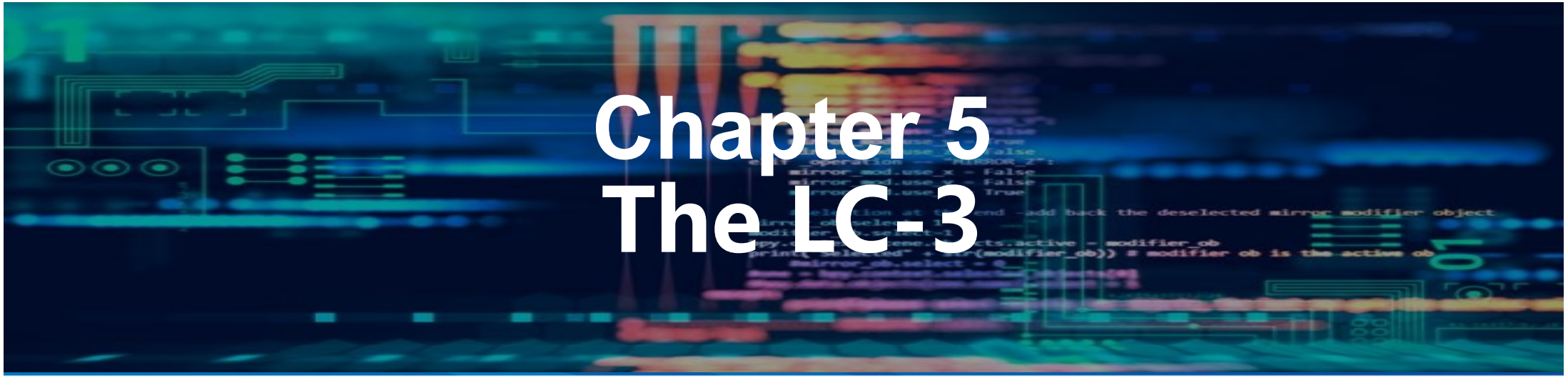
中国科学技术大学
University of Science and Technology of China

计算系统概论A
Introduction to Computing Systems
(CS1002A.03)

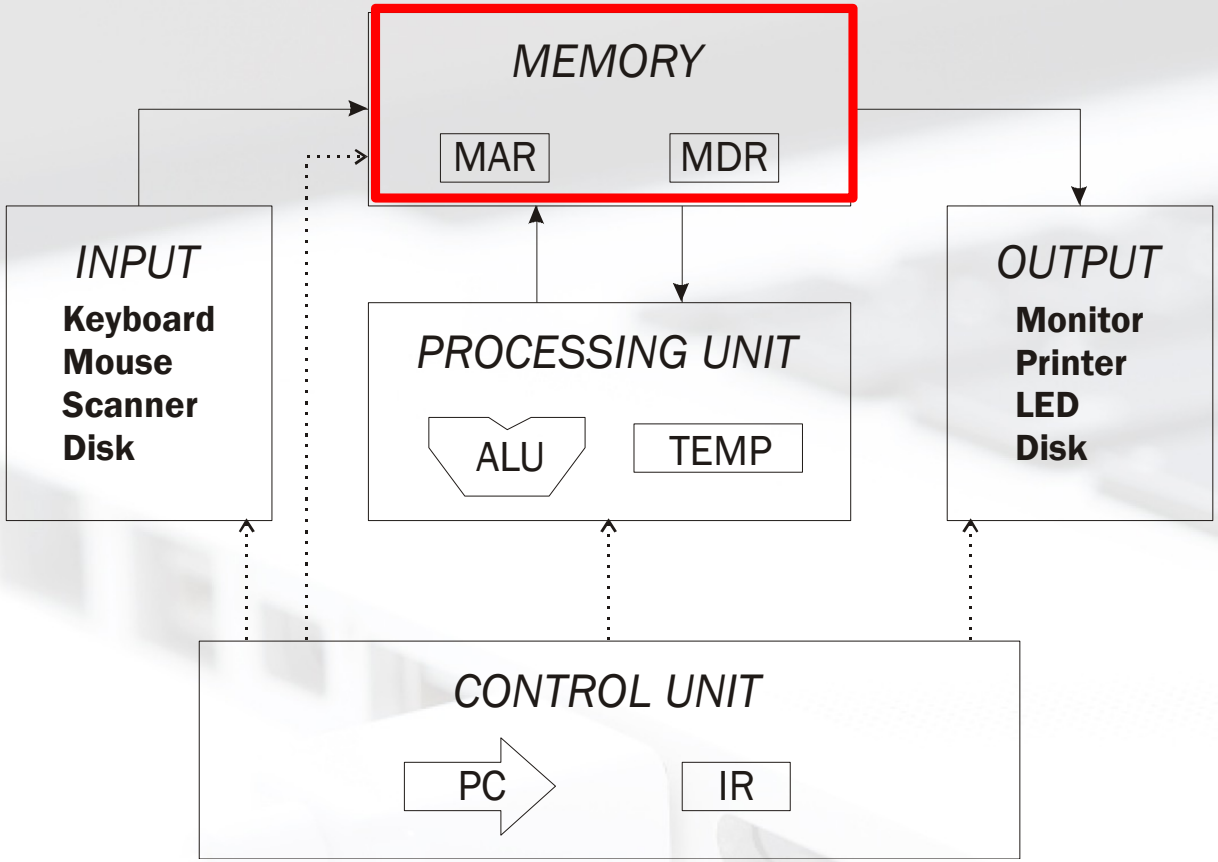


Chapter 5
The LC-3

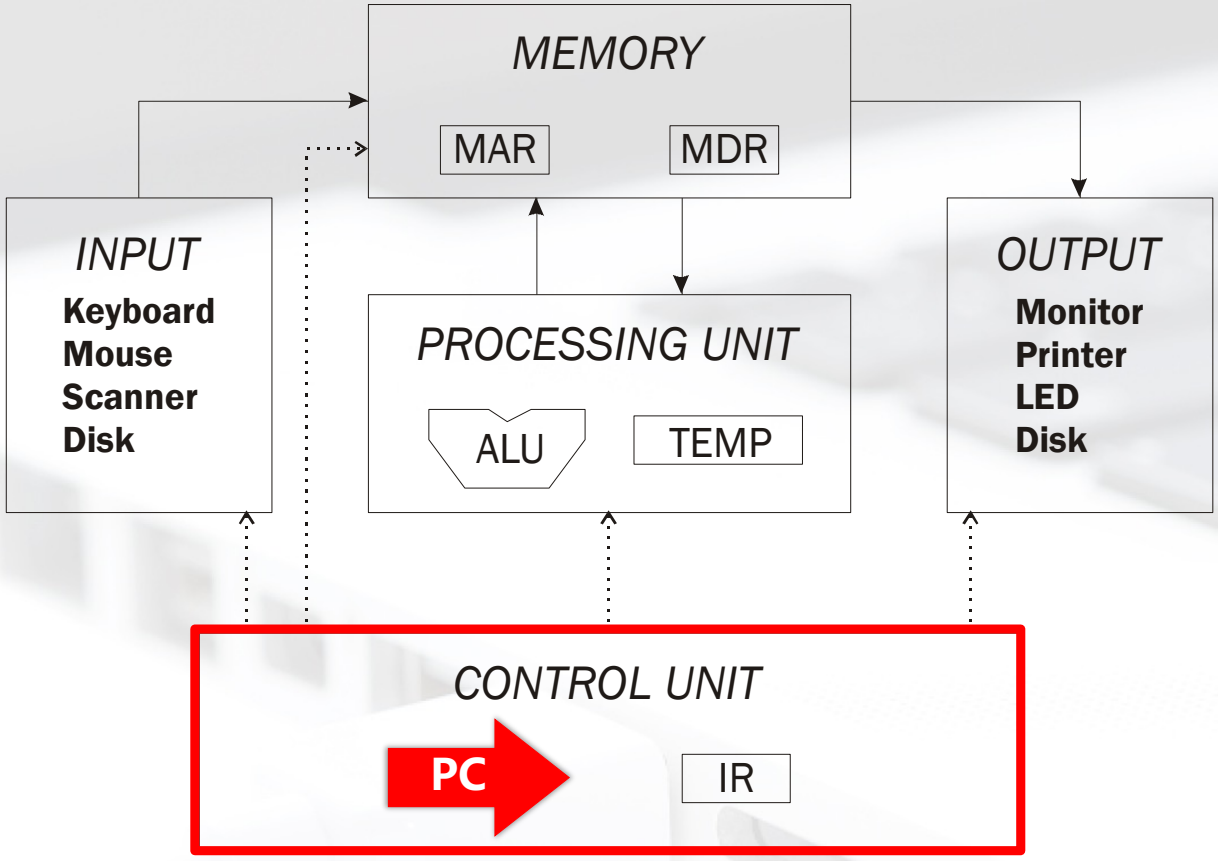
陈俊仕
cjuns@ustc.edu.cn
2023 Fall

计算机科学与技术学院
School of Computer Science and Technology

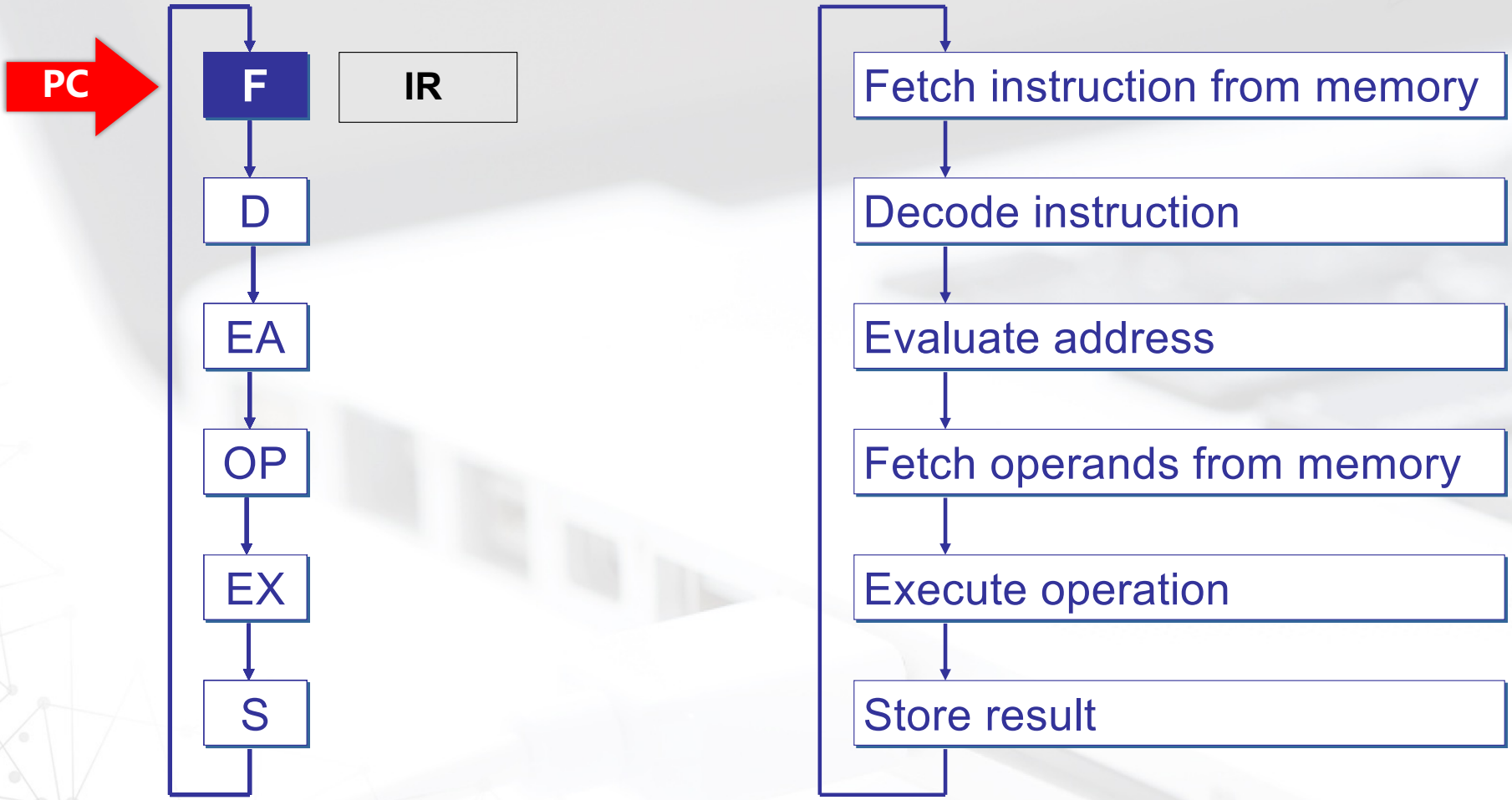
Review: Great Idea #2 Von Neumann Structure （Architecture Model）



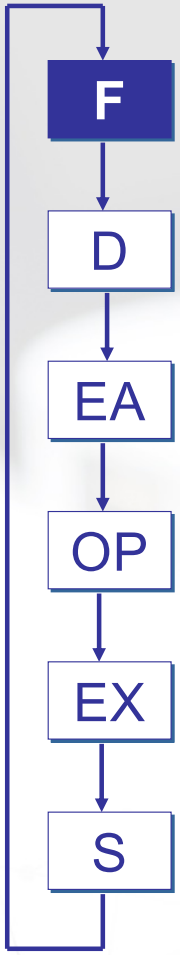
Review: Great Idea #1 Turing Machine（Computational Model)



Review: Instruction Processing(State Transition)



Instruction Processing: Finite State Automata



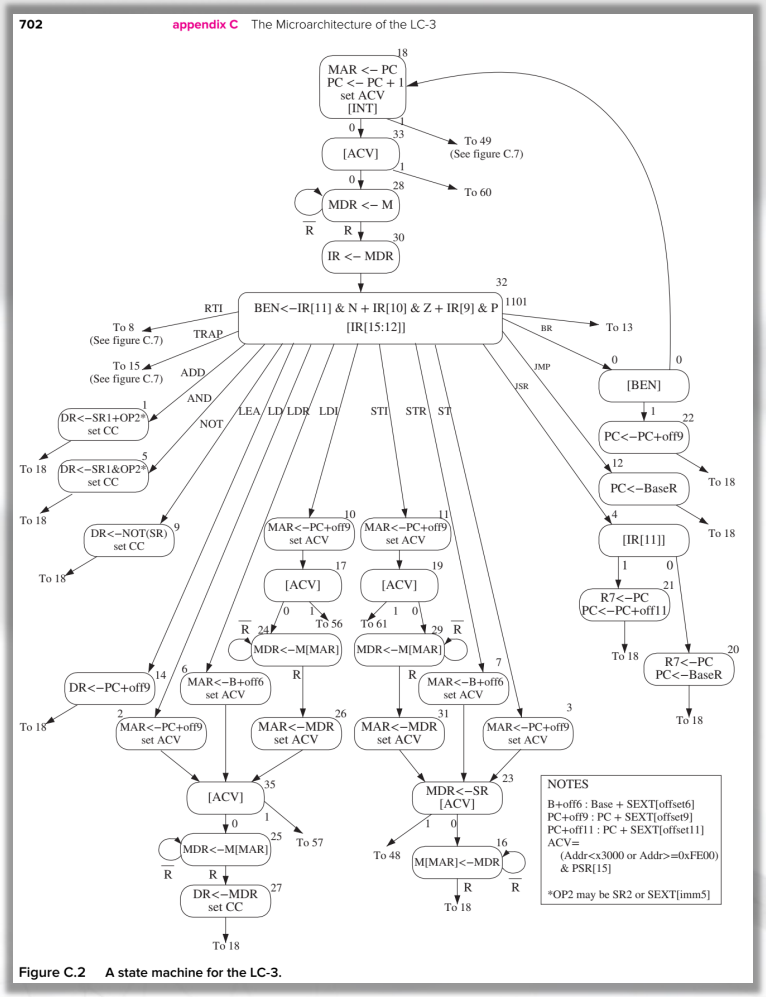


Figure C.2 A state machine for the LC-3.

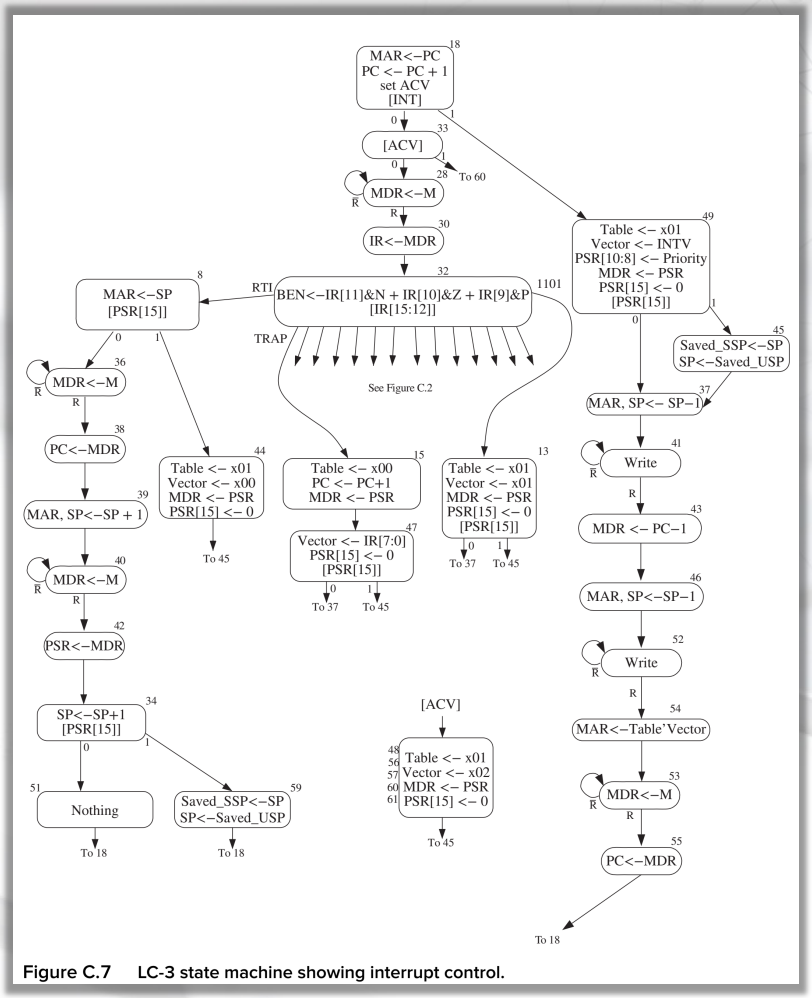
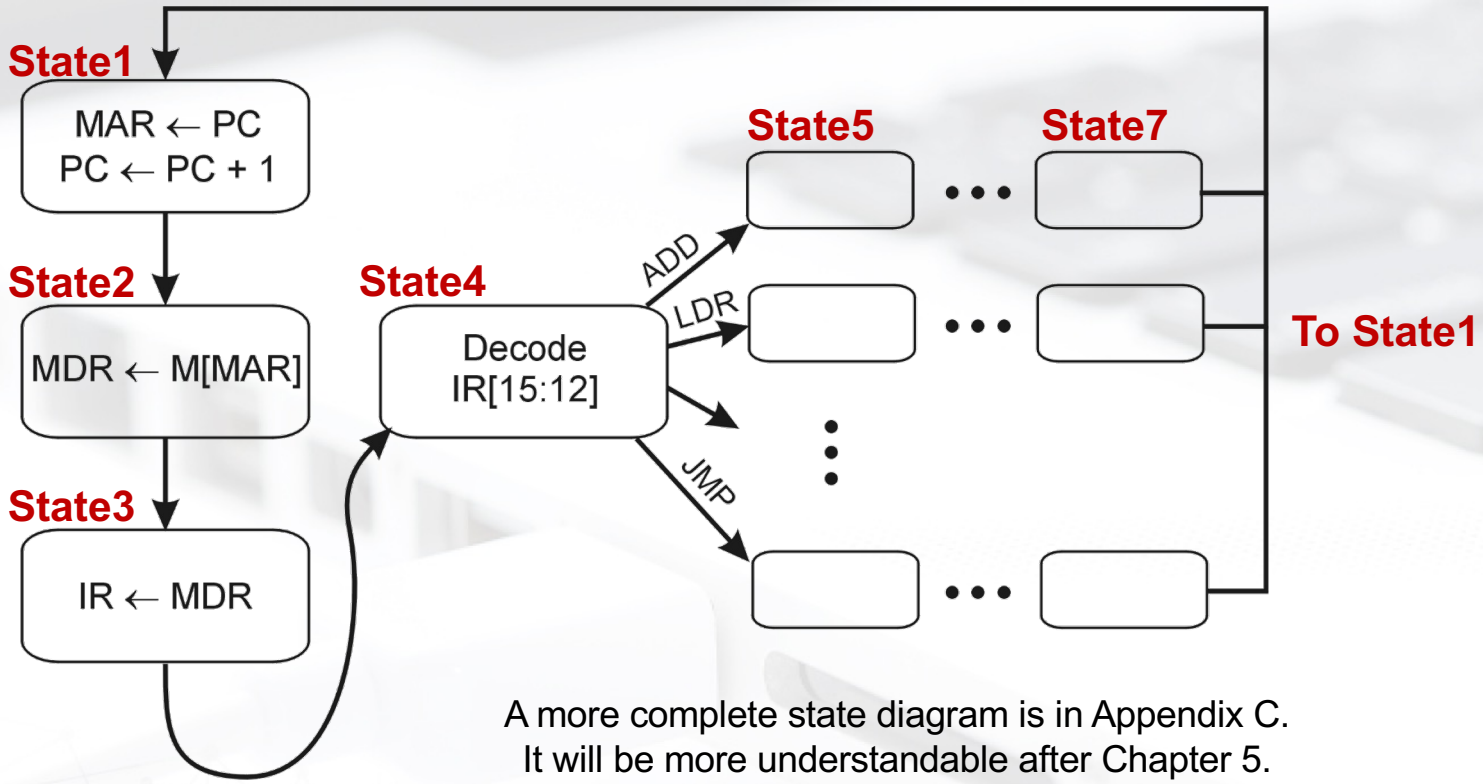


Figure C.7 LC-3 state machine showing interrupt control.

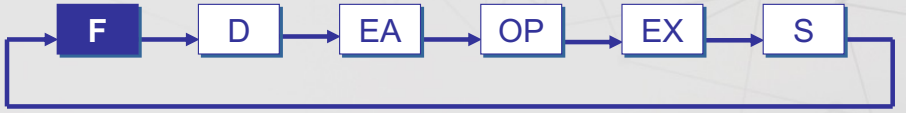
Control of the Instruction Cycle

- The control unit is a state machine. Here is part of a simplified state diagram for the LC-3:

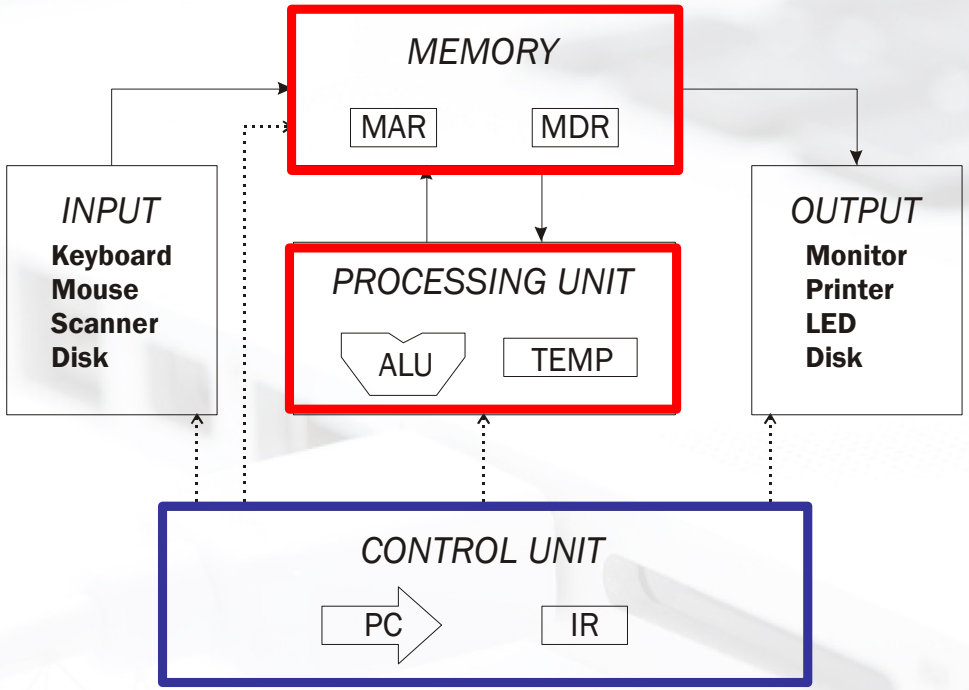


A more complete state diagram is in Appendix C.
It will be more understandable after Chapter 5.

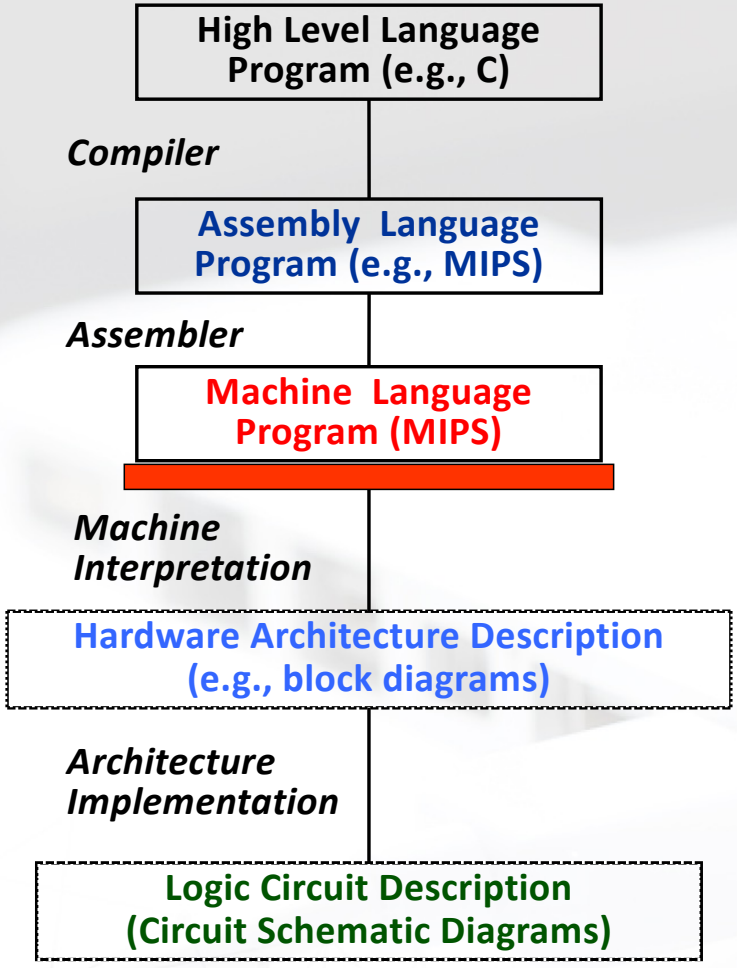
Today



- We are going to learn how to:
  - compute with values in registers
  - load data from memory to registers
  - store data from registers to memory



How do we get the electrons to do the work?



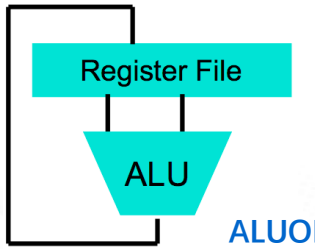

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw      $t0, 0($2)
lw      $t1, 4($2)
sw      $t1, 0($2)
sw      $t0, 4($2)
```

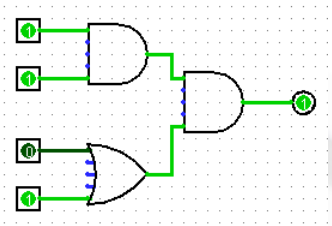
Anything can be represented as a *number*, i.e., data or instructions

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

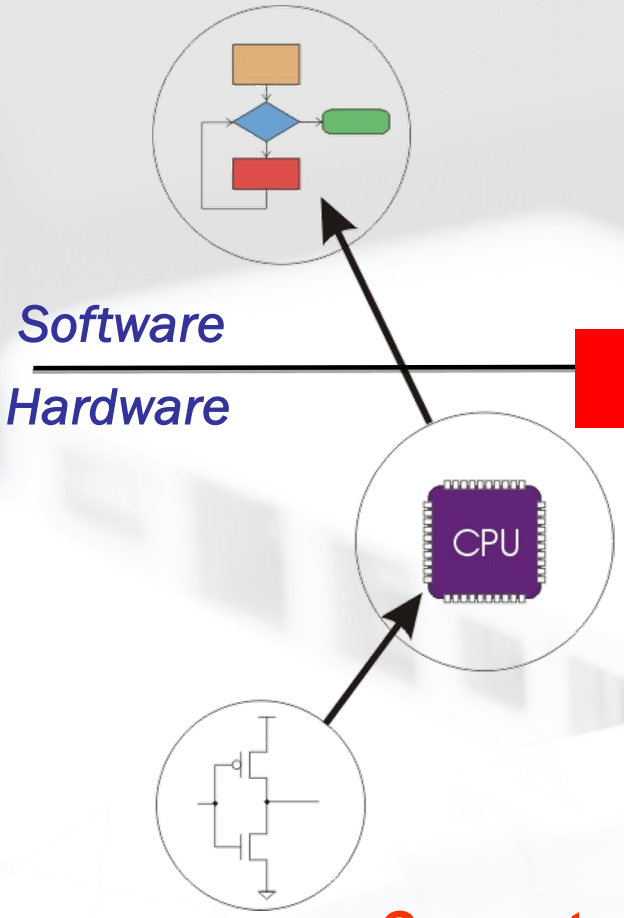
Now, You are Here.



ALUOP[0:3] <= InstReg[9:11] & MASK



Today: Software and Hardware Co-design



Application

Algorithm & Data Structure

Language

Machine Architecture, ISA

Microarchitecture

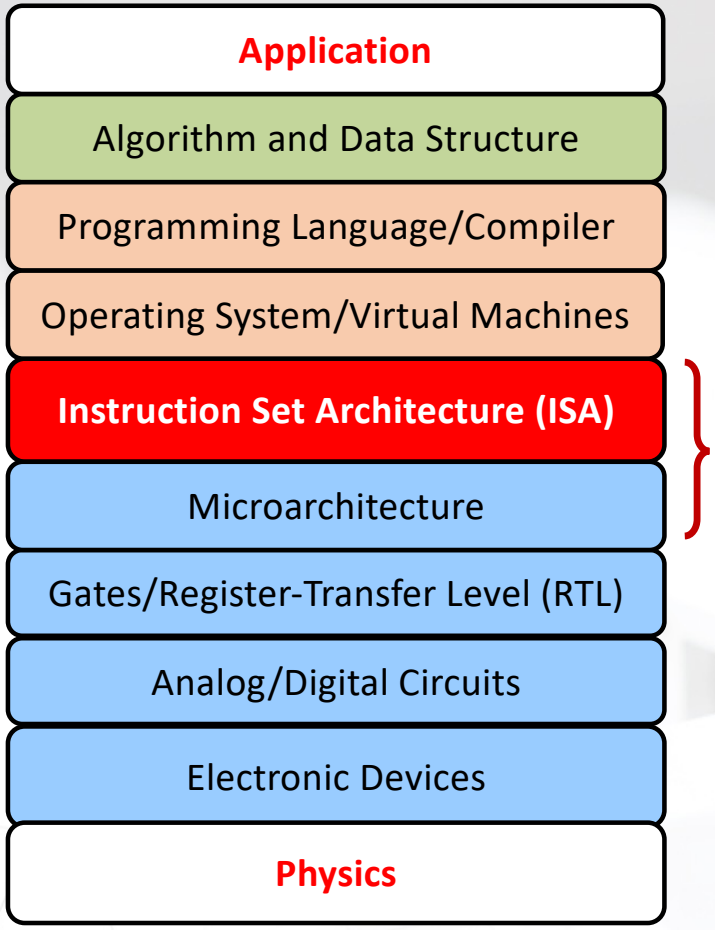
Logic and IC

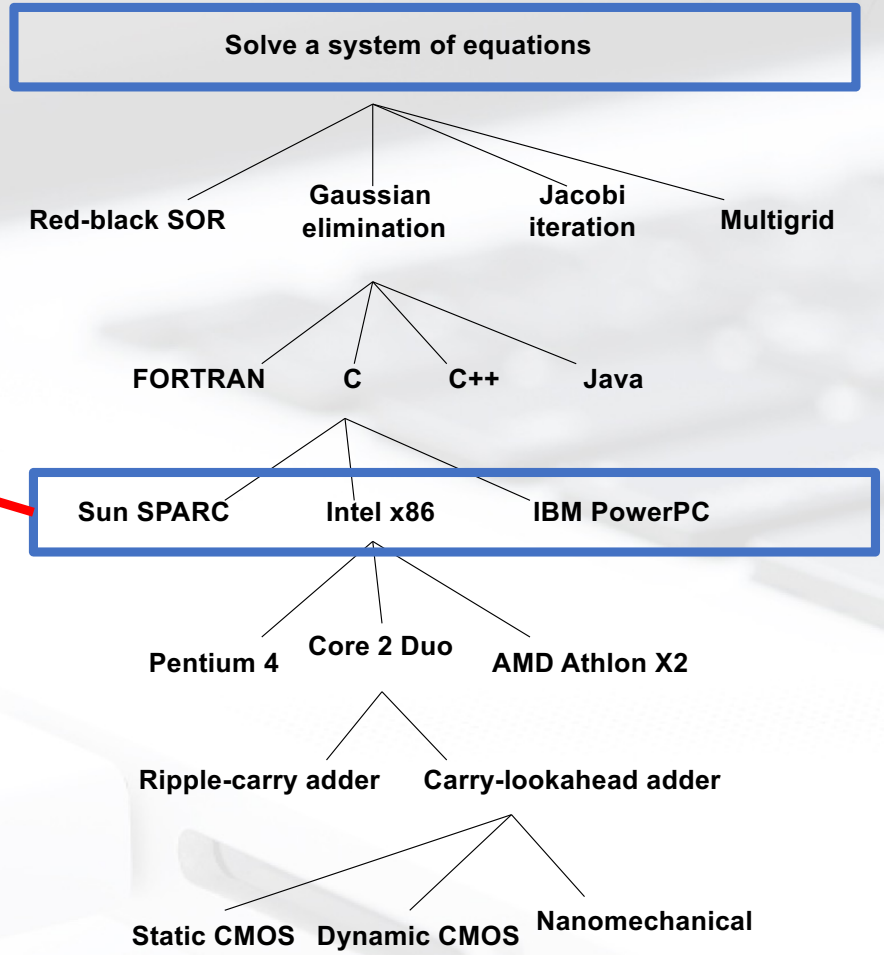
Device

Now, You are Here.

Computer System: Layers of Abstraction

Today: Abstraction Helps Us Manage Complexity



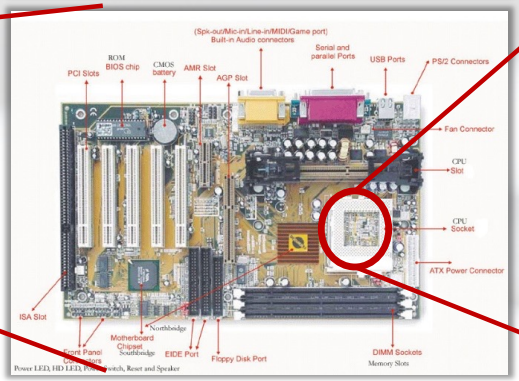


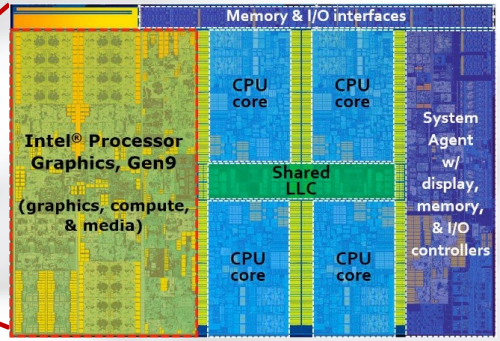
Today: Bottom up approach

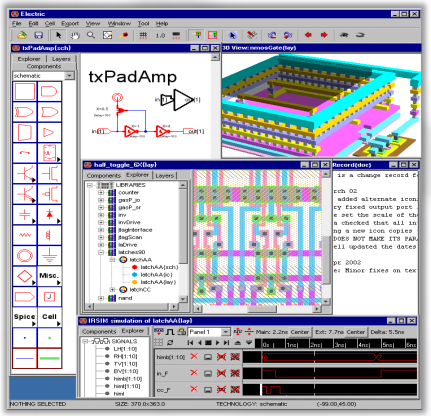
Personal Computer:
Hardware & Software Design
1~10PCBs/System

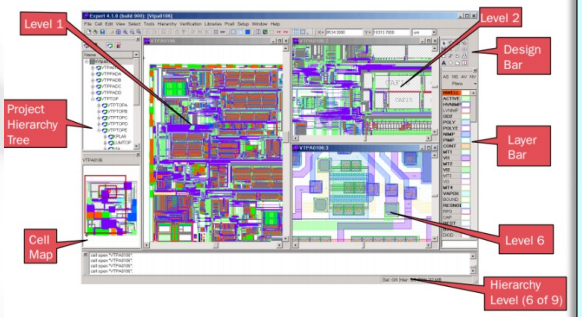
Motherboard Circuit Design
10 ICs/ PCB
1~50G Devices

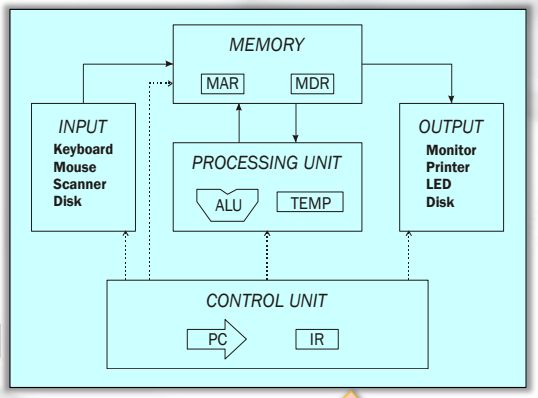
Integrated Circuit Design
100 Modules/ IC
0.25M~20G Devices











Electronic System Level (ESL)Design

Now, You are Here.

Outline

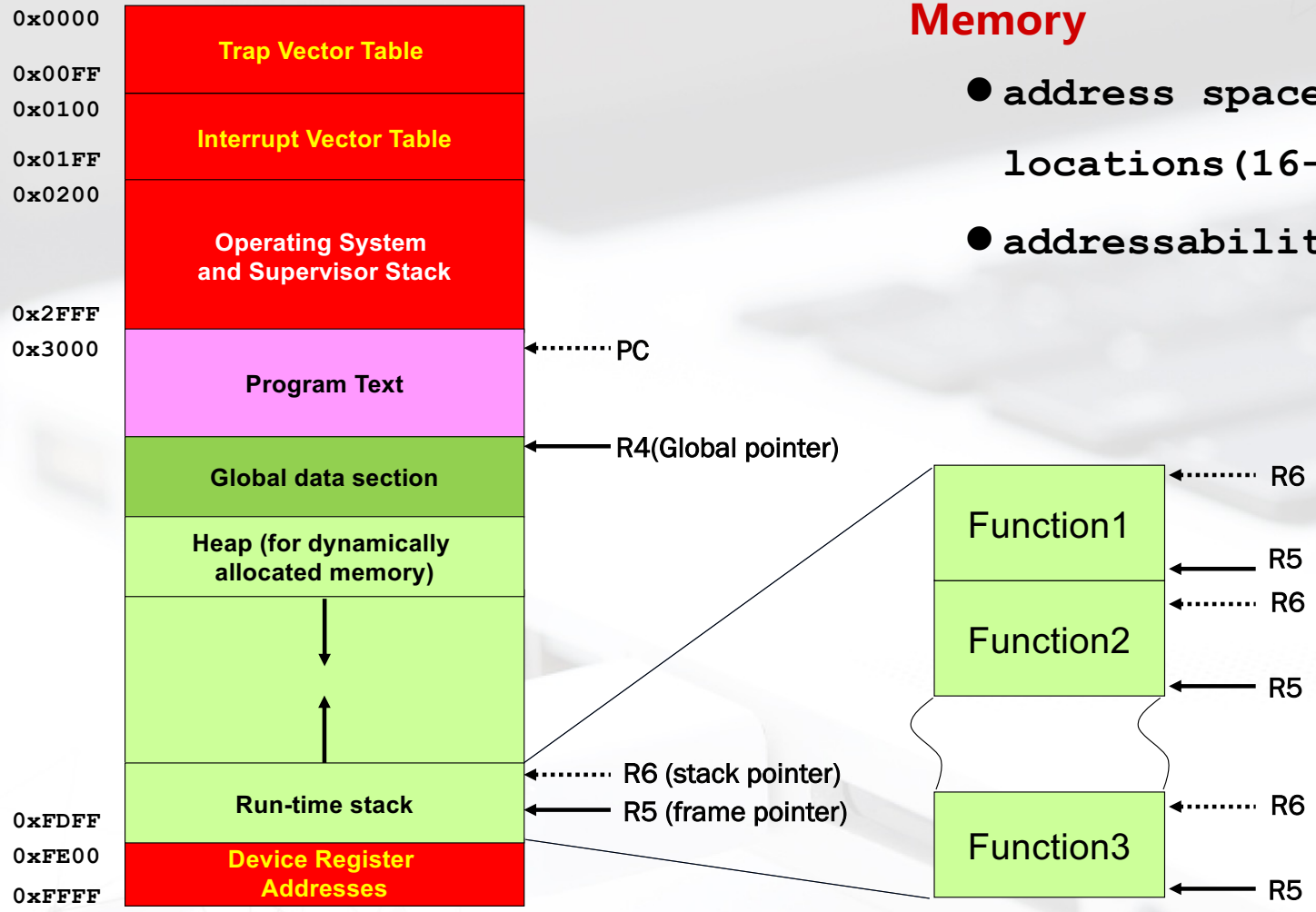
Instruction Set Architecture

ISA = All of the *programmer-visible* components and operations of the computer

- **`memory organization`**
  - address space -- how may locations can be addressed?
  - addressability -- how many bits per location?
- **`register set`**
  - how many? what size? how are they used?
- **`instruction set`**
  - opcodes
  - data types
  - addressing modes

ISA provides all information needed for someone that wants to write a program in machine language (or translate from a high-level language to machine language).

LC-3 Overview: Memory



Memory

- address space: 2^{16} locations(16-bit addresses)
- addressability: 16 bits

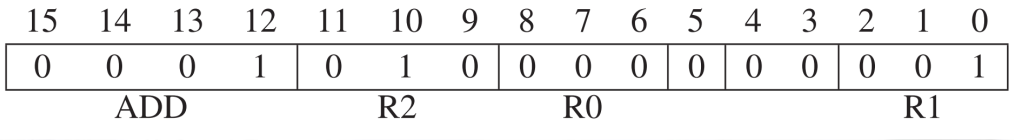
LC-3 Overview: Registers

Registers

- **temporary storage, accessed in a single machine cycle**
  - —accessing memory generally takes longer than a single cycle
- **eight general-purpose registers: R0 - R7**
  - —**each 16 bits wide**
  - —how many bits to uniquely identify a register?
- **other registers**
  - —not directly addressable, but used by (and affected by) instructions
  - —**PC (program counter), condition codes**

Register 0	(R0)	0000000000000001
Register 1	(R1)	0000000000000011
Register 2	(R2)	0000000000000101
Register 3	(R3)	0000000000000111
Register 4	(R4)	1111111111111110
Register 5	(R5)	1111111111111100
Register 6	(R6)	1111111111111010
Register 7	(R7)	1111111111111000

Register 0	(R0)	0000000000000001
Register 1	(R1)	0000000000000011
Register 2	(R2)	0000000000000100
Register 3	(R3)	0000000000000111
Register 4	(R4)	1111111111111110
Register 5	(R5)	1111111111111100
Register 6	(R6)	1111111111111010
Register 7	(R7)	1111111111111000



LC-3 Overview: Instruction Set

Opcodes

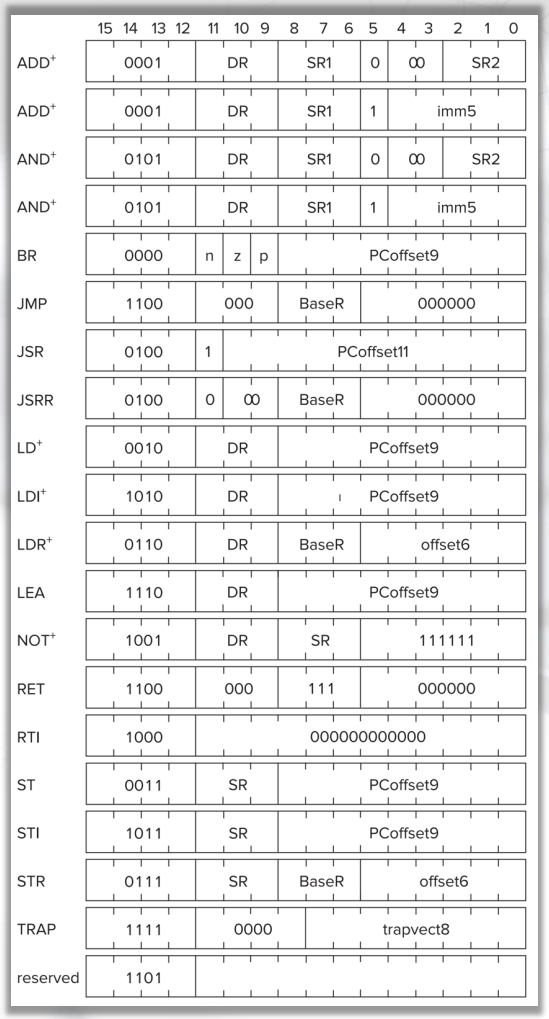
- 15 opcodes
- *Operate* instructions: ADD, AND, NOT
- *Data movement* instructions: LD, LDI, LDR, LEA, ST, STR, STI
- *Control* instructions: BR, JSR/JSRR, JMP(RET), RTI, TRAP
- some opcodes(ADD, AND, NOT; LD, LDI, LDR, LEA) set/clear *condition codes*, based on result:
  - N = negative, Z = zero, P = positive (> 0)

Data Types

- 16-bit 2's complement integer

Addressing Modes

- How is the location of an operand specified?
- non-memory addresses: *immediate, register*
- memory addresses: *PC-relative, indirect, base+offset*



LC-3 ISA Overview

运算指令

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD	0	0	0	1	DR			SR1			0	0	0	SR2		
ADD	0	0	0	1	DR			SR1			1	Imm5				
AND	0	1	0	1	DR			SR1			0	0	0	SR2		
AND	0	1	0	1	DR			SR1			1	Imm5				
NOT	1	0	0	1	DR			SR1			1	1	1	1	1	1
Reserved	1	1	0	1												

控制指令

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BR	0	0	0	0	n	z	p	PCoffset9								
JSR	0	1	0	0	1	PCoffset11										
JSRR	0	1	0	0	0	0	0	BaseR			0	0	0	0	0	0
RTI	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
JMP	1	1	0	0	0	0	0	BaseR			0	0	0	0	0	0
RET	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0
TRAP	1	1	1	1	0	0	0	0	TrapVector8							

移动数据指令

取数指令

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LD	0	0	1	0	DR			PCoffset9								
LDR	0	1	1	0	DR			BaseR			PCoffset6					
LDI	1	0	1	0	DR			PCoffset9								
LEA	1	1	1	0	DR			PCoffset9								

存数指令

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ST	0	0	1	1	SR			PCoffset9								
STR	0	1	1	1	SR			BaseR			PCoffset6					
STI	1	0	1	1	SR			PCoffset9								

Outline

Operate Instructions

Only three operations: ADD, AND, NOT

Source and destination operands are registers

- These instructions ***do not*** reference memory.
- ADD and AND can use "immediate" mode,where one operand is hard-wired into the instruction.

Will show dataflow diagram with each instruction.

- illustrates ***when*** and ***where*** data moves to accomplish the desired operation

LC-3 ISA Operate Instructions

运算指令

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD	0	0	0	1	DR			SR1			0	0	0	SR2		
ADD	0	0	0	1	DR			SR1			1	Imm5				
AND	0	1	0	1	DR			SR1			0	0	0	SR2		
AND	0	1	0	1	DR			SR1			1	Imm5				
NOT	1	0	0	1	DR			SR1			1	1	1	1	1	1
Reserved	1	1	0	1												

控制指令

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BR	0	0	0	0	n	z	p	PCoffset9								
JSR	0	1	0	0	1	PCoffset11										
JSRR	0	1	0	0	0	0	0	BaseR			0	0	0	0	0	0
RTI	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
JMP	1	1	0	0	0	0	0	BaseR			0	0	0	0	0	0
RET	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0
TRAP	1	1	1	1	0	0	0	0	TrapVector8							

移动数据指令

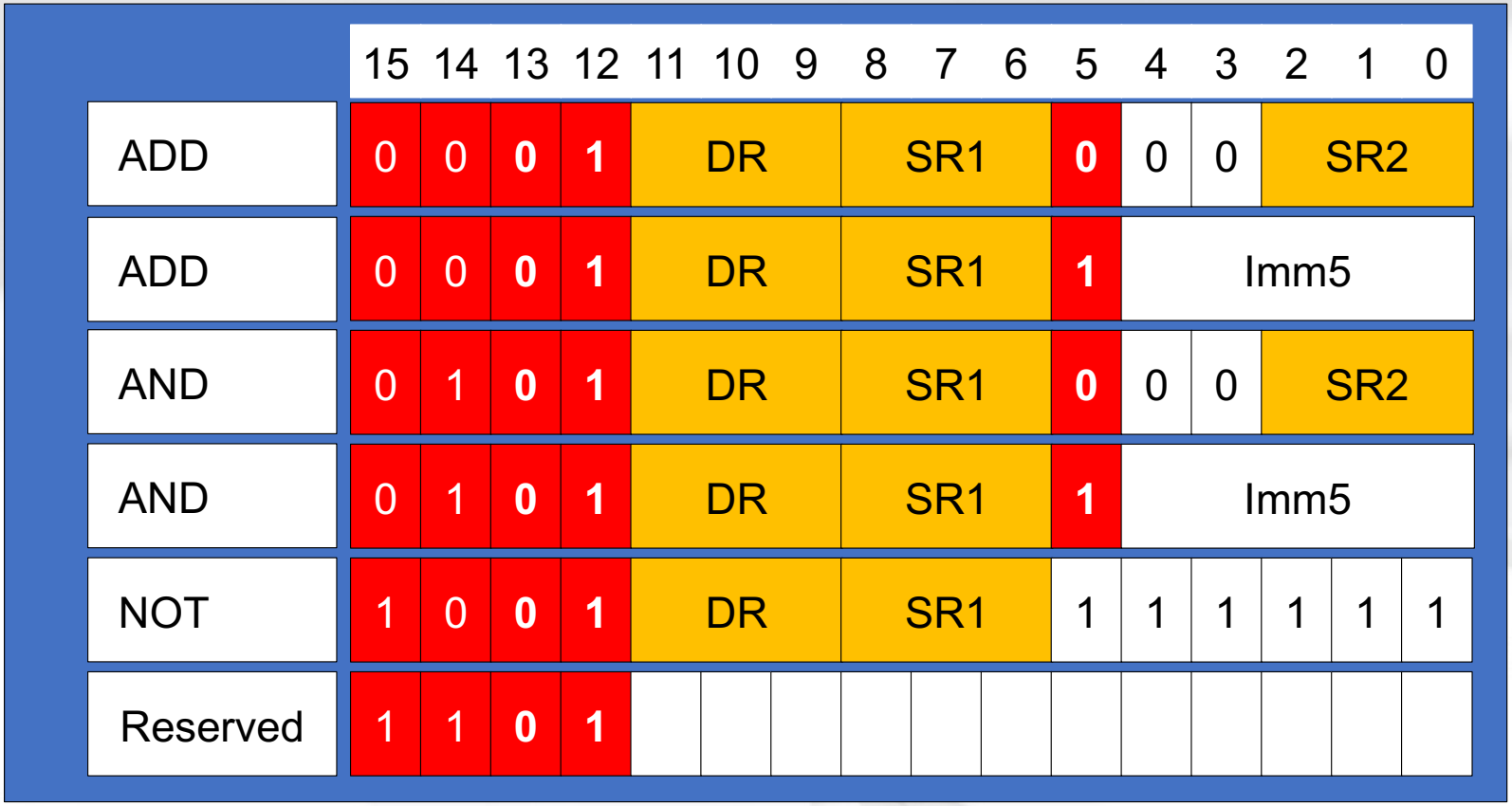
取数指令

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LD	0	0	1	0	DR			PCoffset9								
LDR	0	1	1	0	DR			BaseR			PCoffset6					
LDI	1	0	1	0	DR			PCoffset9								
LEA	1	1	1	0	DR			PCoffset9								

存数指令

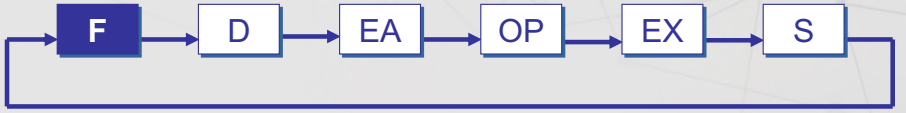
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ST	0	0	1	1	SR			PCoffset9								
STR	0	1	1	1	SR			BaseR			PCoffset6					
STI	1	0	1	1	SR			PCoffset9								

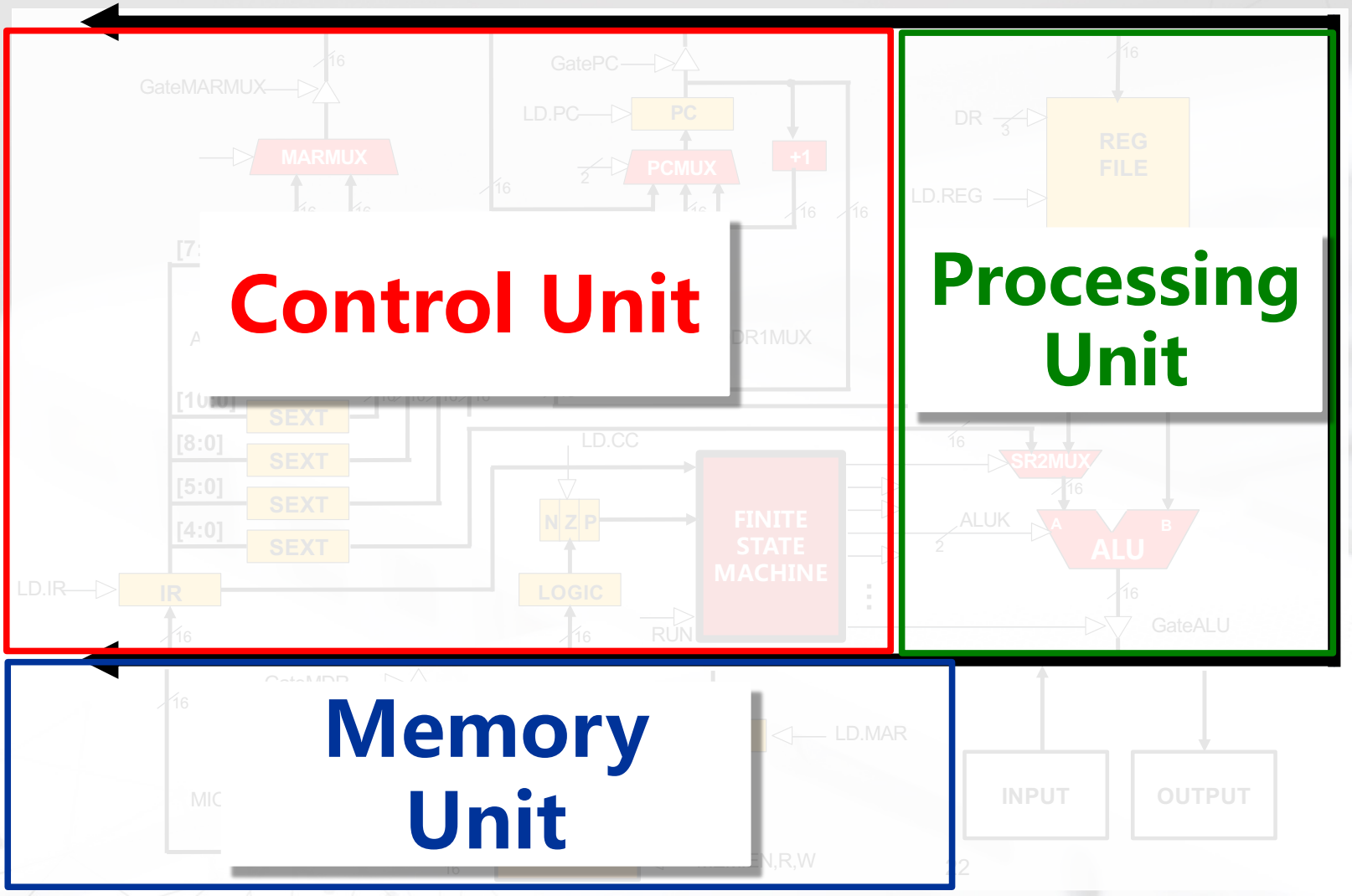
Operate Instructions Overview



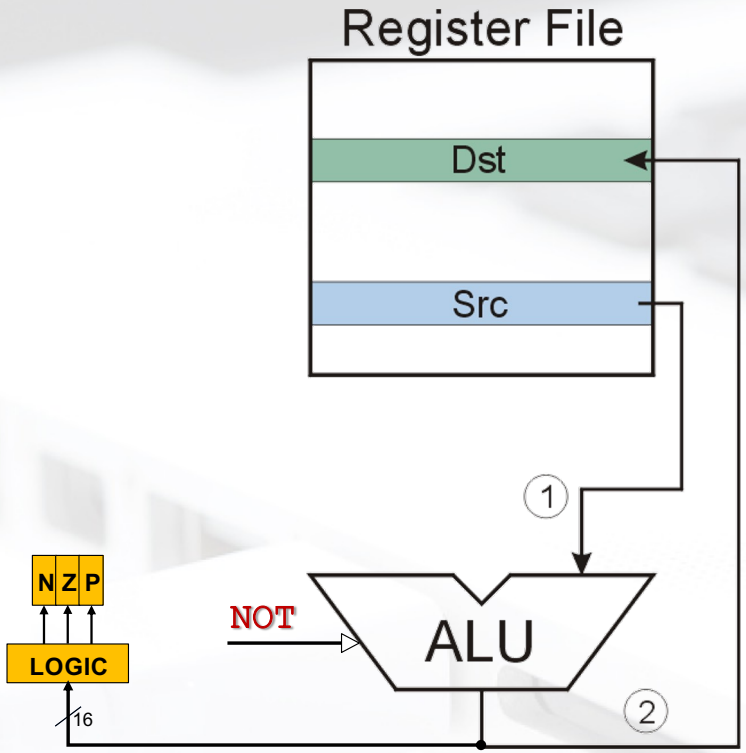
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD	0	0	0	1	DR			SR1			0	0	0	SR2		
ADD	0	0	0	1	DR			SR1			1	Imm5				
AND	0	1	0	1	DR			SR1			0	0	0	SR2		
AND	0	1	0	1	DR			SR1			1	Imm5				
NOT	1	0	0	1	DR			SR1			1	1	1	1	1	1
Reserved	1	1	0	1												

LC-3 Data Path



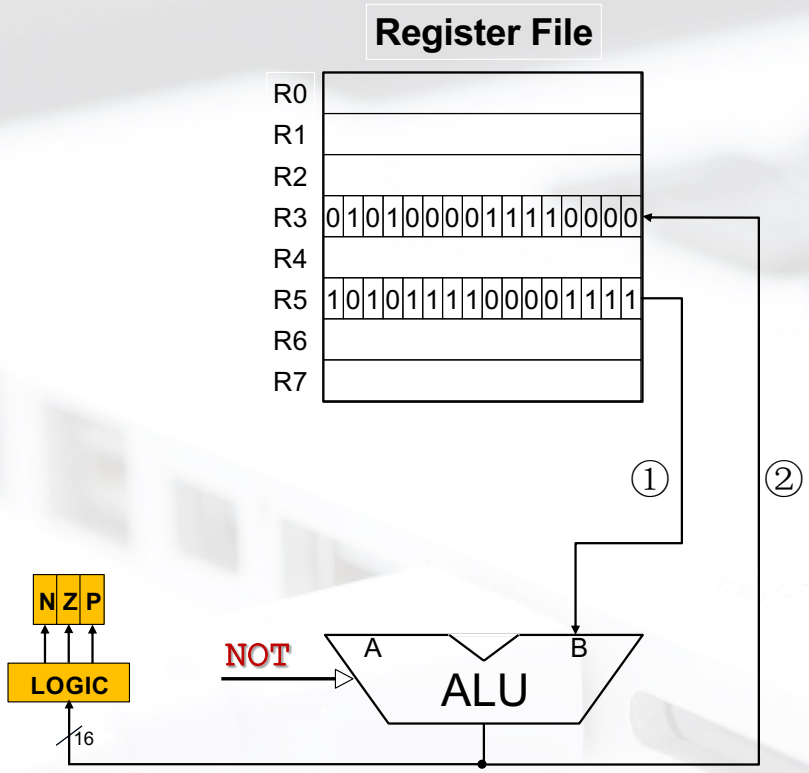


NOT (Register)

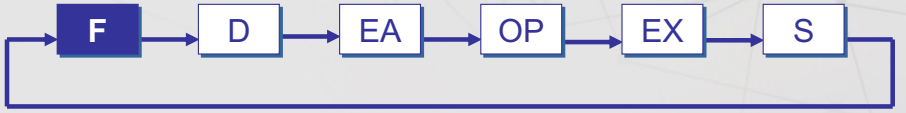


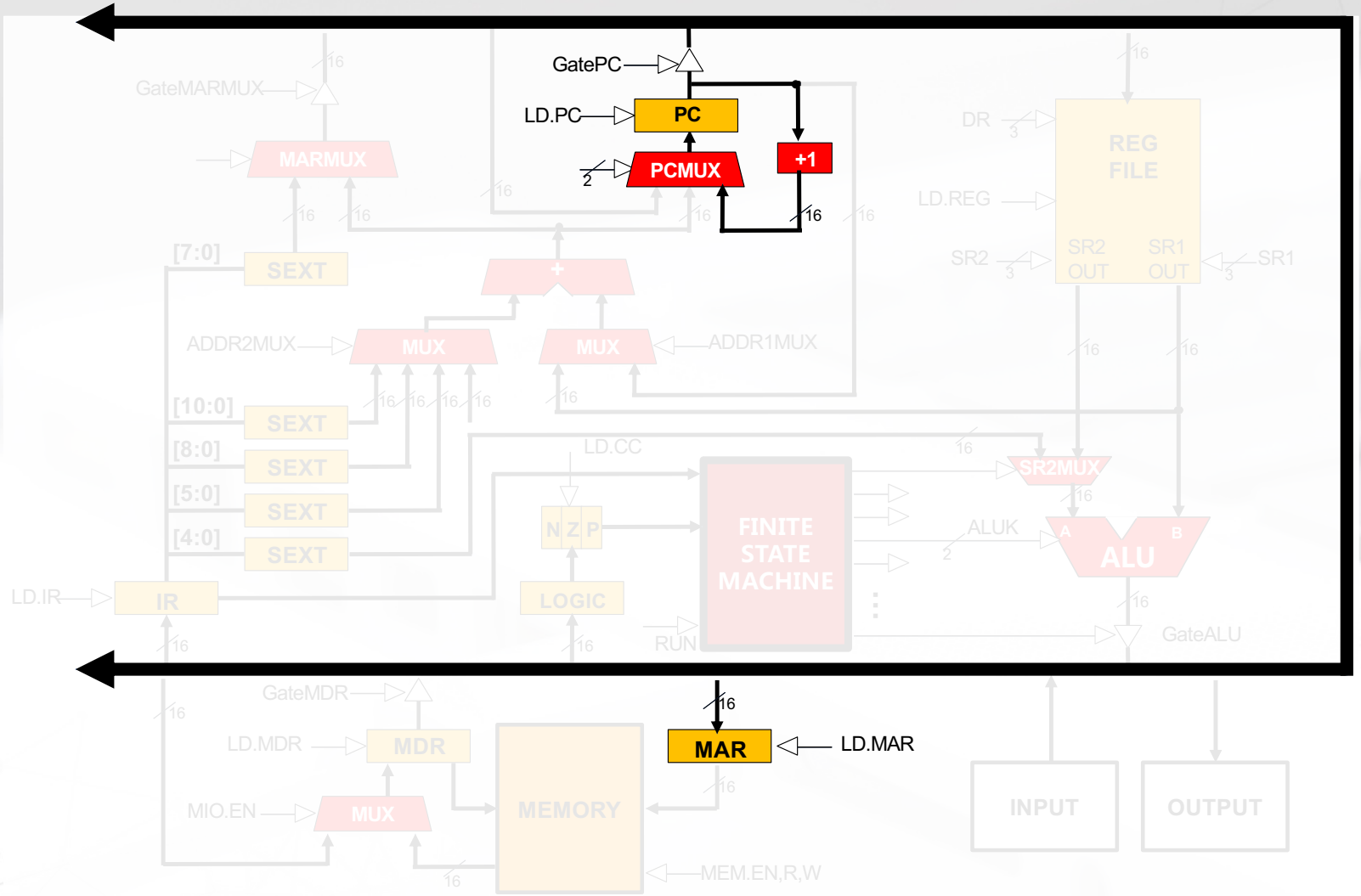
Note: Src and Dst could be the same register.

NOT (Register): NOT R3, R5

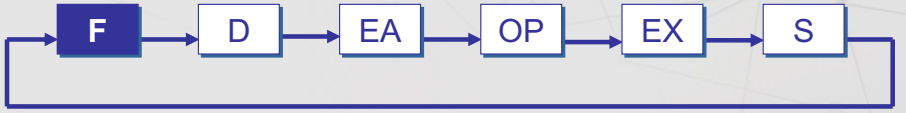


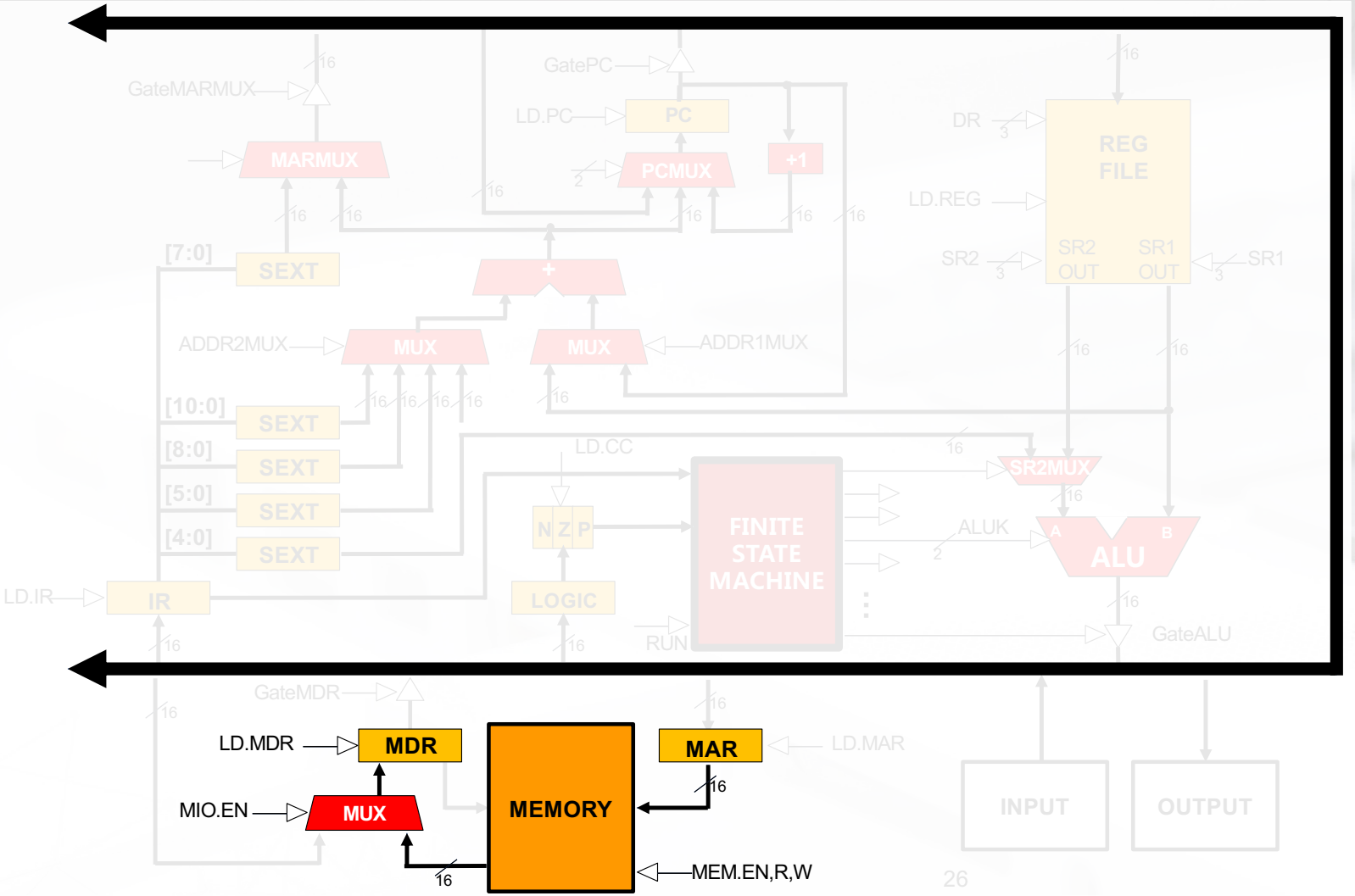
NOT (Register)



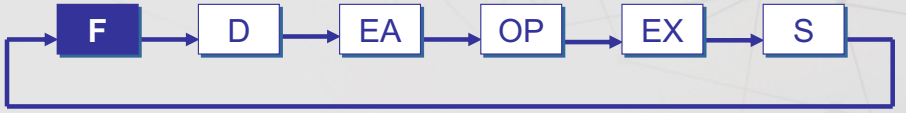


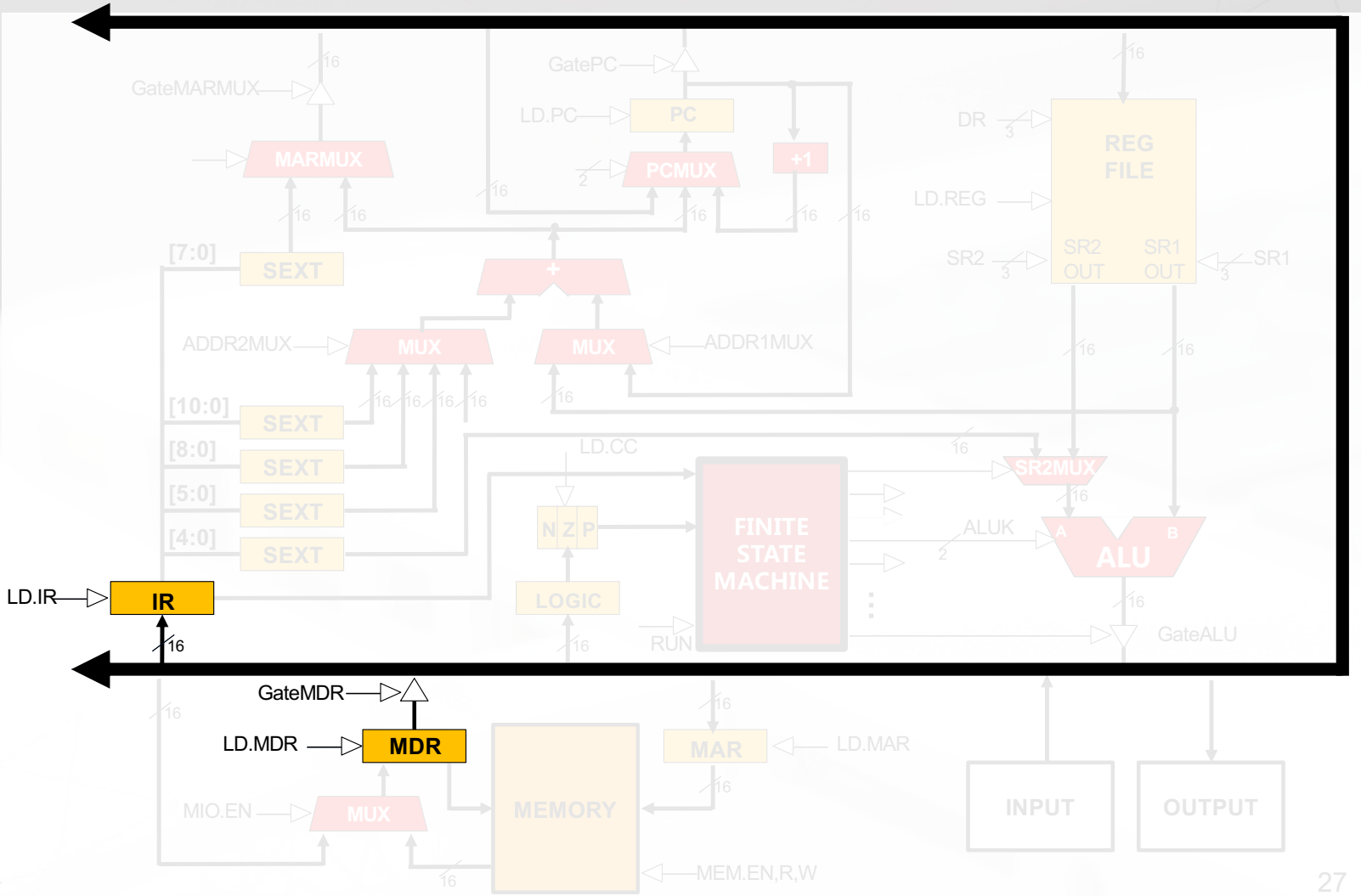
NOT (Register)



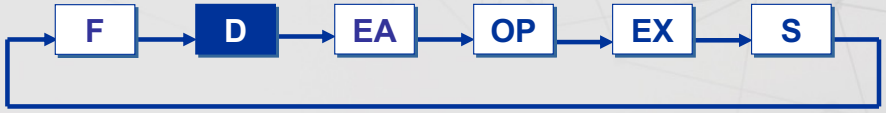


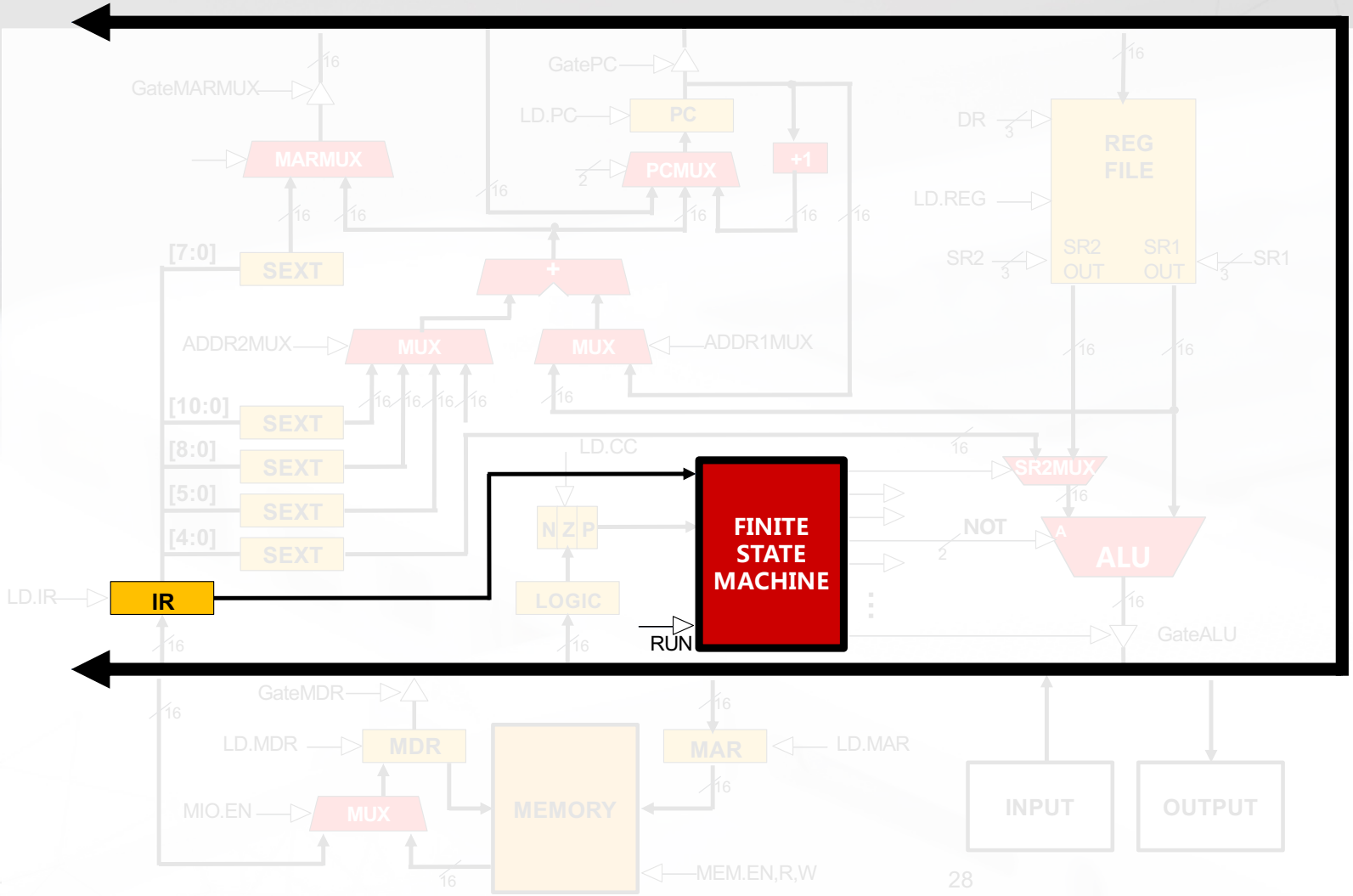
NOT (Register)



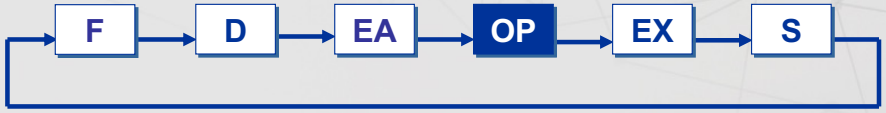


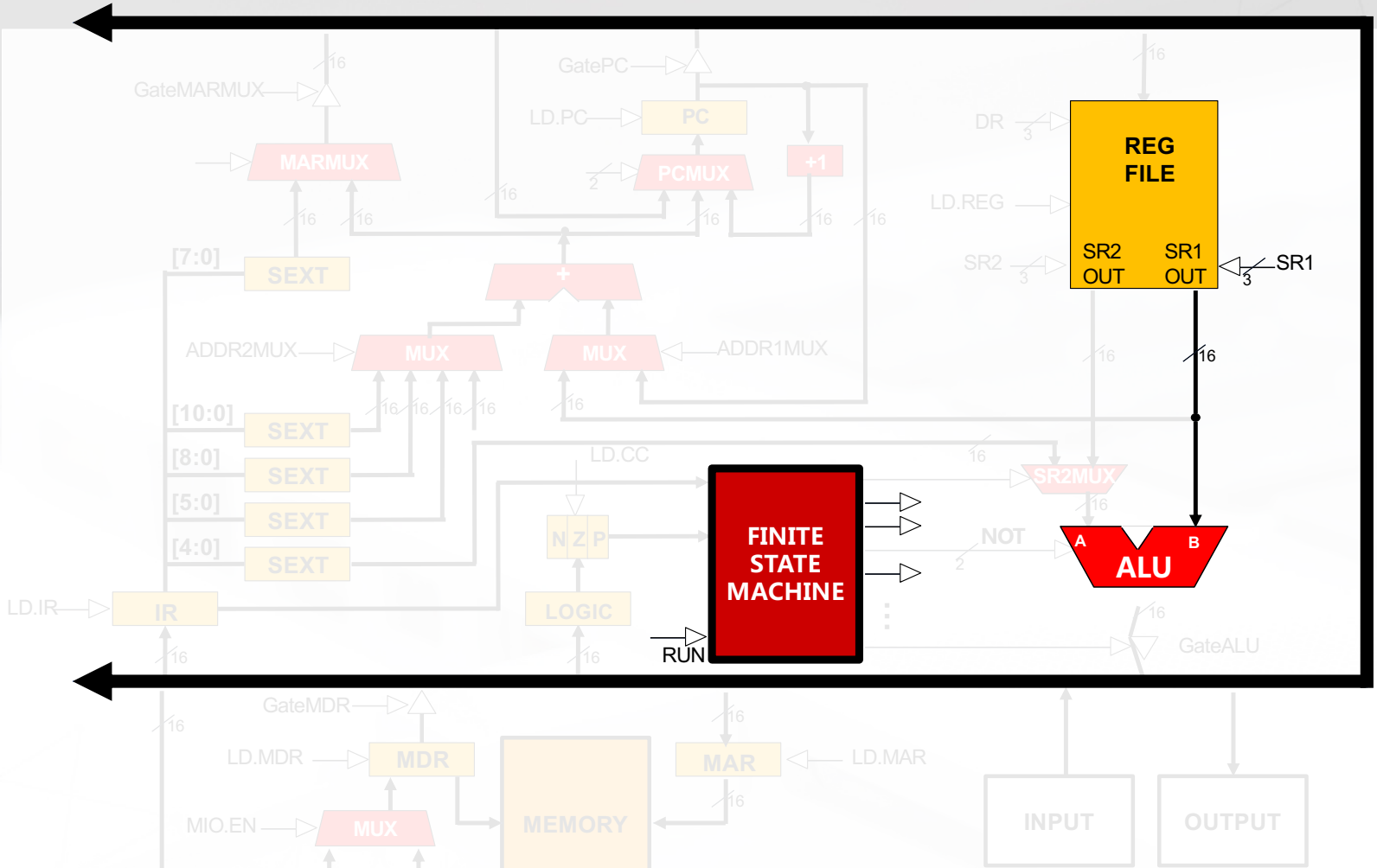
NOT (Register)



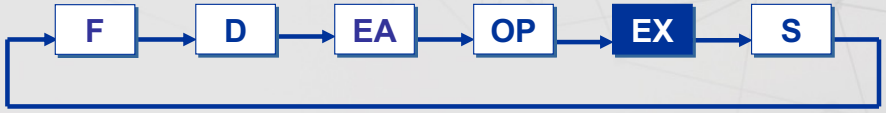


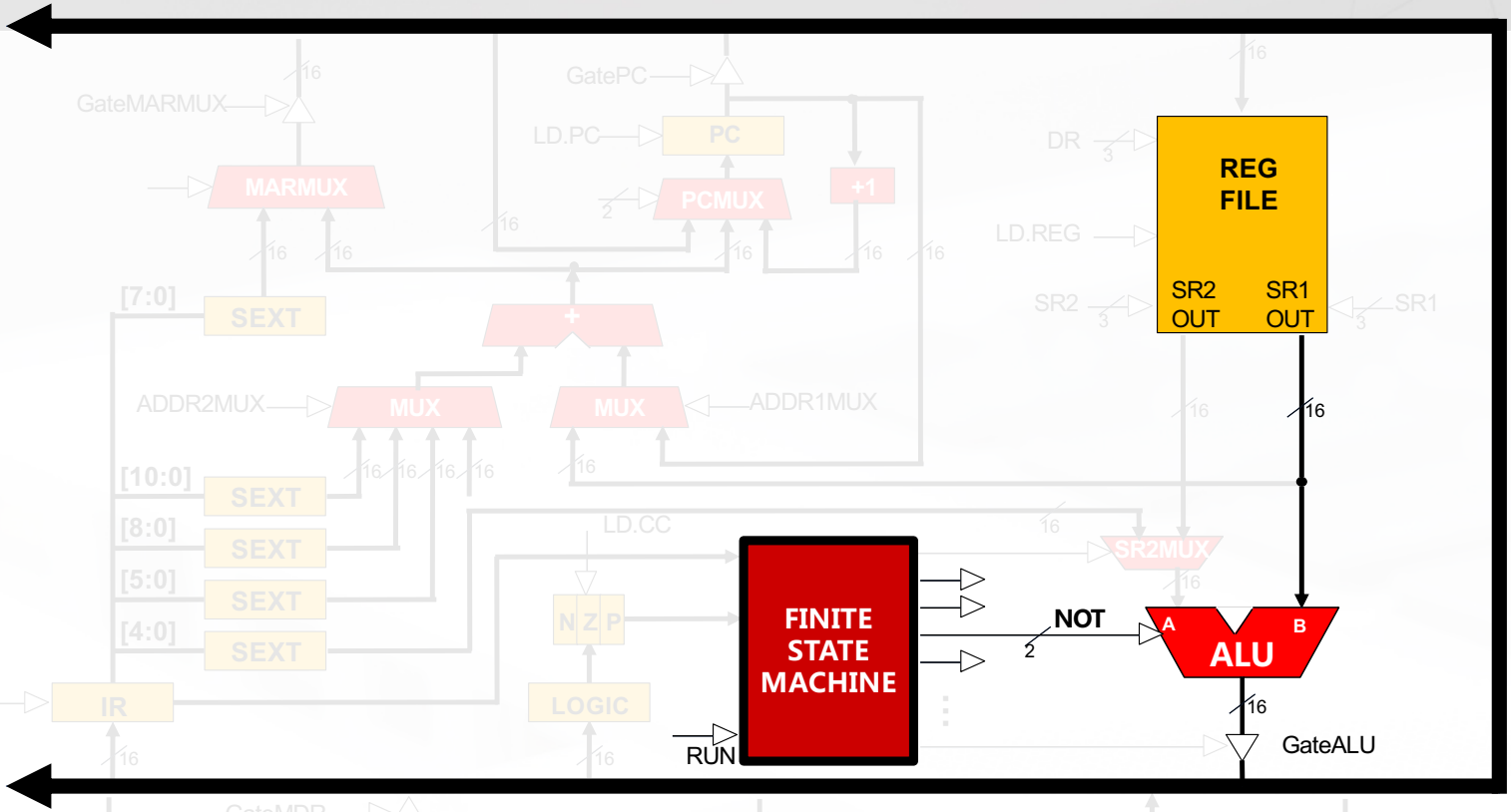
NOT (Register)



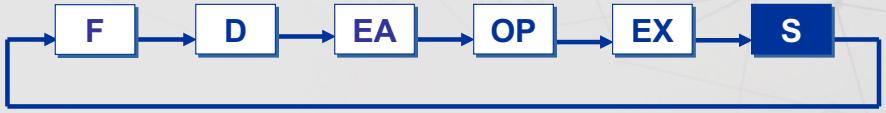


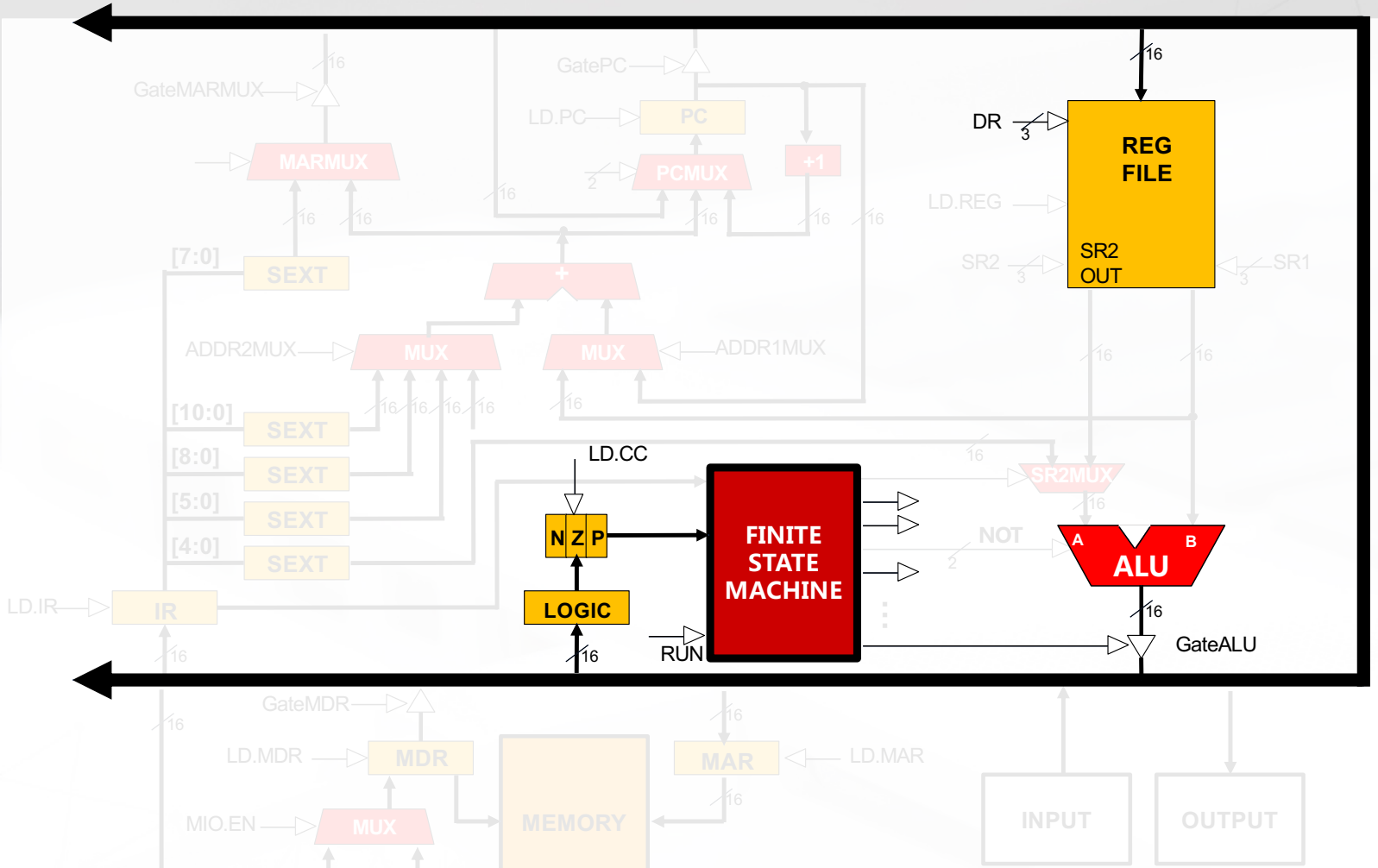
NOT (Register)





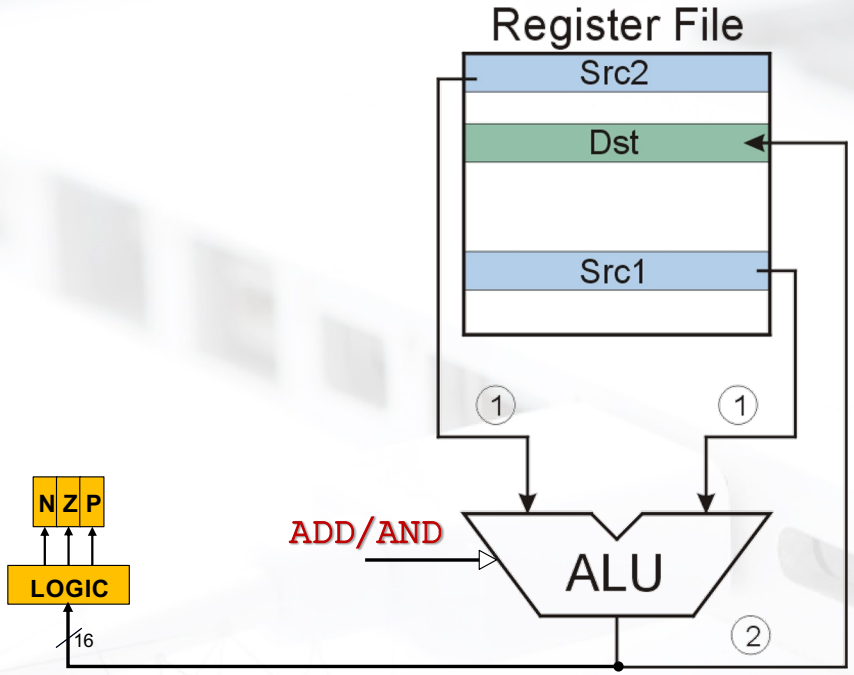
NOT (Register)



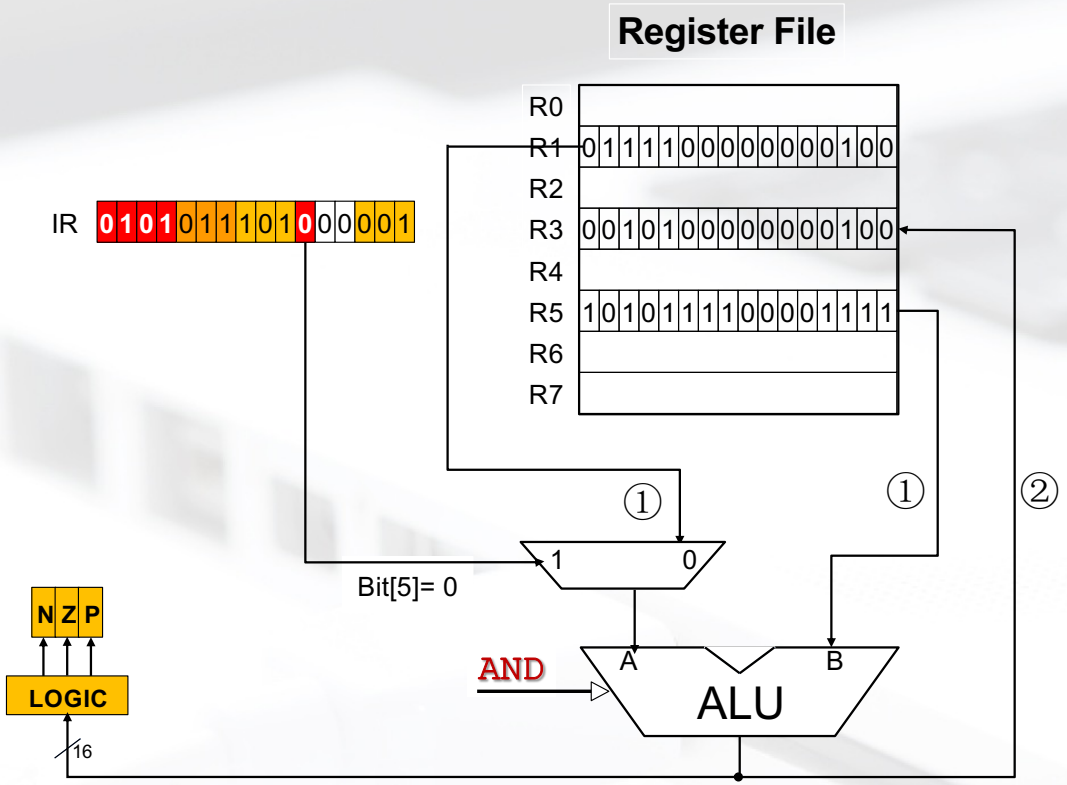


ADD/AND (Register)

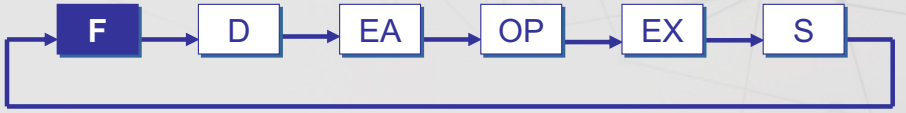
this zero means "register mode"

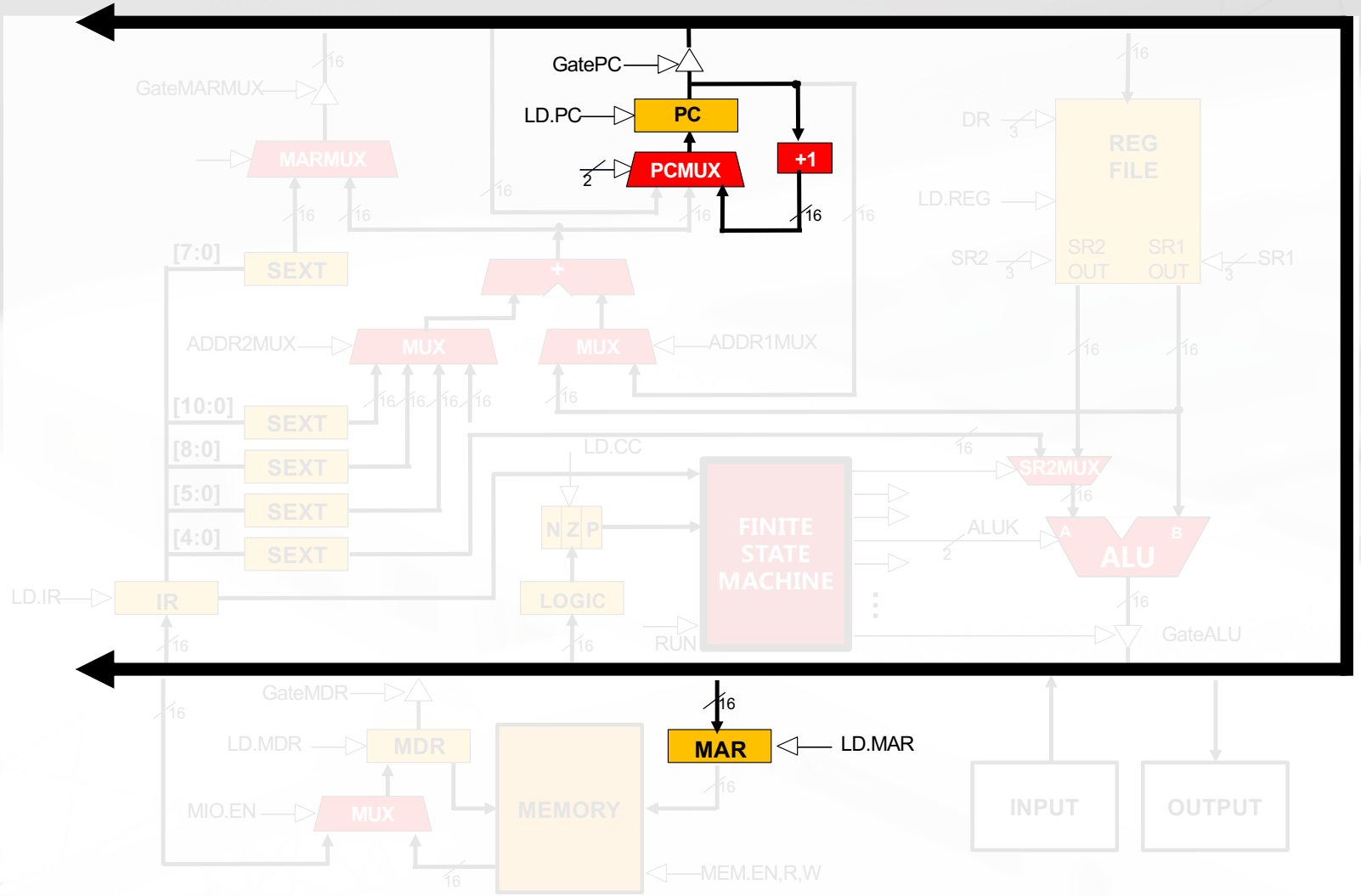


AND (Register): AND R3, R5, R1

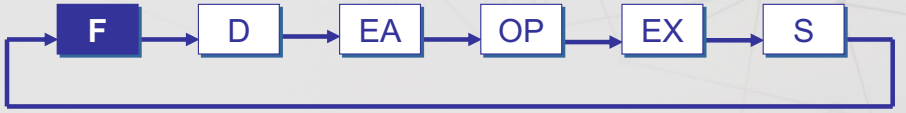


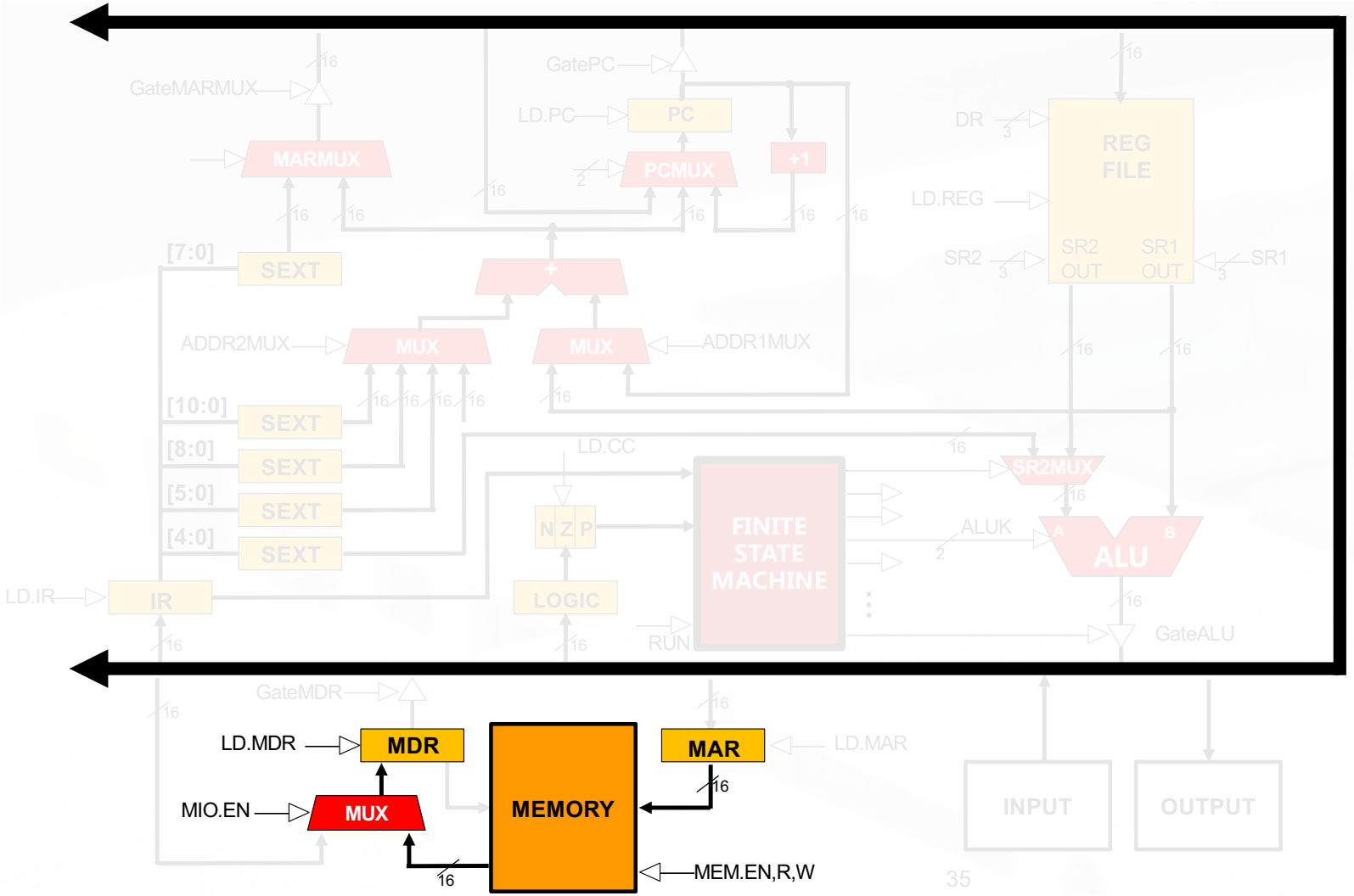
ADD/AND (Register)



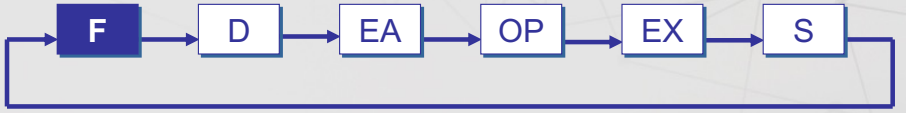


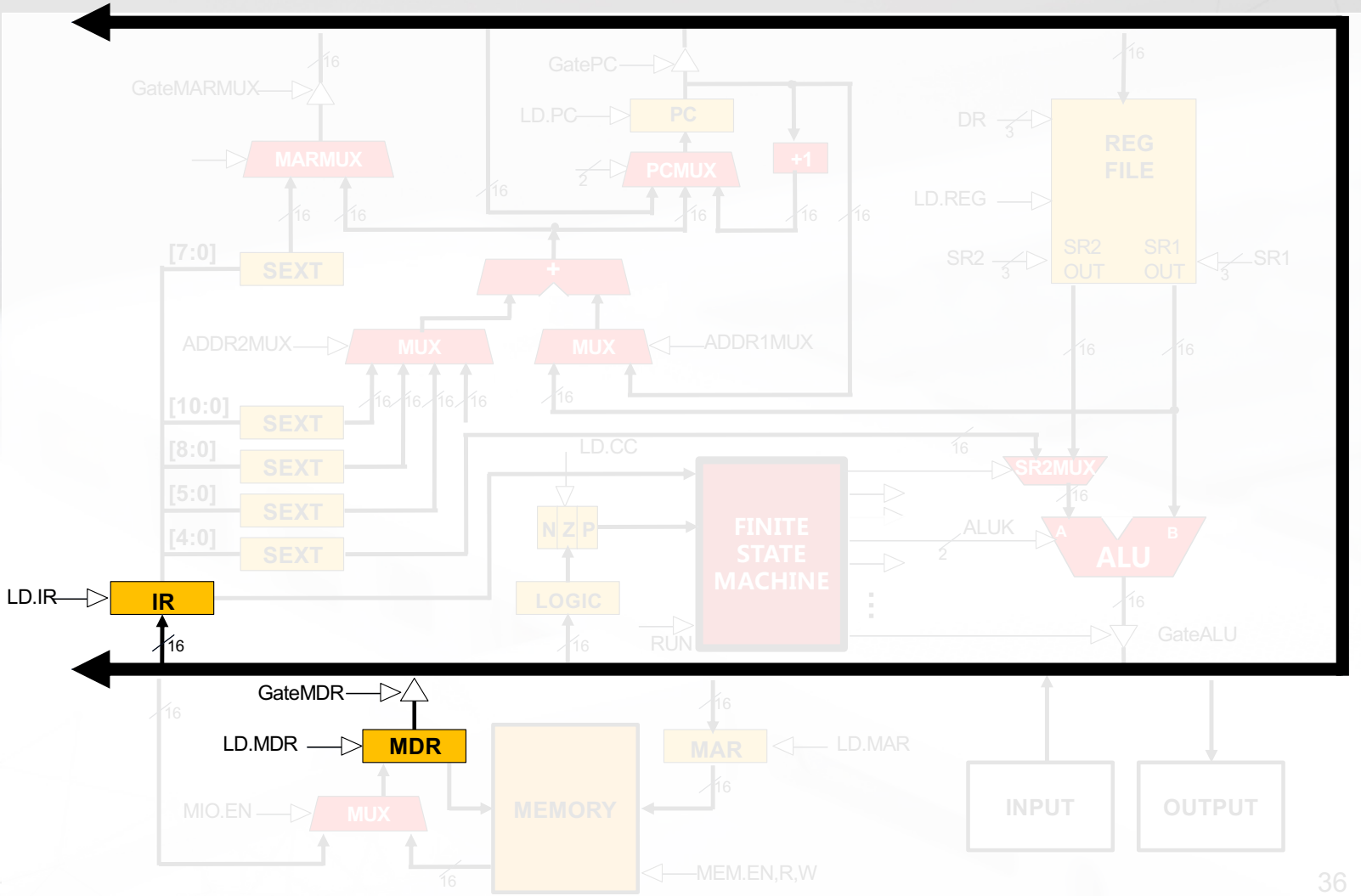
ADD/AND (Register)



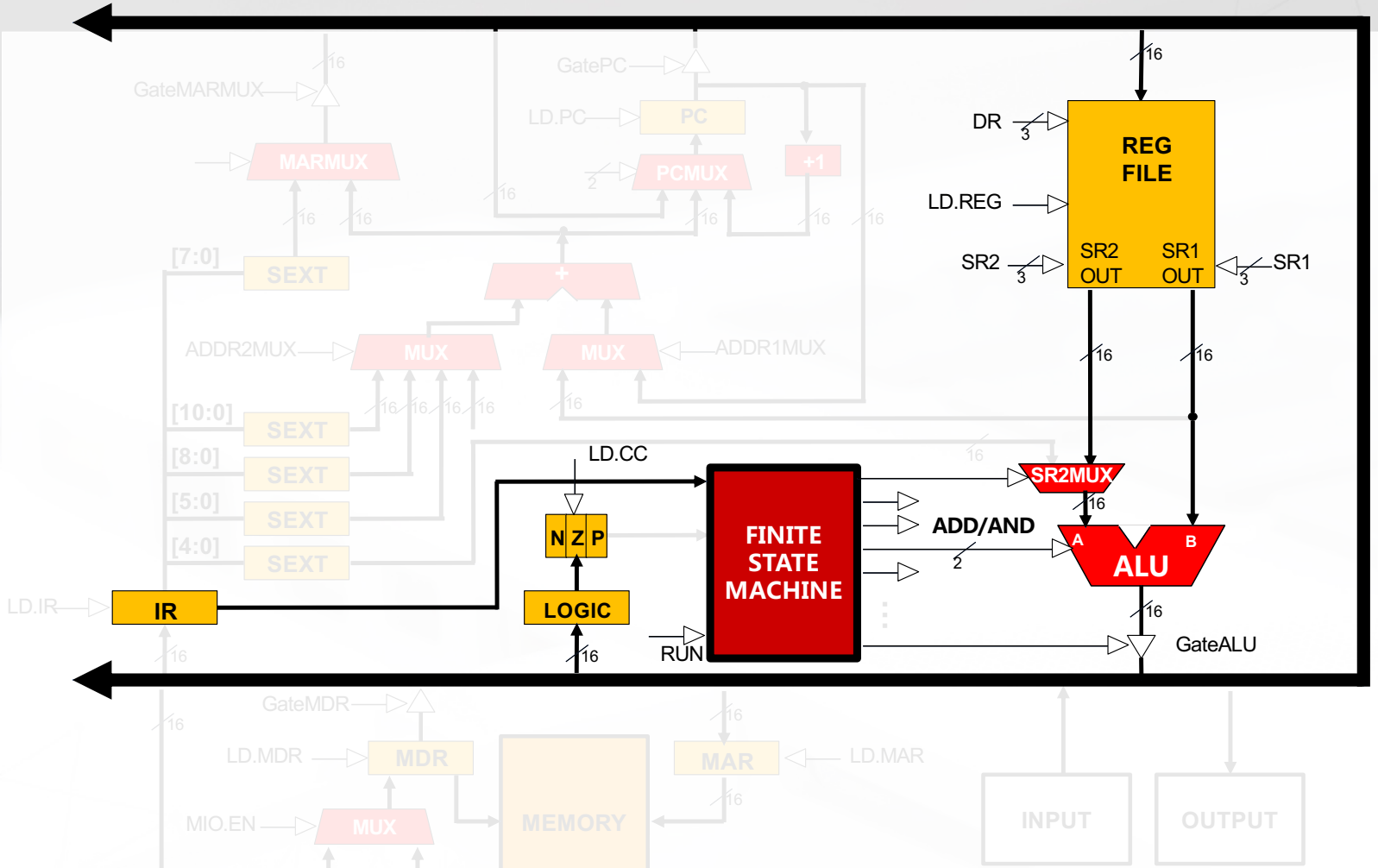


ADD/AND (Register)



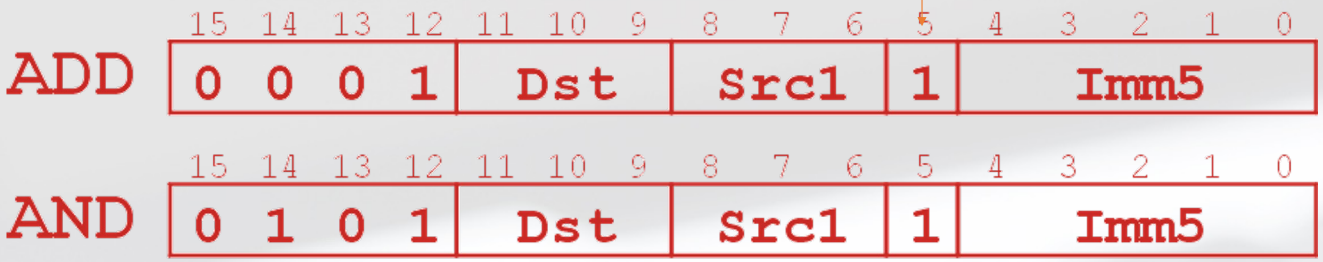


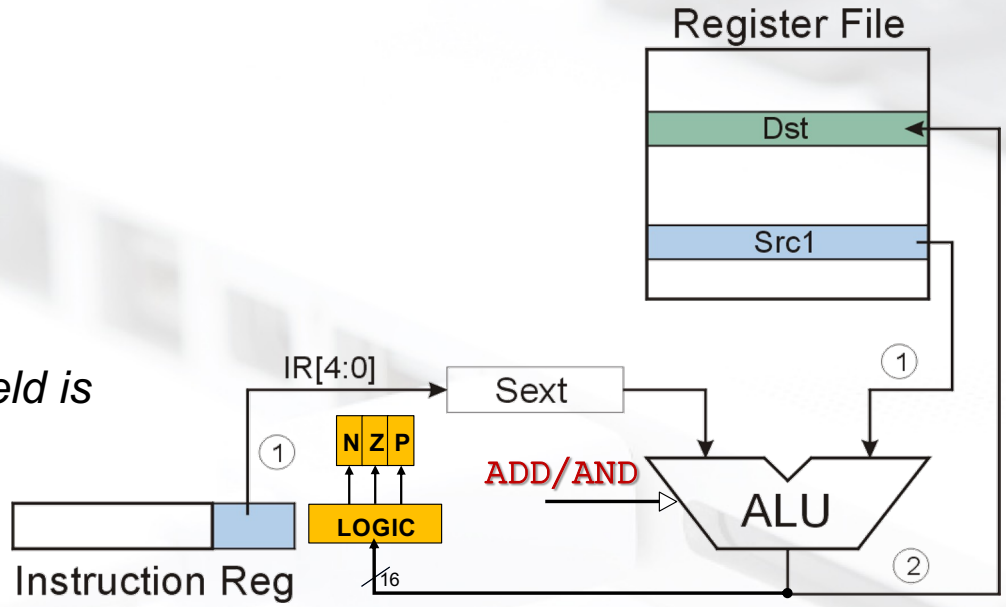
ADD/AND (Register)



ADD/AND (Immediate)

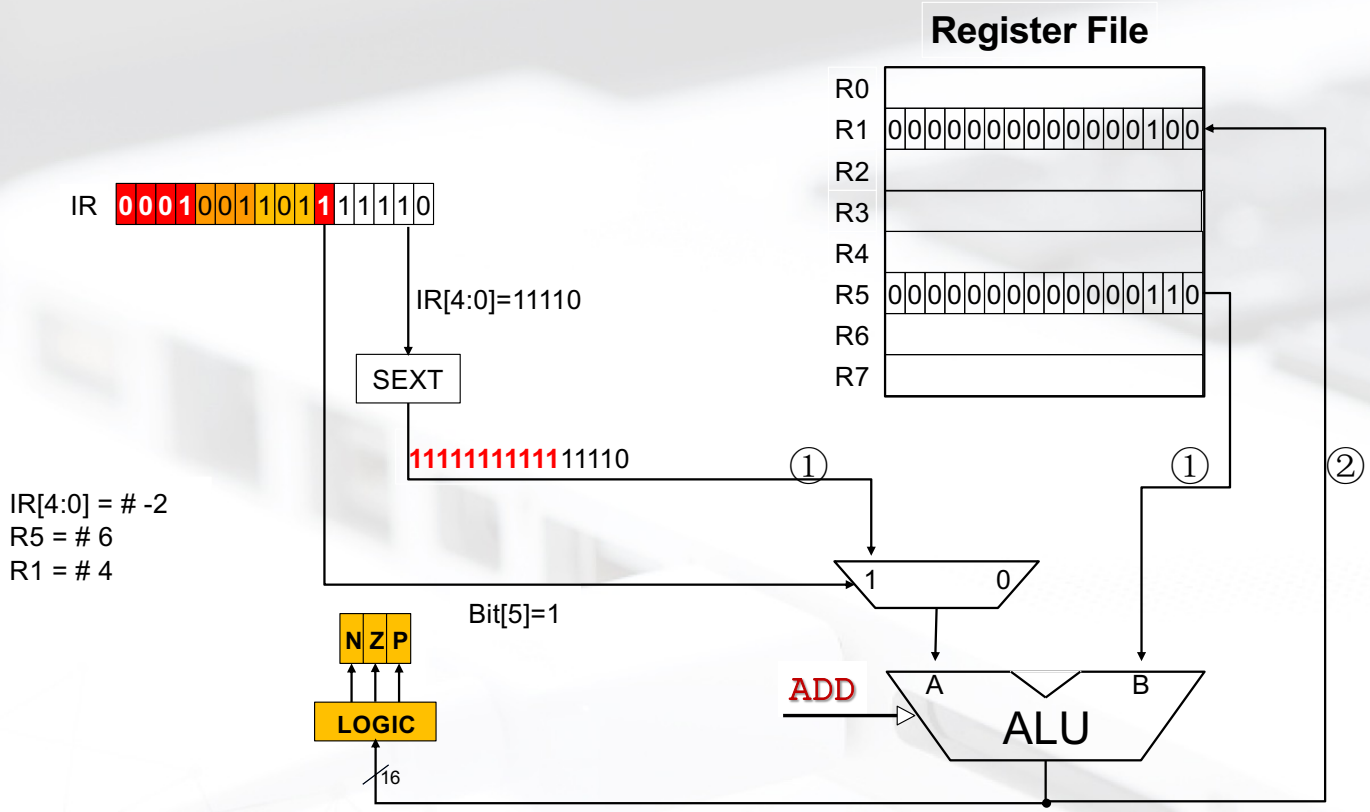
this one means "immediate mode"



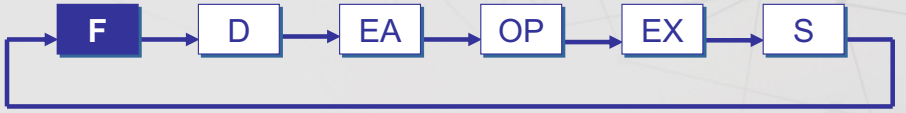


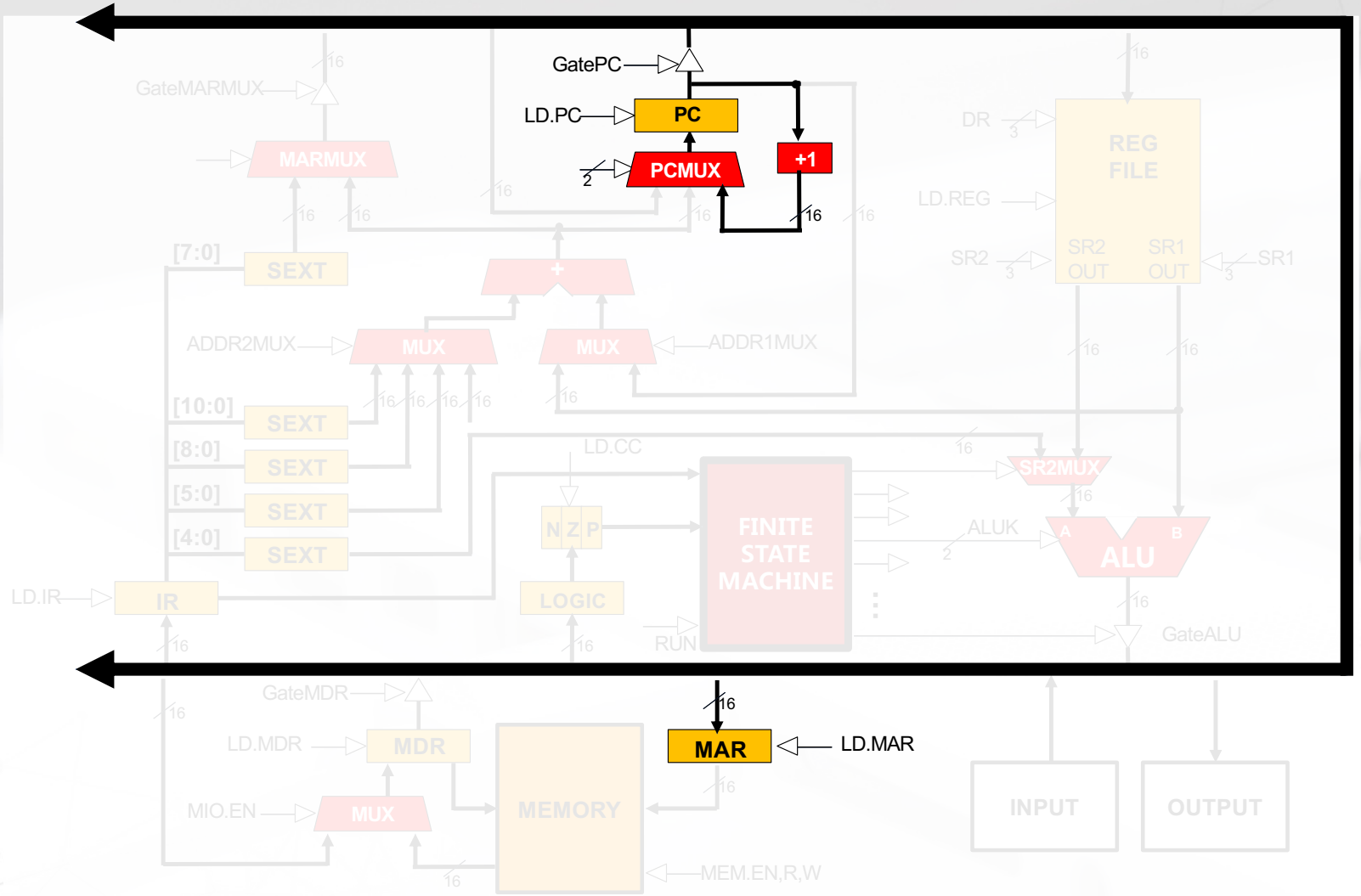
Note: Immediate field is ***sign-extended****.*

ADD (Immediate) ADD R1, R5, #-2

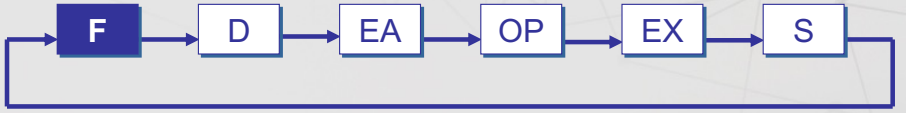


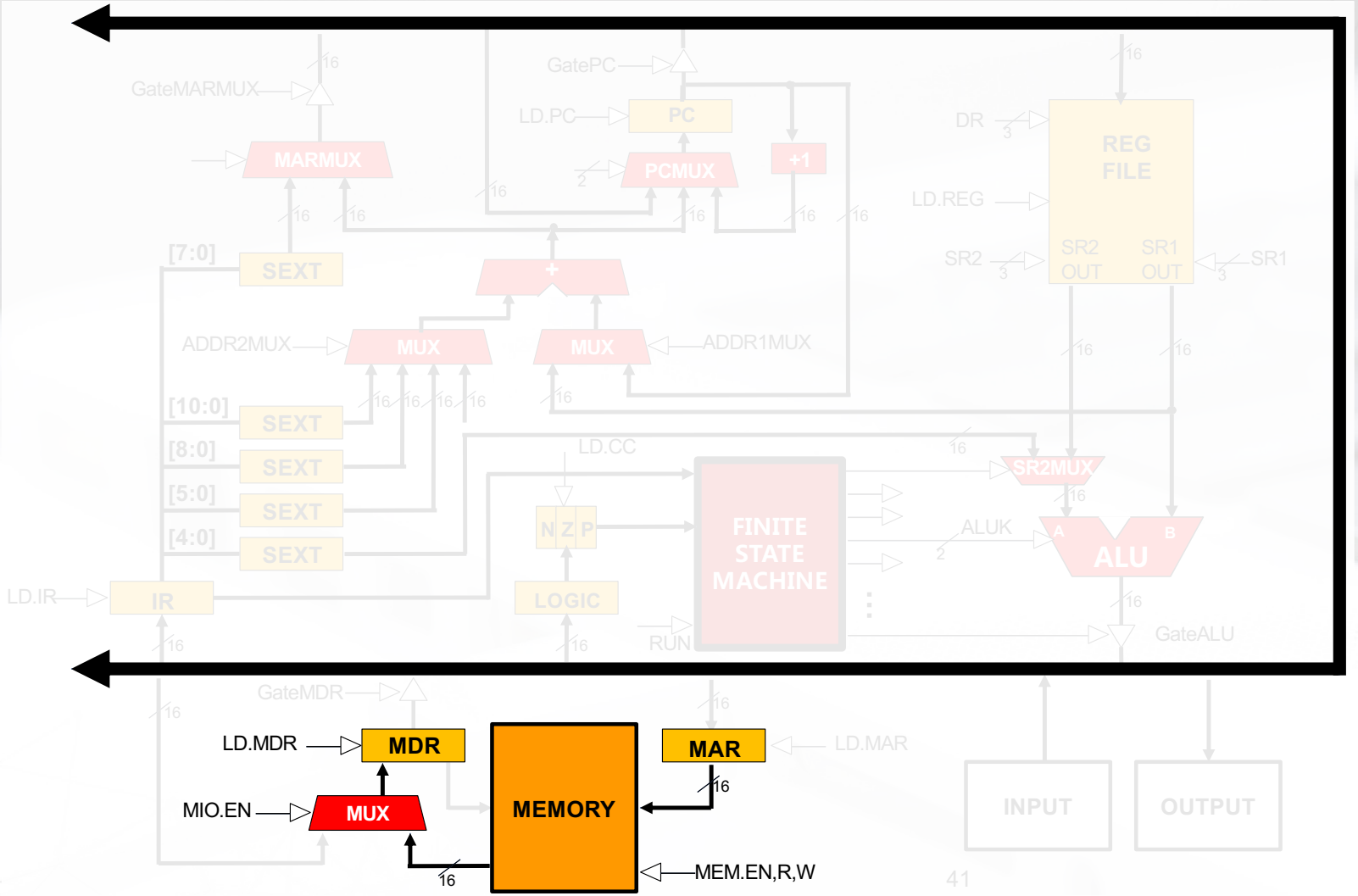
ADD/AND (Immediate)



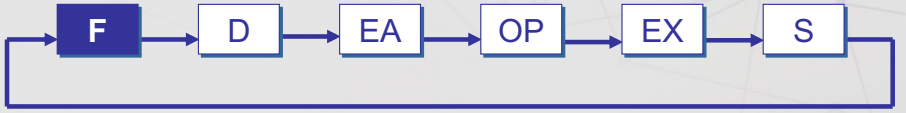


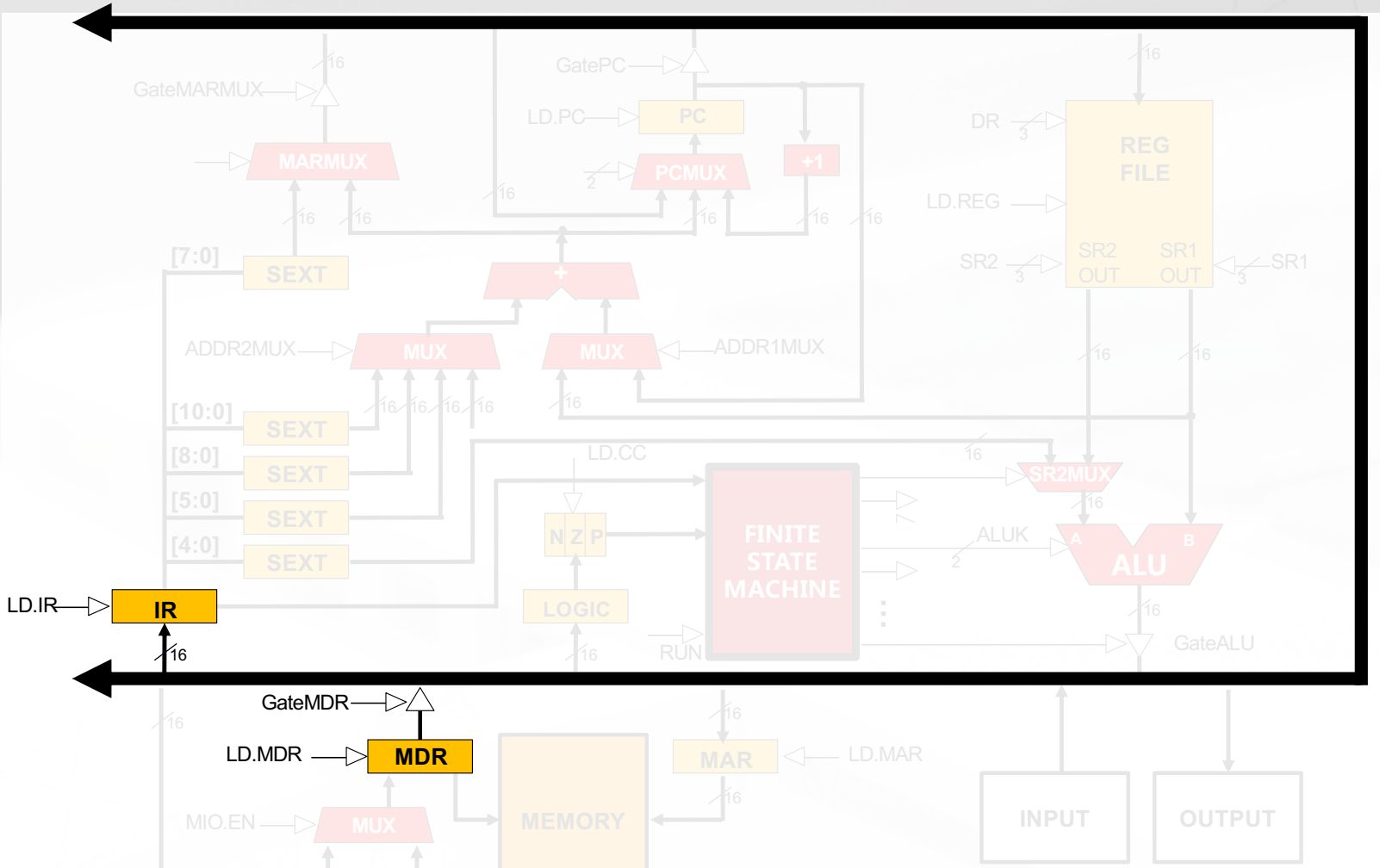
ADD/AND (Immediate)





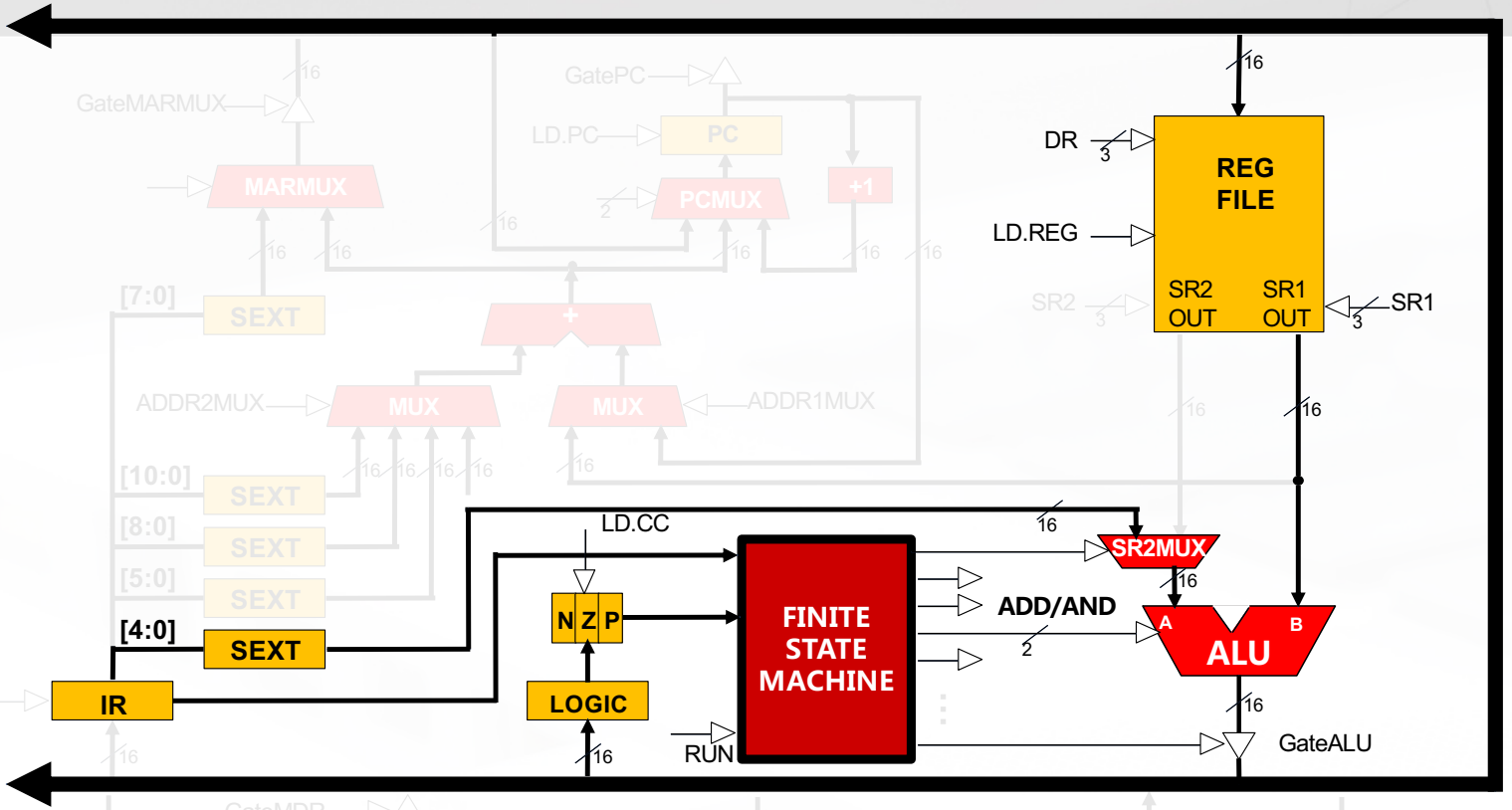
ADD/AND (Immediate)





42

ADD/AND (Immediate)



What does the following instruction do?

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

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

Using Operate Instructions

With only ADD, AND, NOT…

- How do we subtract?

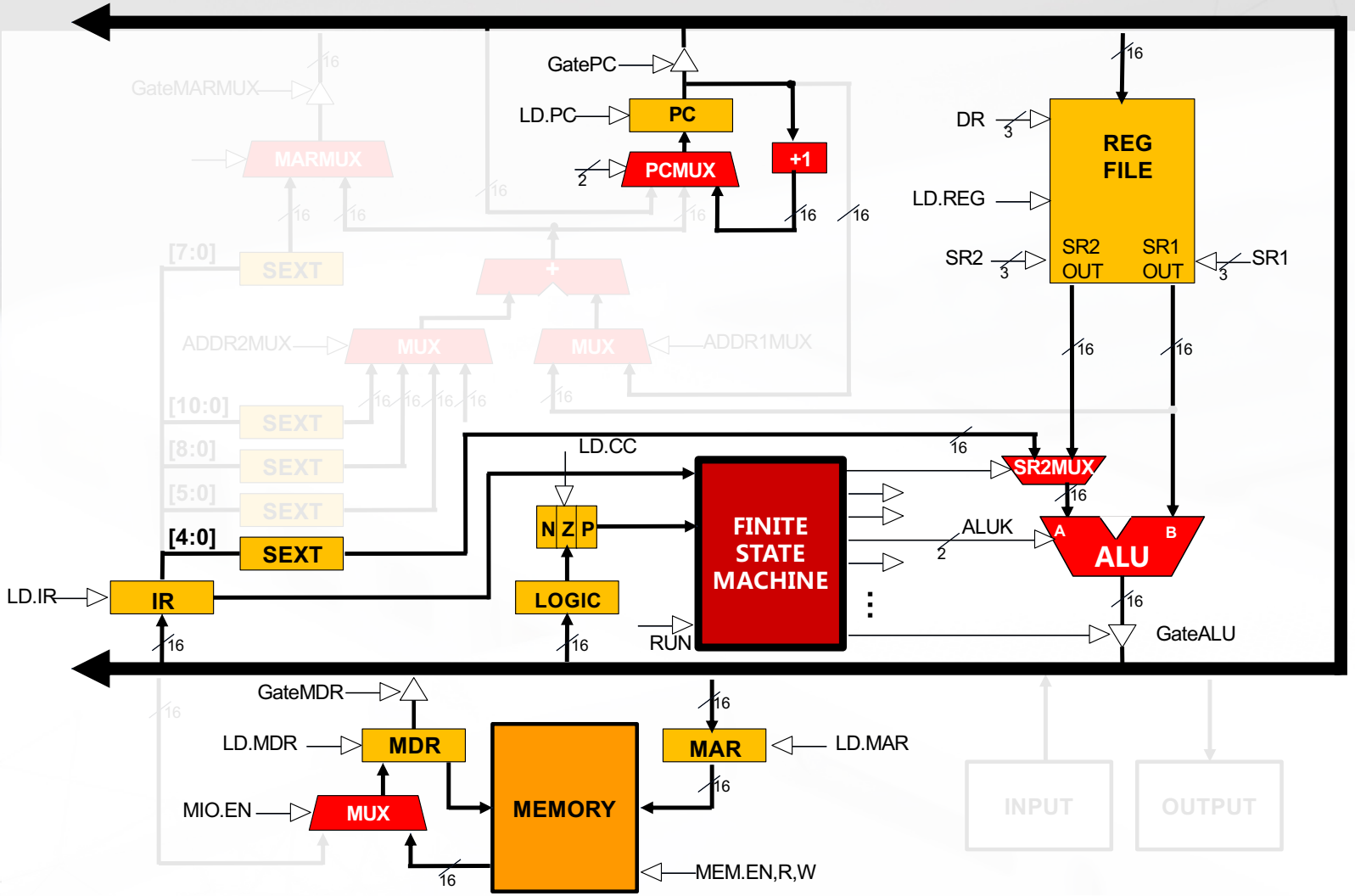
```
A - B= A +(-B)

R1 ← NOT(B)
R2 ← R1+1
R2 ← A + (-B)
```

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

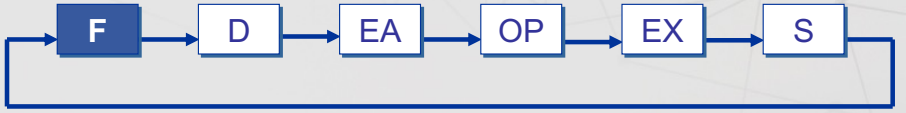
- How do we OR?
- How do we initialize a register to zero?
- How do we copy from one register to another?

LC-3 Data Path After Operate Instruction

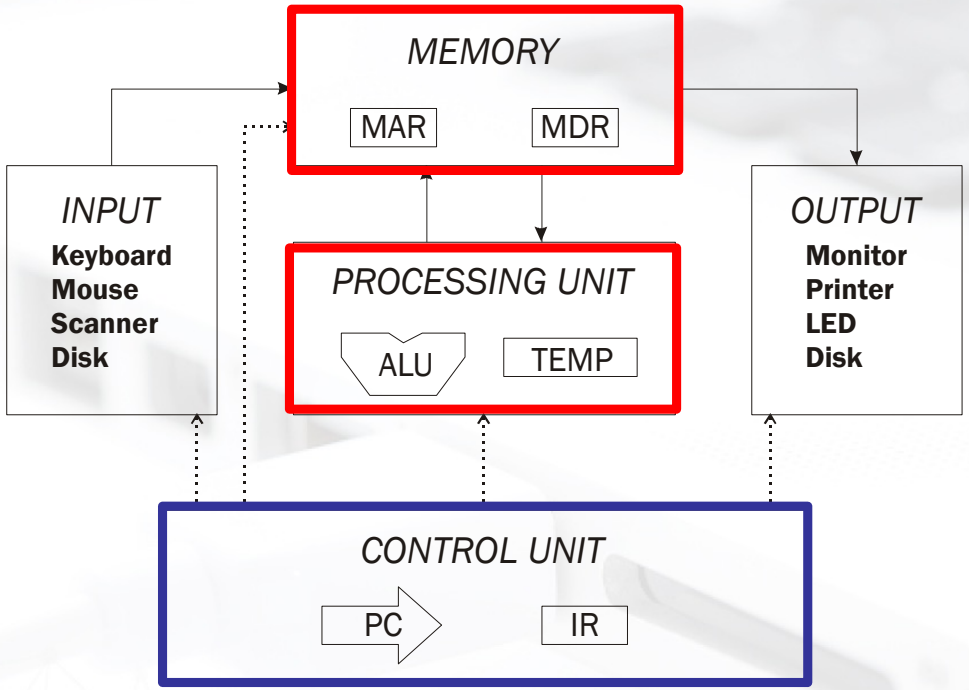


Outline

Today



- We are going to learn how to:
  - compute with values in registers
  - load data from memory to registers
  - store data from registers to memory



LC-3 ISA Data Movement Instructions

运算指令

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD	0	0	0	1	DR			SR1			0	0	0	SR2		
ADD	0	0	0	1	DR			SR1			1	Imm5				
AND	0	1	0	1	DR			SR1			0	0	0	SR2		
AND	0	1	0	1	DR			SR1			1	Imm5				
NOT	1	0	0	1	DR			SR1			1	1	1	1	1	1
Reserved	1	1	0	1												

控制指令

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BR	0	0	0	0	n	z	p	PCoffset9								
JSR	0	1	0	0	1	PCoffset11										
JSRR	0	1	0	0	0	0	0	BaseR			0	0	0	0	0	0
RTI	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
JMP	1	1	0	0	0	0	0	BaseR			0	0	0	0	0	0
RET	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0
TRAP	1	1	1	1	0	0	0	0	TrapVector8							

移动数据指令

取数指令

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LD	0	0	1	0	DR			PCoffset9								
LDR	0	1	1	0	DR			BaseR			PCoffset6					
LDI	1	0	1	0	DR			PCoffset9								
LEA	1	1	1	0	DR			PCoffset9								

存数指令

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ST	0	0	1	1	SR			PCoffset9								
STR	0	1	1	1	SR			BaseR			PCoffset6					
STI	1	0	1	1	SR			PCoffset9								

Data Movement Instructions

取数指令

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LD	0	0	1	0	DR			PCoffset9								
LDR	0	1	1	0	DR			BaseR			PCoffset6					
LDI	1	0	1	0	DR			PCoffset9								
LEA	1	1	1	0	DR			PCoffset9								

存数指令

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ST	0	0	1	1	SR			PCoffset9								
STR	0	1	1	1	SR			BaseR			PCoffset6					
STI	1	0	1	1	SR			PCoffset9								

Data Movement Instructions

Load -- read data from memory to register

- **LD:** PC-relative mode
- **LDR:** base+offset mode
- **LDI:** indirect mode

Store -- write data from register to memory

- **ST:** PC-relative mode
- **STR:** base+offset mode
- **STI:** indirect mode

Load effective address -- compute address, save in register

- **LEA:** immediate mode
  - *does not access memory*

PC-Relative Addressing Mode

Want to specify address directly in the instruction

- But an address is 16 bits, and so is an instruction!
- After subtracting 4 bits for opcode and 3 bits for register, we have 9 bits available for address.

Solution:

- Use the 9 bits as a *signed offset* from the current PC.

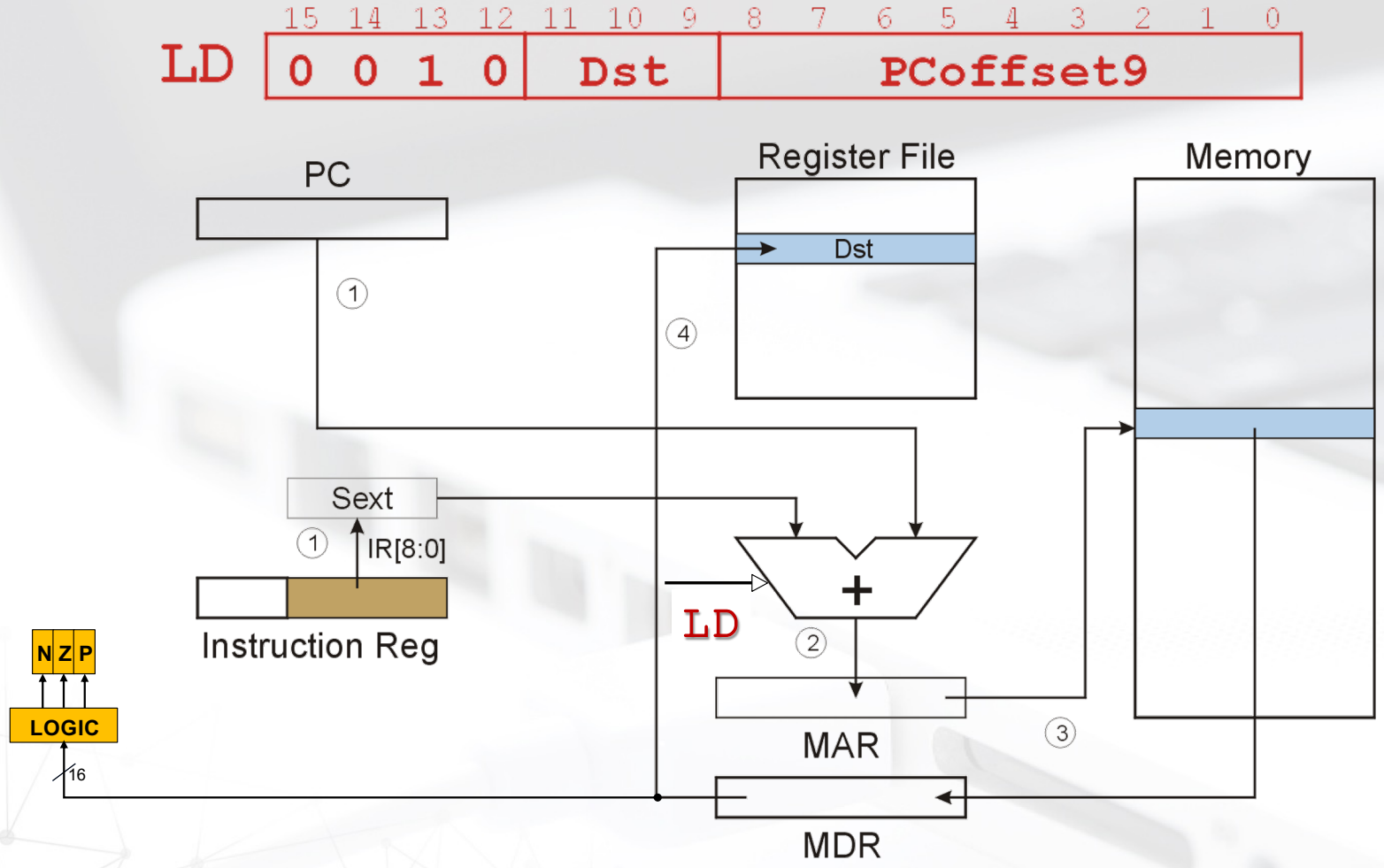
9 bits: $-256 \leq \text{offset} \leq +255$

Can form any address X, such that: $PC - 256 \leq X \leq PC + 255$

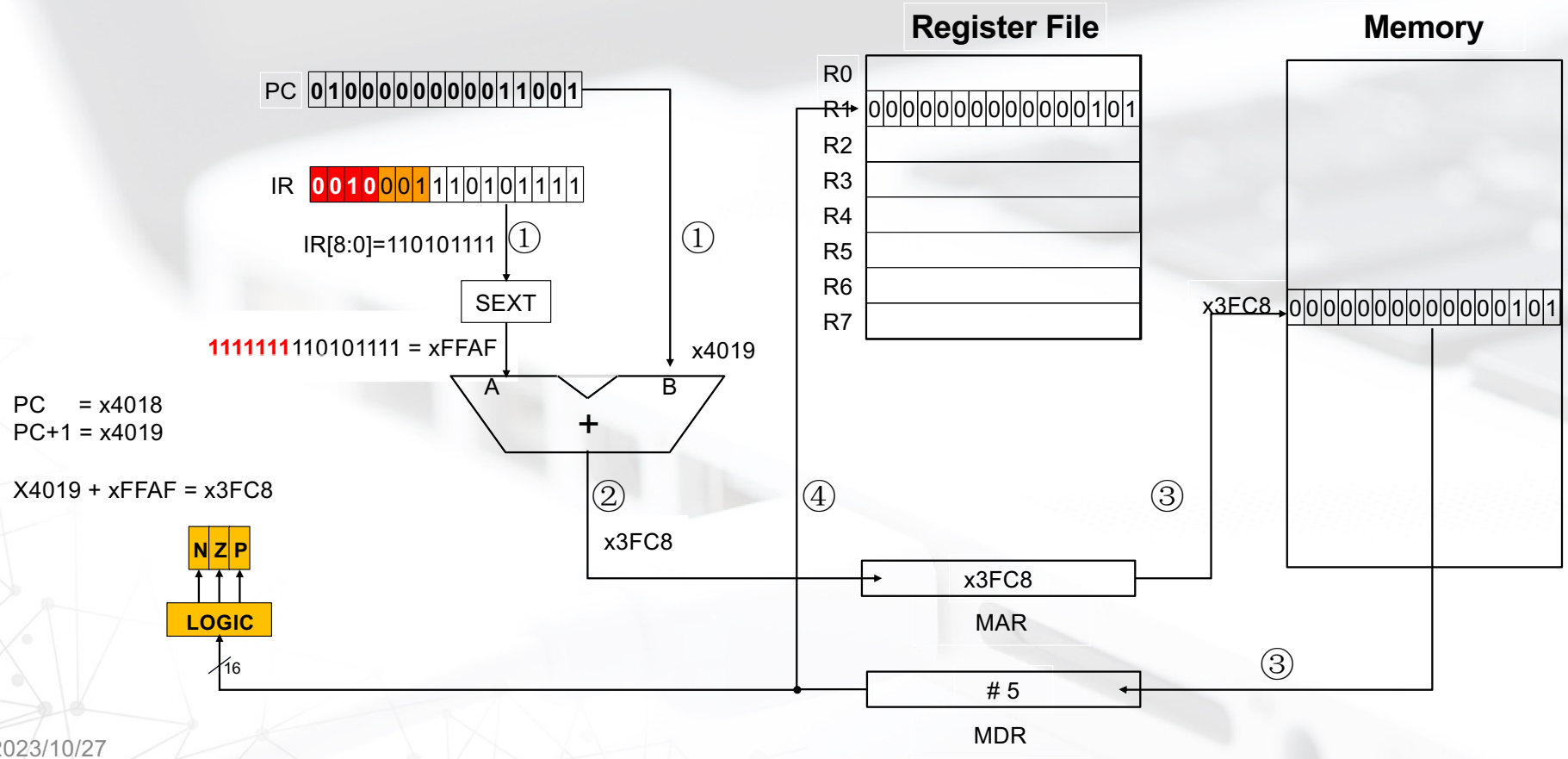
Remember that PC is incremented as part of the FETCH phase;

This is done before the EVALUATE ADDRESS stage.

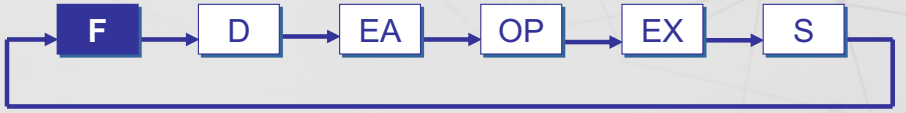
LD (PC-Relative) LD DR, PCoffset9

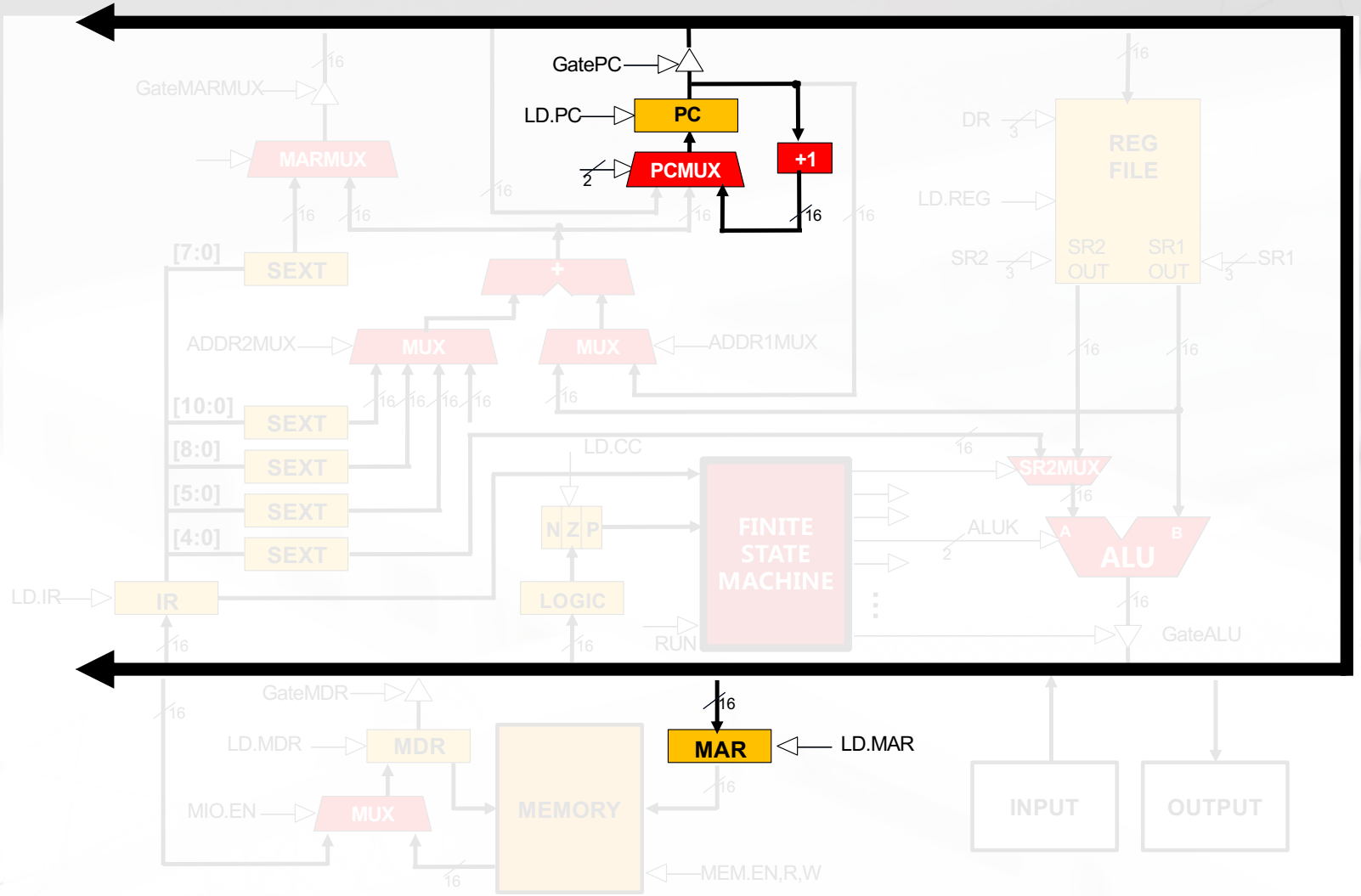


LD (PC-Relative) : LD R1, x1AF

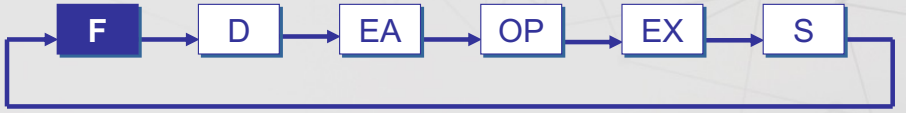


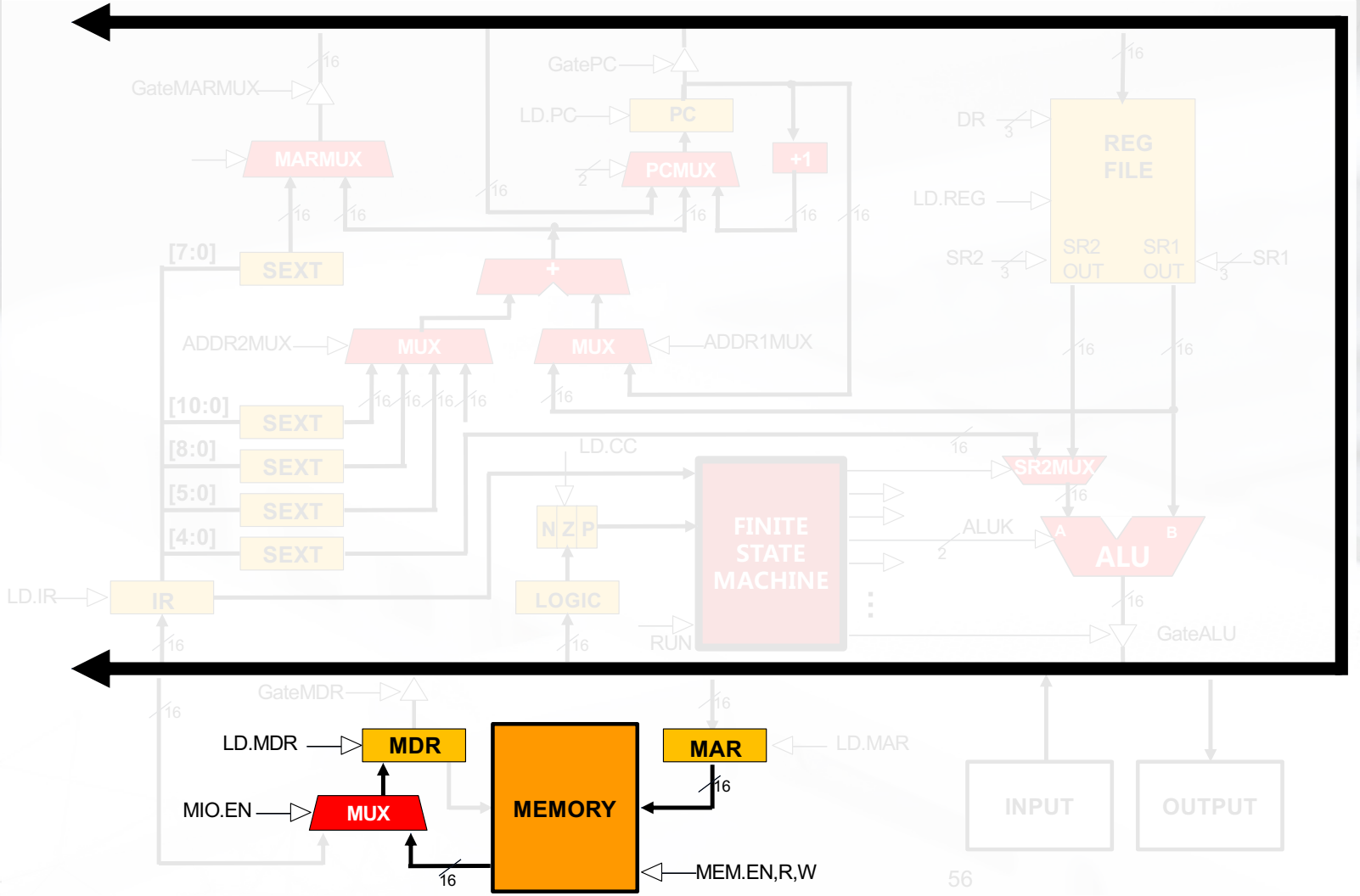
LD (PC-Relative)



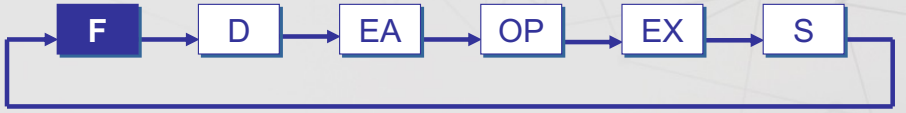


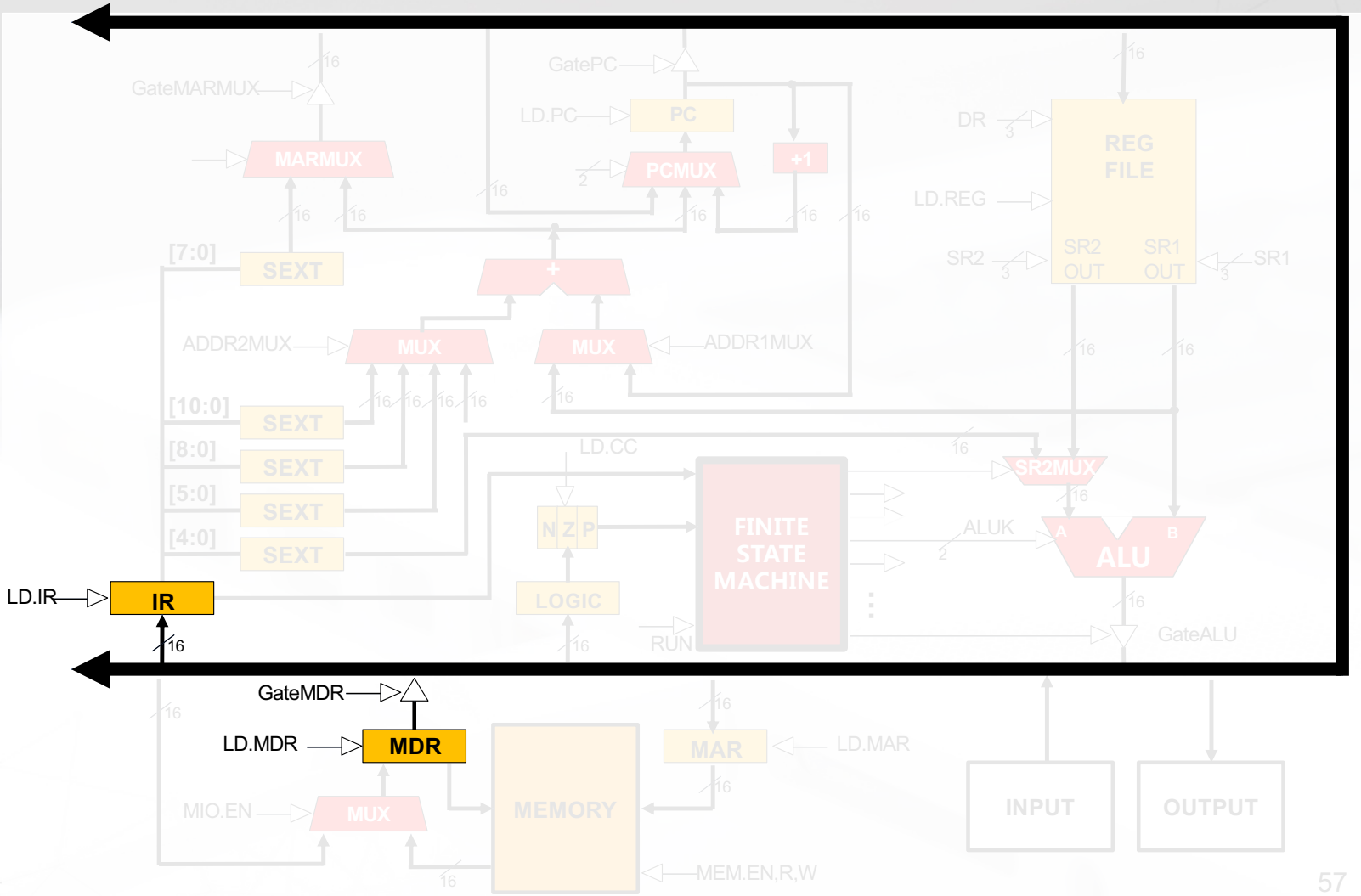
LD (PC-Relative)



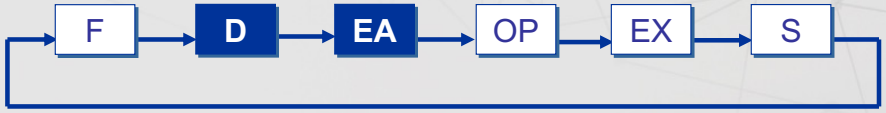


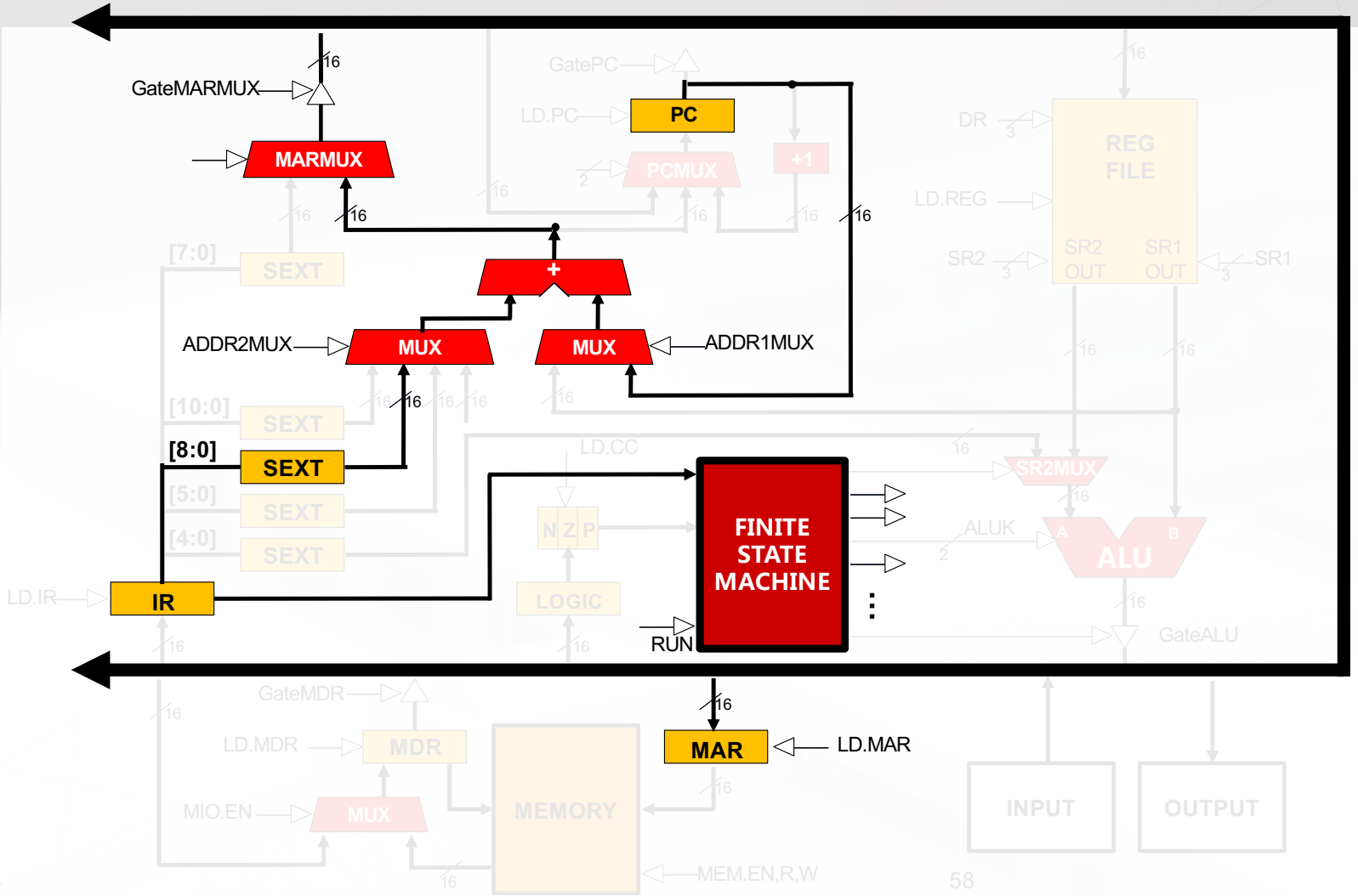
LD (PC-Relative)



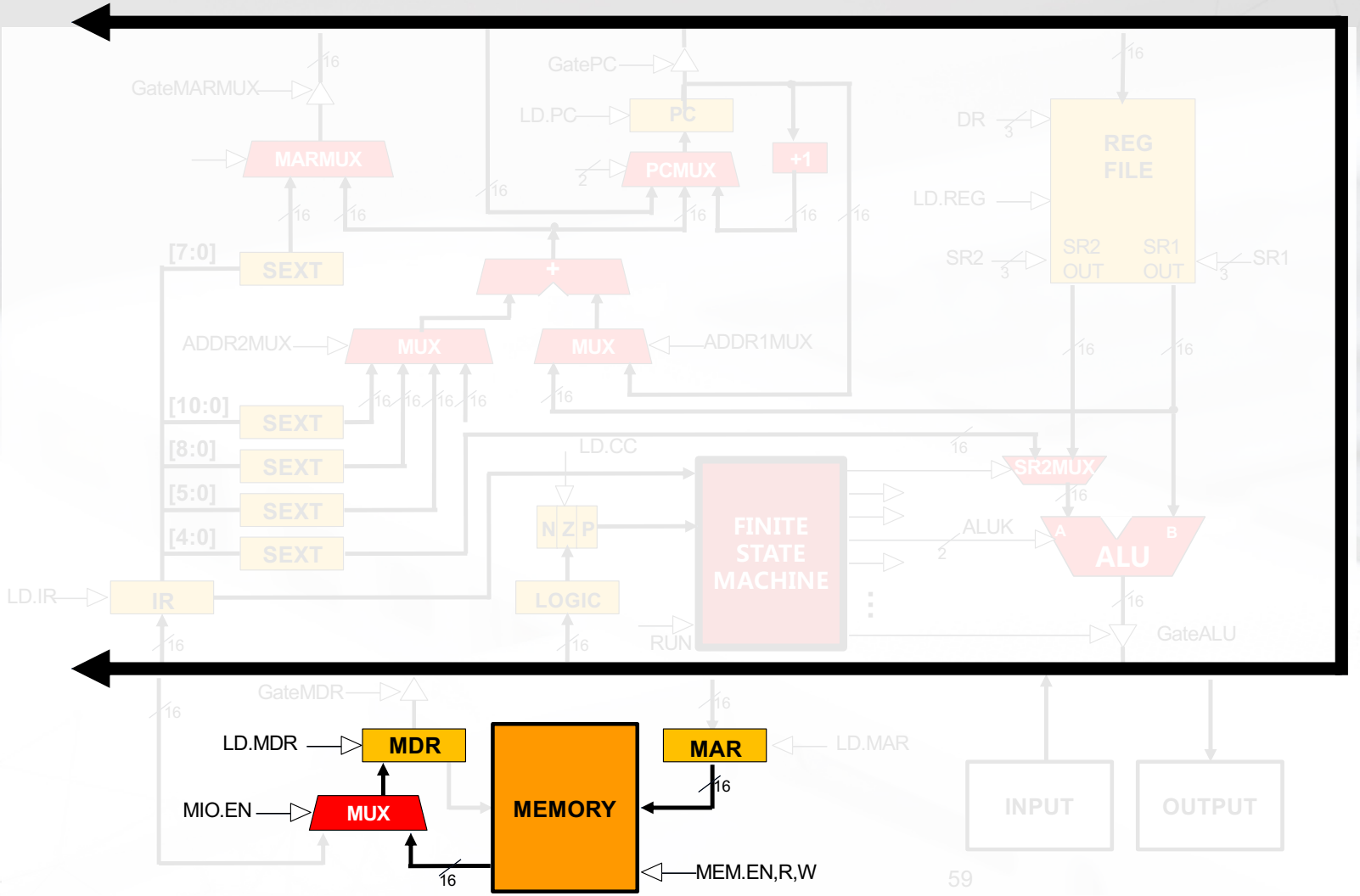


LD (PC-Relative)

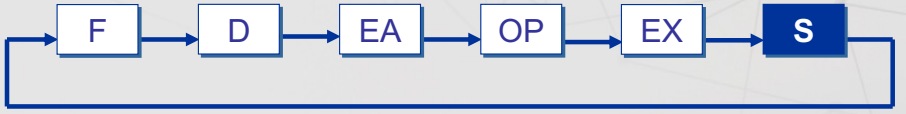


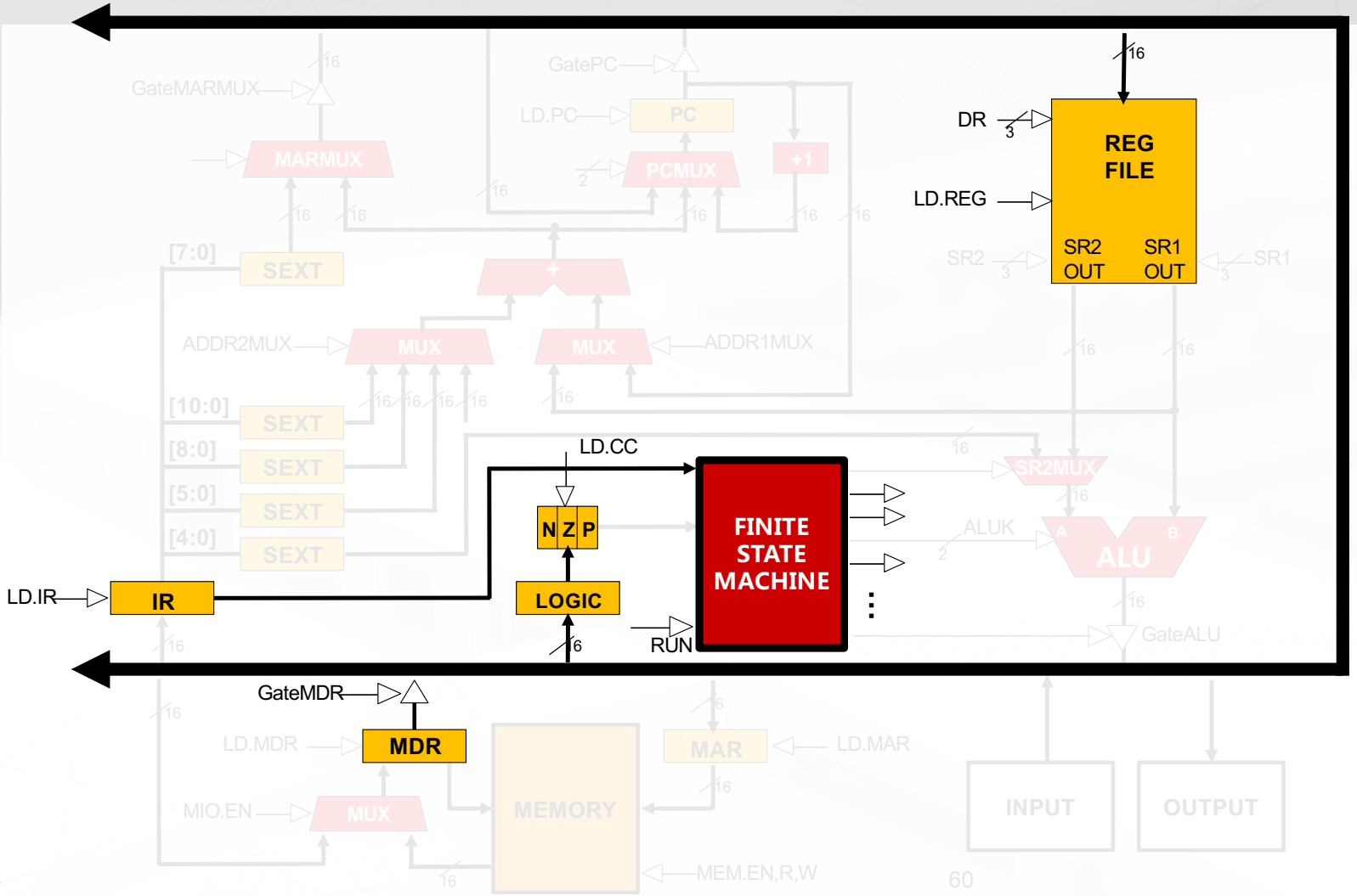


LD (PC-Relative)

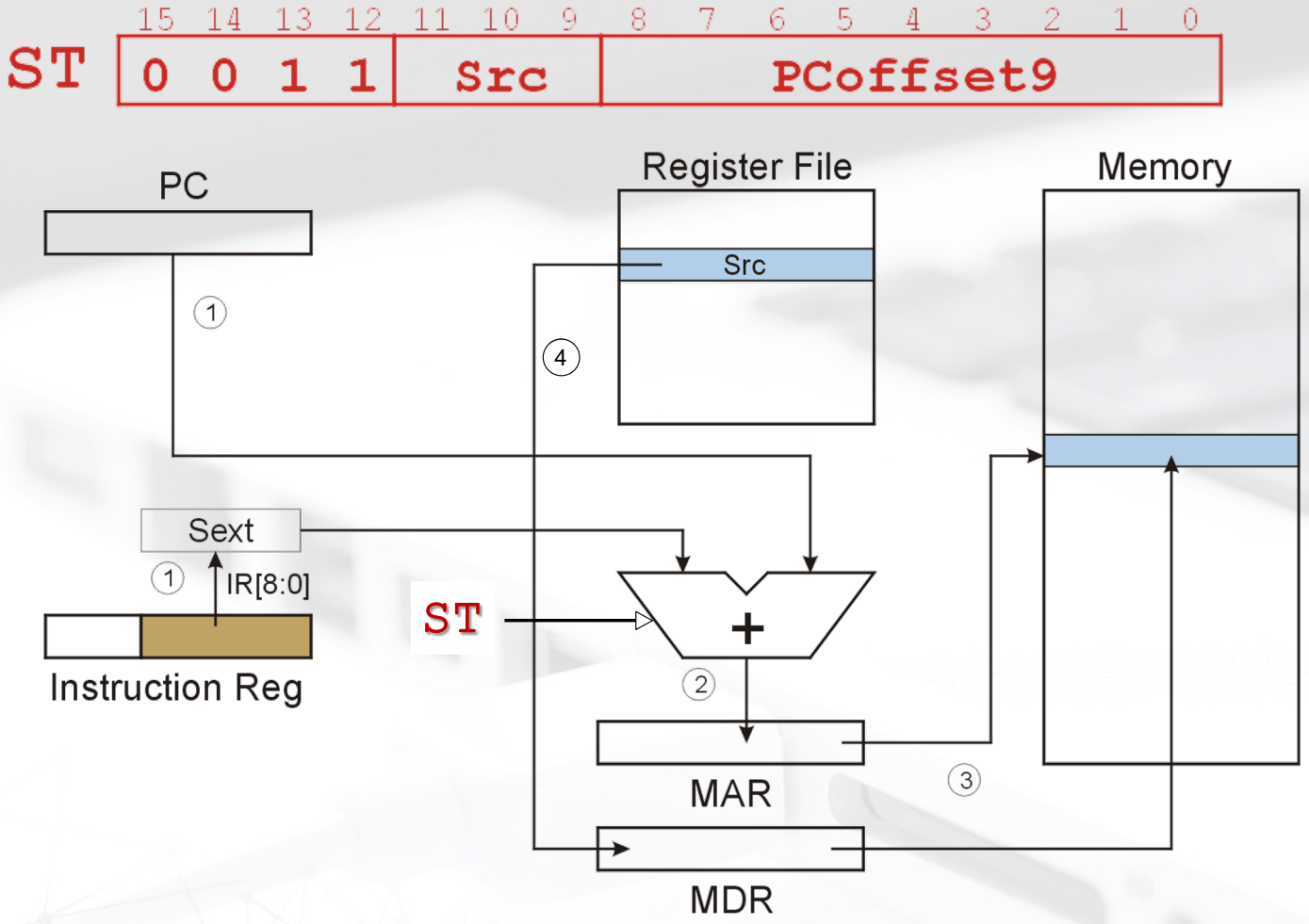


LD (PC-Relative)

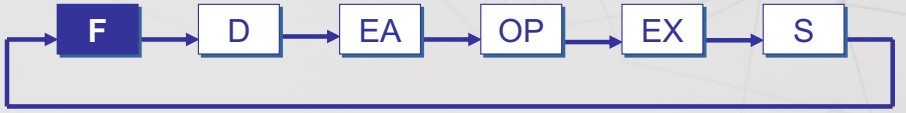


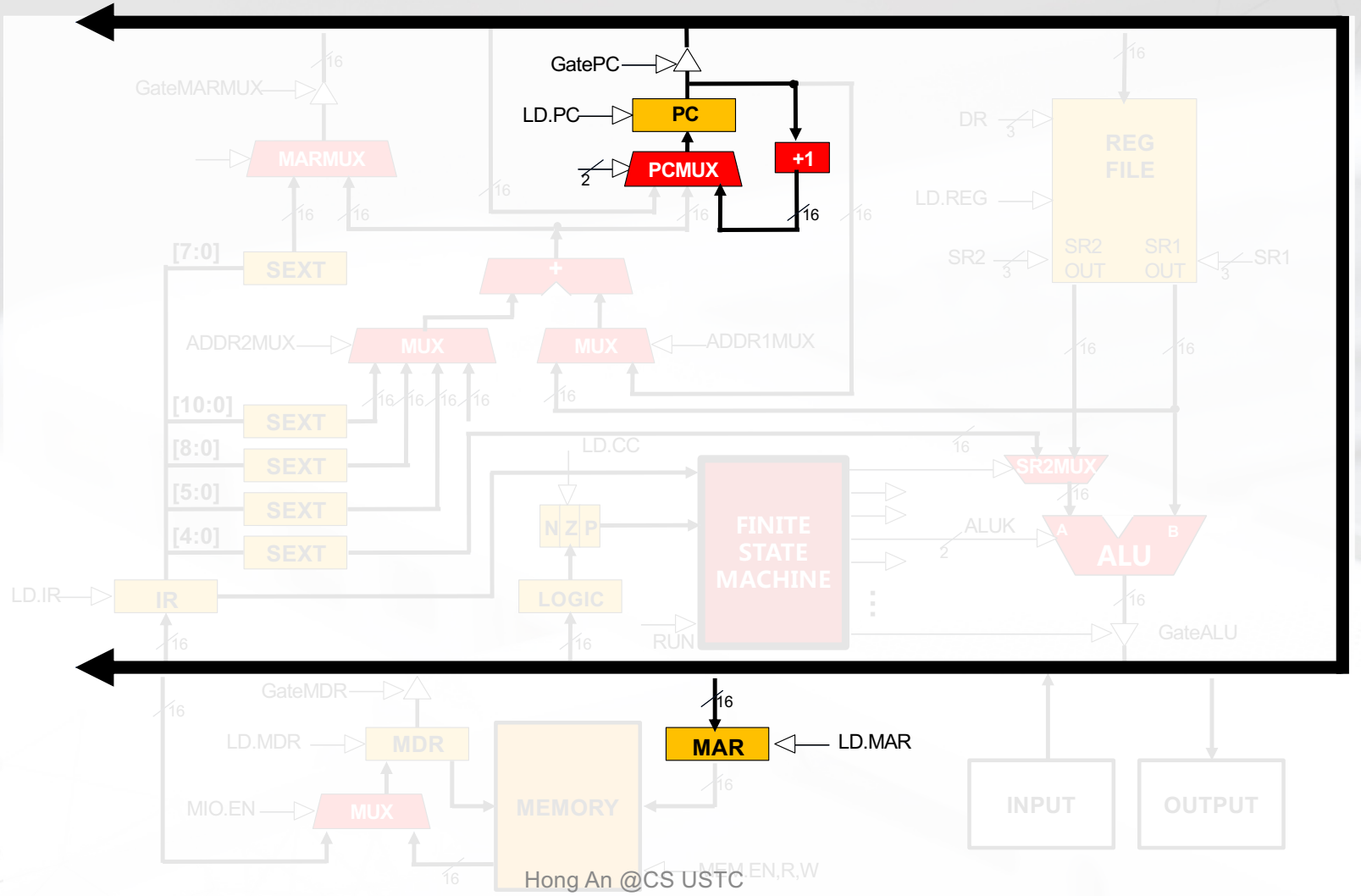


ST (PC-Relative) ST DR, PCoffset9

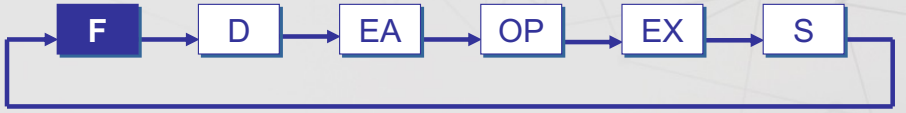


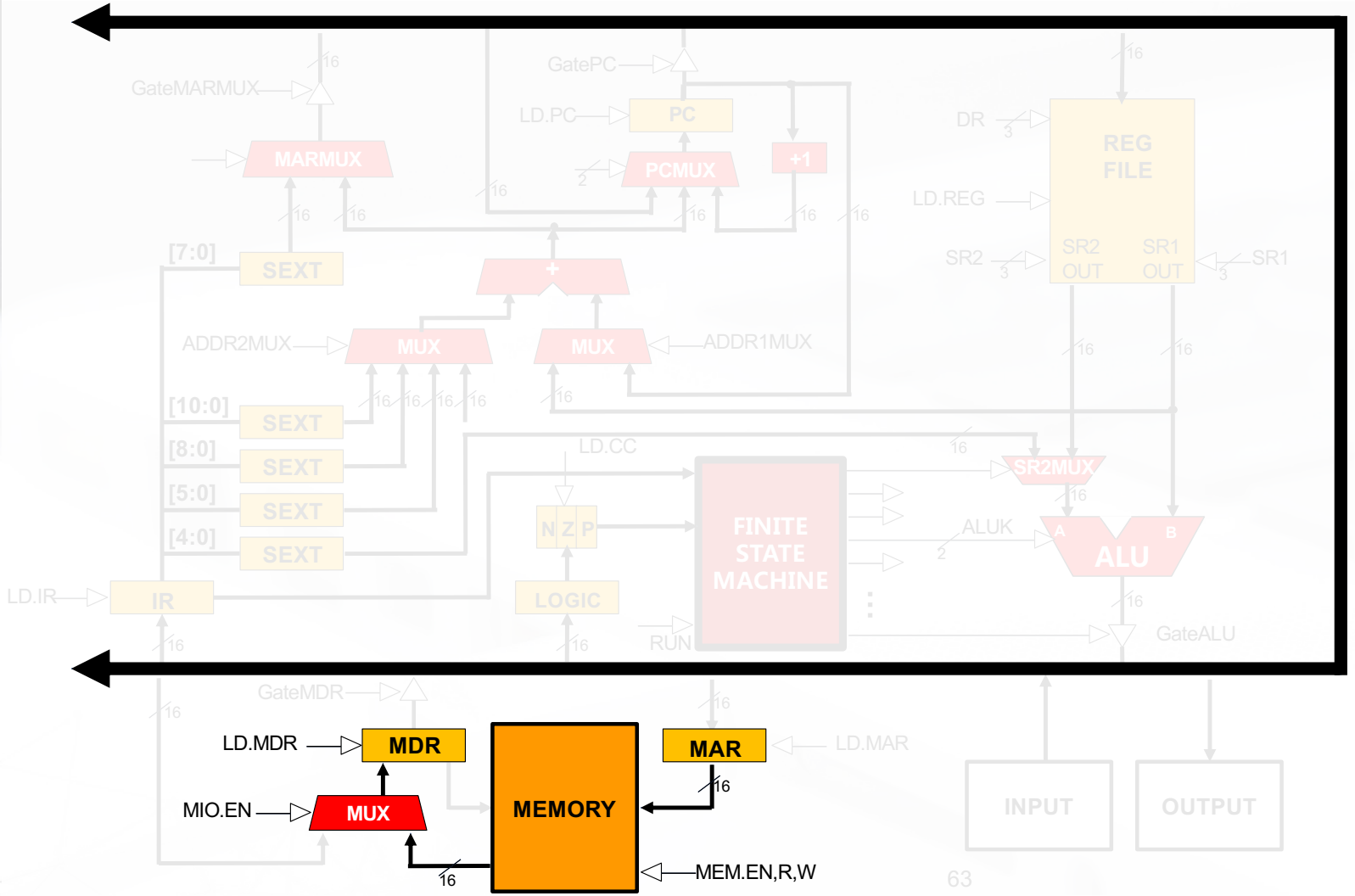
ST (PC-Relative)



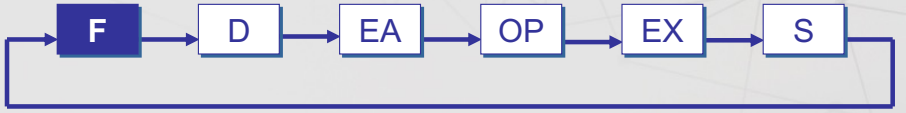


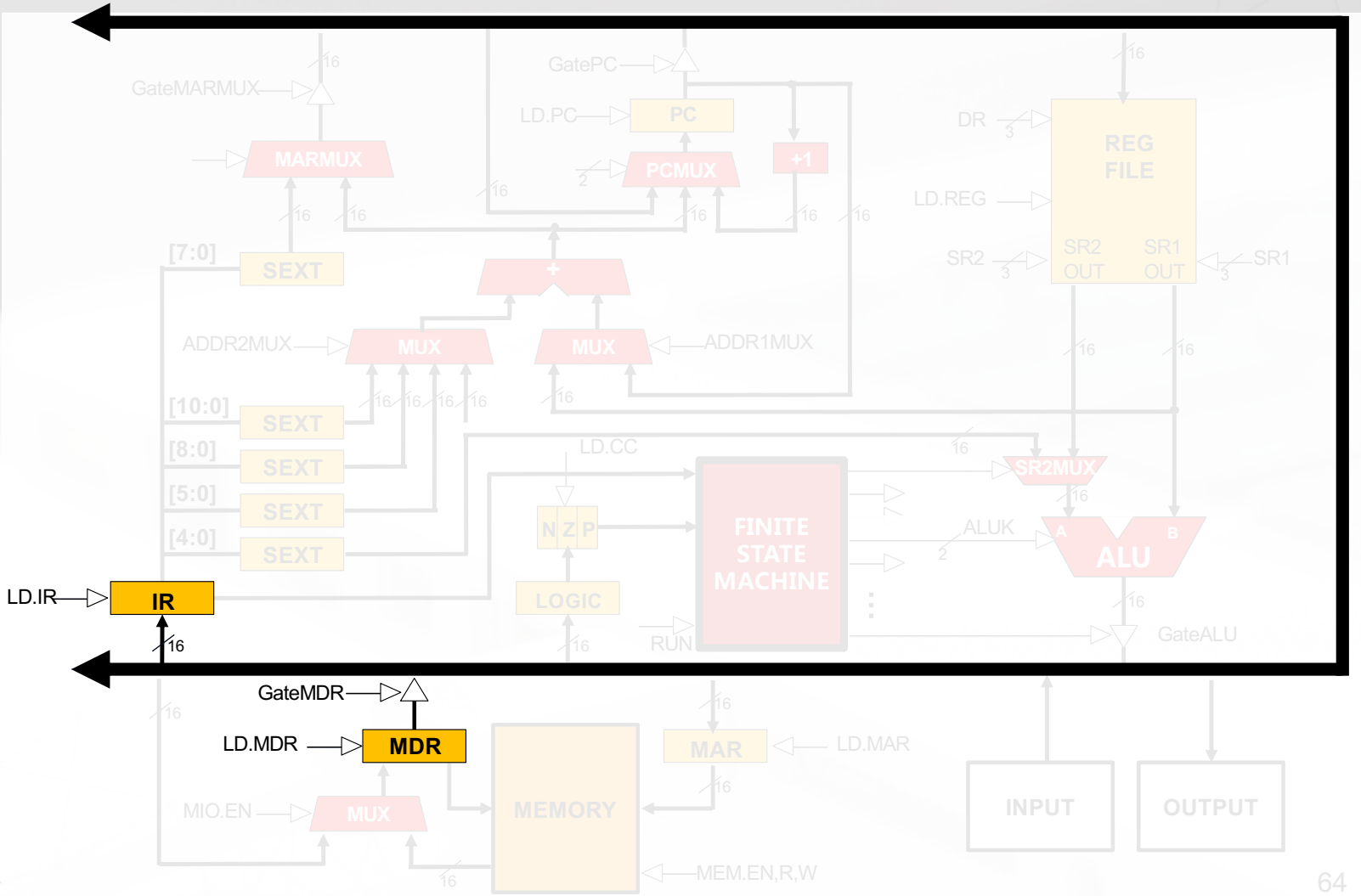
ST (PC-Relative)



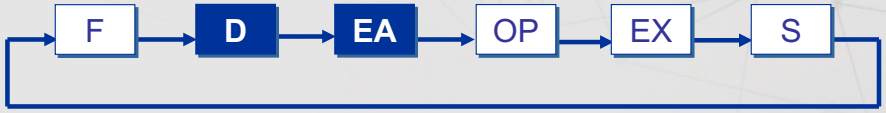


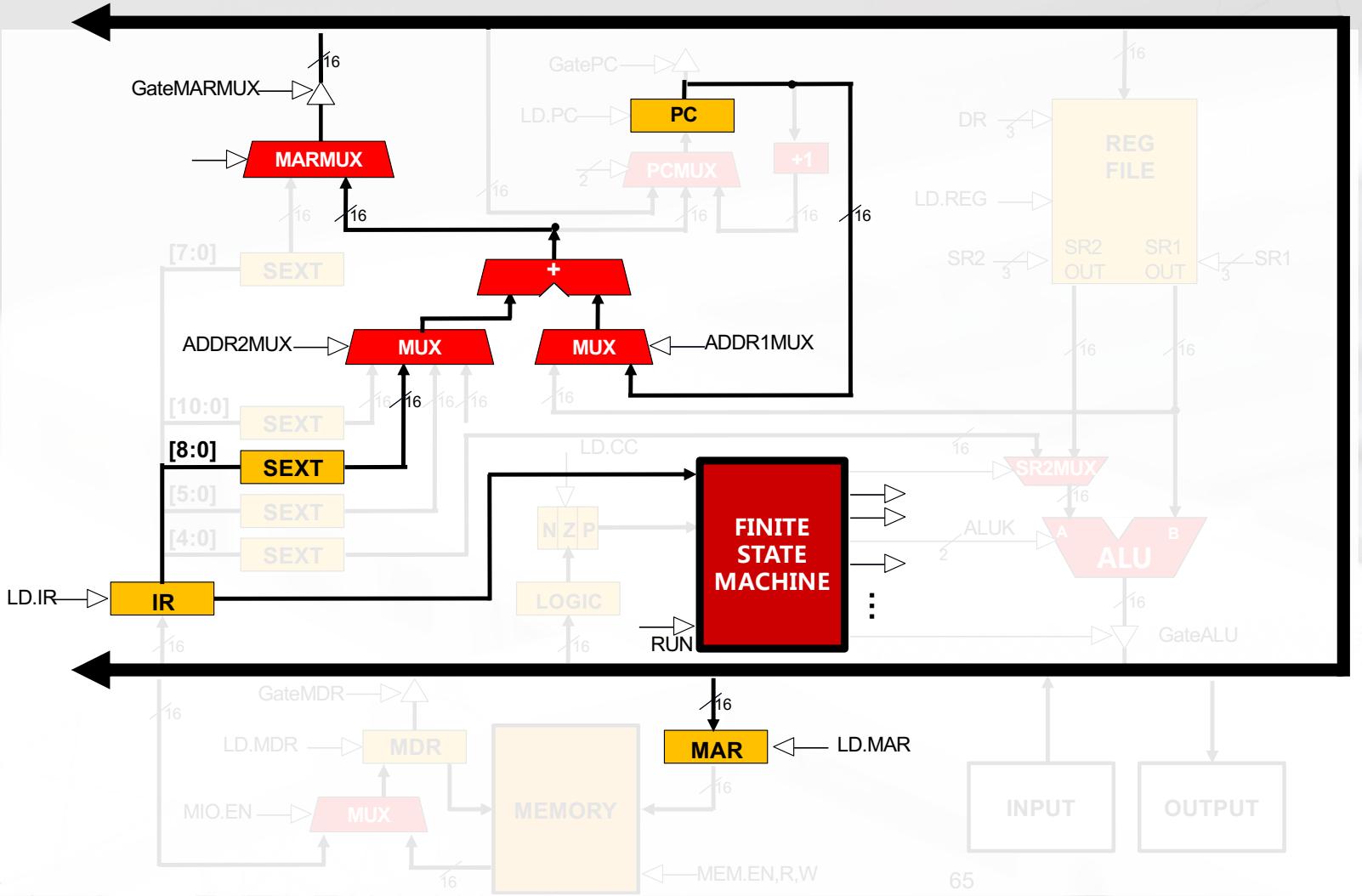
ST (PC-Relative)



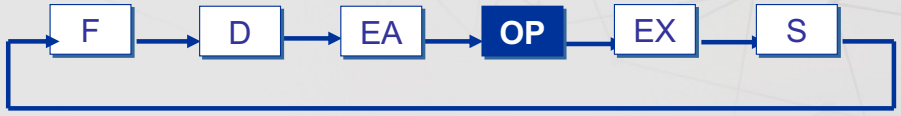


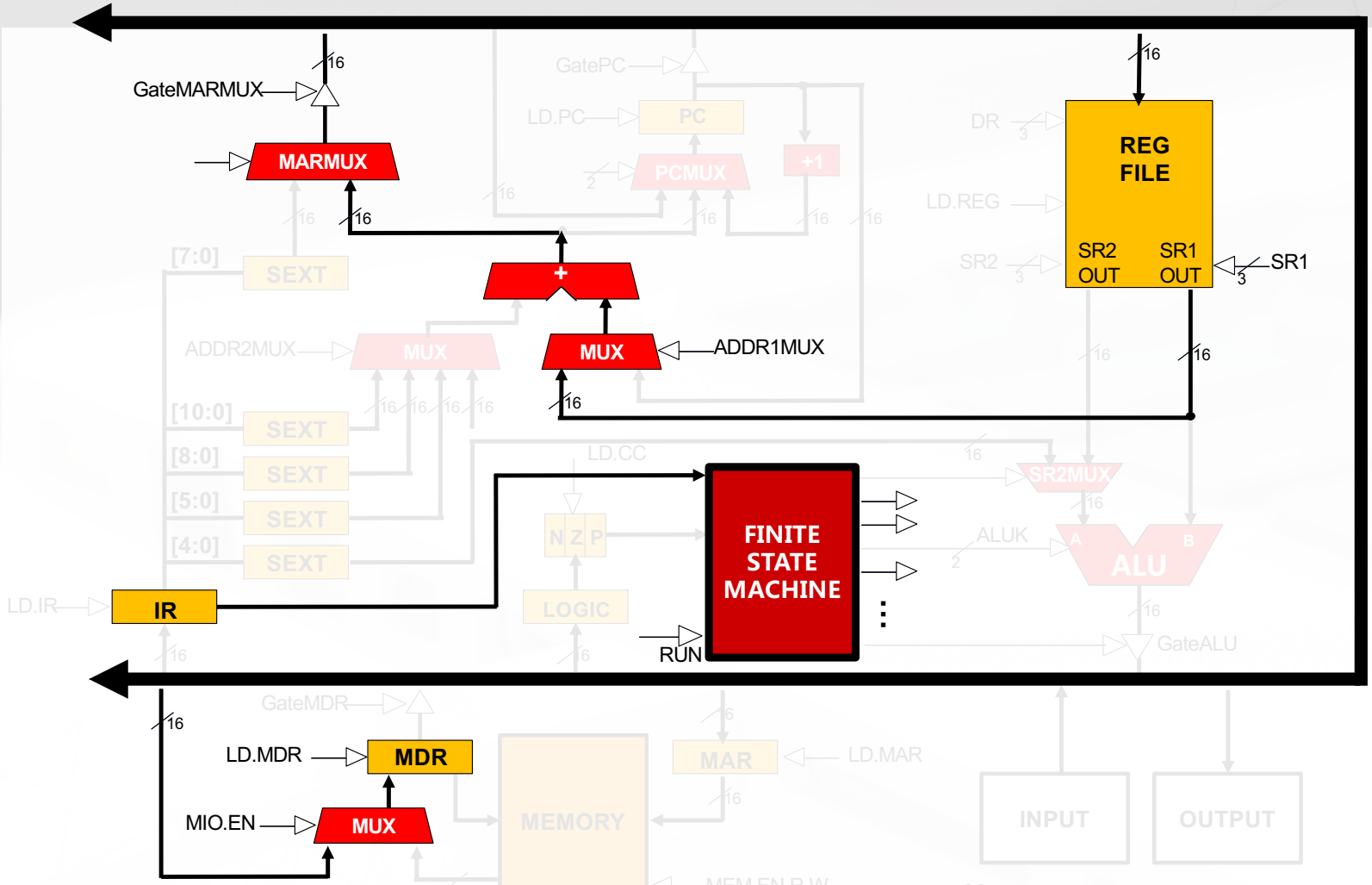
ST (PC-Relative)



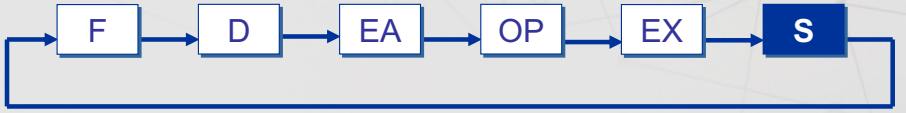


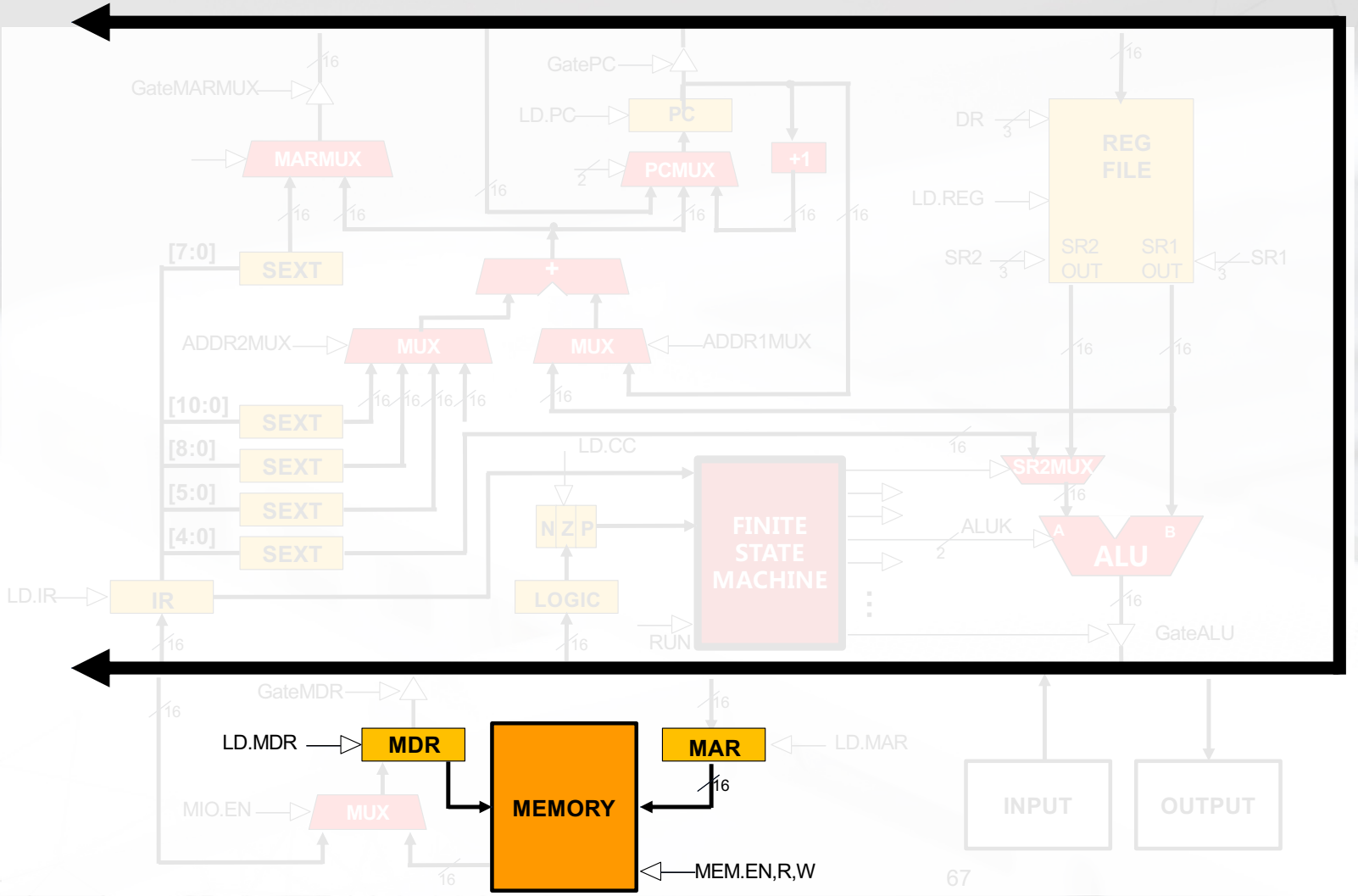
ST (PC-Relative)





ST (PC-Relative)





Indirect Addressing Mode

With PC-relative mode, can only address data within 256 words of the instruction.

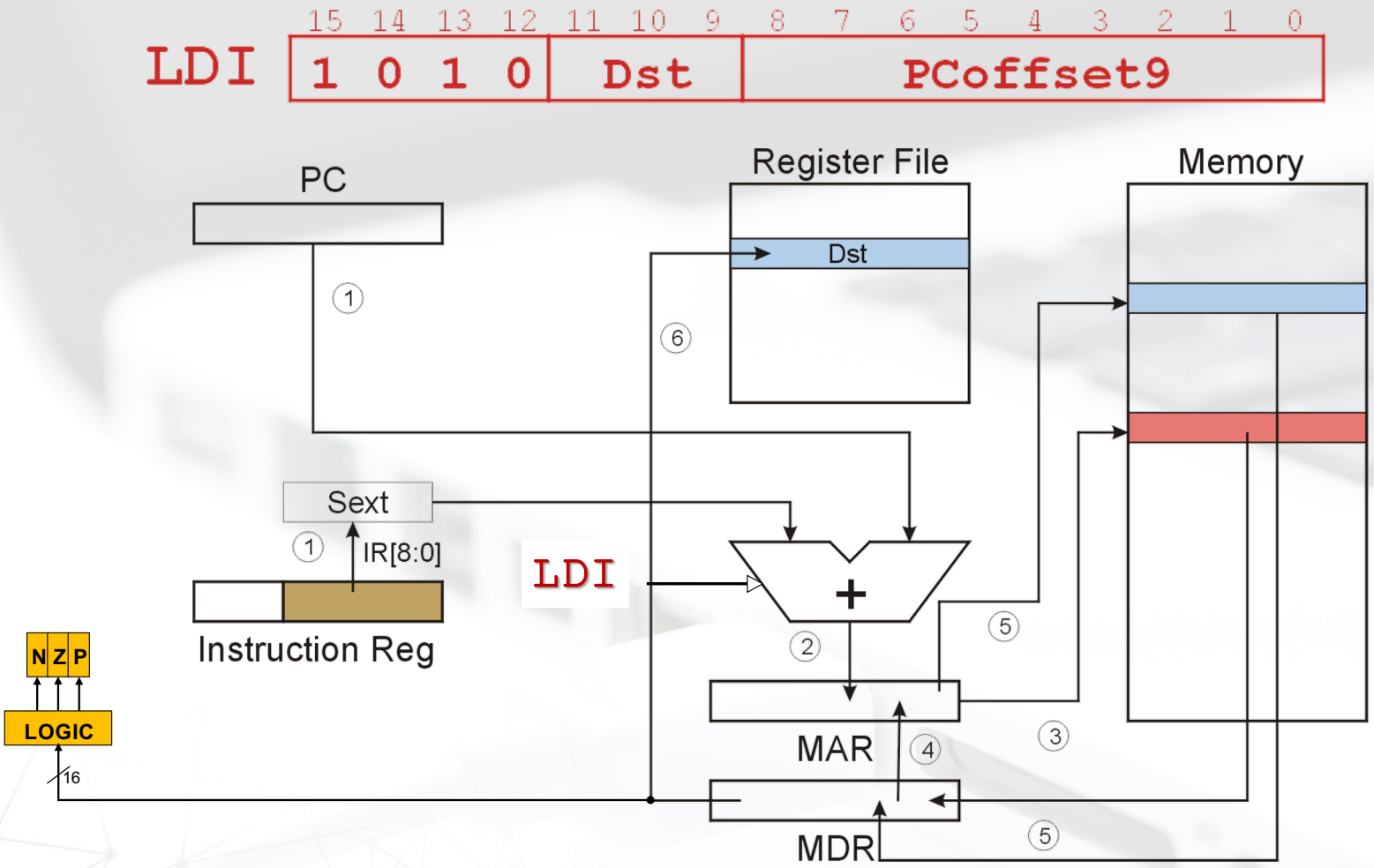
- What about the rest of memory?

Solution #1:

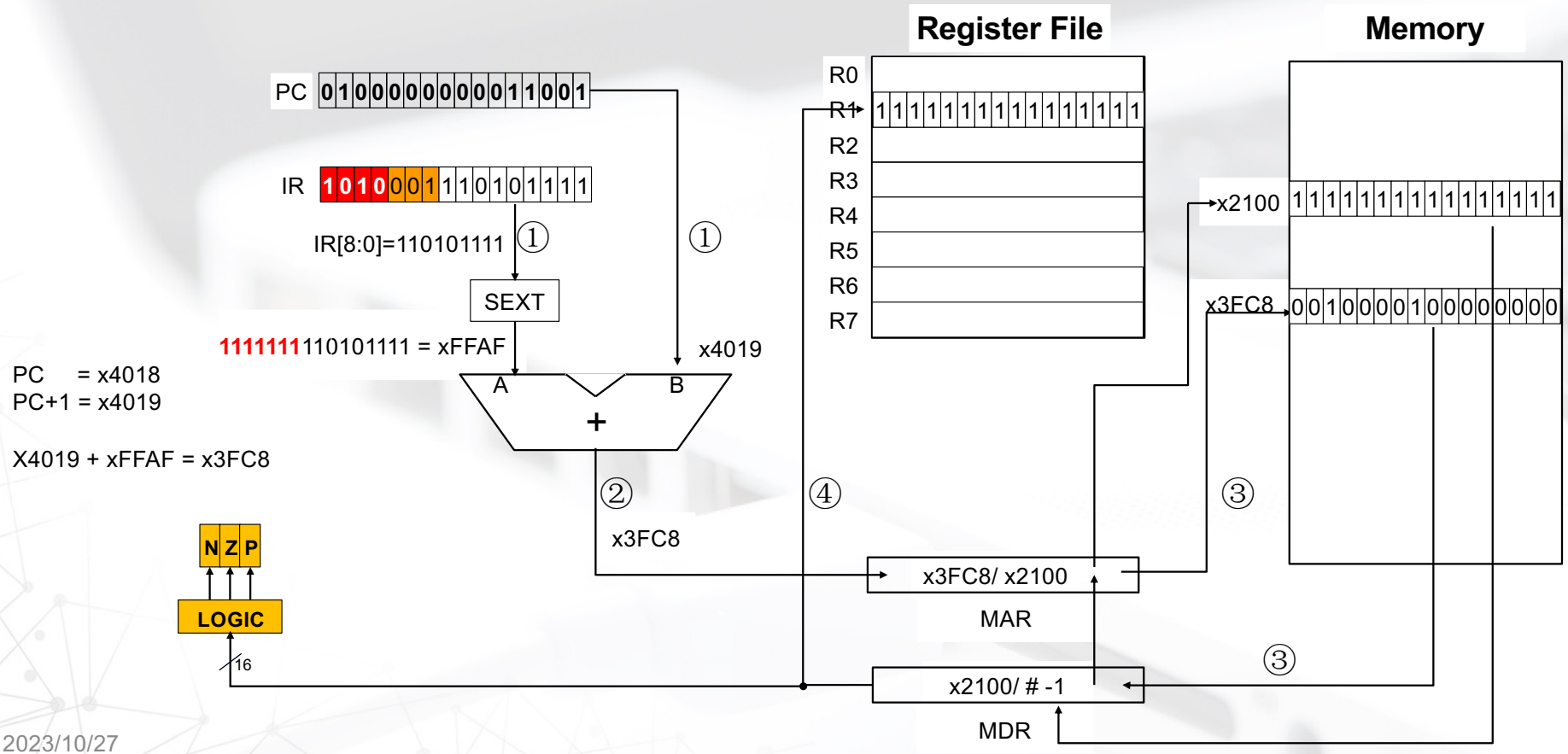
- Read address from memory location, then load/store to that address.

First address is generated from PC and IR (just like PC-relative addressing), then content of that address is used as target for load/store.

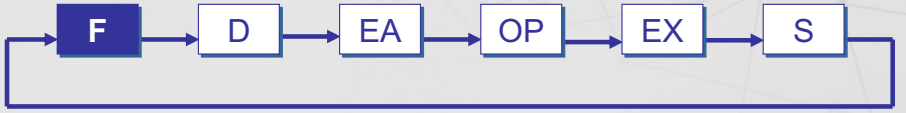
LDI (Indirect) LDI DR, PCoffset9

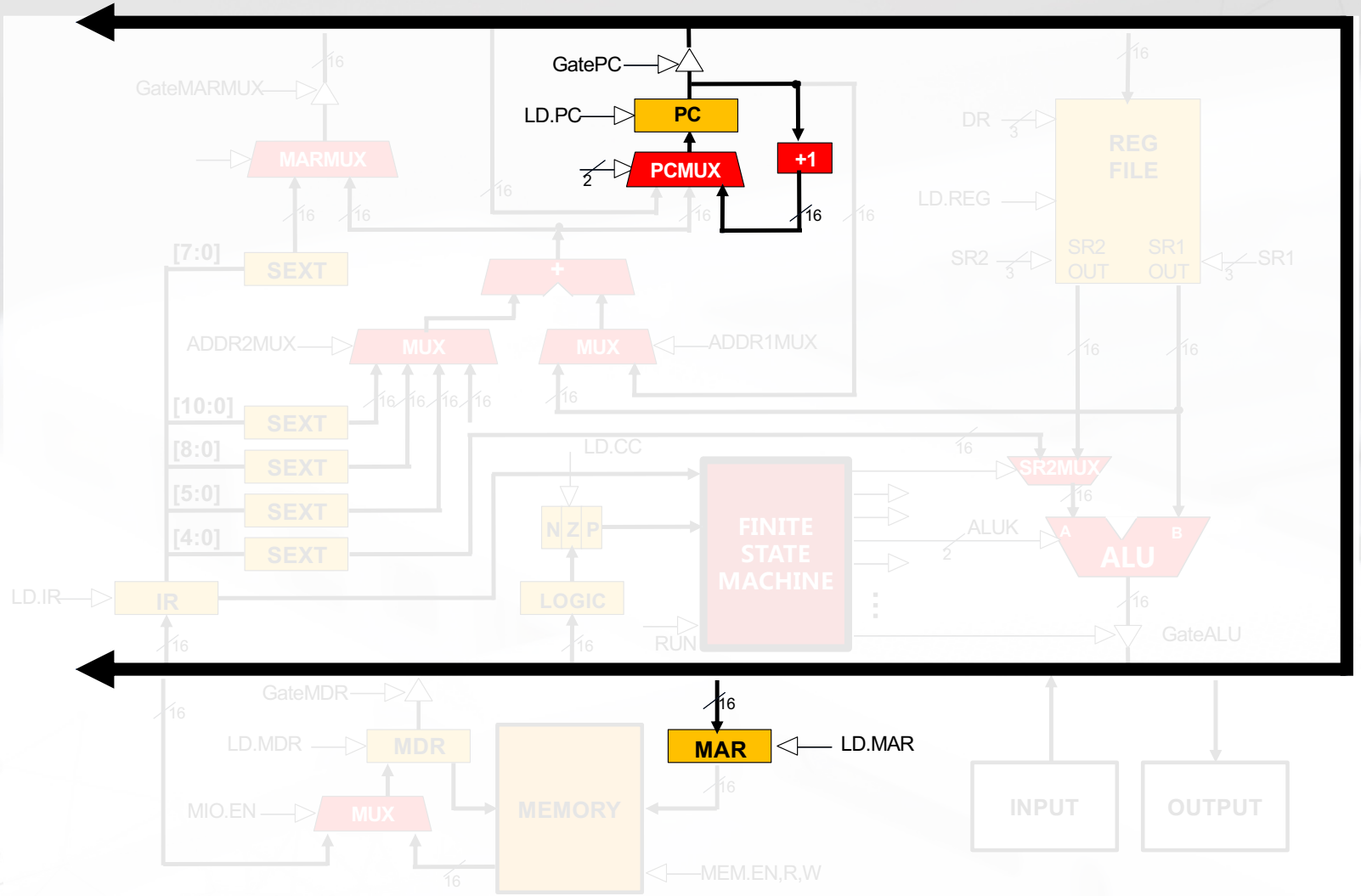


LDI (Indirect) : LDI R1, x1AF

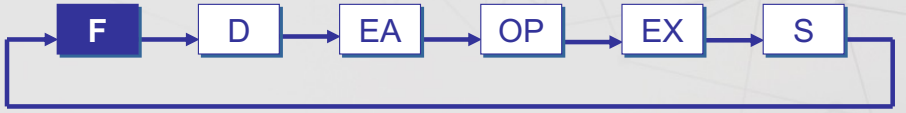


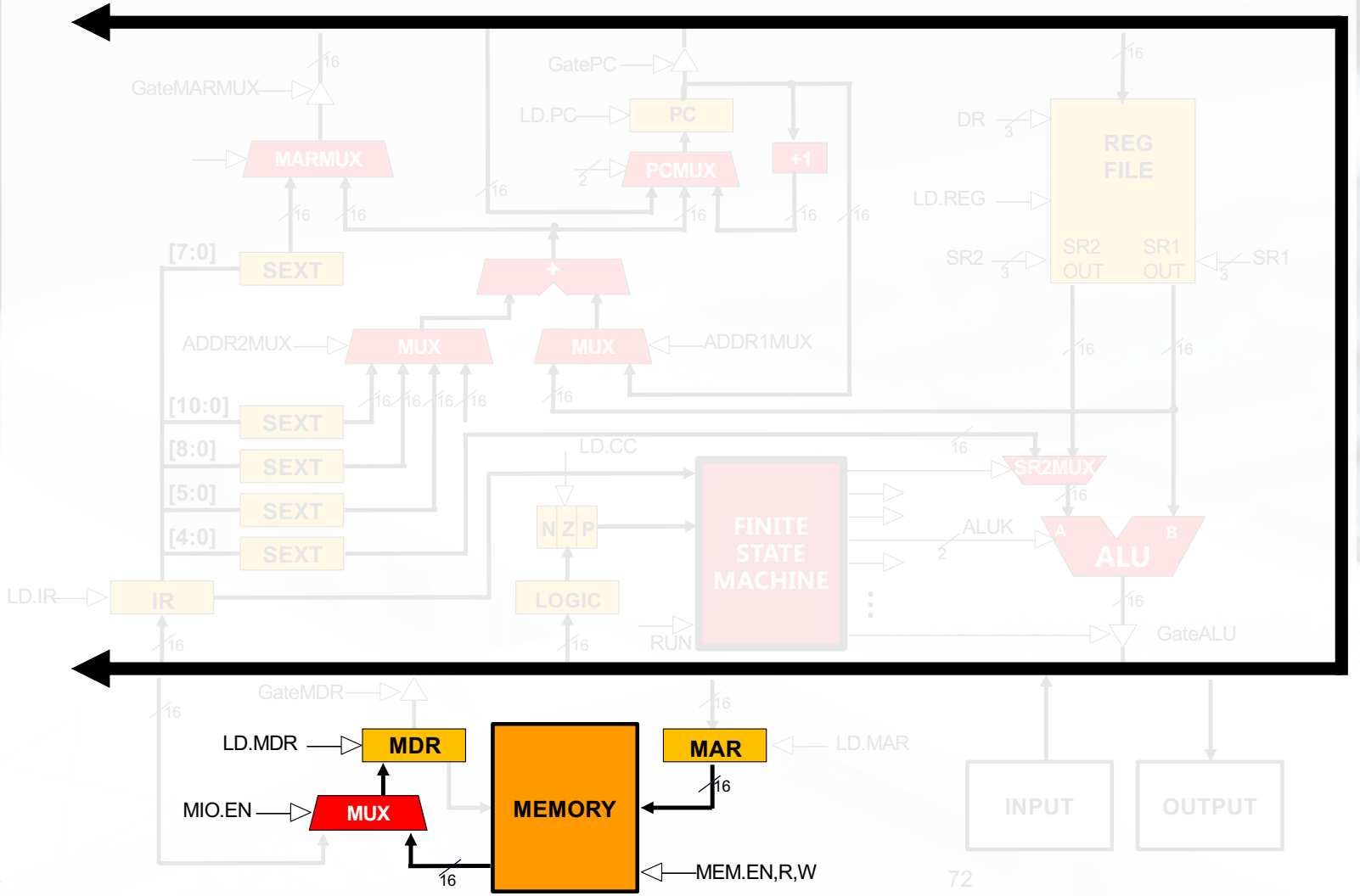
LDI (Indirect)



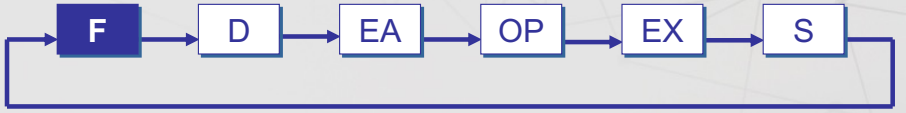


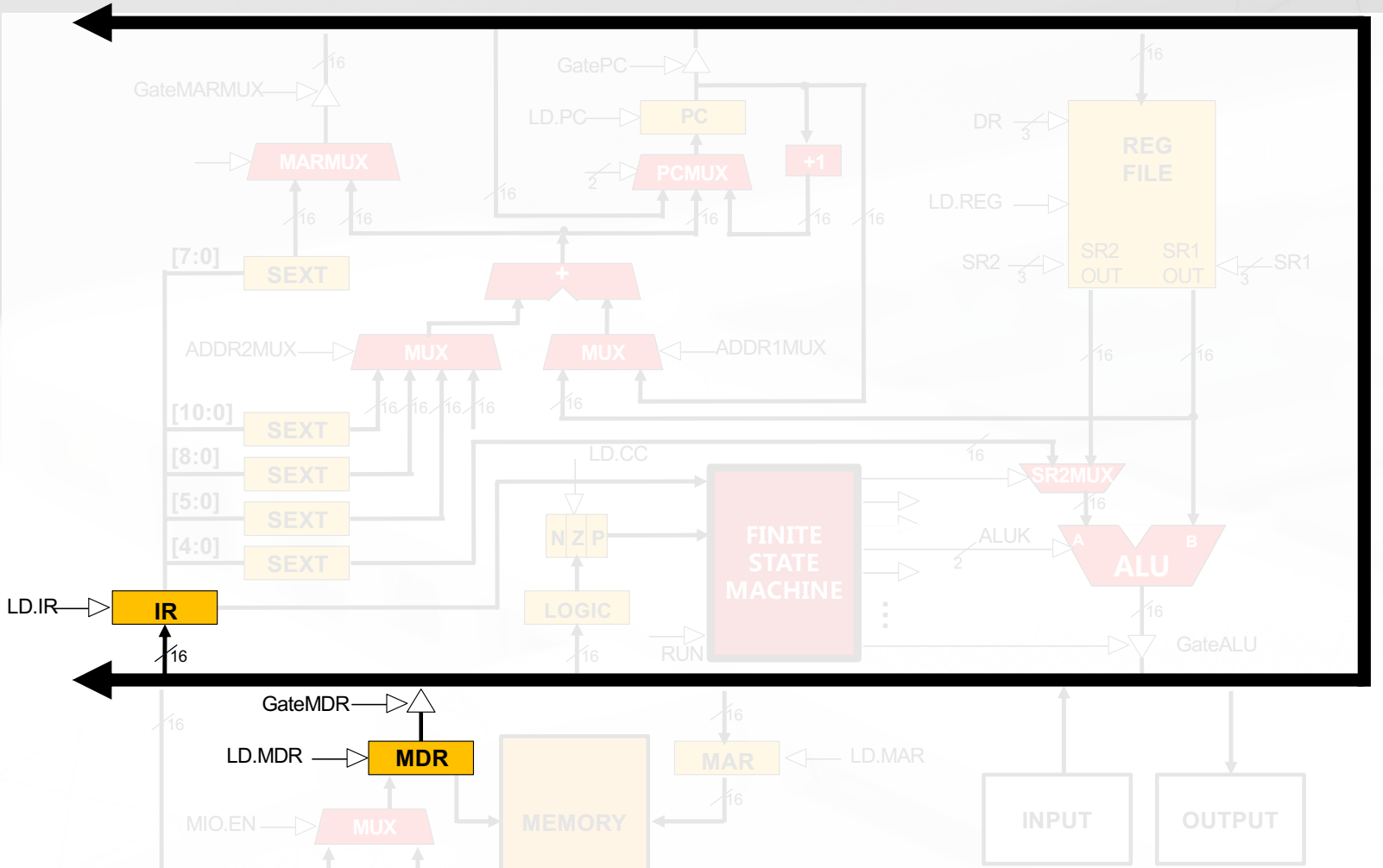
LDI (Indirect)



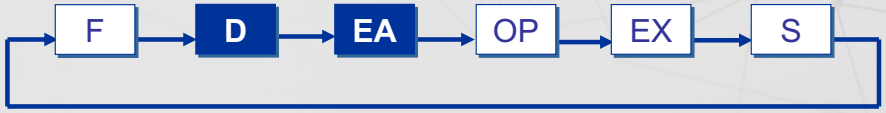


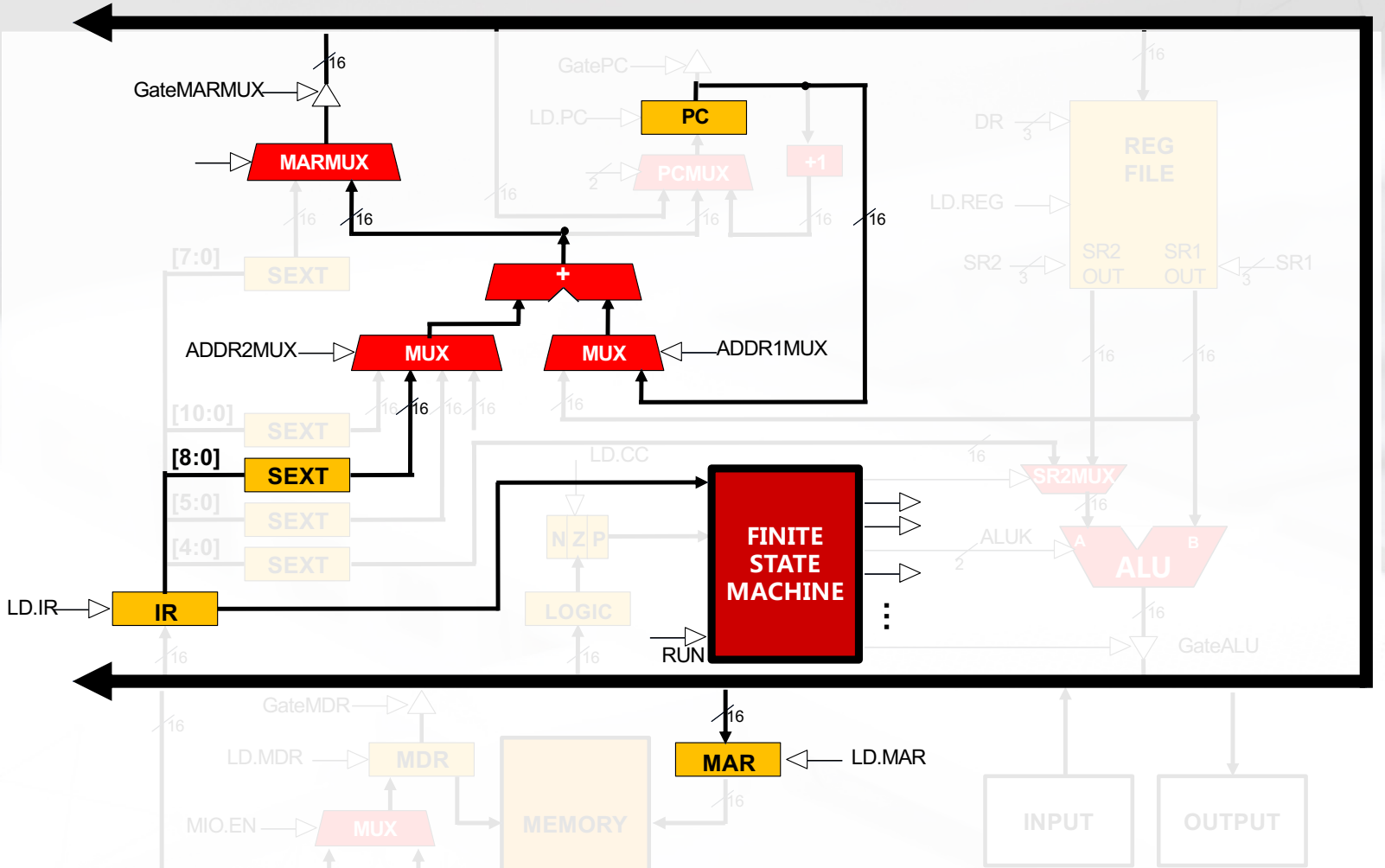
LDI (Indirect)



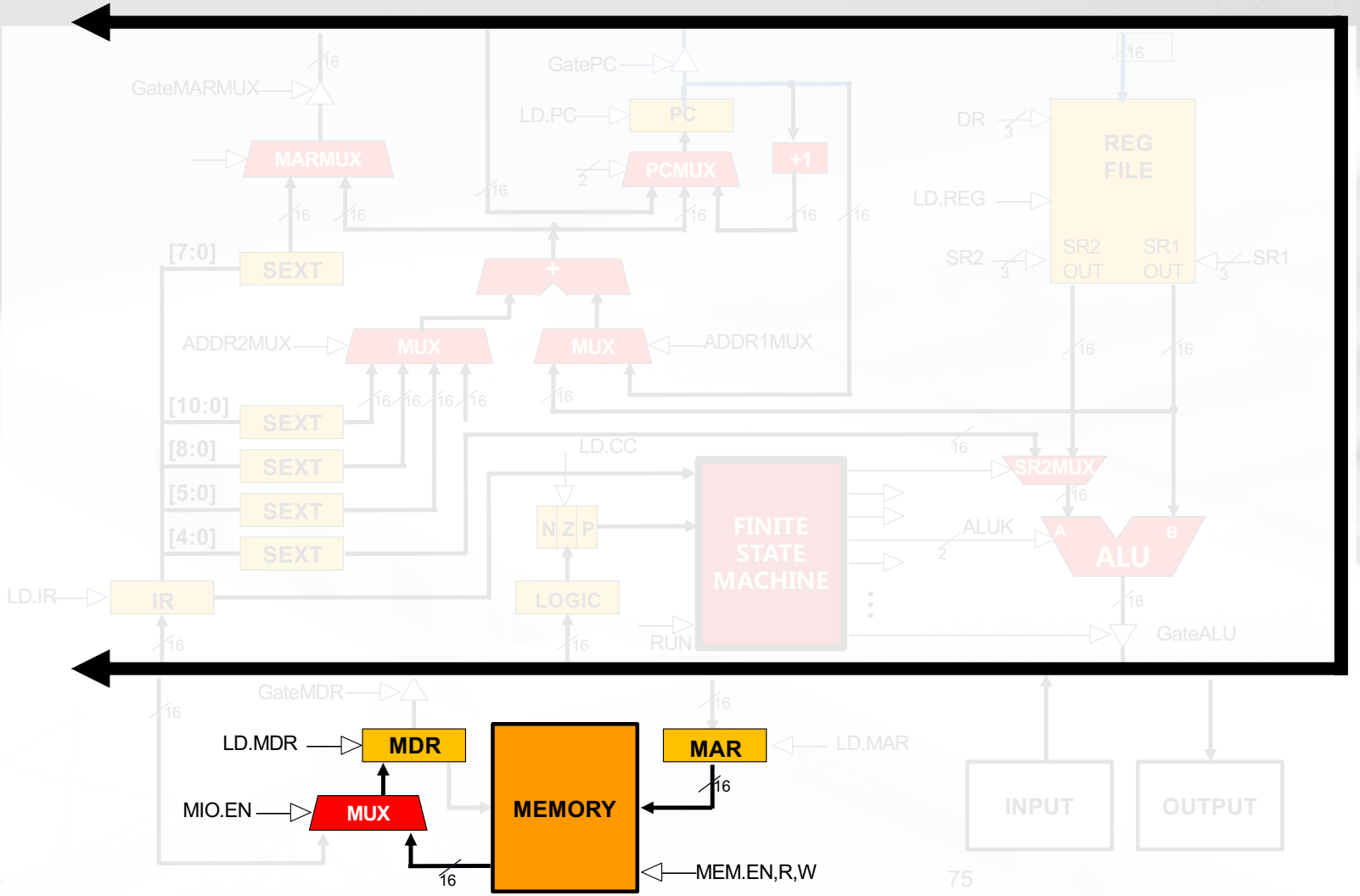


LDI (Indirect)

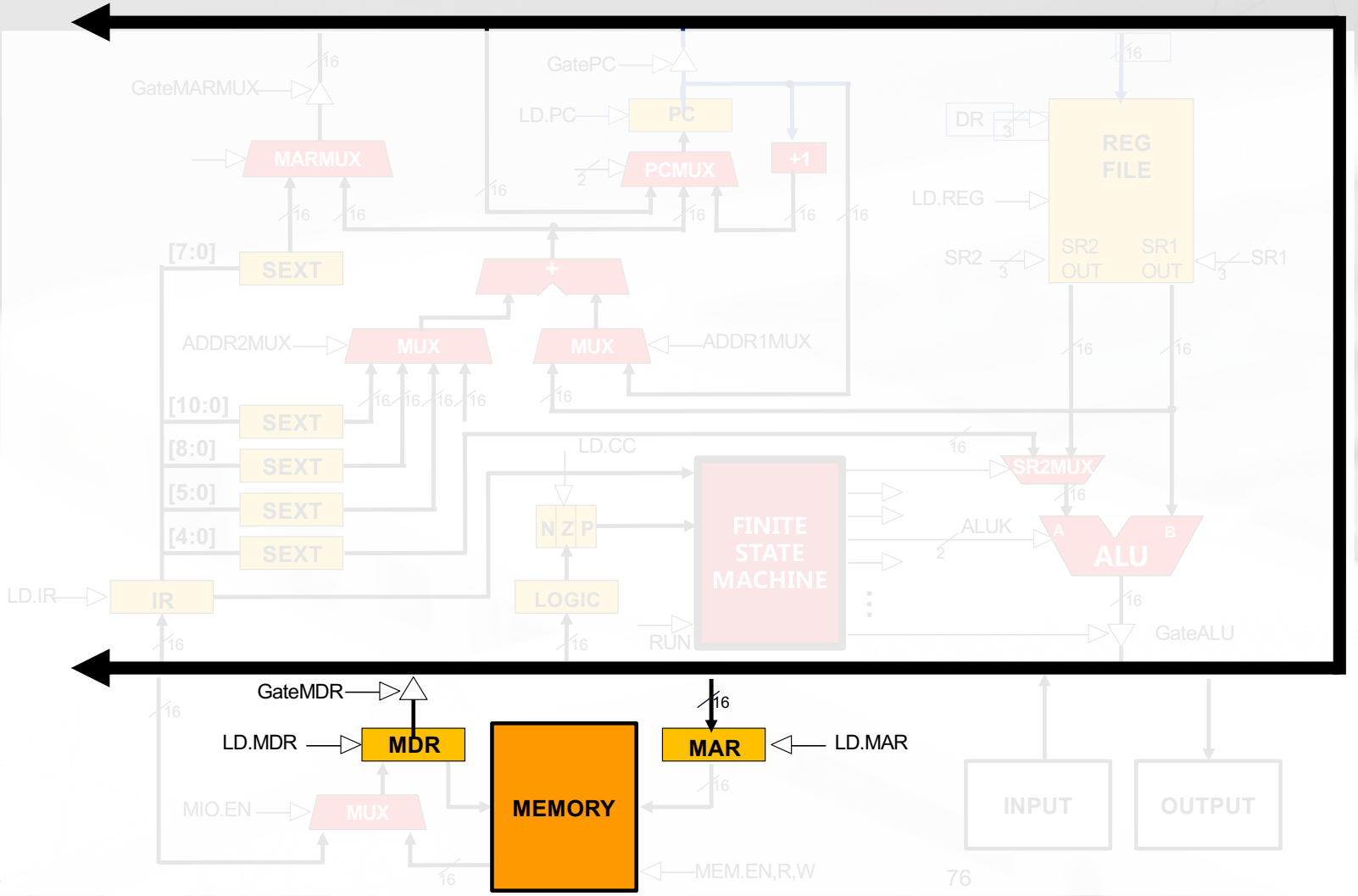




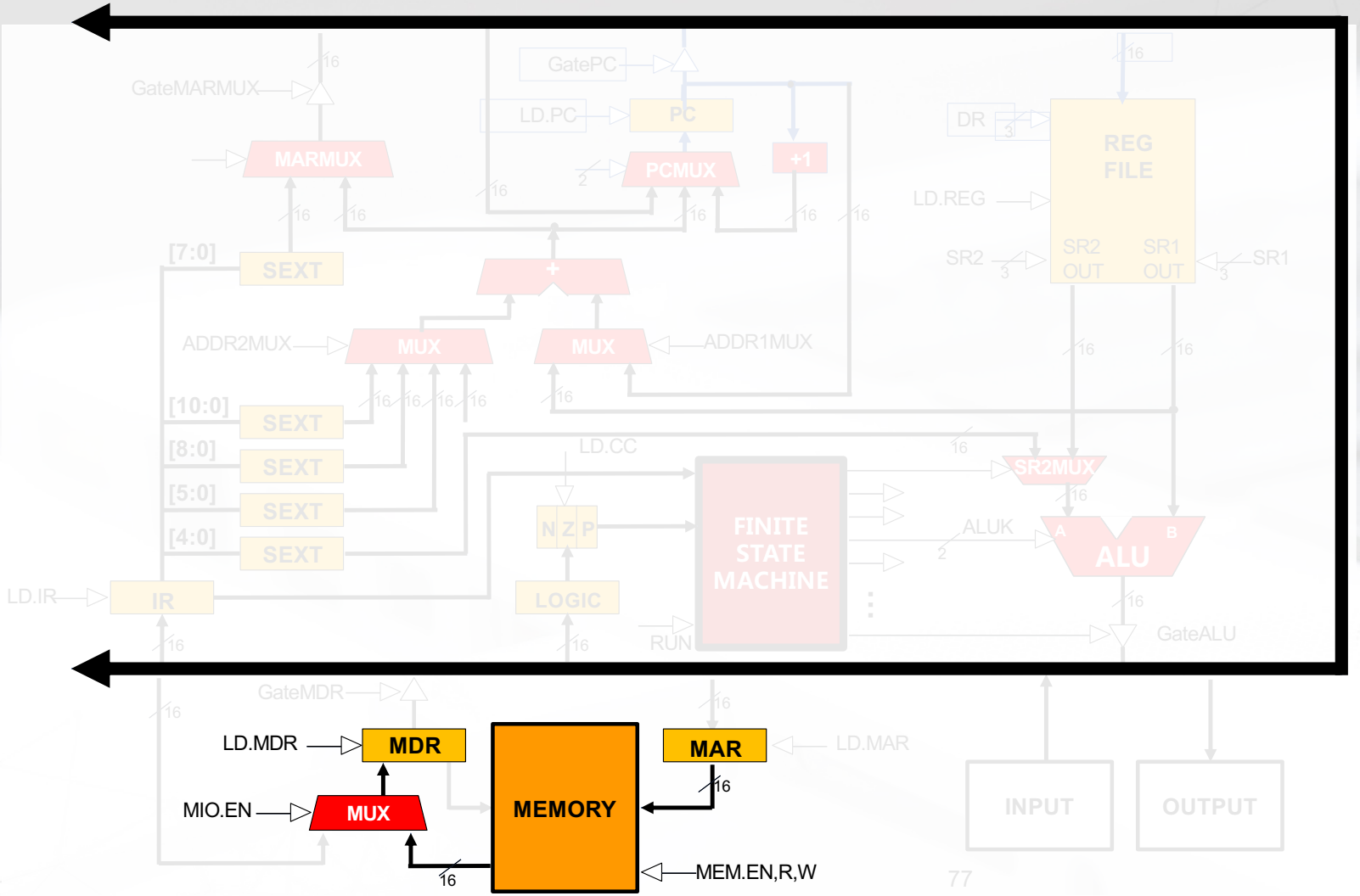
LDI (Indirect)



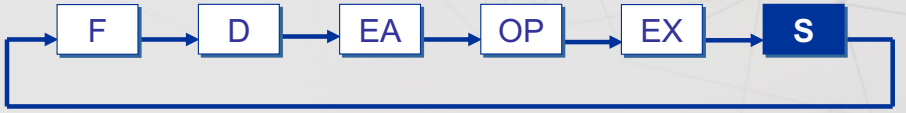
LDI (Indirect)

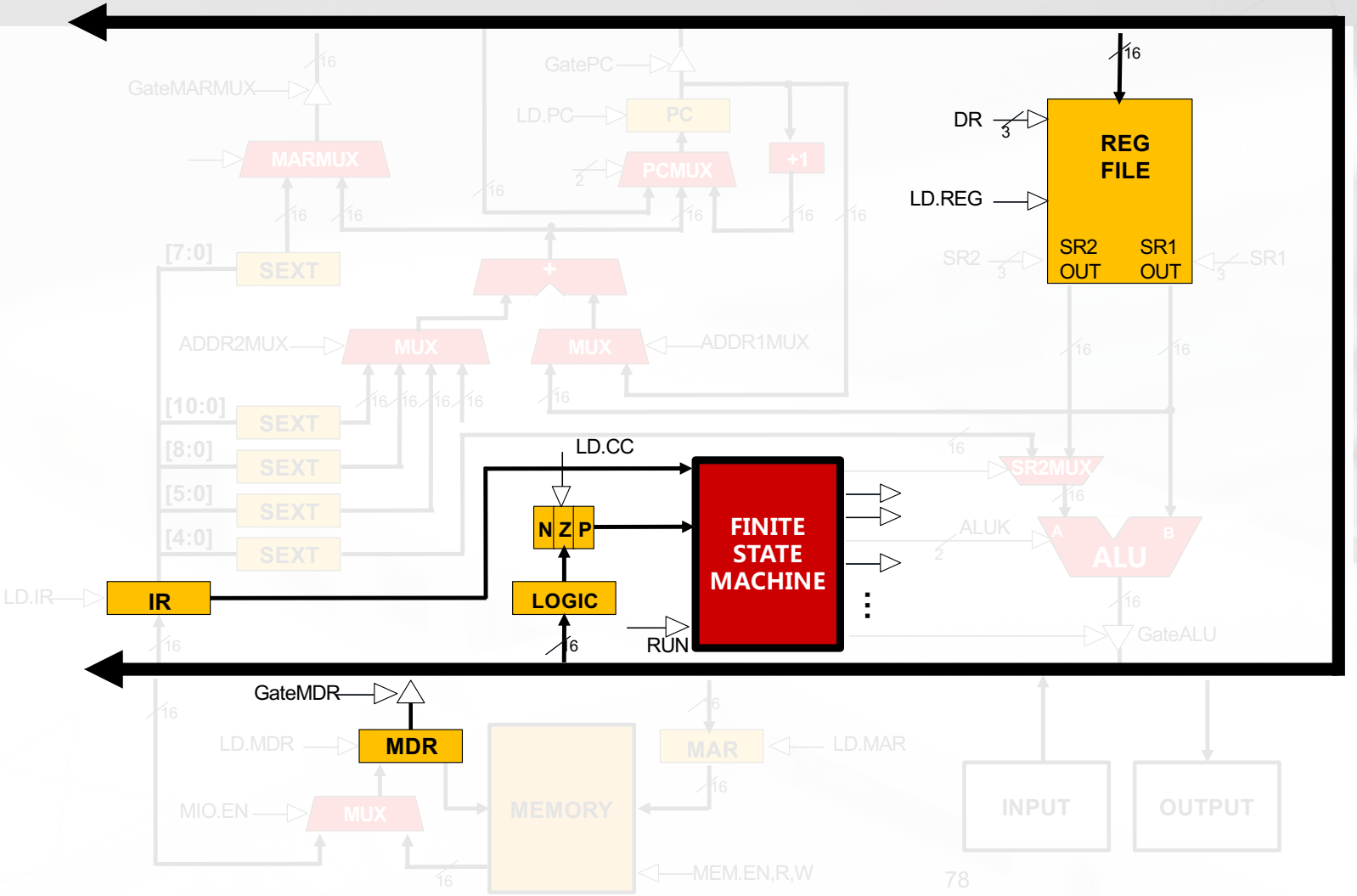


LDI (Indirect)

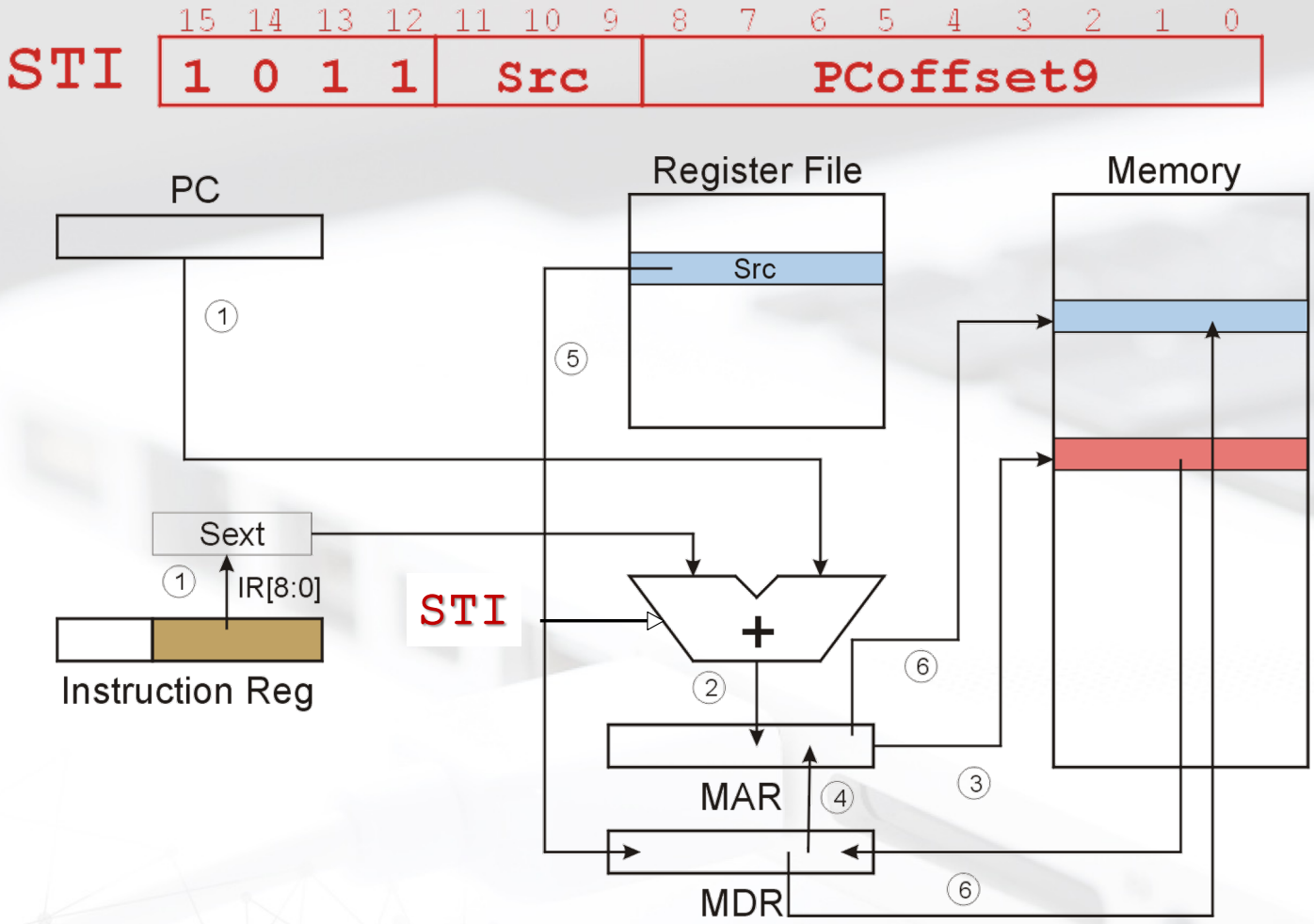


LDI (Indirect)

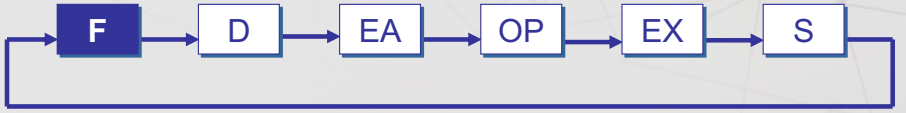


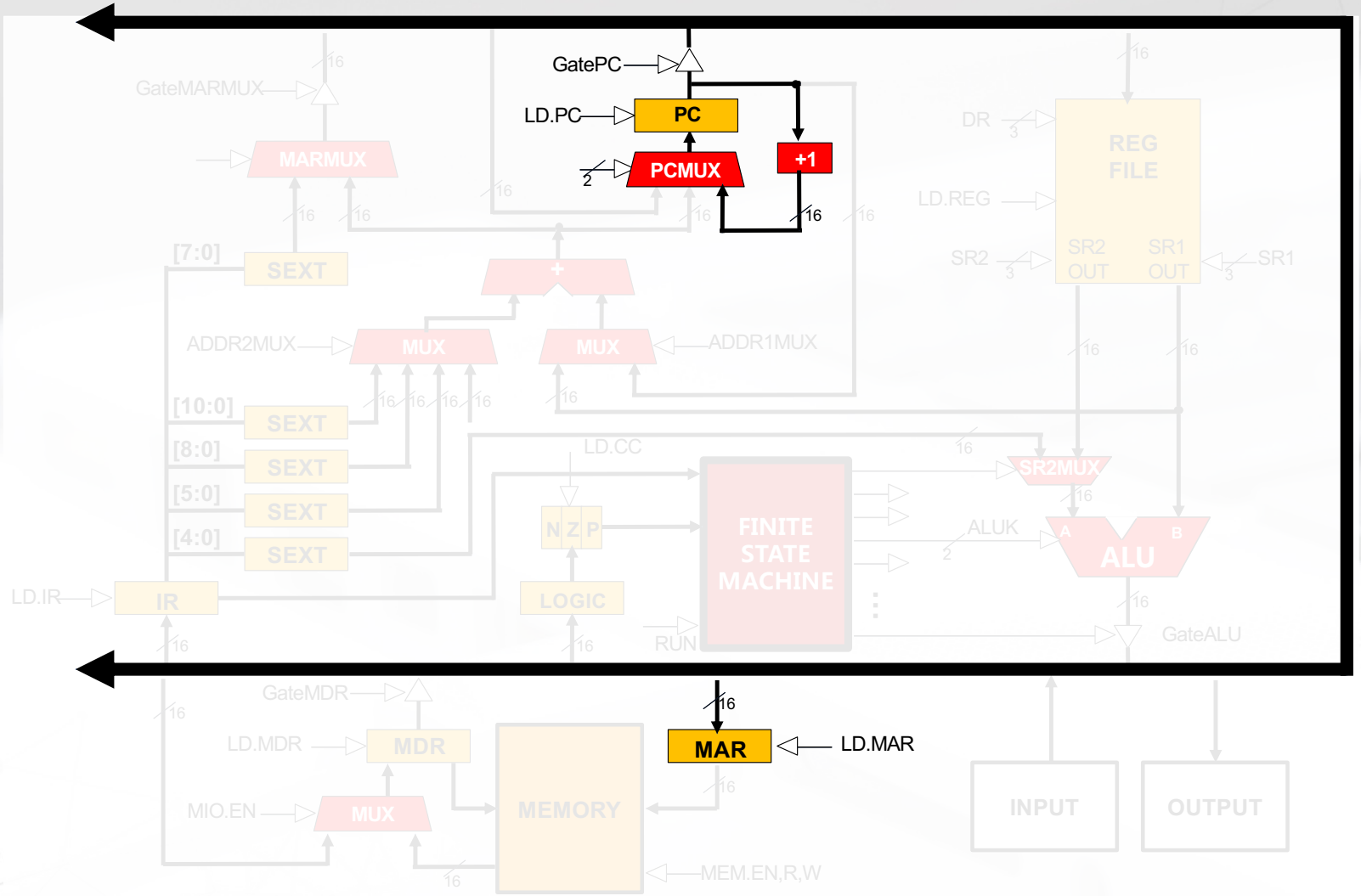


STI (Indirect) STI DR, PCoffset9

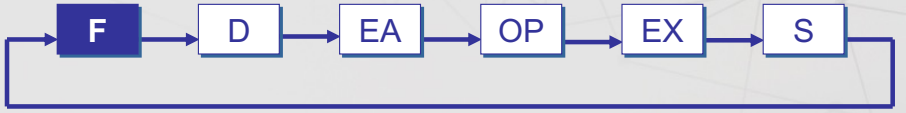


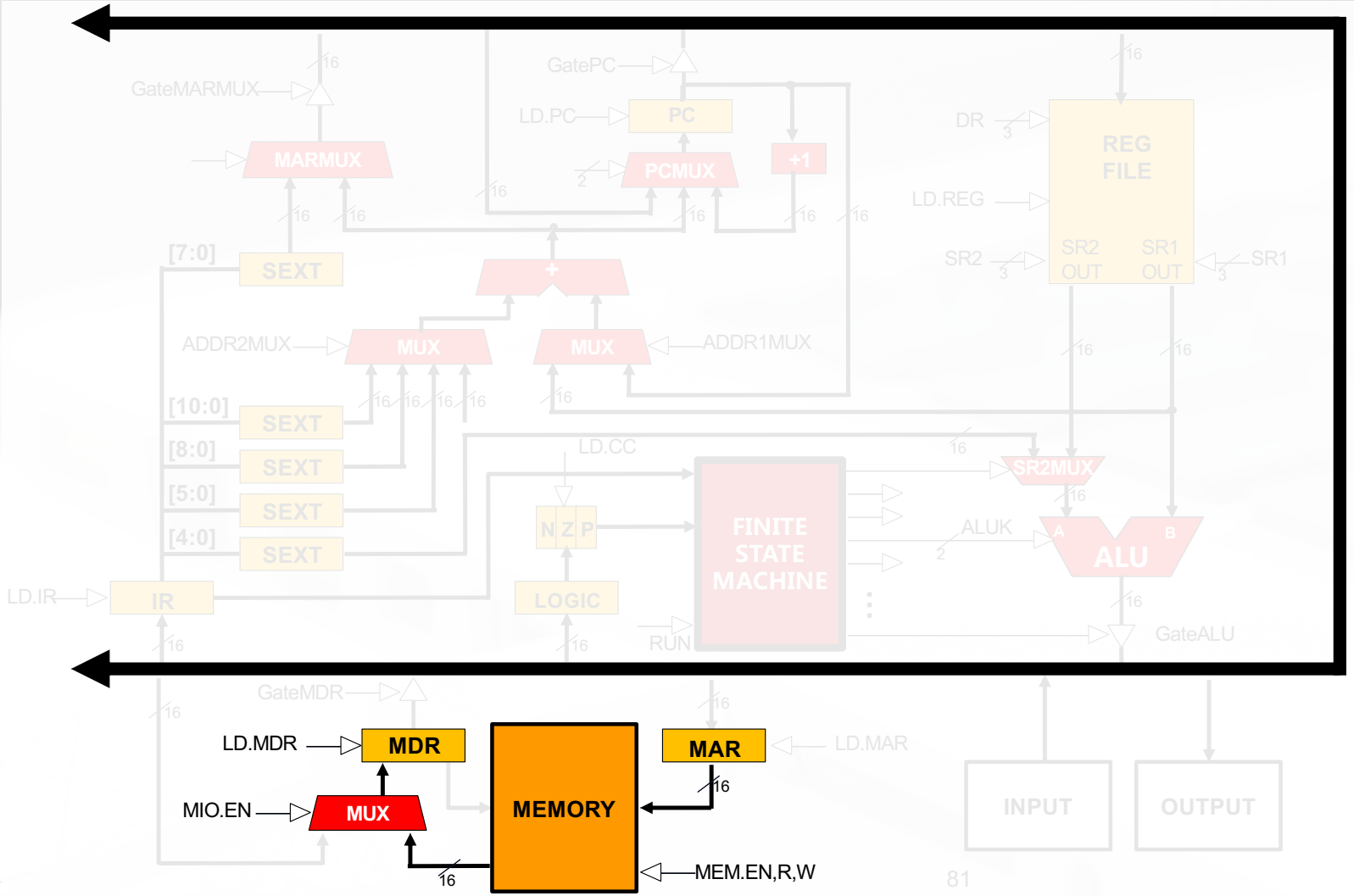
STI (Indirect)



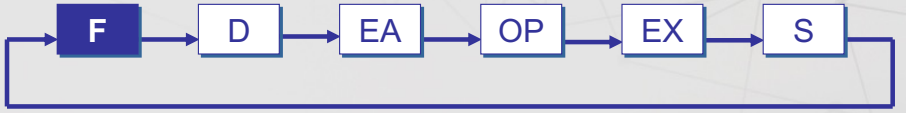


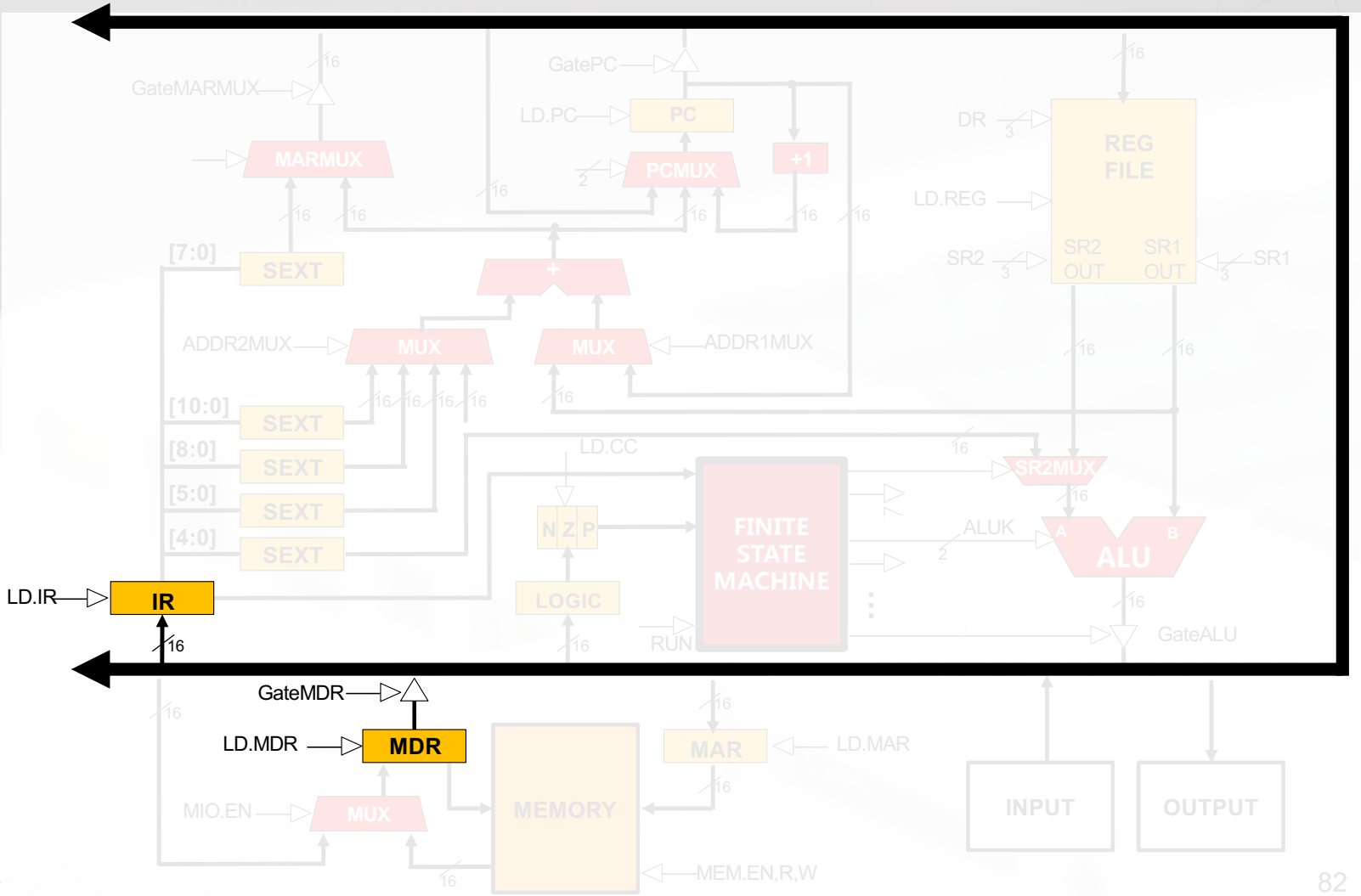
STI (Indirect)



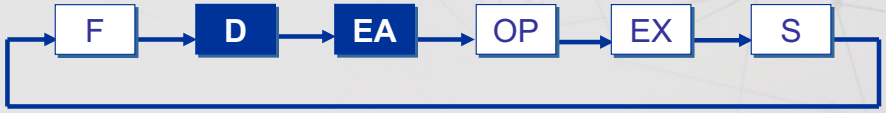


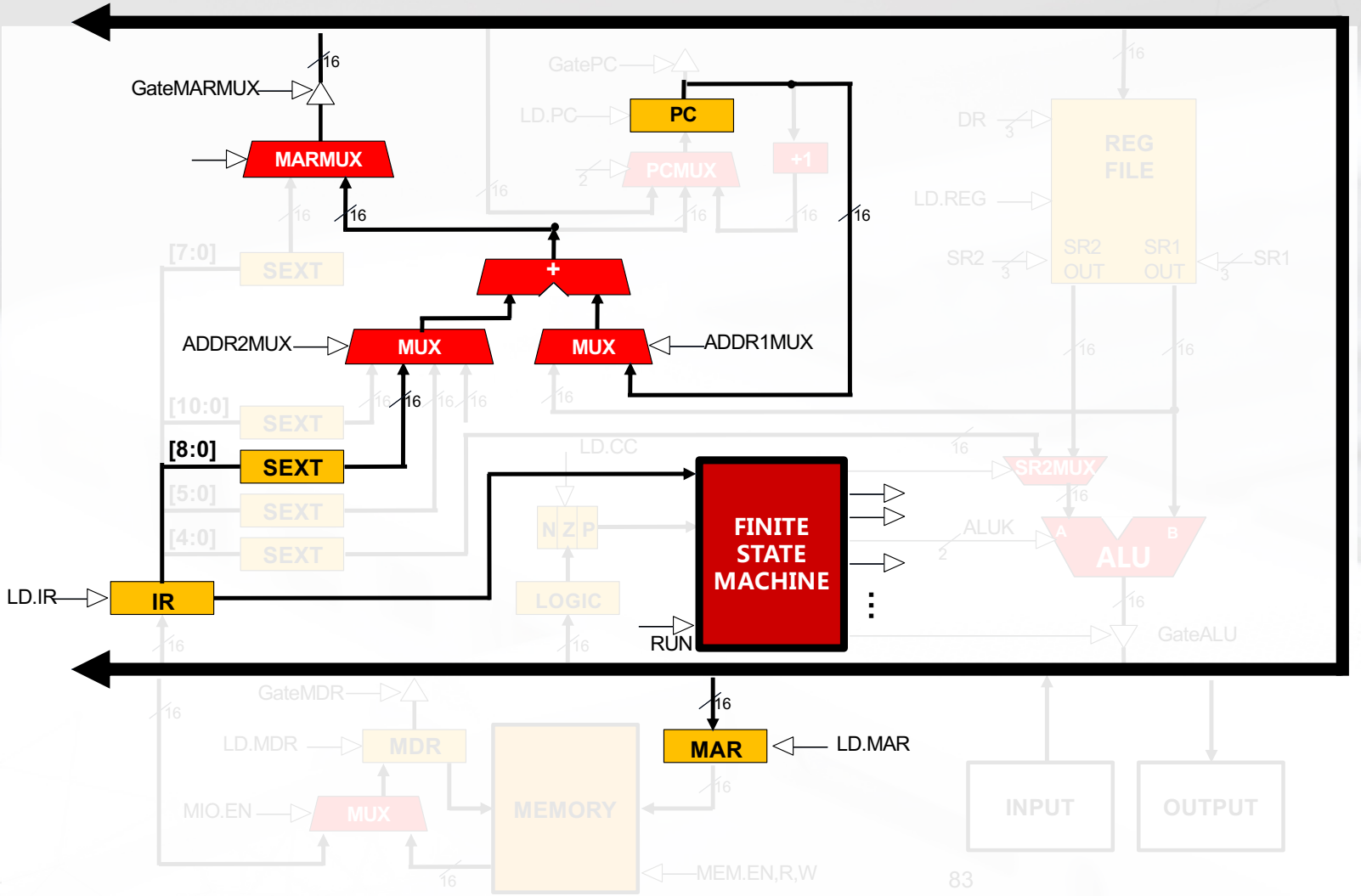
STI (Indirect)



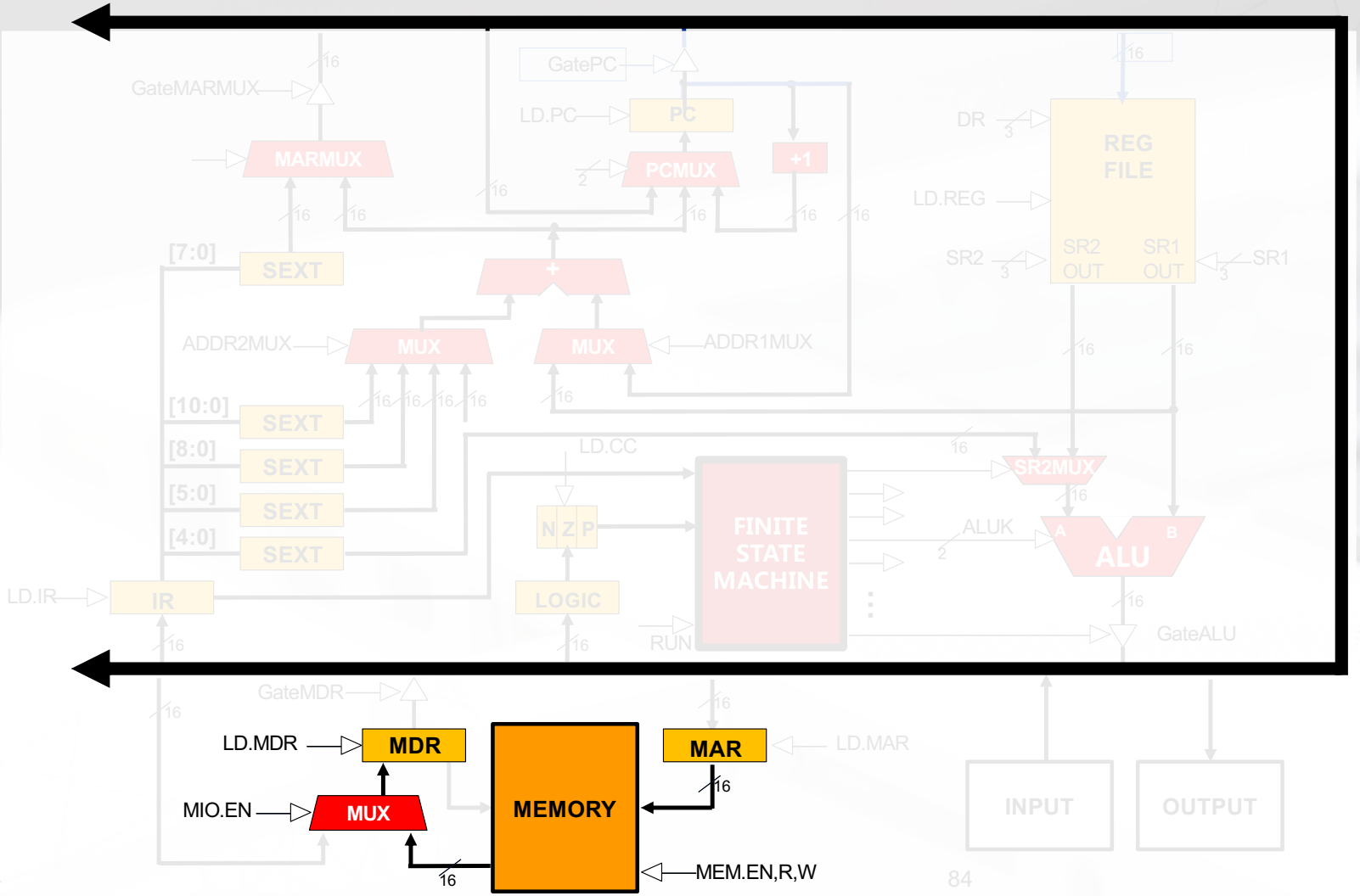


STI (Indirect)

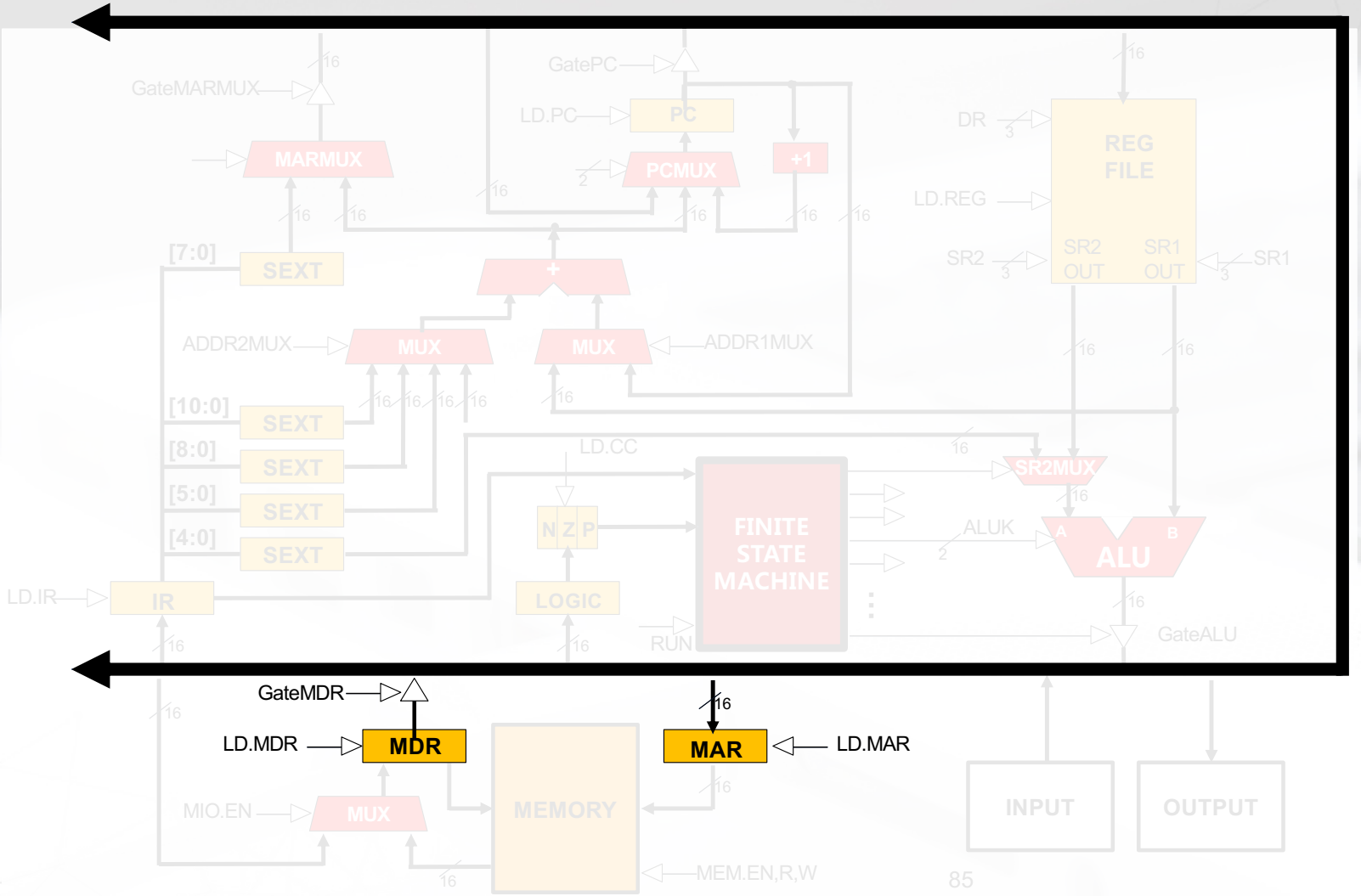




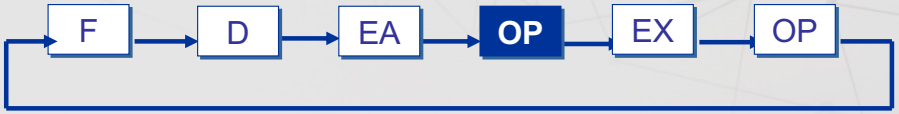
STI (Indirect)

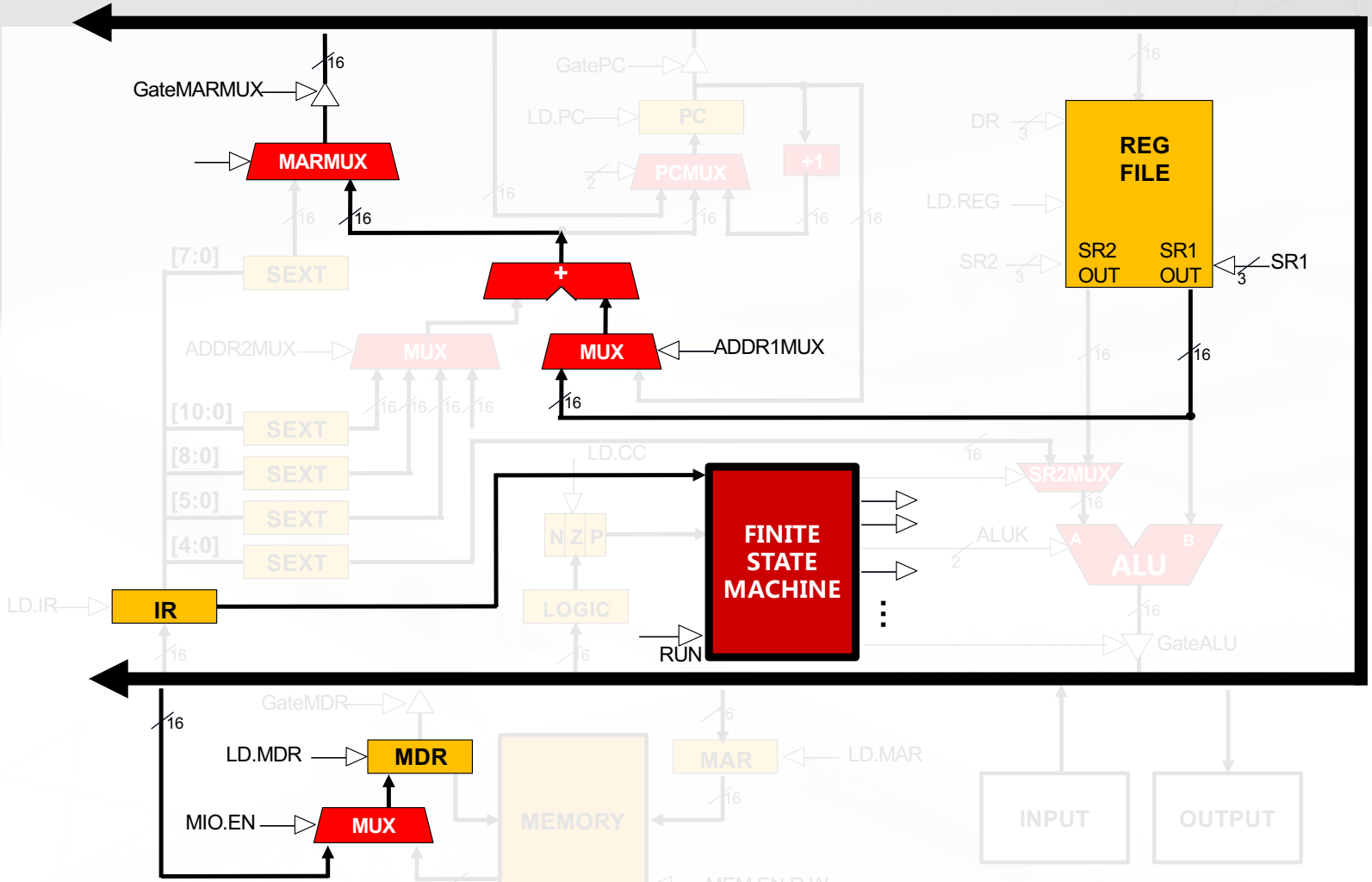


STI (Indirect)

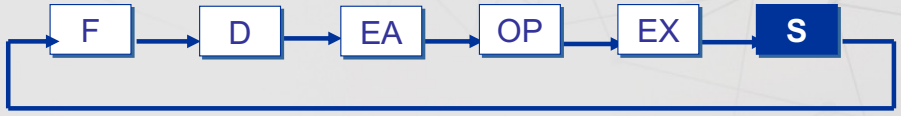


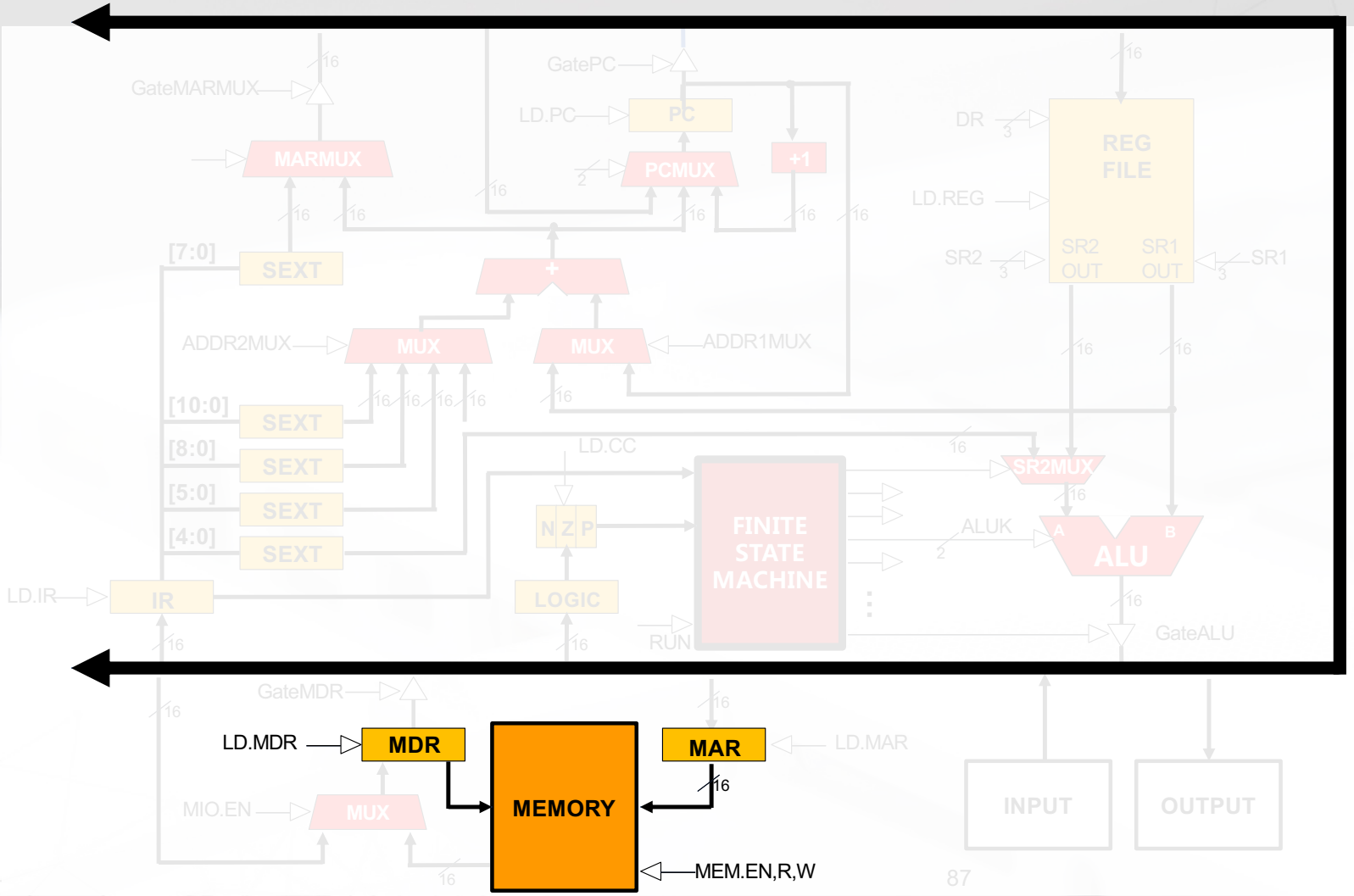
STI (Indirect)





STI (Indirect)





Base + Offset Addressing Mode

With PC-relative mode, can only address data within 256 words of the instruction.

- What about the rest of memory?

Solution #2:

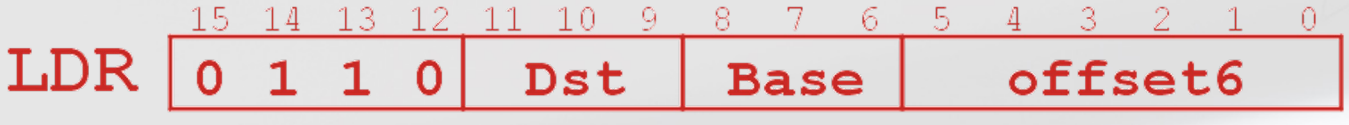
- Use a register to generate a full 16-bit address.

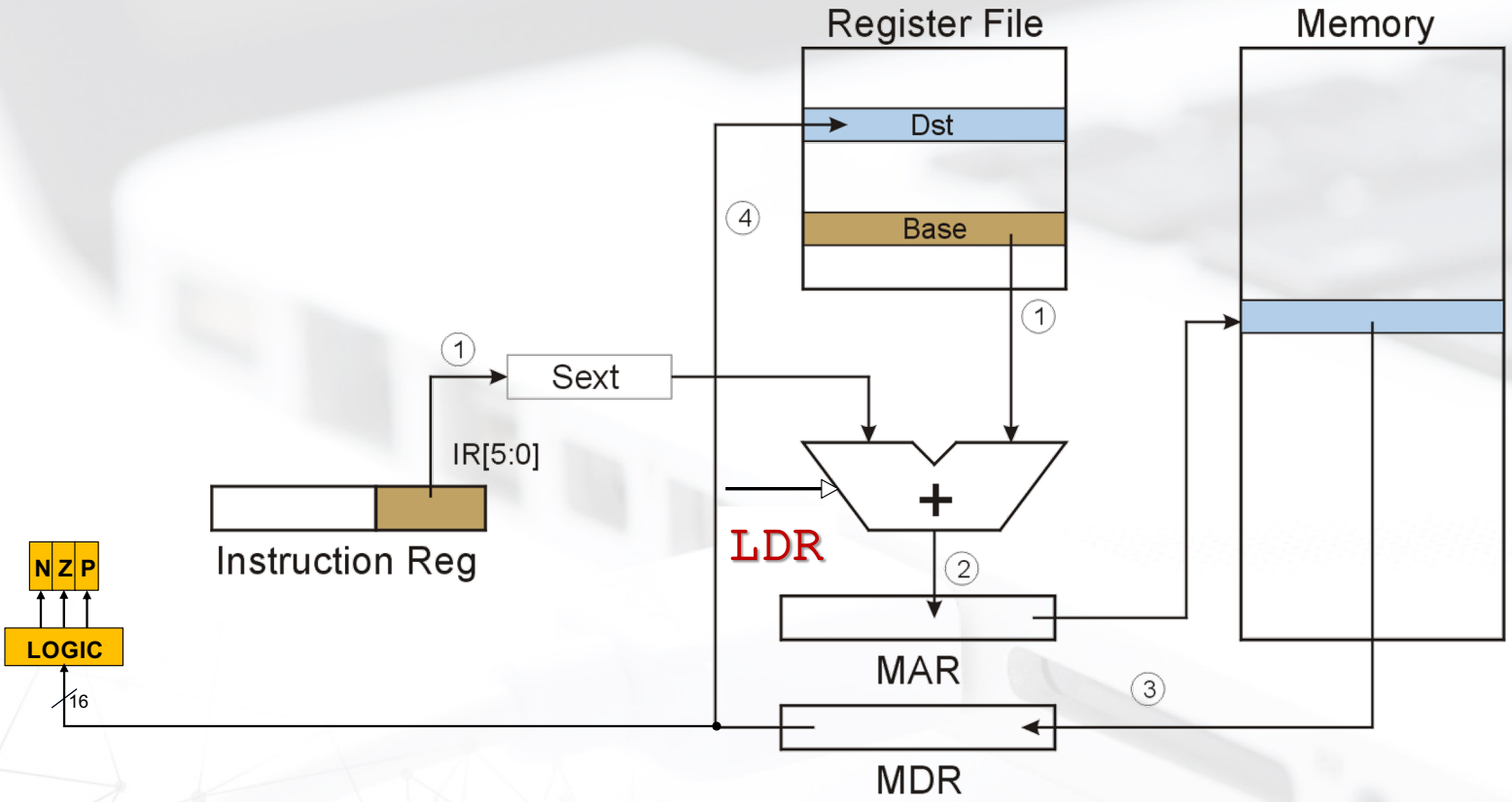
4 bits for opcode, 3 for src/dest register,

3 bits for *base* register -- remaining 6 bits are used as a *signed offset*.

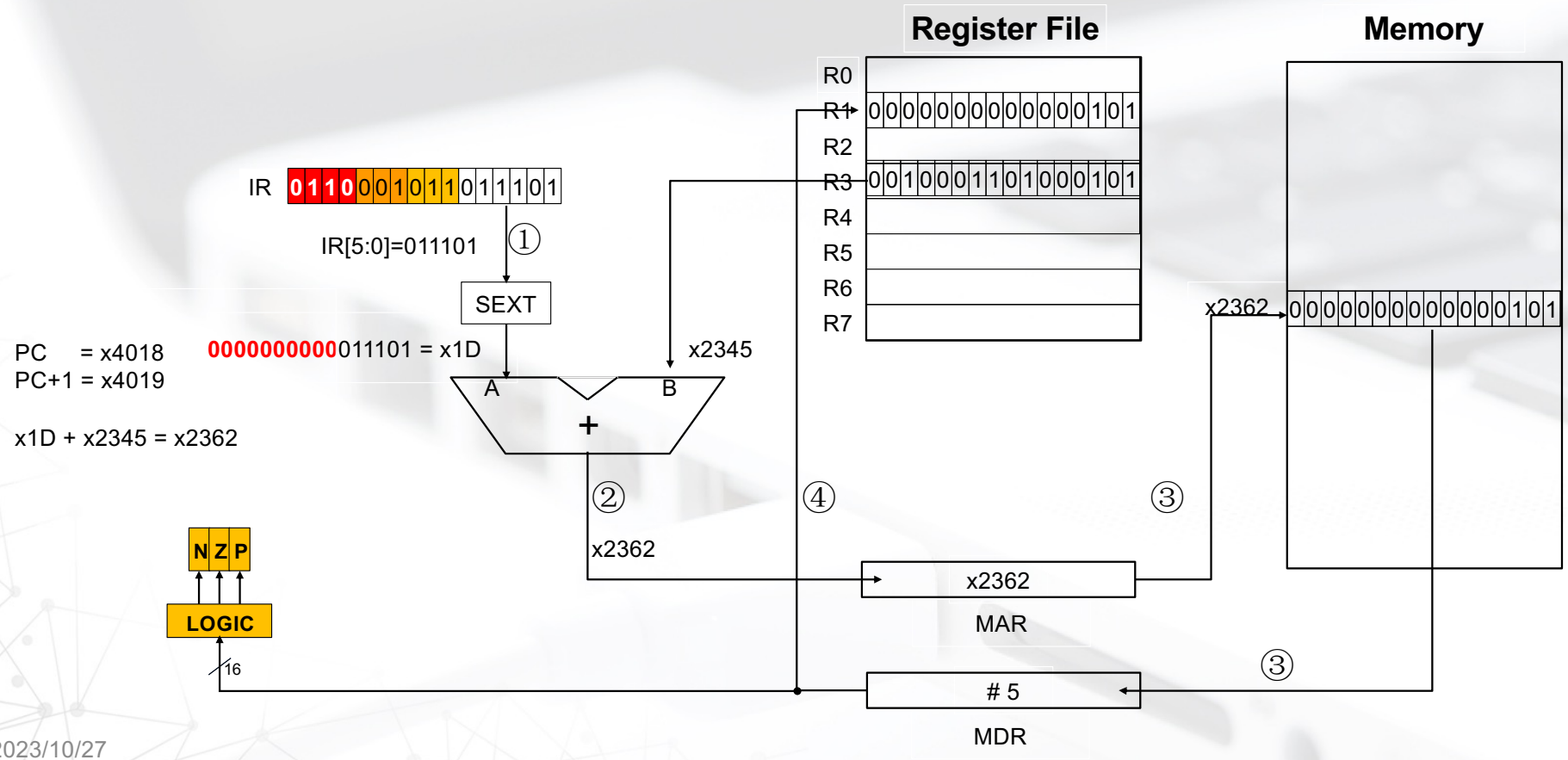
- Offset is sign-extended before adding to base register.

LDR (Base+Offset) LDR DR, BaseR, offset6

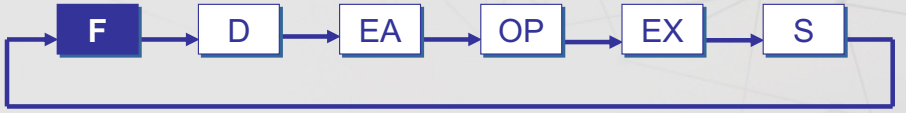


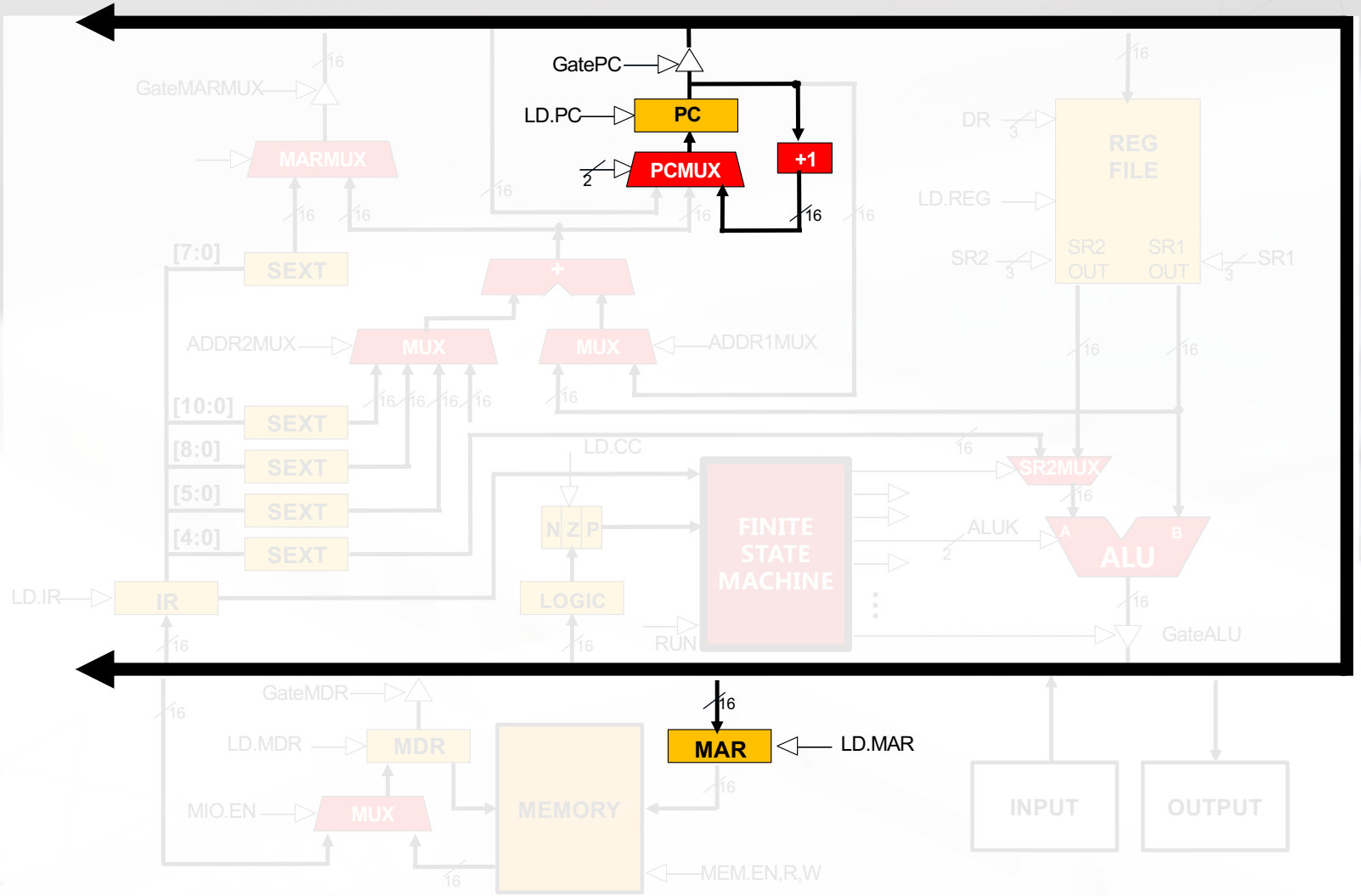


LDR (Base+Offset) : LD R1, R3, x1D

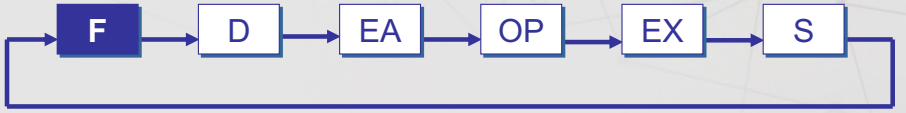


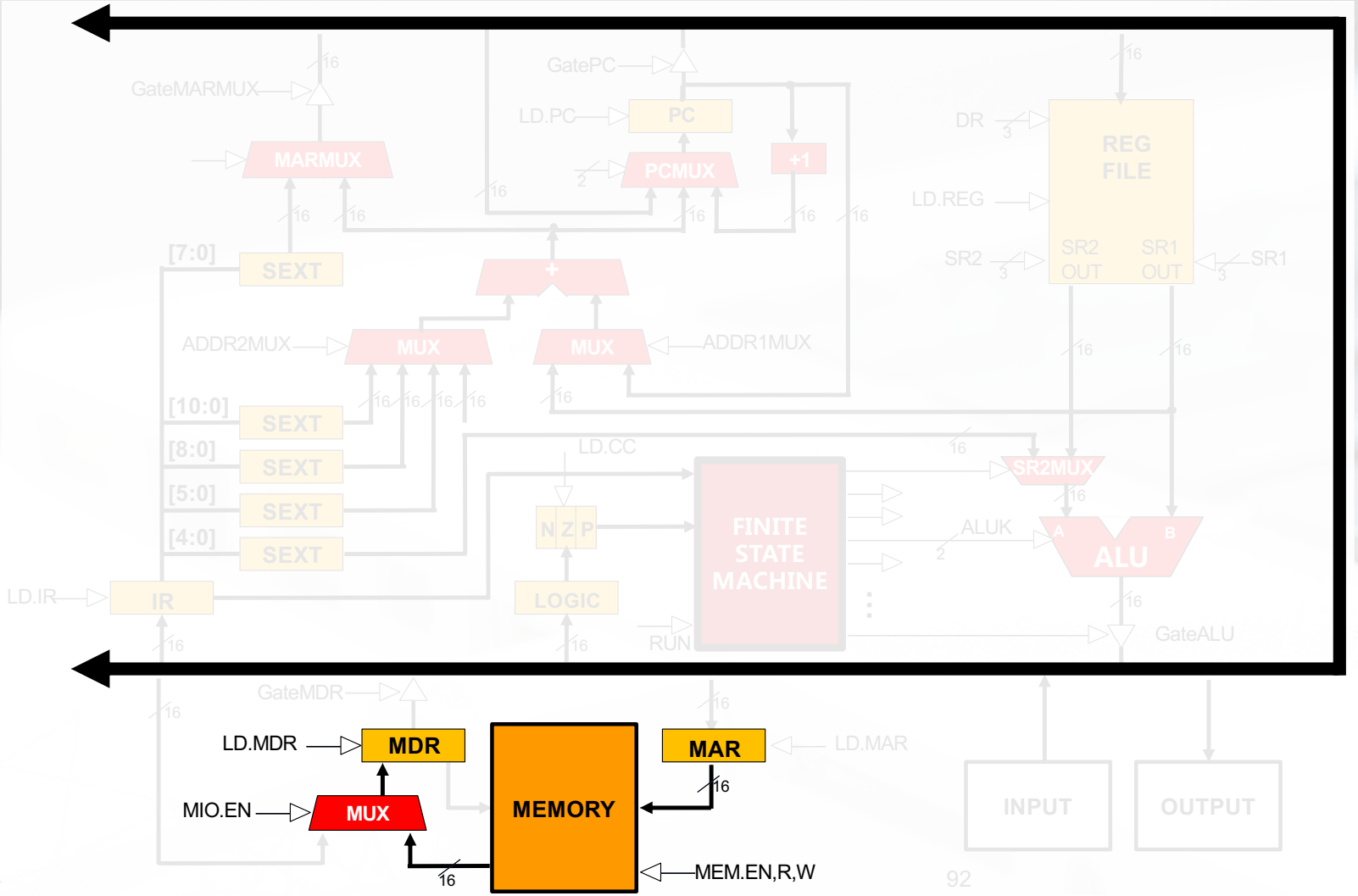
LDR (Base+Offset)



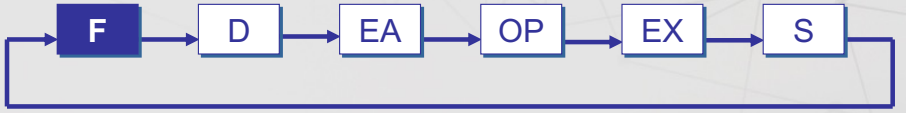


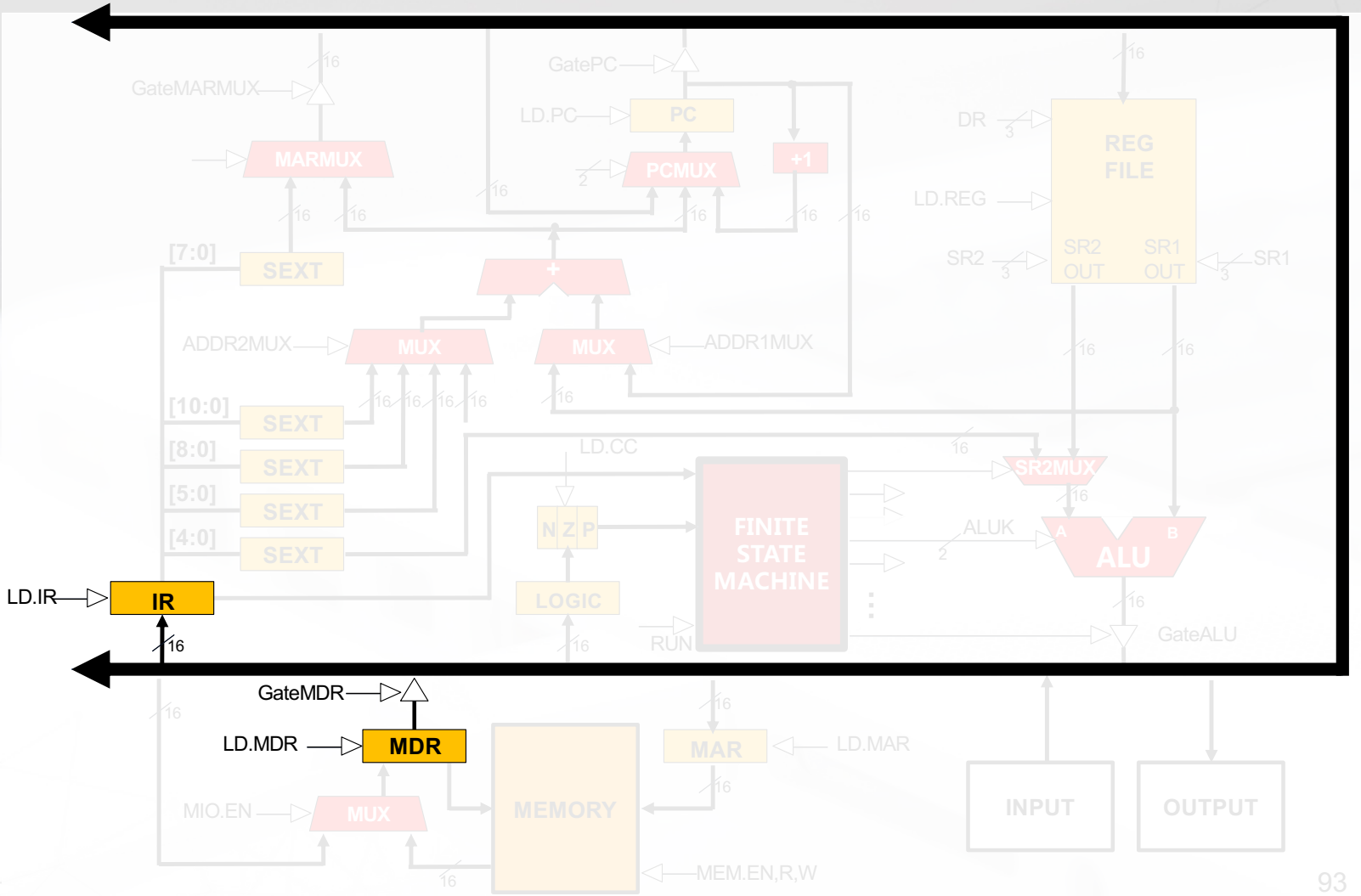
LDR (Base+Offset)



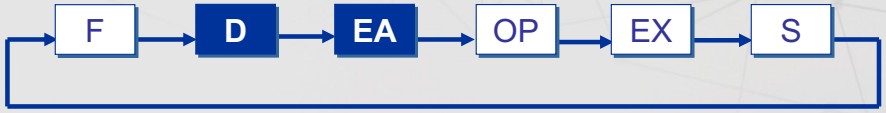


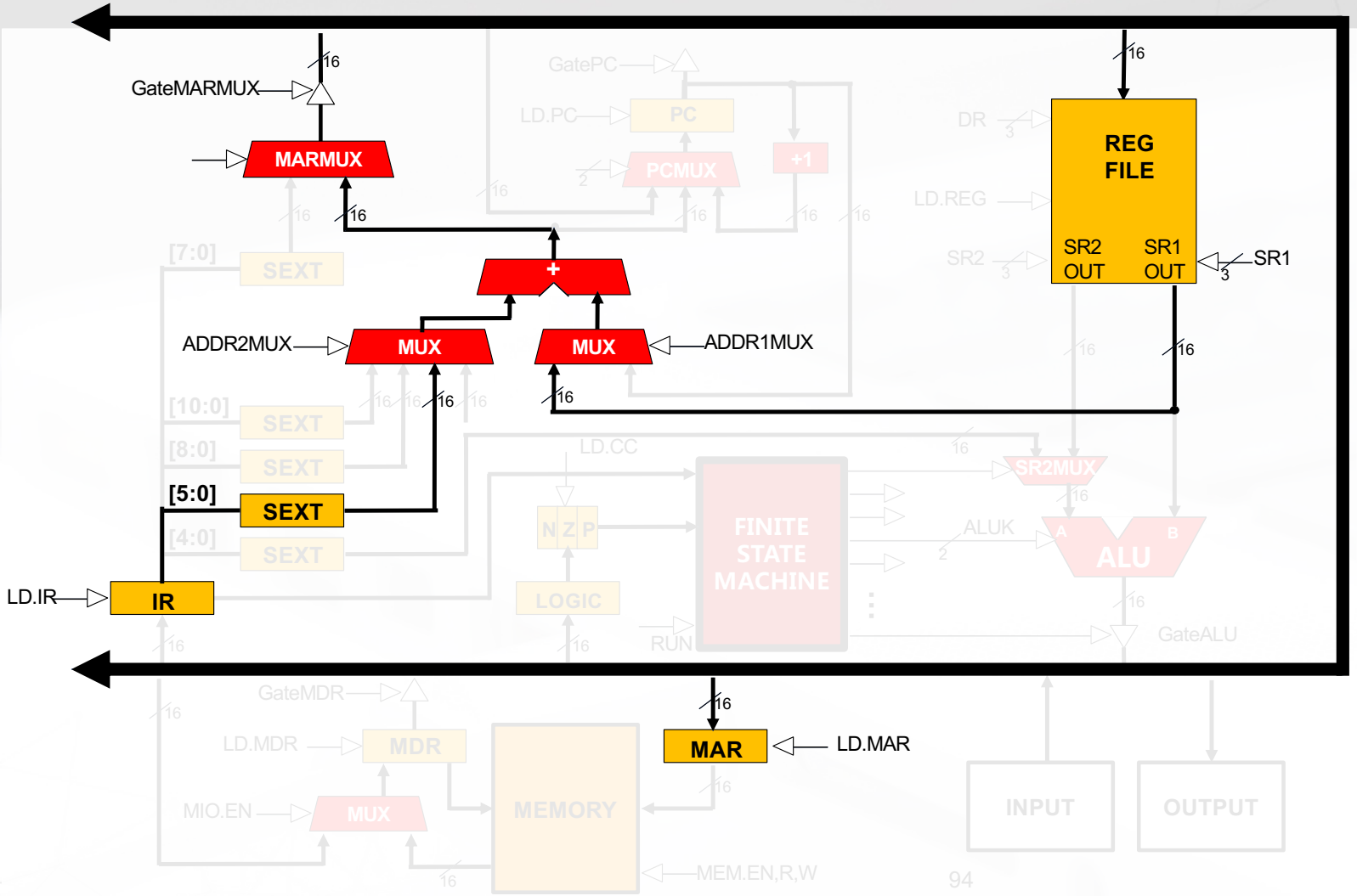
LDR (Base+Offset)



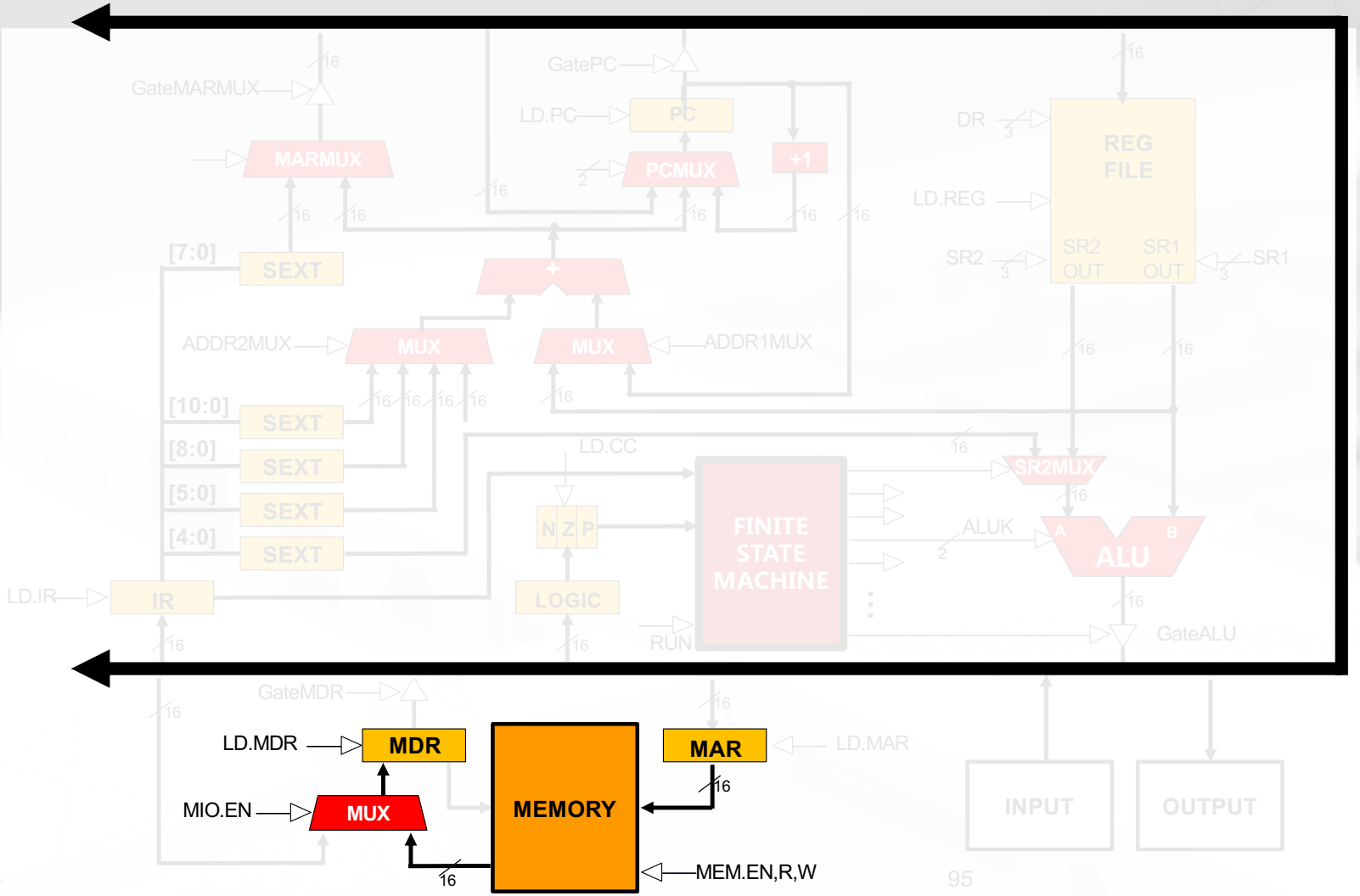


LDR (Base+Offset)

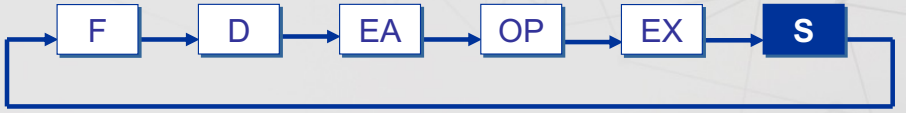


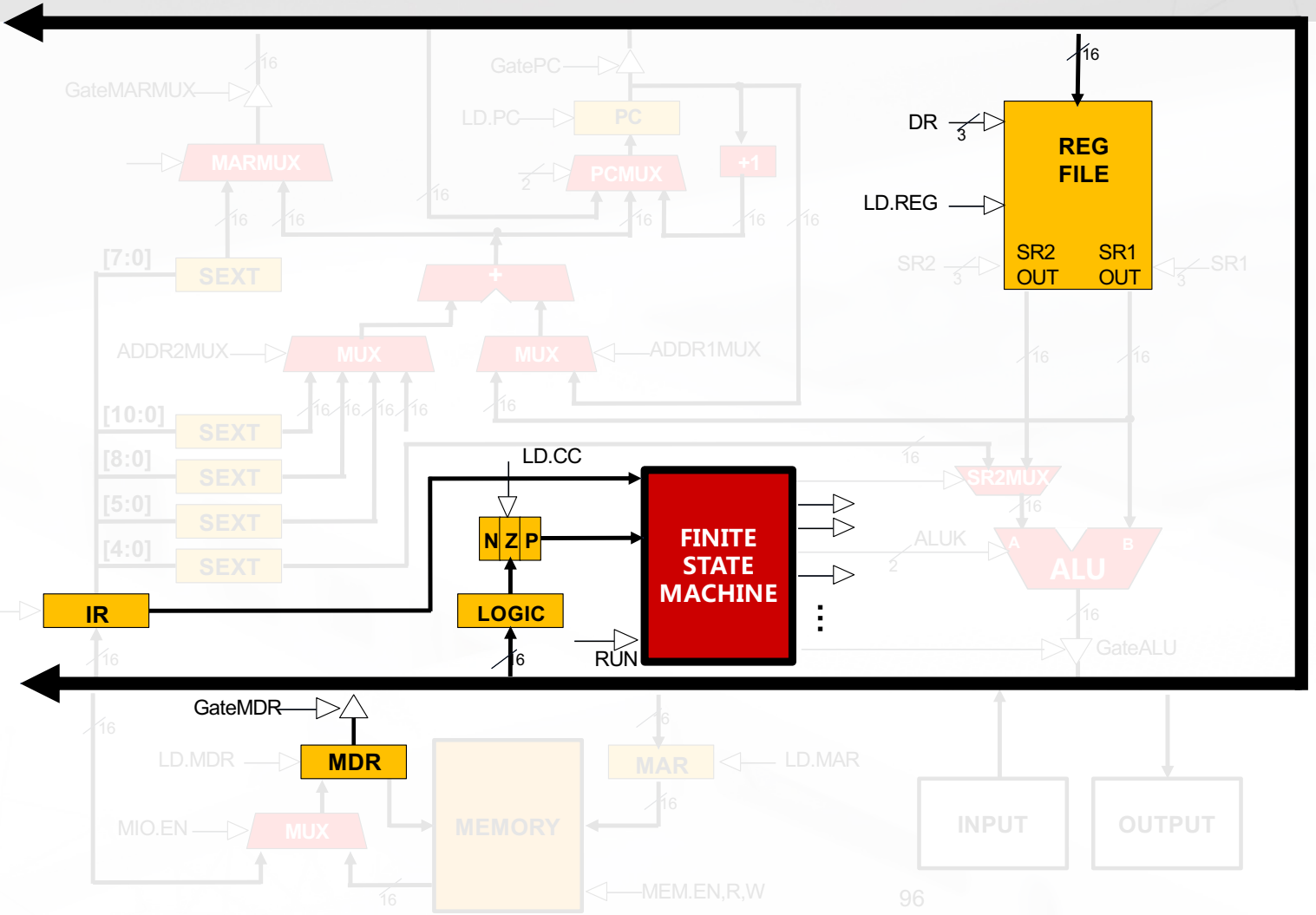


LDR (Base+Offset)

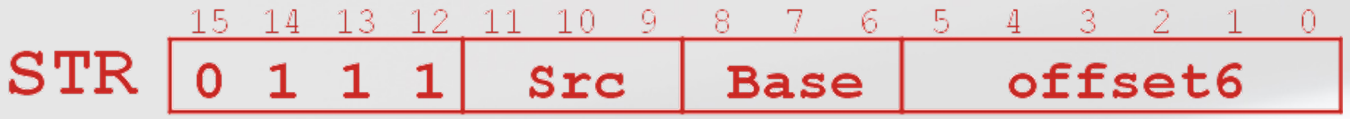


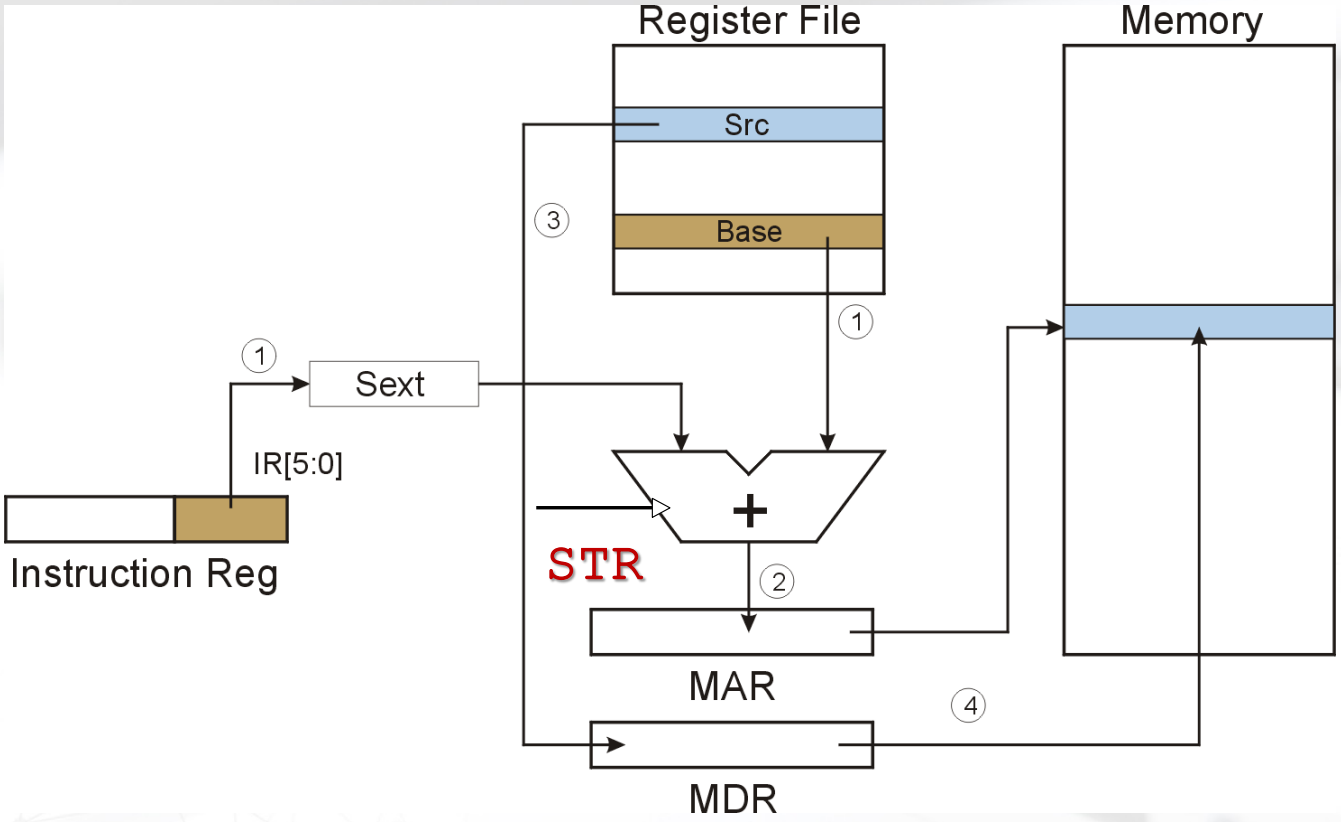
LDR (Base+Offset)



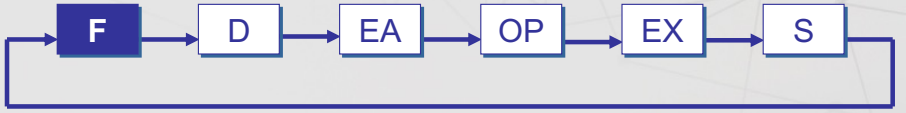


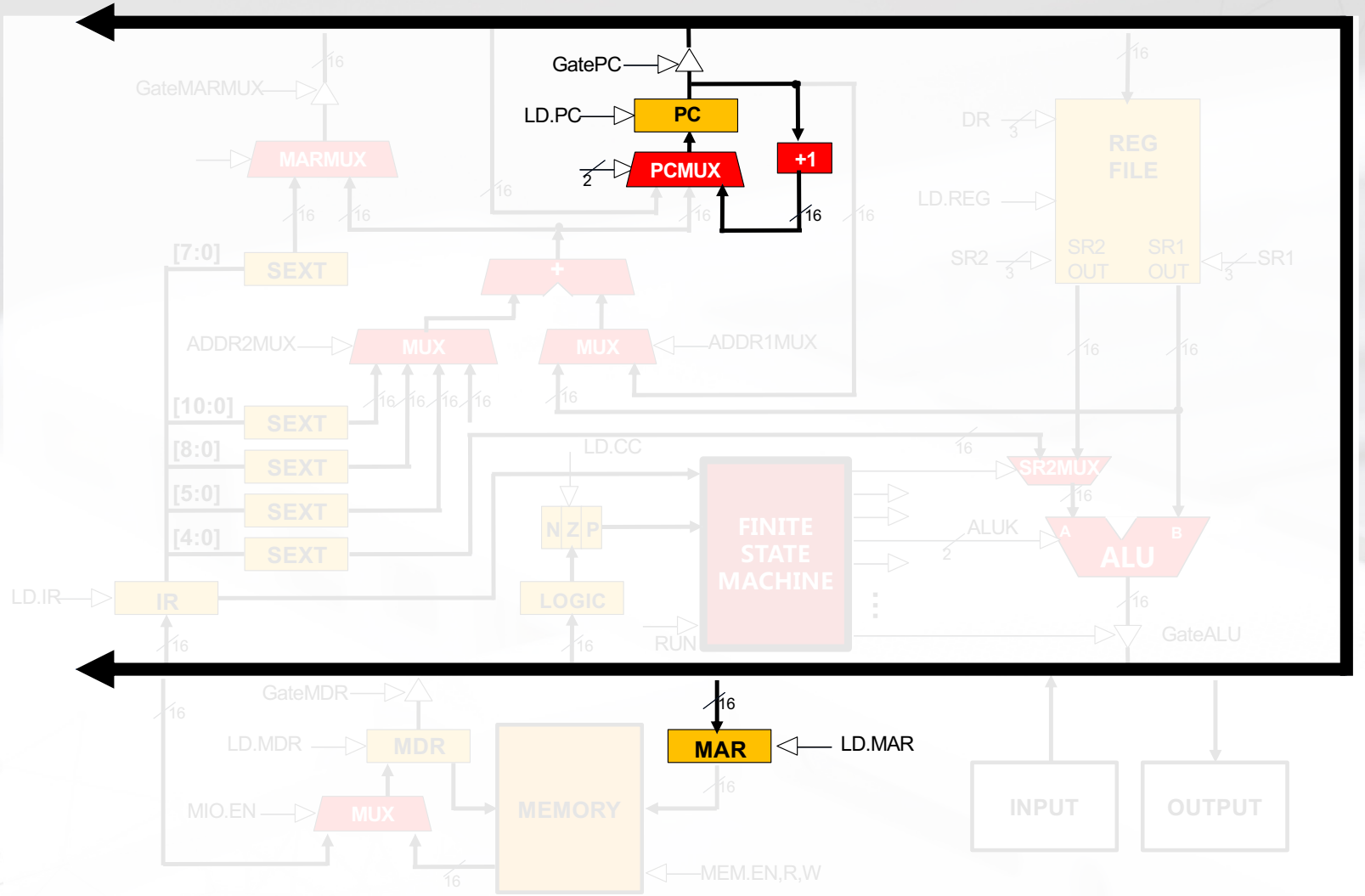
STR (Base+Offset) STR DR, BaseR, offset6



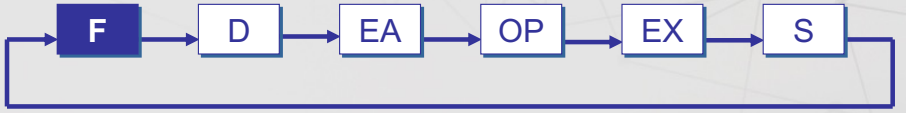


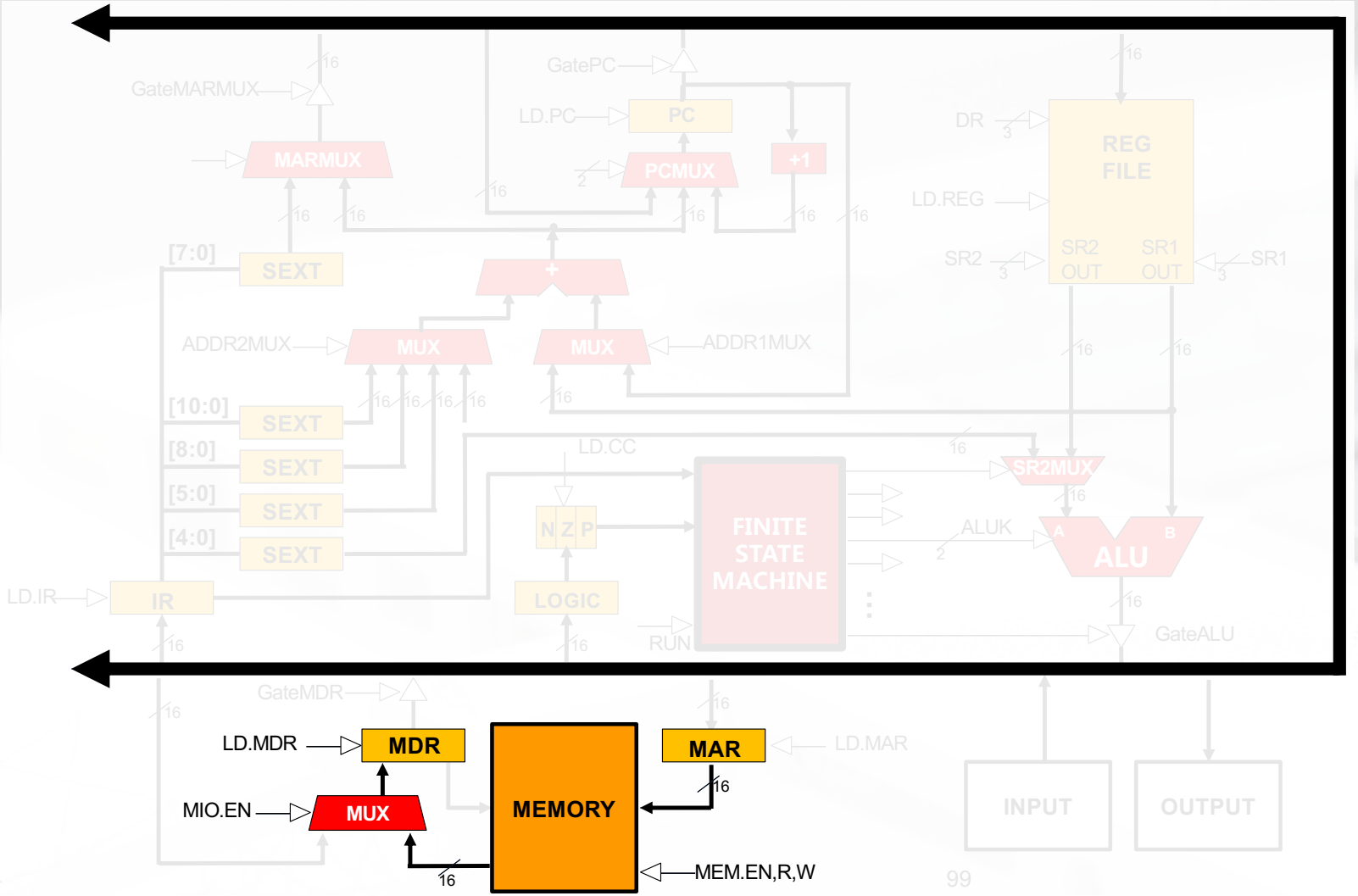
STR (Base+Offset)



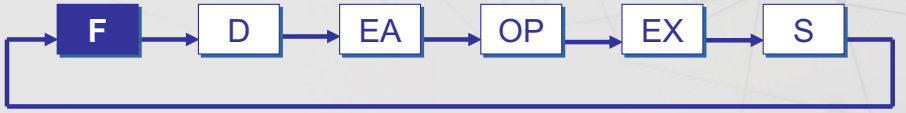


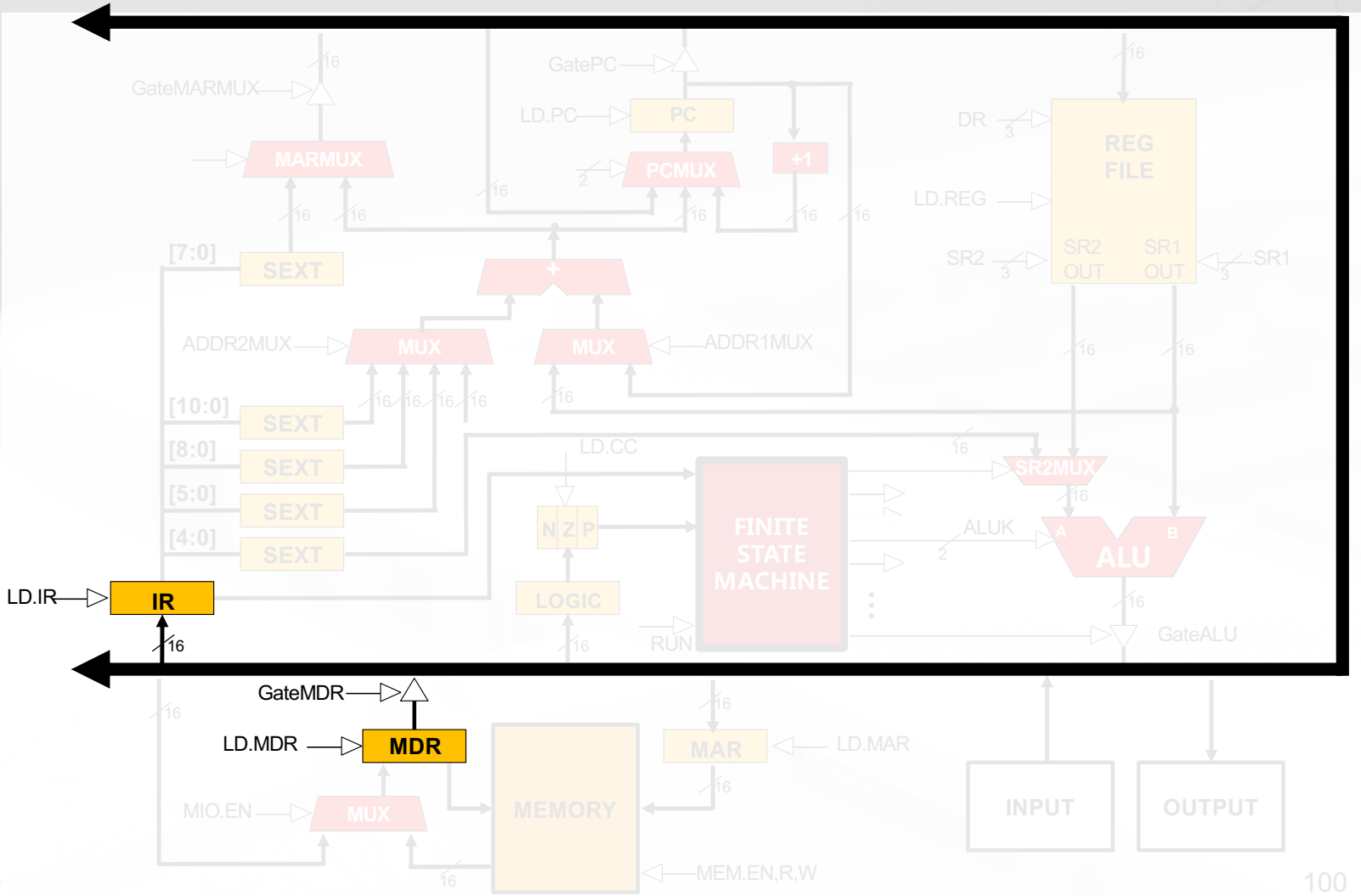
STR (Base+Offset)



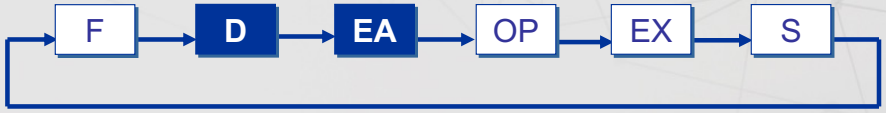


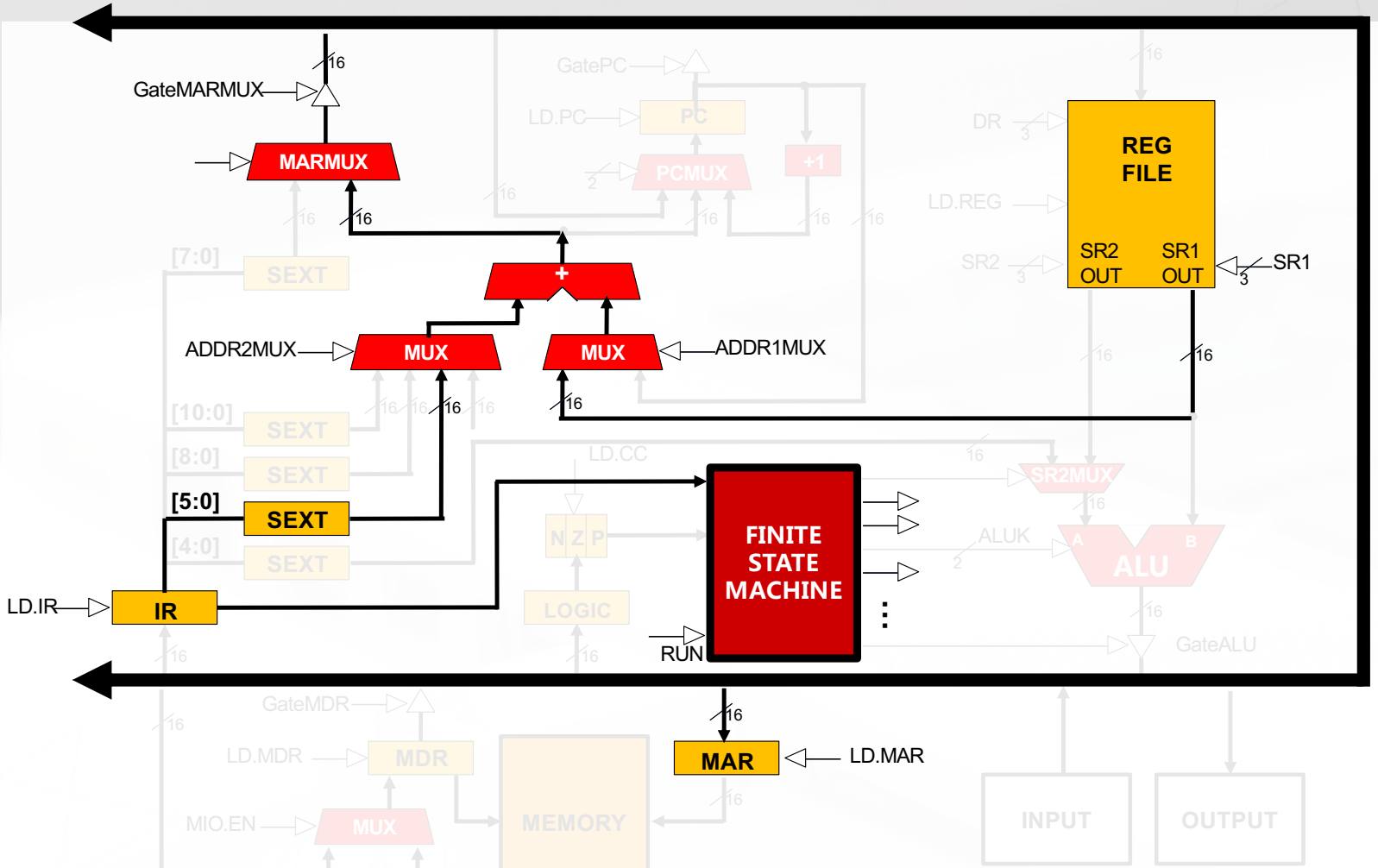
STR (Base+Offset)



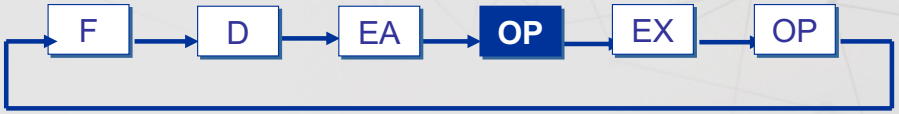


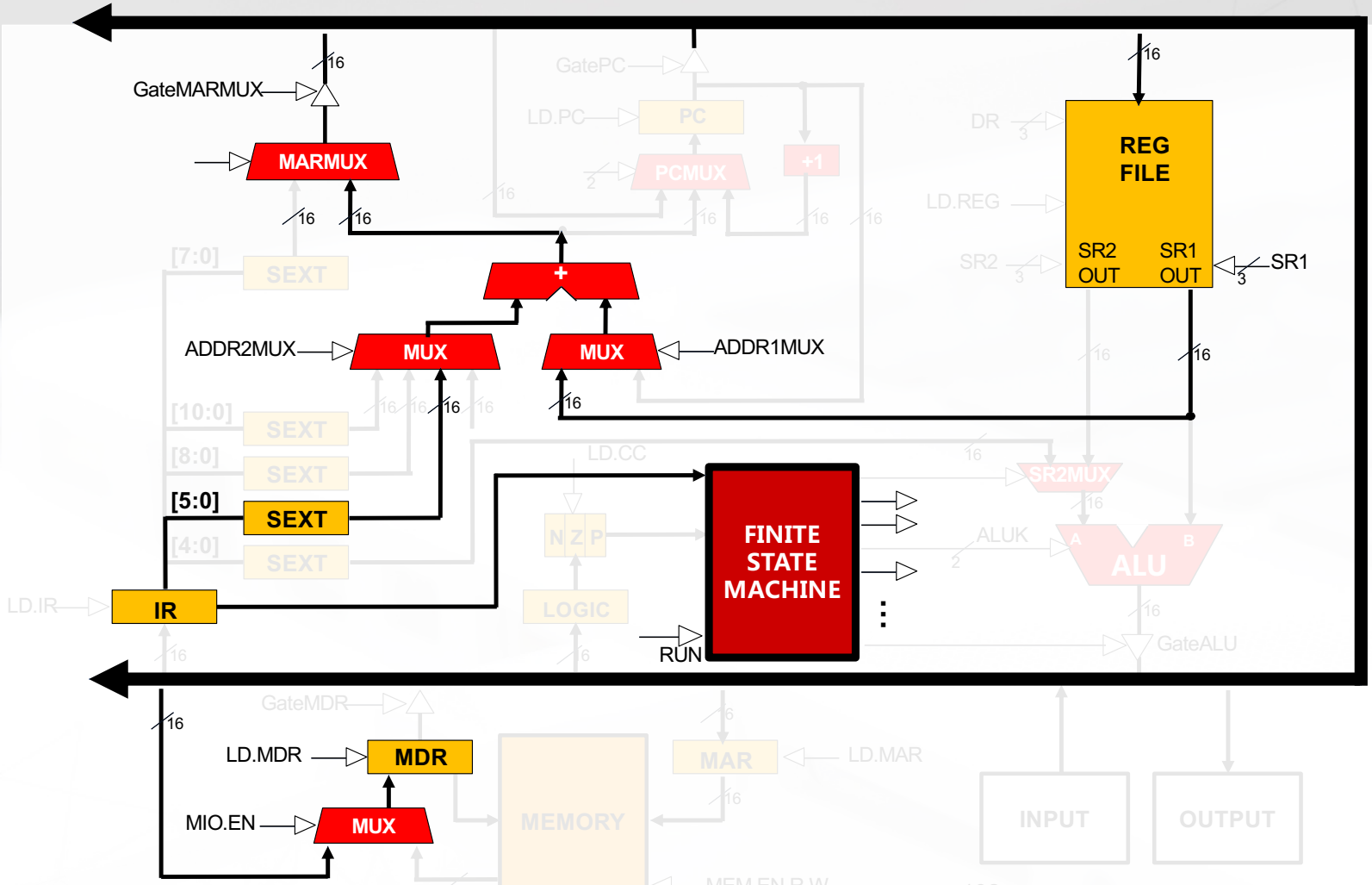
STR (Base+Offset)



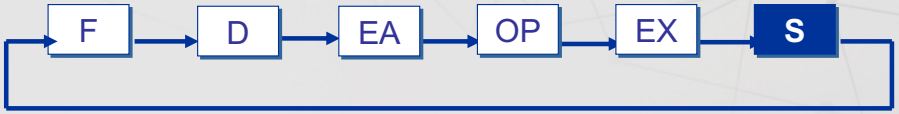


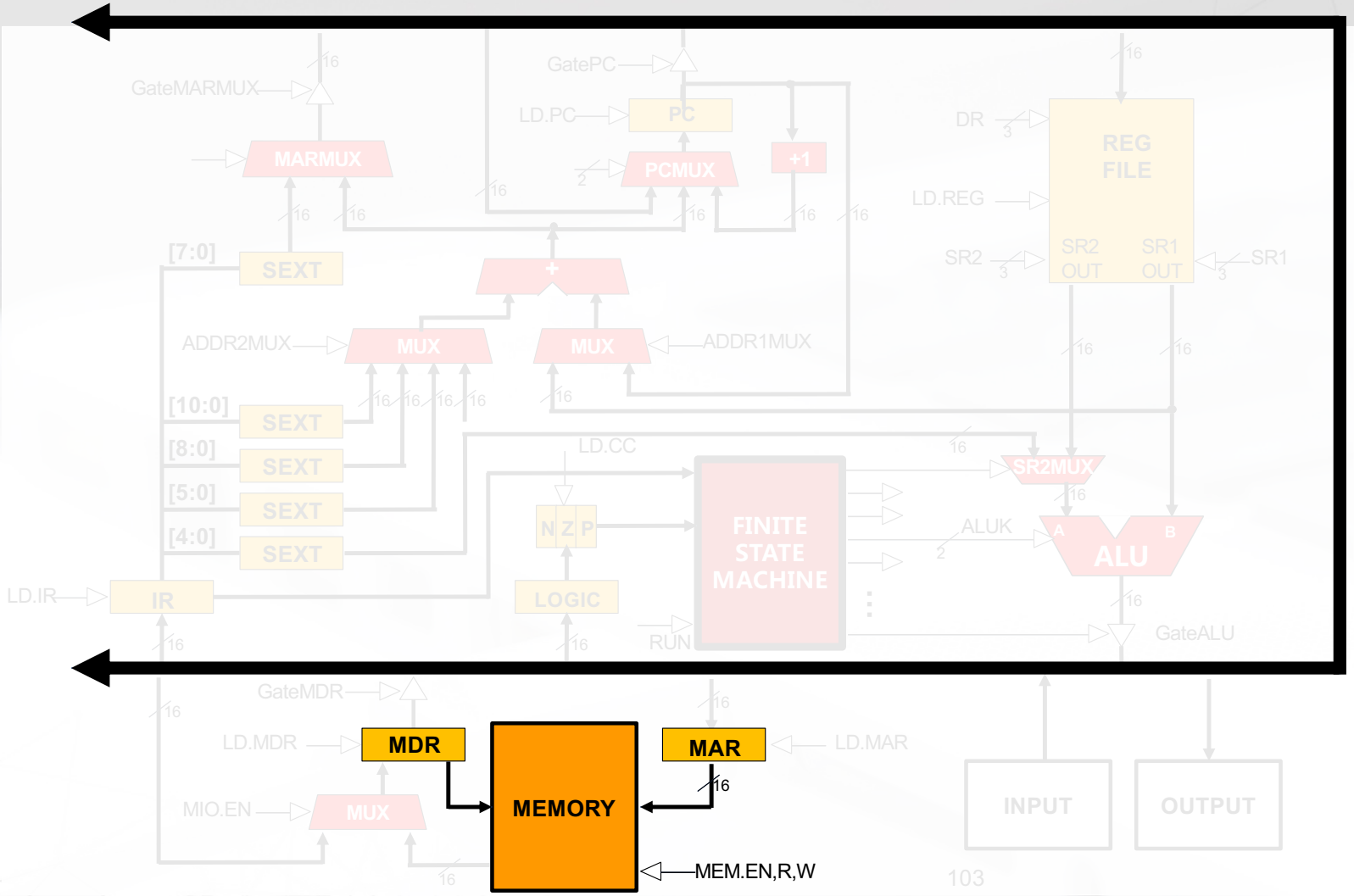
STR (Base+Offset)





STR (Base+Offset)



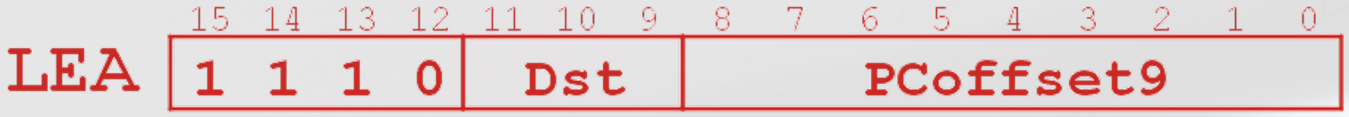


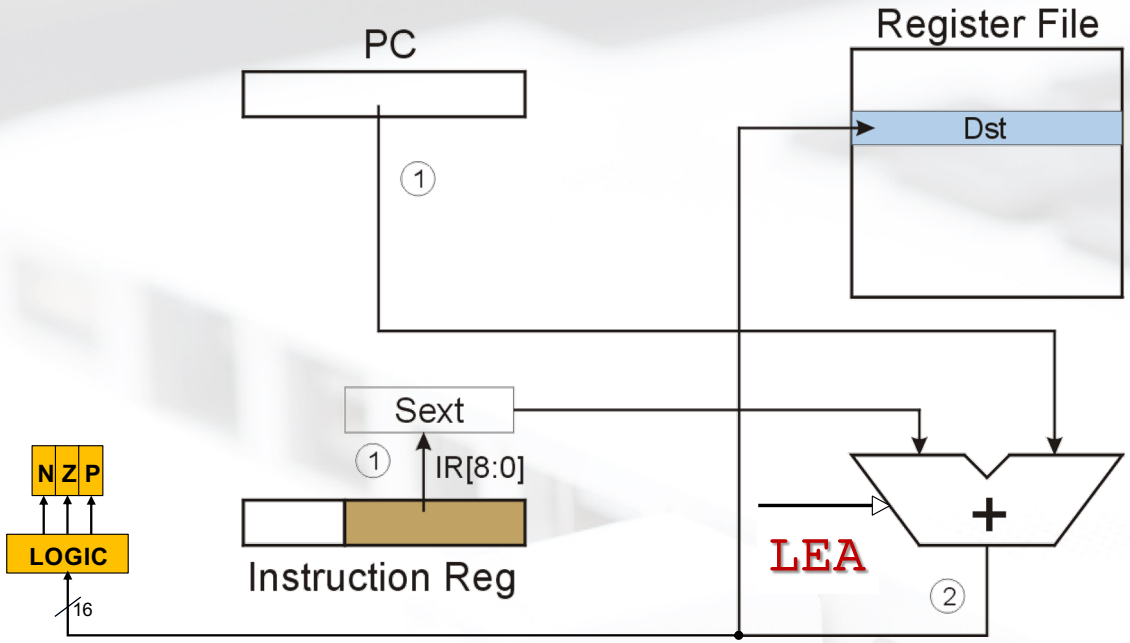
Load Effective Address

Computes address like PC-relative (PC plus signed offset) and stores the result into a register.

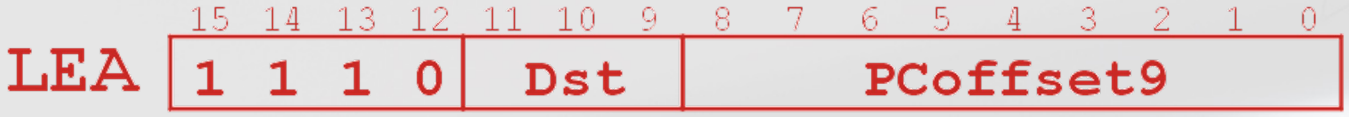
Note: The *address* is stored in the register,
not the contents of the memory location.

LEA (Immediate) LD DR, PCoffset9



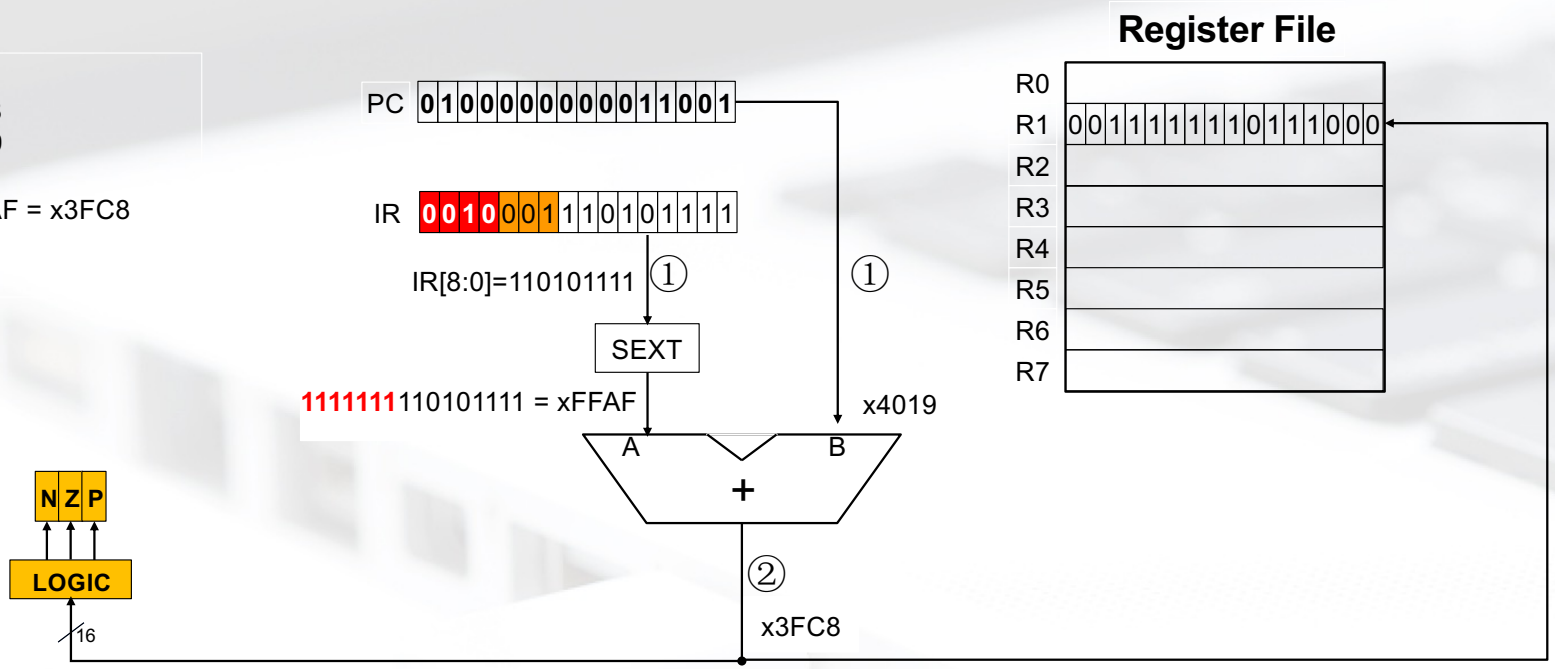


LEA (Immediate): LEA R1, x1AF

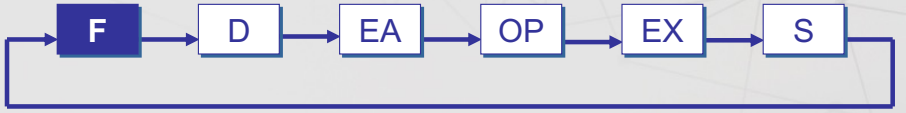


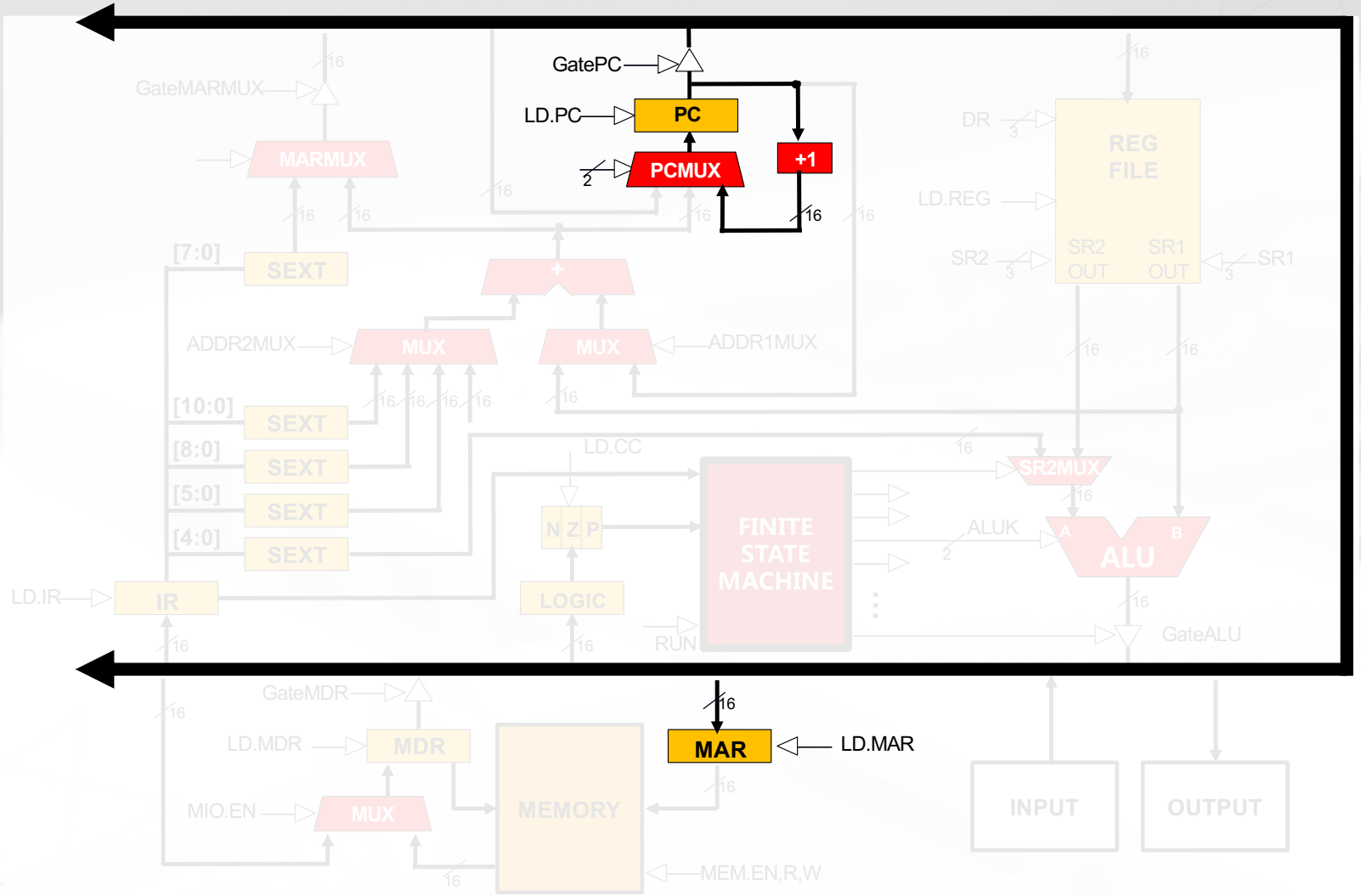
PC = x4018
PC+1 = x4019

X4019 + xFFAF = x3FC8

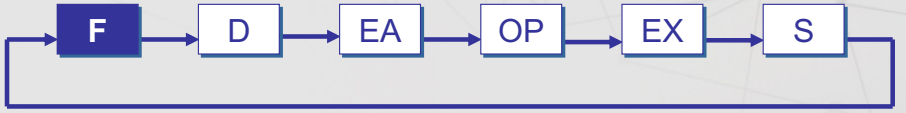


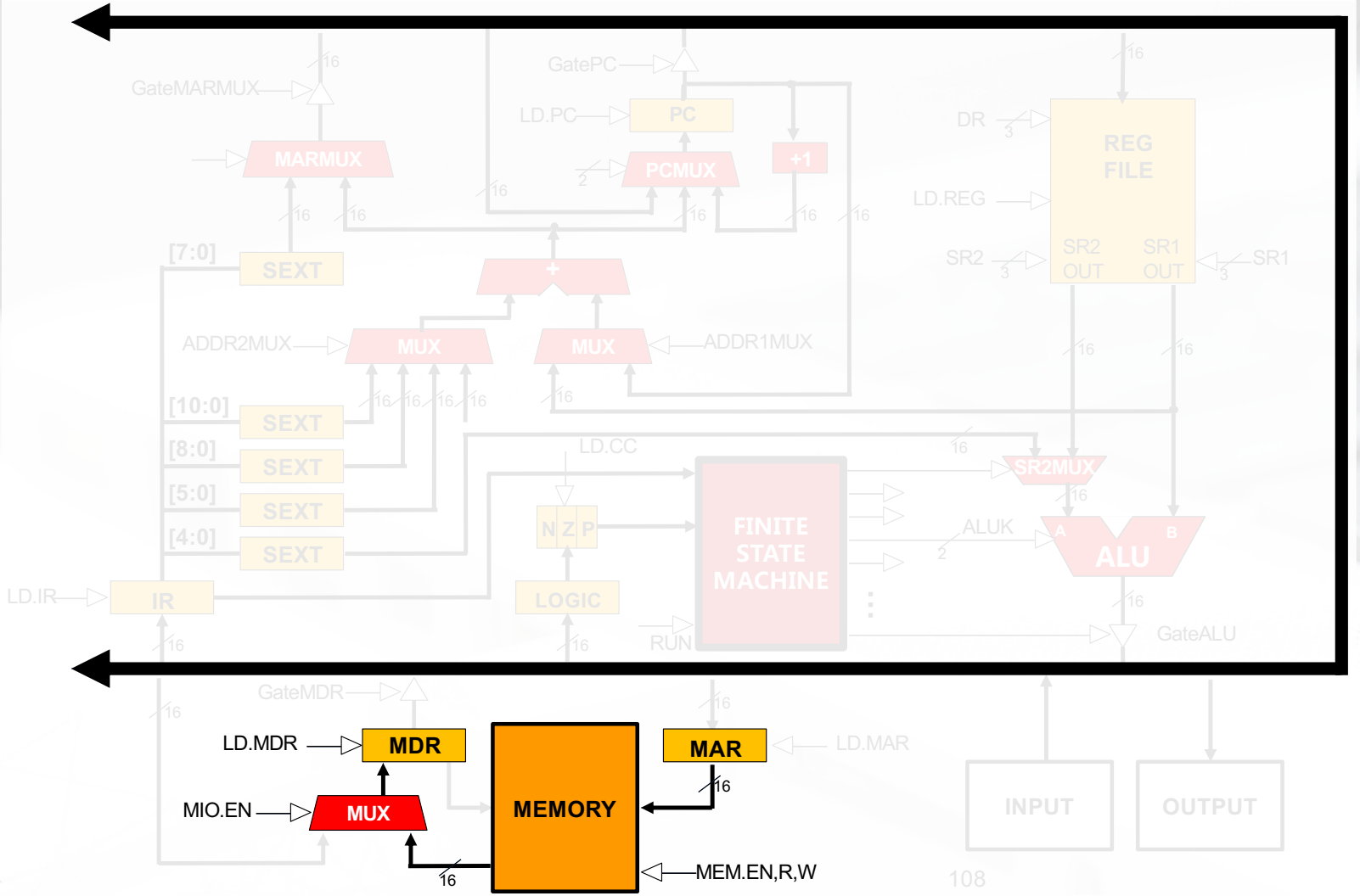
LEA (Immediate)



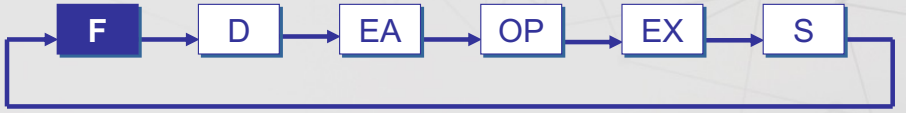


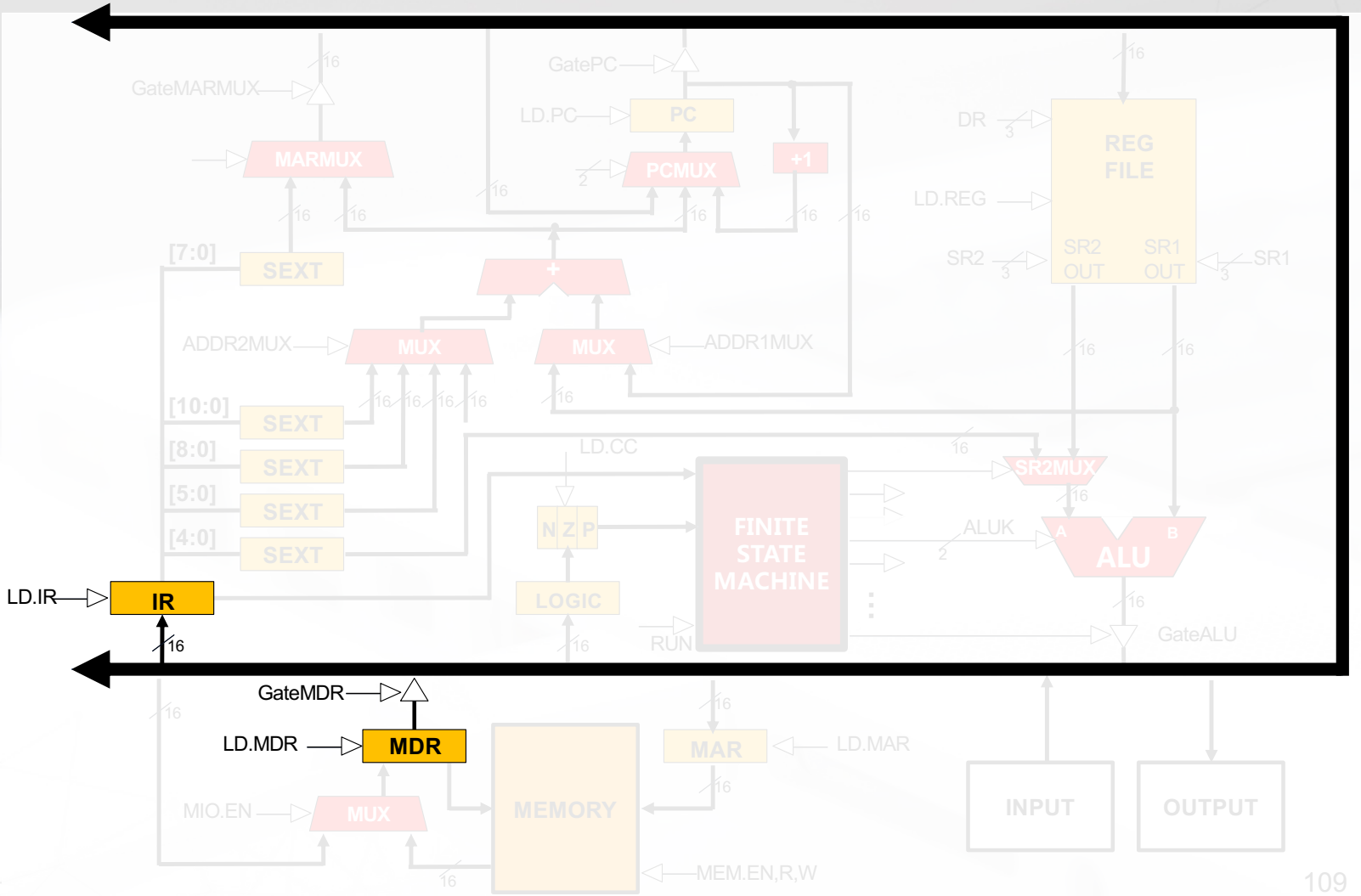
LEA (Immediate)



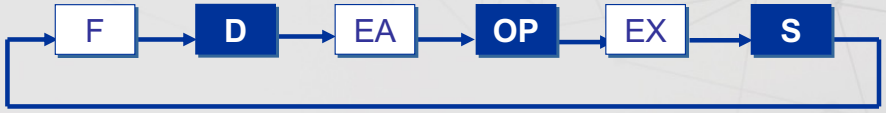


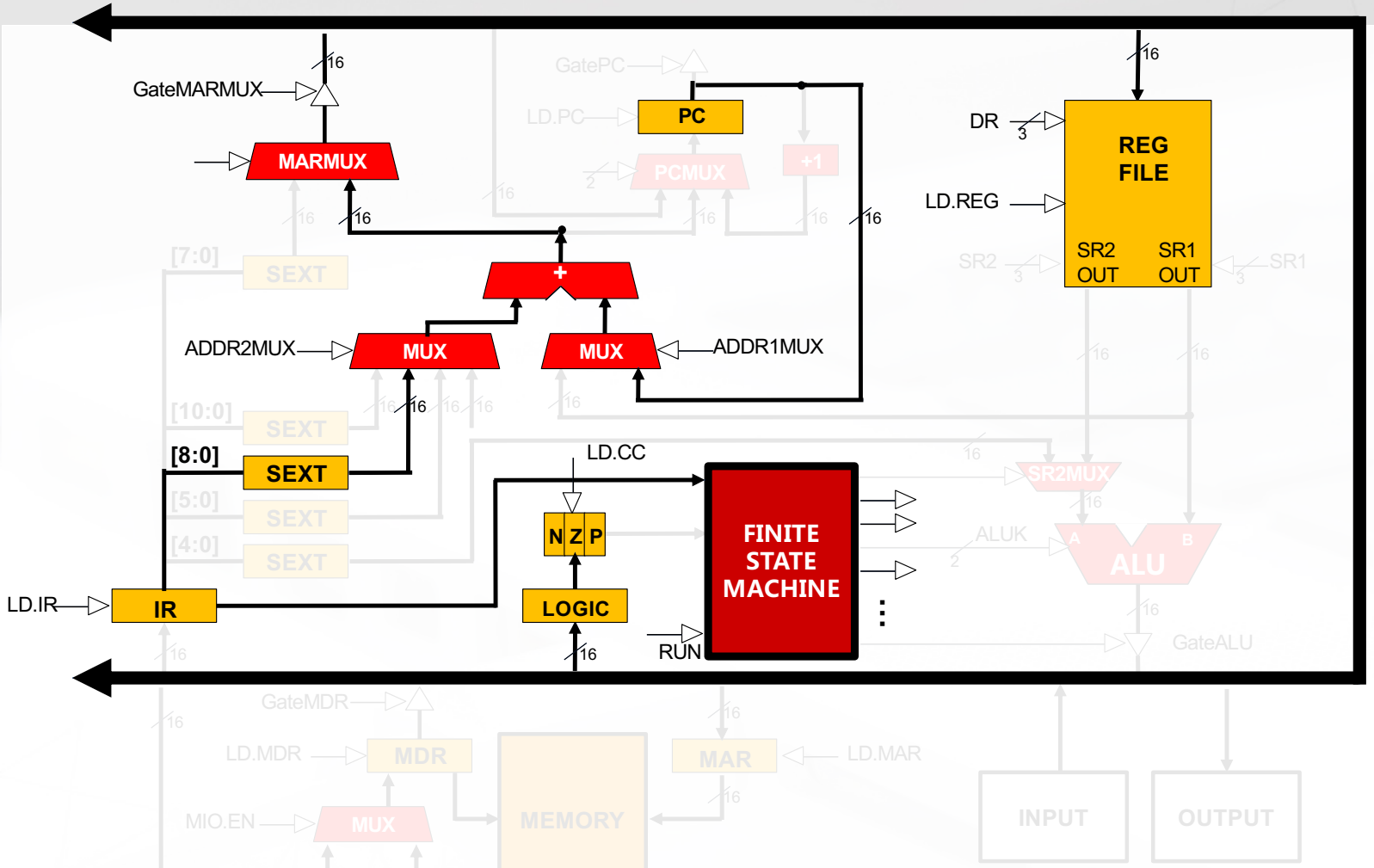
LEA (Immediate)





LEA (Immediate)





Example

Address	Instruction	Comments
x30F6	1 1 1 0 0 0 1 1 1 1 1 1 1 1 0 1	*R1 ← PC – 3 = x30F4*
x30F7	0 0 0 1 0 1 0 0 0 1 1 0 1 1 1 0	*R2 ← R1 + 14 = x3102*
x30F8	0 0 1 1 0 1 0 1 1 1 1 1 1 0 1 1	*M[PC - 5] ← R2; i.e.* *M[x30F4] ← x3102*
x30F9	0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0	*R2 ← 0*
x30FA	0 0 0 1 0 1 0 0 1 0 1 0 0 1 0 1	*R2 ← R2 + 5 = 5*
x30FB	0 1 1 1 0 1 0 0 0 1 0 0 1 1 1 0	*M[R1+14] ← R2; i.e.* *M[x3102] ← 5*
x30FC	1 0 1 0 0 1 1 1 1 1 1 1 0 1 1 1	*R3 ← M[M[PC-9]]* *= M[M[x30F4]]* *= M[x3102]* *= 5*

opcode

LC-3 Data Path After Load/Store Instruction