

RISC Design

Multi-cycle Implementation

Virendra Singh

Associate Professor

Computer Architecture and Dependable Systems Lab

Department of Electrical Engineering

Indian Institute of Technology Bombay

<http://www.ee.iitb.ac.in/~viren/>

E-mail: viren@ee.iitb.ac.in

EE-739: Processor Design



Lecture 17 (17 Feb 2015)

CADSL

Example Instruction Set: MIPS Subset

MIPS Instruction – Subset

❖ Arithmetic and Logical Instructions

➤ add, sub, or, and, slt

❖ Memory reference Instructions

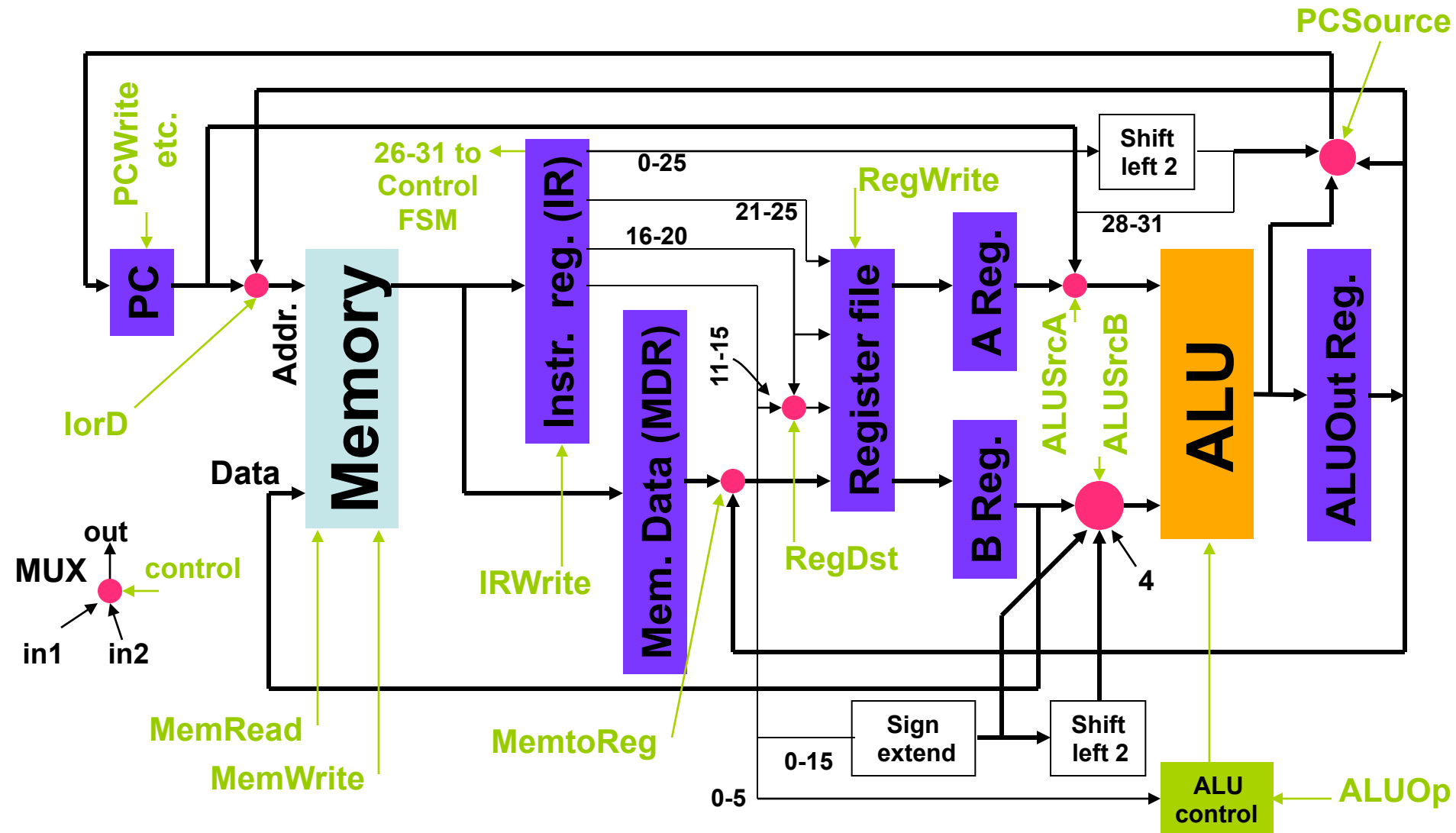
➤ lw, sw

❖ Branch

➤ beq, j



Multicycle Datapath



3 to 5 Cycles for an Instruction

Step	R-type (4 cycles)	Mem. Ref. (4 or 5 cycles)	Branch type (3 cycles)	J-type (3 cycles)
Instruction fetch	$IR \leftarrow \text{Memory}[PC]; PC \leftarrow PC+4$			
Instr. decode/ Reg. fetch	$A \leftarrow \text{Reg}(IR[21-25]); B \leftarrow \text{Reg}(IR[16-20])$ $ALUOut \leftarrow PC + (\text{sign extend } IR[0-15]) \ll 2$			
Execution, addr. Comp., branch & jump completion	$ALUOut \leftarrow A \text{ op } B$	$ALUOut \leftarrow A + \text{sign extend } (IR[0-15])$	If $(A = B)$ then $PC \leftarrow ALUOut$	$PC \leftarrow PC[28-31] \parallel (IR[0-25] \ll 2)$
Mem. Access or R-type completion	$\text{Reg}(IR[11-15]) \leftarrow ALUOut$	$MDR \leftarrow M[ALUOut]$ or $M[ALUOut] \leftarrow B$		
Memory read completion		$\text{Reg}(IR[16-20]) \leftarrow MDR$		



Cycle 1 of 5: Instruction Fetch (IF)

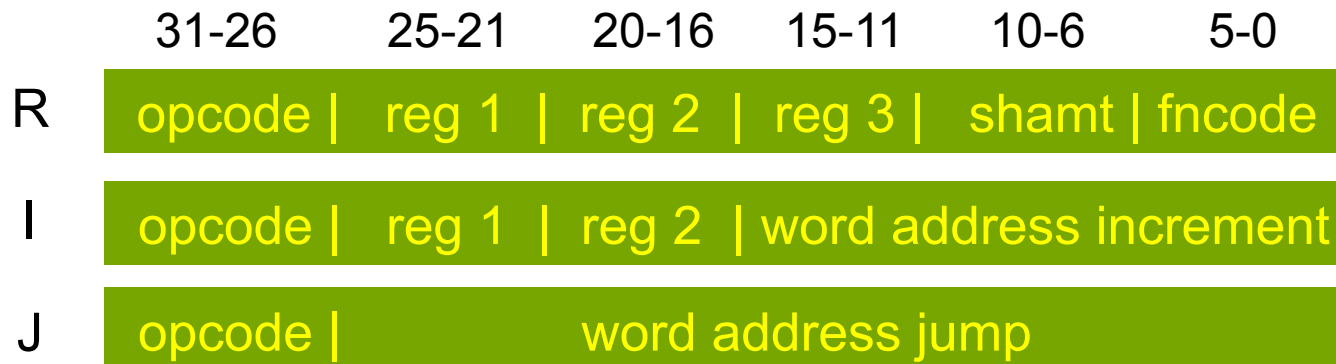
- Read instruction into IR, $M[PC] \rightarrow IR$
 - Control signals used:

» IorD	=	0	select PC
» MemRead	=	1	read memory
» IRWrite	=	1	write IR
- Increment PC, $PC + 4 \rightarrow PC$
 - Control signals used:

» ALUSrcA	=	0	select PC into ALU
» ALUSrcB	=	01	select constant 4
» ALUOp	=	00	ALU adds
» PCSource	=	00	select ALU output
» PCWrite	=	1	write PC



Cycle 2 of 5: Instruction Decode (ID)



- Control unit decodes instruction
- Datapath prepares for execution
 - R and I types, reg 1 → A reg, reg 2 → B reg
 - » No control signals needed
 - Branch type, compute branch address in ALUOut
 - » ALUSrcA = 0 select PC into ALU
 - » ALUSrcB = 11 Instr. Bits 0-15 shift 2 into ALU
 - » ALUOp = 00 ALU adds



Cycle 3 of 5: Execute (EX)

- R type: execute function on reg A and reg B, result in ALUOut
 - Control signals used:
 - » ALUSrcA = 1 A reg into ALU
 - » ALUsrcB = 00 B reg into ALU
 - » ALUOp = 10 instr. Bits 0-5 control ALU
- I type, lw or sw: compute memory address in ALUOut \leftarrow A reg + sign extend IR[0-15]
 - Control signals used:
 - » ALUSrcA = 1 A reg into ALU
 - » ALUSrcB = 10 Instr. Bits 0-15 into ALU
 - » ALUOp = 00 ALU adds



Cycle 3 of 5: Execute (EX)

- I type, beq: subtract reg A and reg B, write ALUOut to PC

- Control signals used:

» ALUSrcA	=	1	A reg into ALU
» ALUsrcB	=	00	B reg into ALU
» ALUOp	=	01	ALU subtracts
» If zero = 1, PCSource	=	01	ALUOut to PC
» If zero = 1, PCwriteCond	=	1	write PC
» Instruction complete, go to IF			

- J type: write jump address to PC \leftarrow IR[0-25] shift 2 and four leading bits of PC

- Control signals used:

» PCSource	=	10	
» PCWrite	=	1	write PC

» Instruction complete, go to IF



Cycle 4 of 5: Reg Write/Memory

- R type, write destination register from ALUOut
 - Control signals used:
 - » **RegDst** = 1 Instr. Bits 11-15 specify reg.
 - » **MemtoReg** = 0 ALUOut into reg.
 - » **RegWrite** = 1 write register
 - » **Instruction complete, go to IF**
- I type, lw: read M[ALUOut] into MDR
 - Control signals used:
 - » **lorD** = 1 select ALUOut into mem adr.
 - » **MemRead** = 1 read memory to MDR
- I type, sw: write M[ALUOut] from B reg
 - Control signals used:
 - » **lorD** = 1 select ALUOut into mem adr.
 - » **MemWrite** = 1 write memory
 - » **Instruction complete, go to IF**



Cycle 5 of 5: Reg Write

- I type, lw: write MDR to reg[IR(16-20)]
 - Control signals used:
 - » RegDst = 0 instr. Bits 16-20 are write reg
 - » MemtoReg = 1 MDR to reg file write input
 - » RegWrite = 1 read memory to MDR
 - » **Instruction complete, go to IF**

For an alternative method of designing datapath, see
N. Tredennick, *Microprocessor Logic Design, the Flowchart Method*,
Digital Press, 1987.



1-bit Control Signals

Signal name	Value = 0	Value =1
RegDst	Write reg. # = bit 16-20	Write reg. # = bit 11-15
RegWrite	No action	Write reg. \leftarrow Write data
ALUSrcA	First ALU Operand \leftarrow PC	First ALU Operand \leftarrow Reg. A
MemRead	No action	Mem.Data Output \leftarrow M[Addr.]
MemWrite	No action	M[Addr.] \leftarrow Mem. Data Input
MemtoReg	Reg.File Write In \leftarrow ALUOut	Reg.File Write In \leftarrow MDR
IorD	Mem. Addr. \leftarrow PC	Mem. Addr. \leftarrow ALUOut
IRWrite	No action	IR \leftarrow Mem.Data Output
PCWrite	No action	PC is written
PCWriteCond	No action	PC is written if zero(ALU)=1

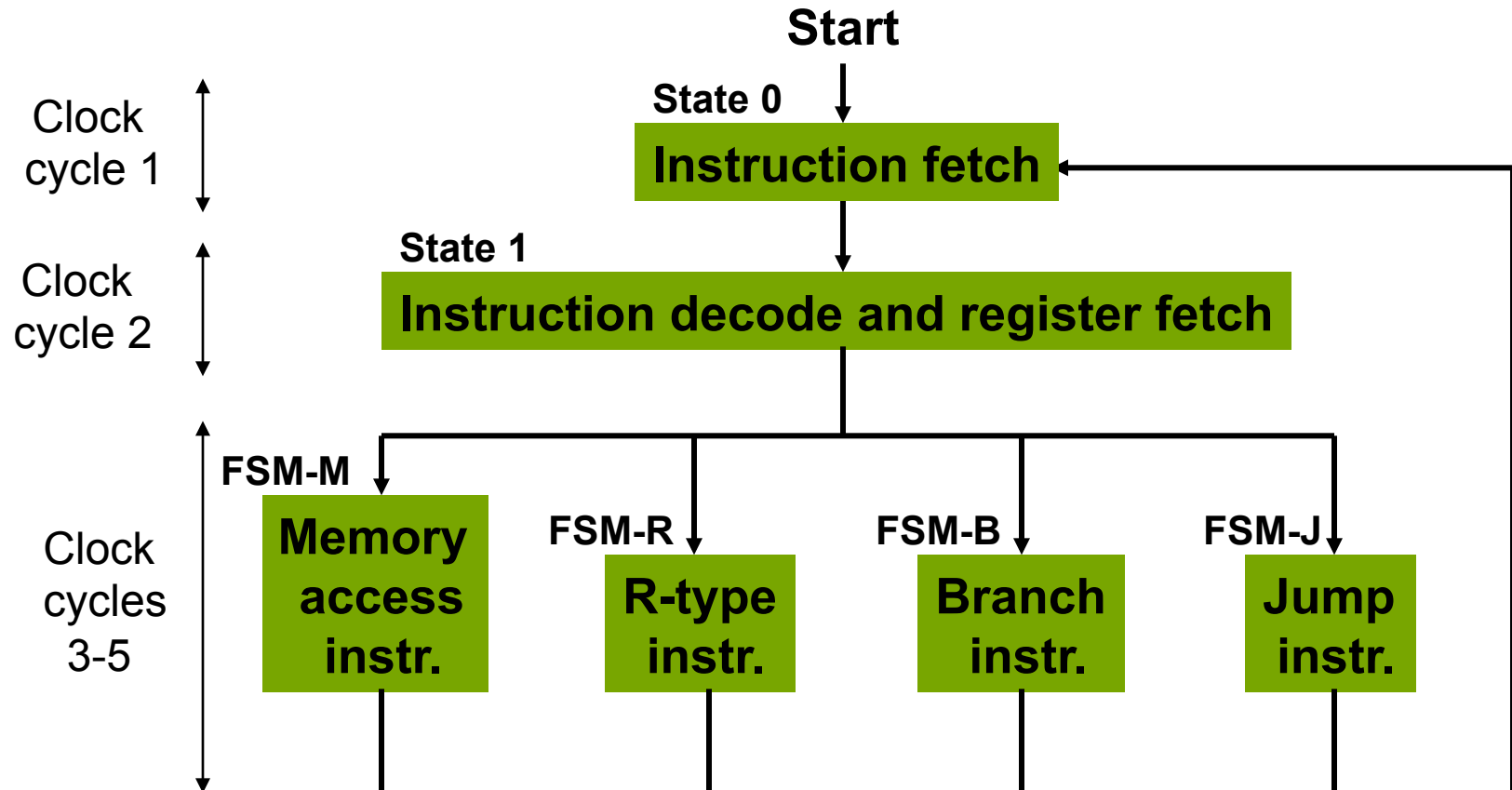


2-bit Control Signals

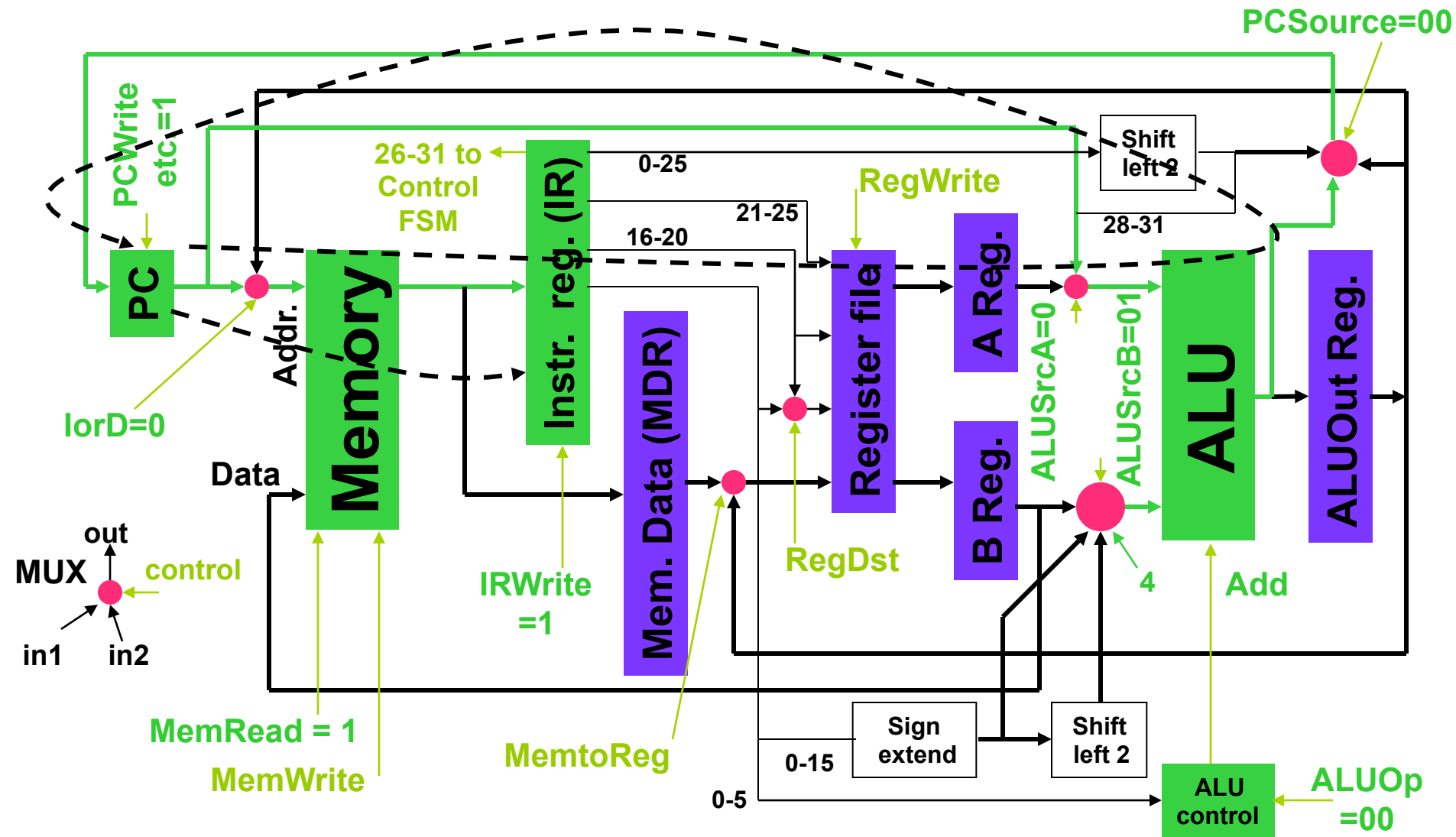
Signal name	Value	Action
ALUOp	00	ALU performs add
	01	ALU performs subtract
	10	Funct. field (0-5 bits of IR) determines ALU operation
ALUSrcB	00	Second input of ALU \leftarrow B reg.
	01	Second input of ALU \leftarrow 4 (constant)
	10	Second input of ALU \leftarrow 0-15 bits of IR sign ext. to 32b
	11	Second input of ALU \leftarrow 0-15 bits of IR sign ext. and left shift 2 bits
PCSource	00	ALU output (PC +4) sent to PC
	01	ALUOut (branch target addr.) sent to PC
	10	Jump address IR[0-25] shifted left 2 bits, concatenated with PC+4[28-31], sent to PC



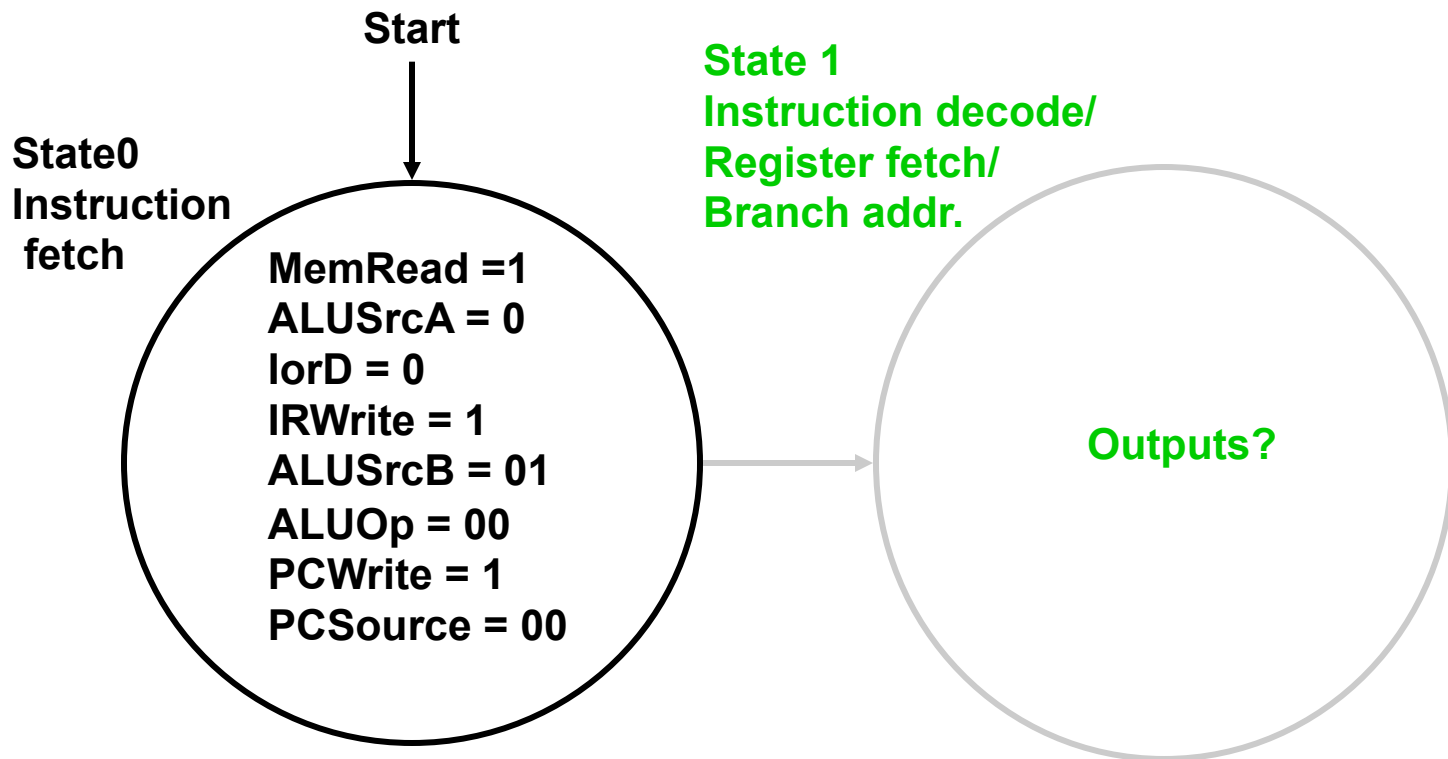
Control: Finite State Machine



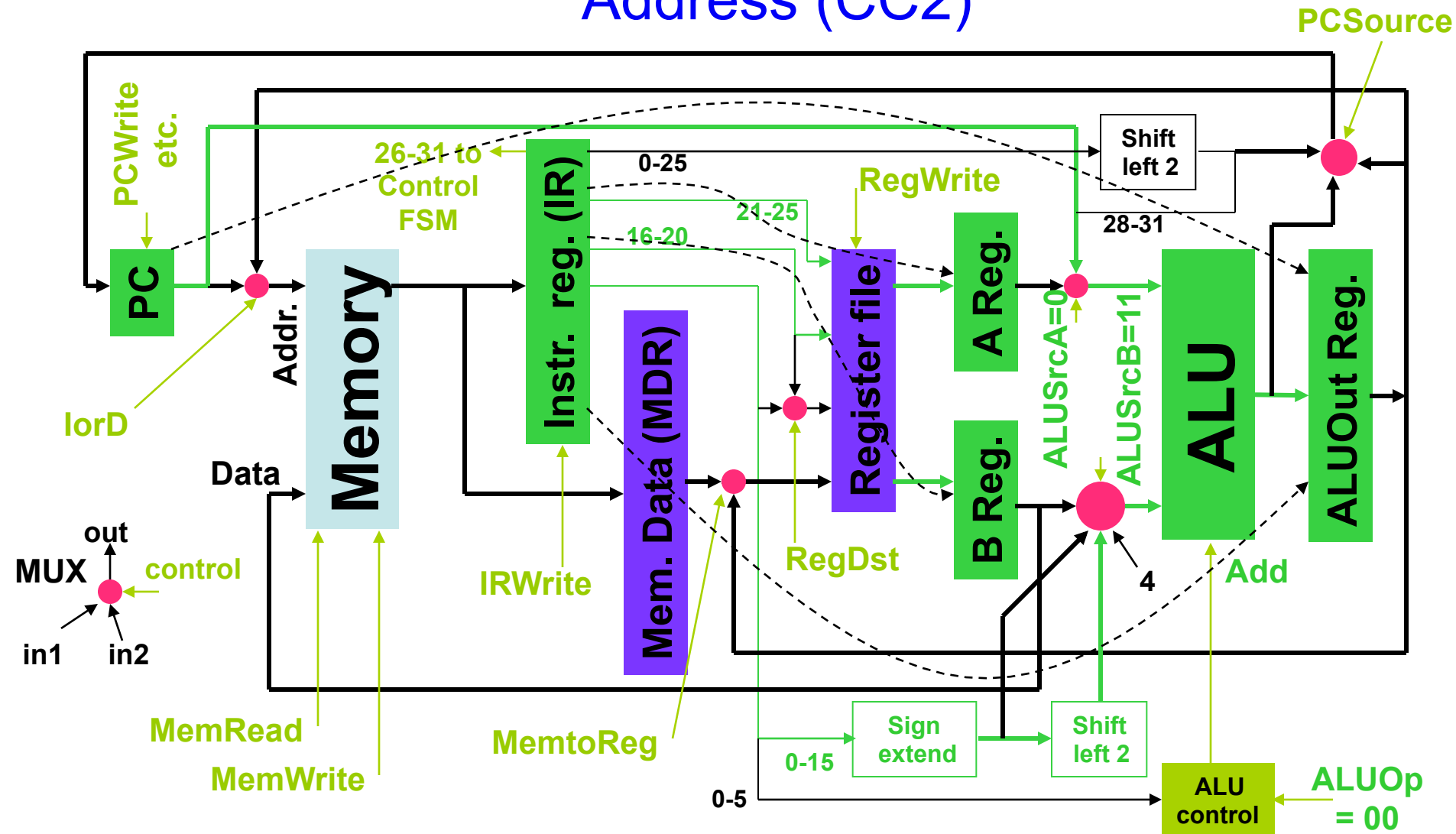
State 0: Instruction Fetch (CC1)



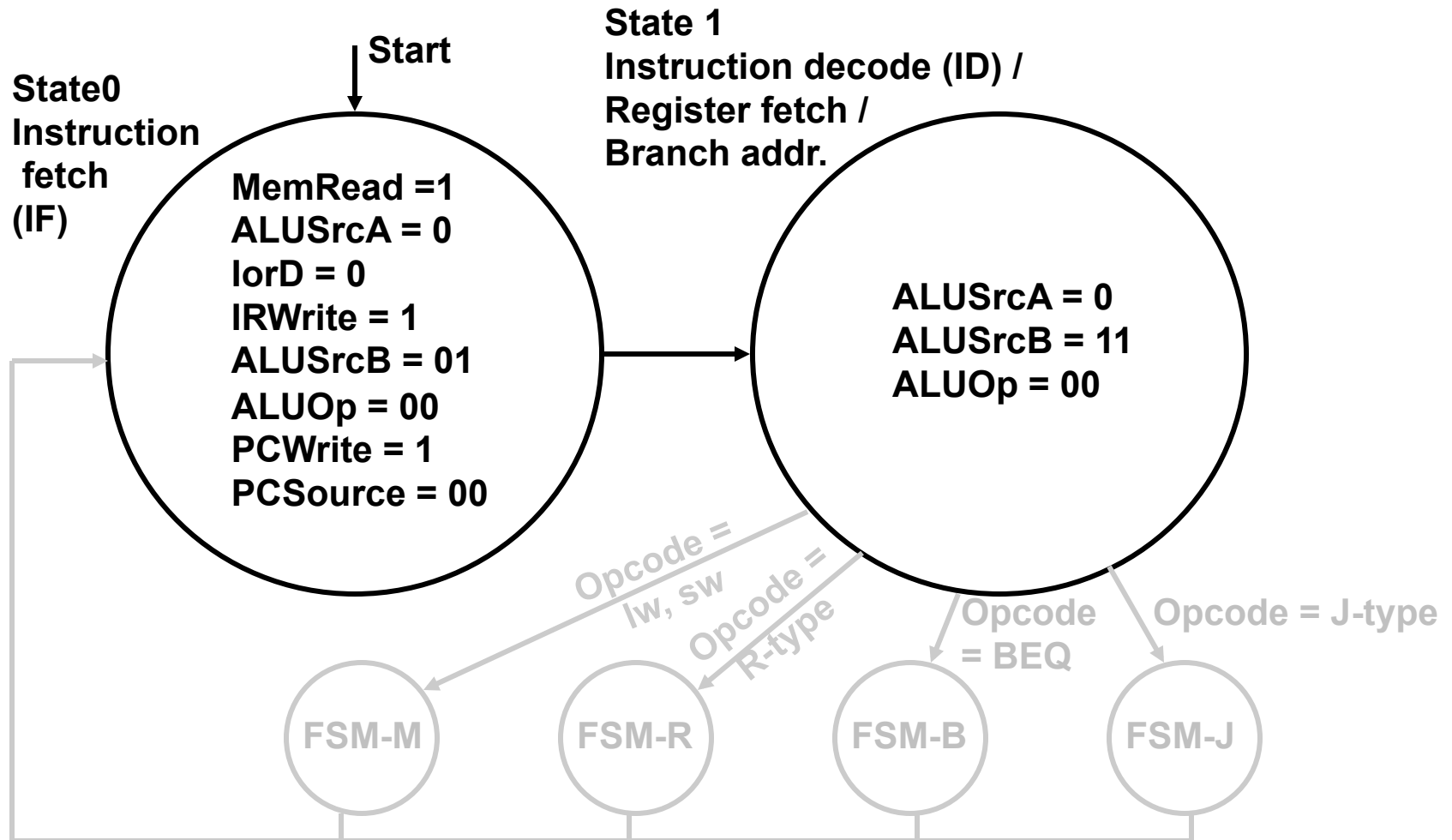
State 0 Control FSM Outputs



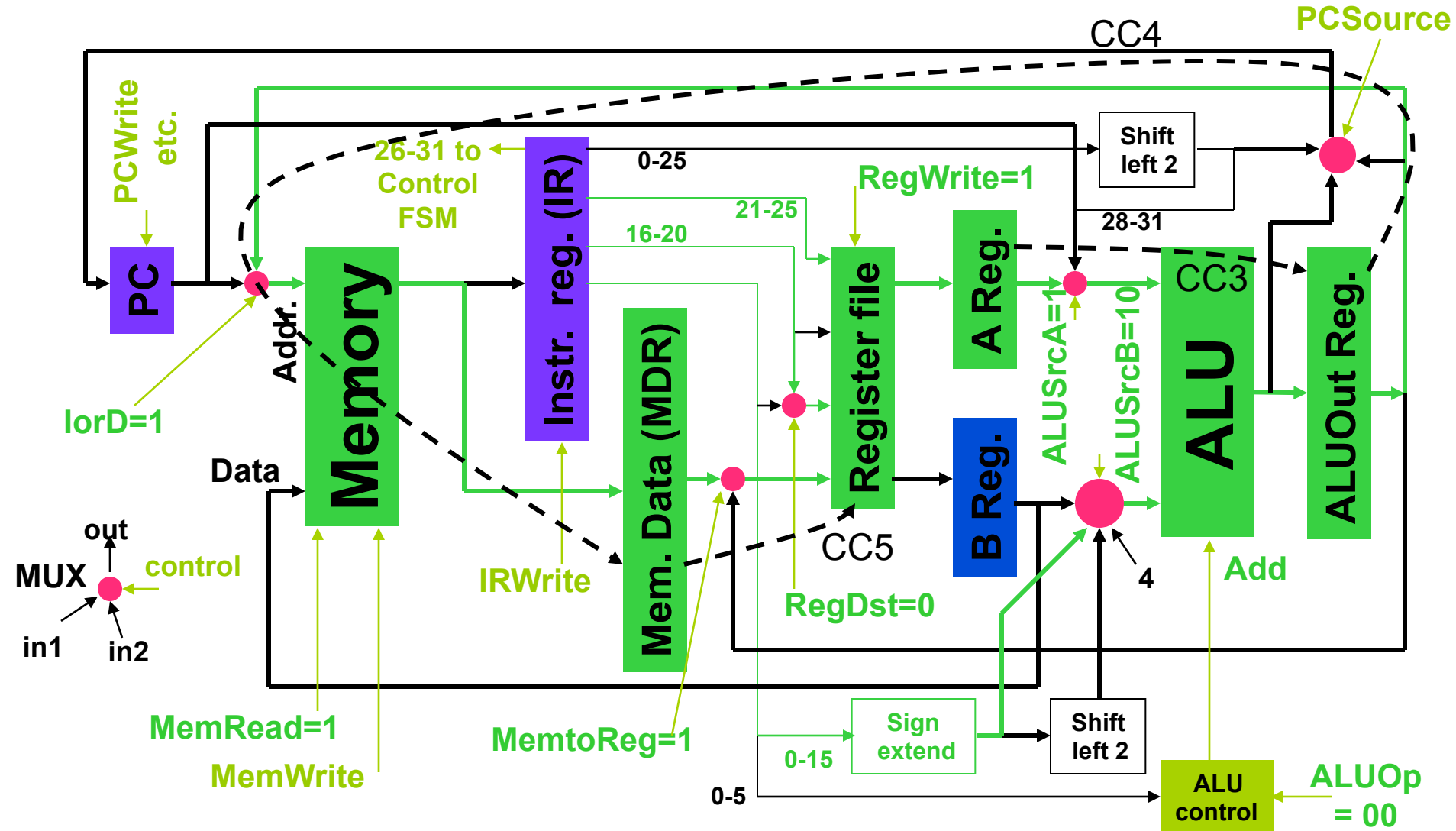
State 1: Instr. Decode/Reg. Fetch/ Branch Address (CC2)



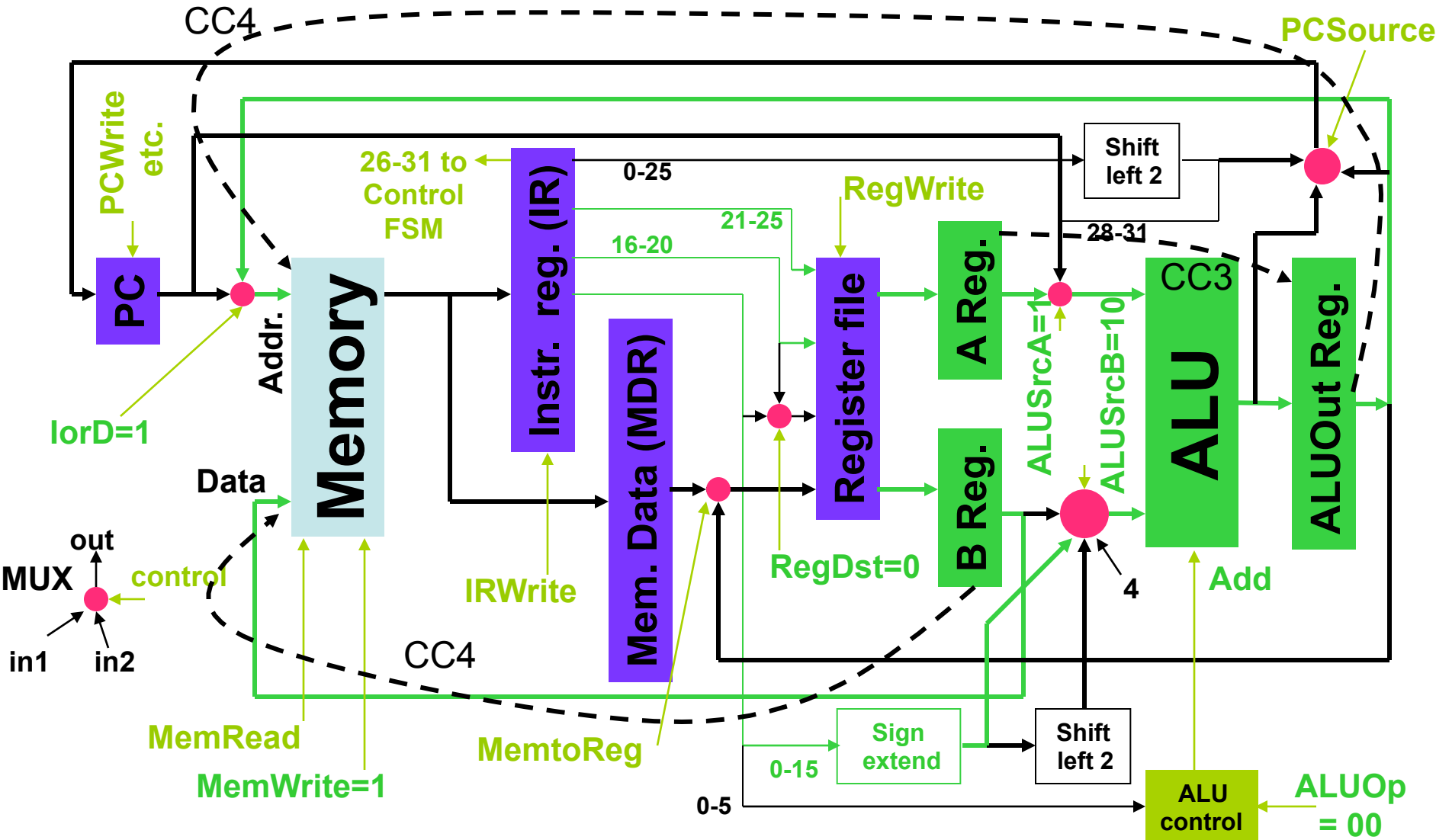
State 1 Control FSM Outputs



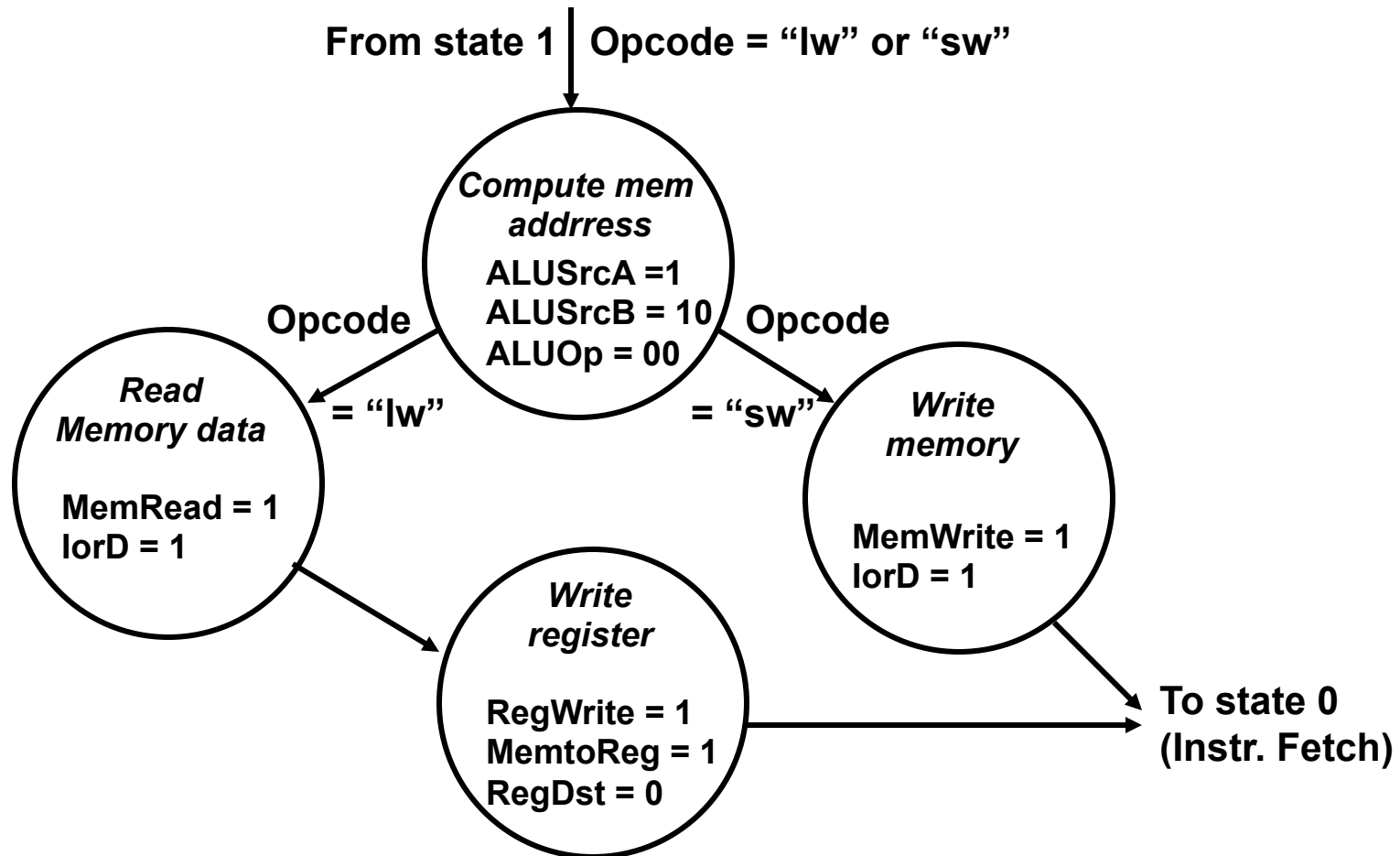
State 1 (Opcode = lw) → FSM-M (CC3-5)



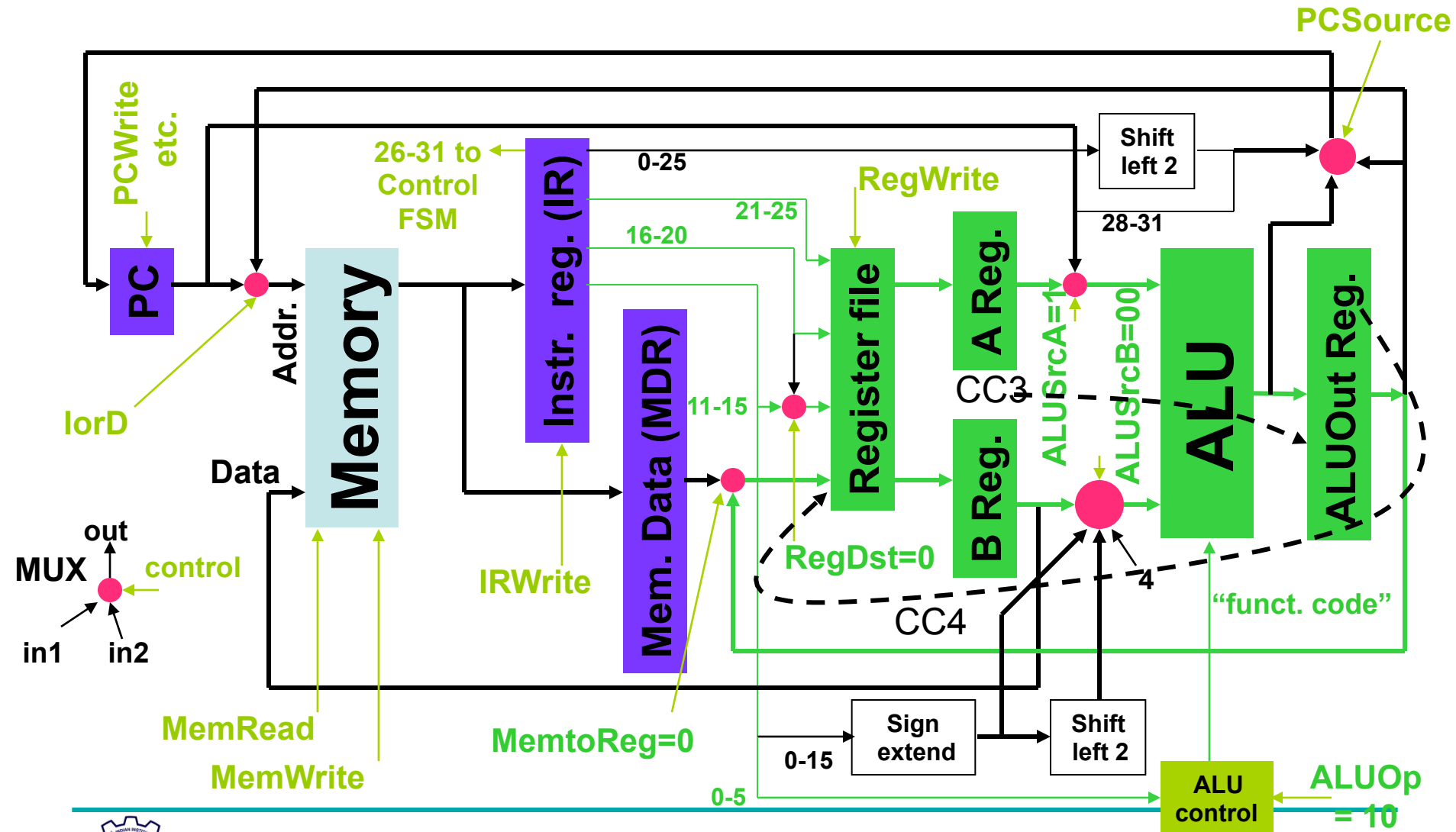
State 1 (Opcode= sw)→FSM-M (CC3-4)



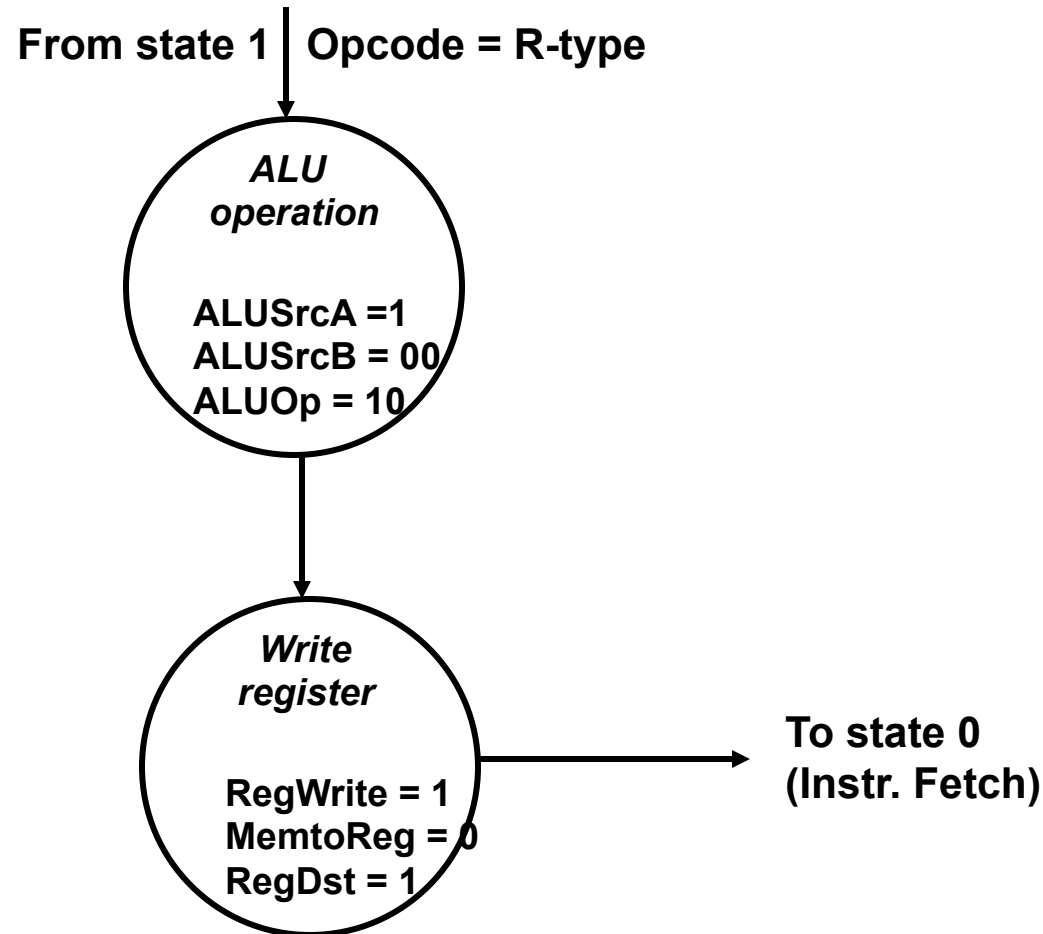
FSM-M (Memory Access)



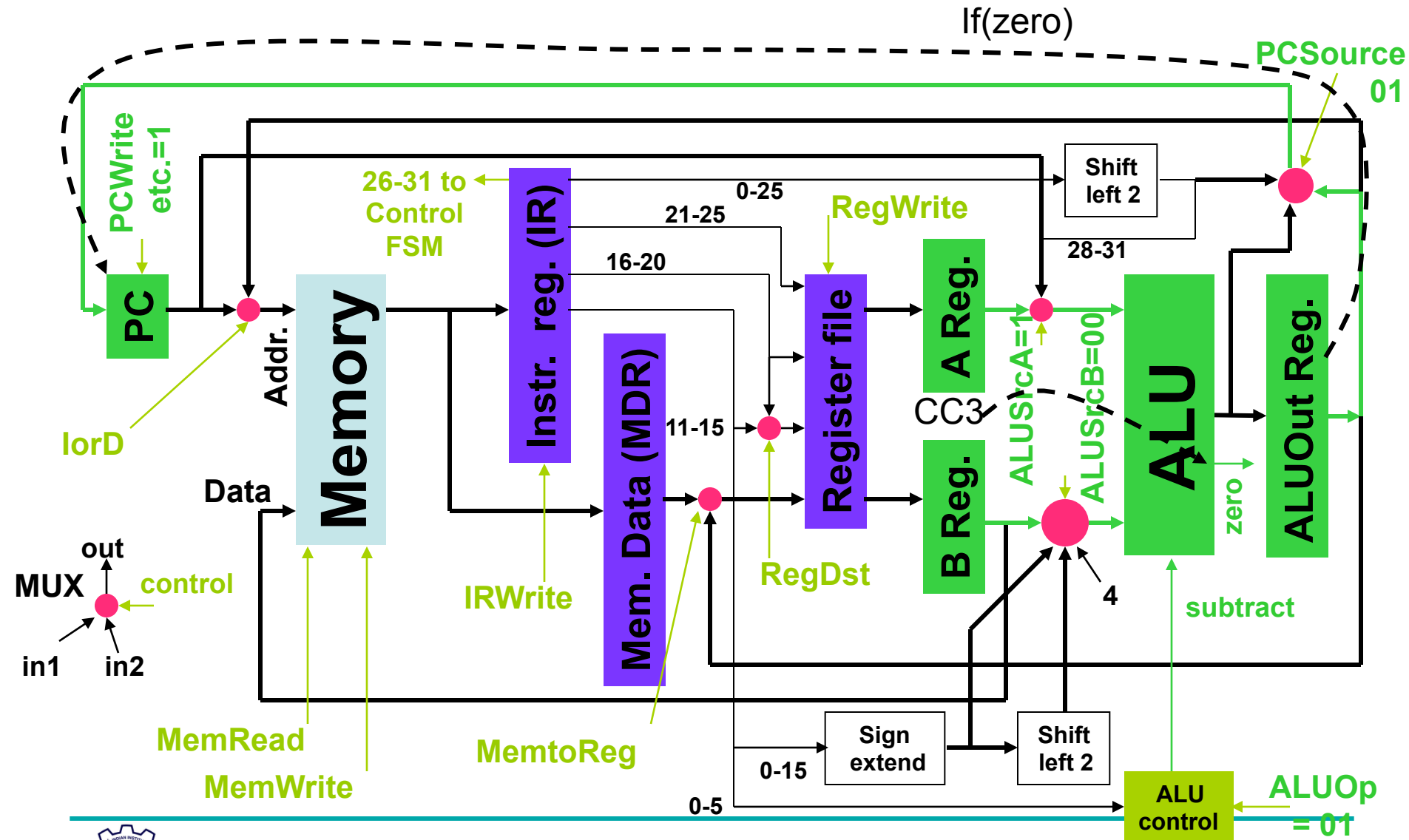
State 1 (Opcode=R-type) → FSM-R (CC3-4)



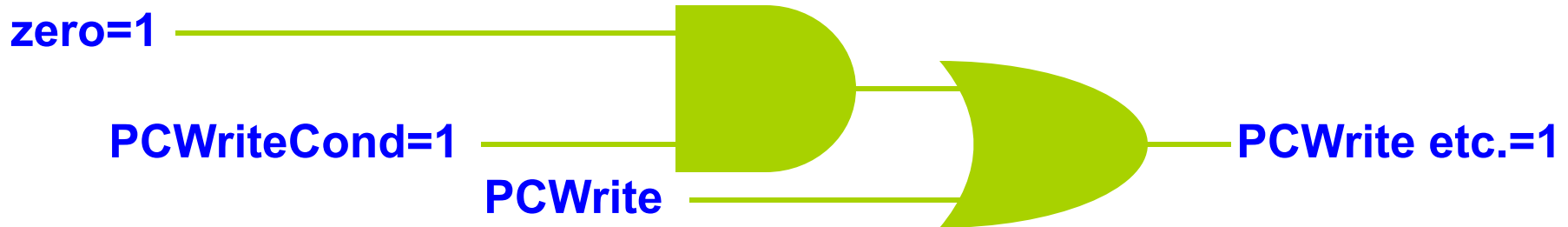
FSM-R (R-type Instruction)



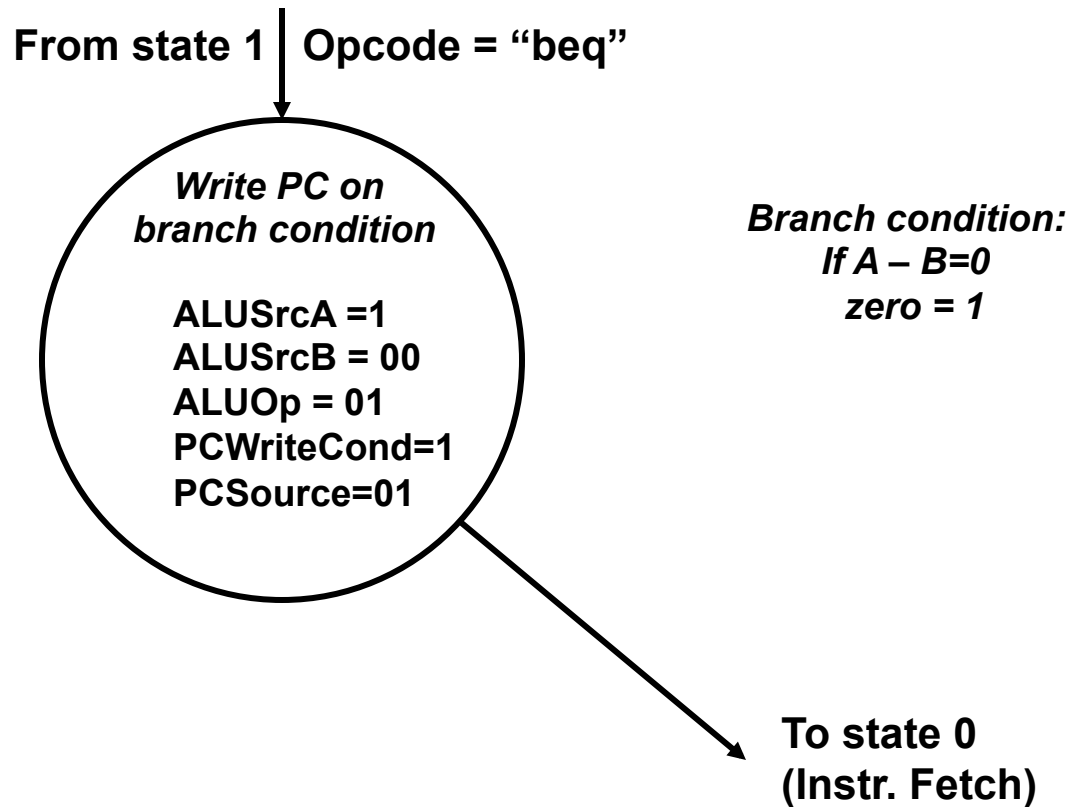
State 1 (Opcode = beq) → FSM-B (CC3)



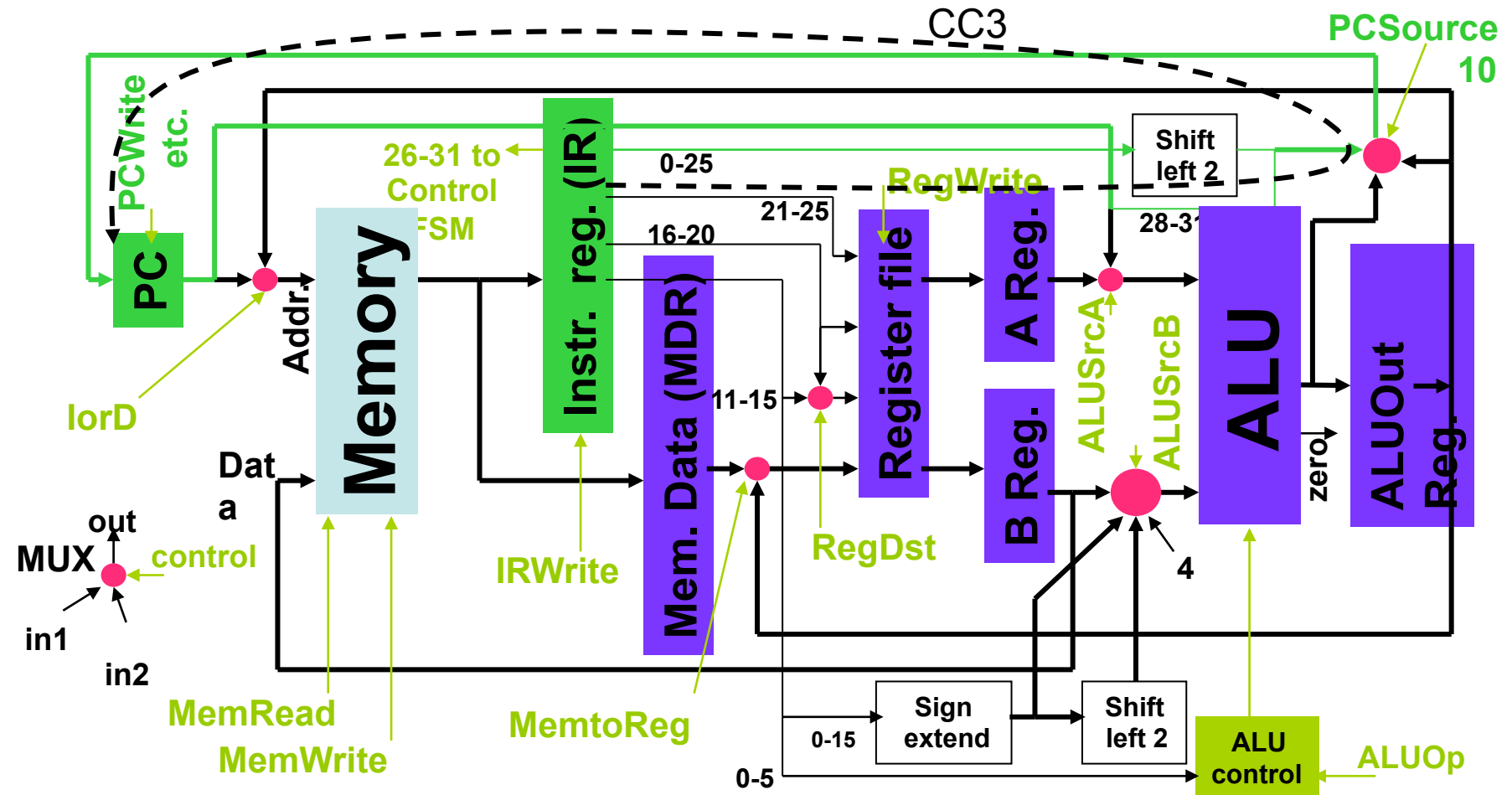
Write PC on “zero”



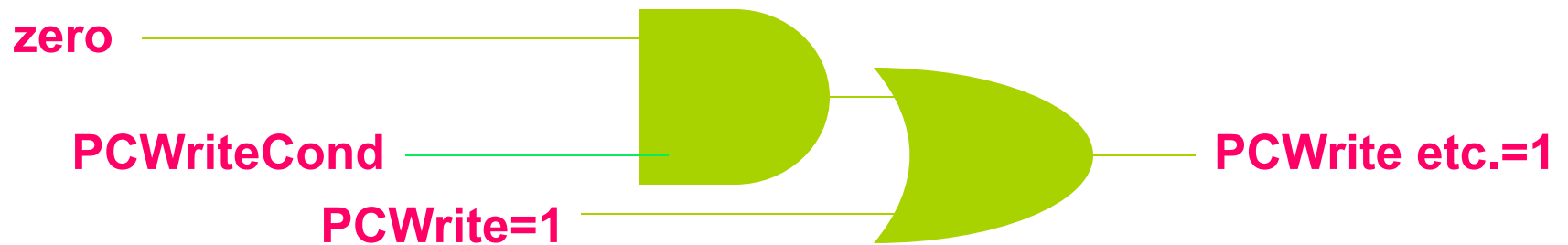
FSM-B (Branch)



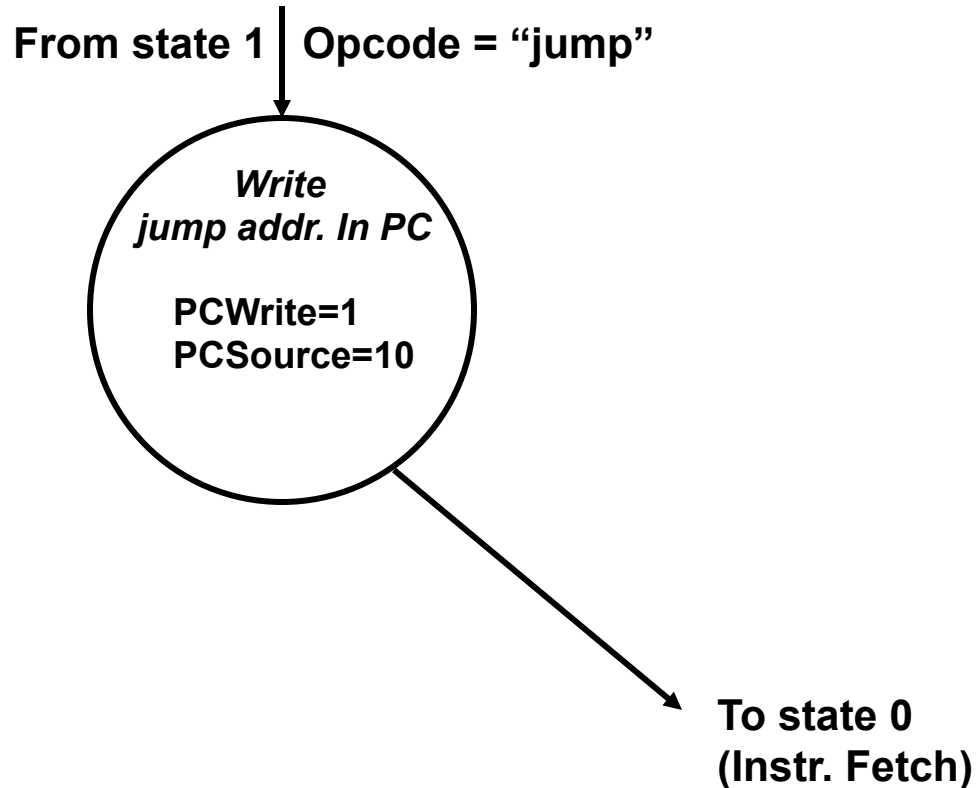
State 1 (Opcode = j) → FSM-J (CC3)



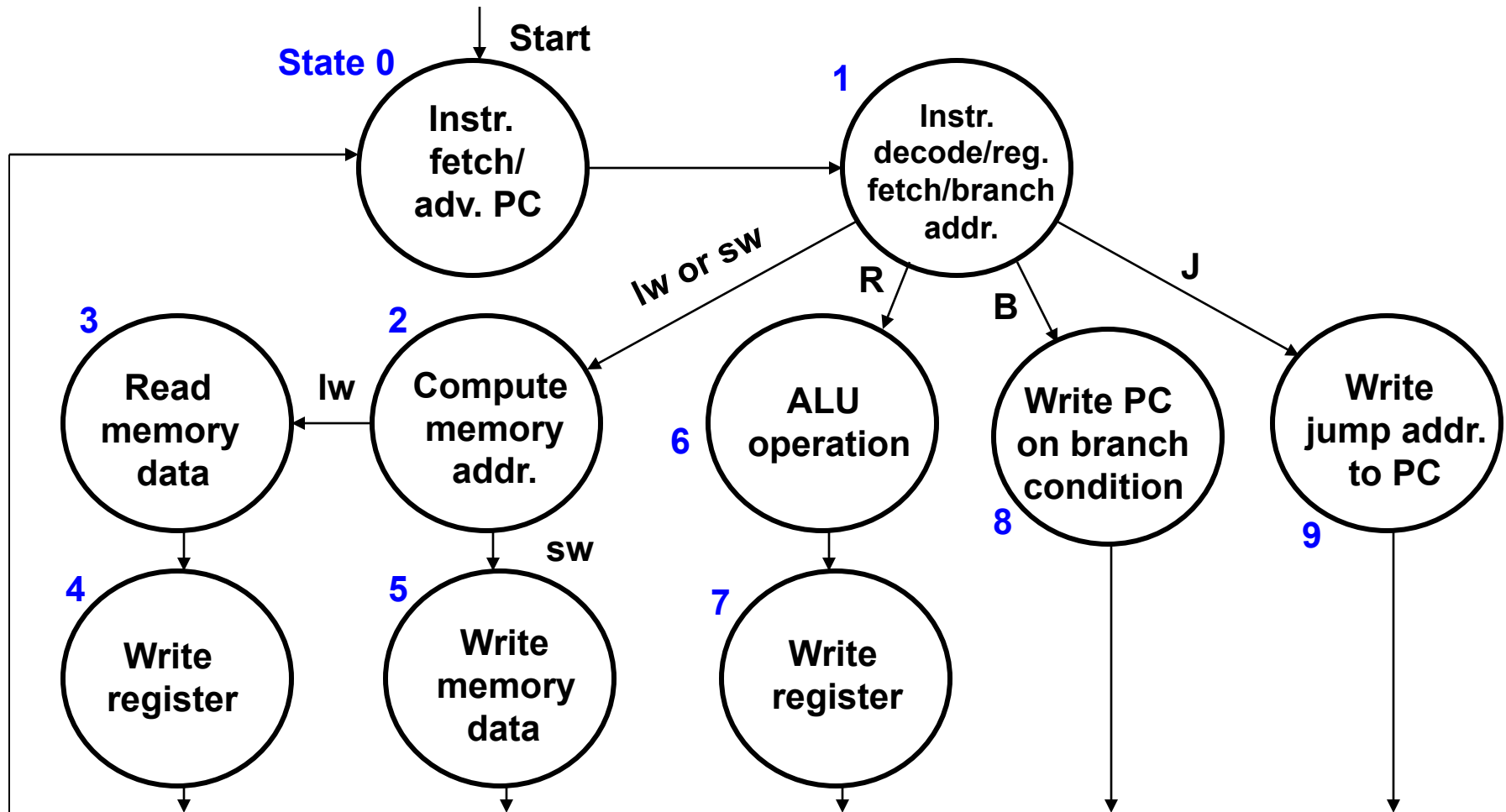
Write PC



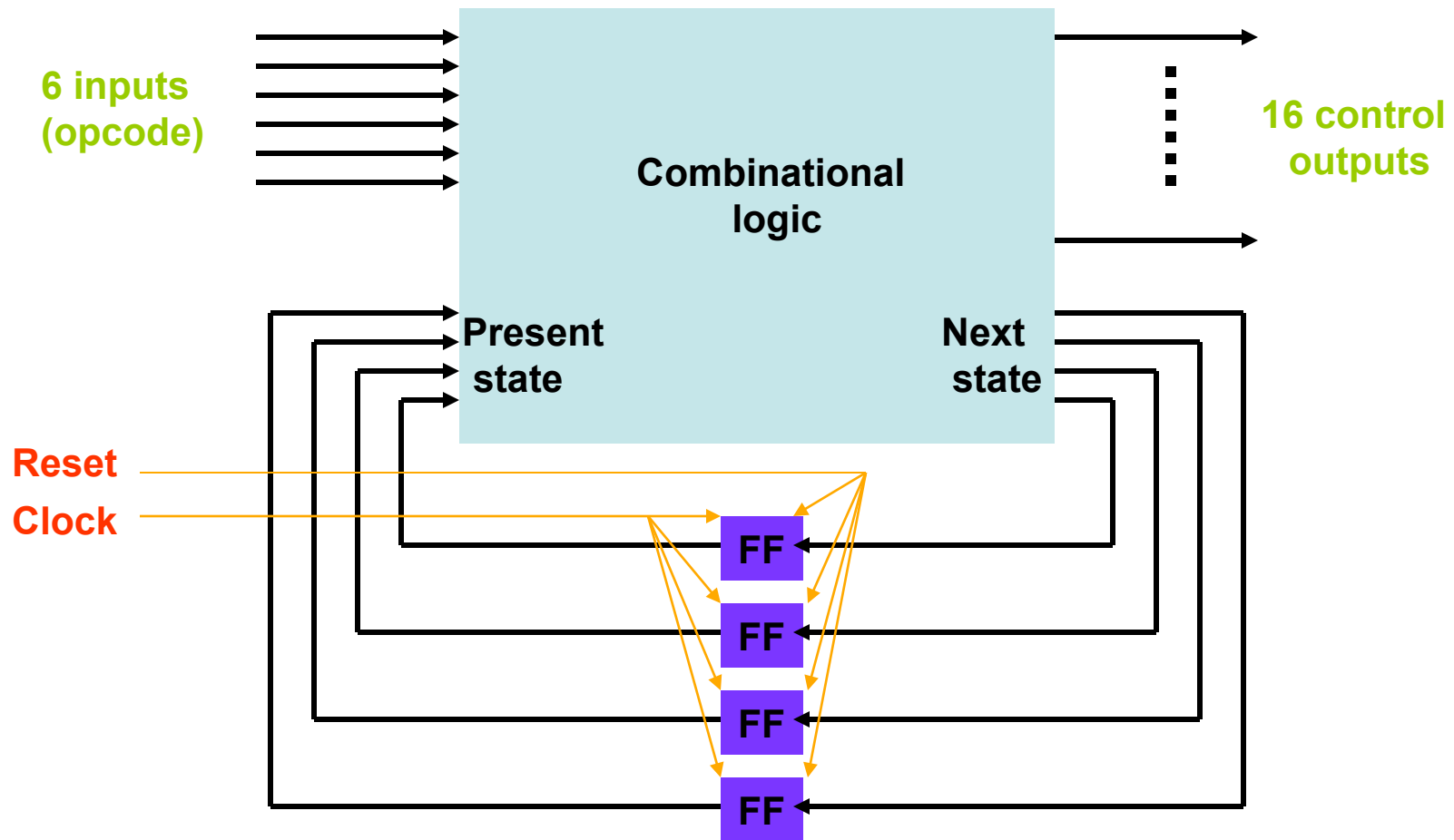
FSM-J (Jump)



Control FSM



Control FSM (Controller)



Designing the Control FSM

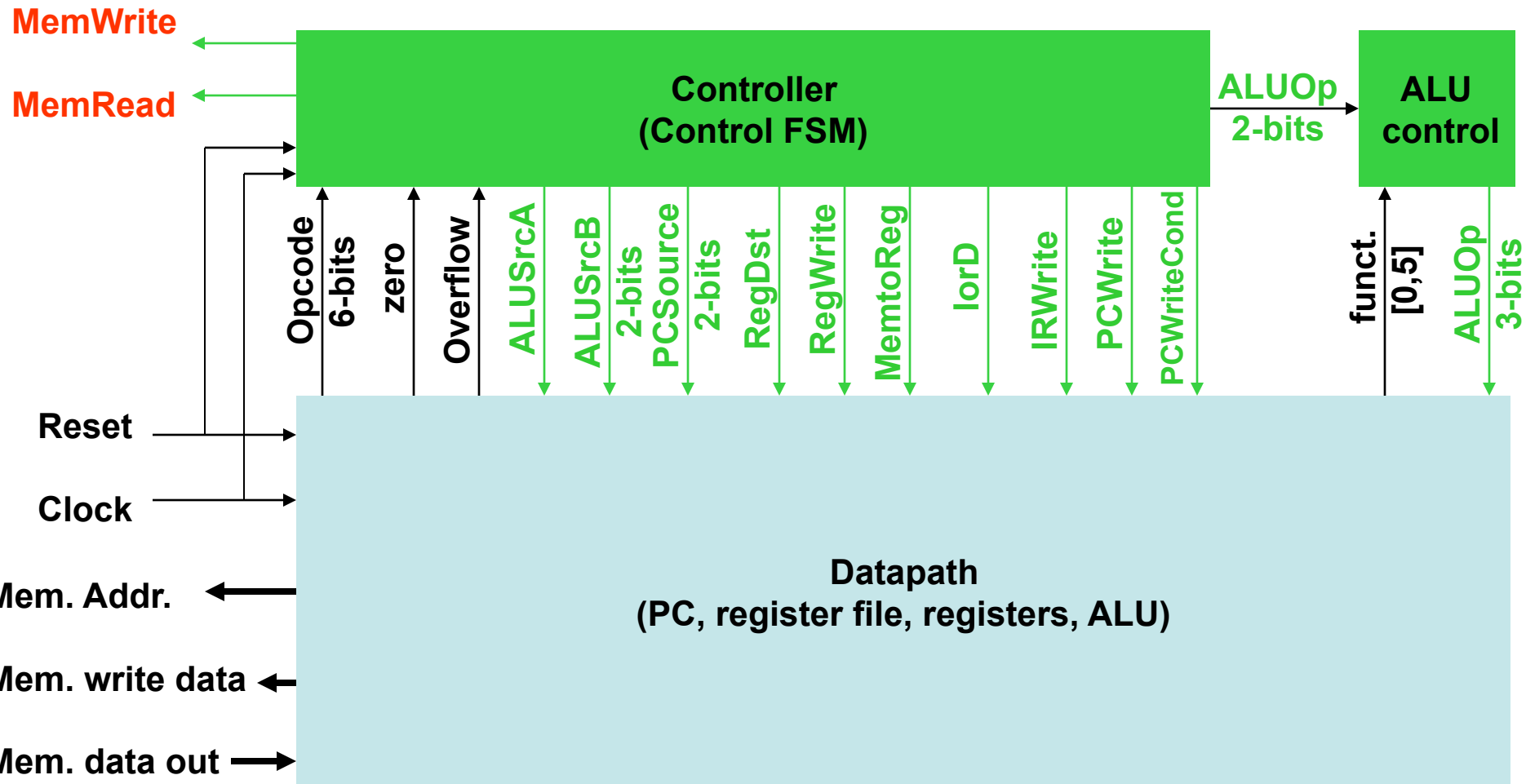
- Encode states; need 4 bits for 10 states, e.g.,
 - State 0 is 0000, state 1 is 0001, and so on.
- Write a truth table for combinational logic:

Opcode	Present state	Control signals	Next state
000000	0000	0001000110000100	0001
.....	

- Synthesize a logic circuit from the truth table.
- Connect four flip-flops between the next state outputs and present state inputs.



Block Diagram of a Processor

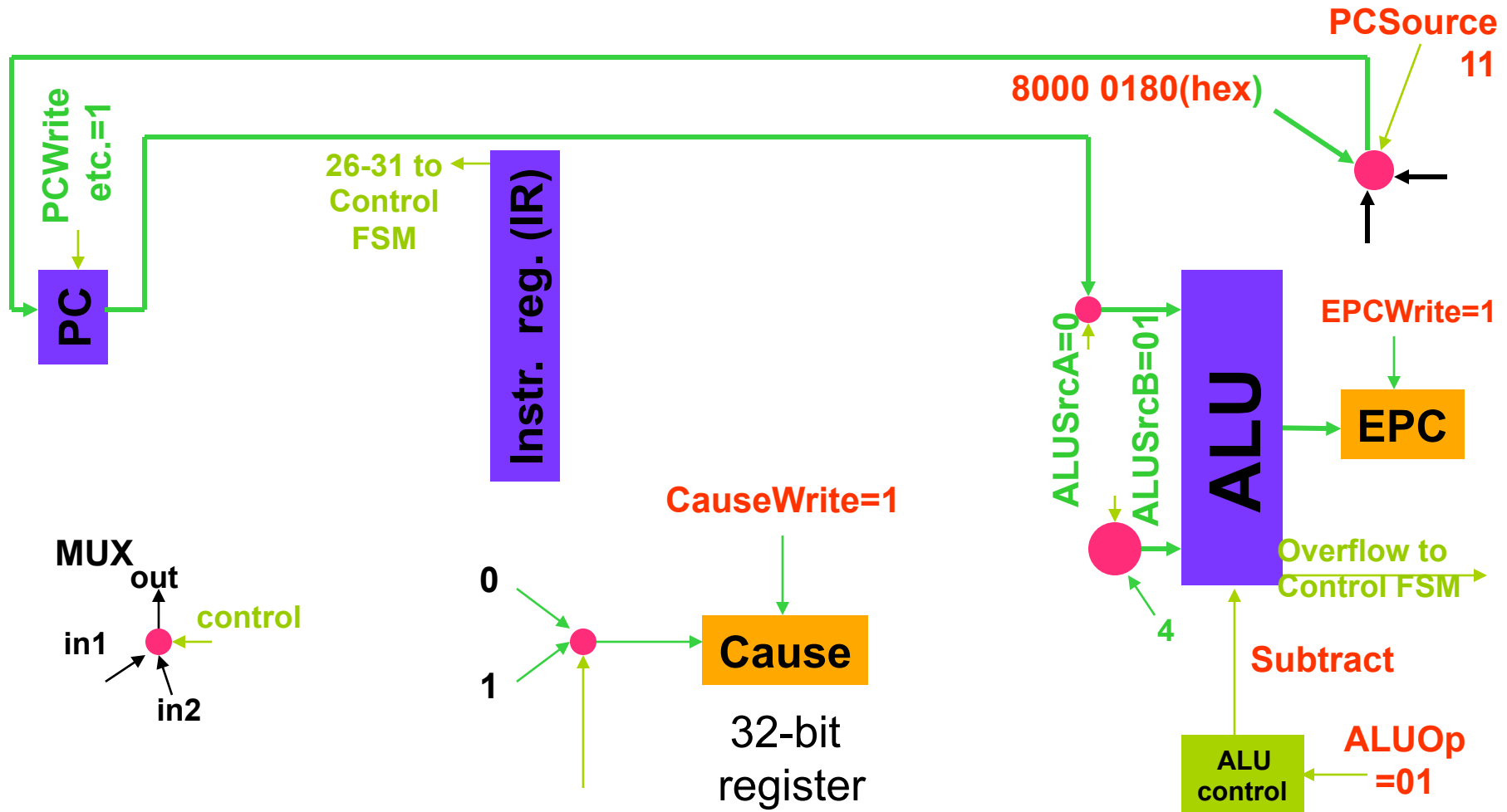


Exceptions or Interrupts

- Conditions under which the processor may produce incorrect result or may “hang”.
 - Illegal or undefined opcode.
 - Arithmetic overflow, divide by zero, etc.
 - Out of bounds memory address.
- EPC: 32-bit register holds the affected instruction address.
- Cause: 32-bit register holds an encoded exception type. For example,
 - 0 for undefined instruction
 - 1 for arithmetic overflow



Implementing Exceptions



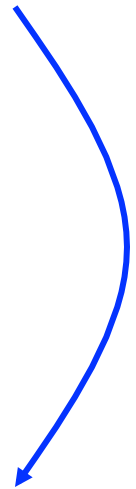
How Long Does It Take? Again

- Assume control logic is fast and does not affect the critical timing. Major time components are ALU, memory read/write, and register read/write.
- Time for hardware operations, suppose
 - Memory read or write 2ns
 - Register read 1ns
 - ALU operation 2ns
 - Register write 1ns



Single-Cycle Datapath

- R-type 6ns
- Load word (I-type) 8ns
- Store word (I-type) 7ns
- Branch on equal (I-type) 5ns
- Jump (J-type) 2ns
- Clock cycle time = 8ns
- Each instruction takes *one* cycle



Multicycle Datapath

- Clock cycle time is determined by the longest operation, ALU or memory:

- Clock cycle time = 2ns

- Cycles per instruction (CPI):

• lw	5	(10ns)
• sw	4	(8ns)
• R-type	4	(8ns)
• beq	3	(6ns)
• j	3	(6ns)



CPI of a Computer

$$\text{CPI} = \frac{\sum_k (\text{Instructions of type } k) \times \text{CPI}_k}{\sum_k (\text{instructions of type } k)}$$

where

$$\text{CPI}_k = \text{Cycles for instruction of type } k$$

Note: CPI is dependent on the instruction mix of the program being run. Standard benchmark programs are used for specifying the performance of CPUs.



Example

- Consider a program containing:
 - loads 25%
 - stores 10%
 - branches 11%
 - jumps 2%
 - Arithmetic 52%
- $$\text{CPI} = 0.25 \times 5 + 0.10 \times 4 + 0.11 \times 3 + 0.02 \times 3 + 0.52 \times 4$$
$$= 4.12 \text{ for multicycle datapath}$$
- $\text{CPI} = 1.00$ for single-cycle datapath



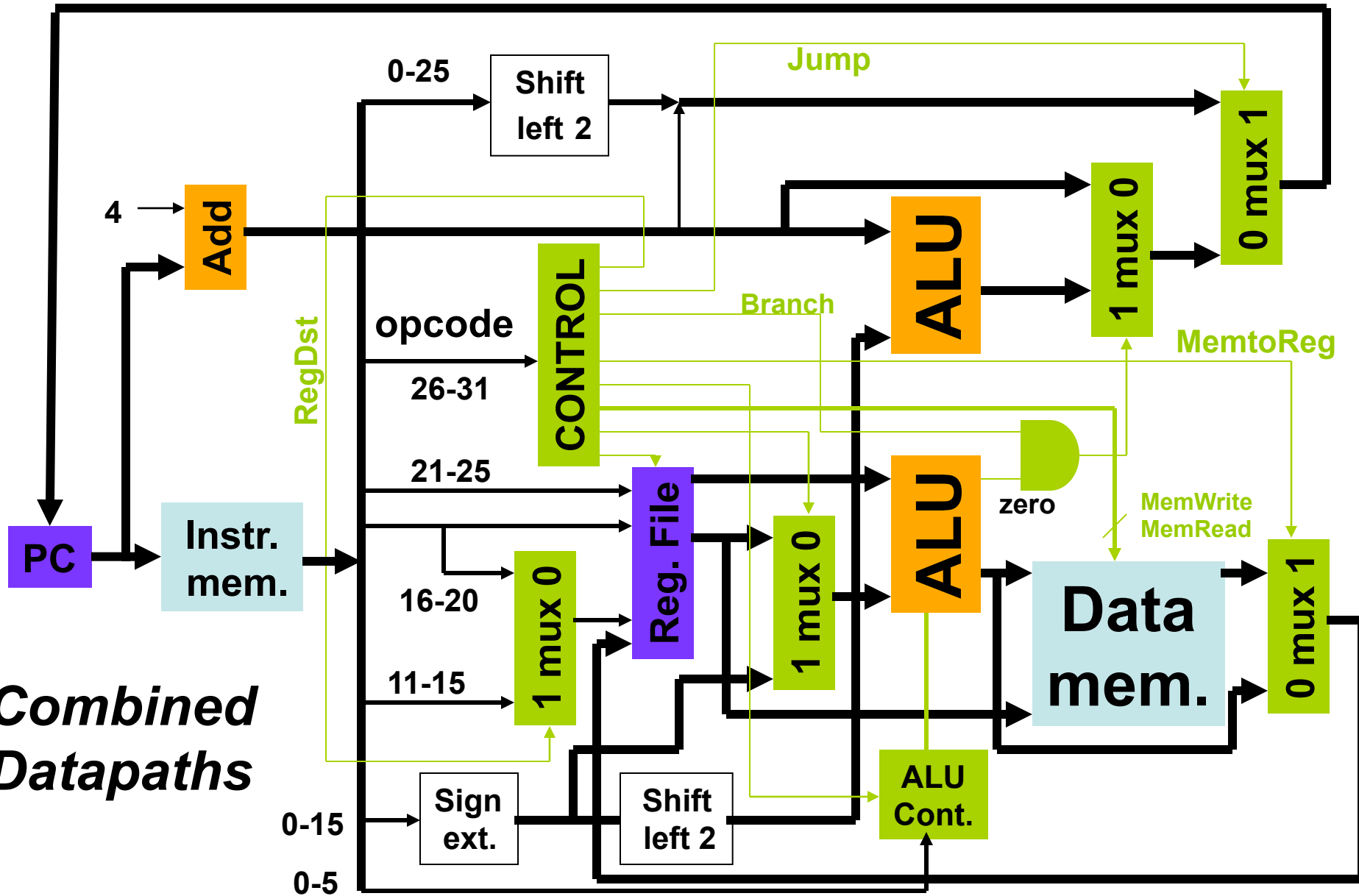
Multicycle vs. Single-Cycle

$$\begin{aligned}\text{Performance ratio} &= \text{Single cycle time} / \text{Multicycle time} \\ &= \frac{(\text{CPI} \times \text{cycle time}) \text{ for single-cycle}}{(\text{CPI} \times \text{cycle time}) \text{ for multicycle}} \\ &= \frac{1.00 \times 8\text{ns}}{4.12 \times 2\text{ns}} = 0.97\end{aligned}$$

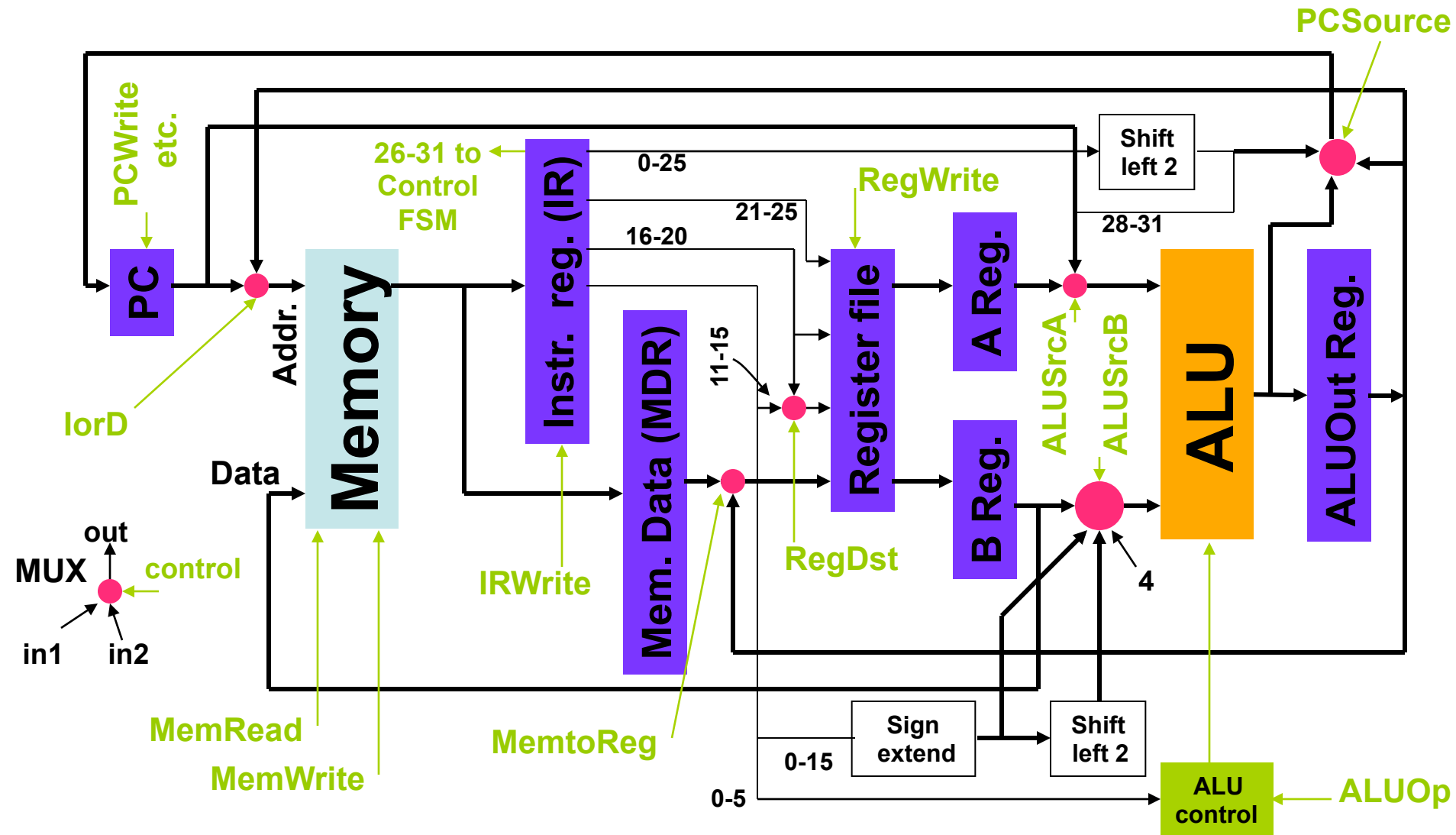
Single cycle is faster in this case, but remember, performance ratio depends on the instruction mix.



Combined Datapaths



Multicycle Datapath



Traffic Flow



Thank You

