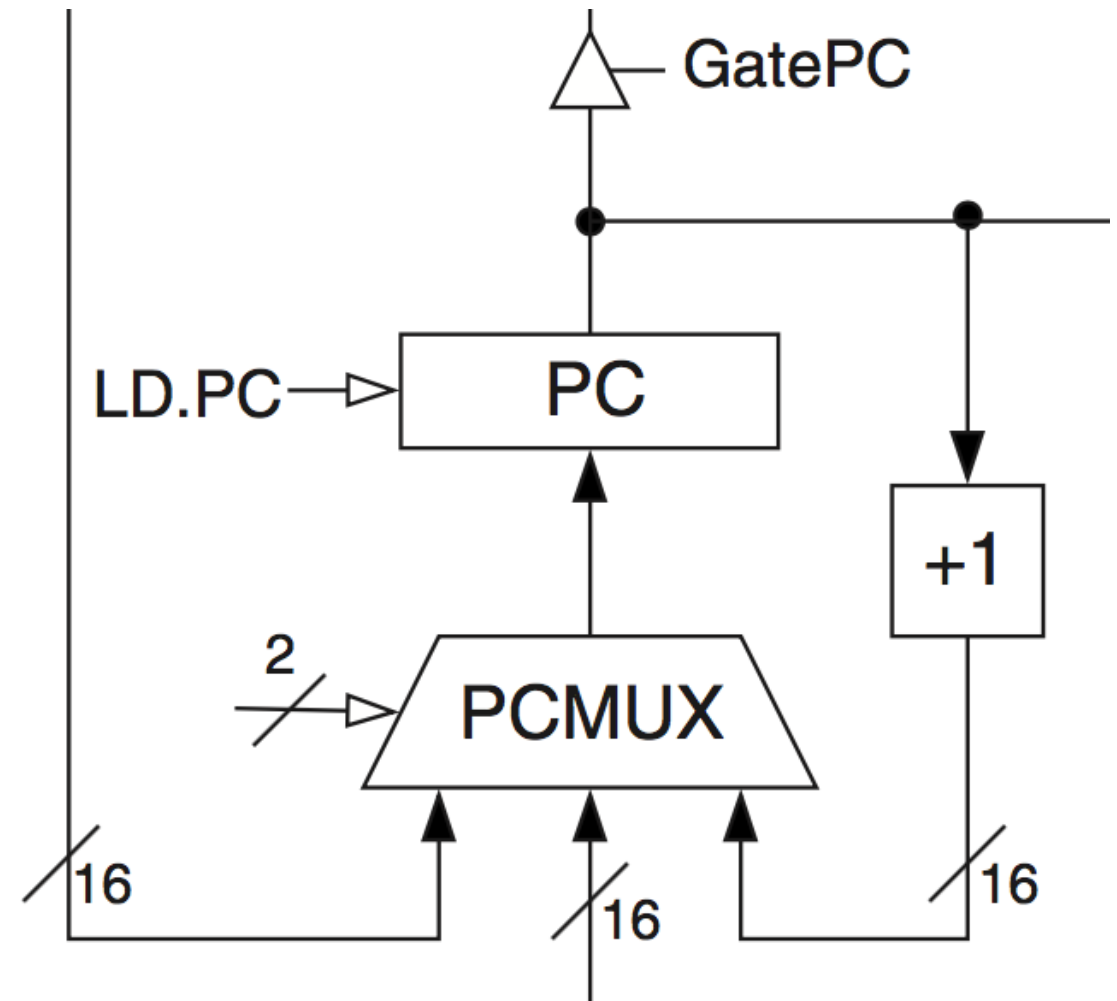
Instruction Register (IR)

The Instruction Register (IR) holds

- A. All legal op codes
- B. The address of the next instruction to be executed
- C. All data needed to execute the current instruction
-  D. The instruction currently being executed

Program Counter circuit

- PCMUX
 - Adder to increment PC (+1)
 - Data from bus to PC
 - Data from effective address* calculator to PC
- LD.PC
 - Write enable for PC reg
- GatePC
 - Put PC value on bus



* We'll explain effective address, and why you need it, later. But you know what a MUX is.

Control Signals for PC circuit

- GatePC (tri-state buffer)
- LD.PC (write enable for PC register)
- PC MUX (2 bit mux selector)

- 4 bits of control signals total

FETCH and DECODE

- The ADD machine code instruction – that we saw earlier – lives in memory (*in our machine code program*).
- Before we do the add (or any other instruction), we must get the machine code instruction from memory.
 - The PC register tells us the memory address where the instruction is located.
 - The IR holds the value read from memory (the machine code 16-bit instruction itself)

FETCH and DECODE

➤ FETCH

- Takes 3 clock cycles to read data (the instruction) from memory

DECODE

- Takes 1 clock cycle

- This is where the FSM generates the control signals for the specific instruction (such as ADD).

- Total: 4 clock cycles (states) to fetch and decode an instruction.

- This happens at the beginning of EVERY machine code instruction.

Machine Code Instruction Cycle: Progression through States

- Fetch
- Decode
- Evaluate Address *[optional]*
- Fetch Operands *[optional]*
- Execute *[optional]*
- Store Result *[optional]*



ADD instruction only uses Fetch, Decode, Execute

➤ Fetch

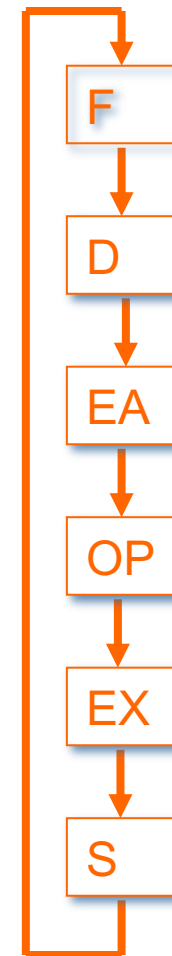
➤ Decode

~~➤ Evaluate Address *[optional]*~~

~~➤ Fetch Operands *[optional]*~~

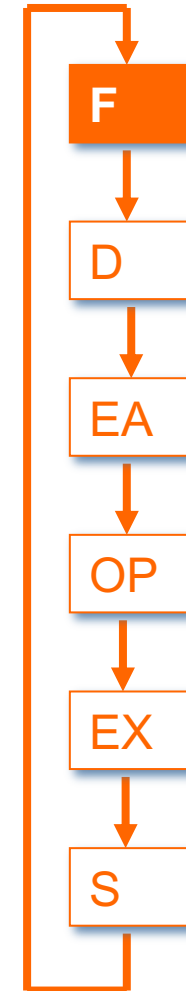
➤ Execute *[optional]*

~~➤ Store Result *[optional]*~~



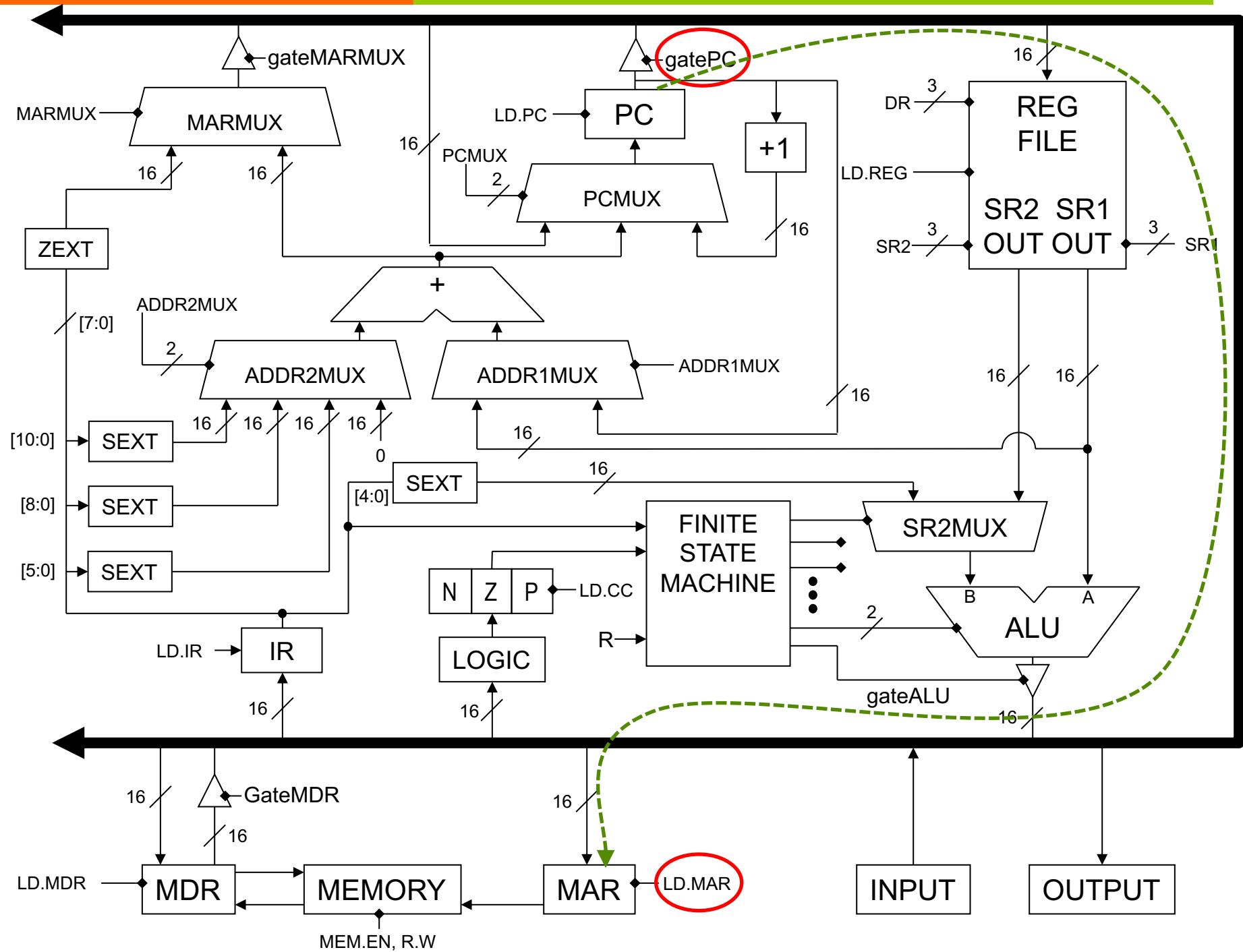
Instruction Processing: FETCH

- Load next instruction (at address stored in PC) from memory into Instruction Register (IR).
 - Copy contents of PC into MAR.
 - Send “read” signal to memory.
 - Copy contents of MDR into IR.
- Increment PC, so that it points to the next instruction in sequence.
 - PC becomes PC+1.



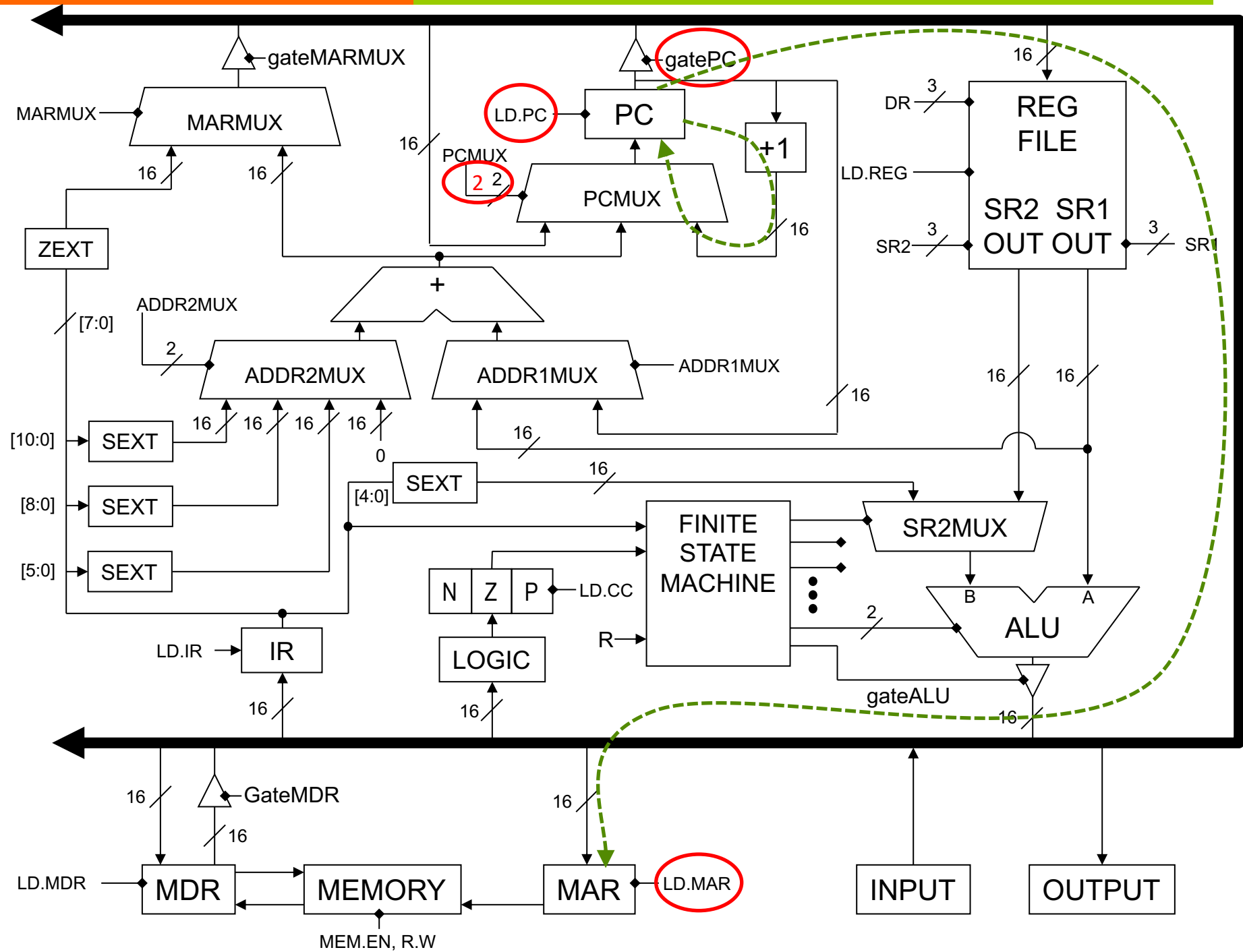
FETCH (Phase 1)

- Copy PC to MAR



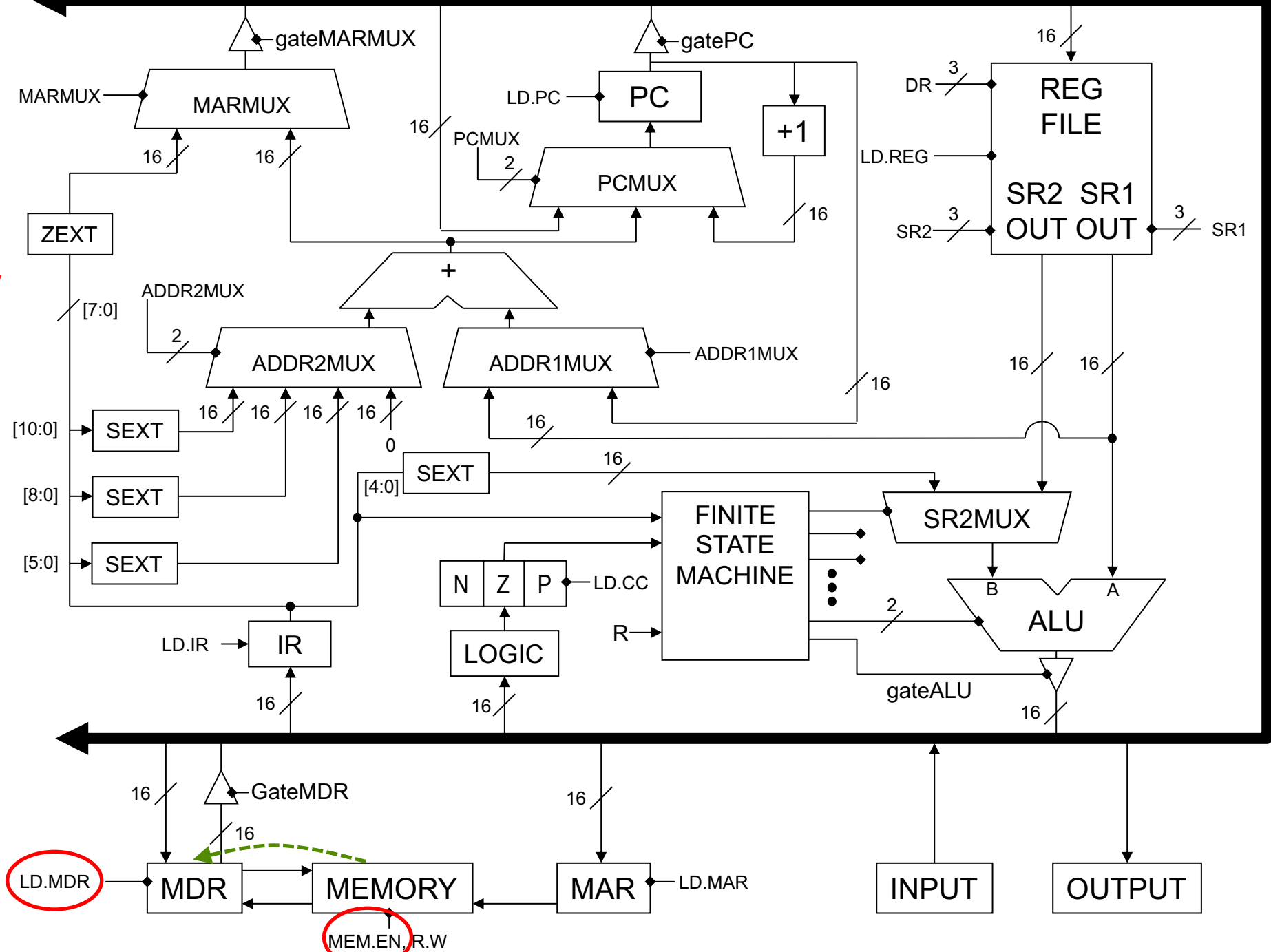
FETCH (Phase 1)

- Copy PC to MAR
- Increment PC



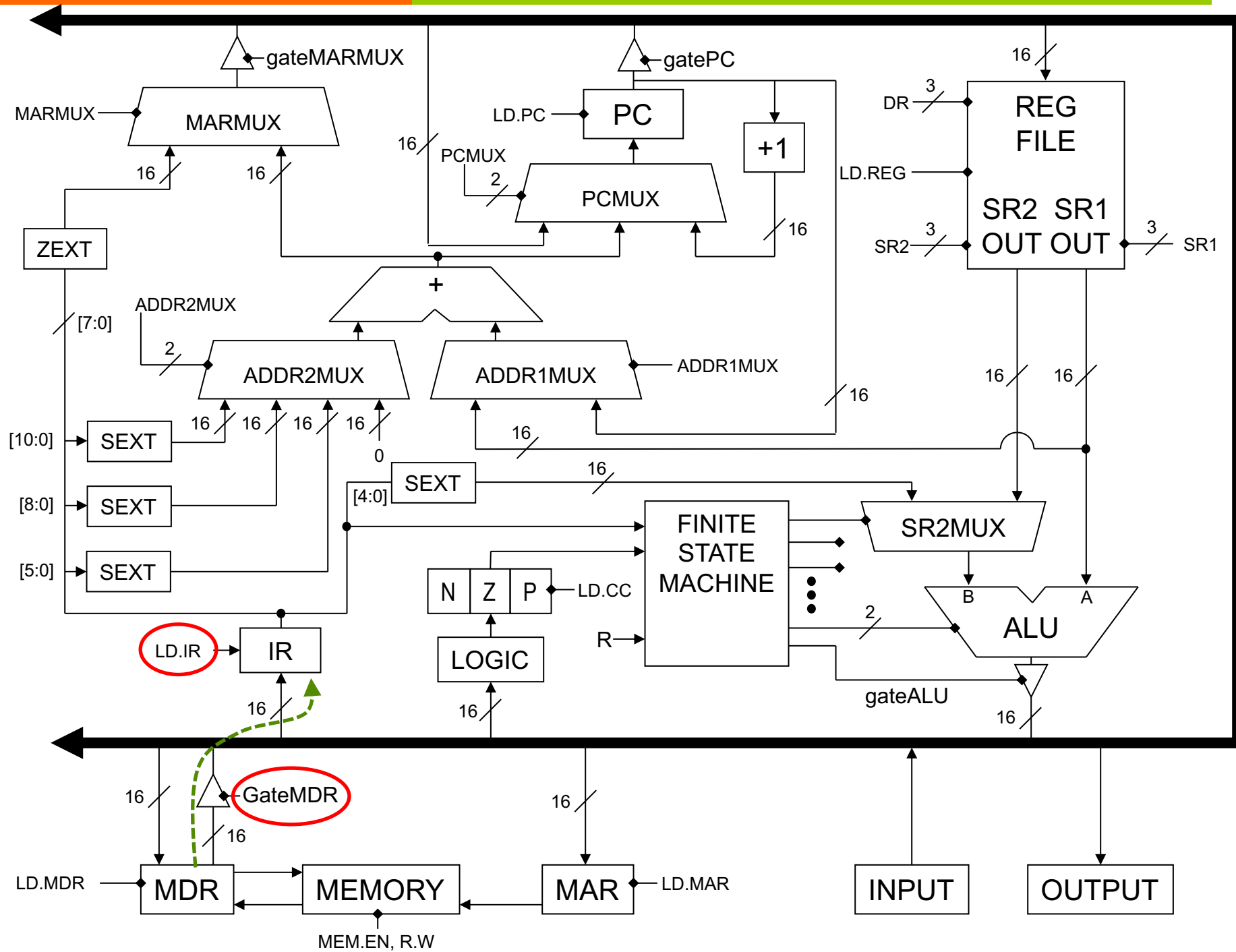
FETCH (Phase 2)

- Read from memory into MDR



FETCH (Phase 3)

- Copy MDR into IR



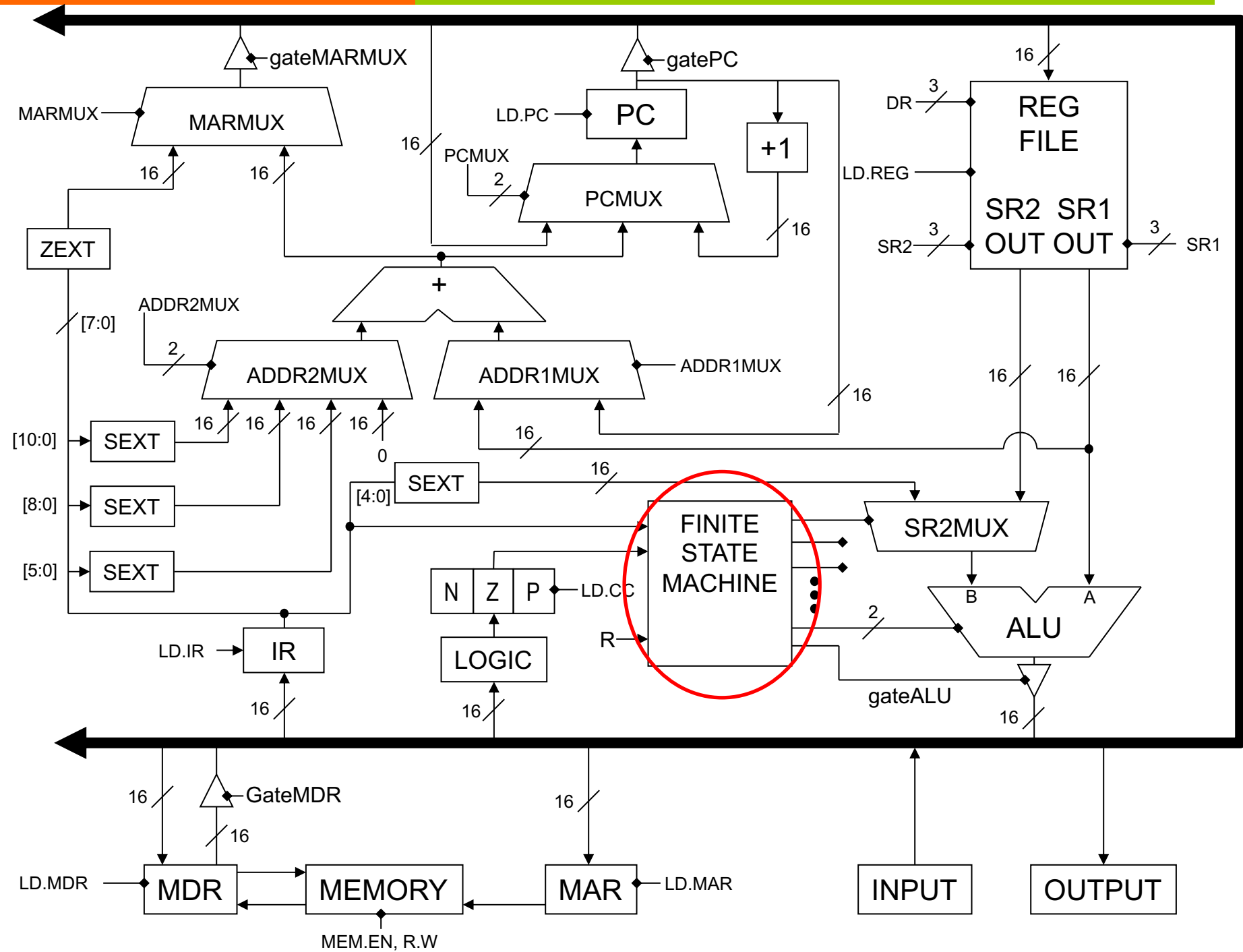
Instruction Processing: DECODE

- First identify the opcode.
 - On the LC-3, this is 4 bits [15:12].
 - The FSM chooses a next state corresponding to the desired opcode.
- Depending on opcode, identify other operands from the remaining bits.
 - Example:
 - for LDR, last six bits is offset
 - for ADD, last three bits is source operand #2
- DECODE changes Finite State Machine state!
 - It uses the opcode to choose a next state
 - It's similar to a 16-way "go to" for the next state



DECODE

- FSM



Instruction Processing: EXECUTE

- Perform the operation, using the source operands.
- Examples:
 - send operands to ALU and assert ADD control signal
 - calculate nothing (e.g., for loads and stores to memory – more on this later)



- It depends
 - Every instruction has a different execute phase
 - With different control signals
- But every instruction starts with:
 - FETCH
 - FETCH
 - FETCH
 - DECODE

Instruction Processing Review

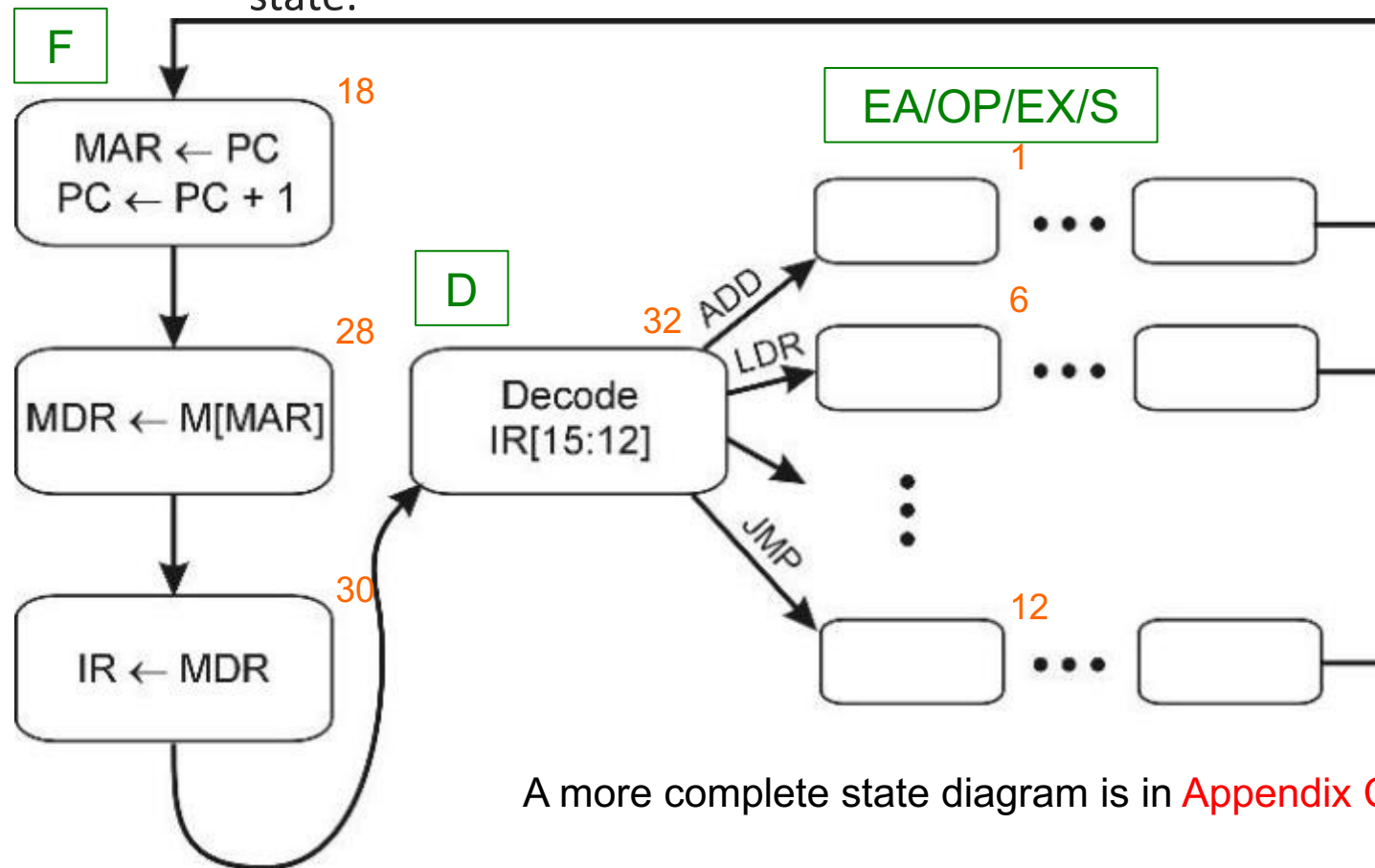
- Instruction bits look just like data bits in memory – it's all a matter of our interpretation
- Three basic kinds of instructions:
 - computational instructions (ADD, AND, NOT)
 - data movement instructions (LD, ST, ...)
 - control instructions (JMP, BRnz, ...)
- Six basic phases of instruction processing:
 - $F \rightarrow D \rightarrow EA \rightarrow OP \rightarrow EX \rightarrow S$
 - not all phases are needed by every instruction
 - phases may take variable number of machine cycles (states)

The LC-3 is a Finite State Machine

- LC-3 Finite State Machine
 - Has 64 possible states
 - Orchestrates 42 control signals
 - Multiplexor selectors
 - PCMUX, MARMUX, ADDR2MUX, ...
 - Tri-state buffer enables
 - gatePC, gateMARMUX, gateALU,...
 - Register write-enables
 - LD.PC, LD.REG, LD.MAR, LD.CC, ...
 - Other control signals
 - ALUK, MEM.EN, R.W, ...
- The wires aren't all shown explicitly on the datapath diagram – *to avoid clutter*, but each signal has a name

What Makes the Fetch-Execute Cycle Happen?


- The FSM, which sequences through the states.
- It turns on the appropriate control signals as it enters each state.




A more complete state diagram is in [Appendix C Fig C.3](#)

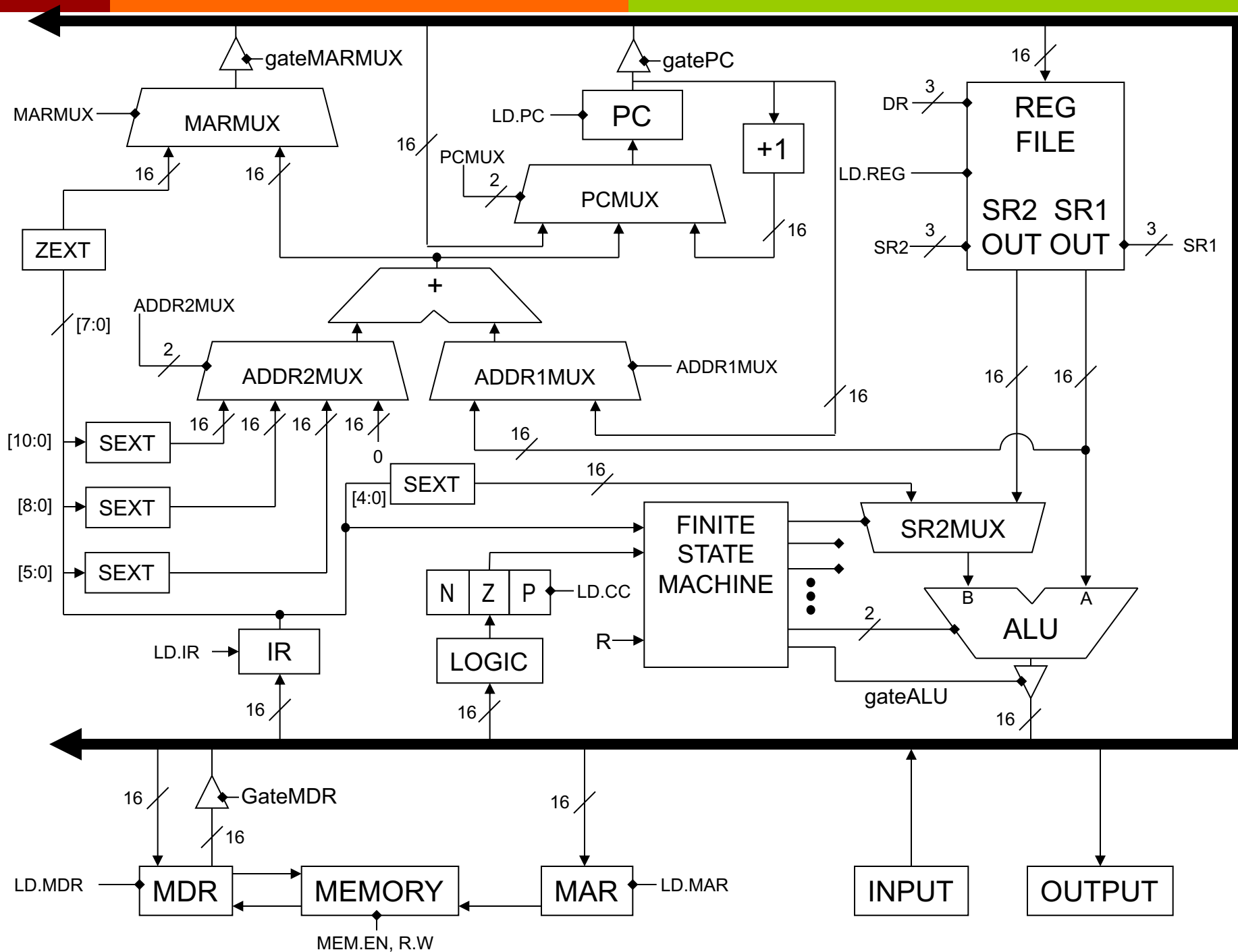
The LC-3 is a Finite State Machine

- How do we implement the LC-3 state machine?
 - Just like the garage door opener, we build logic to produce output control signals.
 - The garage door had 2 output signals: UP and DOWN.
- The LC-3 has 42 control signals.
 - These signals control the datapath (the MUX selectors, tri-state buffers, write enables (e.g LD.REG), etc.
 - The output of the FSM is the 42 control signals.
- The LC-3 is a glorified state machine.
 - But works just like the garage door opener.
 - It has 64 states, and 42 control signal outputs.



When the Fetch-Execute cycle reaches the EXECUTE phase for an instruction

- A. The PC contains the address of the next instruction to be executed Why not?
- B. The PC contains the address of the instruction being executed
- C. The PC contains the address of the word after the instruction being executed 
- D. The PC contains the instruction being executed



- Basic CPU Components
 - Memory, Processing Unit, Input & Output, Control Unit
- Architectures
 - Harvard
 - Von Neumann
- LC-3: A von Neumann Architecture
- Tri-State Buffers
- Instruction Cycle
 - Instructions
 - Fetch, Decode, Evaluate Address, Fetch Operands, Execute, Store Result