

# Computer Architecture

## RISC Design

---

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](mailto:viren@ee.iitb.ac.in)

*CP-226: Computer Architecture*

---



Lecture 7 (10 Feb 2013)

CADSL

# Example processor: MIPS subset

---

## MIPS Instruction – Subset

### ❖ Arithmetic and Logical Instructions

➤ *add, sub, or, and, slt*

### ❖ Memory reference Instructions

➤ *lw, sw*

### ❖ Branch

➤ *beq, j*



# Overview of MIPS

- ❖ simple instructions, all 32 bits wide
- ❖ very structured, no unnecessary baggage
- ❖ only three instruction formats

<b>R</b>	<b>op</b>	<b>rs1</b>	<b>rs2</b>	<b>rd</b>	<b>shmt</b>	<b>funct</b>
<b>I</b>	<b>op</b>	<b>rs1</b>	<b>rd</b>	<b>16 bit address</b>		
<b>J</b>	<b>op</b>	<b>26 bit address</b>				

- ❖ rely on compiler to achieve performance



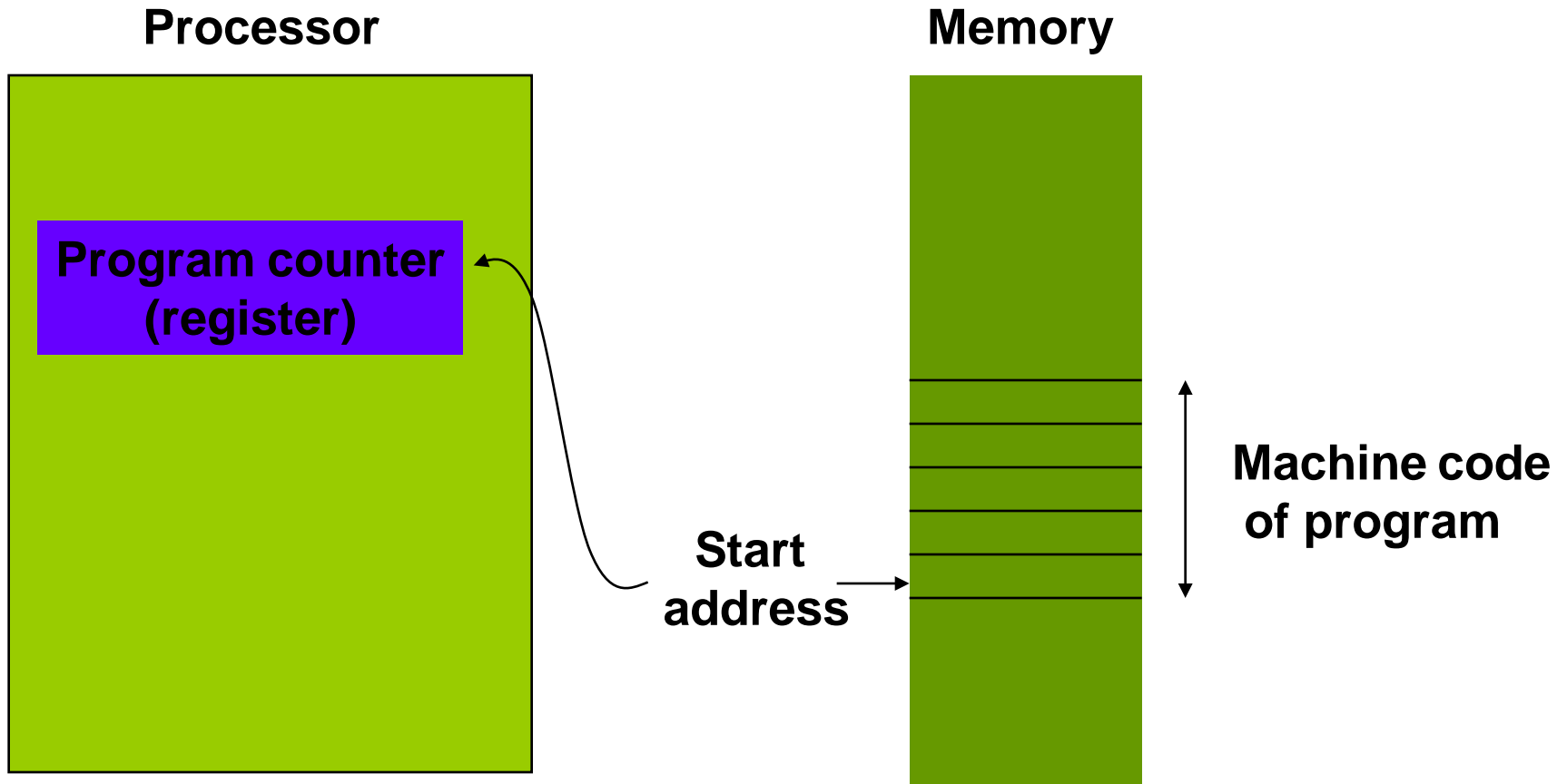
# Where Does It All Begin?

---

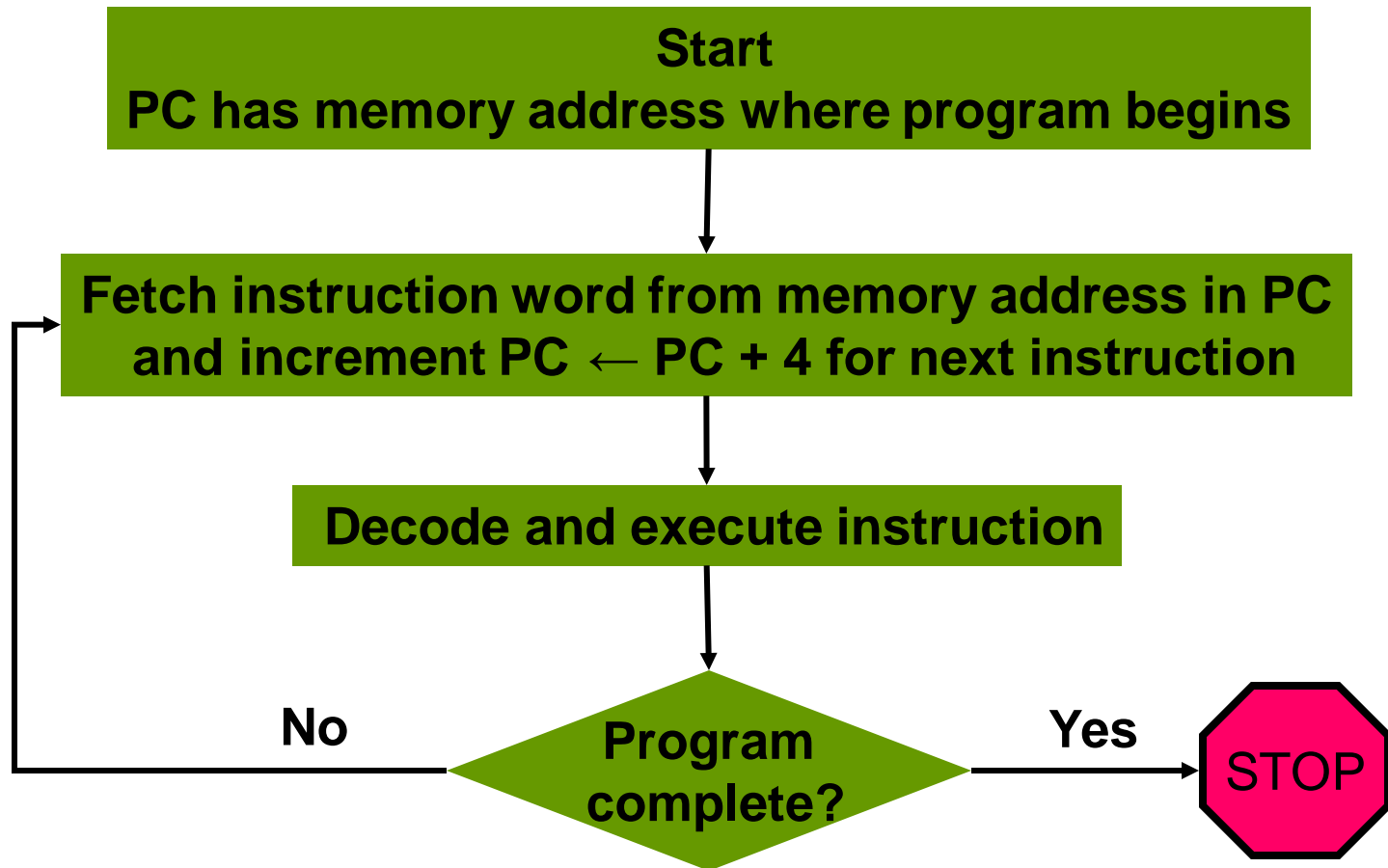
- In a register called *program counter (PC)*.
- PC contains the memory address of the next instruction to be executed.
- In the beginning, PC contains the address of the memory location where the program begins.



# Where is the Program?



# How Does It Run?



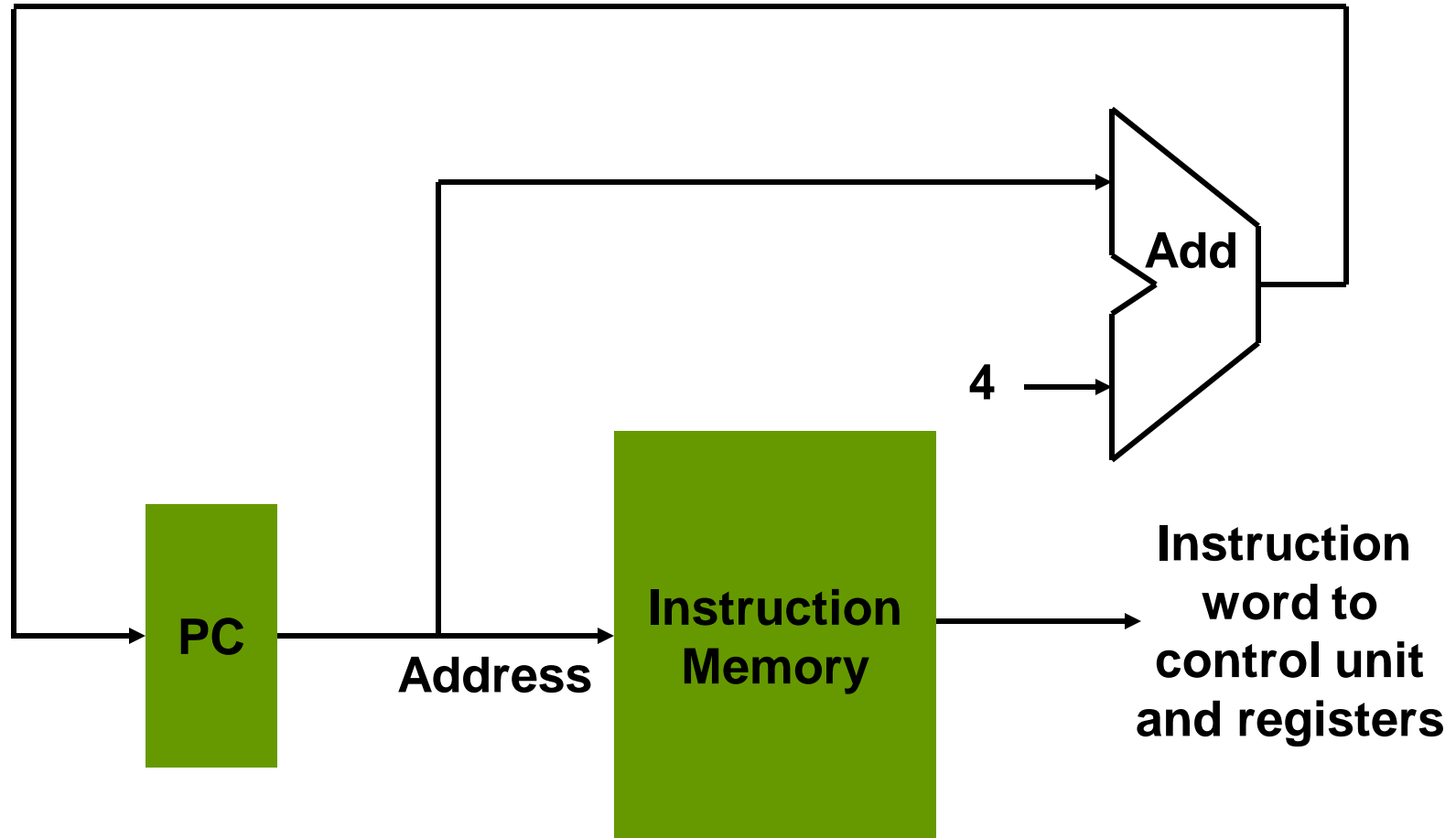
# Datapath and Control

---

- Datapath: Memory, registers, adders, ALU, and communication buses. Each step (fetch, decode, execute) requires communication (data transfer) paths between memory, registers and ALU.
- Control: Datapath for each step is set up by control signals that set up dataflow directions on communication buses and select ALU and memory functions. Control signals are generated by a control unit consisting of one or more finite-state machines.

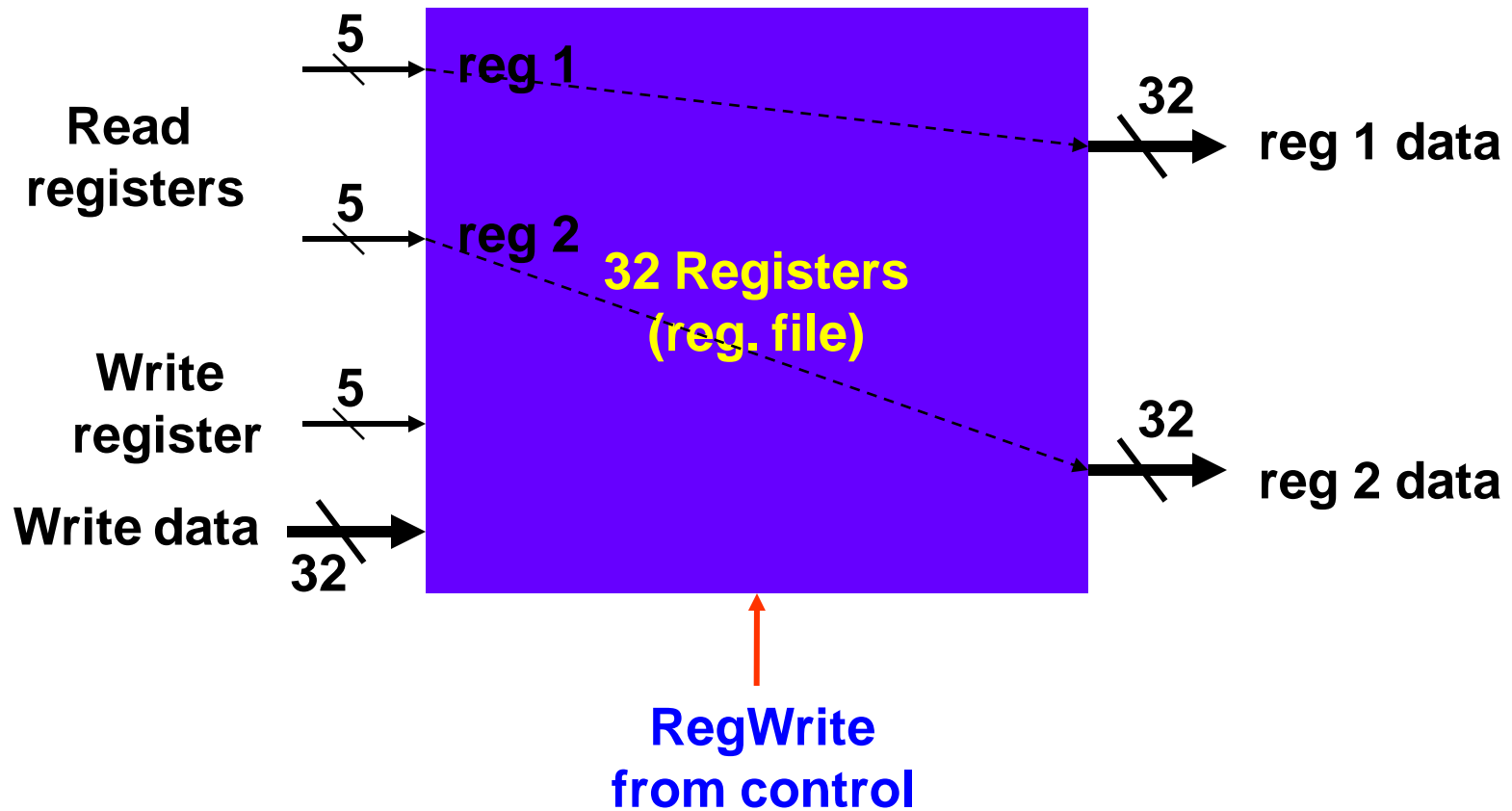


# Datapath for Instruction Fetch





# Register File: A Datapath Component

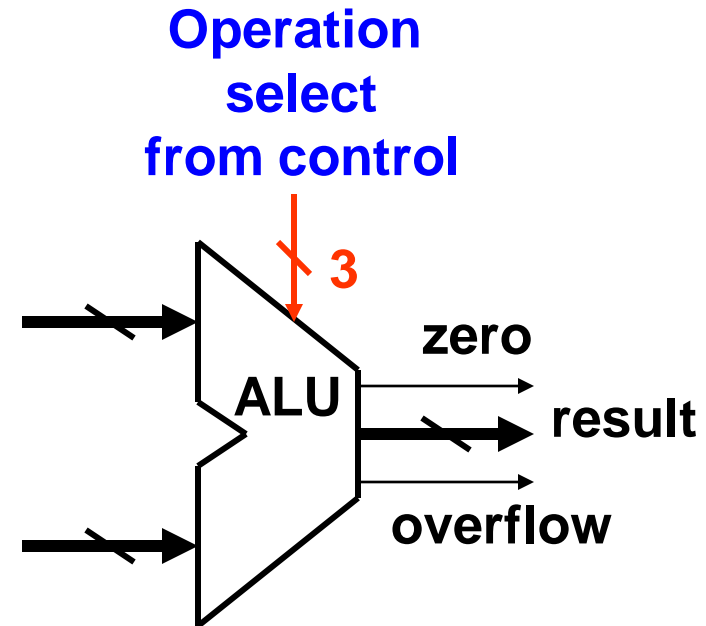


# Multi-Operation ALU

Operation  
select

ALU function

000	AND
001	OR
010	Add
110	Subtract
111	Set on less than



**zero = 1, when all bits of result are 0**



# R-Type Instructions

---

- Also known as arithmetic-logical instructions

- **add, sub, slt**

- Example: add \$t0, \$s1, \$s2

- Machine instruction word

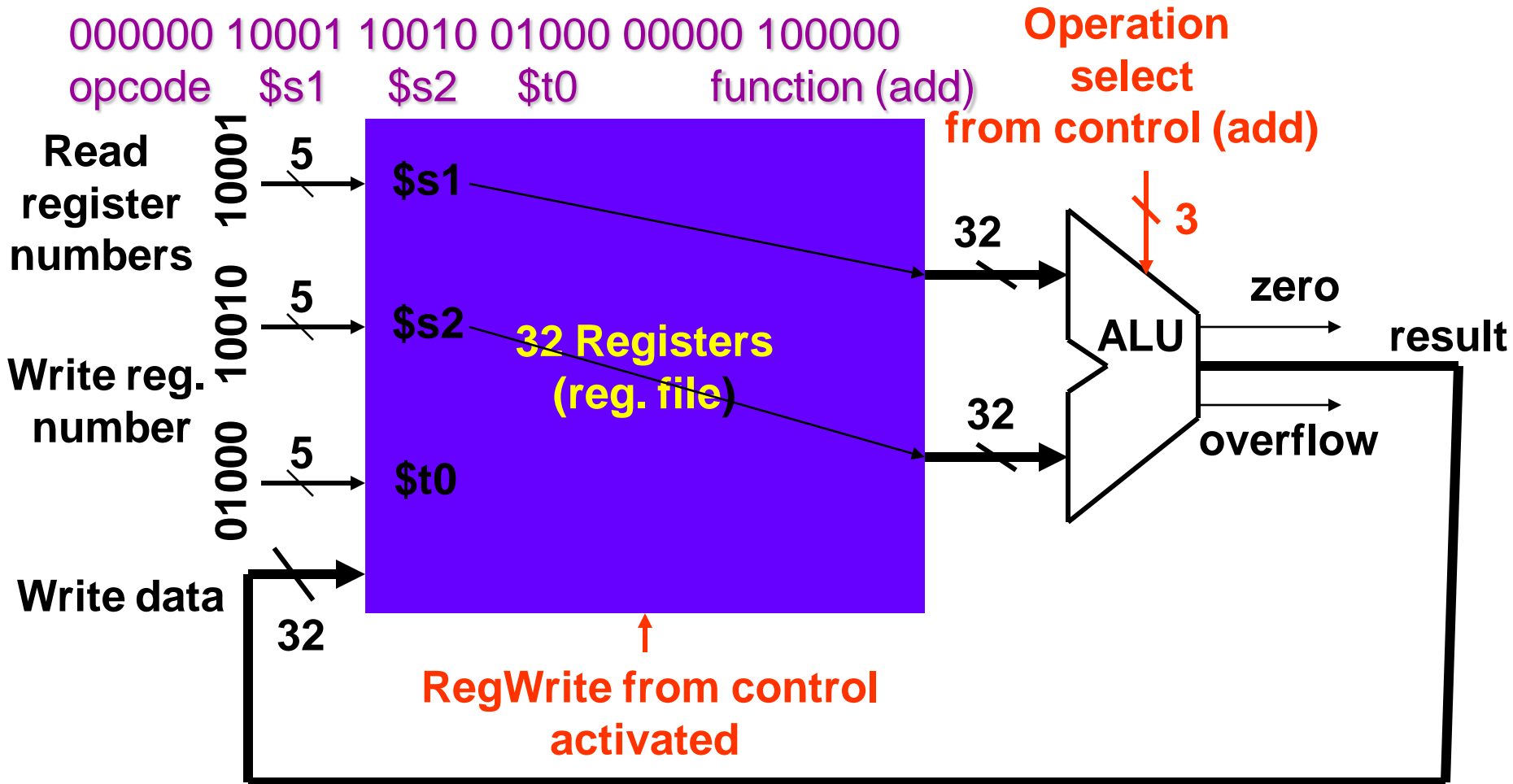
000000 10001 10010 01000 00000 100000

opcode \$s1 \$s2 \$t0 function

- Read two registers
- Write one register
- Opcode and function code go to control unit that generates RegWrite and ALU operation code.



# Datapath for R-Type Instruction



# Load and Store Instructions

---

- I-type instructions
- lw            \$t0, 1200 (\$t1)            # incr. in bytes

100011 01001 01000 0000 0100 1011 0000  
opcode    \$t1    \$t0            1200

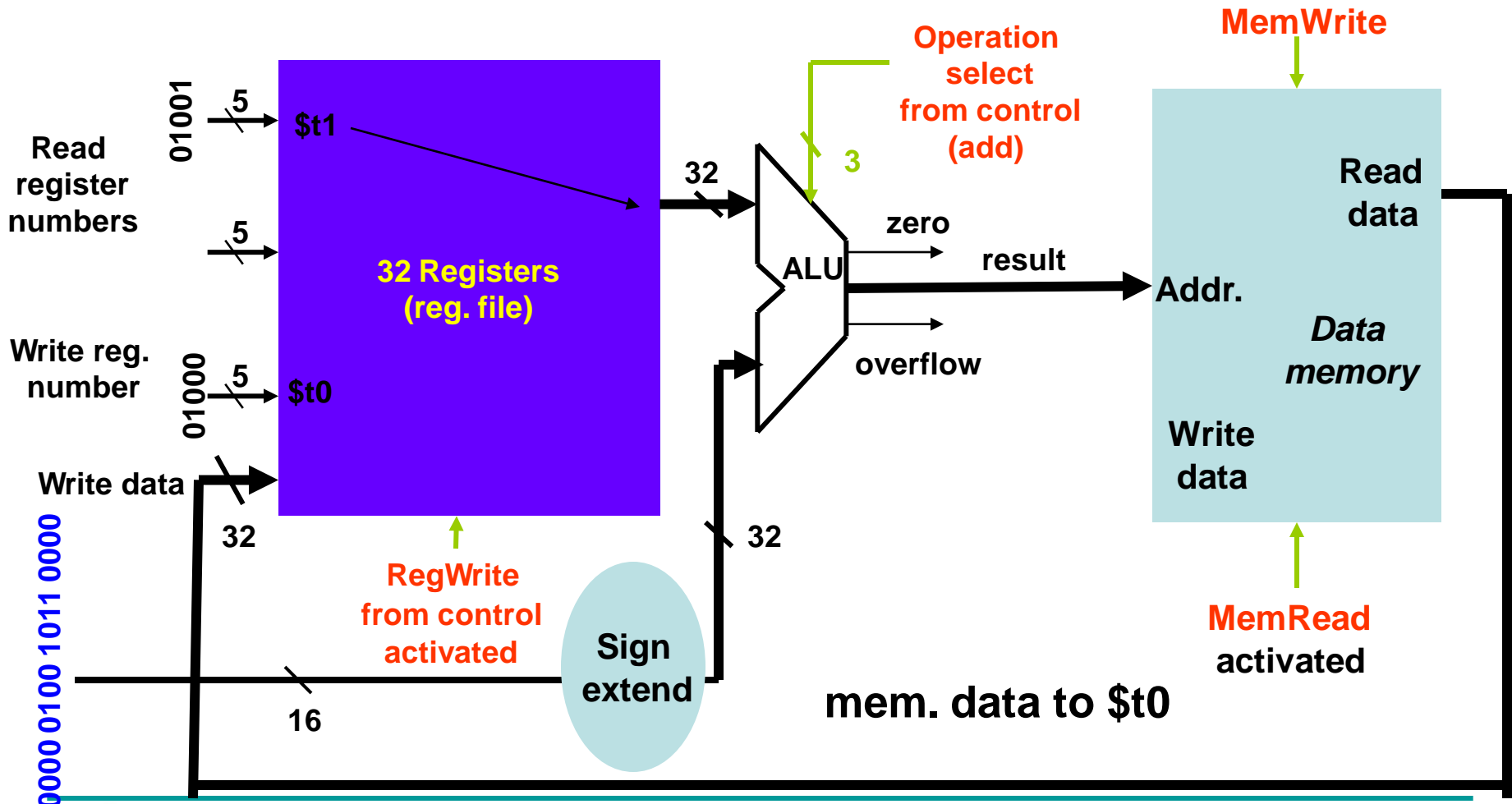
- sw \$t0, 1200 (\$t1)            # incr. in bytes

101011 01001 01000 0000 0100 1011 0000  
opcode    \$t1    \$t0            1200



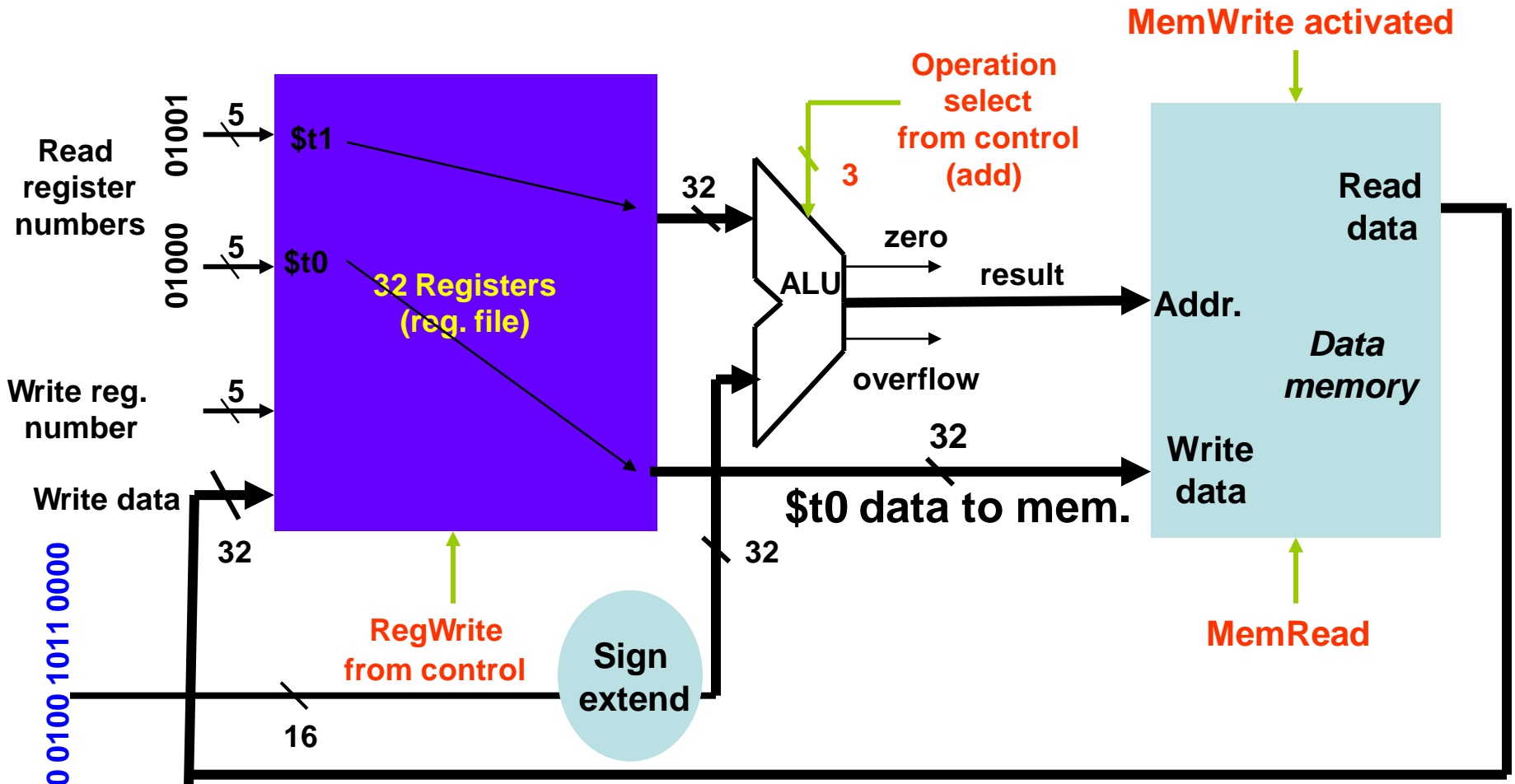
# Datapath for lw Instruction

100011 01001 01000 0000 0100 1011 0000  
 opcode \$t1 \$t0 1200



# Datapath for sw Instruction

101011 01001 01000 0000 0100 1011 0000  
 opcode \$t1 \$t0 1200

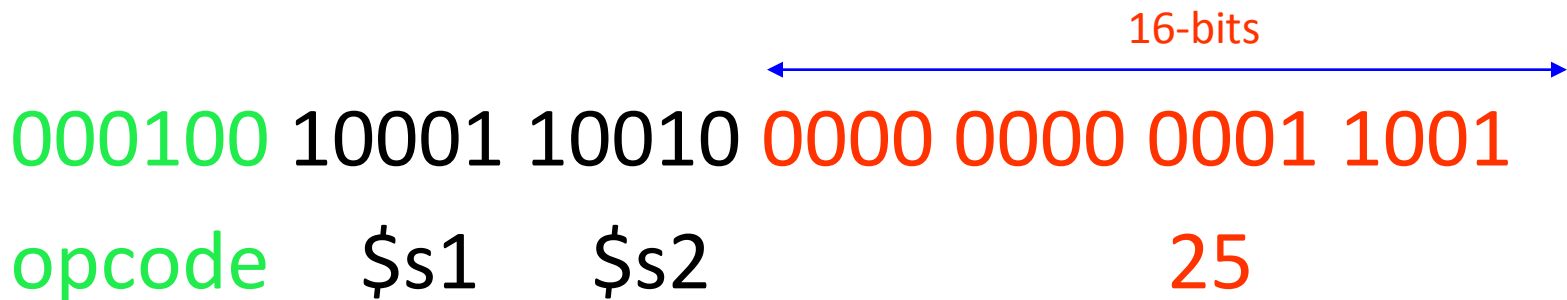


0000 0100 1011 0000



# Branch Instruction (I-Type)

- `beq $s1, $s2, 25` # if  $\$s1 = \$s2$ , advance PC through 25 instructions

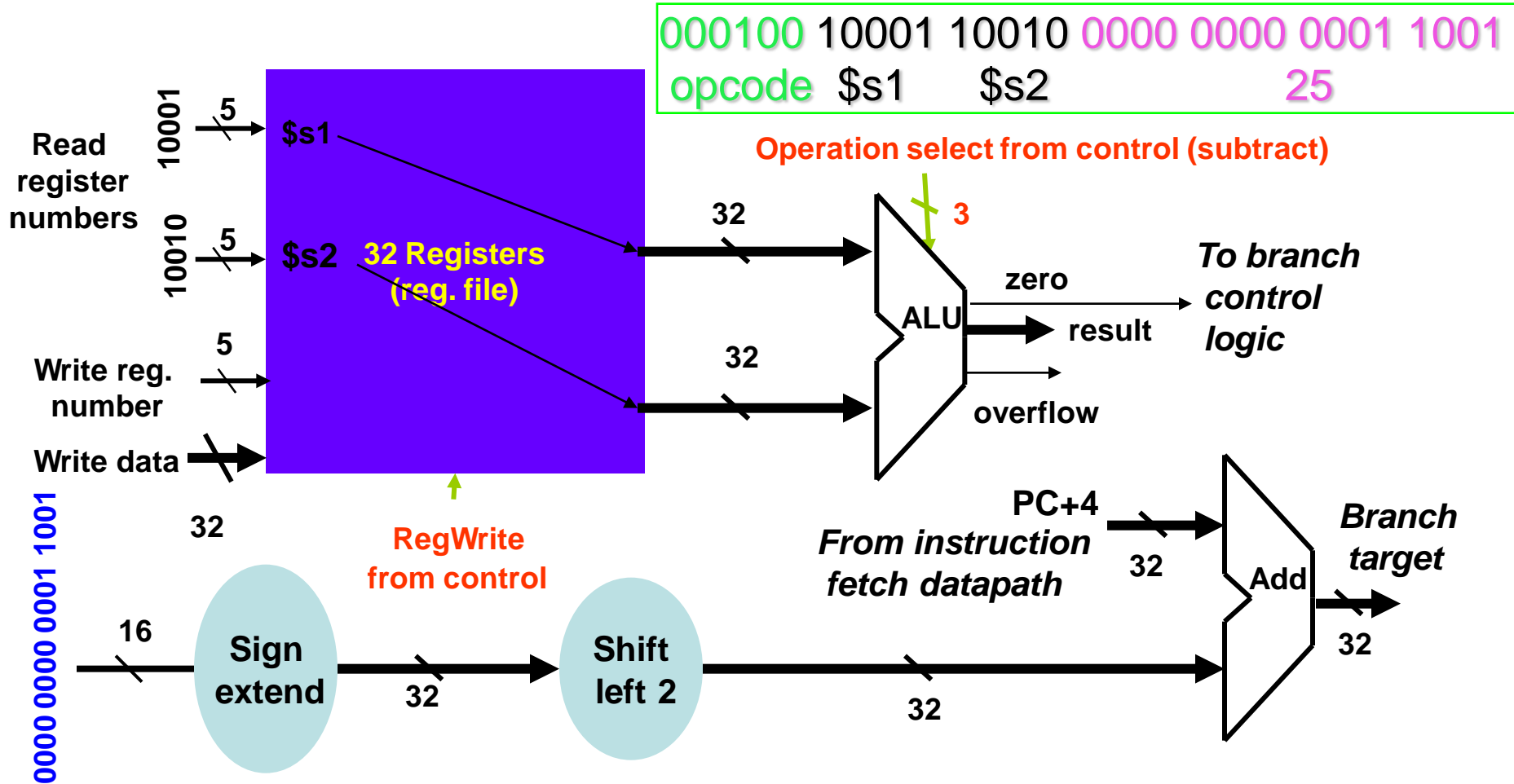


**Note:** Can branch within  $\pm 2^{15}$  words from the current instruction address in PC.



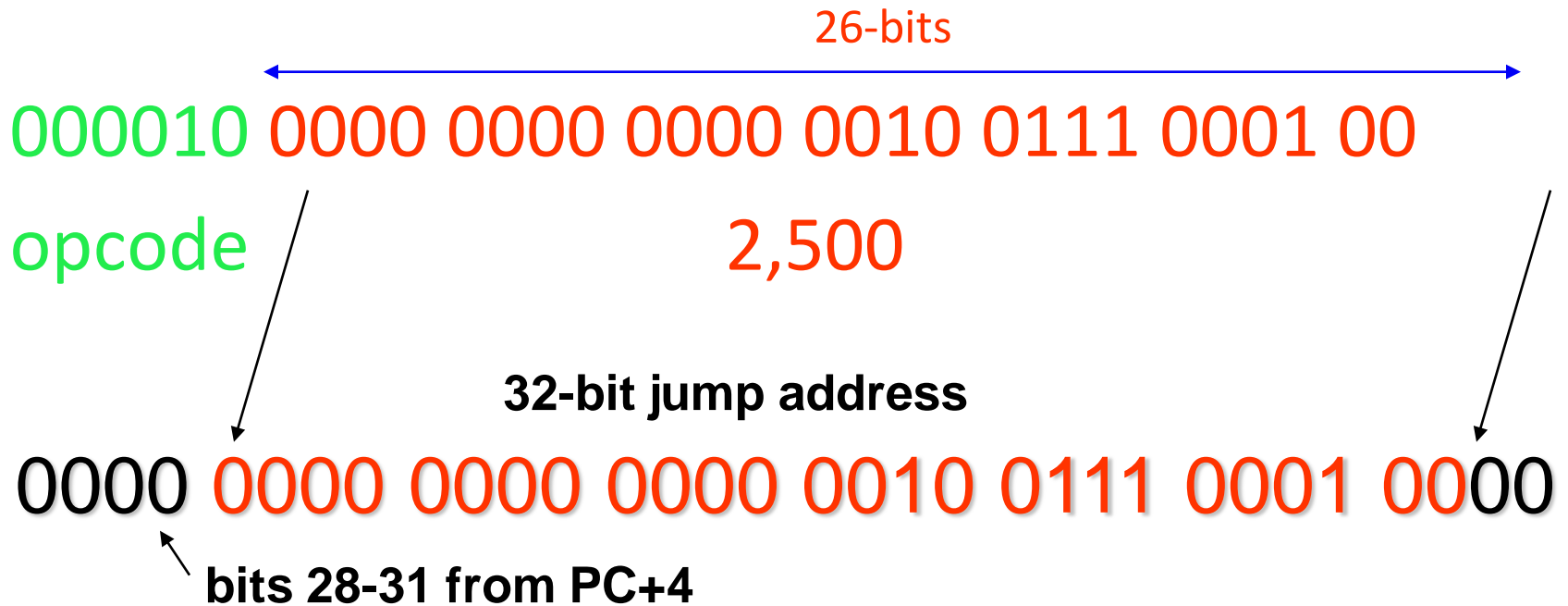


# Datapath for beq Instruction

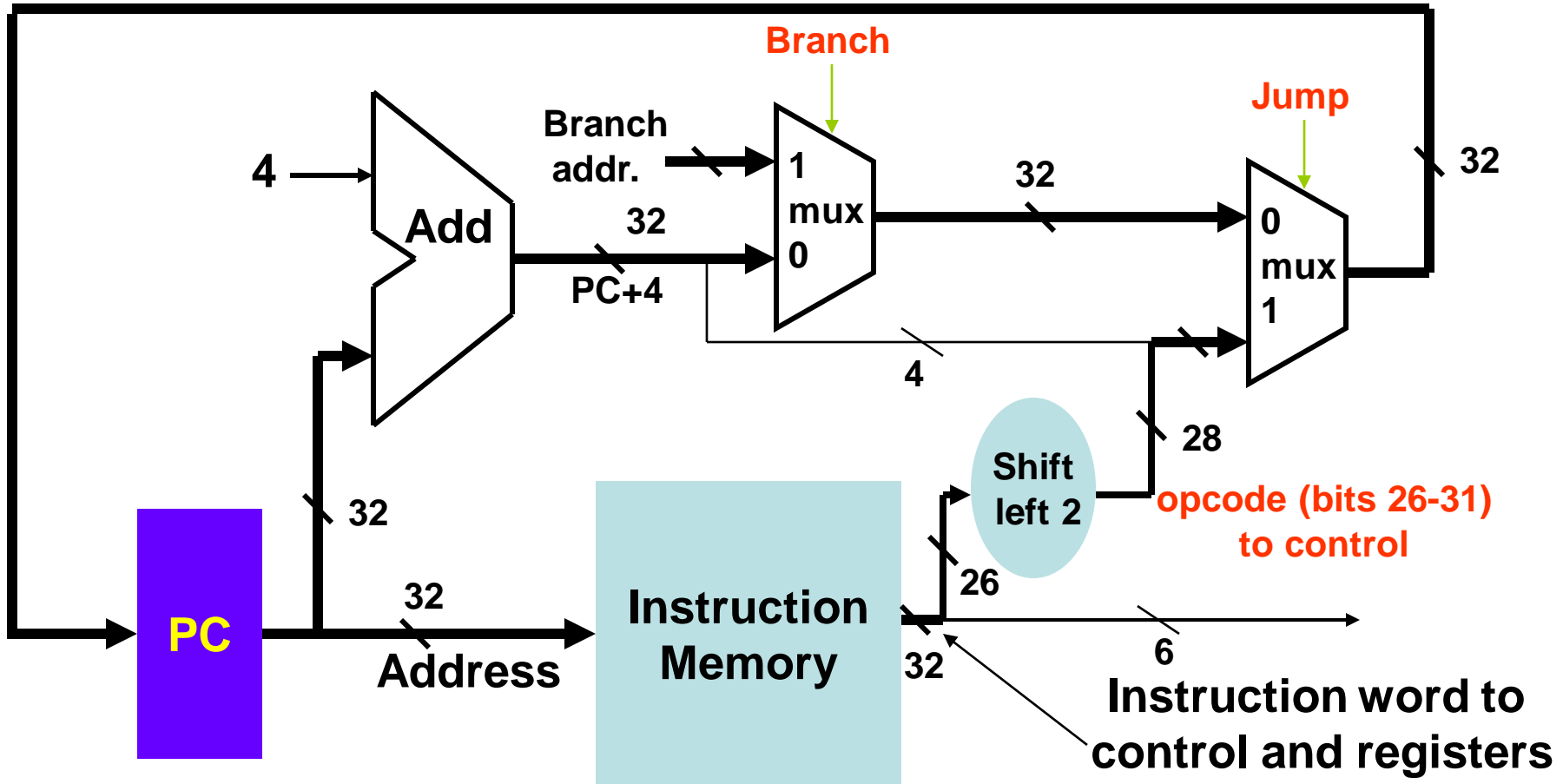


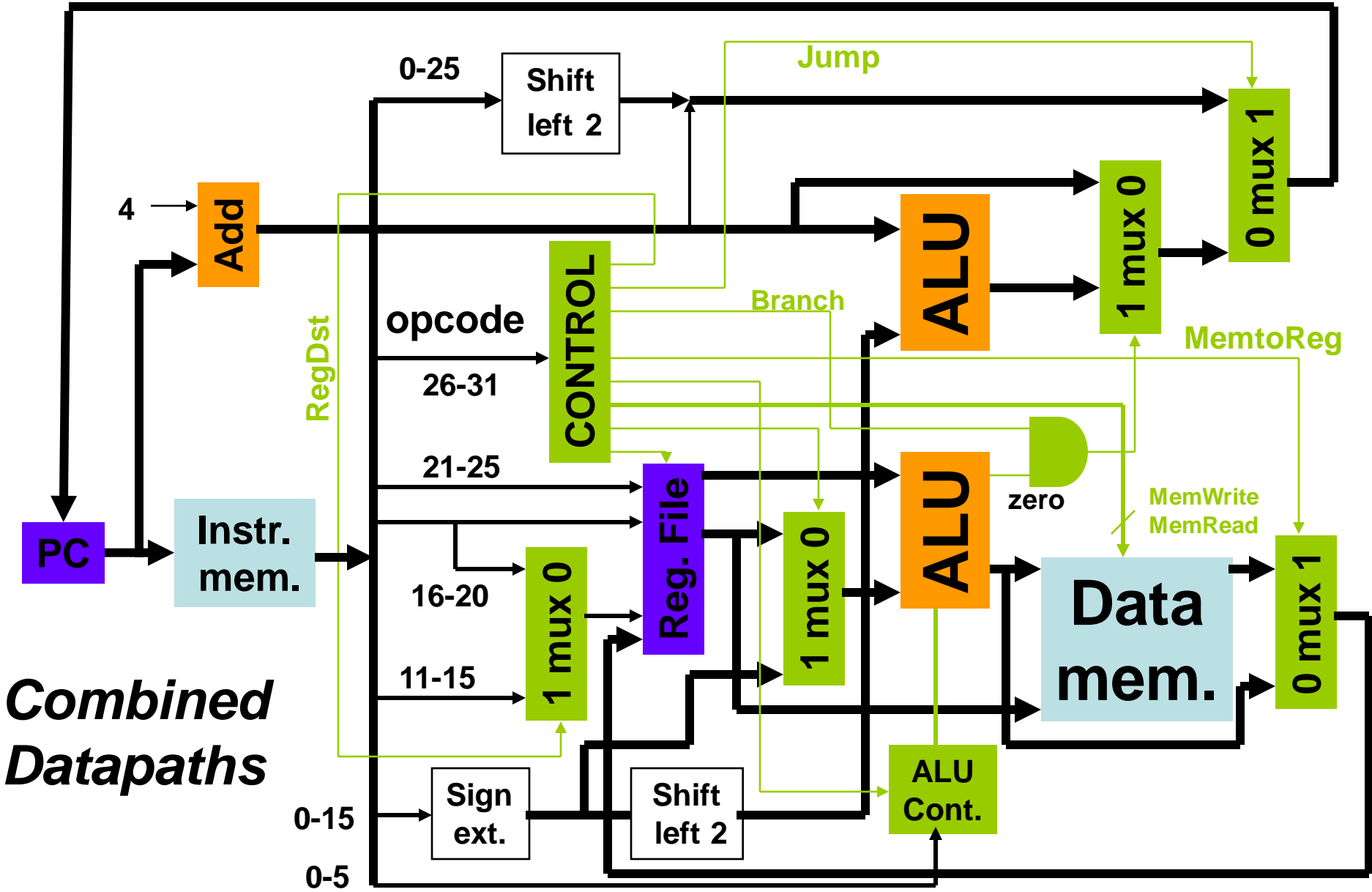
# J-Type Instruction

- **j 2500** # jump to instruction 2,500



# Datapath for Jump Instruction

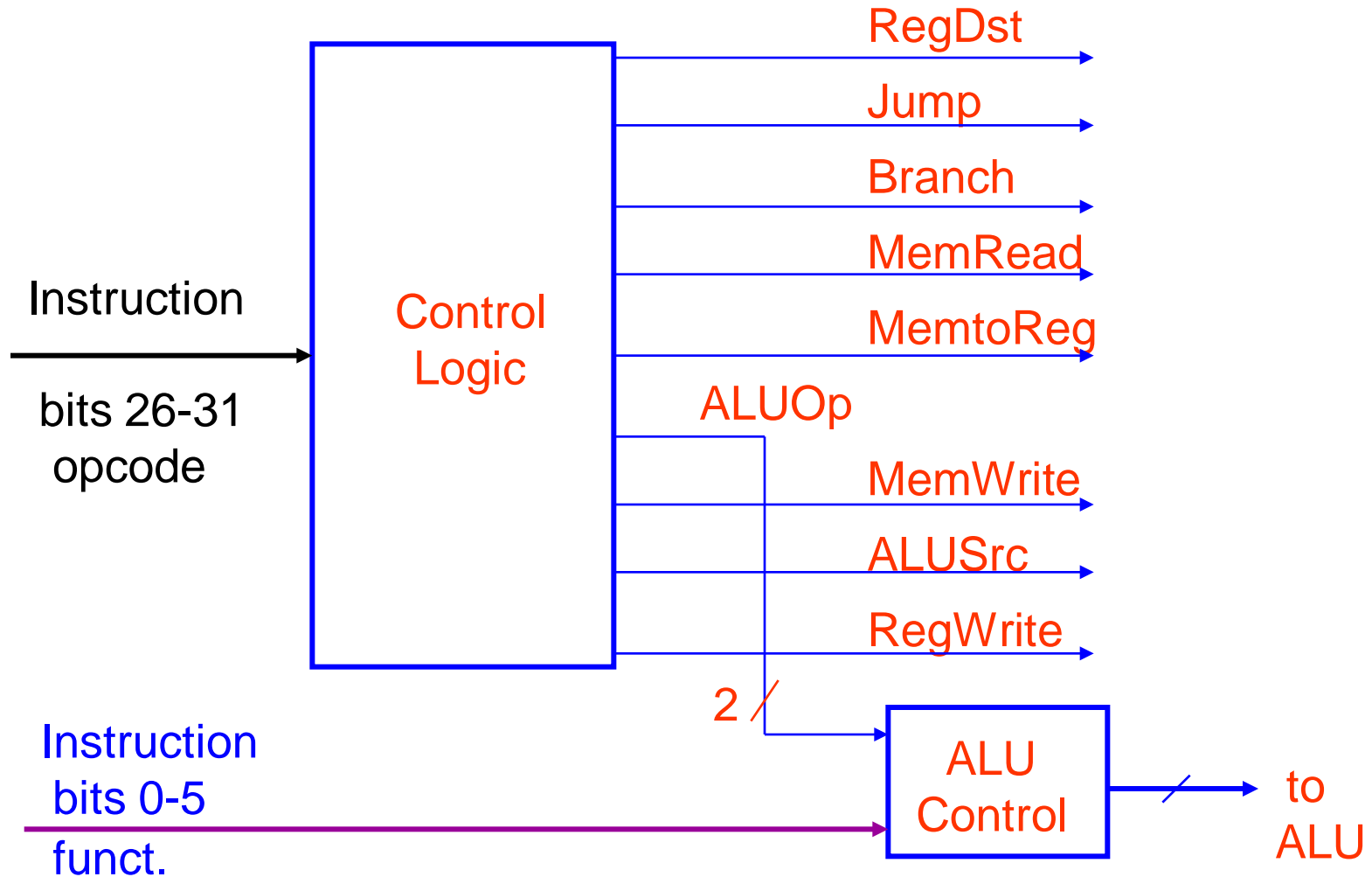




**Combined Datapaths**



# Control Logic



# Control Logic: Truth Table

Instr type	Inputs: instr. opcode bits						Outputs: control signals									
	31	30	29	28	27	26	RegDst	Jump	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALOp1	ALOp2
R	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0
lw	1	0	0	0	1	1	0	0	1	1	1	1	0	0	0	0
sw	1	0	1	0	1	1	X	0	1	X	0	0	1	0	0	0
beq	0	0	0	1	0	0	X	0	0	X	0	0	0	1	0	1
j	0	0	0	0	1	0	X	1	X	X	X	X	X	X	X	X



# How Long Does It Take?

---

- Assume control logic is fast and does not affect the critical timing. Major time delay components are ALU, memory read/write, and register read/write.
- Arithmetic-type (R-type)
  - Fetch (memory read) 2ns
  - Register read 1ns
  - ALU operation 2ns
  - Register write 1ns
  - **Total 6ns**



# Time for lw and sw (I-Types)

- ALU (R-type) 6ns
- Load word (I-type)
  - Fetch (memory read) 2ns
  - Register read 1ns
  - ALU operation 2ns
  - Get data (mem. Read) 2ns
  - Register write 1ns
  - Total 8ns
- Store word (no register write) 7ns





# Time for beq (I-Type)

---

- ALU (R-type) 6ns
- Load word (I-type) 8ns
- Store word (I-type) 7ns
- Branch on equal (I-type)
  - Fetch (memory read) 2ns
  - Register read 1ns
  - ALU operation 2ns
  - **Total 5ns**



# Time for Jump (J-Type)

---

- ALU (R-type) 6ns
- Load word (I-type) 8ns
- Store word (I-type) 7ns
- Branch on equal (I-type) 5ns
- Jump (J-type)
  - Fetch (memory read) 2ns
  - **Total** **2ns**



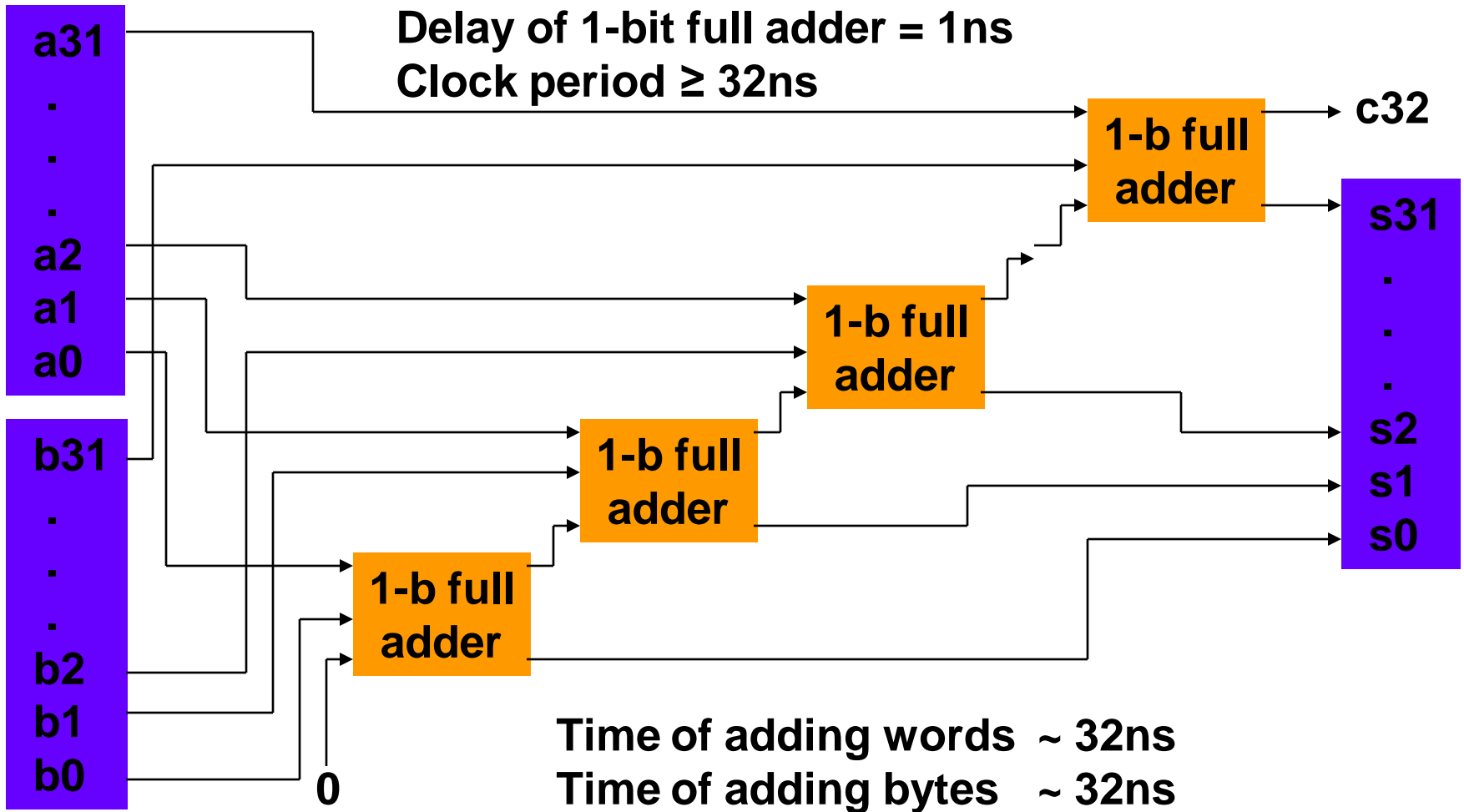
# How Fast Can the Clock Be?

---

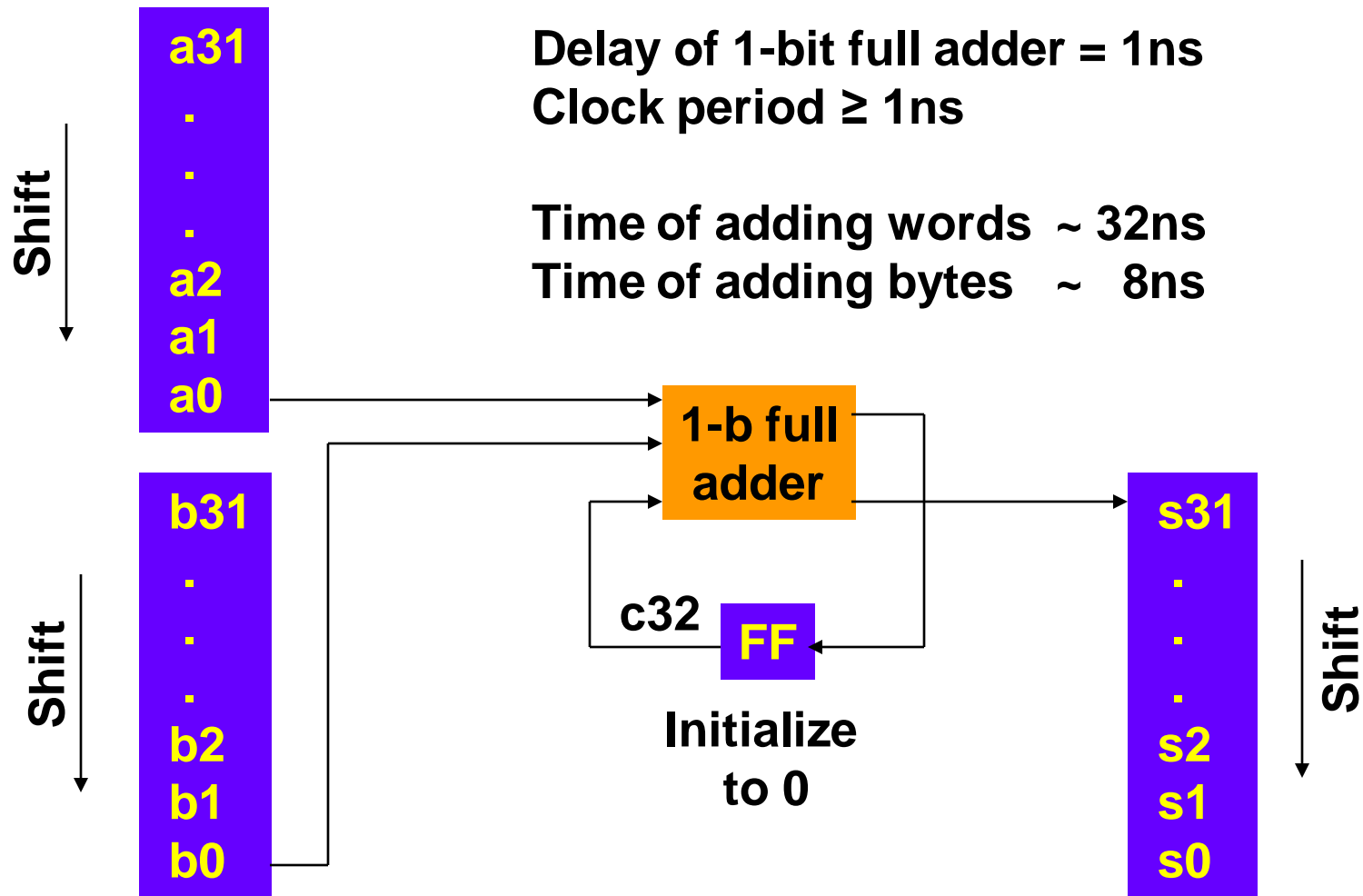
- If every instruction is executed in one clock cycle, then:
  - Clock period must be at least  $8\text{ns}$  to perform the longest instruction, i.e.,  $1/w$ .
  - This is a single cycle machine.
  - It is slower because many instructions take less than  $8\text{ns}$  but are still allowed that much time.
- Method of speeding up: Use multicycle datapath.



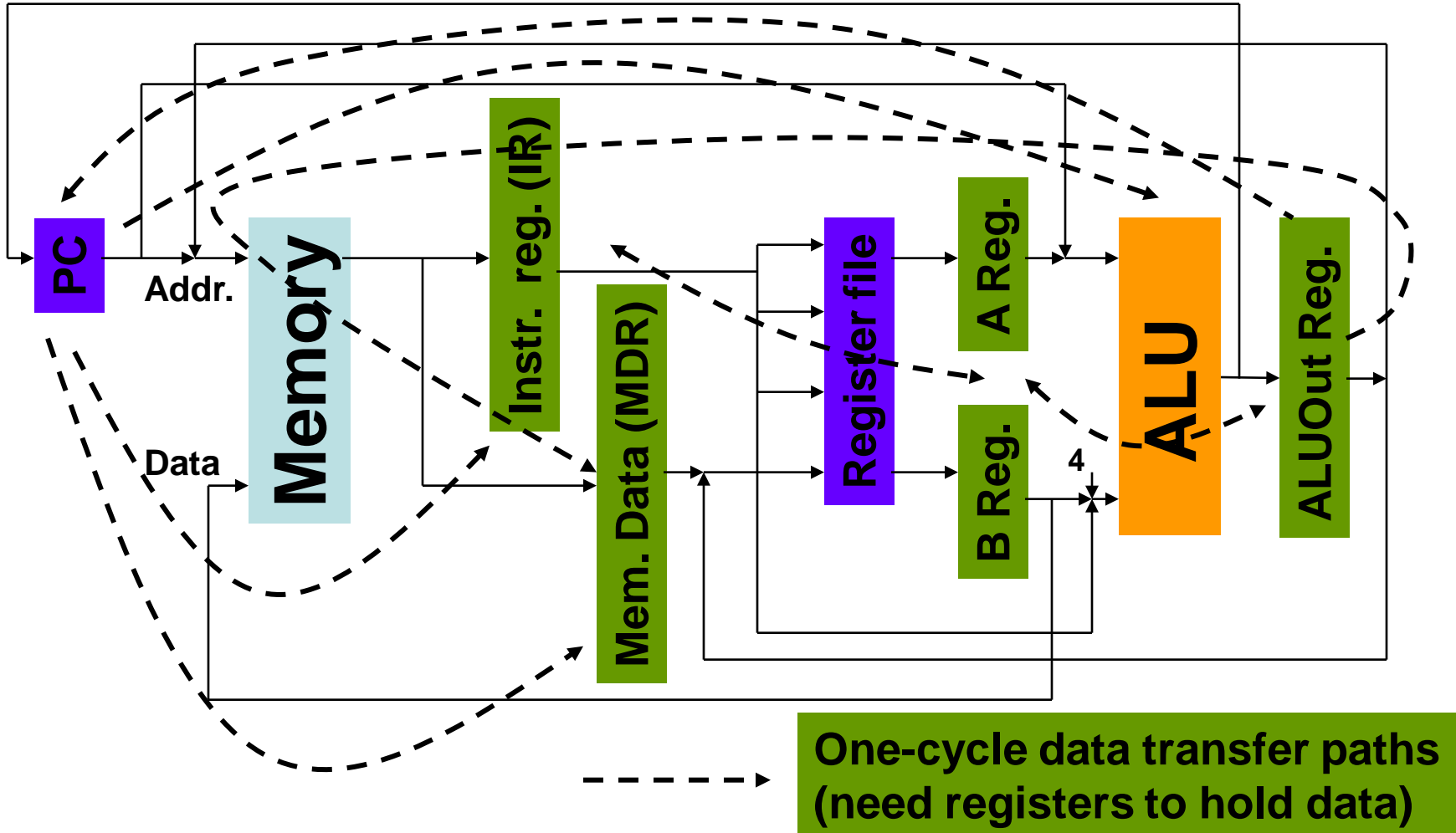
# A Single Cycle Example



# A Multicycle Implementation



# Multicycle Datapath



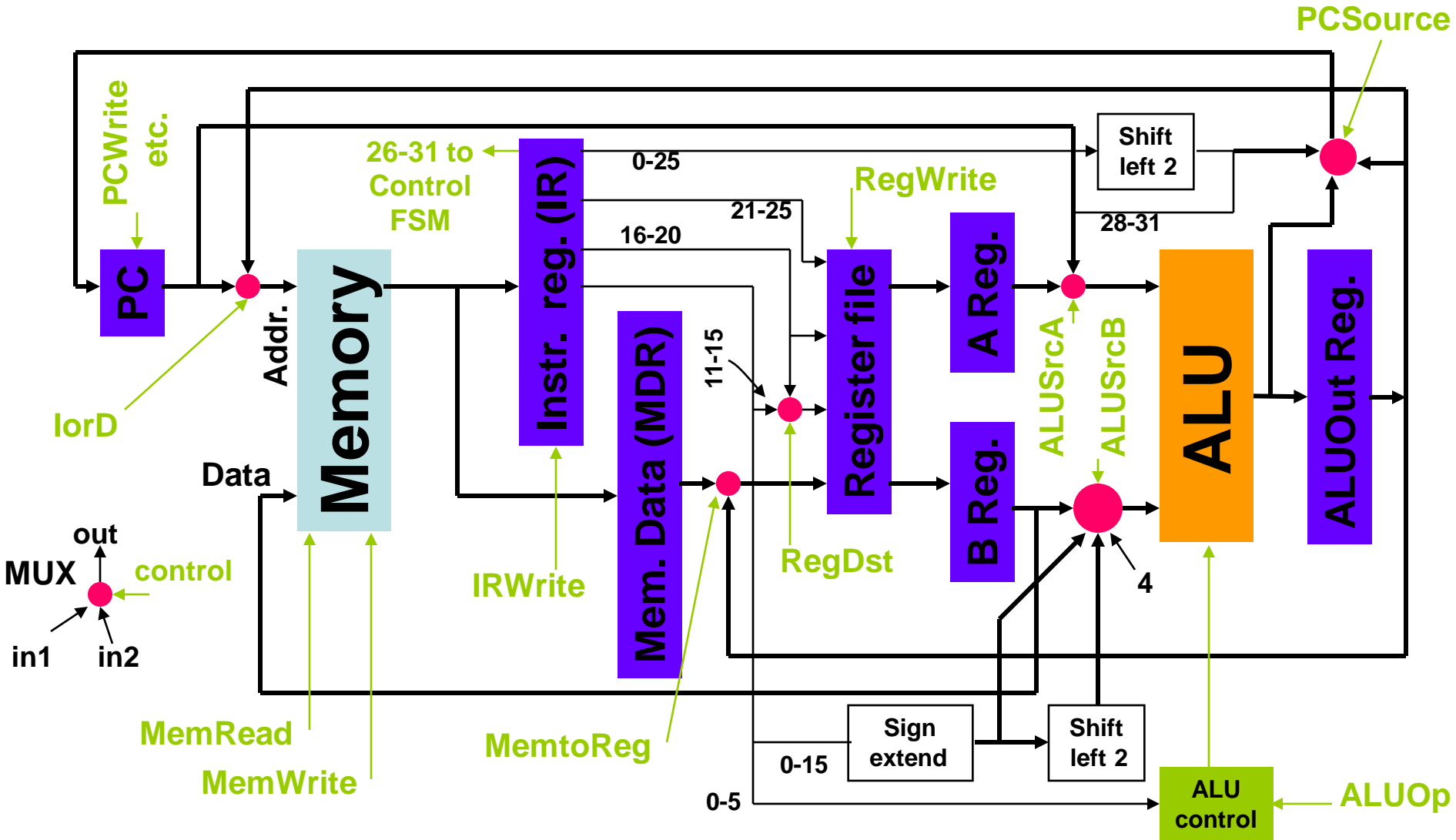
# Multicycle Datapath Requirements

---

- Only one ALU, since it can be reused.
- Single memory for instructions and data.
- Five registers added:
  - Instruction register (IR)
  - Memory data register (MDR)
  - Three ALU registers, A and B for inputs and ALUOut for output



# 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 $PC \leftarrow ALUOut$	$PC \leftarrow PC[28-31]$    $(IR[0-25] \ll 2)$
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		$\text{Reg}(IR[16-20]) \leftarrow$ $MDR$		



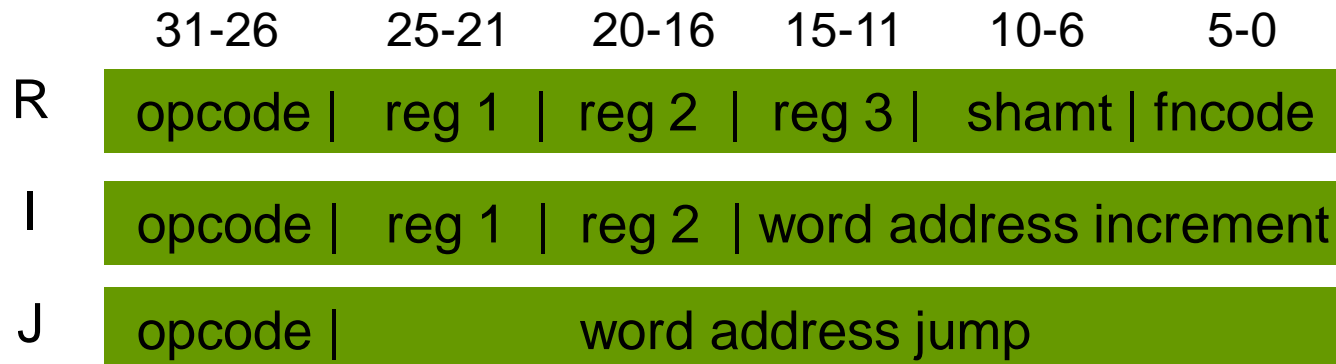
# Cycle 1 of 5: Instruction Fetch (IF)

---

- Read instruction into IR,  $M[PC] \rightarrow IR$ 
  - Control signals used:
    - » `lorD` = 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
» <b>Instruction complete, go to IF</b>			

- 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
» <b>Instruction complete, go to IF</b>			



# 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



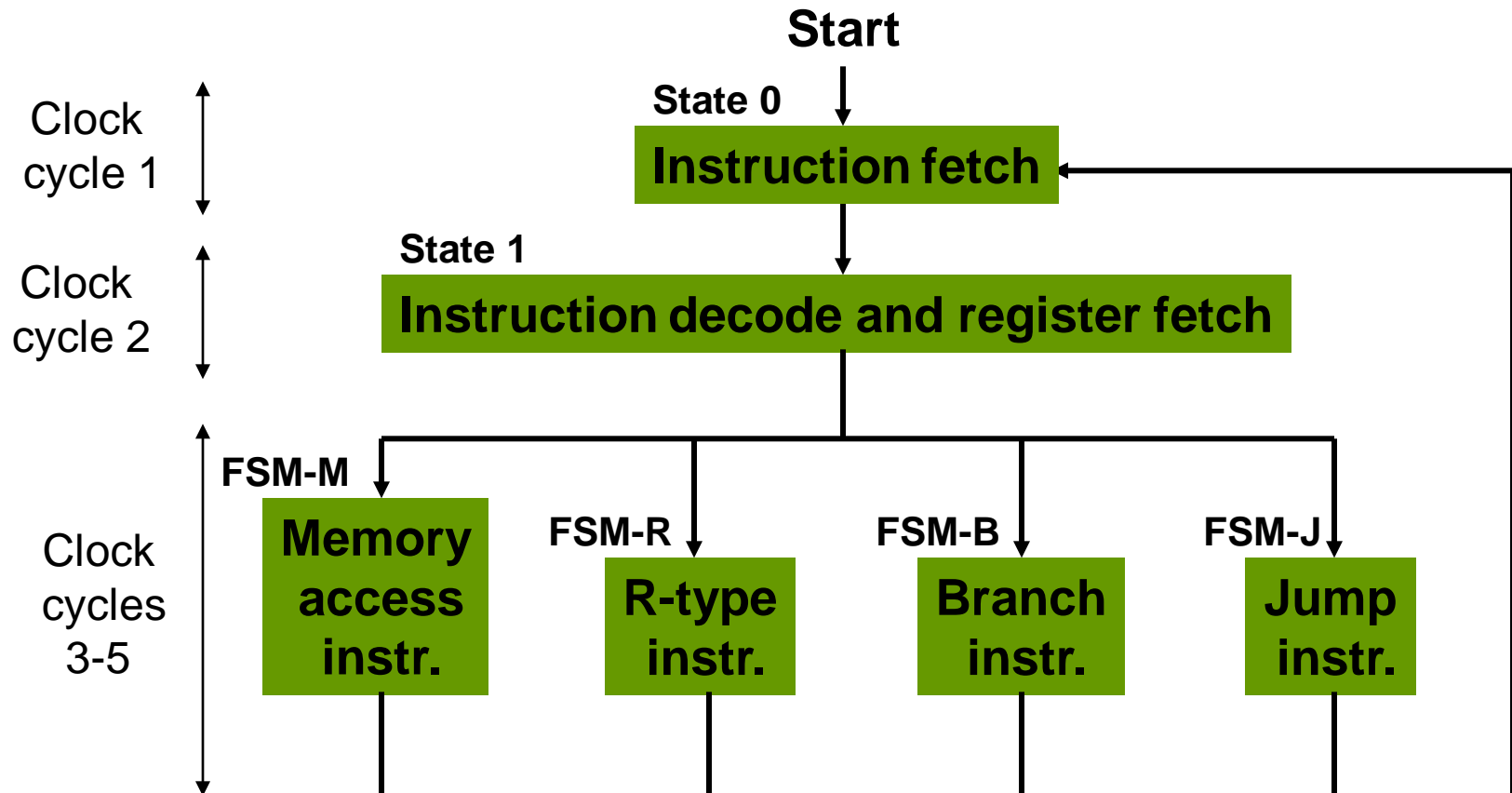


# 2-bit Control Signals

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

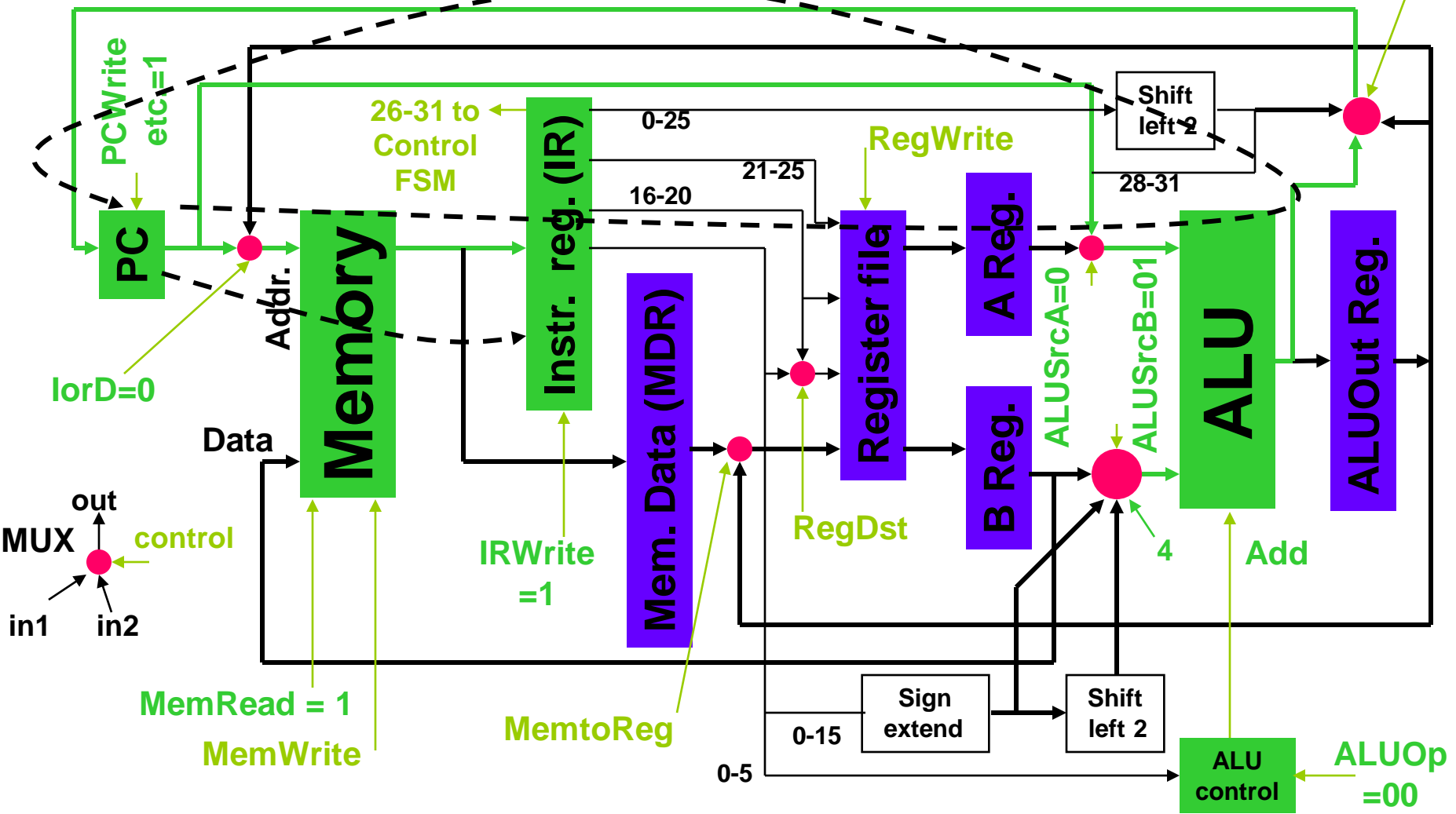


# Control: Finite State Machine

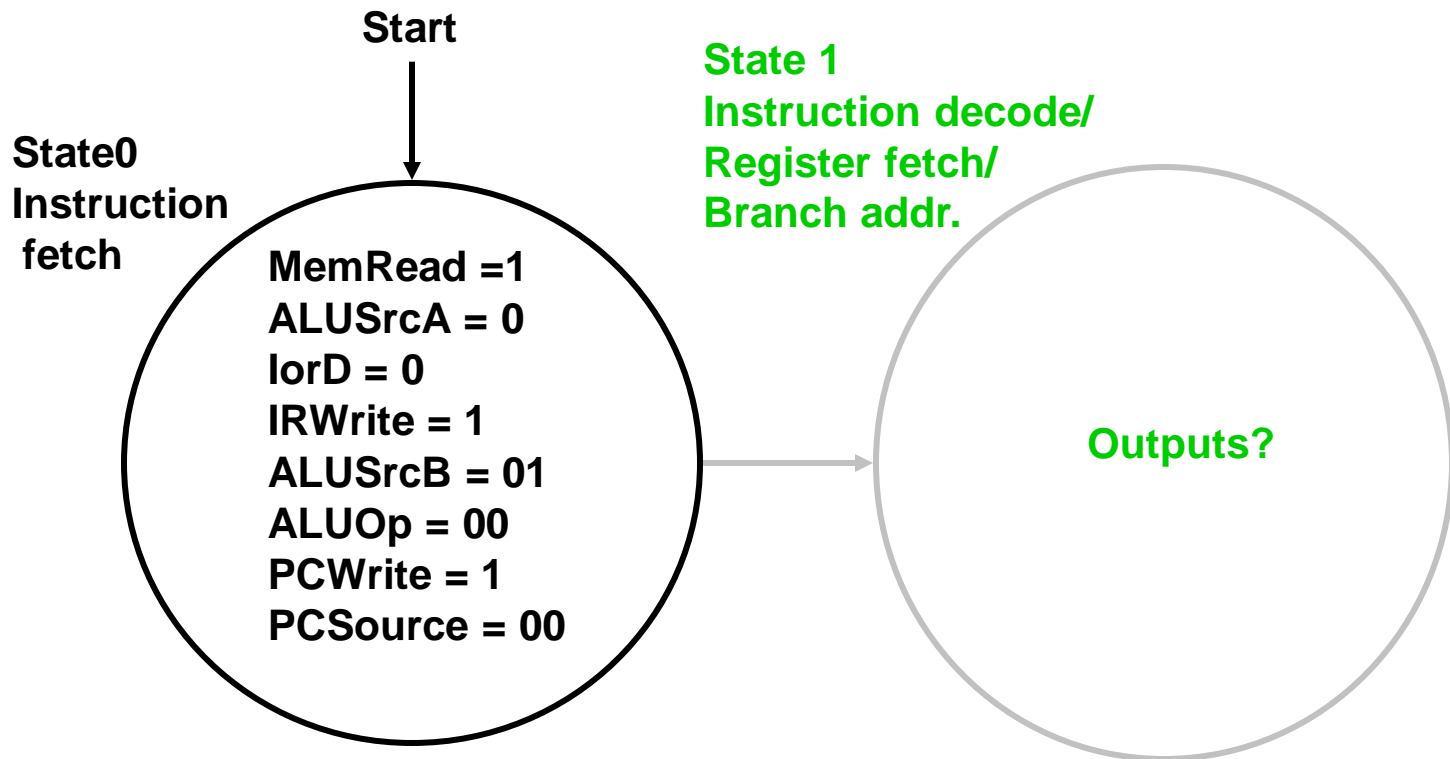


# State 0: Instruction Fetch (CC1)

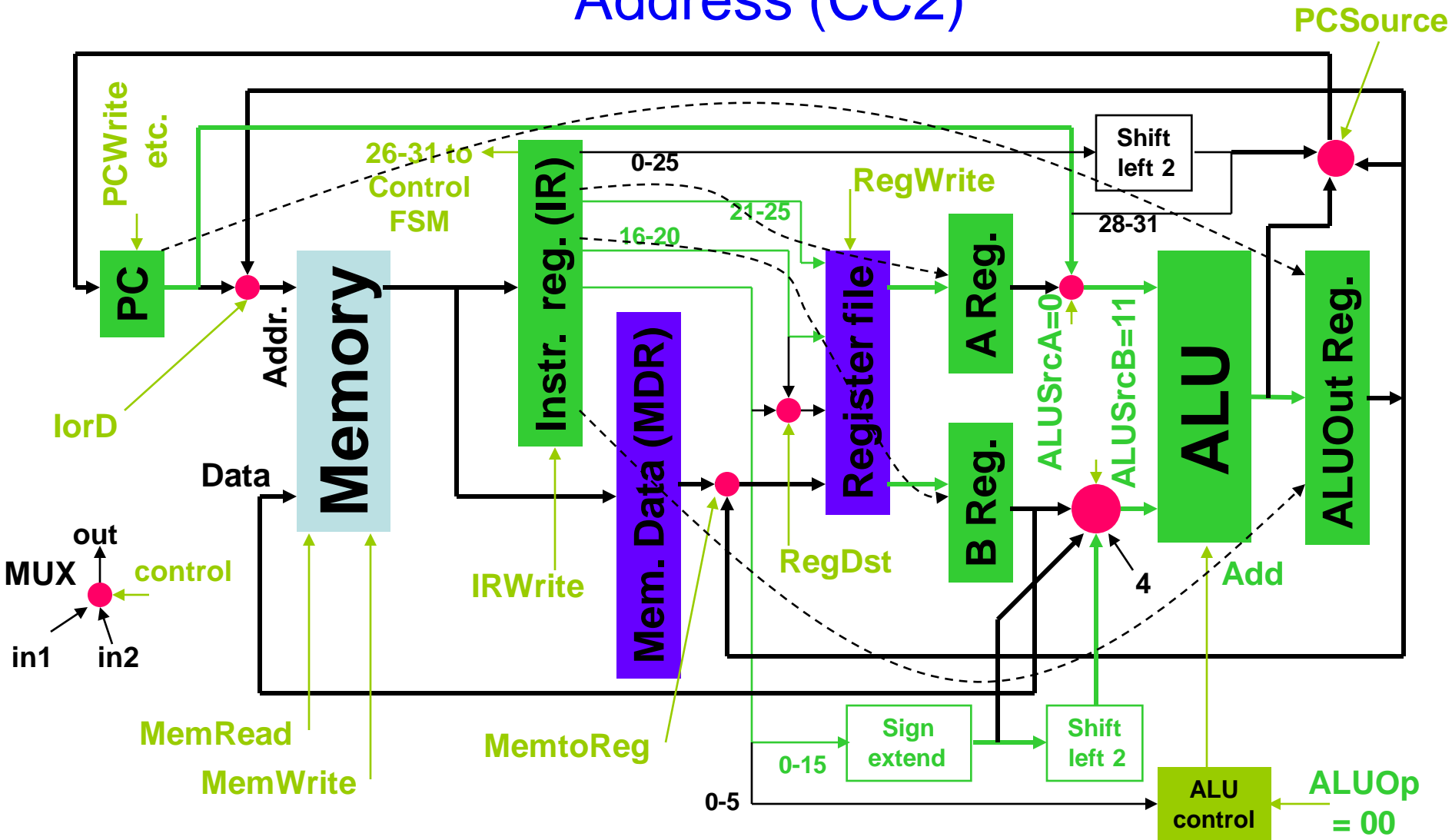
PCSource=00



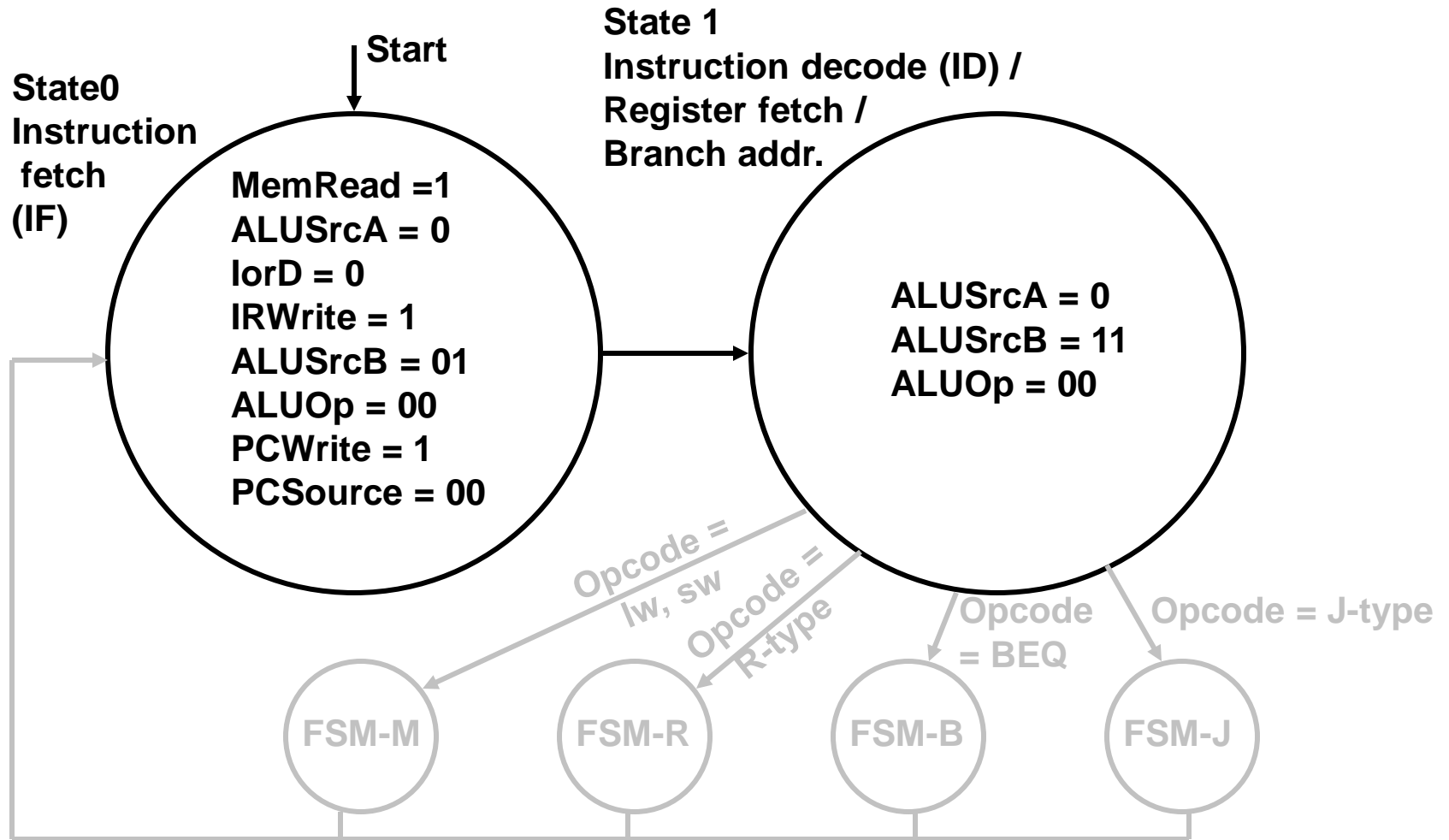
# State 0 Control FSM Outputs



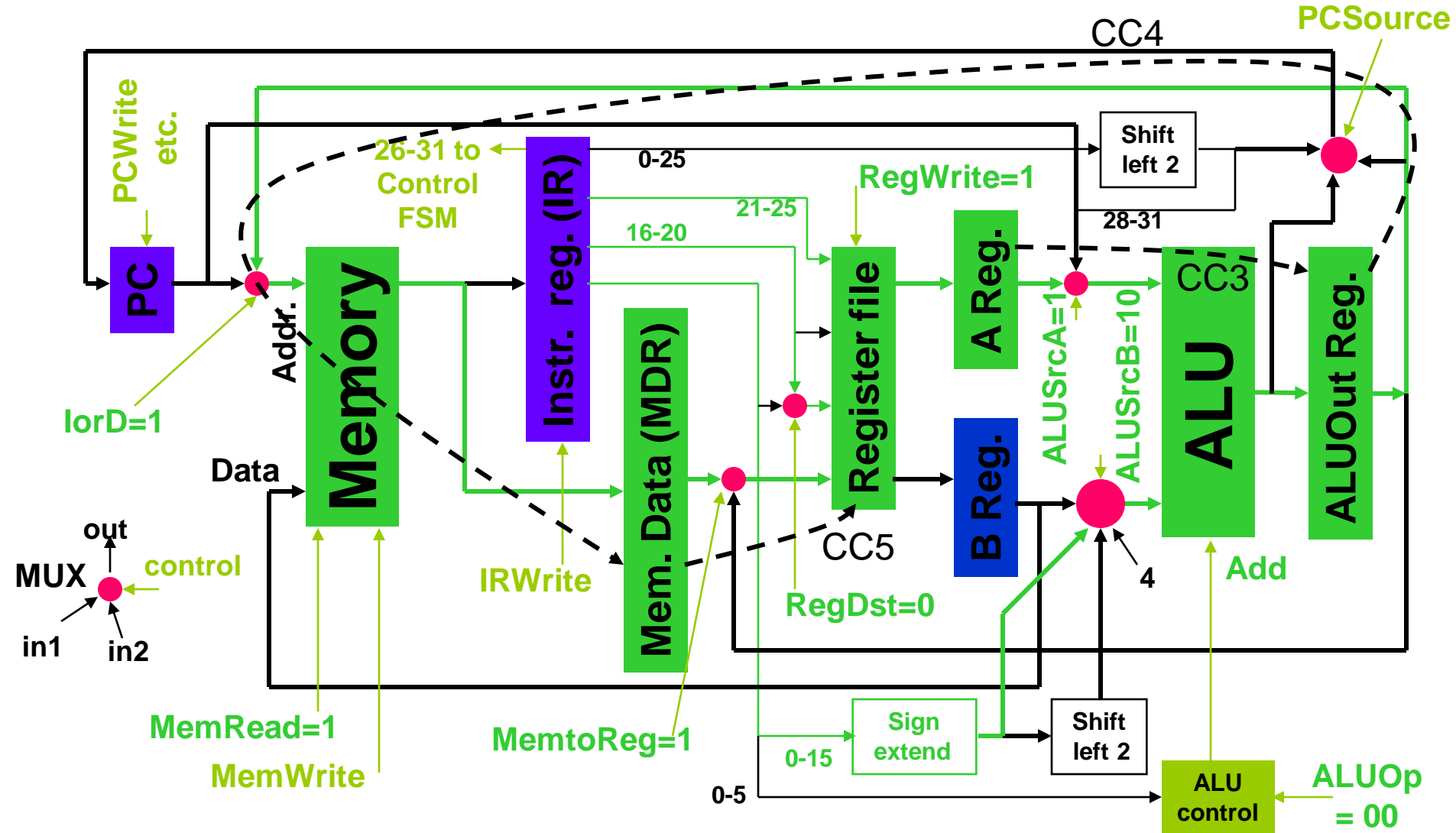
# State 1: Instr. Decode/Reg. Fetch/ Branch Address (CC2)



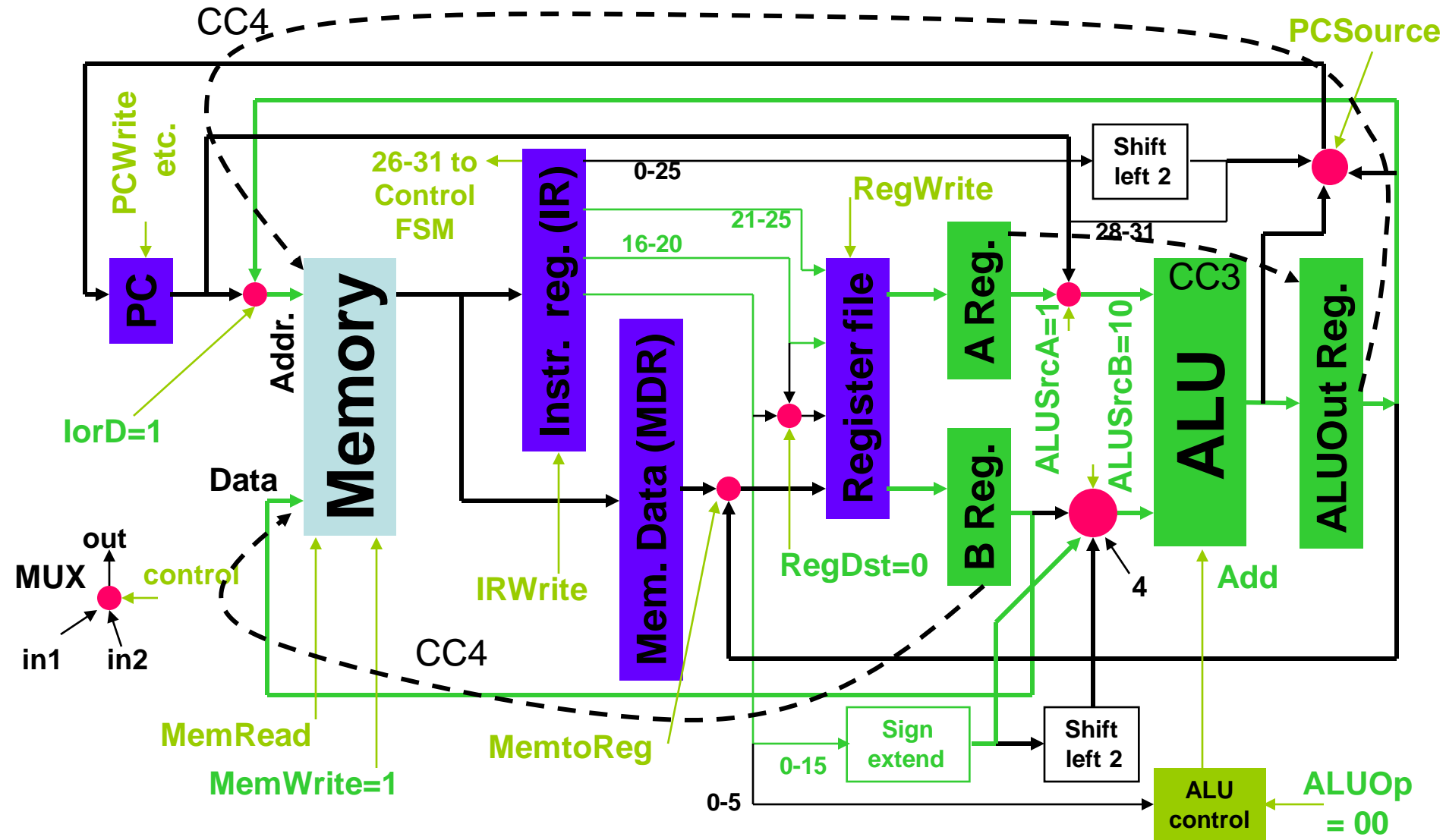
# State 1 Control FSM Outputs



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

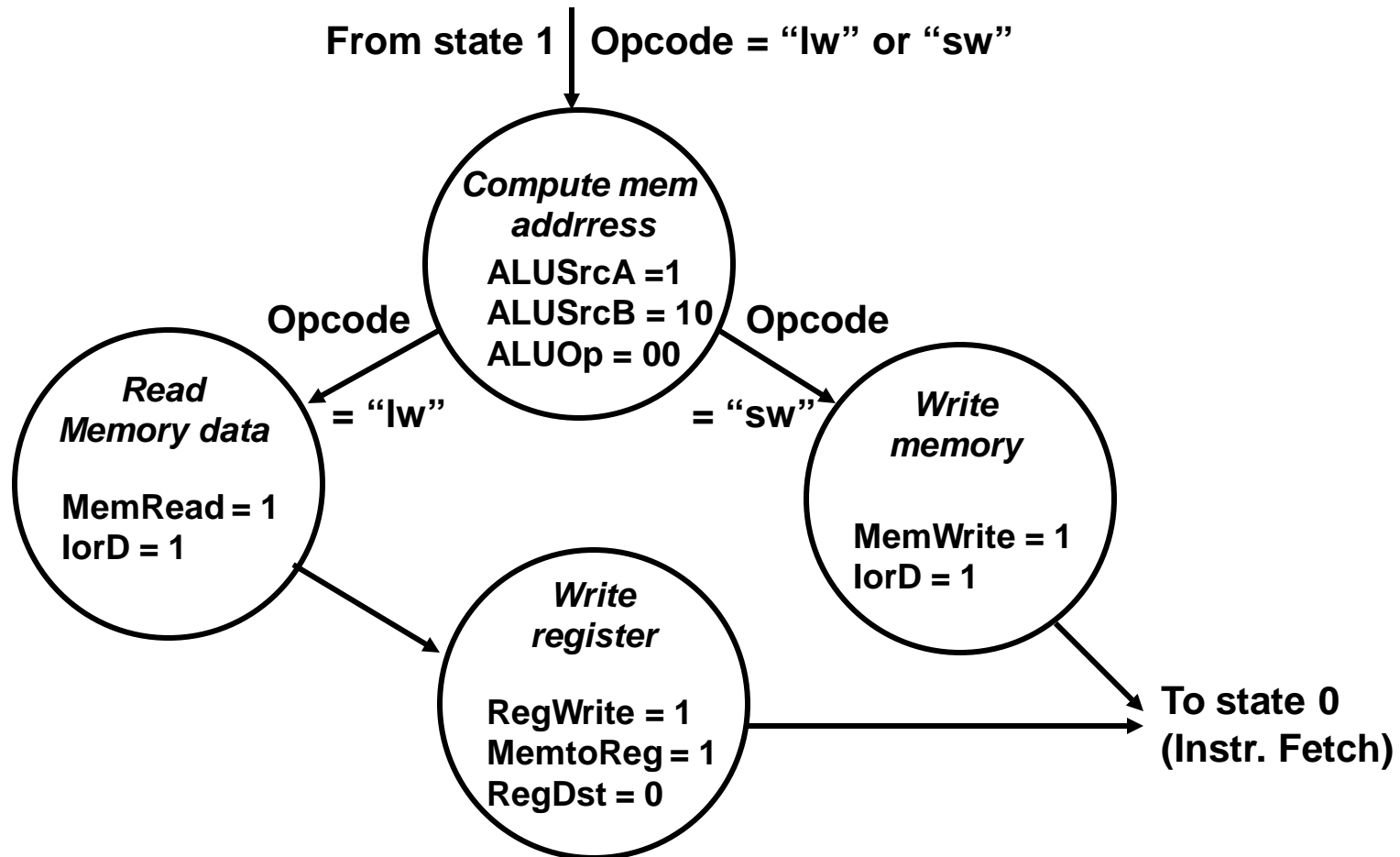


# State 1 (Opcode= sw) → FSM-M (CC3-4)

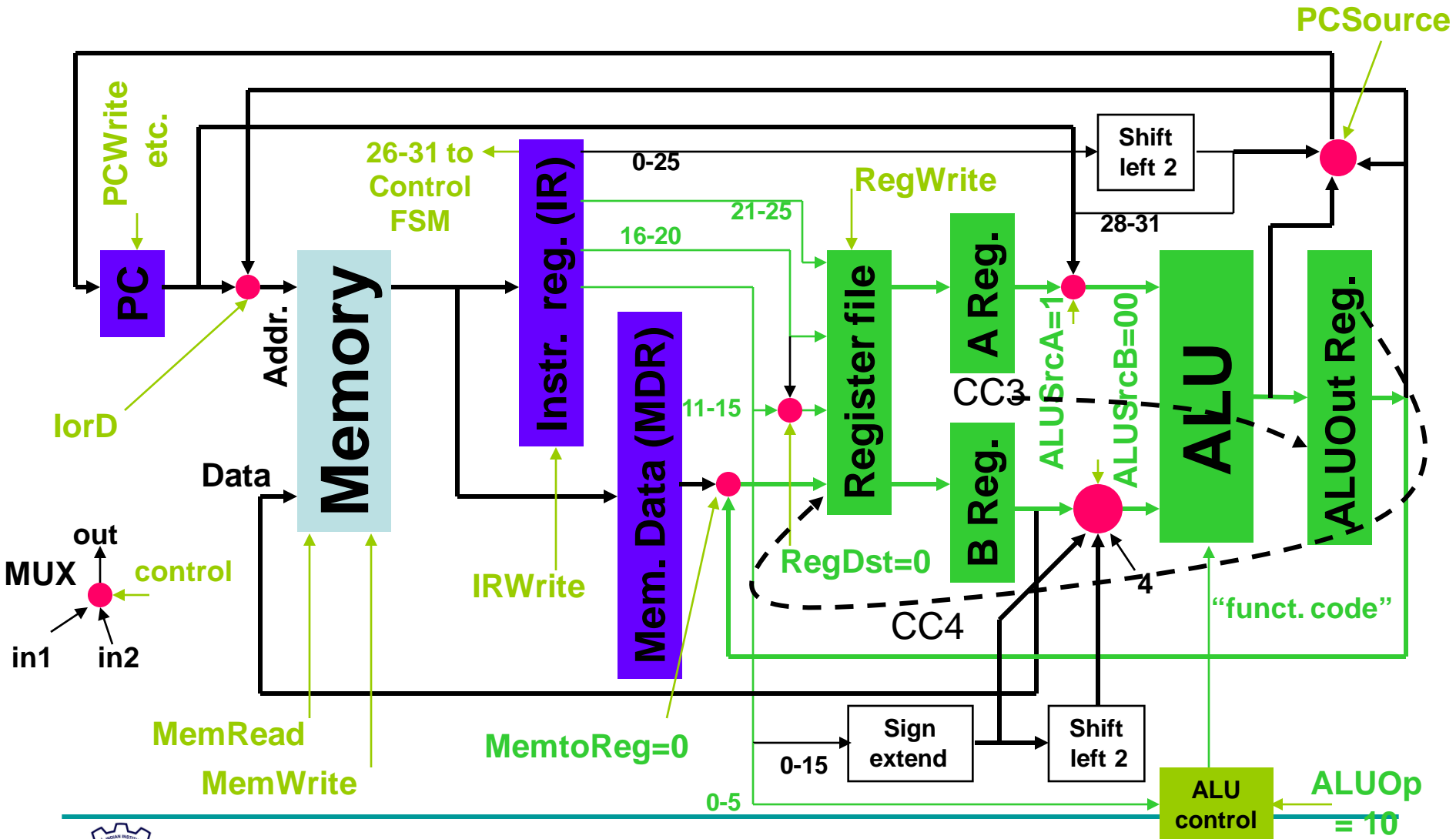




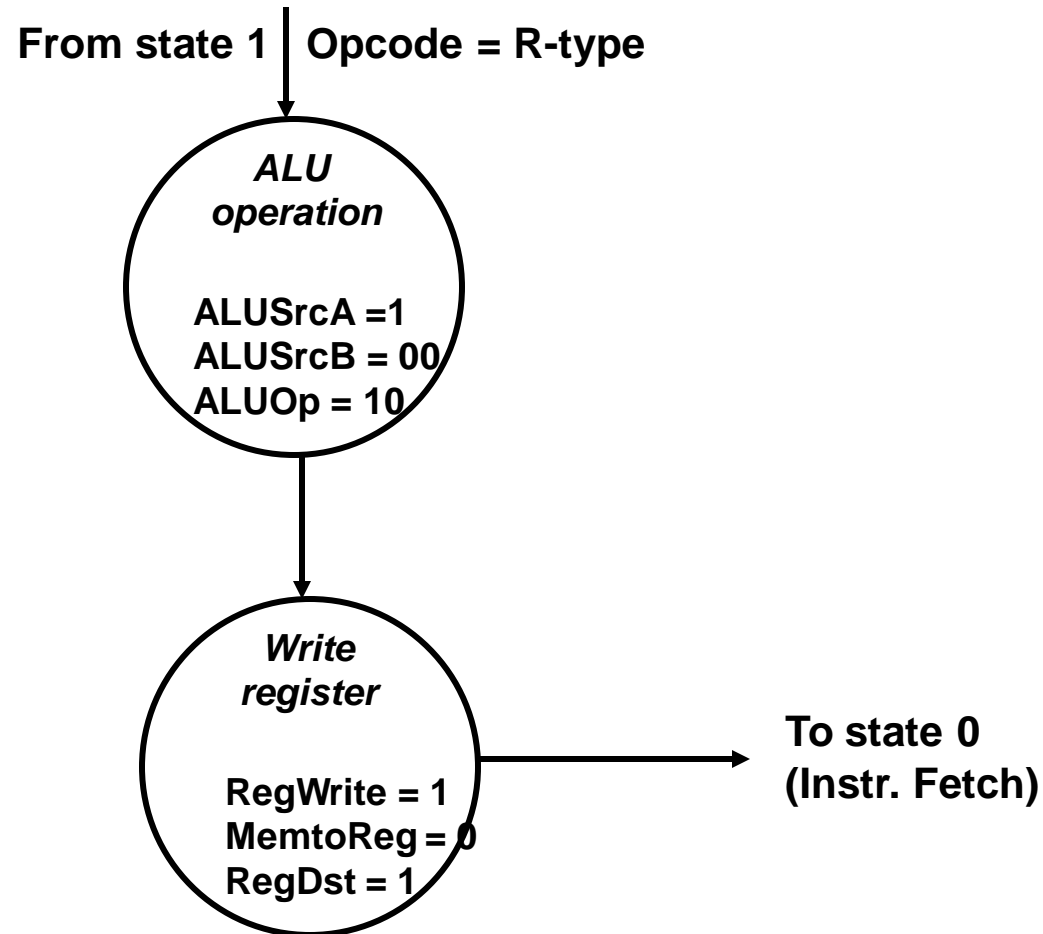
# FSM-M (Memory Access)



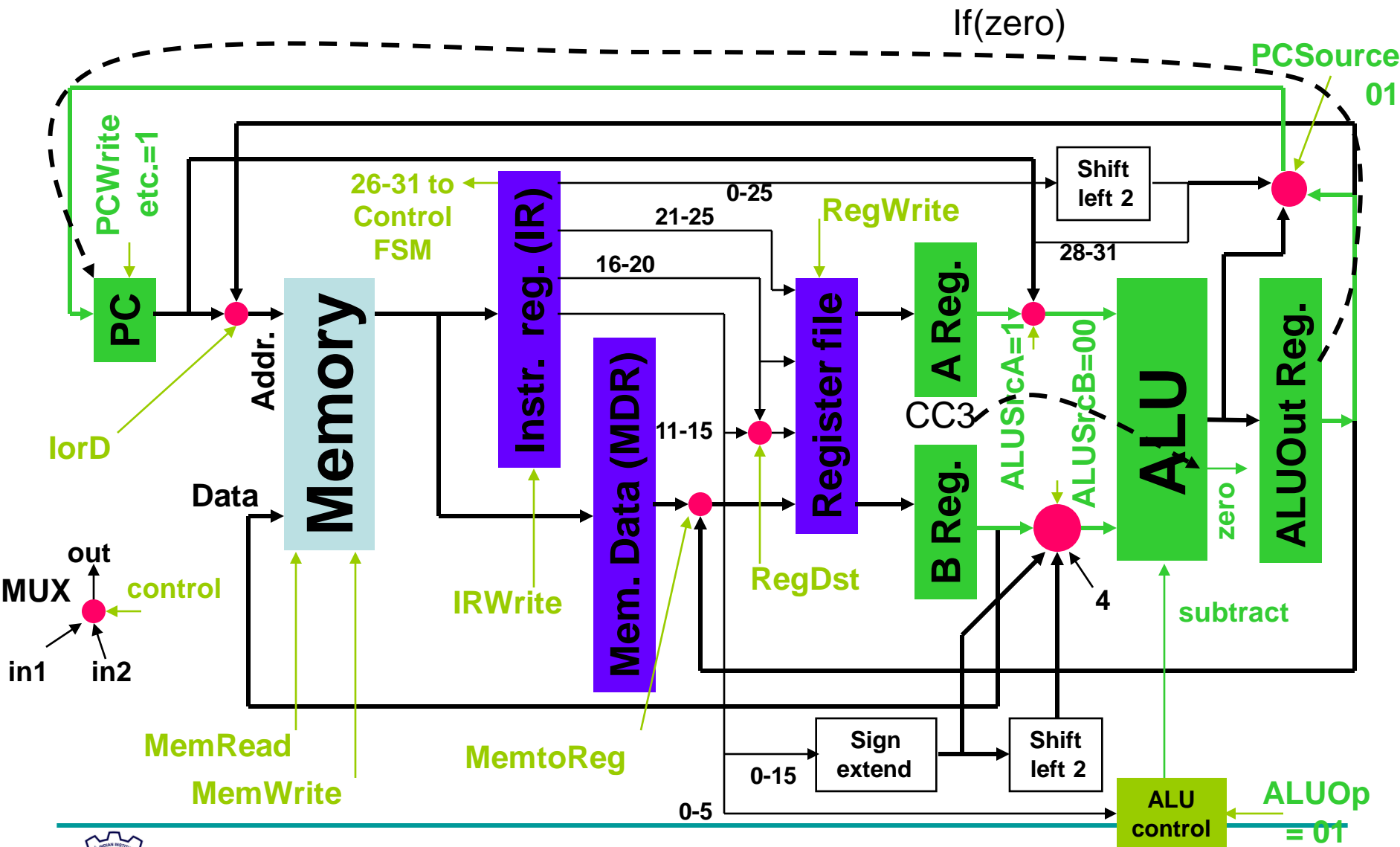
# State 1 (Opcode=R-type) → FSM-R (CC3-4)



# FSM-R (R-type Instruction)

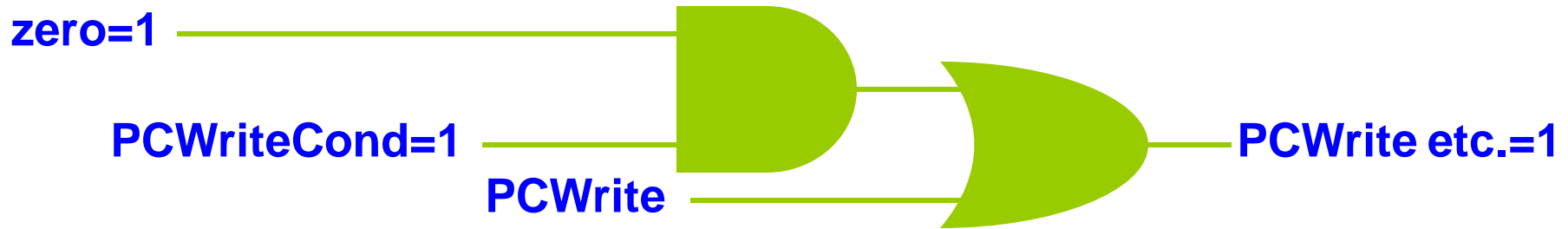


# State 1 (Opcode = beq) → FSM-B (CC3)

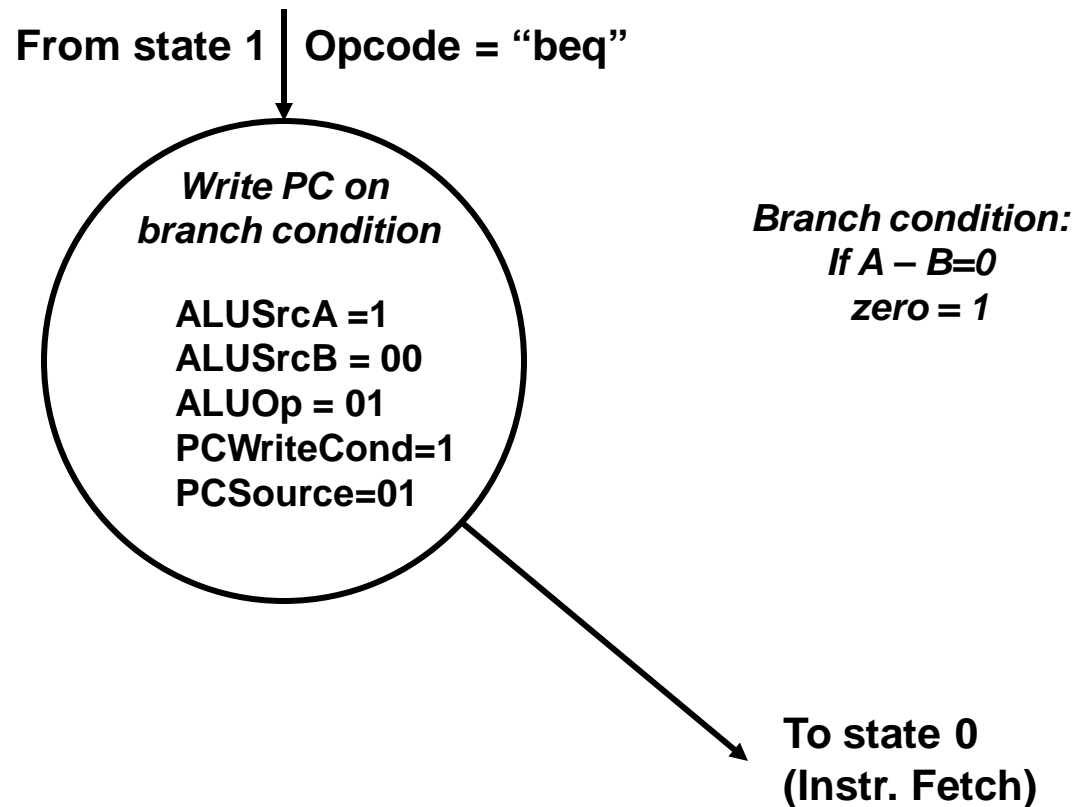


# Write PC on “zero”

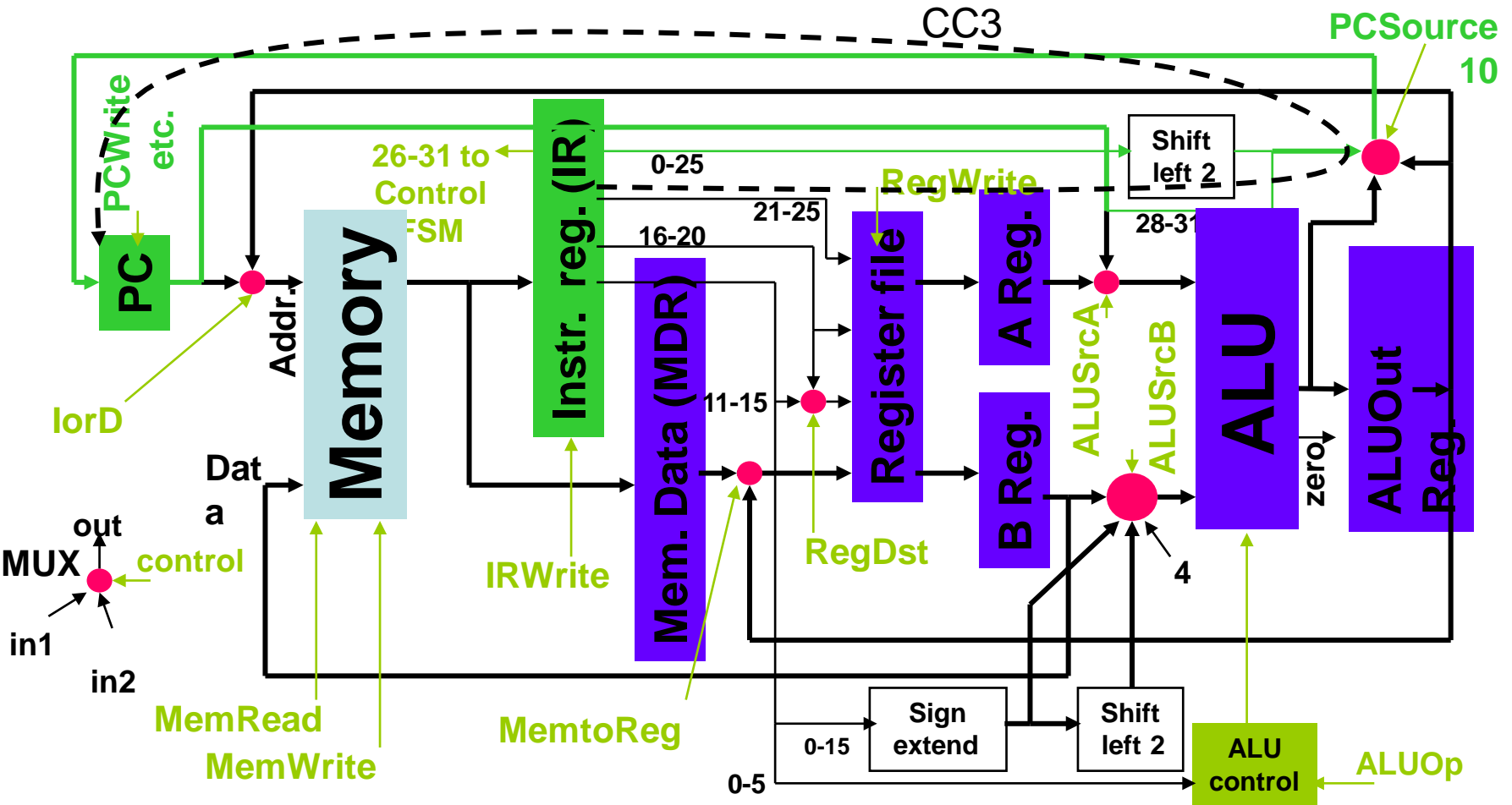
---



# FSM-B (Branch)



# State 1 (Opcode = j) → FSM-J (CC3)



# Write PC

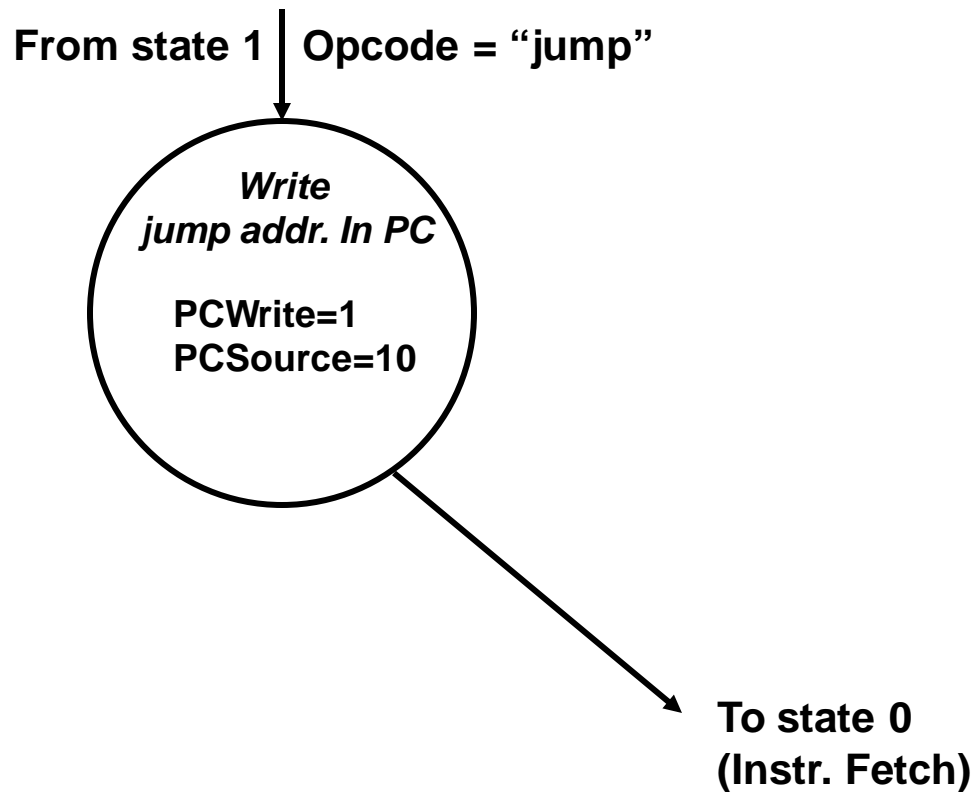
---



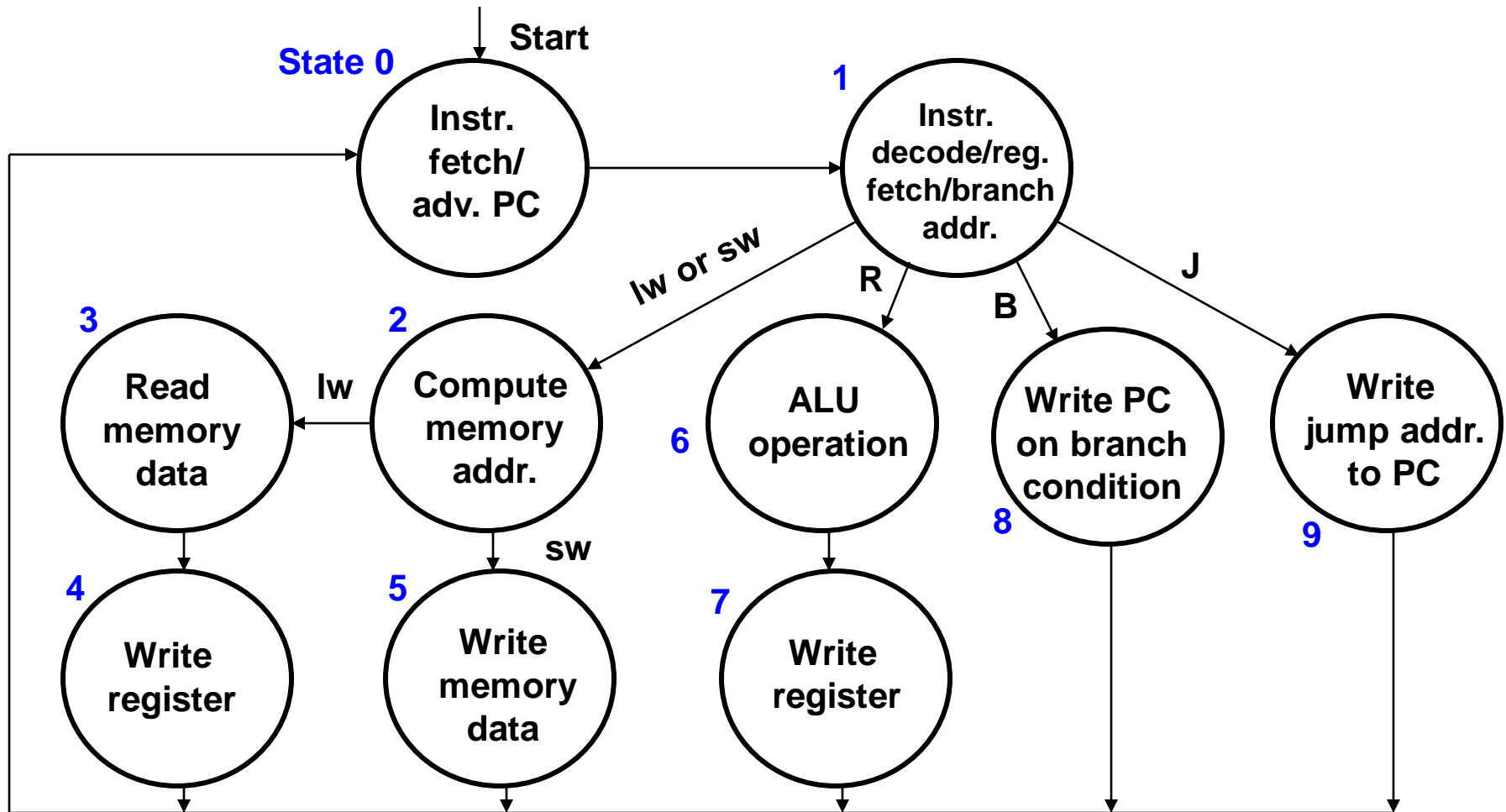


# FSM-J (Jump)

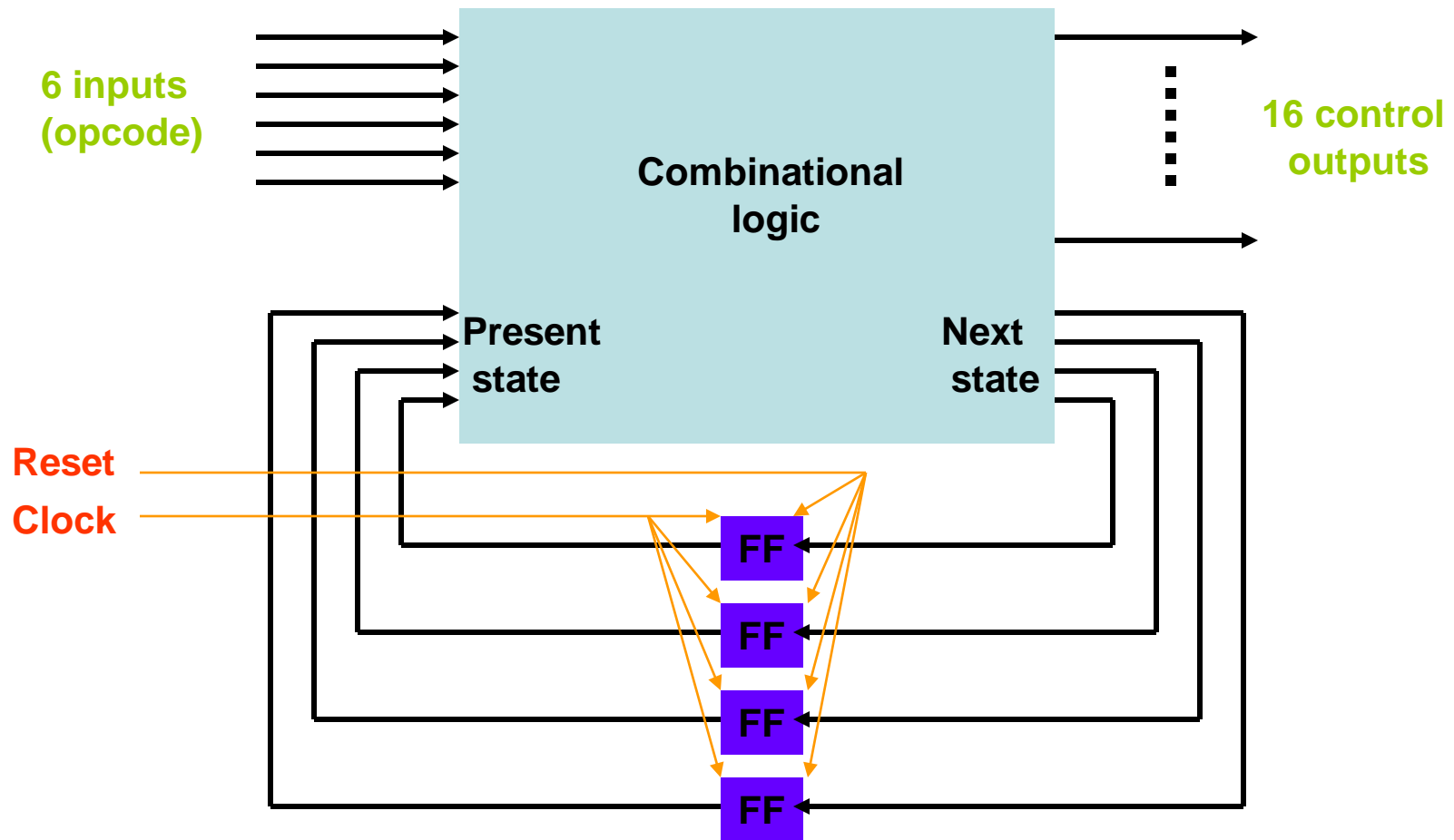
---



# Control FSM



# Control FSM (Controller)



# Designing the Control FSM

---

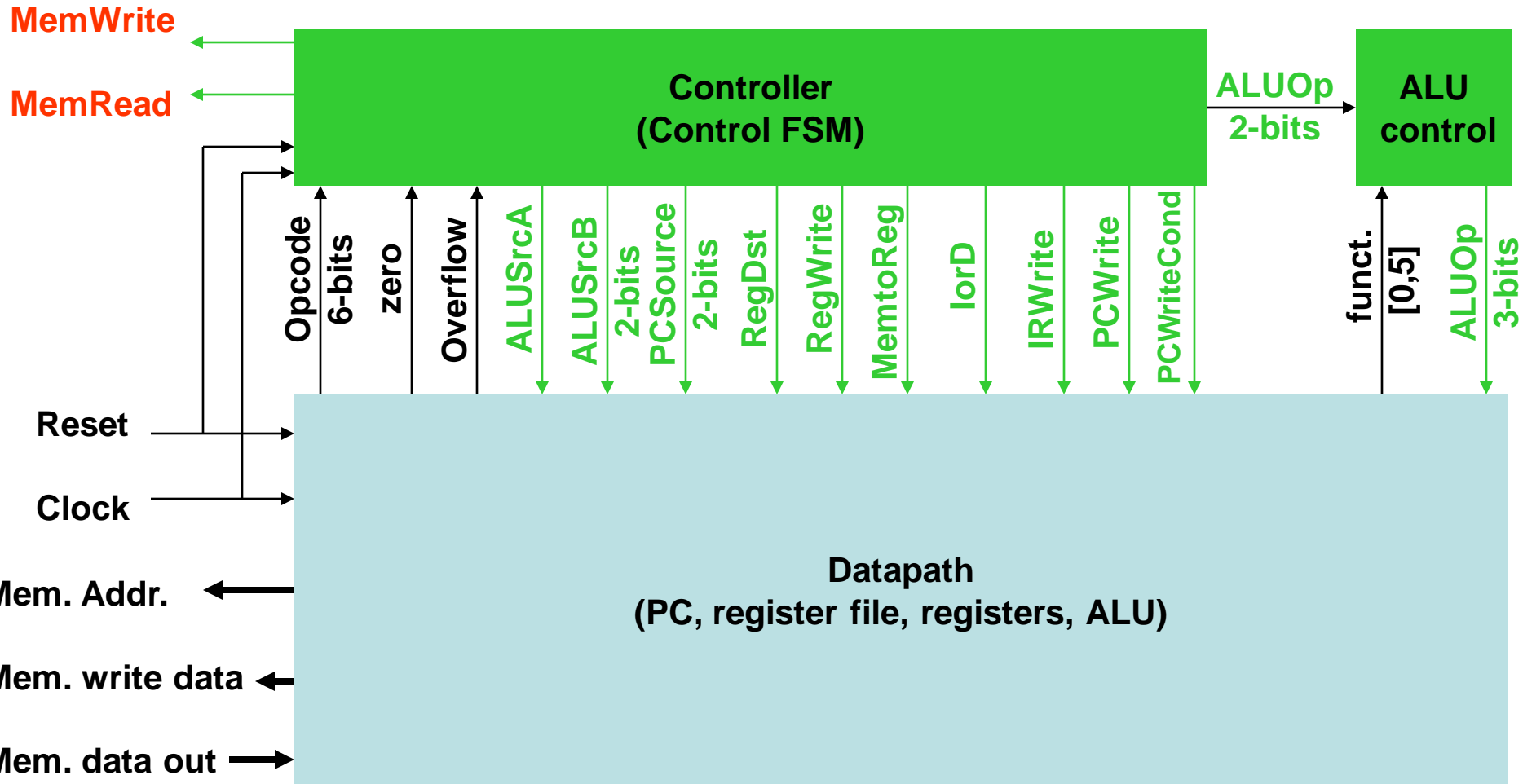
- Encode states; need 4 bits for 10 states, e.g.,
  - State 0 is 0000, state 1 is 0001, and so on.
- Write a truth table for combinational logic:

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



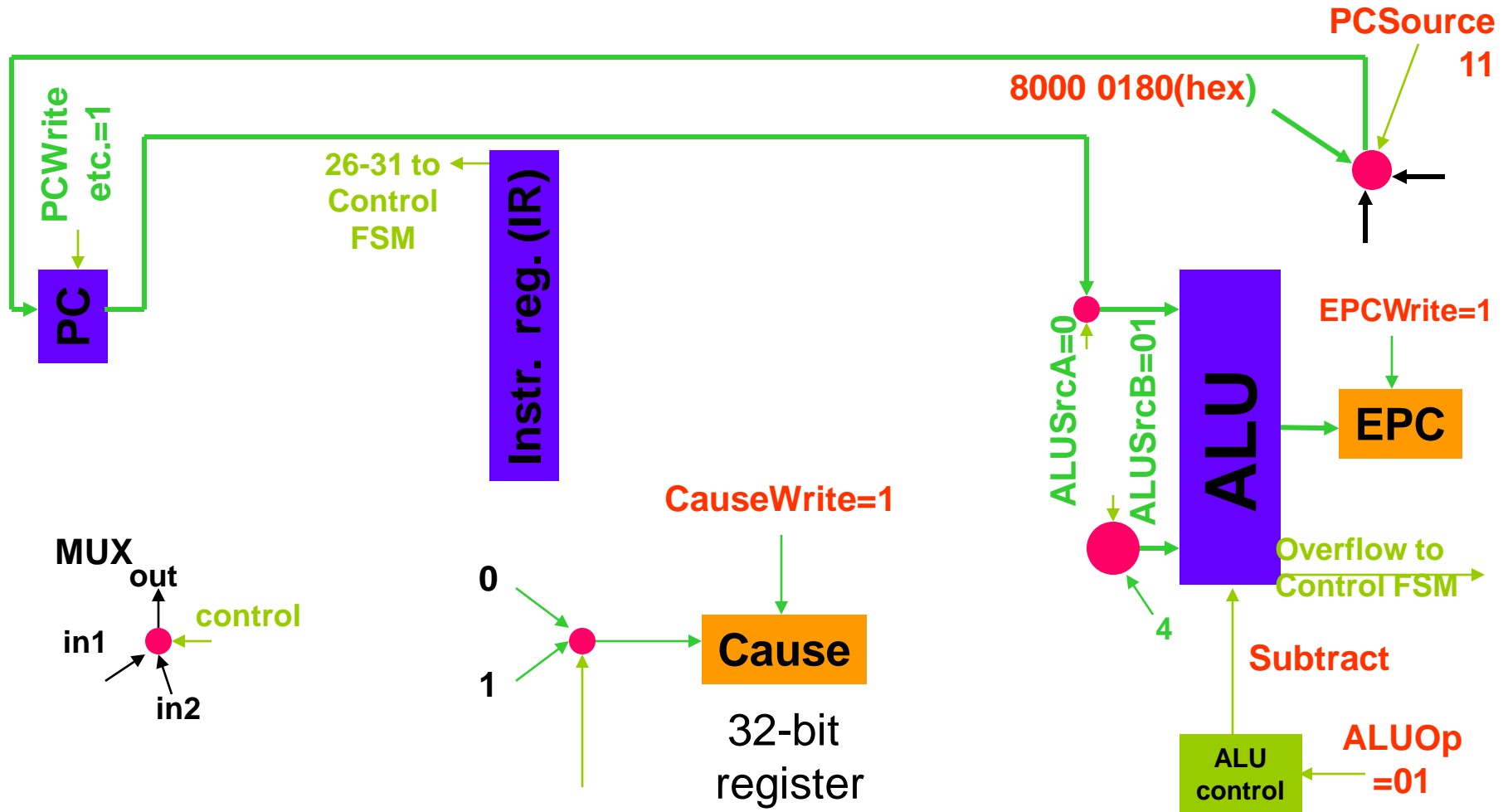
# Exceptions or Interrupts

---

- 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

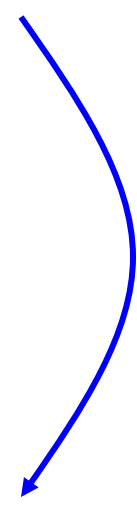




# Single-Cycle Datapath

---

- 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.*



# Thank You

