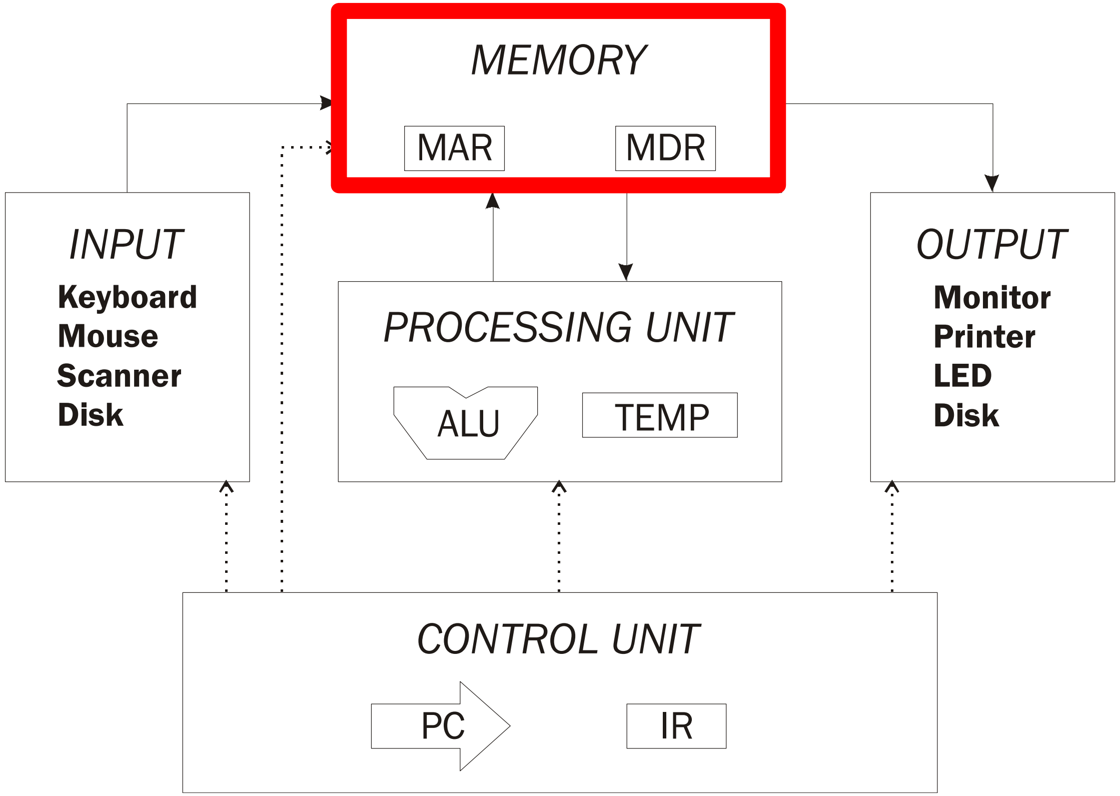
310-2202 โครงสร้างของระบบคอมพิวเตอร์ (Computer Organization)

Topic 5: The LC-3 Instructions: Language of the Computer

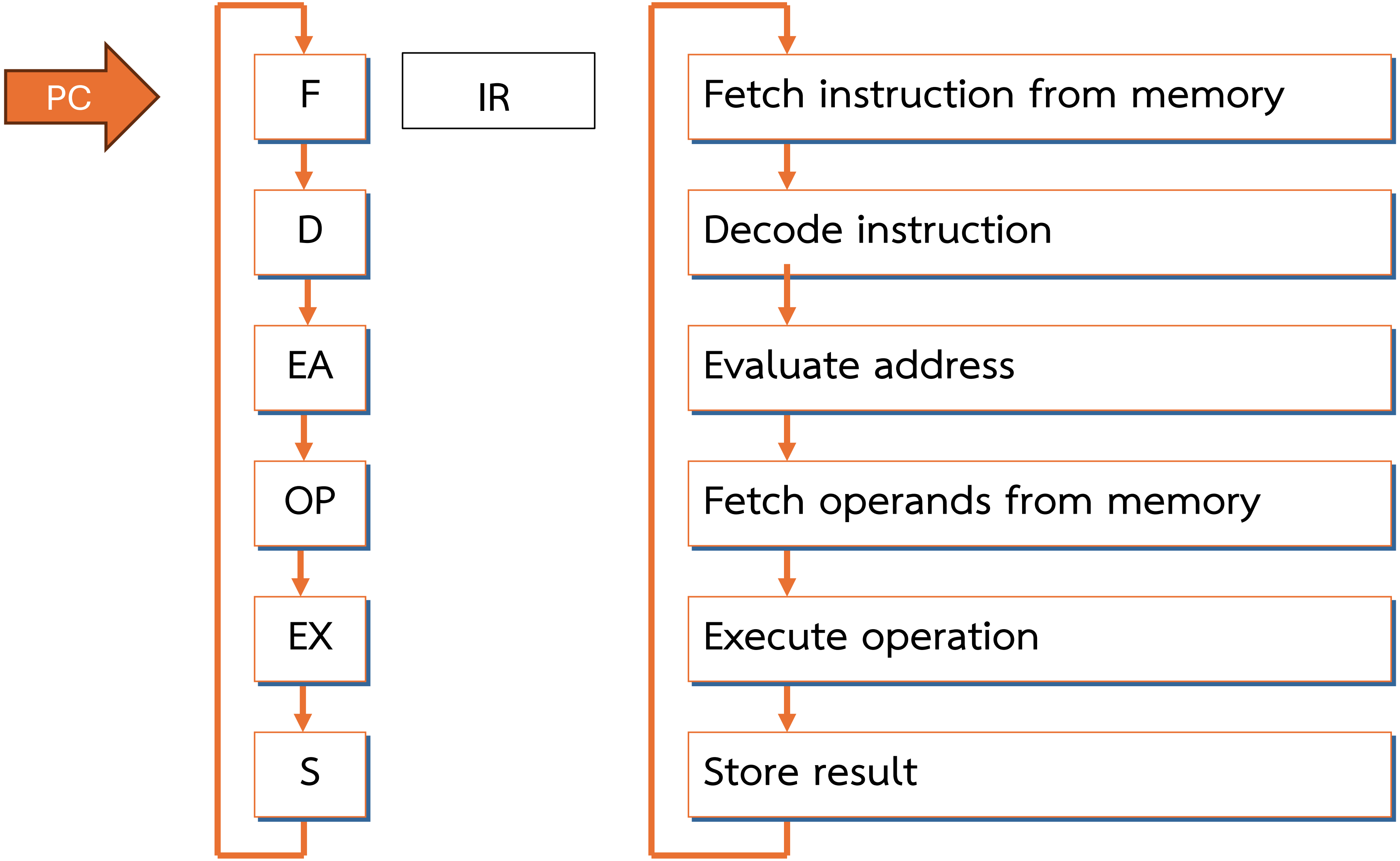
Damrongrit Setsirichok

Topic

- LC-3 ISA Overview
- LC-3 Operate Instructions and Data Path



Instruction Processing: State Transition



PC

F

IR

D

EA

OP

EX

S

Fetch instruction from memory

Decode instruction

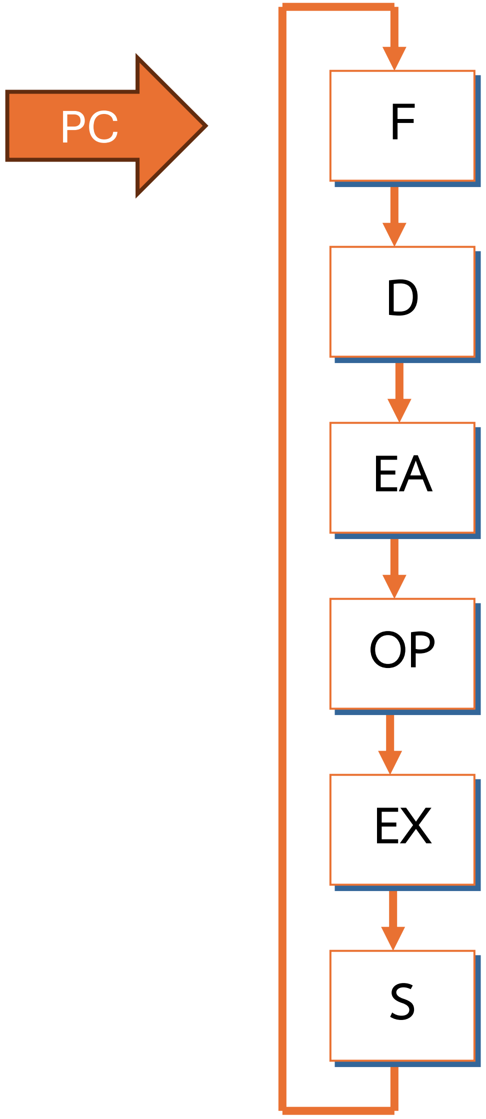
Evaluate address

Fetch operands from memory

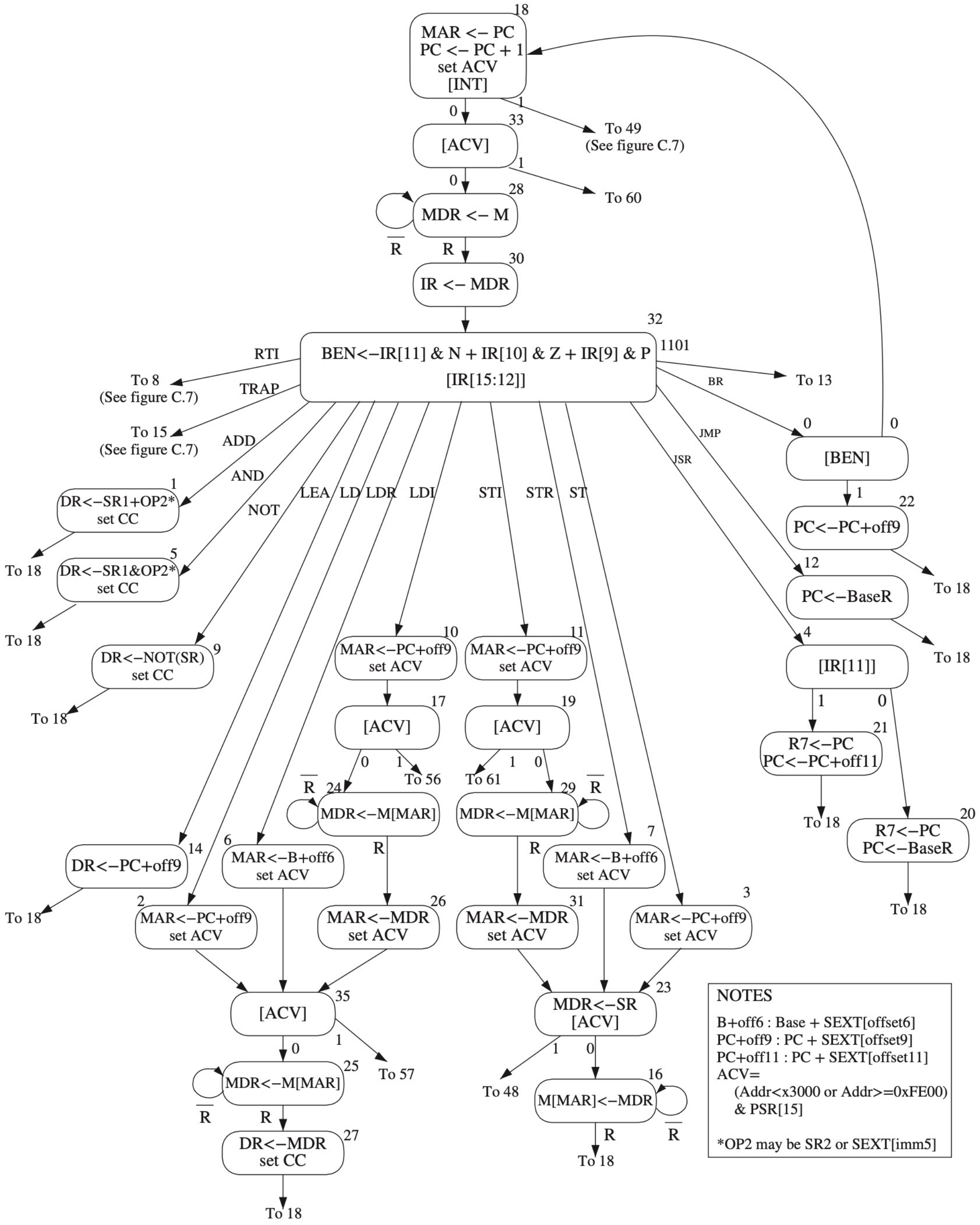
Execute operation

Store result

Instruction Processing: Finite State Automata



702 **appendix C** The Microarchitecture of the LC-3

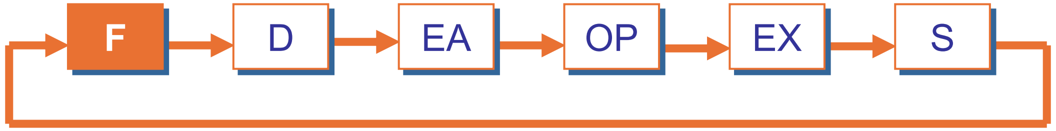


NOTES

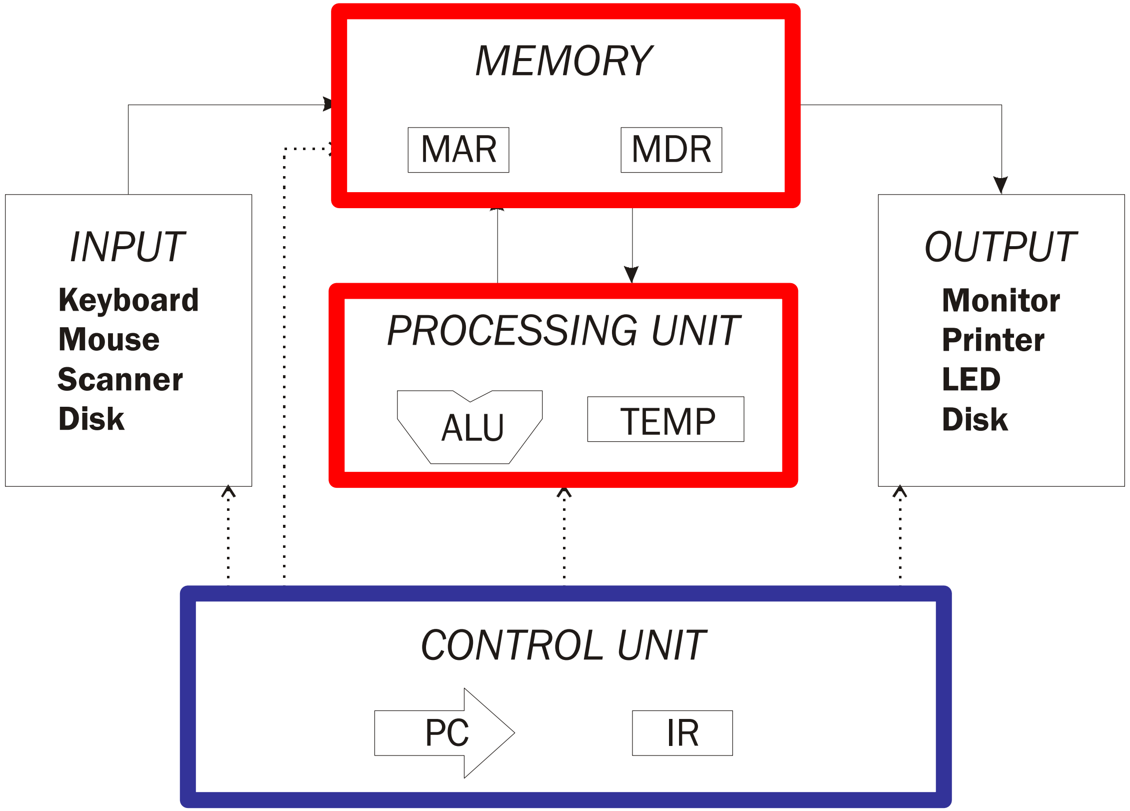
B+off6 : Base + SEXT[offset6]
PC+off9 : PC + SEXT[offset9]
PC+off11 : PC + SEXT[offset11]
ACV=
(Addr<x3000 or Addr>=0xFE00)
& PSR[15]

*OP2 may be SR2 or SEXT[imm5]

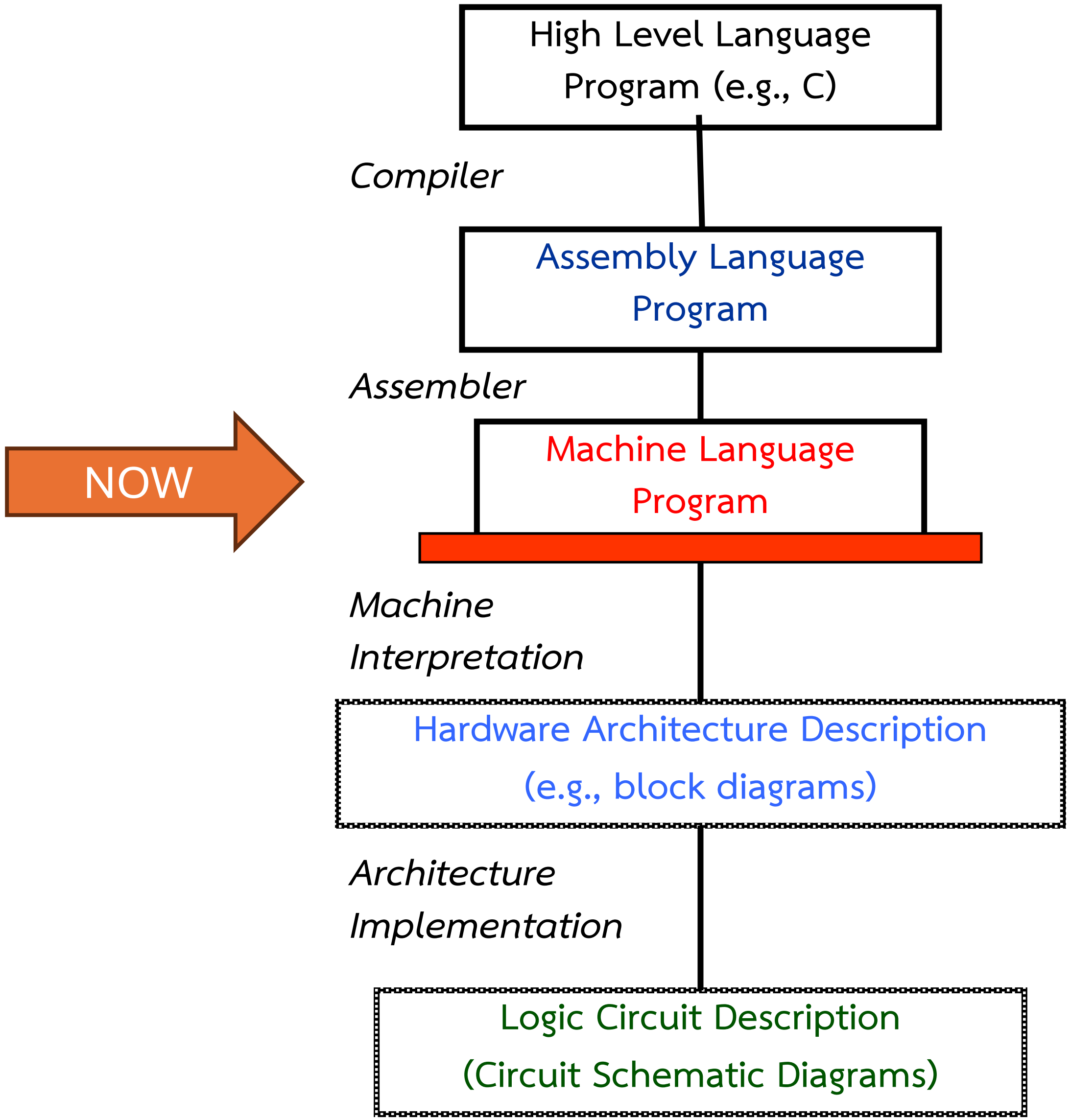
Figure C.2 **A state machine for the LC-3.**



- 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?



High Level Language Program (e.g., C)

Compiler

Assembly Language Program

Assembler

NOW

Machine Language Program

Machine Interpretation

Hardware Architecture Description (e.g., block diagrams)

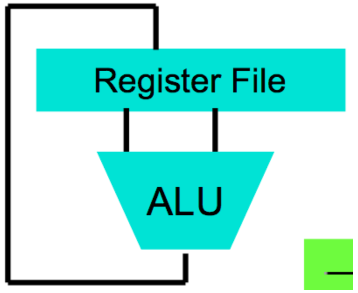
Architecture Implementation

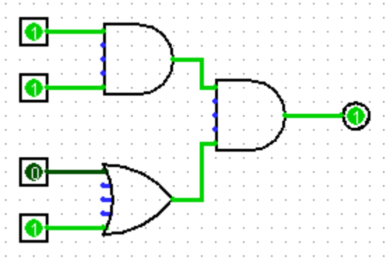
Logic Circuit Description (Circuit Schematic Diagrams)

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

```
lw  $t0, 0(a0)
lw  $t1, 4(a0)
sw  $t1, 0(a0)
sw  $t0, 4(a0)
```

```
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
```





LC-3 ISA Overview

Instruction Set Architecture

- ISA = ***Programmer-visible*** components & operations
  - 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
- All information needed to write/gen machine language program

LC-3 Overview: Memory and Registers

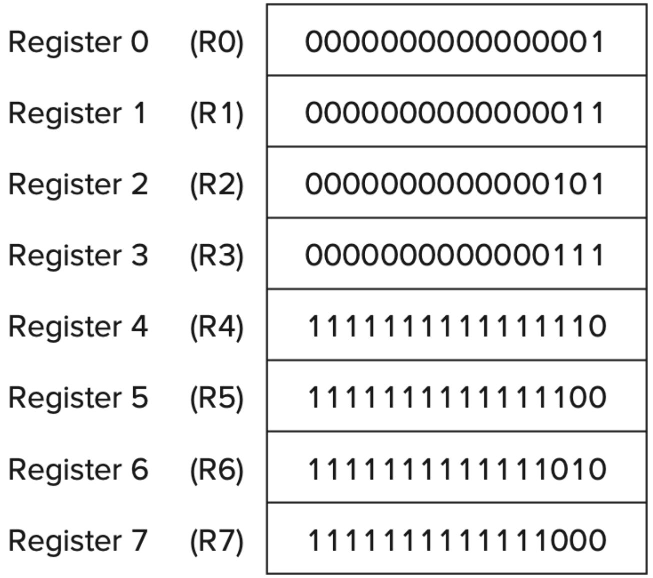
- Memory
  - Address space: 2^{16} locations (16-bit addresses)
  - Addressability: 16 bits
- Registers
  - Temporary storage, accessed in a single machine cycle, Memory access generally takes longer
  - Eight general-purpose **Registers: R0 - R7**

    - Each 16 bits wide

    - How many bits to uniquely identify a register? → 000 - 111
  - Other registers

    - Not directly addressable, but used by (and affected by) instructions

    - **PC (program counter), condition codes, MAR, MDR**, etc.



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

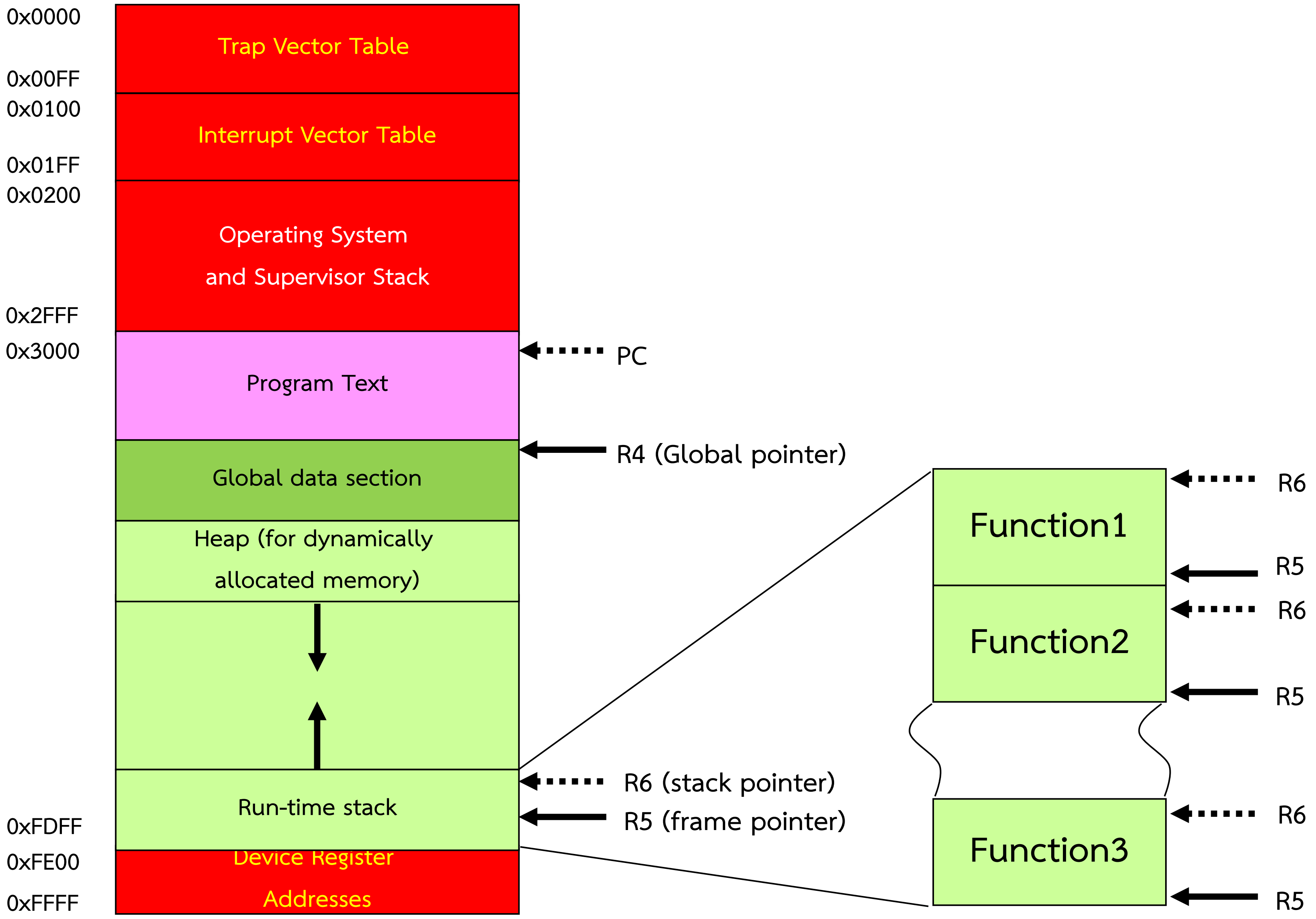
A snapshot of the LC-3's register file.

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

Figure 5.2 The register file of Figure 5.1 after the ADD instruction.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	1
ADD				R2			R0						R1		

LC-3 Overview: Memory Map



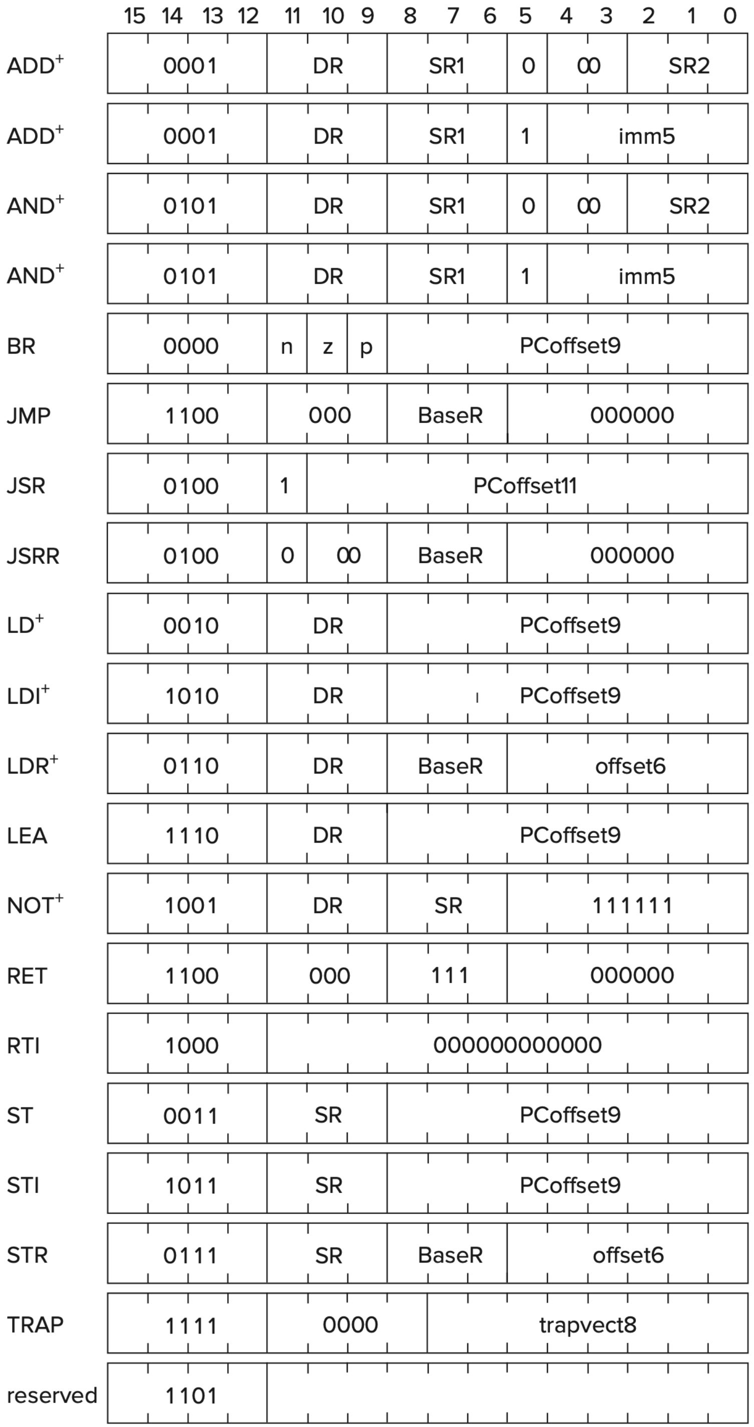
LC-3 Overview: Instruction Set

- Opcodes
  - 16 opcodes
  - *Operate* instructions: ADD, AND, NOT, (MUL)
  - *Data movement* instructions: LD, LDI, LDR, LEA, ST, STR, STI
  - *Control* instructions: BR, JSR, JSRR, RET, RTI, TRAP
  - Some opcodes set/clear *condition codes (CC)*, based on result

    N = negative (<0), Z = zero (=0), P = positive (> 0)

- Data Types
  - 16-bit 2's complement integer
- Addressing Modes
  - How is the location of an operand specified?
  - Non-memory addresses: *register*, *immediate (literal)*
  - Memory addresses: *base+offset*, *PC-relative*, *indirect*

LC-3 Instruction Summary



	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
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								
LDR$^+$	0110				DR			BaseR			offset6					
LEA	1110				DR			PCoffset9								
NOT$^+$	1001				DR			SR			111111					
RET	1100				000			111			000000					
RTI	1000				000000000000											
ST	0011				SR			PCoffset9								
STI	1011				SR			PCoffset9								
STR	0111				SR			BaseR			offset6					
TRAP	1111				0000				trapvect8							
reserved	1101															

LC-3 ISA Group

Operate Instructions

	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												

Control Instructions

	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							

Data Movement Instructions

Load

	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								

Store

	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								

LC-3 Operate Instructions and Data Path

Operate Instructions

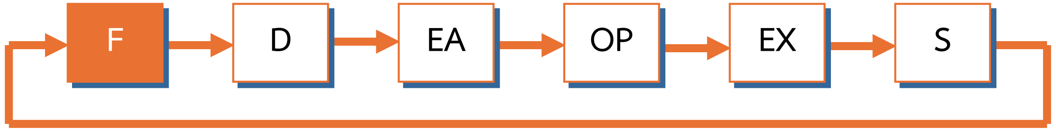
- Only three operations: ADD, AND, NOT
- Source and destination operands are registers
  - *Do not* reference memory
  - ADD and AND can use "immediate" mode,
    (*i.e.,* one operand is hard-wired into instruction)
- Will show abstracted datapath with each instruction
  - illustrate *when* and *where* data moves to accomplish desired operation.

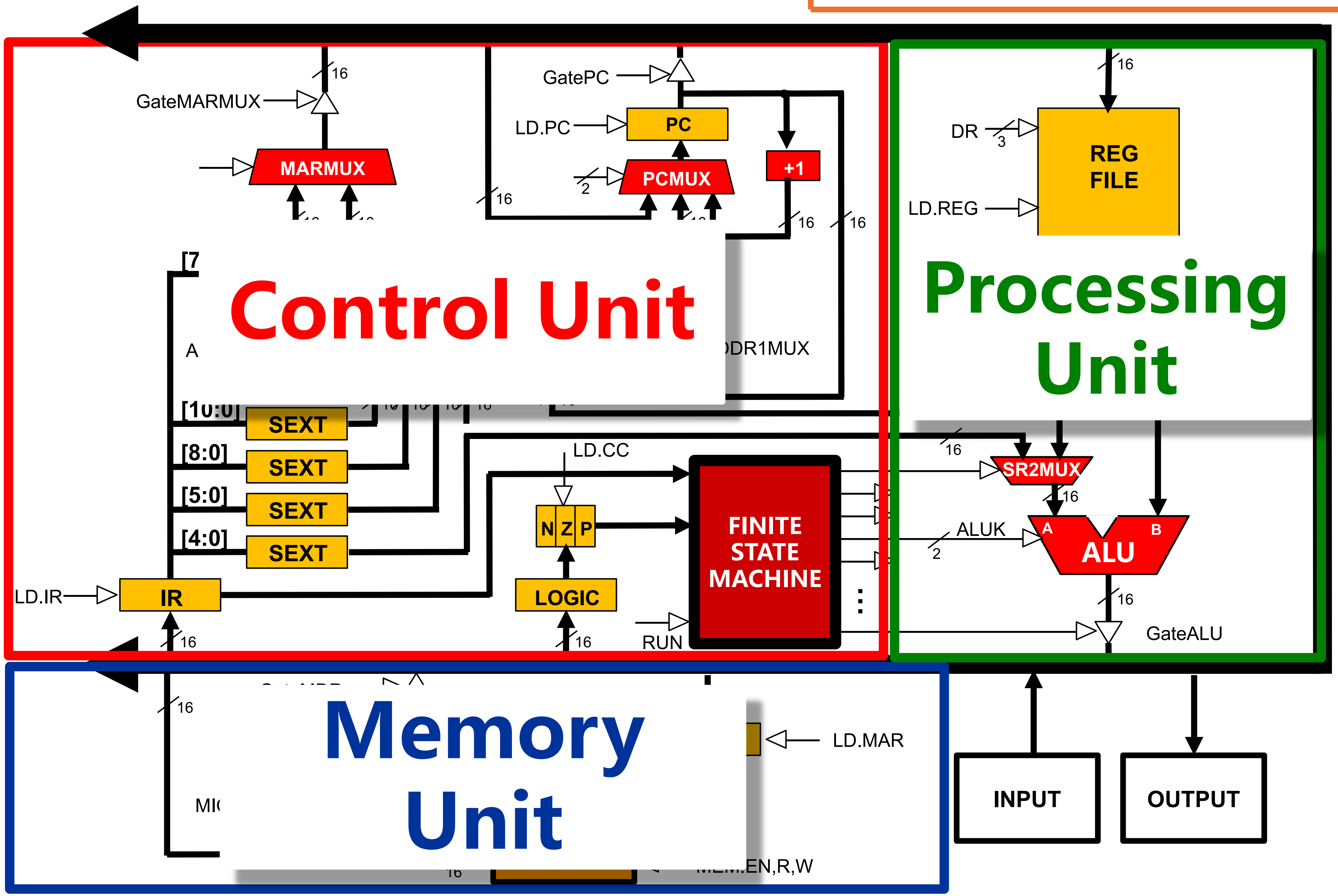
LC-3 ISA Operate Instructions

Operate Instructions

	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												

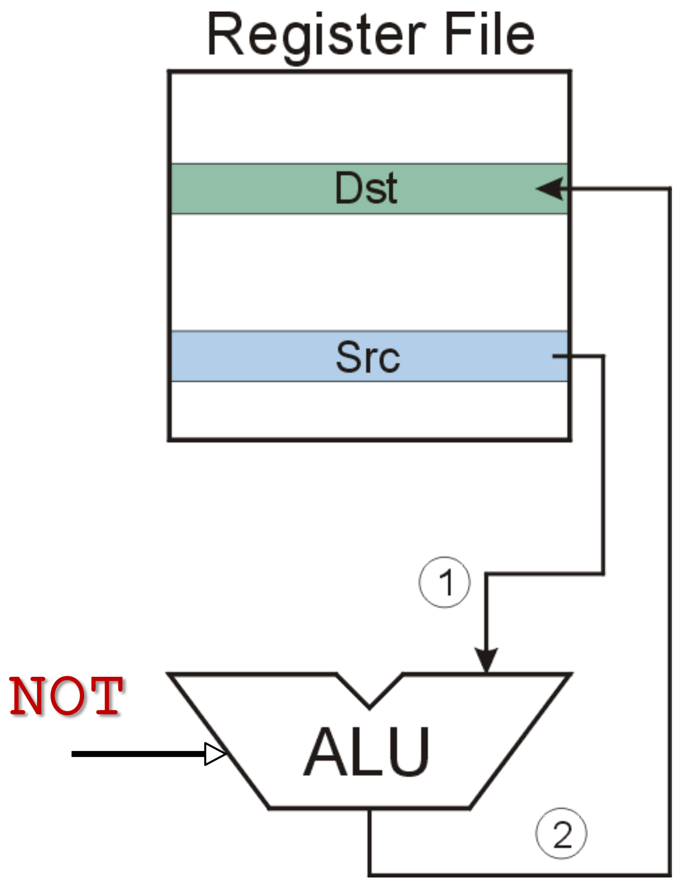
Recall LC-3 Data Path





NOT (Register)

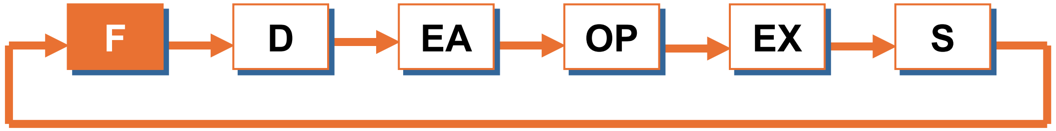
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NOT	1	0	0	1	Dst			Src			1	1	1	1	1	1

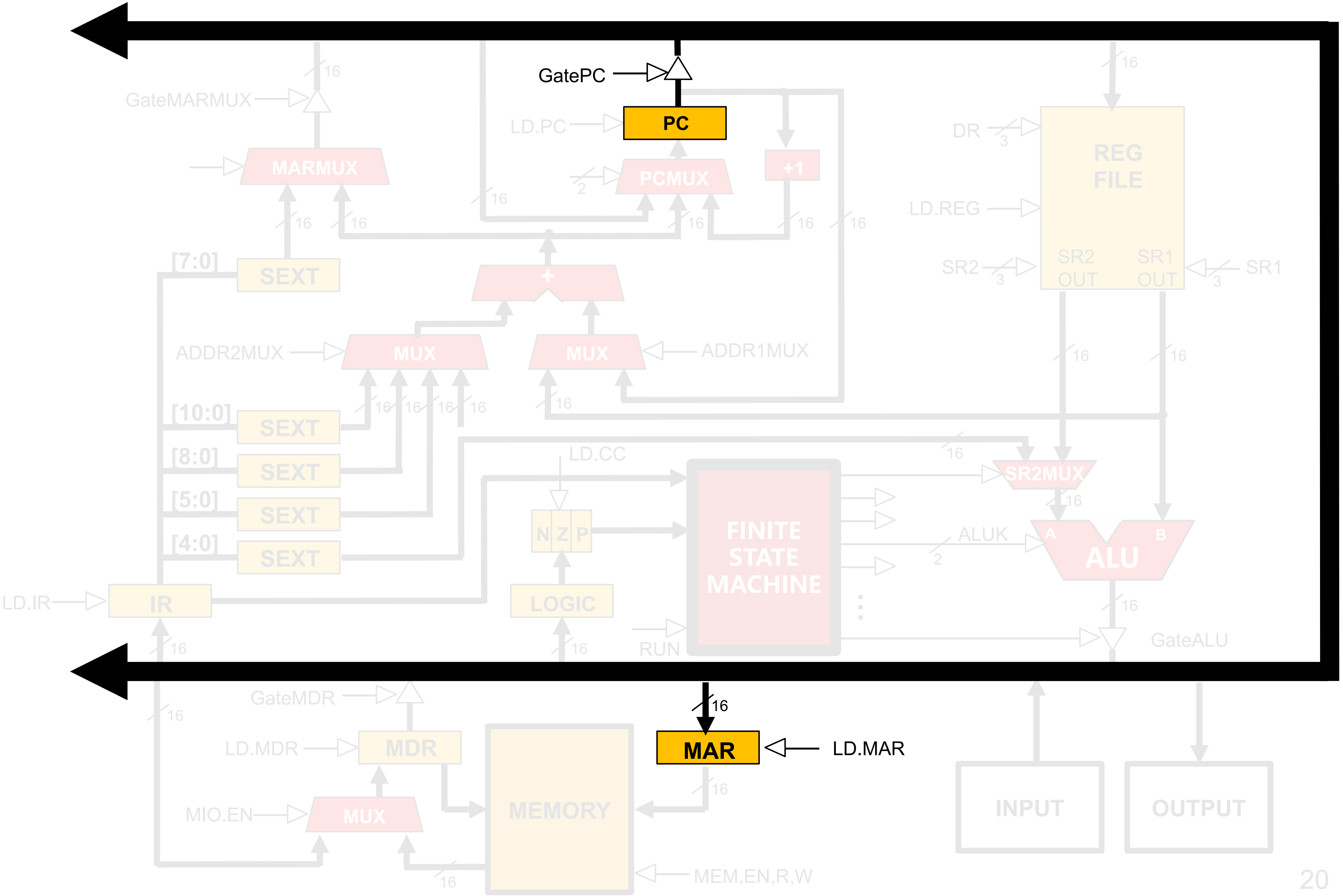


Note: Src and Dst

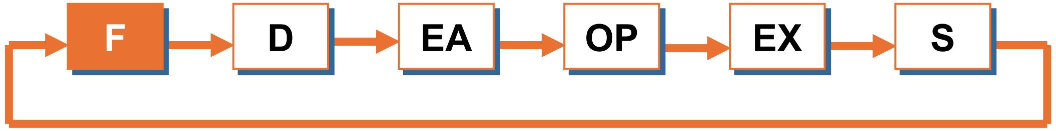
could be the same register

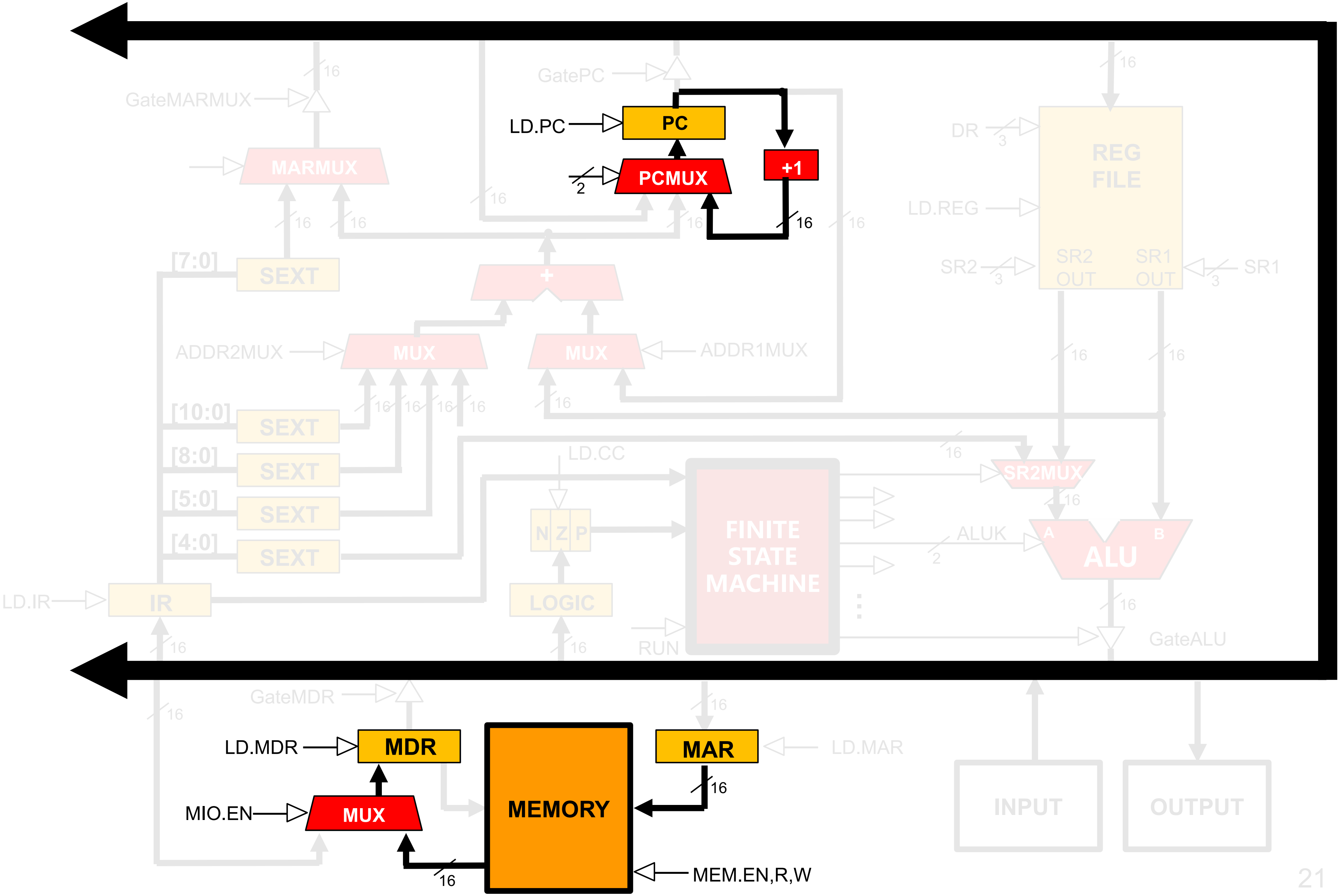
NOT (Register)



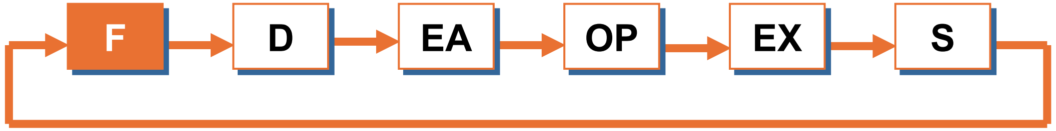


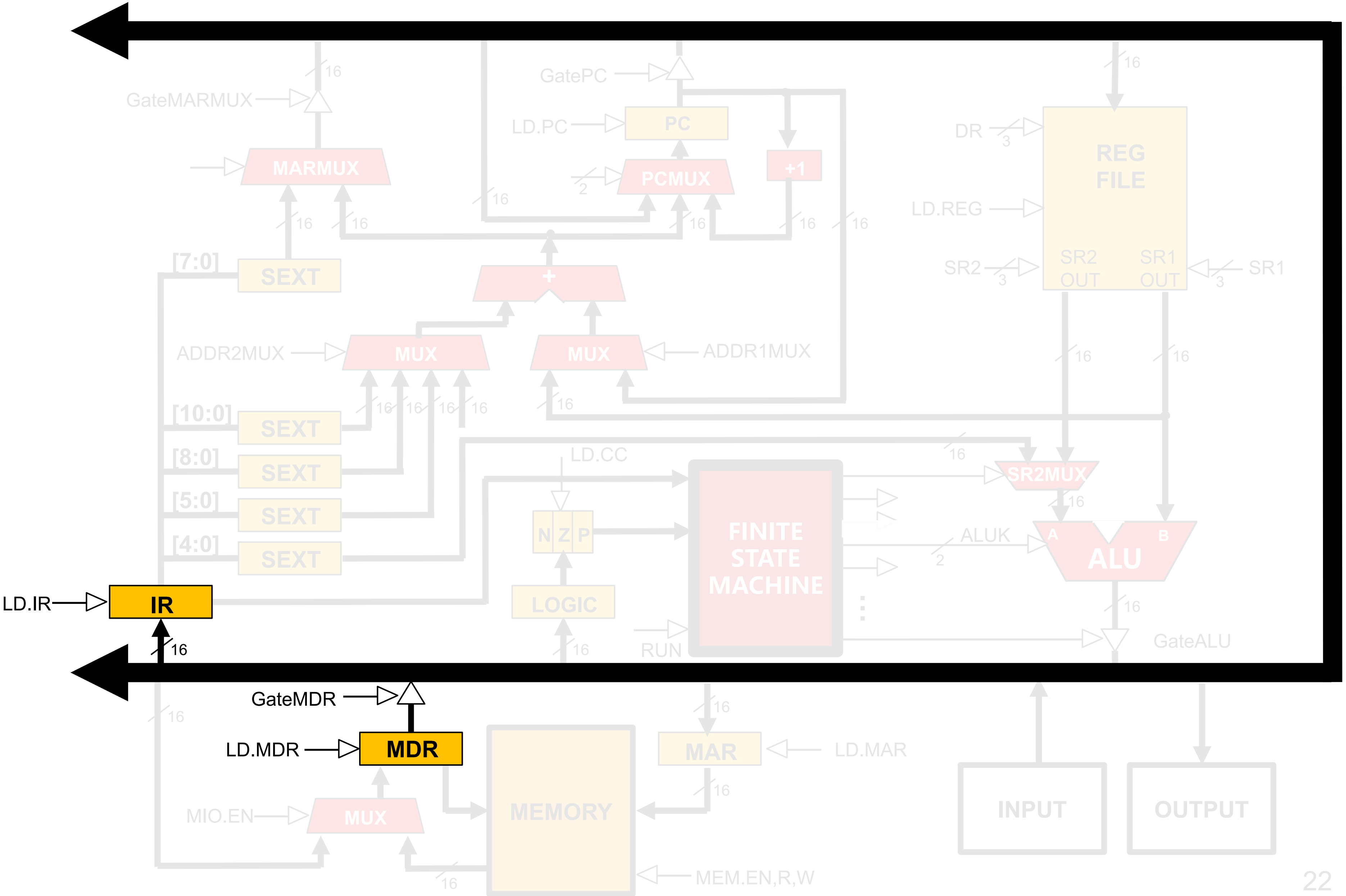
NOT (Register)



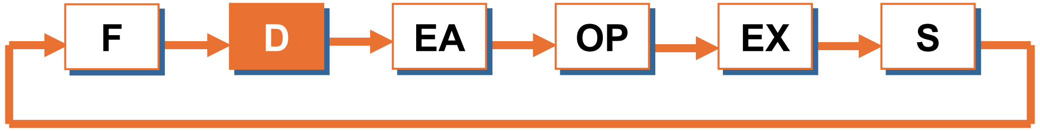


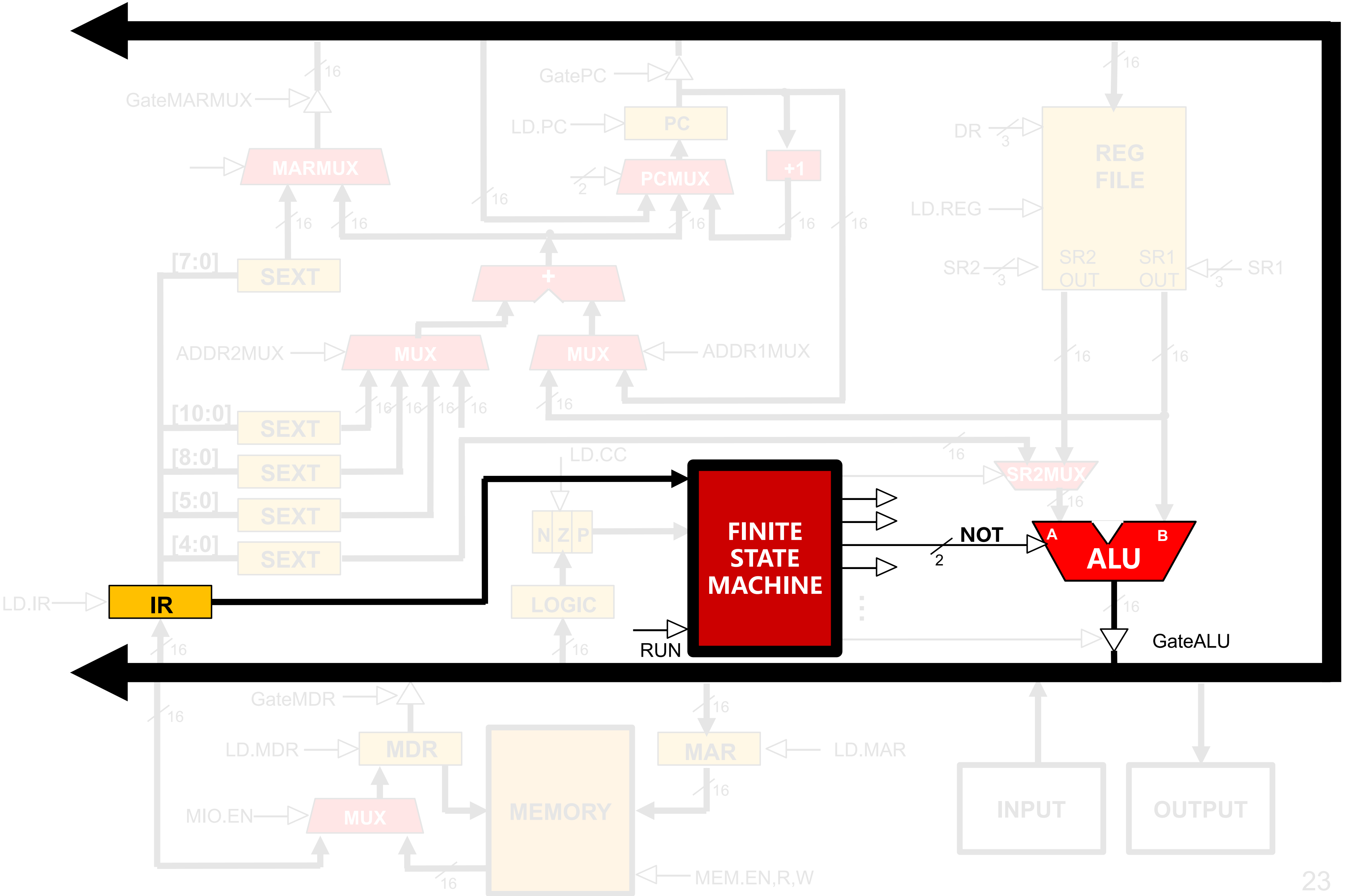
NOT (Register)



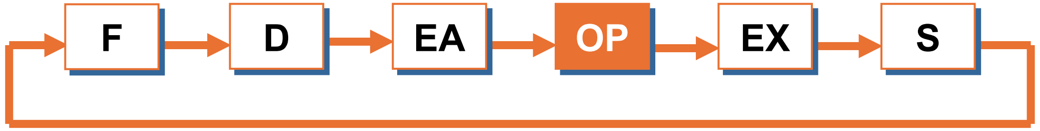


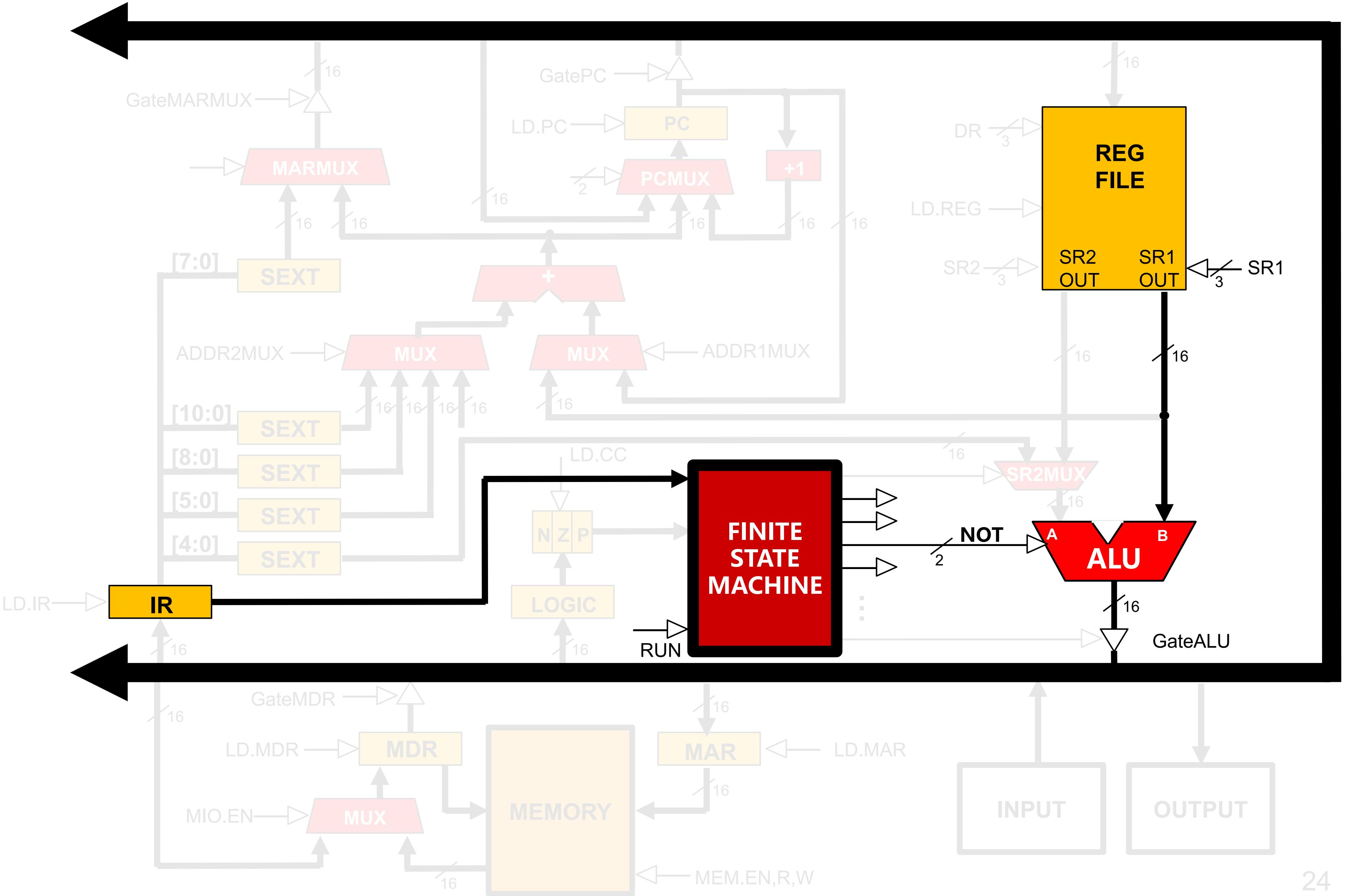
NOT (Register)



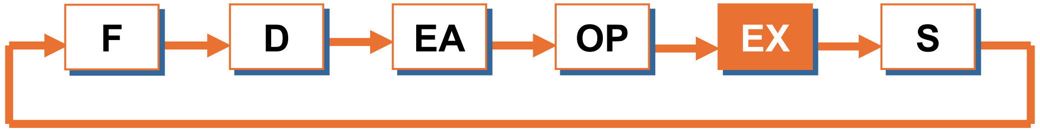


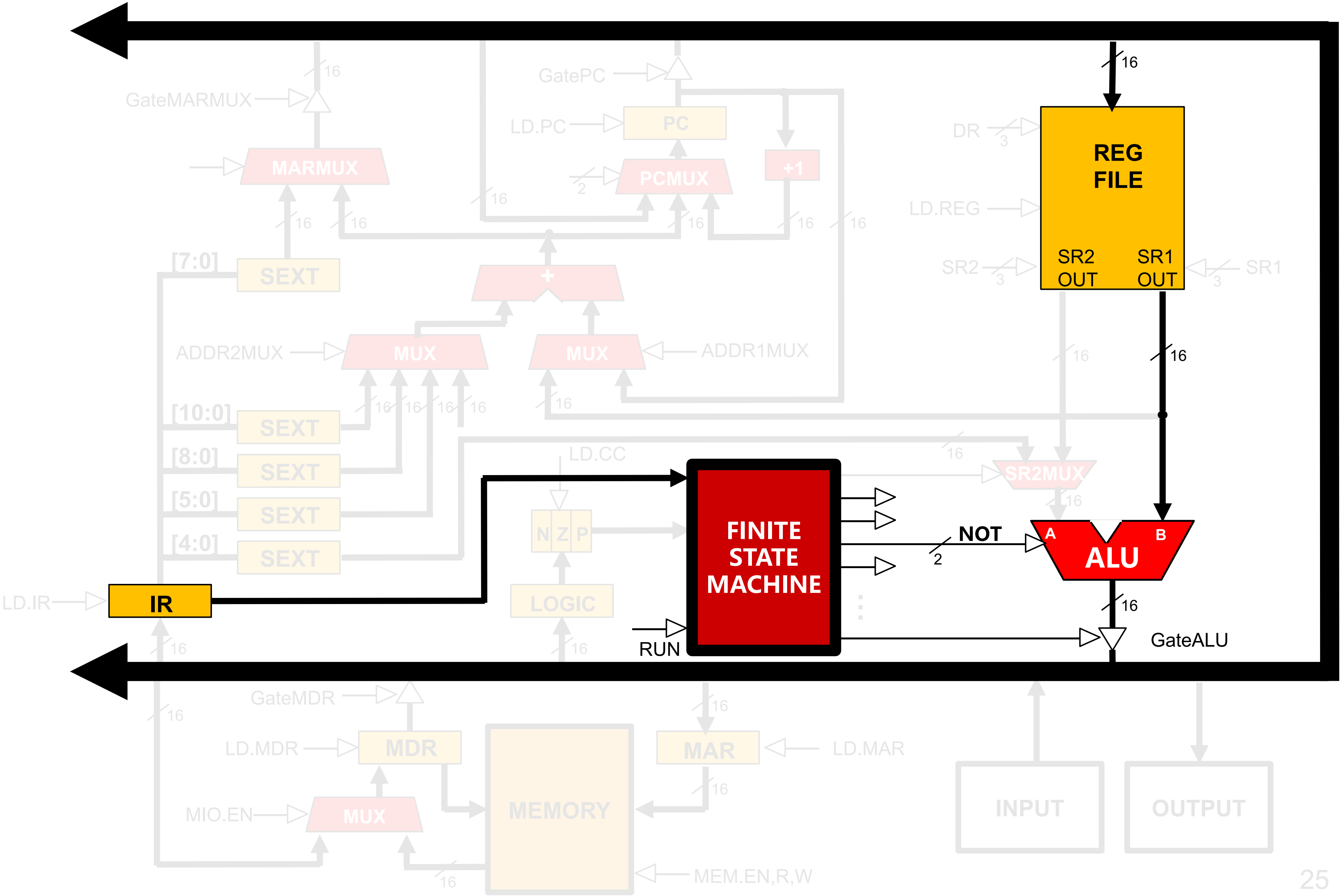
NOT (Register)



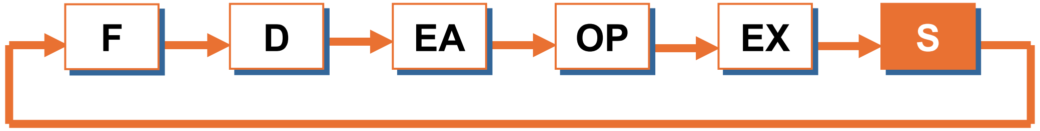


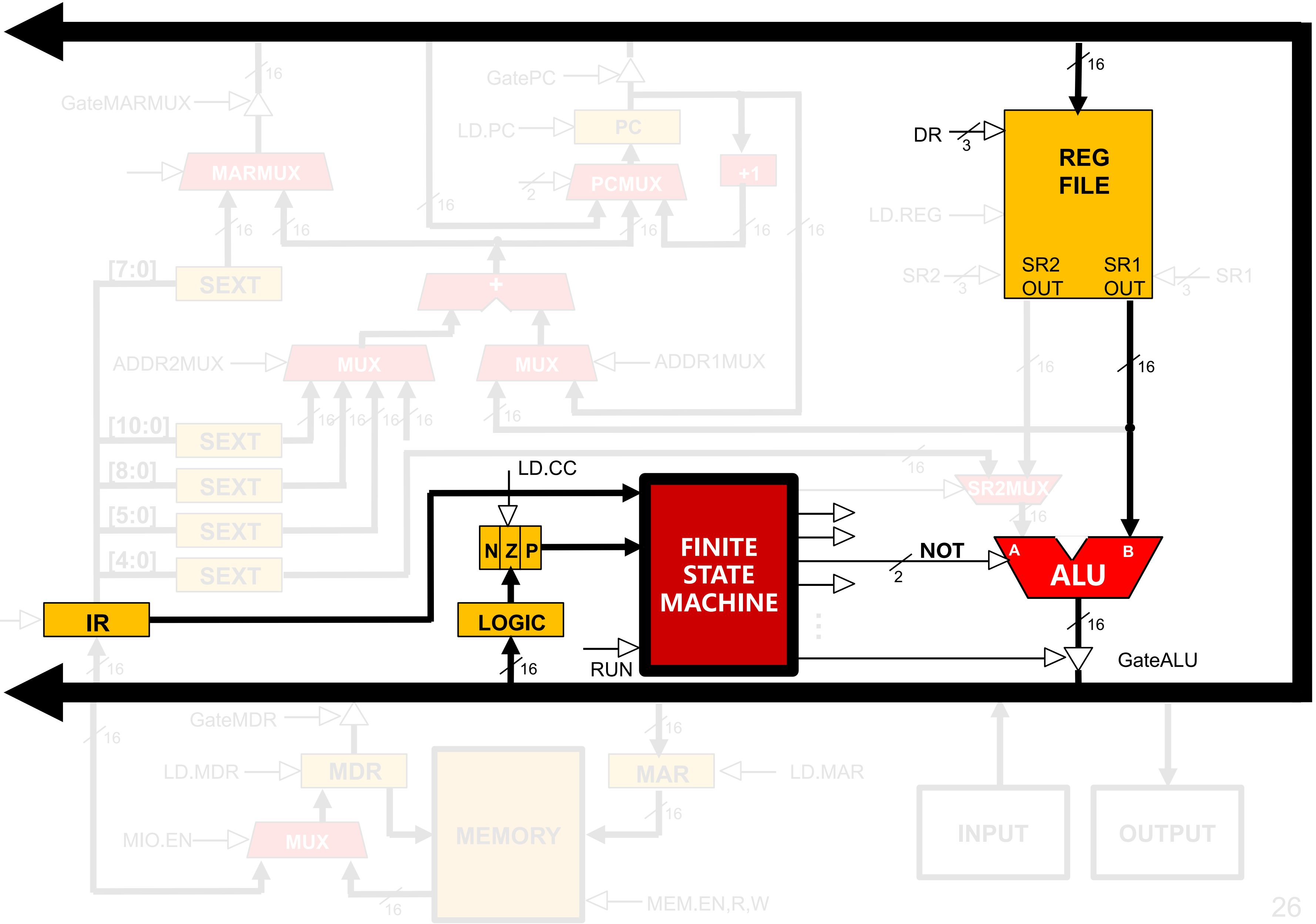
NOT (Register)





NOT (Register)





NOT (Register): NOT R3, R5

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NOT	1	0	0	1	Dst			Src			1	1	1	1	1	1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	1	1	1	0	1	1	1	1	1	1	1
NOT				R3			R5								

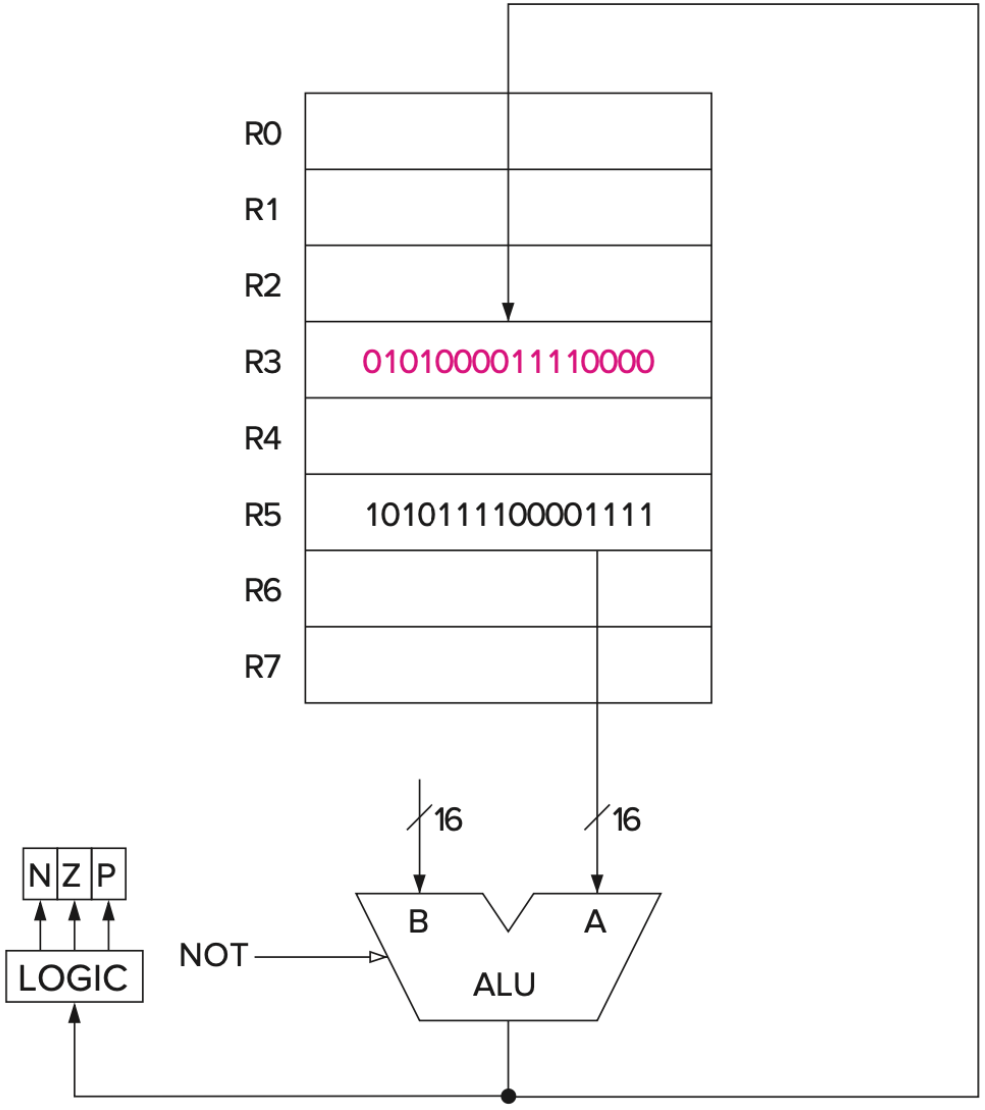
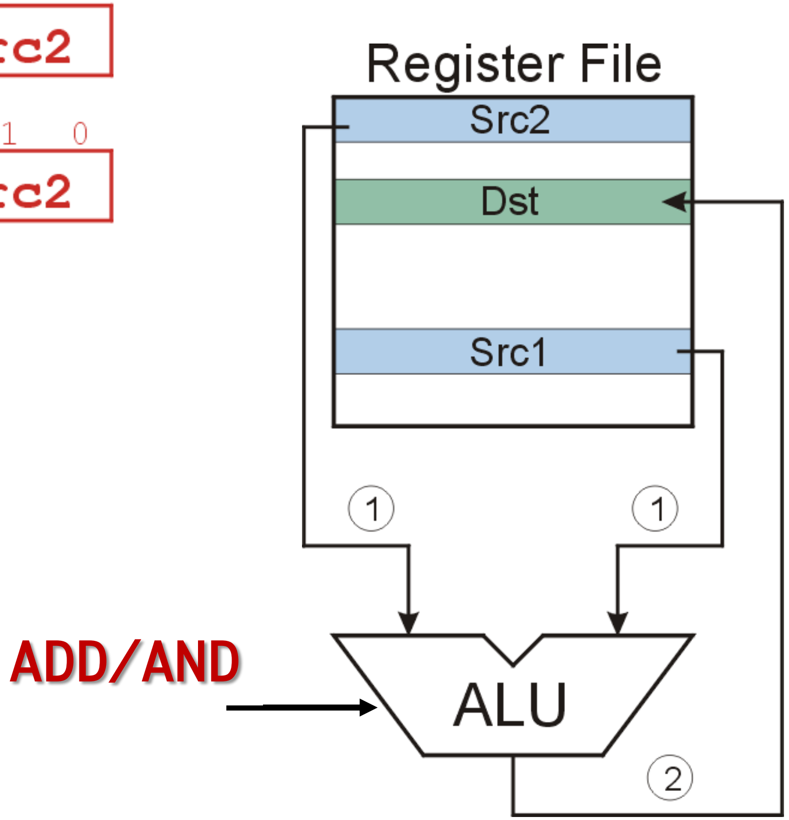


Figure 5.4 **Data path relevant to the execution of NOT R3, R5.**

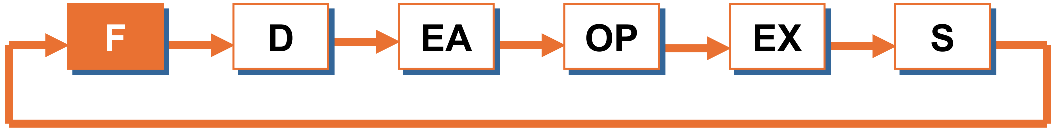
ADD/AND (Register)

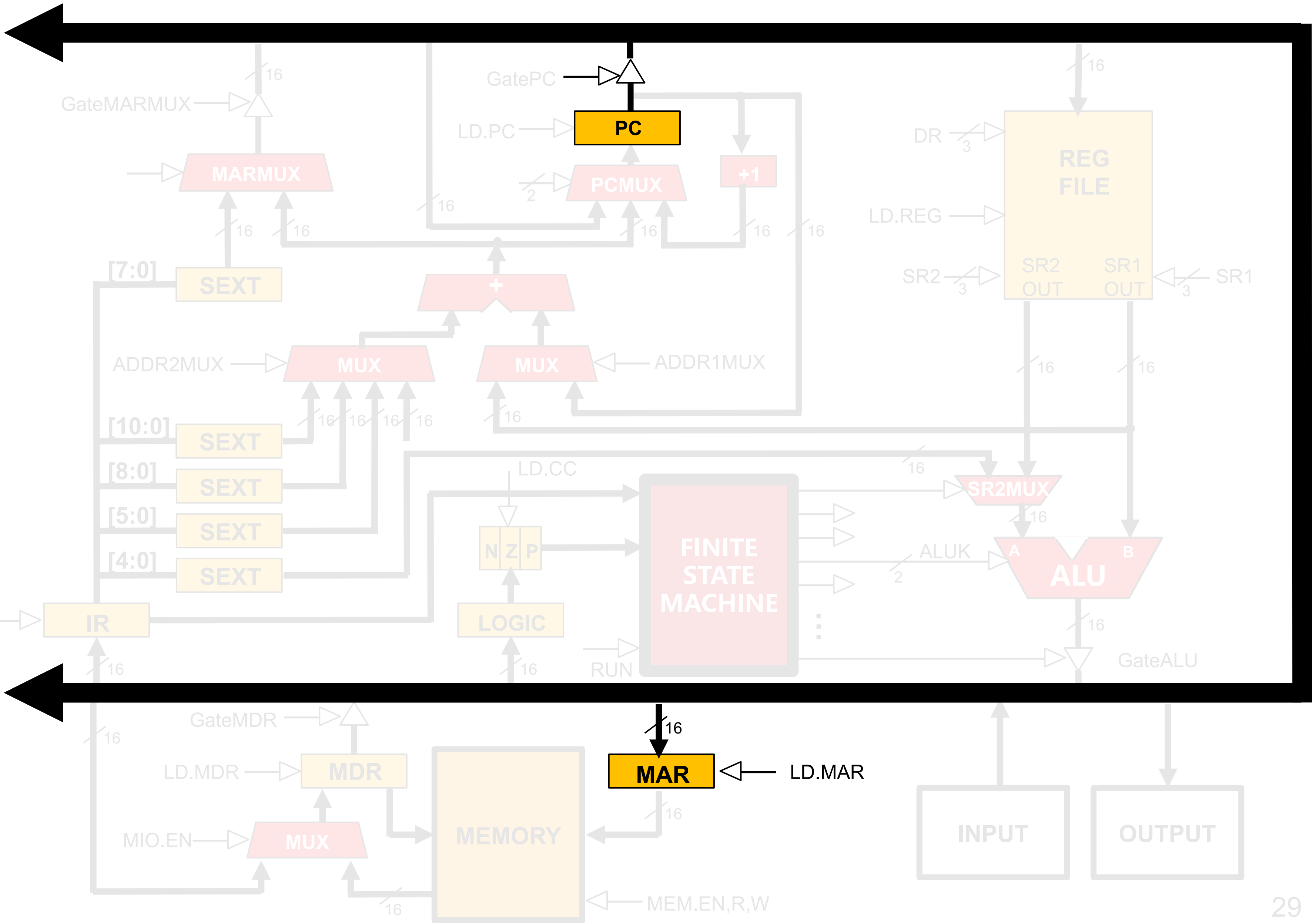
this zero means “register mode” instead of “immediate”

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD	0	0	0	1	Dst			Src1			0	0	0	Src2		
AND	0	1	0	1	Dst			Src1			0	0	0	Src2		

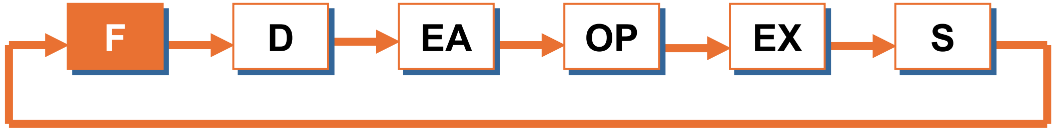


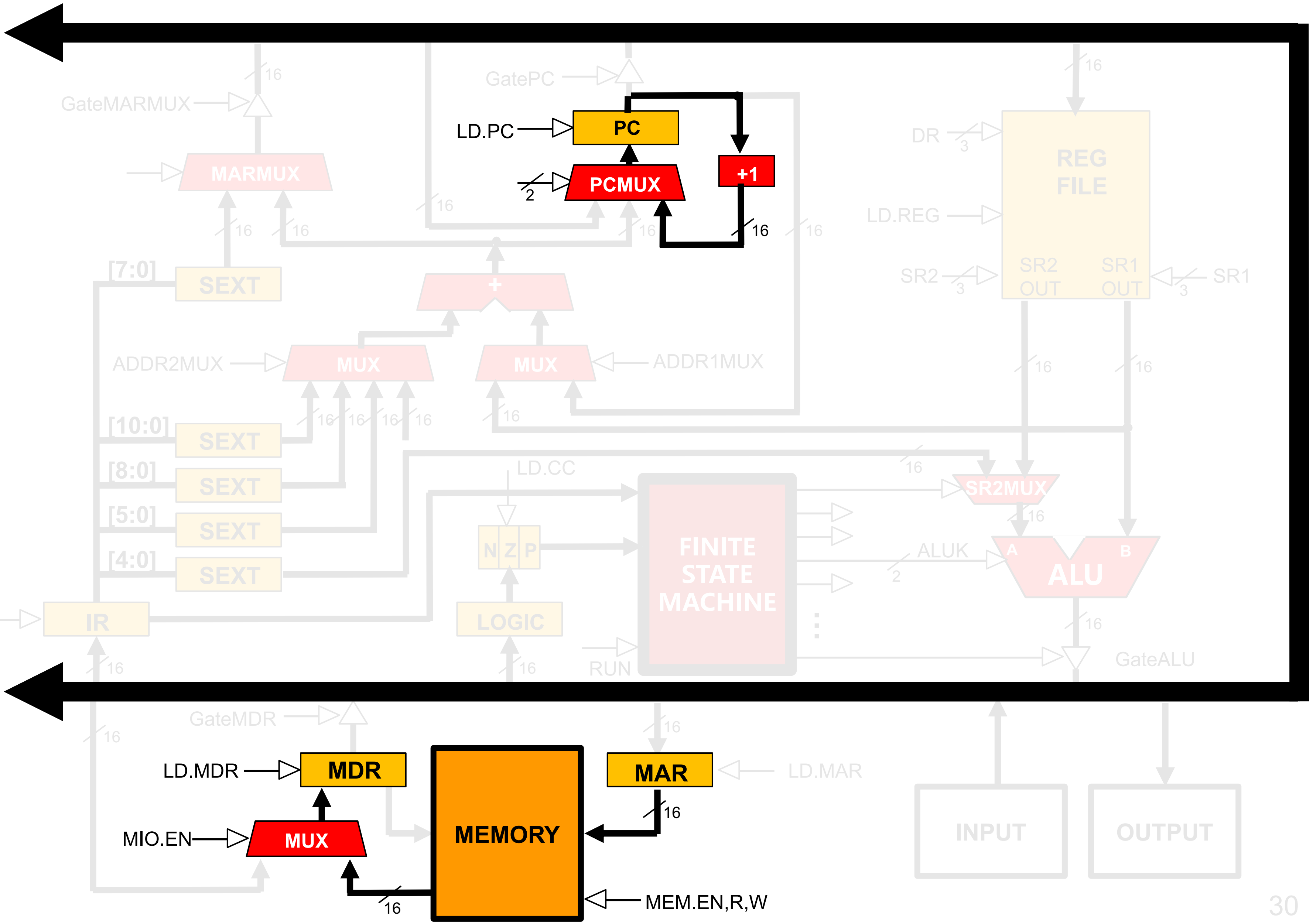
ADD/AND (Register)



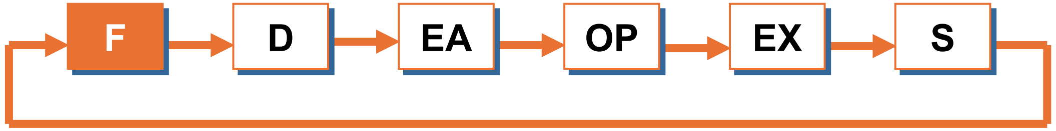


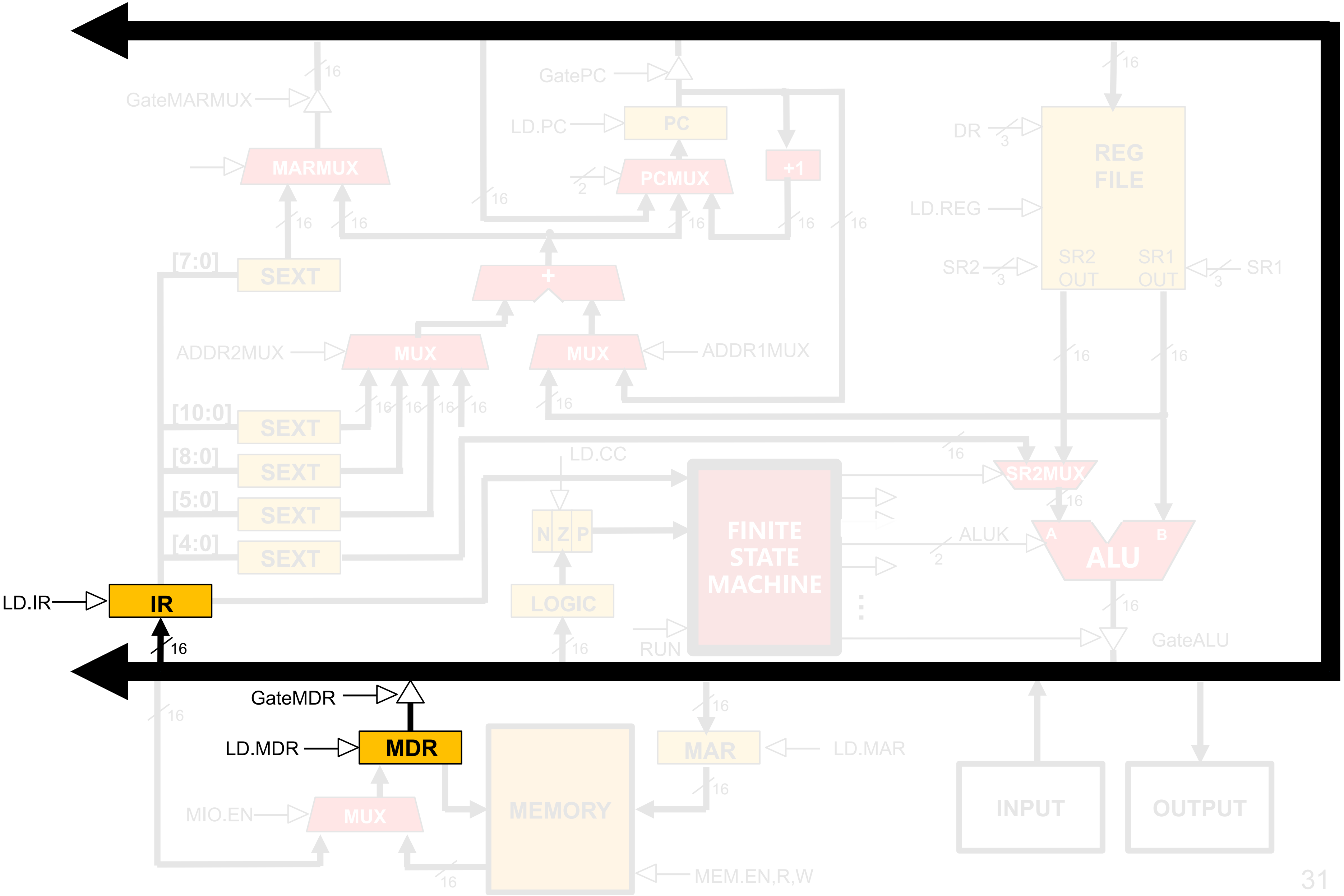
ADD/AND (Register)



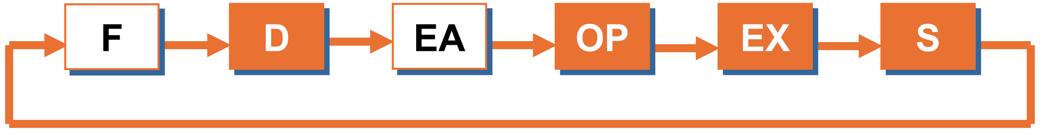


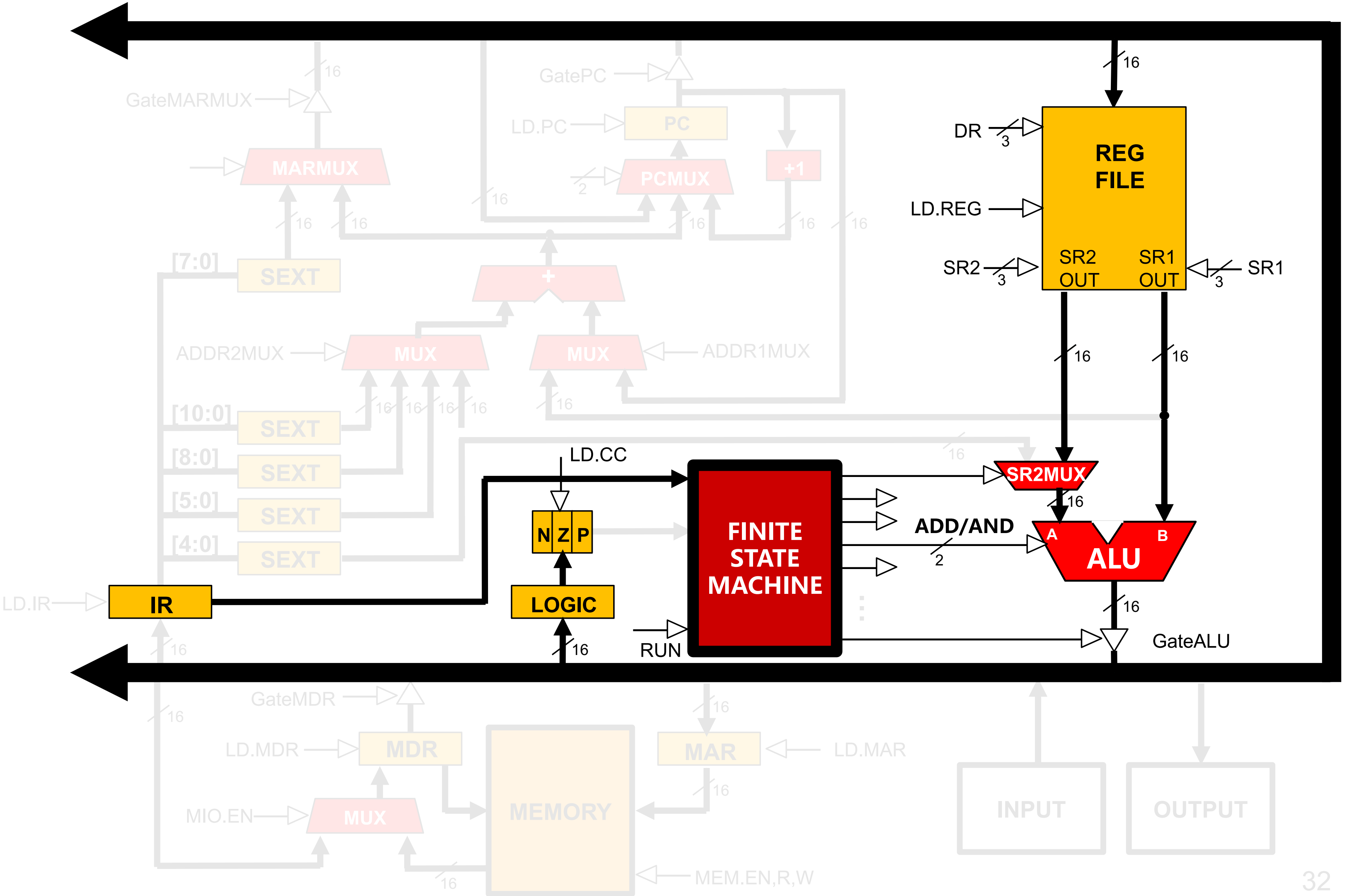
ADD/AND (Register)





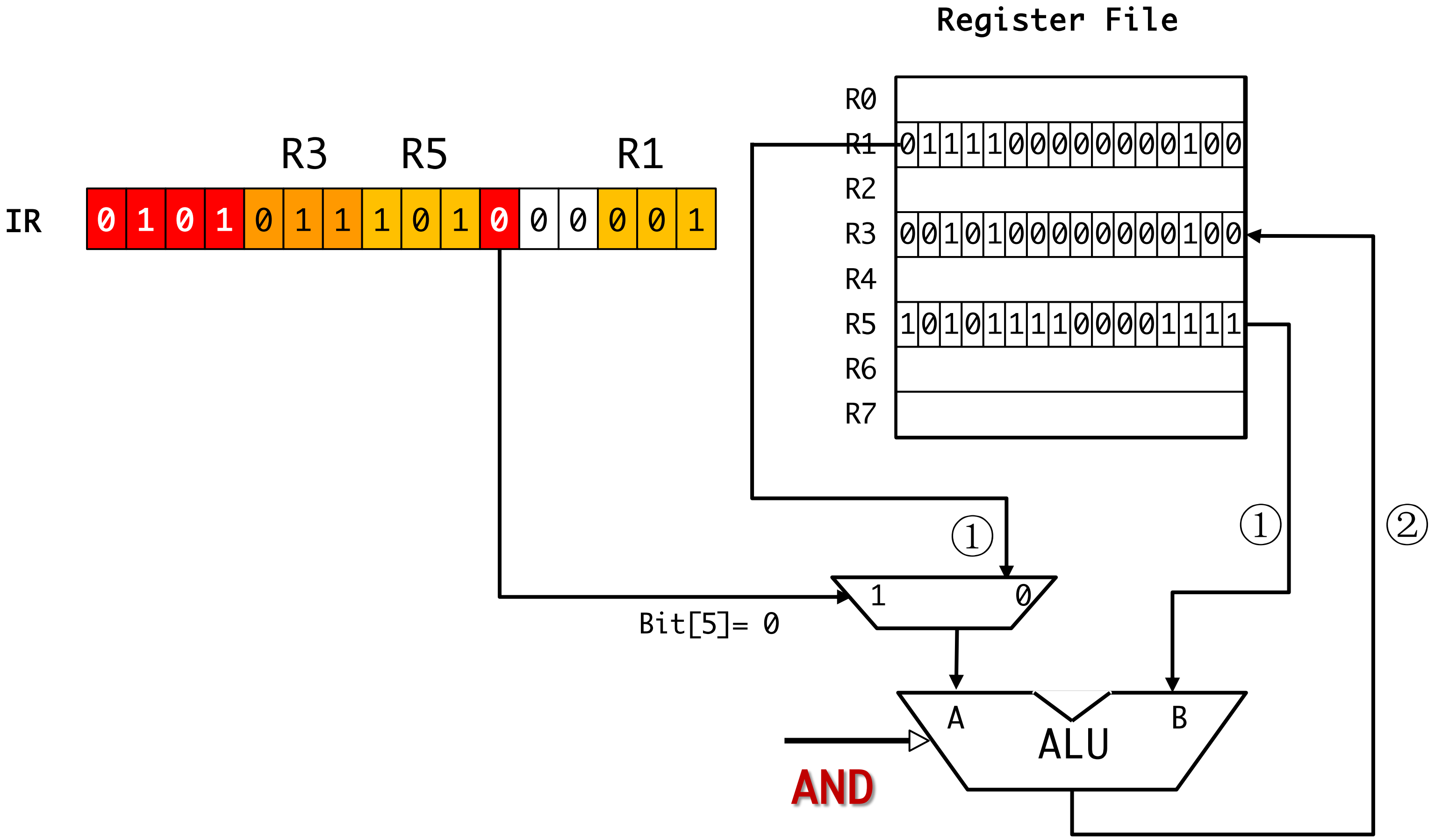
ADD/AND (Register)





AND (Register): AND R3, R5, R1

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AND	0	1	0	1	Dst			Src1			0	0	0	Src2		



					R3			R5							R1	
IR	0	1	0	1	0	1	1	1	0	1	0	0	0	0	0	1

Register File

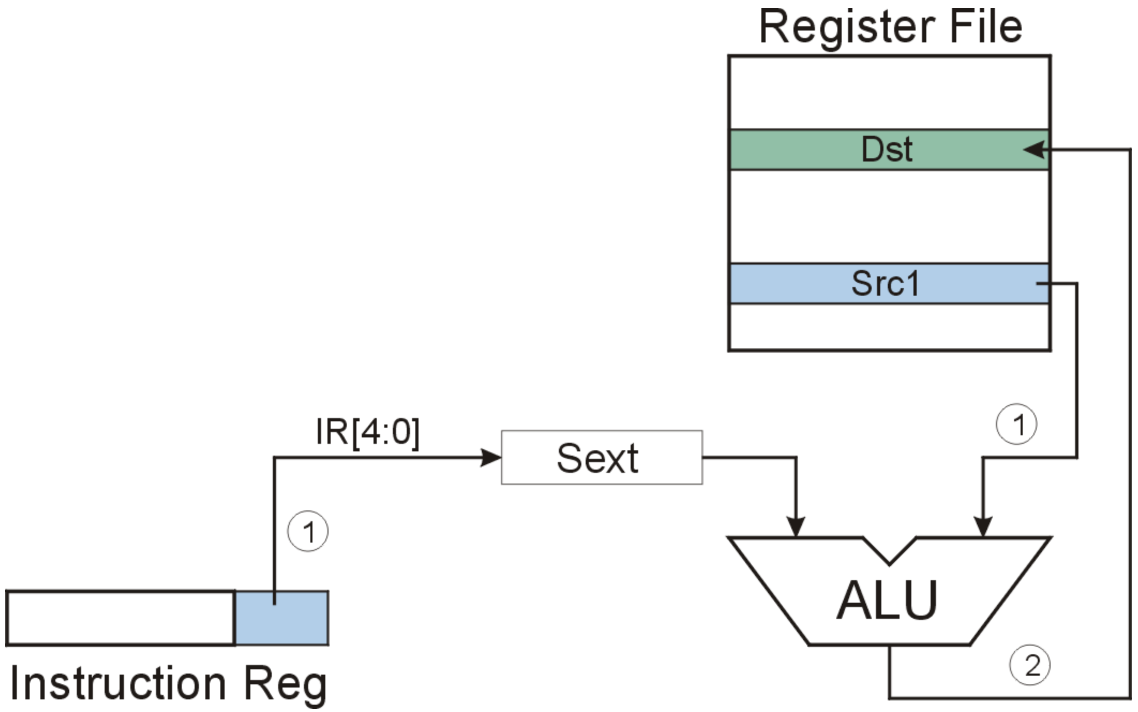
R0																
R1	0	1	1	1	1	0	0	0	0	0	0	0	0	1	0	0
R2																
R3	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0
R4																
R5	1	0	1	0	1	1	1	1	0	0	0	0	1	1	1	1
R6																
R7																

ADD/AND (Immediate)

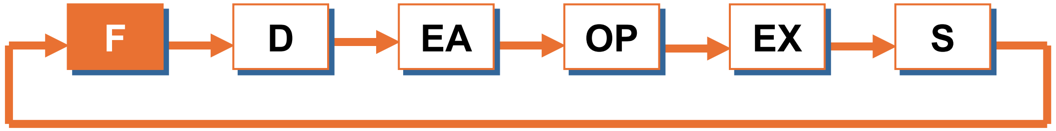
this one means "immediate mode"

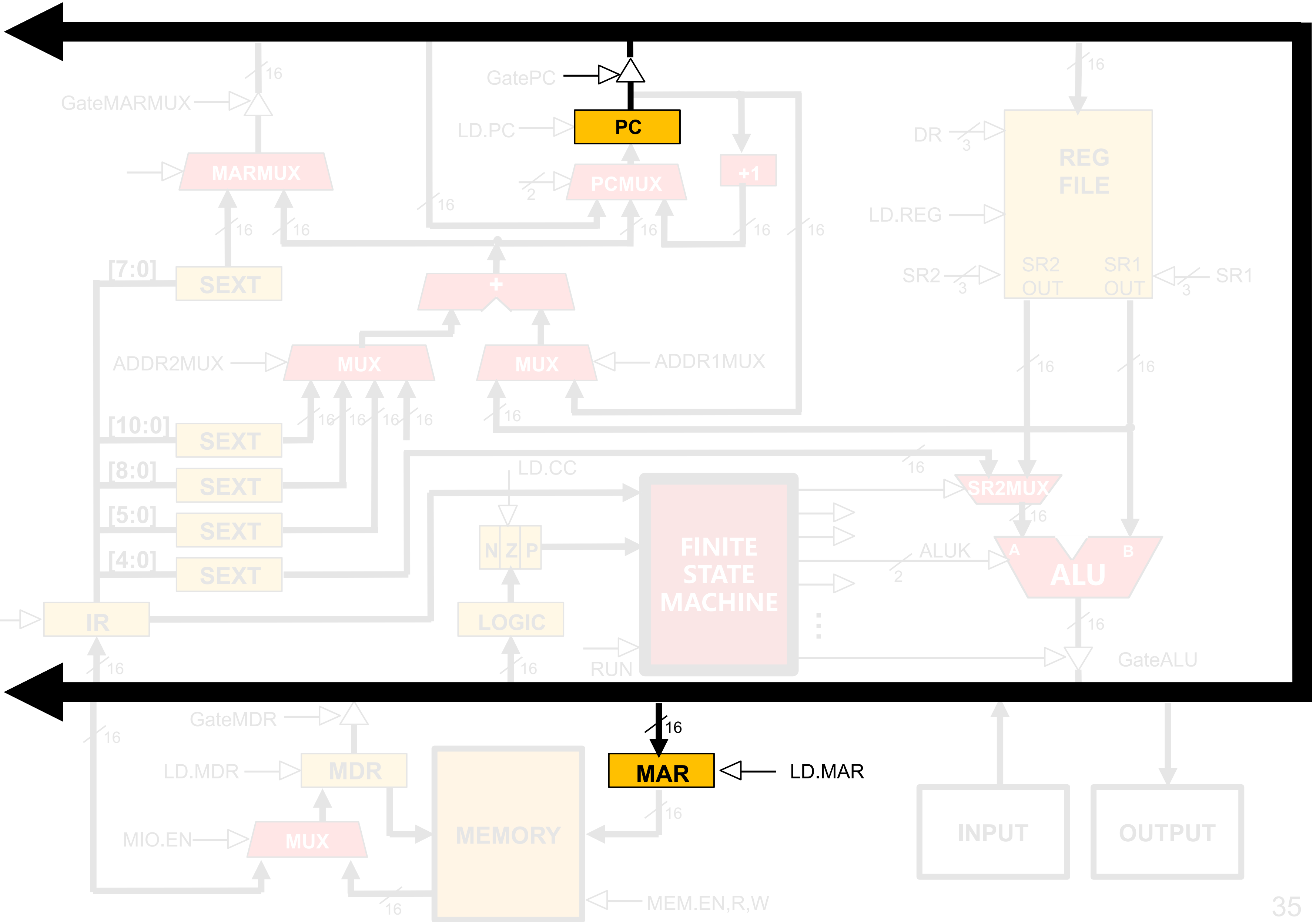
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD	0	0	0	1	Dst			Src1			1	Imm5				
AND	0	1	0	1	Dst			Src1			1	Imm5				

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

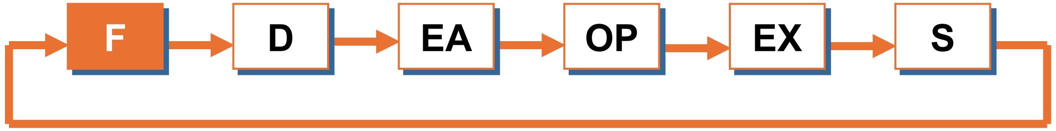


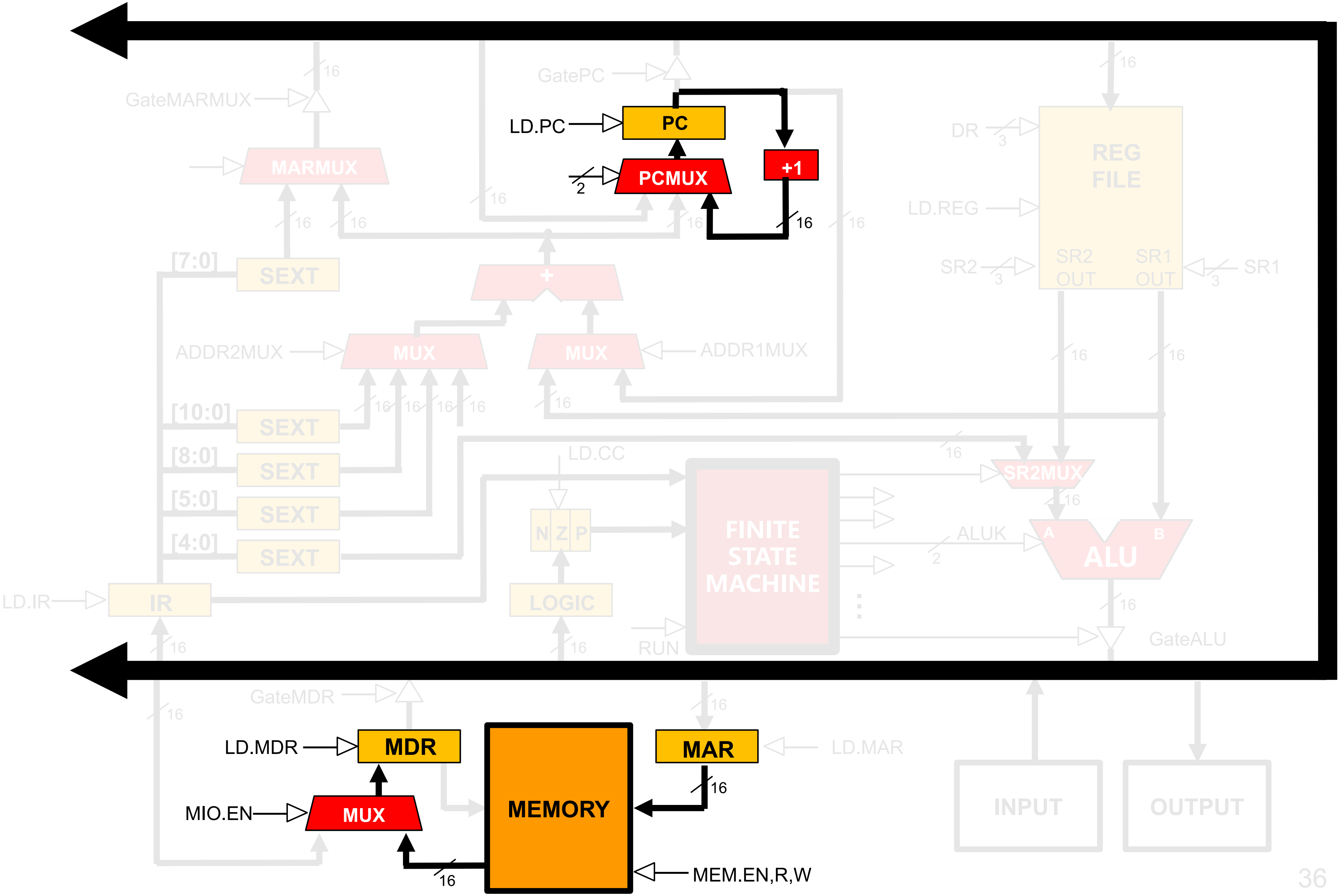
ADD/AND (Immediate)



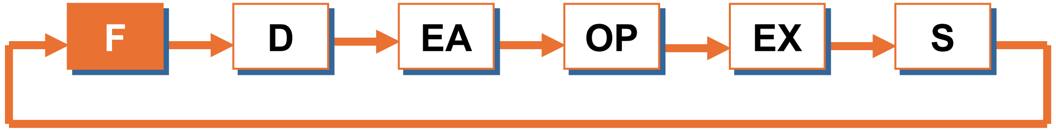


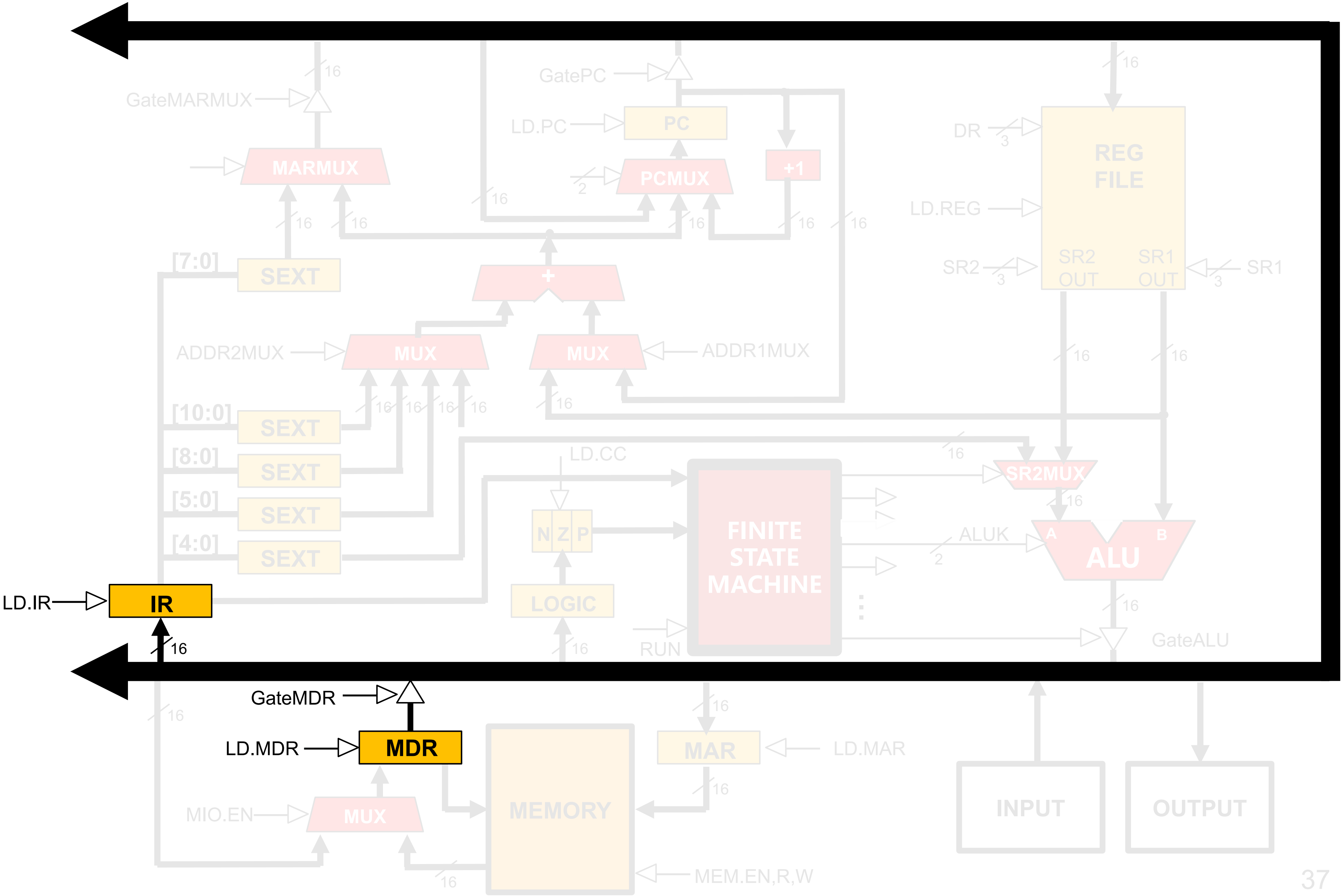
ADD/AND (Immediate)



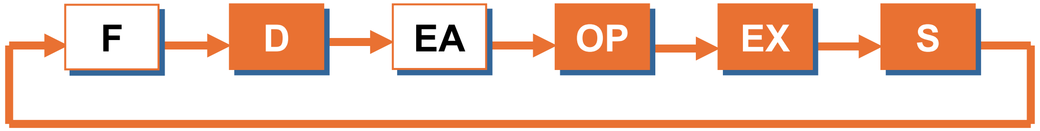


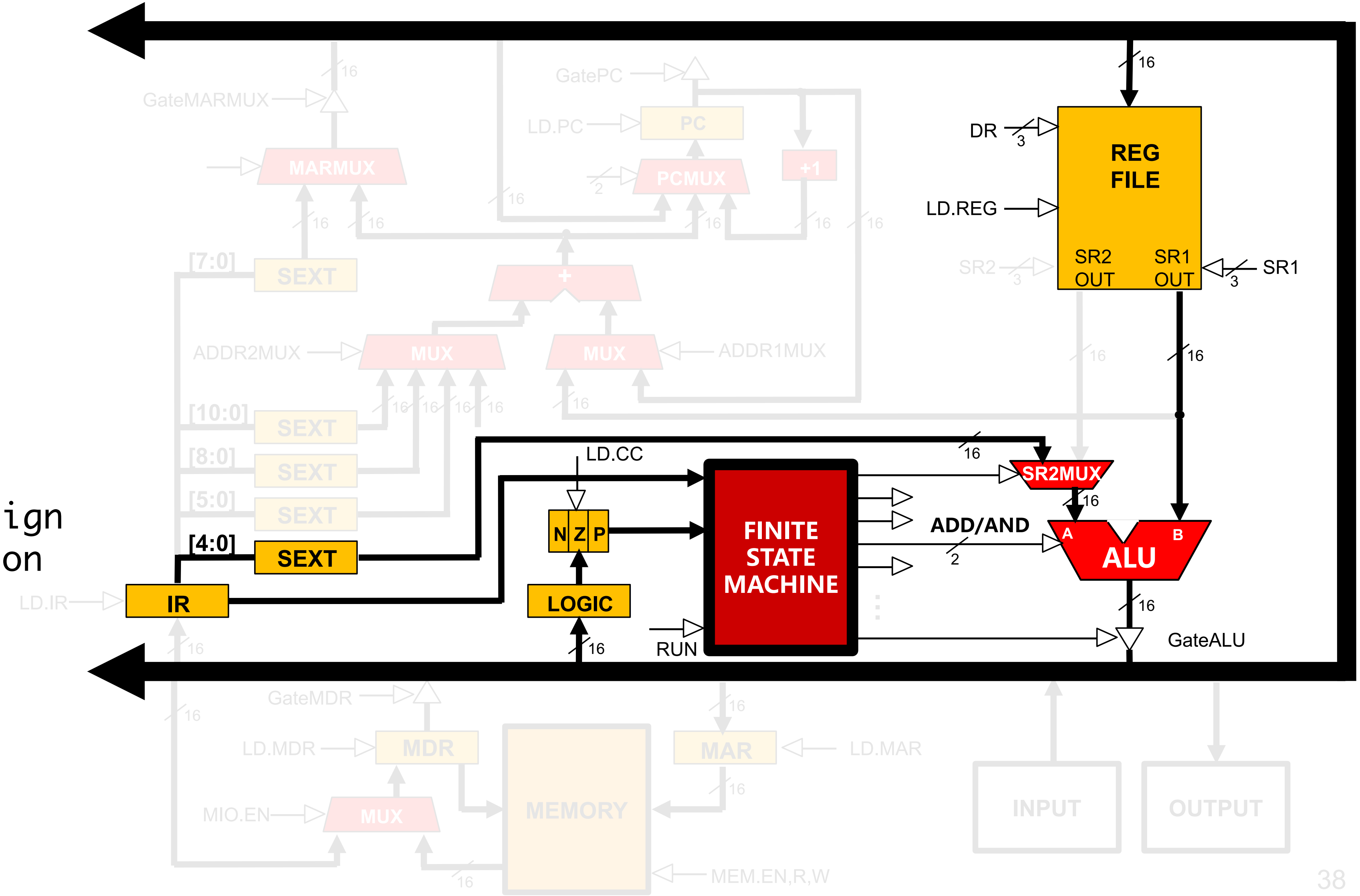
ADD/AND (Immediate)





ADD/AND (Immediate)

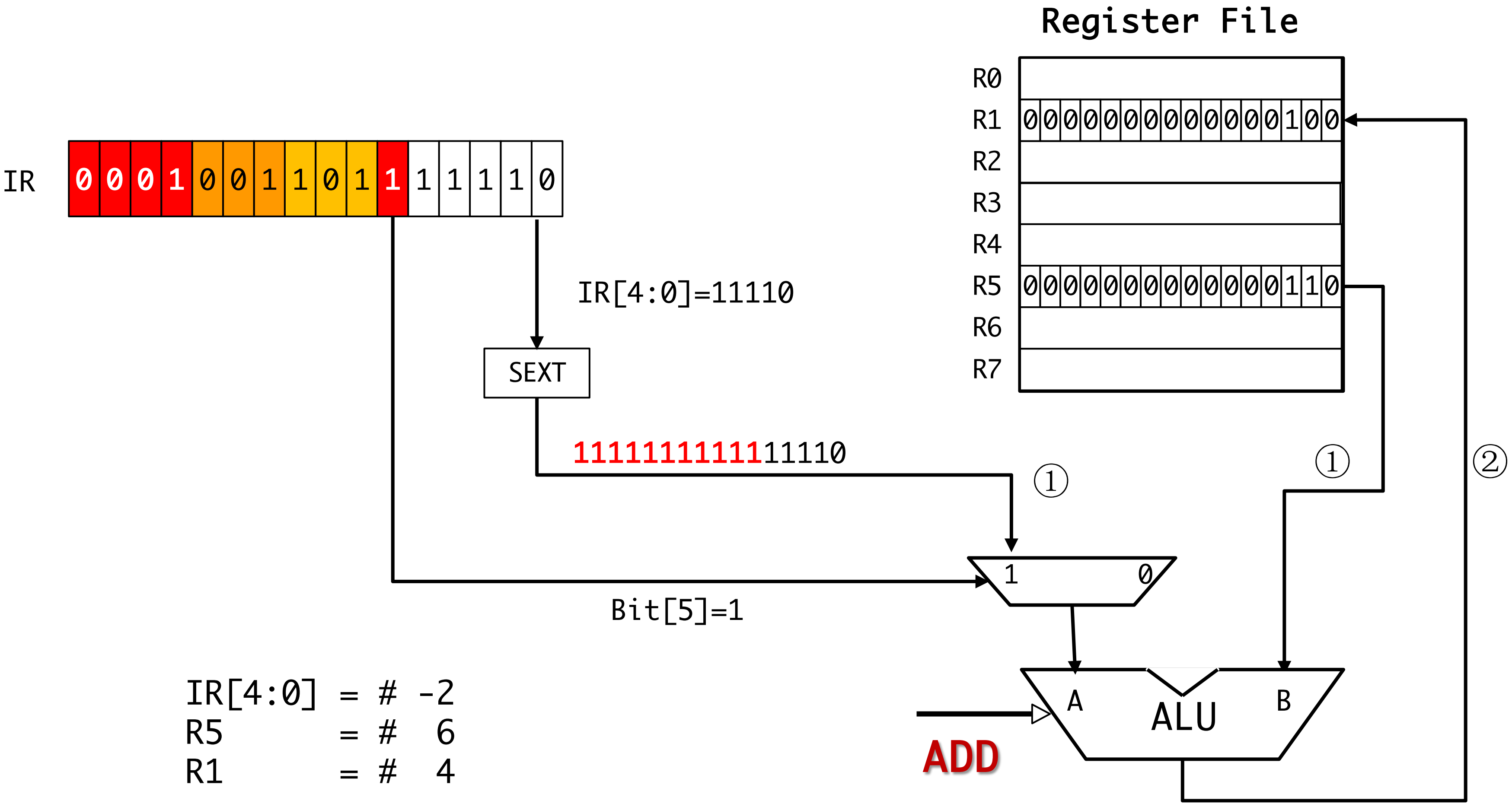




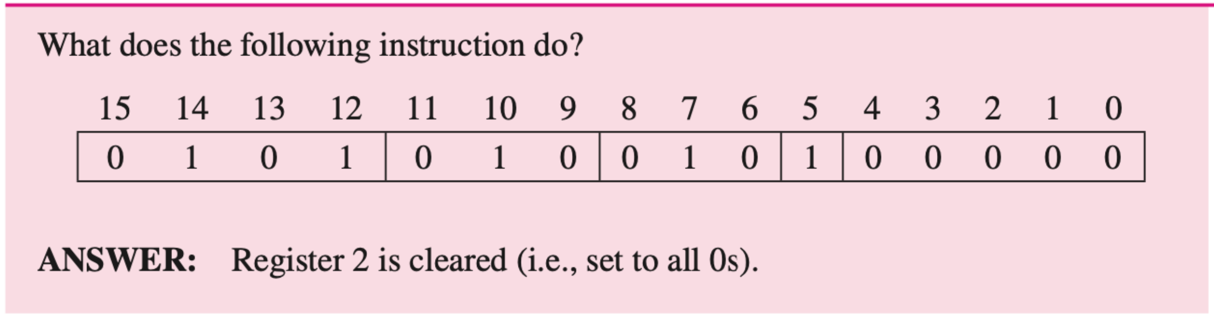
SEXT: Sign
Extension

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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD	0	0	0	1	Dst			Src1			1	Imm5				



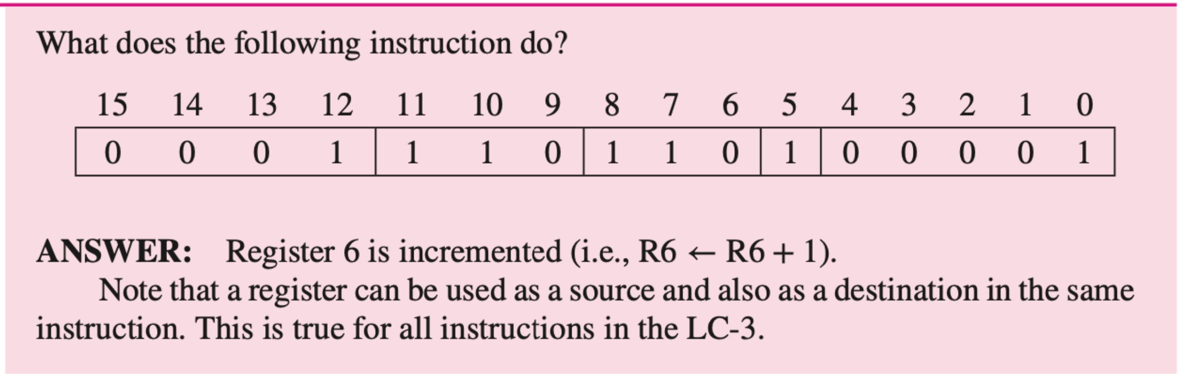
IR[4:0] = # -2
R5 = # 6
R1 = # 4



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

ANSWER: Register 2 is cleared (i.e., set to all 0s).

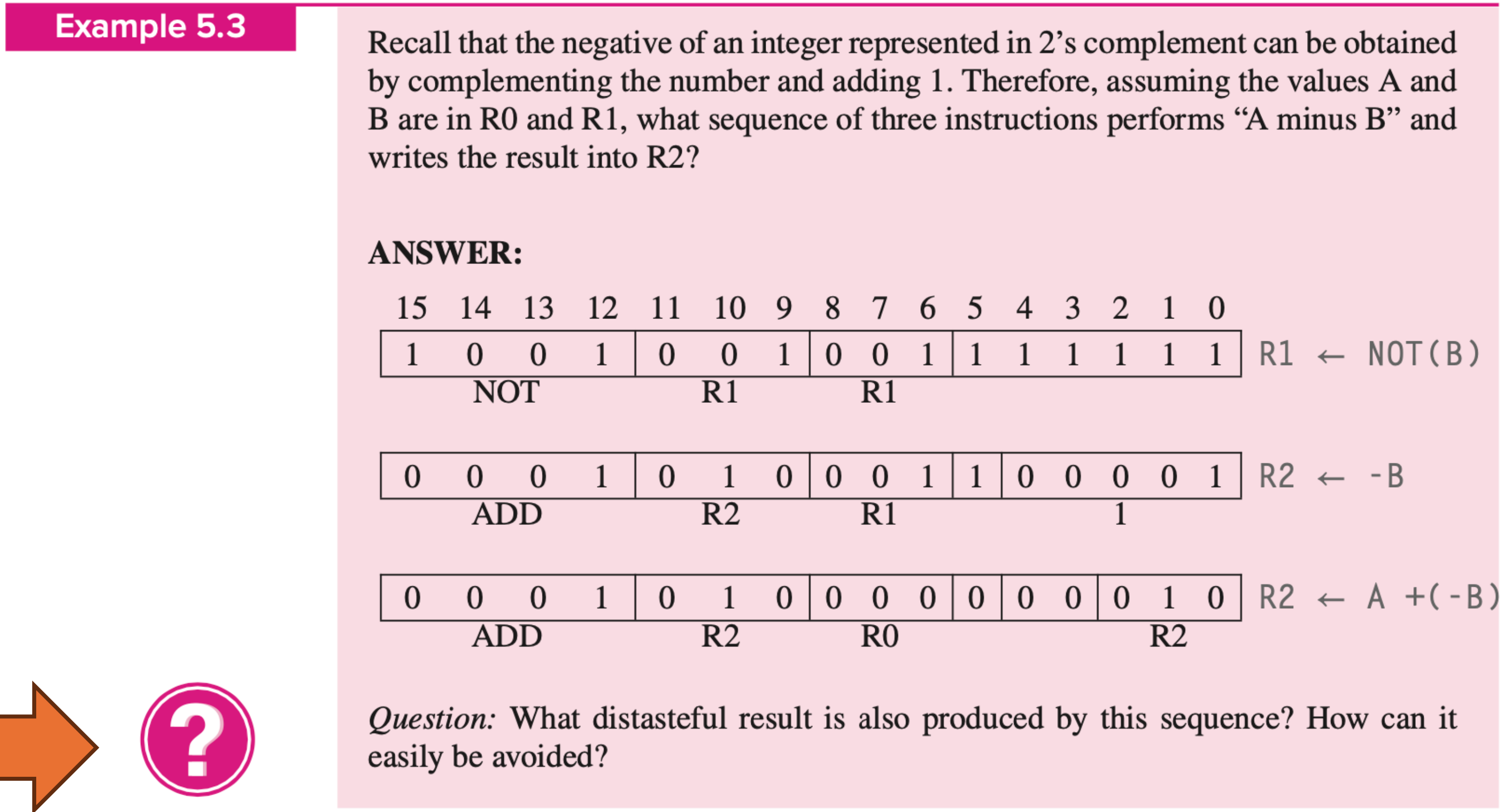


What does the following instruction do?

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

ANSWER: Register 6 is incremented (i.e., R6 ← R6 + 1).

Note that a register can be used as a source and also as a destination in the same instruction. This is true for all instructions in the LC-3.



Example 5.3

Recall that the negative of an integer represented in 2's complement can be obtained by complementing the number and adding 1. Therefore, assuming the values A and B are in R0 and R1, what sequence of three instructions performs "A minus B" and writes the result into R2?

ANSWER:

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	R1 ← NOT(B)
	NOT				R1			R1								
0	0	0	1	0	1	0	0	0	1	1	0	0	0	0	1	R2 ← -B
	ADD				R2			R1					1			
0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	R2 ← A +(-B)
	ADD				R2			R0						R2		

Question: What distasteful result is also produced by this sequence? How can it easily be avoided?

???

Using Operate Instructions

- With only ADD, AND, NOT…
  - How do we subtract?
  - How do we OR?
  - How do we copy from one register to another?
  - How do we initialize a register to zero?

Using Operate Instructions: Subtraction

- Goal: **`R1 <- R2 - R3`**
- Idea (2's complement)

```
1.  R1<-NOT(R3)
2.  R1<-R1+1
3.  R1<-R2+R1
```

- If 2nd operand is known and small, easy → **`R1 <- R2 + (-3)`**

Using Operate Instructions: OR

- Goal: **`R1 <- R2 OR R3`**
- Idea (DeMorgan’s Law)

```
A OR B = NOT( NOT(A) AND NOT(B) )
1.  R4<-NOT(R2)
2.  R5<-NOT(R3)
3.  R1<-R4 AND R5
4.  R5<-NOT(R1)
```

Using Operate Instructions: Copying

- Goal: **R1 <- R2**
- Idea (Use immediate)

```
R1 <- R2 + 0
```

Using Operate Instructions: Clearing

- Goal: **R1 <- 0**
- Idea

```
R1 <- R1 AND 0
```

Q & A