RISC Design

Virendra Singh
Professor
Computer Architecture and Dependable Systems Lab
Department of Electrical Engineering, and
Dept. of Computer Science & Engineering
Indian Institute of Technology Bombay
http://www.ee.iitb.ac.in/~viren/
E-mail: viren@{ee, cse}.iitb.ac.in

EE-739: Processor Design @ IITB

Lecture 15 (16 February 2022)

CADSL

$S \xrightarrow{I} S'$

Single cycle.

τ. (

$S \xrightarrow{I} S'$

$S \rightarrow P \rightarrow P' \rightarrow P'' \rightarrow S'$

Multicycle implementation

↑

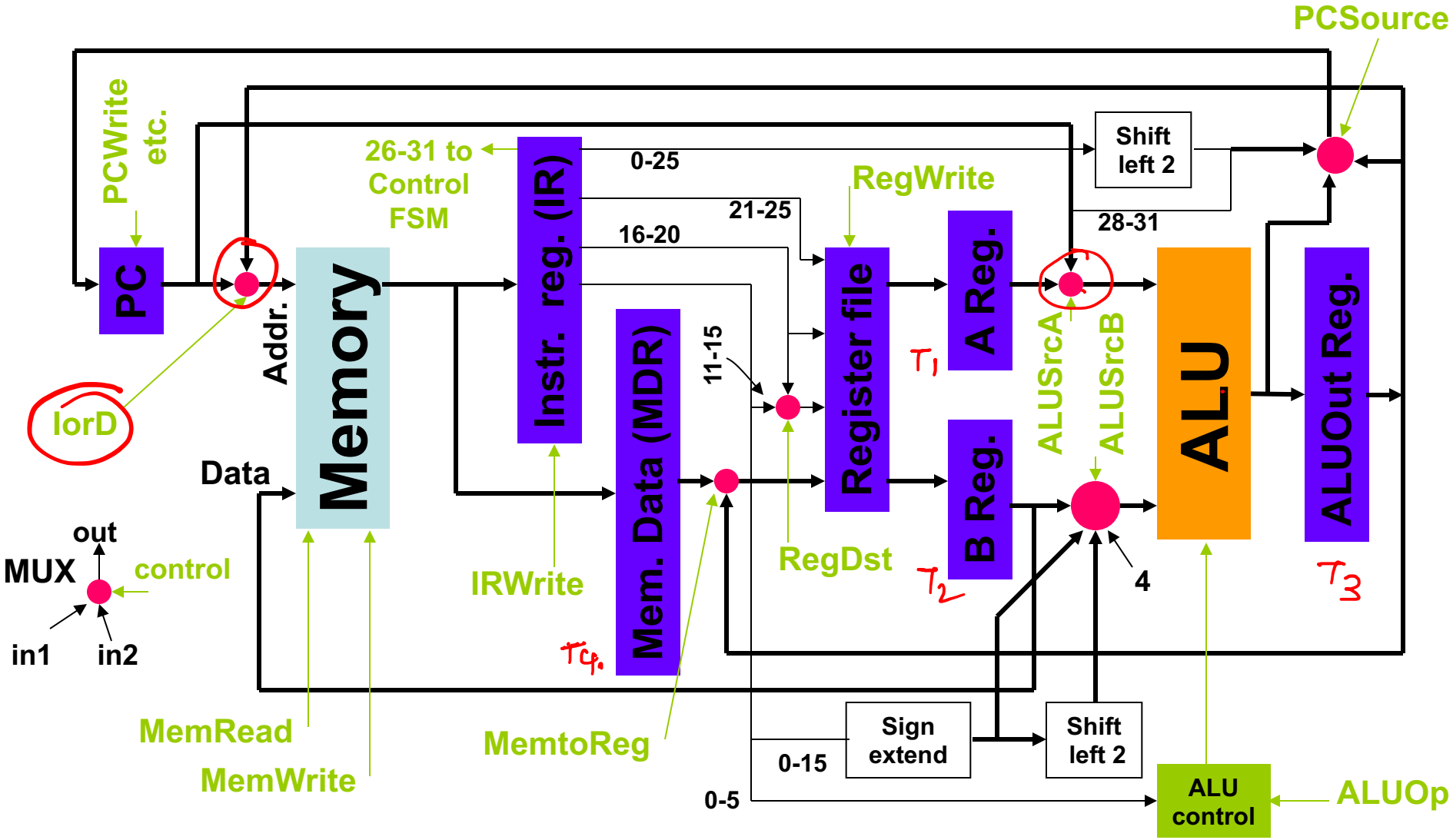
{ Reuse of resources (cost effective)

↑

(HFC)

Merged level 2 FC

Multicycle Datapath

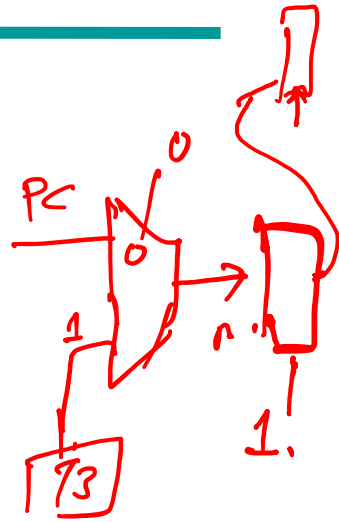


3 to 5 Cycles for an Instruction

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

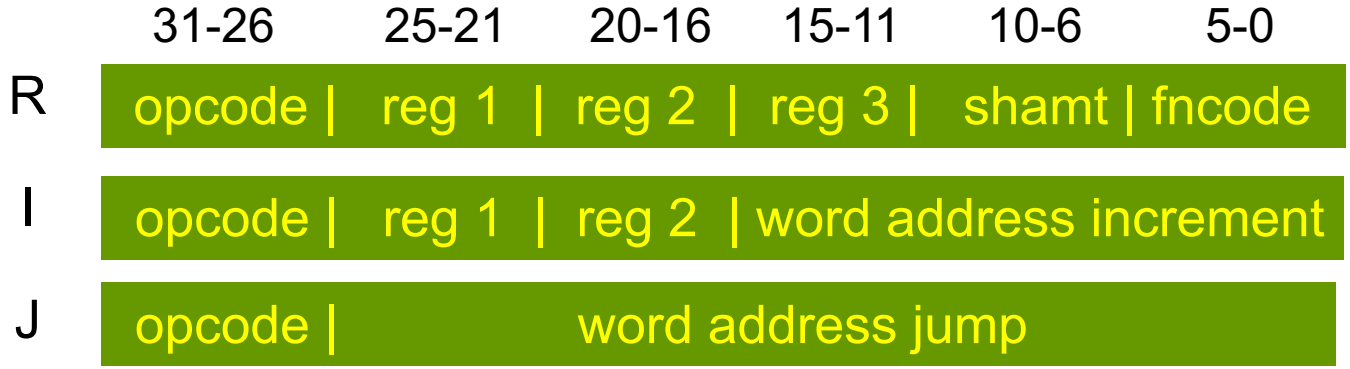
Cycle 1 of 5: Instruction Fetch (IF)

- Read instruction into IR, M[PC] → IR
  - Control signals used:
    - » IorD = 0 select PC
    - » MemRead = 1 read memory
    - » IRWrite = 1 write IR



- Increment PC, PC + 4 → 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 ← 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 ← 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:
    - » **IorD = 1 select ALUOut into mem adr.**
    - » **MemRead = 1 read memory to MDR**
- I type, sw: write M[ALUOut] from B reg
  - Control signals used:
    - » **IorD = 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.

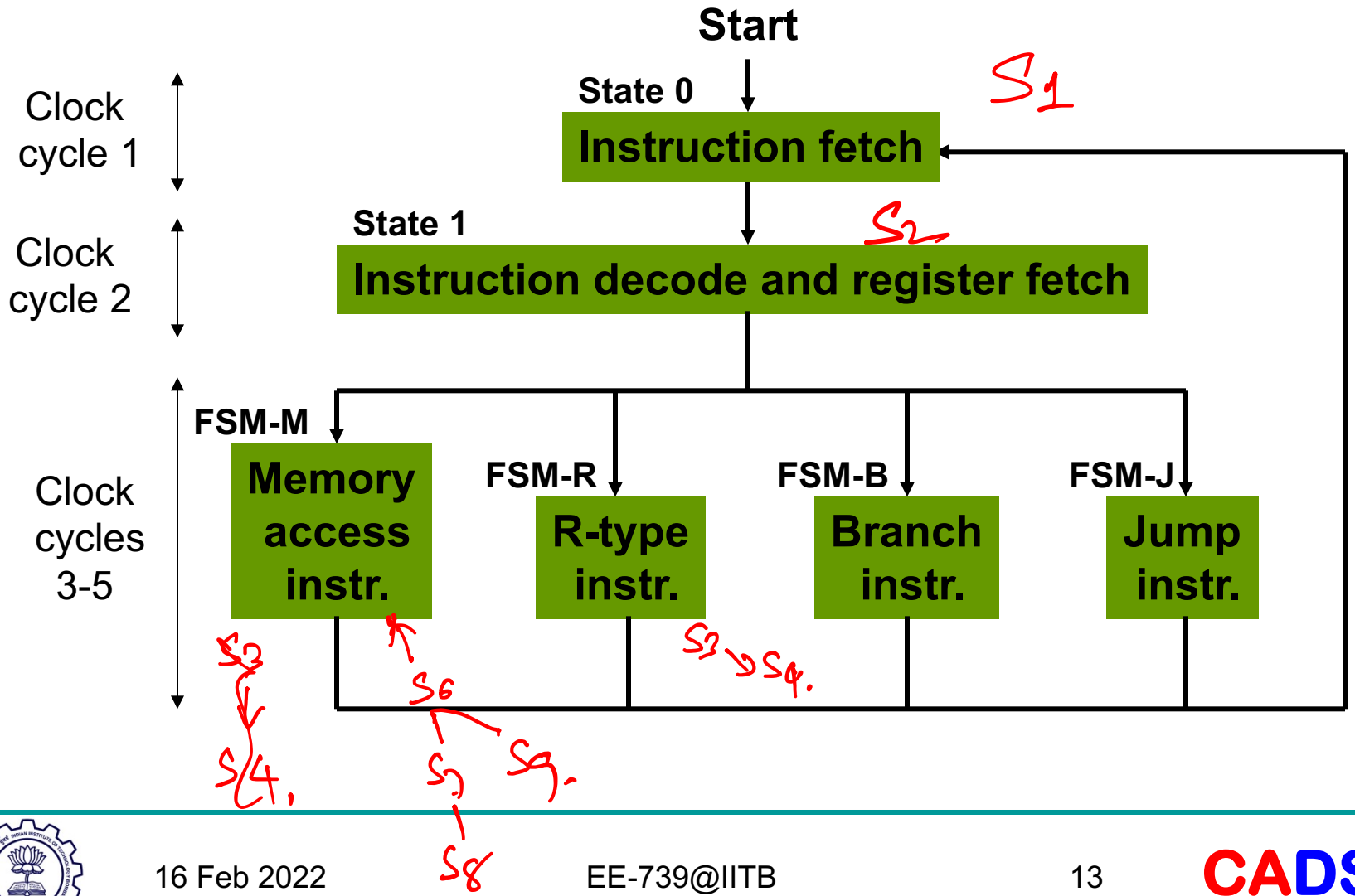
1-bit Control Signals

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

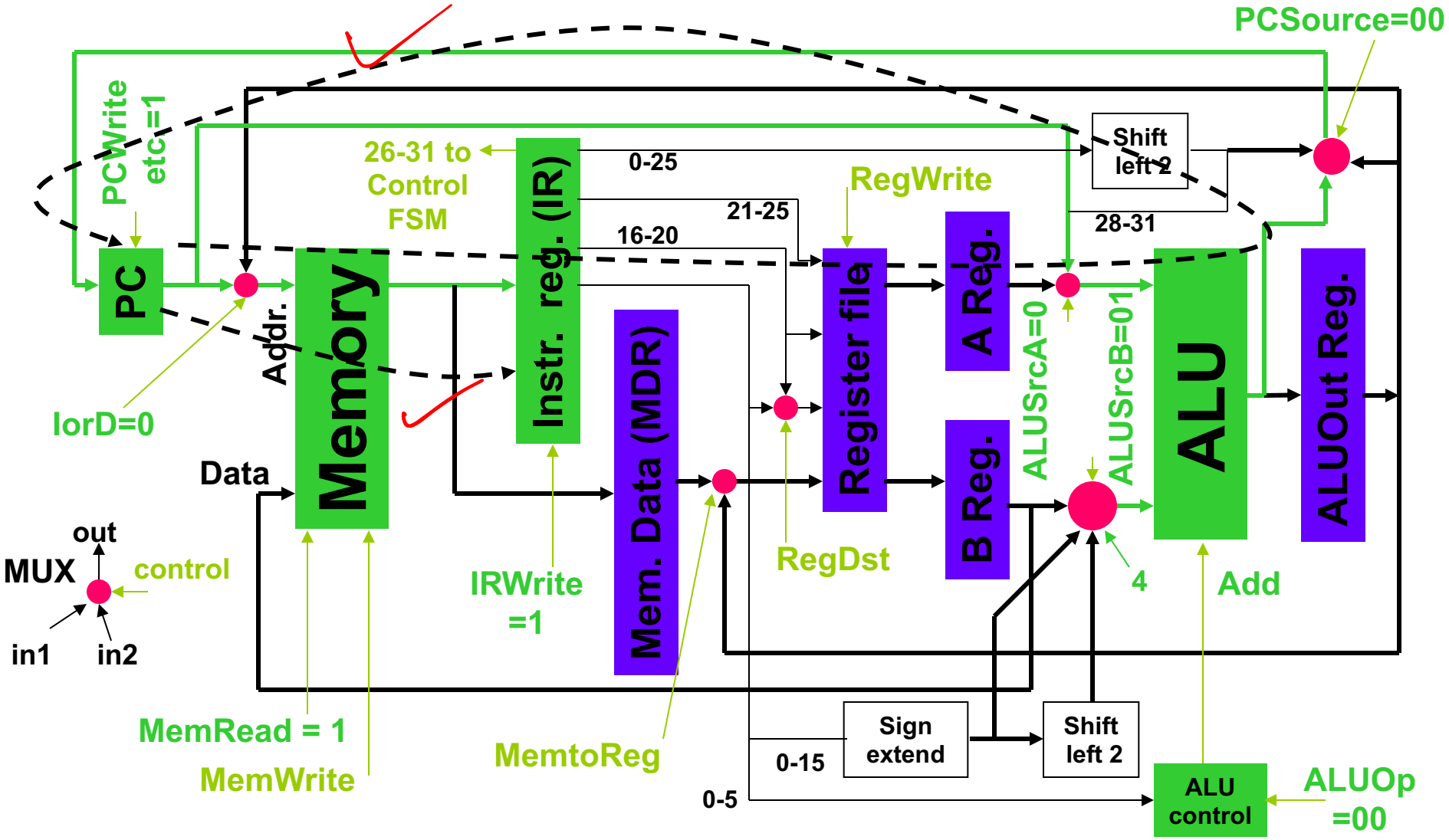
2-bit Control Signals

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 ← B reg.
	01	Second input of ALU ← 4 (constant)
	10	Second input of ALU ← 0-15 bits of IR sign ext. to 32b
	11	Second input of ALU ← 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

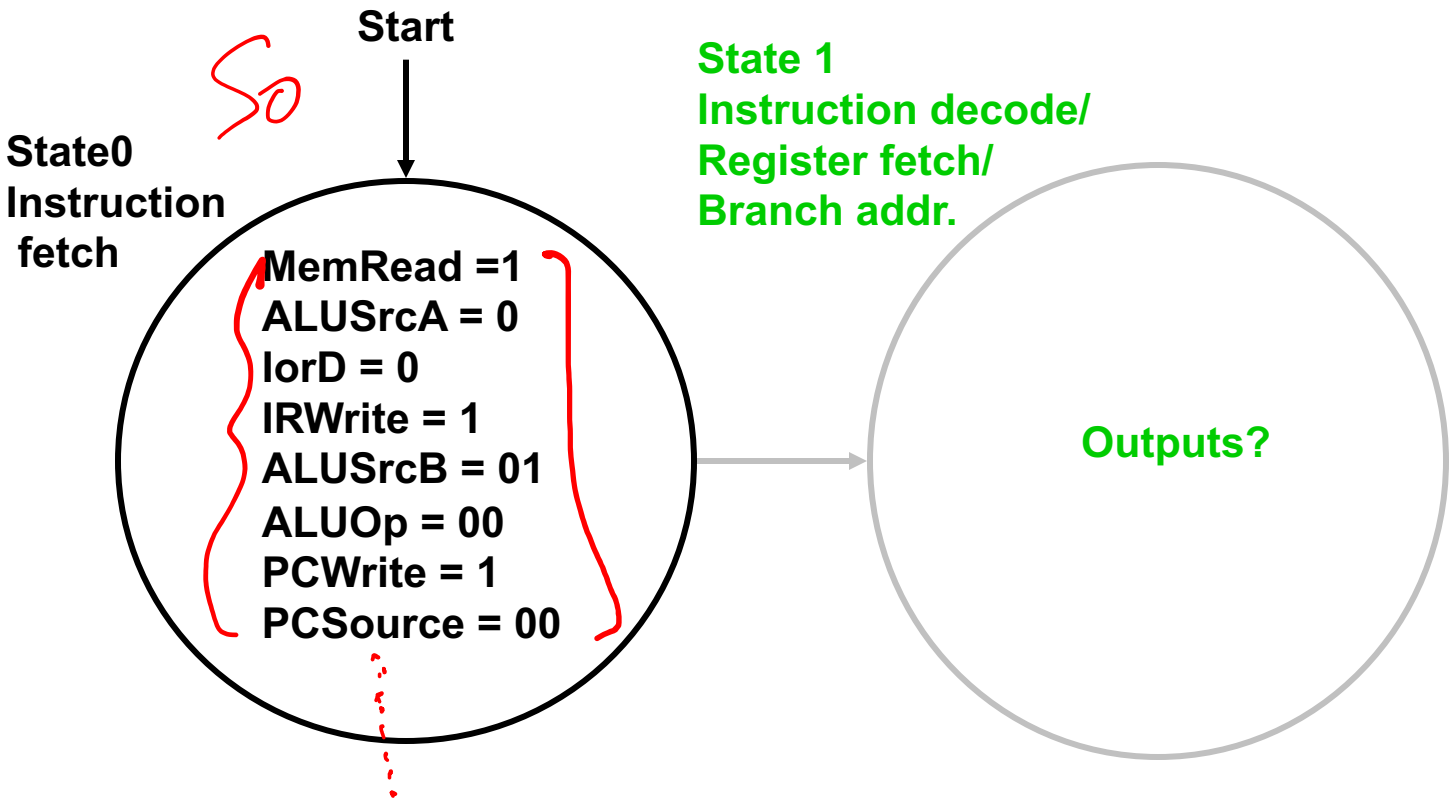
Control: Finite State Machine



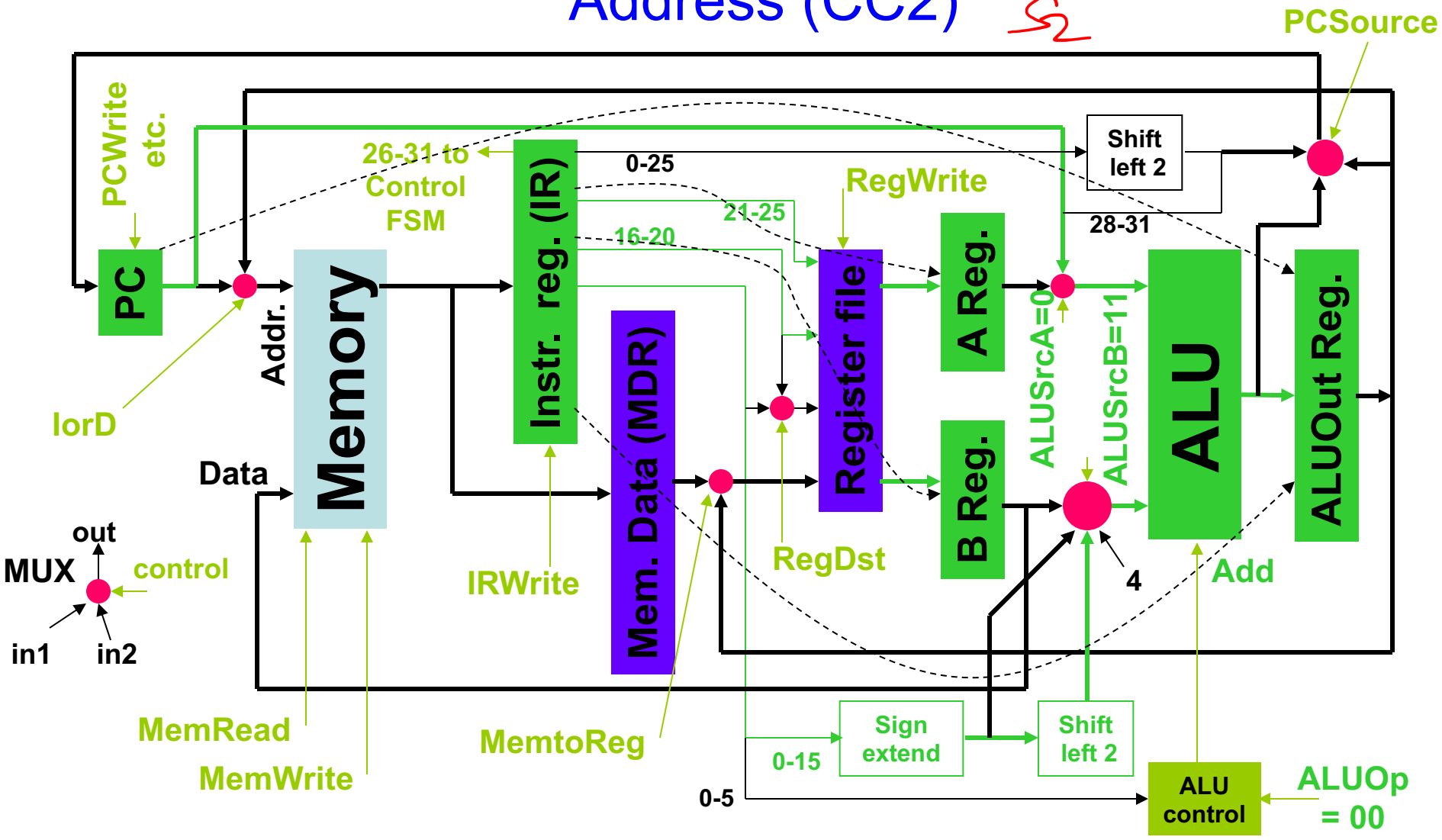
State 0: Instruction Fetch (CC1)



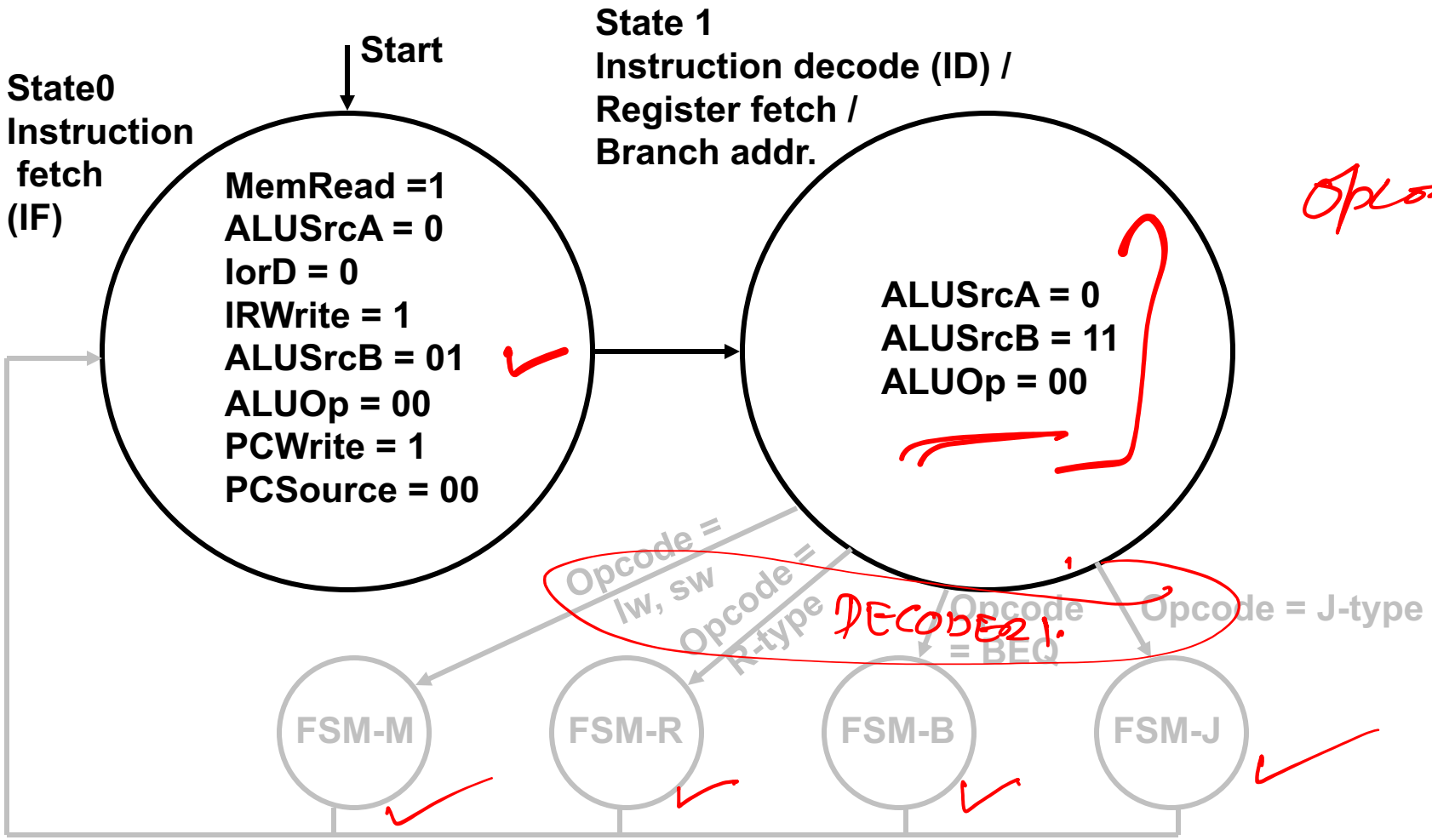
State 0 Control FSM Outputs



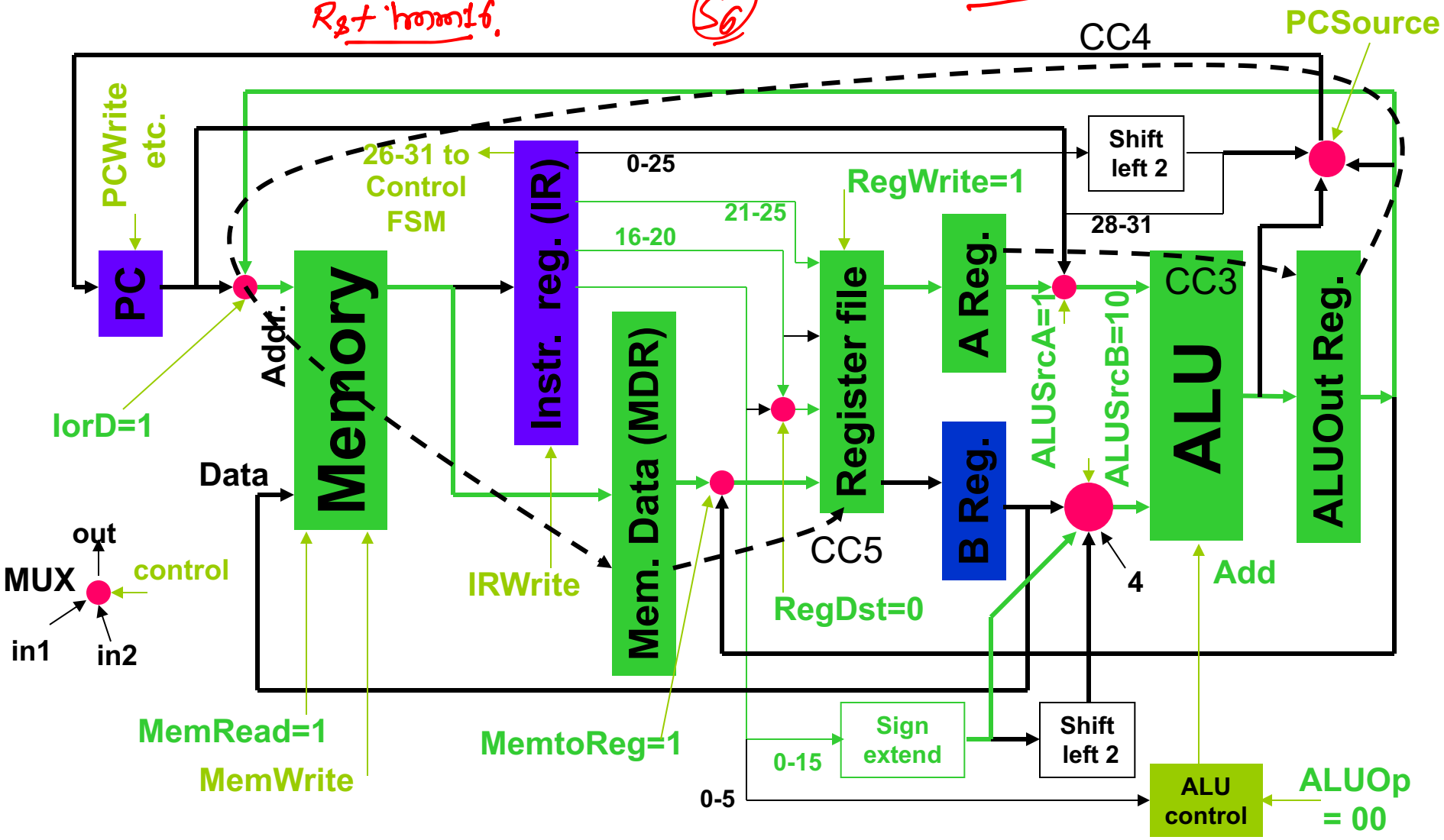
State 1: Instr. Decode/Reg. Fetch/ Branch Address (CC2)



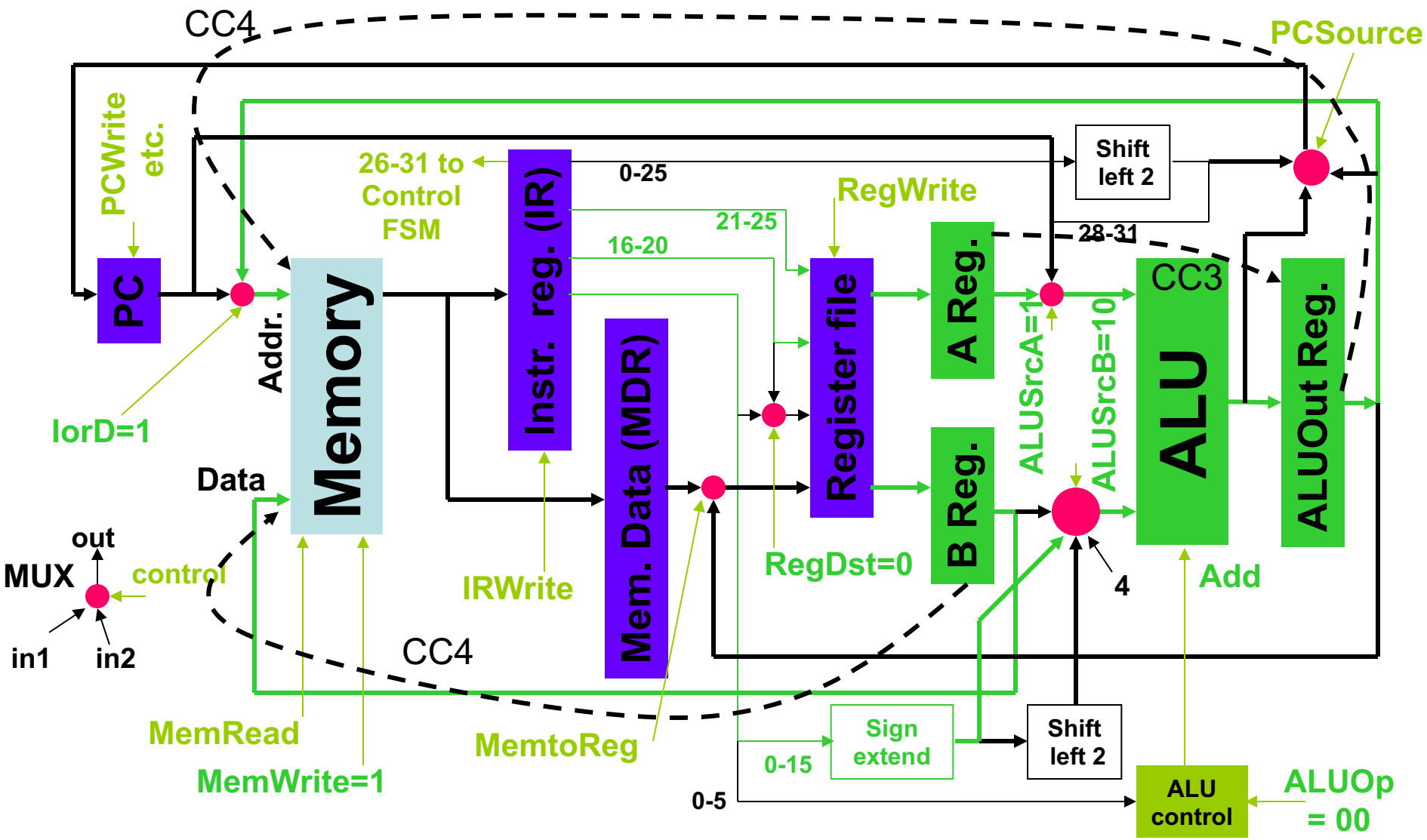
State 1 Control FSM Outputs



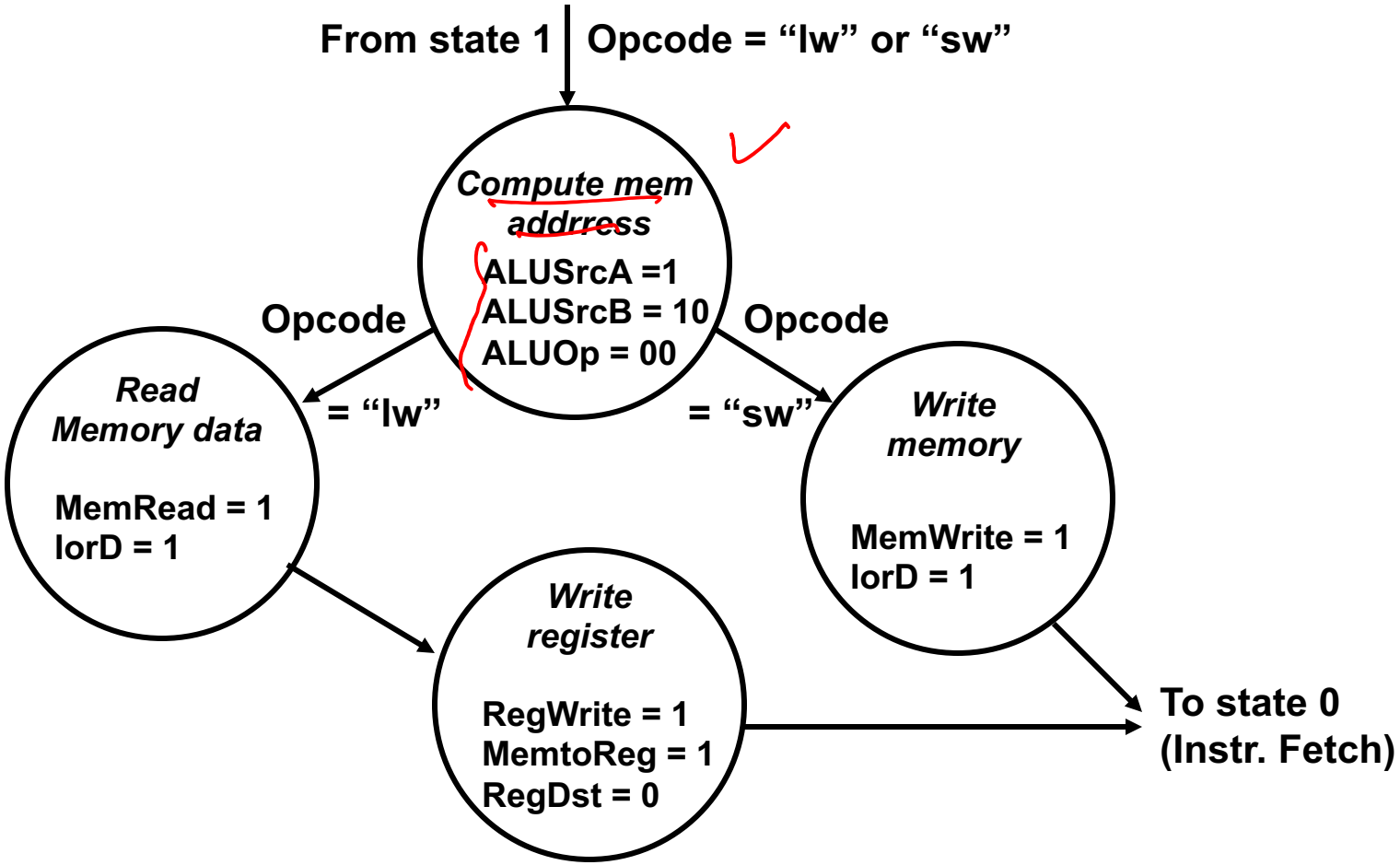
State 1 (Opcode = lw) → FSM-M (CC3-5)



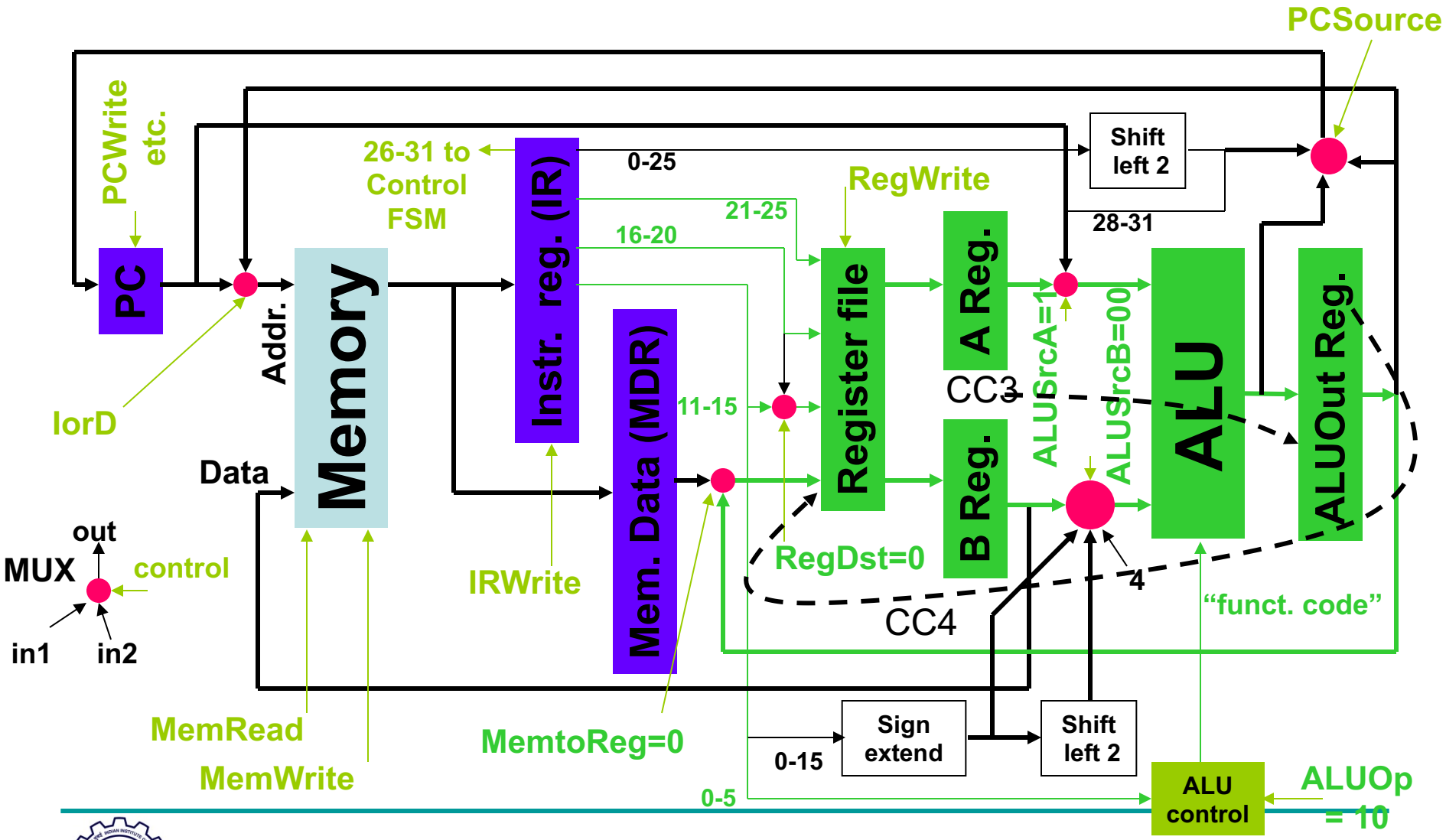
State 1 (Opcode= sw)→FSM-M (CC3-4)



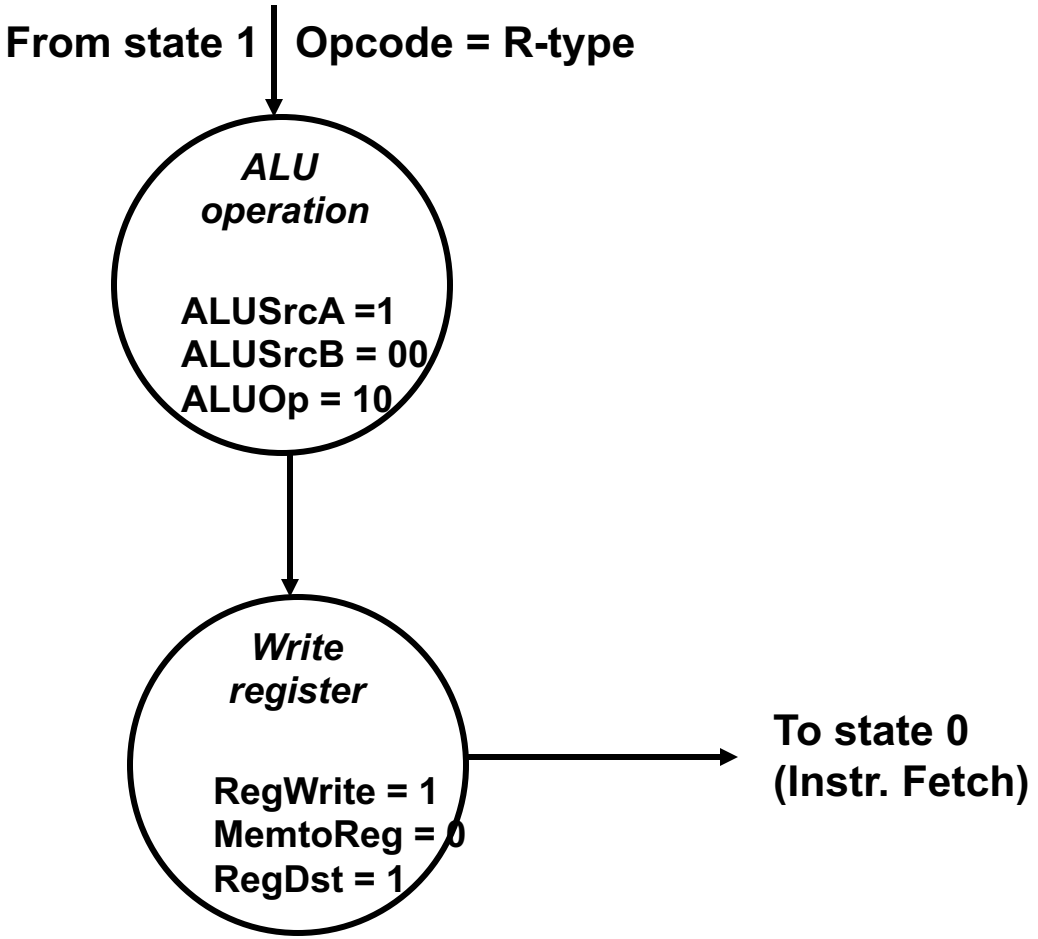
FSM-M (Memory Access)



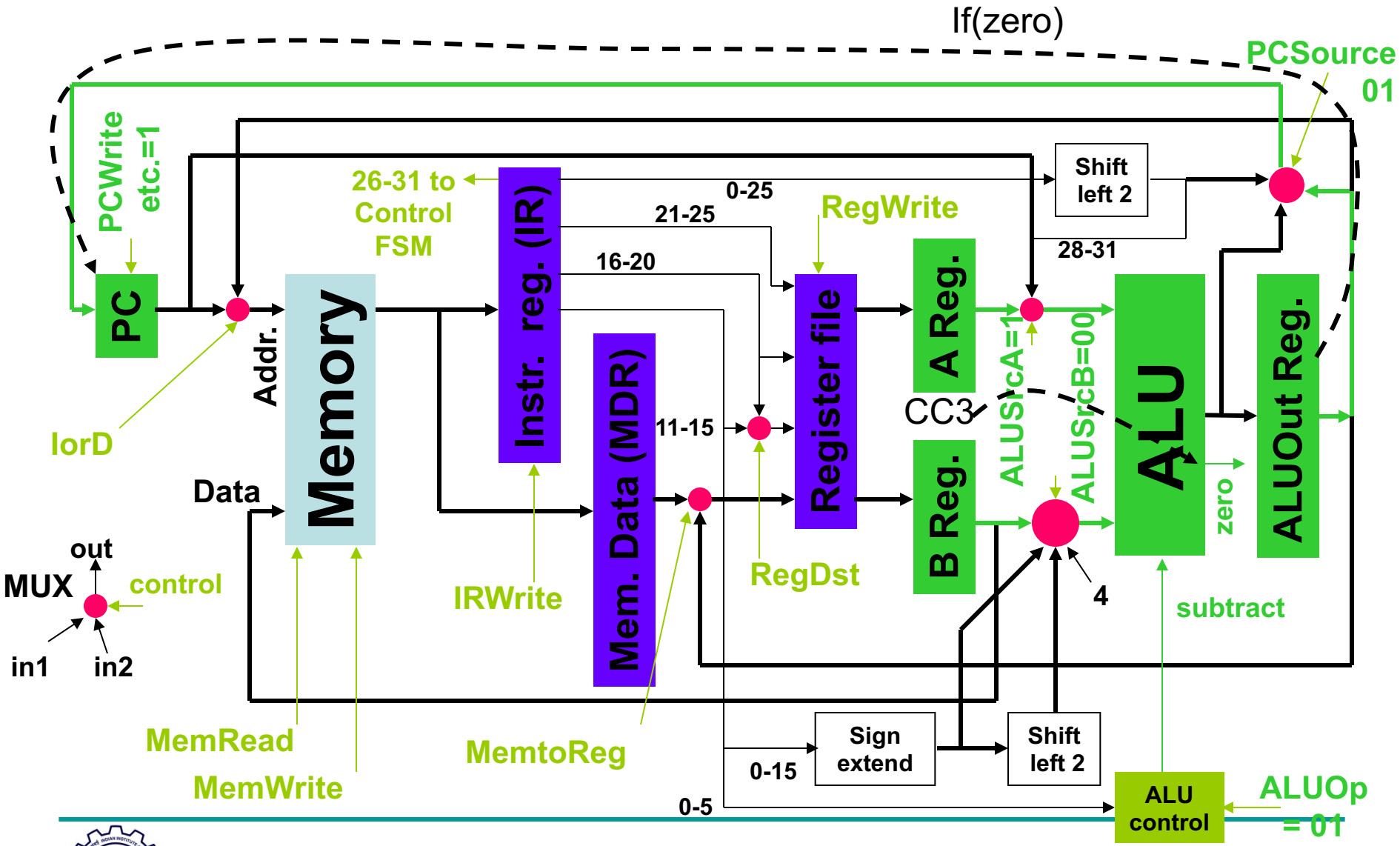
State 1(Opcode=R-type)→FSM-R (CC3-4)



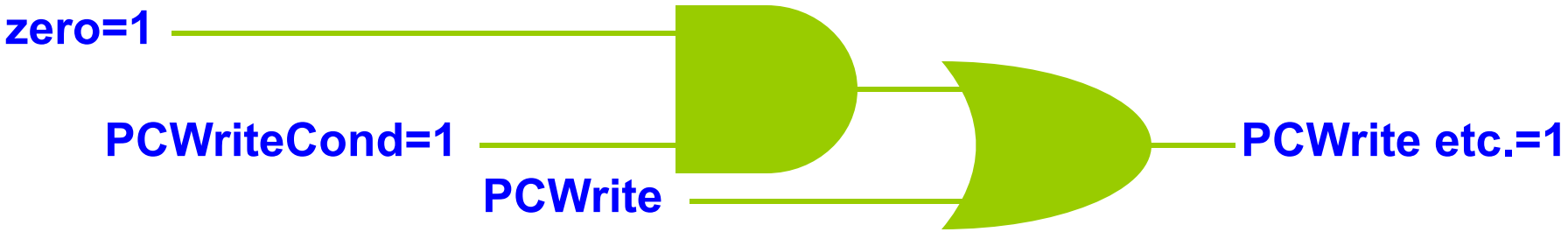
FSM-R (R-type Instruction)



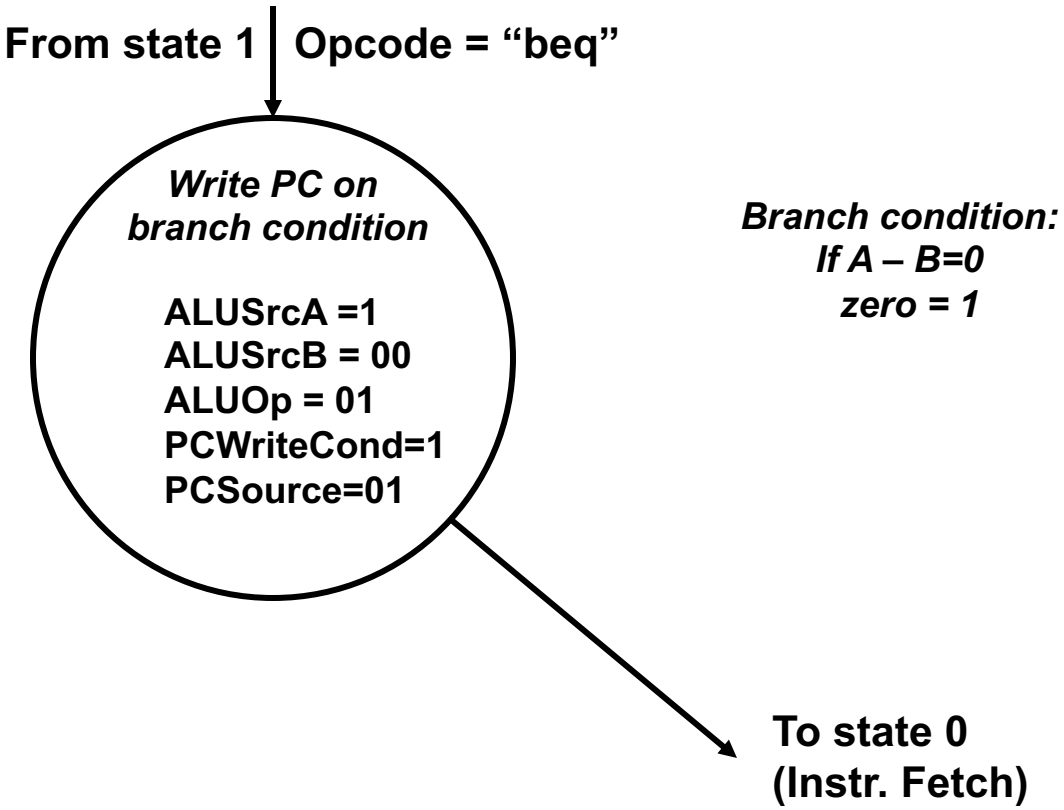
State 1 (Opcode = beq) → FSM-B (CC3)



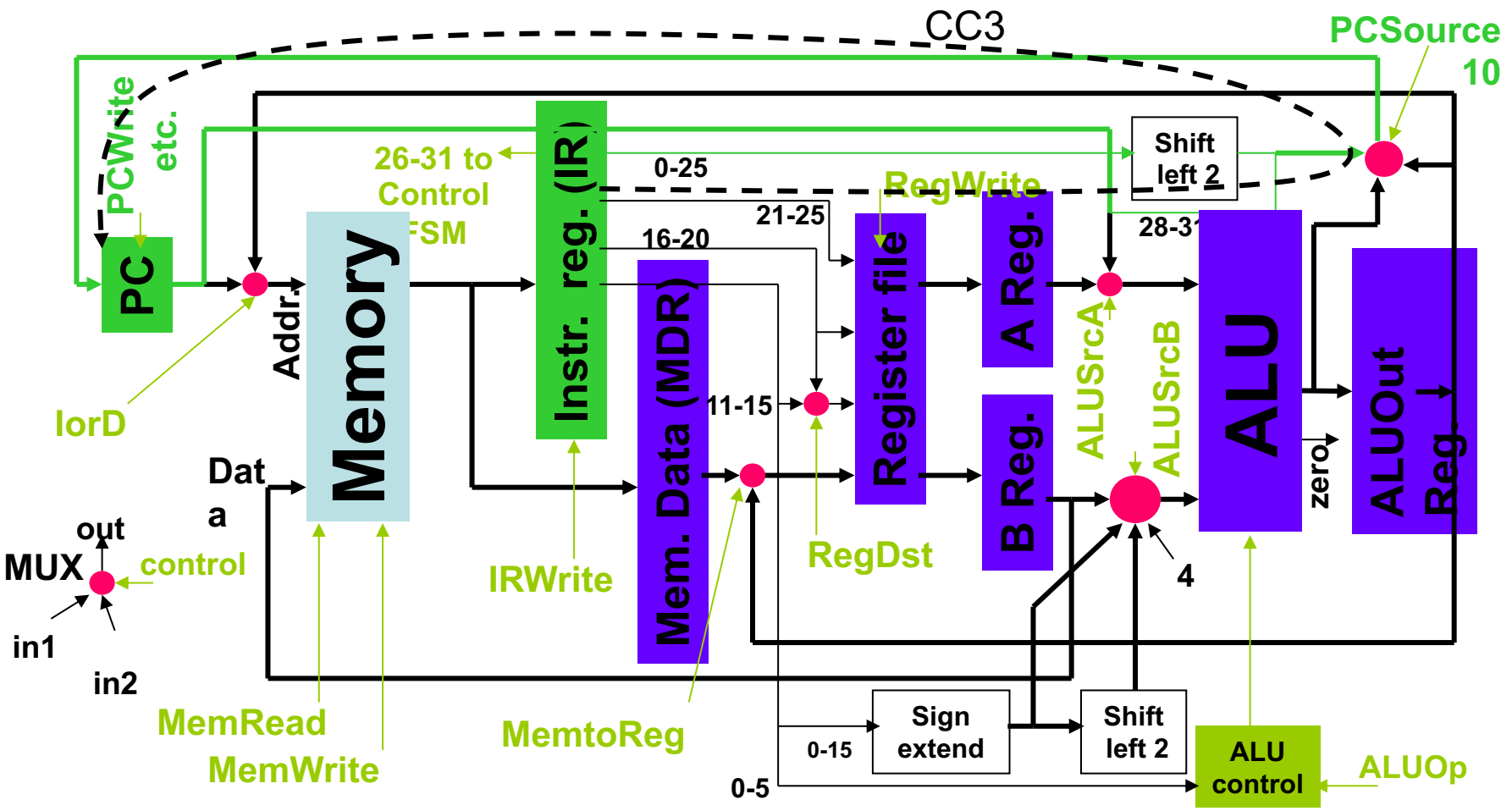
Write PC on “zero”



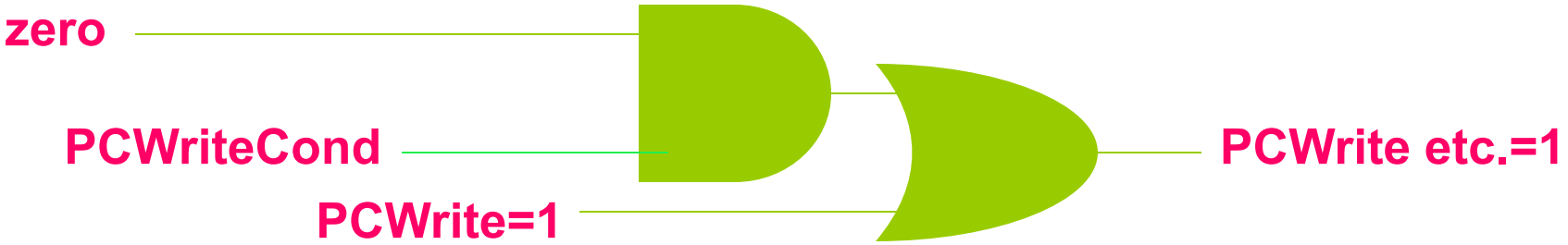
FSM-B (Branch)


From state 1
Opcode = “beq”
Write PC on branch condition
ALUSrcA =1
ALUSrcB = 00
ALUOp = 01
PCWriteCond=1
PCSource=01
Branch condition:
If A – B=0
zero = 1
To state 0
(Instr. Fetch)

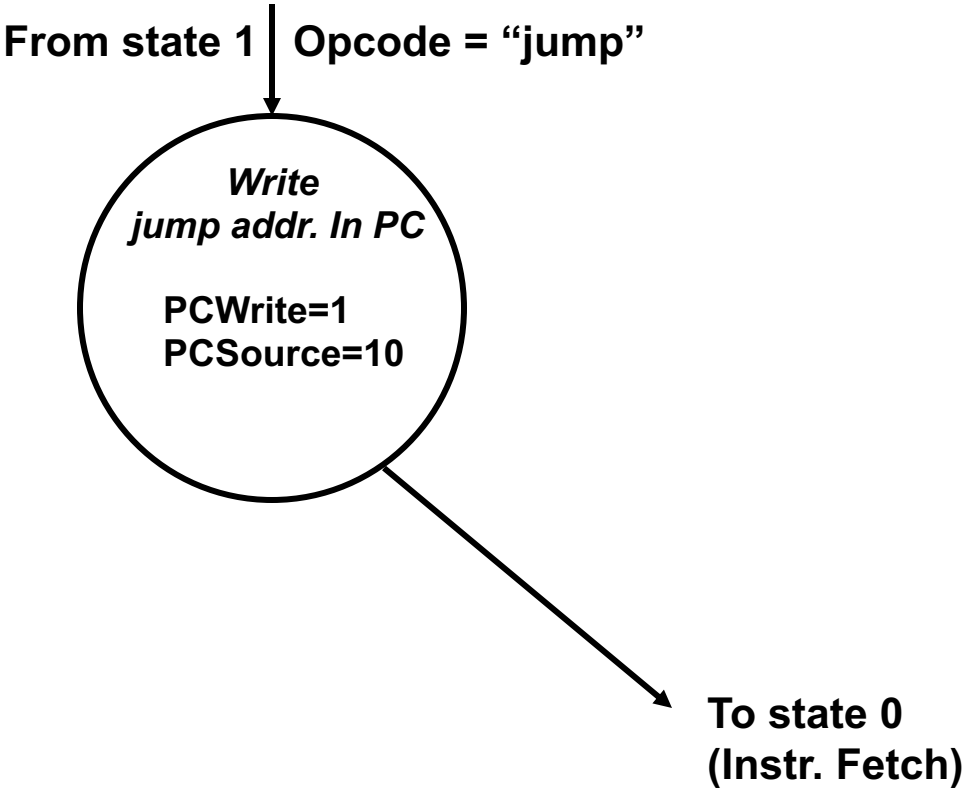
State 1 (Opcode = j) → FSM-J (CC3)



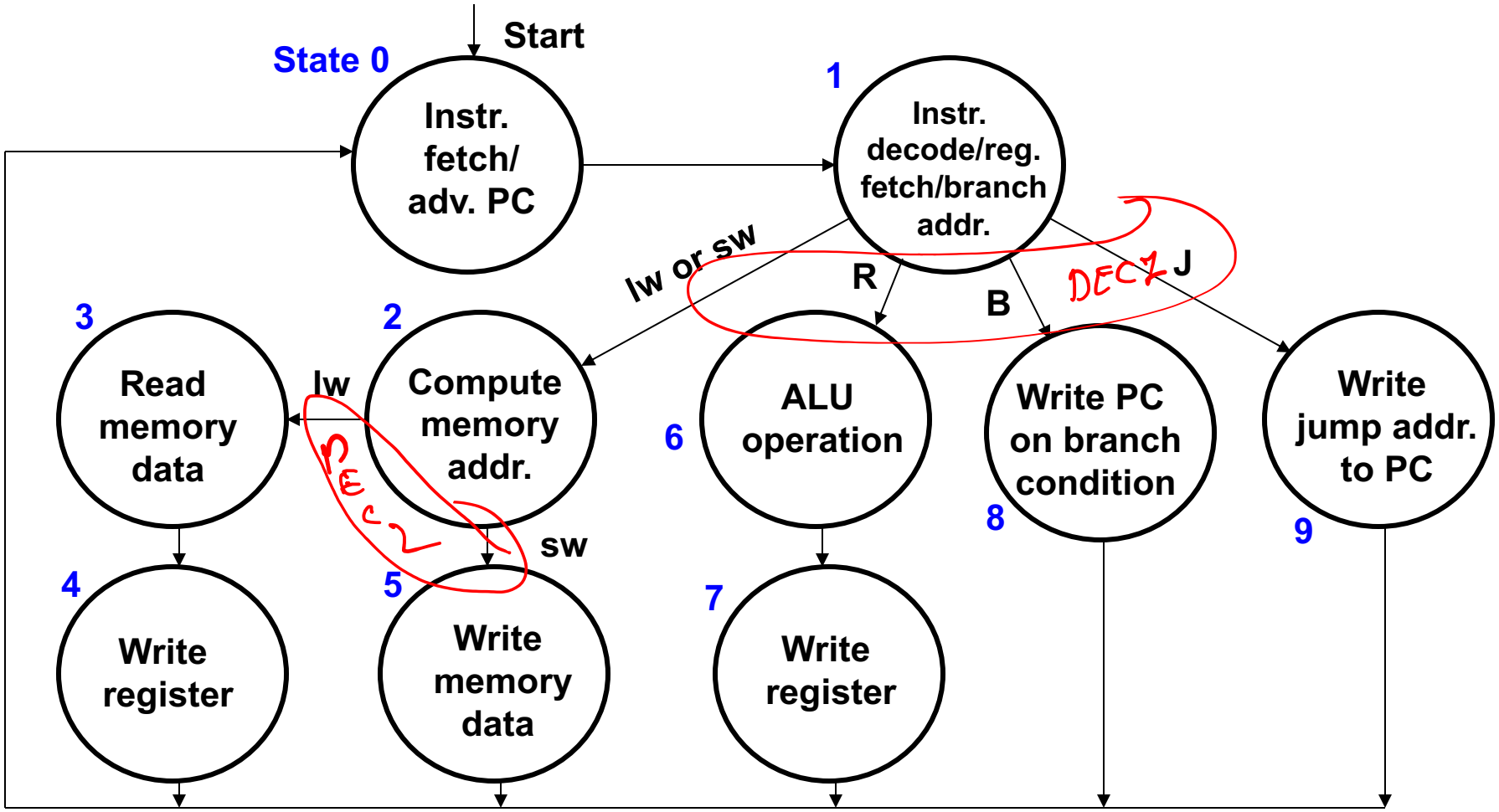
Write PC



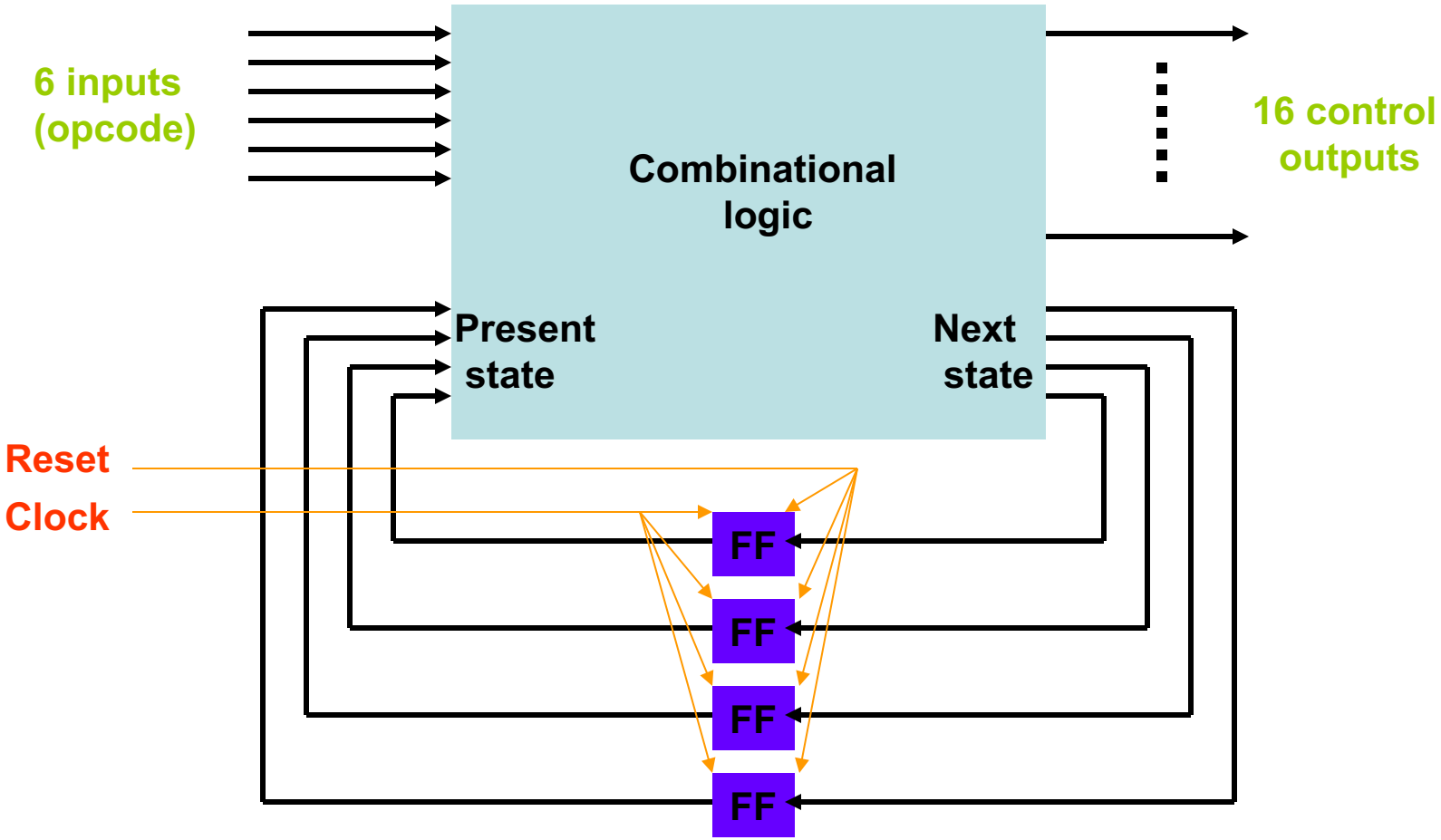
FSM-J (Jump)


From state 1
Opcode = “jump”
Write
jump addr. In PC
PCWrite=1
PCSource=10
To state 0
(Instr. Fetch)

Control FSM


Start
State 0
Instr. fetch/ adv. PC
1
Instr. decode/reg. fetch/branch addr.
lw or sw
R
B
J
DEC 2
DEC 1
3
Read memory data
2
Compute memory addr.
lw
sw
6
ALU operation
Write PC on branch condition
8
Write jump addr. to PC
9
4
Write register
5
Write memory data
7
Write register

Control FSM (Controller)



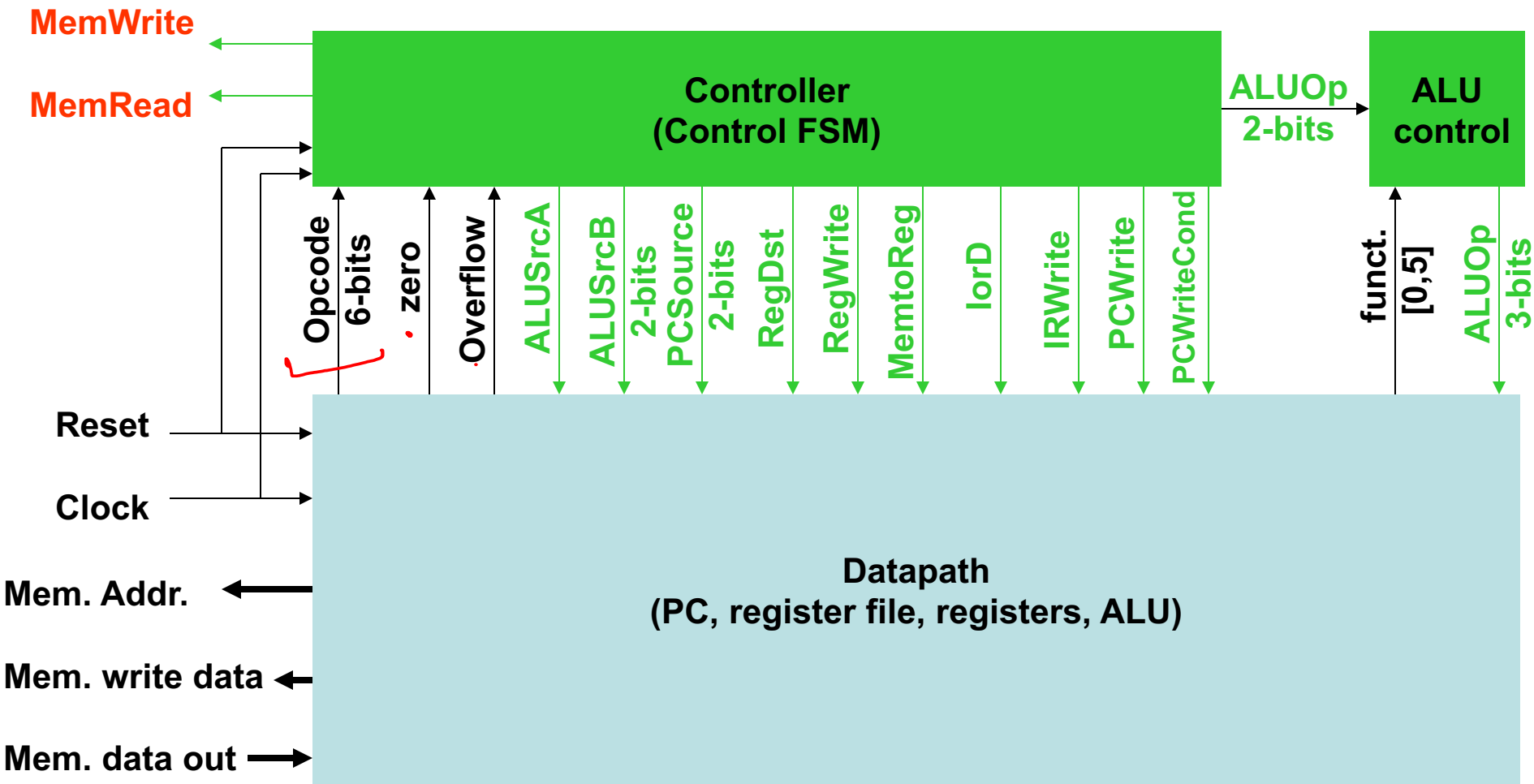
Designing the Control FSM

- 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	00010001100000100	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



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

2 ns.

T = 2 ns.

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 (A/L) — 4 — (8ns)
  - beq — 3 — (6ns)
  - j — 3 — (6ns)

R: $S_1 \to S_2 \to S_3 \to S_4$

Lw: $S_1 \to S_2 \to S_5 \to S_6 \to S_7$

Sw: $S_1 \to S_2 \to S_5 \to S_8$

Uniform distr. CPI $= 5+4+4+3+3 = \frac{19}{5} = 3.8$

Time for Prog = IC × CPI × τ.

= IC × 3.8 × 2 ns

= IC × 7.6.

Multiply.

Single cyc

= IC × CPI × τ

= IC × 1 × 8

Saving

Weighted CPI $\Leftarrow$ freq

Mix	CPI	freq
R	4	50%
LW	5	20%
Store	4	10%
branch	3	10%
Jump	3	10%

$= 4 \times .5 + 5 \times .2 + 4 \times .1 + 3 \times .1 + 3 \times .1$

$= 2 + 1 + .4 + .3 + .3$

$= 4$

$\frac{\text{Time}}{\text{Prog}} = IC \times CPI \times T = IC \times 4 \times 2 = 8 \times IC$

CPI of a Computer

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

where

CPI_k = 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%

dynamic.

mixture.

- CPI = $0.25 \times 5 + 0.10 \times 4 + 0.11 \times 3 + 0.02 \times 3 + 0.52 \times 4$

  = 4.12 for multicycle datapath

- CPI = 1.00 for single-cycle datapath

Time/Prog = Ic × 4.12 × 2

= 8.24 × Ic

Multicycle vs. Single-Cycle

$$\text{Performance ratio} = \text{Single cycle time / Multicycle time}$$

$$= \frac{(\text{CPI} \times \text{cycle time}) \text{ for single-cycle}}{(\text{CPI} \times \text{cycle time}) \text{ for multicycle}}$$

$$= \frac{1.00 \times 8\text{ns}}{4.12 \times 2\text{ns}} = 0.97$$

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

Reuse of resource

Single Cycle

CPI = 1

τ ↑

Multicycle

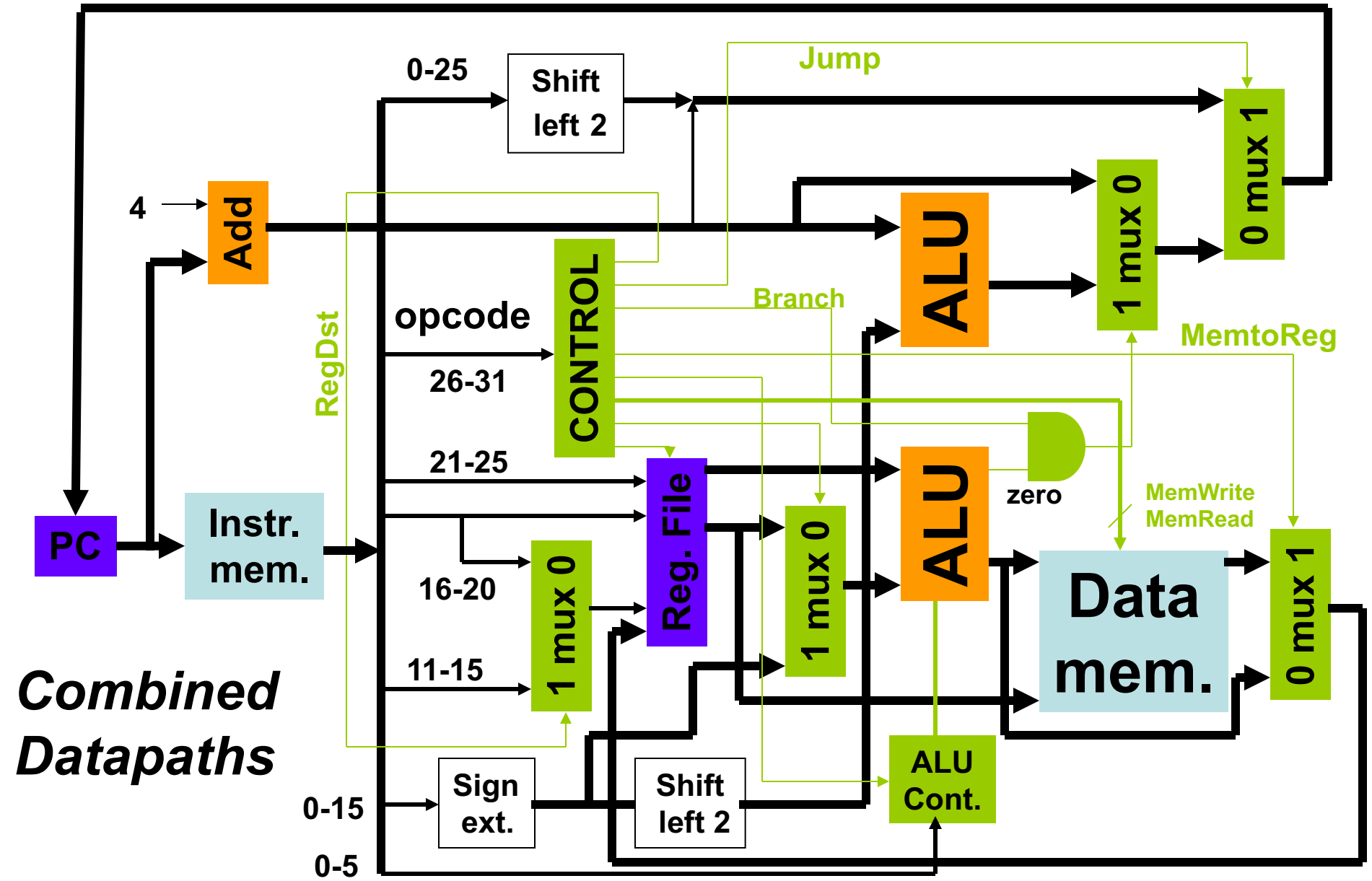
τ ↓

2 ns

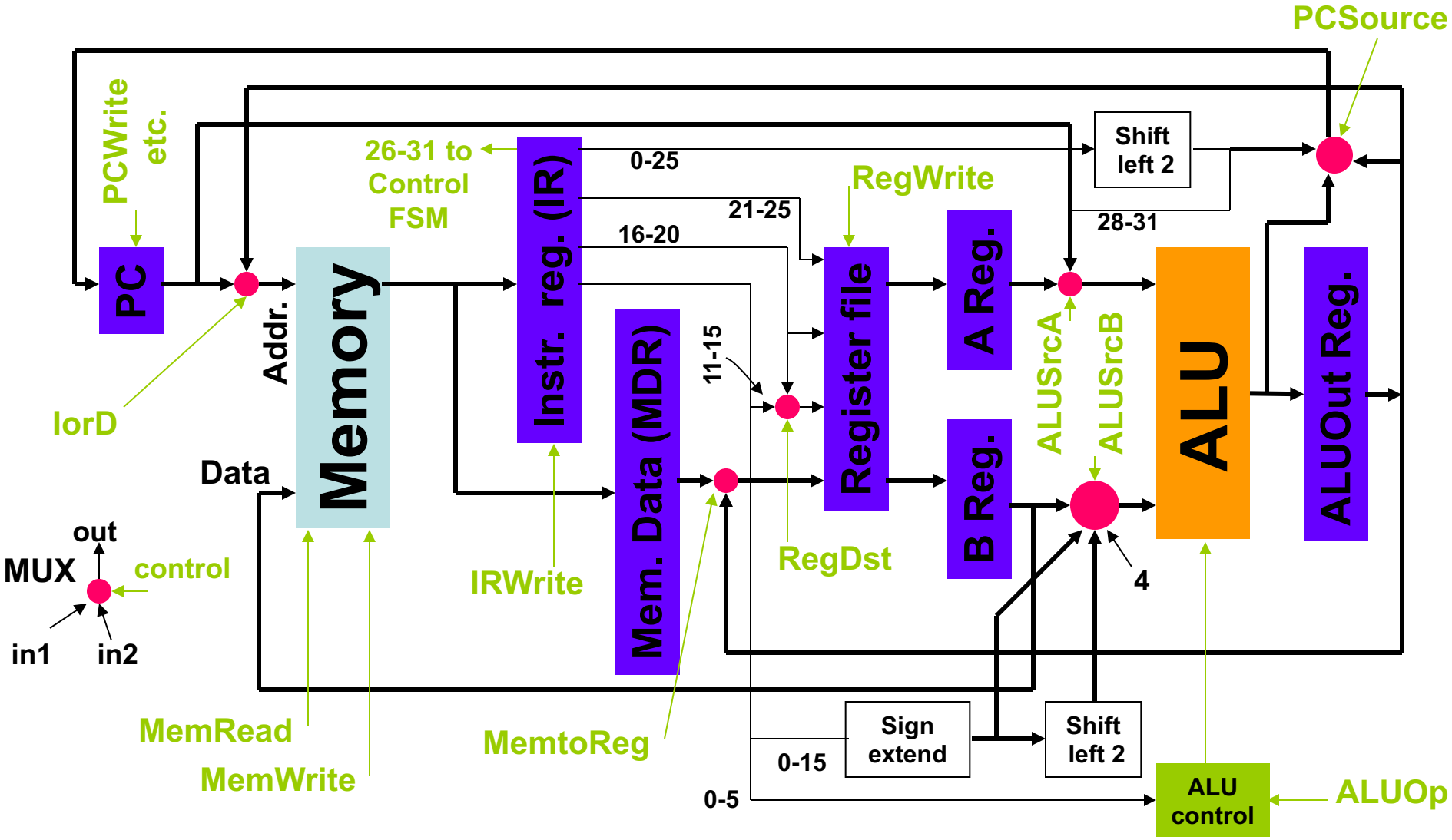
CPI ↑. CPI > 1.

Way to go beyond this performance.

Time/Prog. = IC x CPI x τ

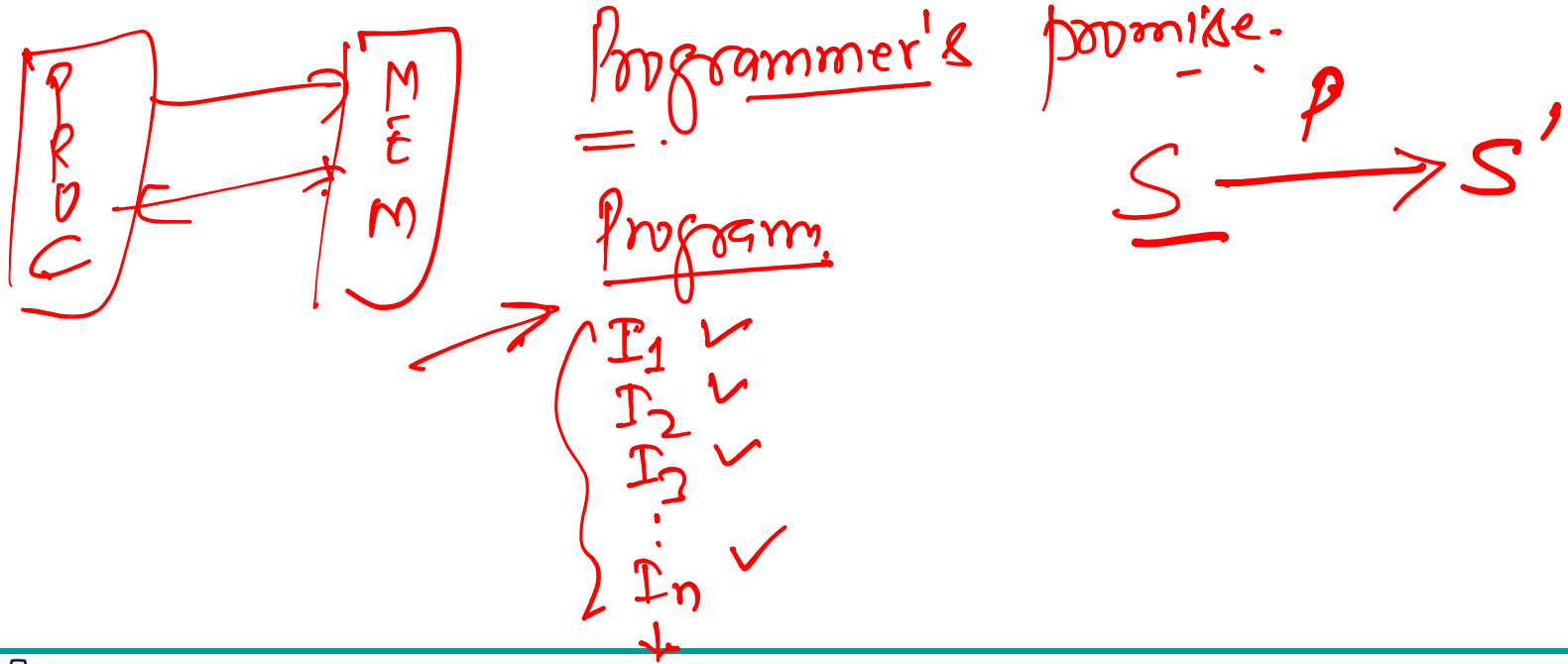

Combined Datapaths
0-25
Shift left 2
Jump
4
Add
ALU
1 mux 0
0 mux 1
RegDst
opcode
26-31
CONTROL
Branch
MemtoReg
21-25
Reg. File
zero
MemWrite
MemRead
PC
Instr. mem.
16-20
1 mux 0
ALU
Data mem.
0 mux 1
11-15
1 mux 0
0-15
Sign ext.
Shift left 2
ALU Cont.
0-5

Multicycle Datapath



ILP: Instruction Level Parallelism

- Single-cycle and multi-cycle datapaths execute one instruction at a time.
- How can we get better performance?

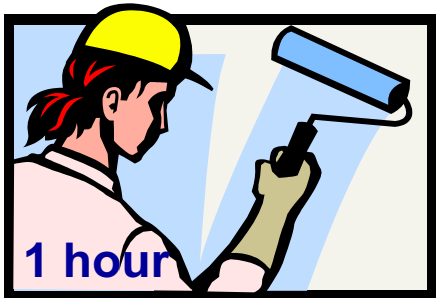


Traffic Flow

Way Ahead. . . .


4·60 m
GIVE WAY
HEAVY VEHICLE
BRIDGE LIMITS
SPEED 20 km/h
TARAMAKAU
RIVER
alamy stock photo
HEWNJW
www.alamy.com

Automobile Team Assembly



1 car assembled every four hours

6 cars per day

180 cars per month

2,040 cars per year

Automobile Assembly Line

First car assembled in 4 hours (pipeline latency)

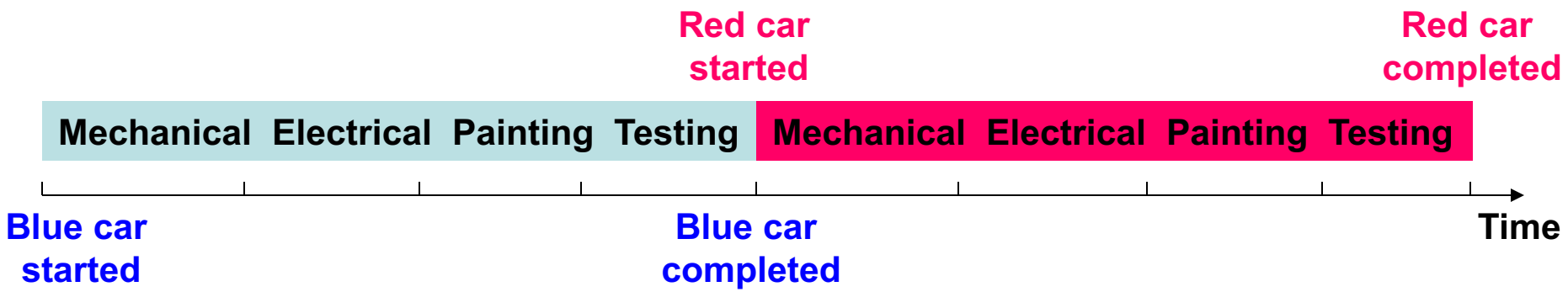
thereafter 1 car per hour

21 cars on first day, thereafter 24 cars per day

717 cars per month

8,637 cars per year

Throughput: Team Assembly

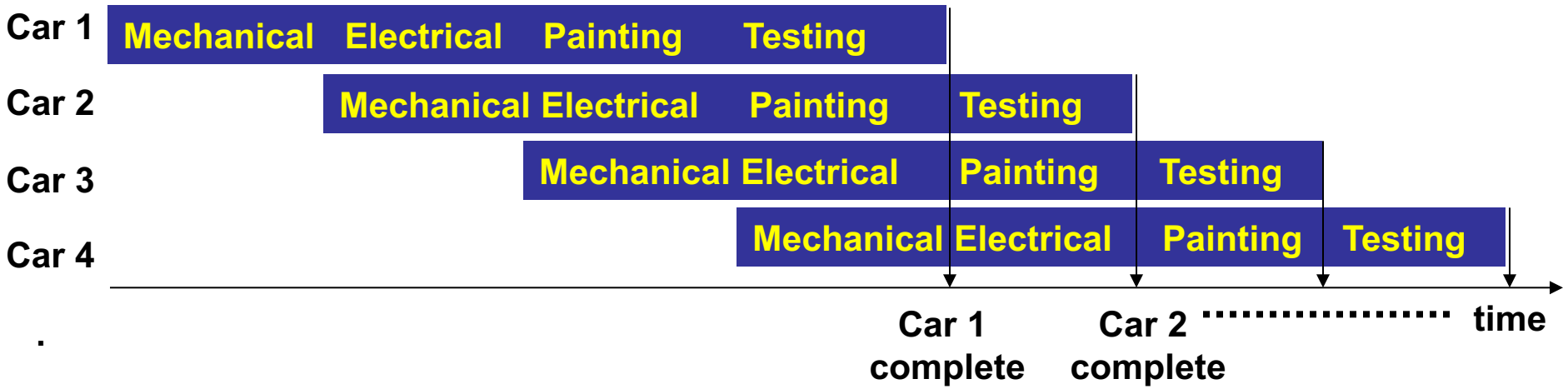


Time of assembling one car = n hours

where n is the number of nearly equal subtasks, each requiring 1 unit of time

Throughput = $1/n$ cars per unit time

Throughput: Assembly Line



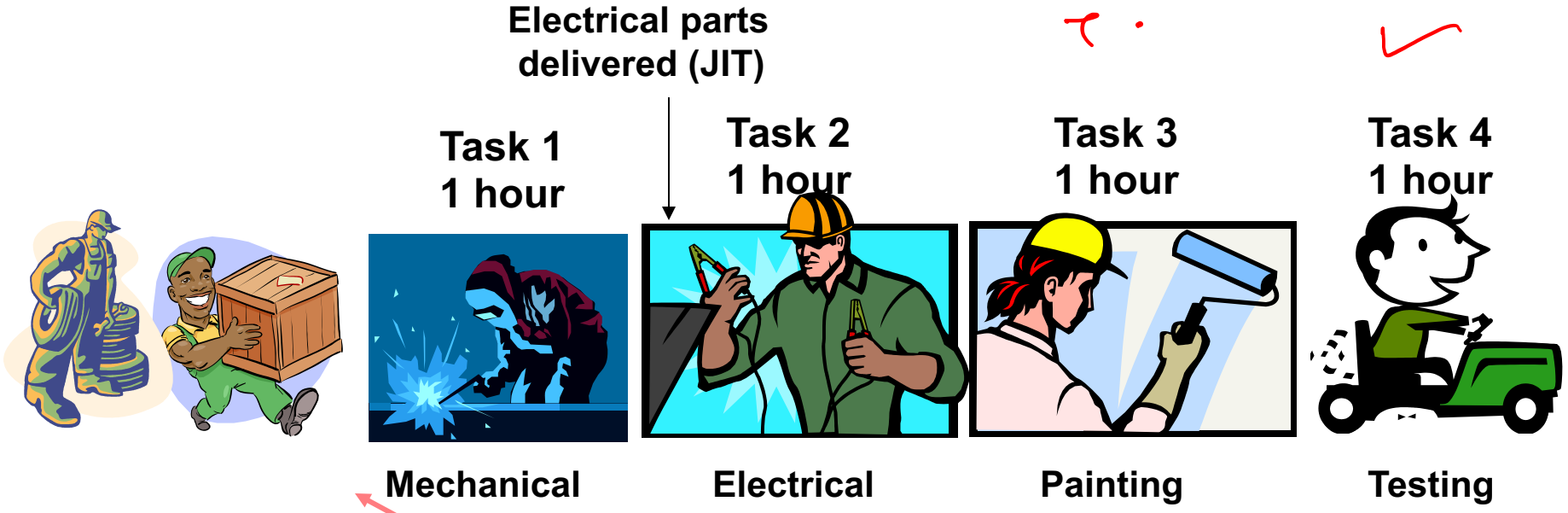
Time to complete first car = n time units (latency)

Cars completed in time T = $T - n + 1$

Throughput = $1 - (n - 1)/T$ car per unit time

$$\frac{\text{Throughput (assembly line)}}{\text{Throughput (team assembly)}} = \frac{1 - (n-1)/T}{1/n} = n - \frac{n(n-1)}{T} \rightarrow n \text{ as } T \rightarrow \infty$$

Some Features of Assembly Line



Instruction

Subtask

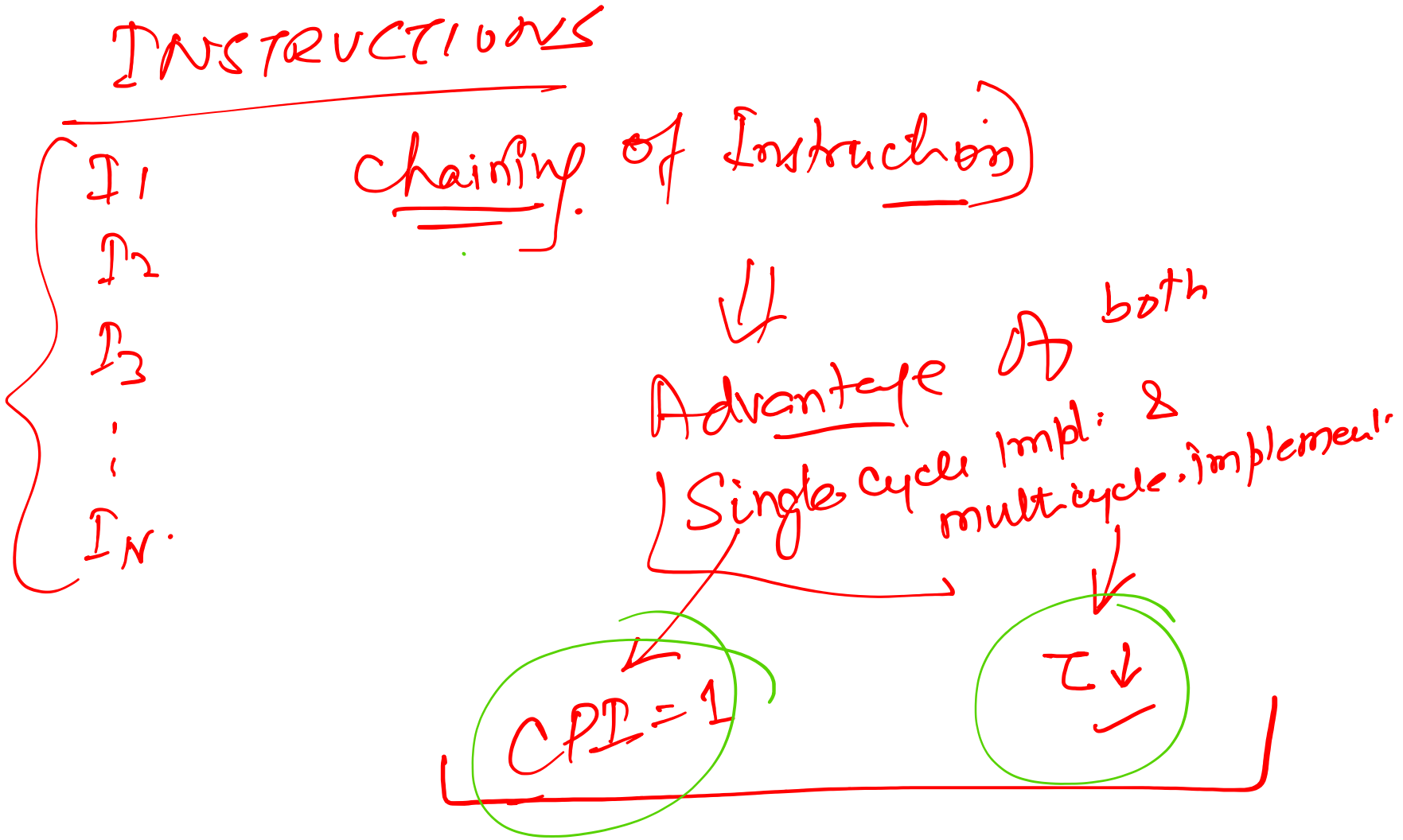
- Fetch instruction
- Read operands
- Execute / compute the address of memory
- Access memory
- Write back (update system state)

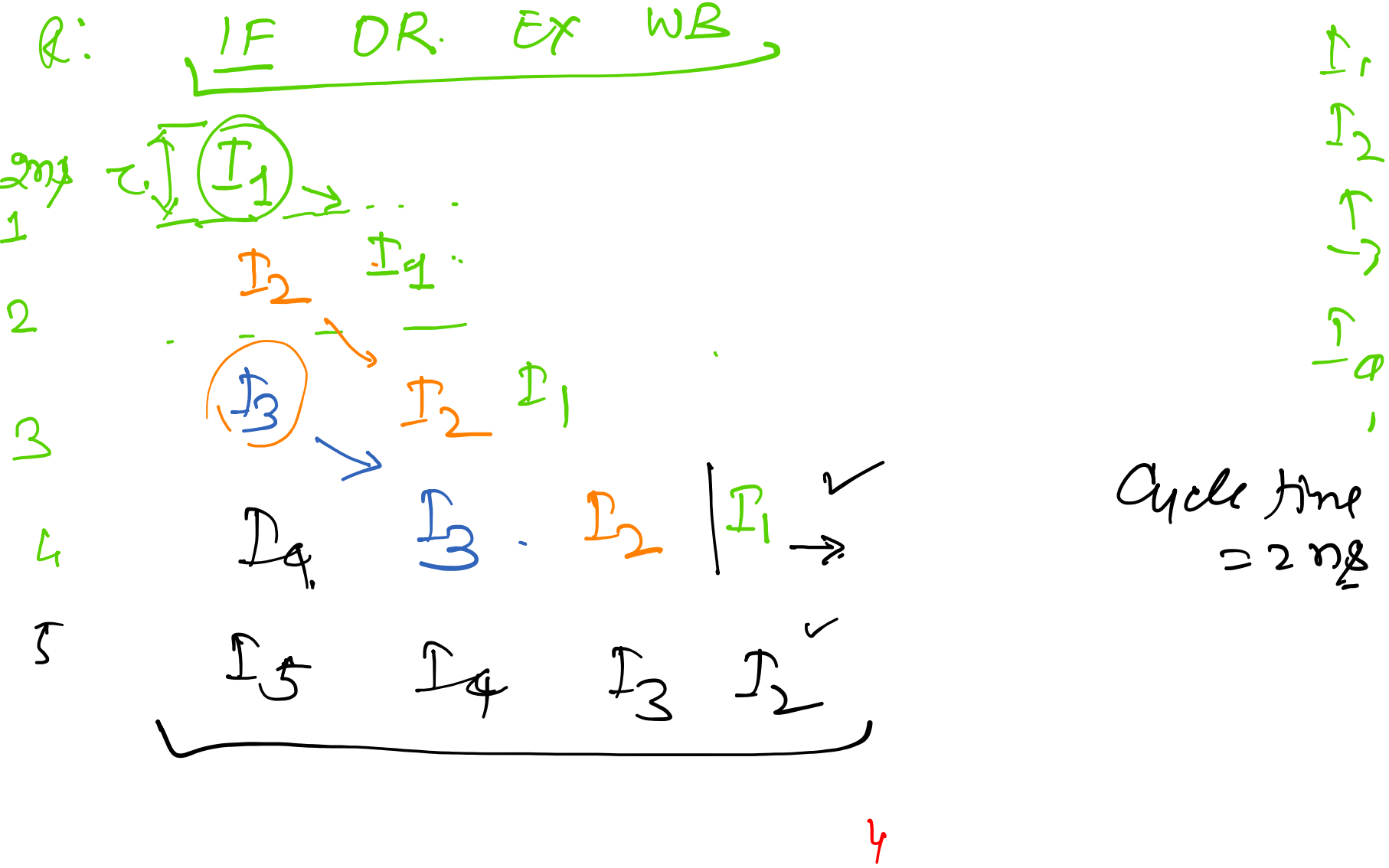
✓ Breach of programmer's promise

I_1 I_2 ... I_n

Update exactly in the same sequence.

{ILLUSION}


INSTRUCTIONS
I1
I2
I3
IN.
Chaining of Instructions
Advantage of both
Single cycle impl: &
multi-cycle. implement.
CPI = 1
T ↓


Q: IF OR EX WB
2ns
1
2
3
4
5
I_1
I_2
I_3
I_4
I_5
Cycle time = 2 ns

1. Independence of various Cars:

INSTRUCTION

2. Sequence — Change in flow

Assumption

Both do not meet

$I_1 : a = b + c$

$I_2 : d = a + e$

I_1
I_2
I_3
I_4
I_5
I_6

How often these assumptions fail ?

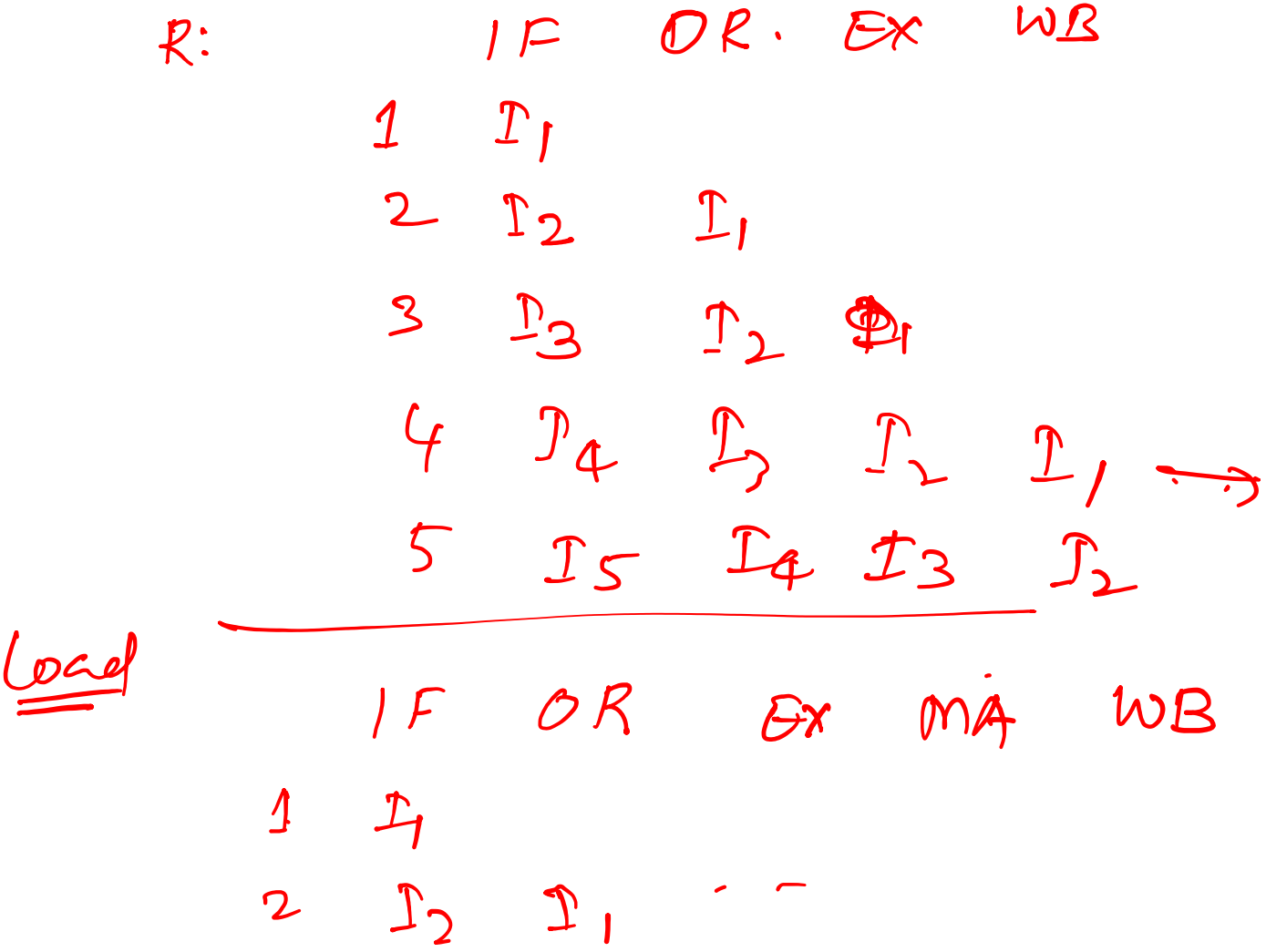
✓ NOT OFTEN

$a = b + c$
$d = p + q$
$r = s + t$

Implement — assuming that these assumptions are always respected.

⊗ Should have a way to detect the failure

⊗ Mechanism to correct.


R:
IF OR EX WB
1 I1
2 I2 I1
3 I3 I2 I1
4 I4 I3 I2 I1
5 I5 I4 I3 I2
Load
IF OR EX MA WB
1 I1
2 I2 I1

R: IF - OR - EX - WB

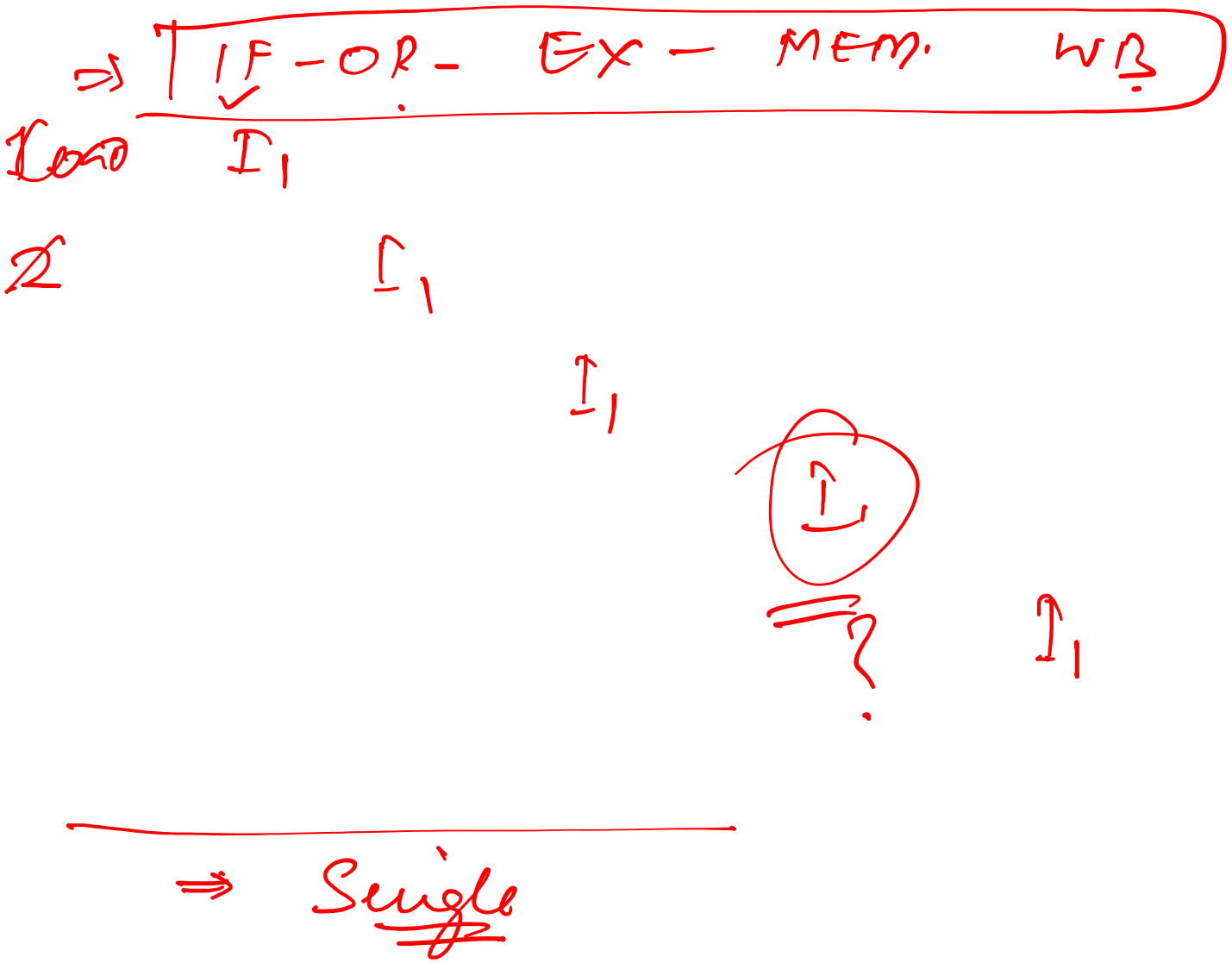
L: IF - OR - EX - MA - WB

S IF - OR - EX - MA

B: IF - OR - EX

J. IF - OR - EX

IF - OR - EX - MA - WB.


IF - OR - EX - MEM. WB
Load
I1
I1
I1
I1
?
Single

Thank You