RISC Architecture:

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

Computer Organization & Architecture

Lecture 13 (16 April 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

Instruction Format

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

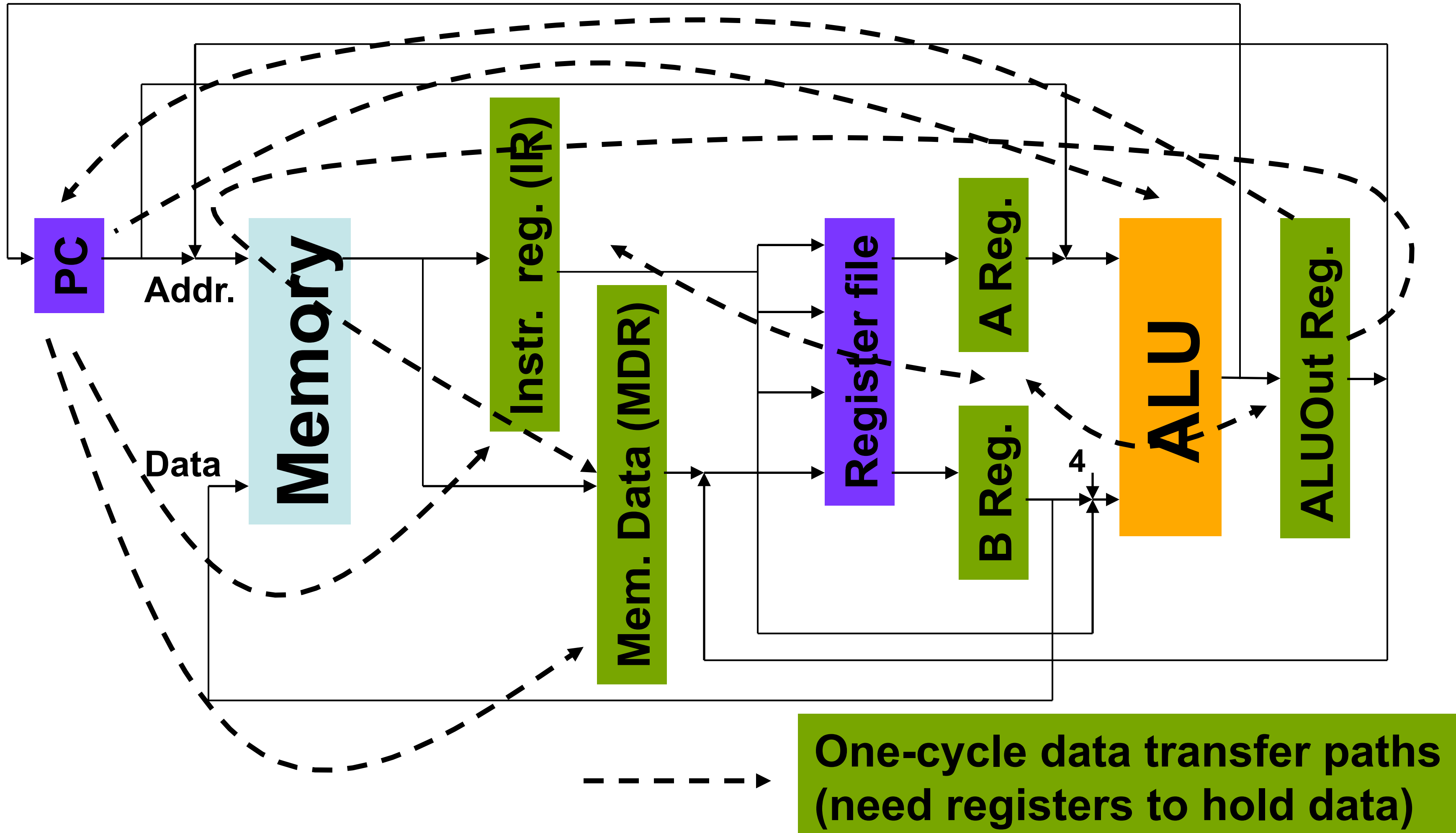
R	op	rs1	rs2	rd	shmt	funct
I	op	rs1	rd	16 bit address/data		
J	op	26 bit address				

- rely on compiler to achieve performance

How Fast Can the Clock Be?

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

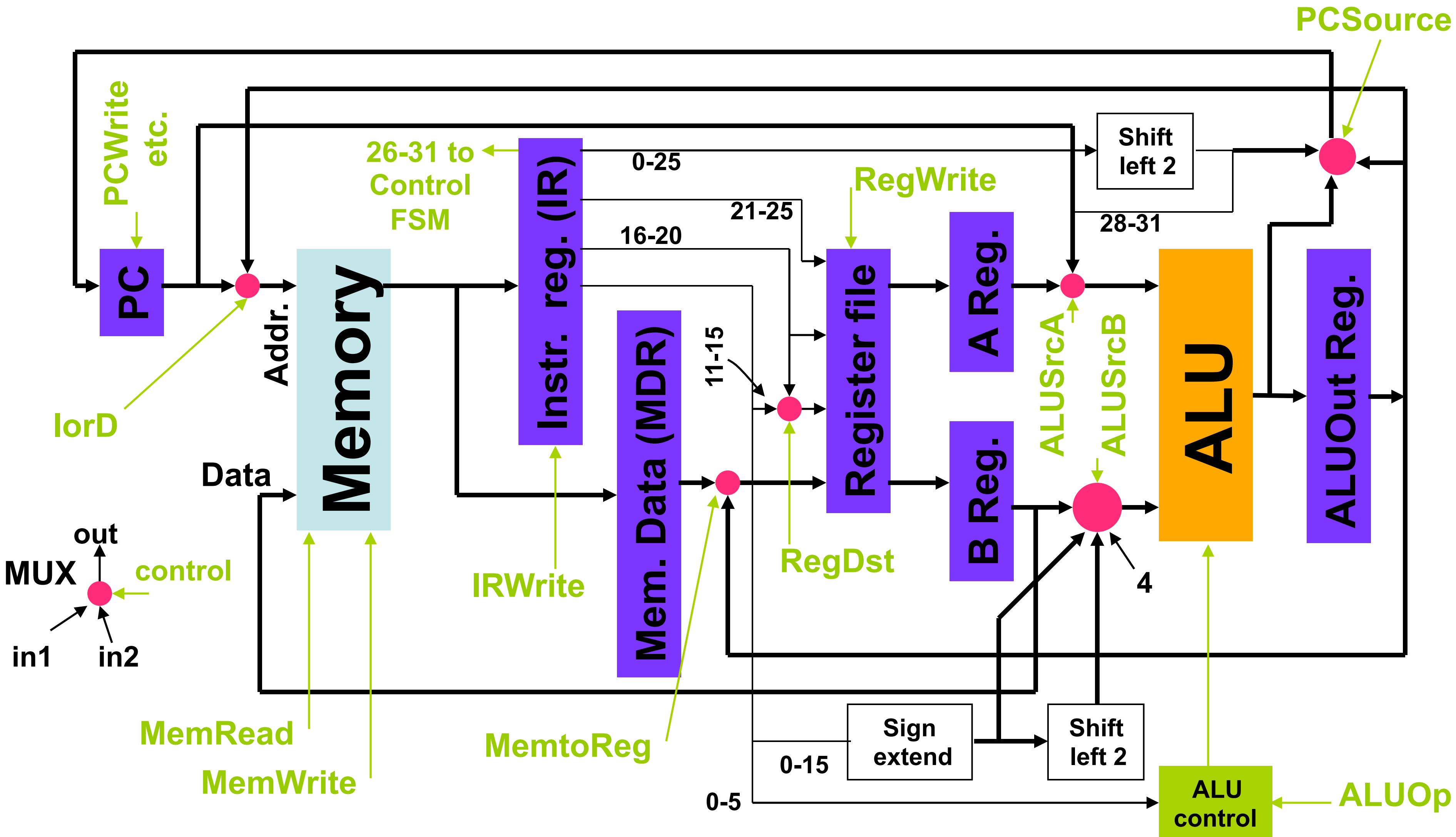
Multicycle Datapath



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 ← 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)

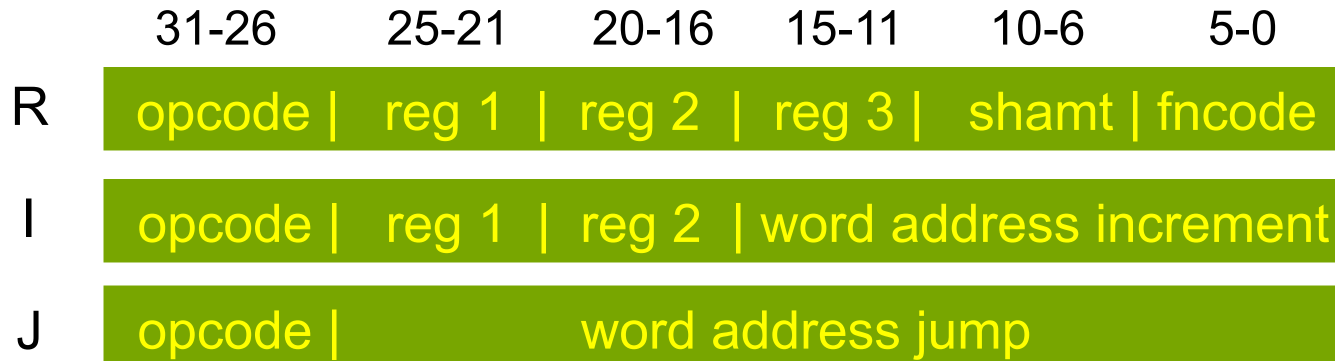
- Read instruction into IR, M[PC] → IR
  - Control signals used:

» 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

zero(ALU)

PCWriteCond

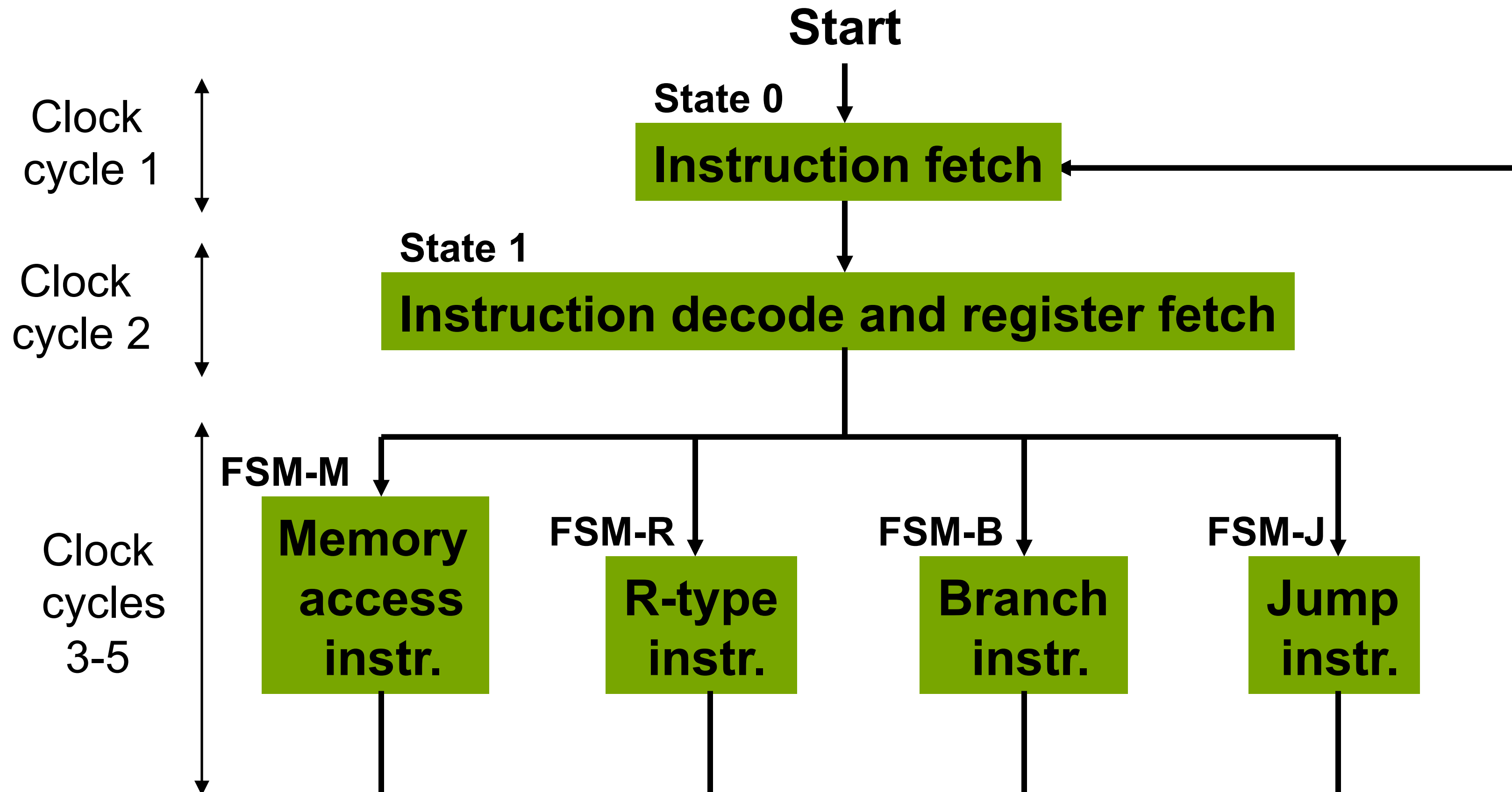
PCWrite

PCWrite etc.

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

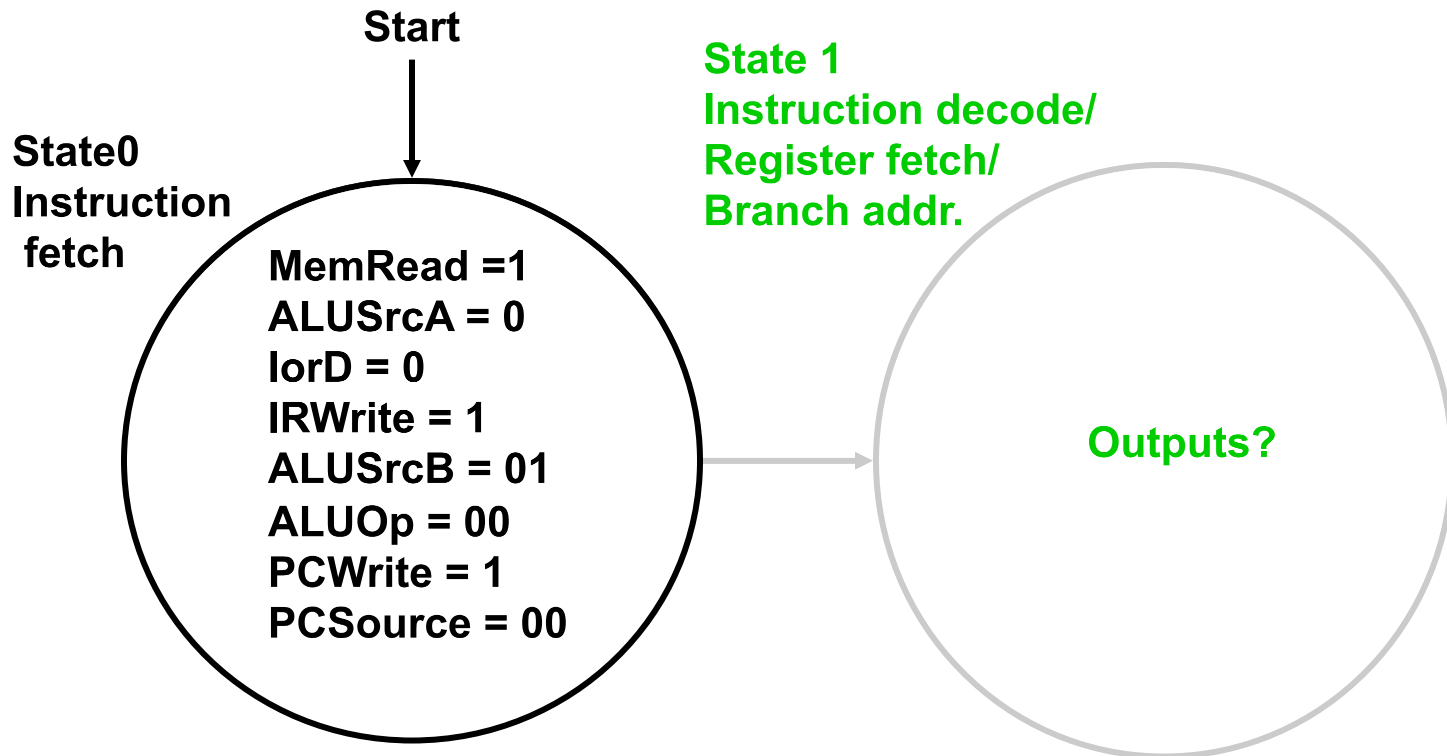
Control: Finite State Machine



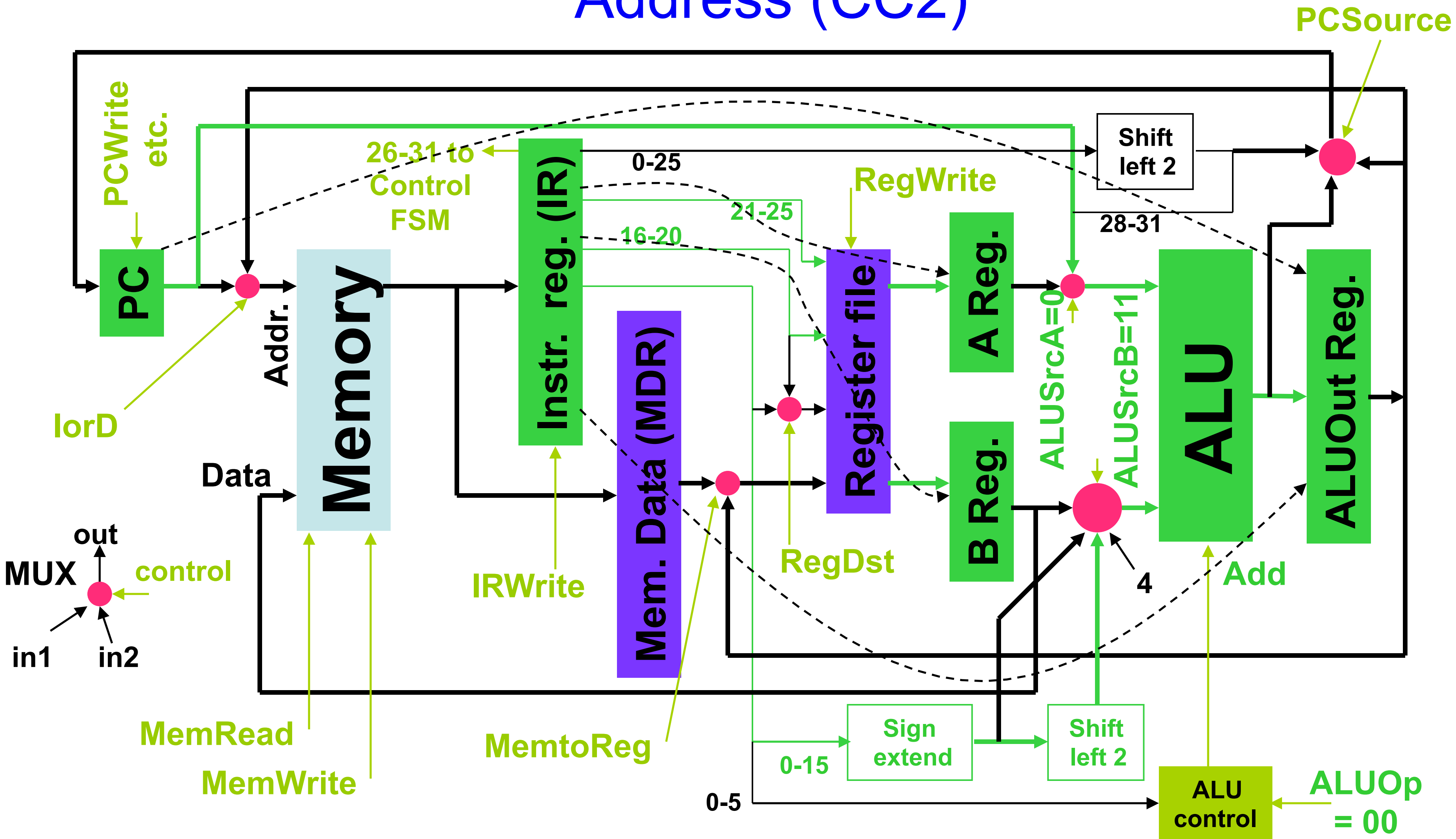
State 0: Instruction Fetch (CC1)

PCSource=00
PCWrite etc=1
26-31 to Control FSM
0-25
Shift left 2
RegWrite
21-25
28-31
16-20
PC
Addr.
Memory
Instr. reg. (IR)
Register file
A Reg.
ALUSrcA=0
ALUSrcB=01
ALU
ALUOut Reg.
IorD=0
Data
Mem. Data (MDR)
B Reg.
out
MUX
control
in1
in2
RegDst
4
Add
IRWrite =1
MemRead = 1
MemWrite
MemtoReg
Sign extend
Shift left 2
0-15
0-5
ALU control
ALUOp =00

State 0 Control FSM Outputs



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



State 1 Control FSM Outputs

State0
Instruction
fetch
(IF)

Start

MemRead =1
ALUSrcA = 0
IorD = 0
IRWrite = 1
ALUSrcB = 01
ALUOp = 00
PCWrite = 1
PCSource = 00

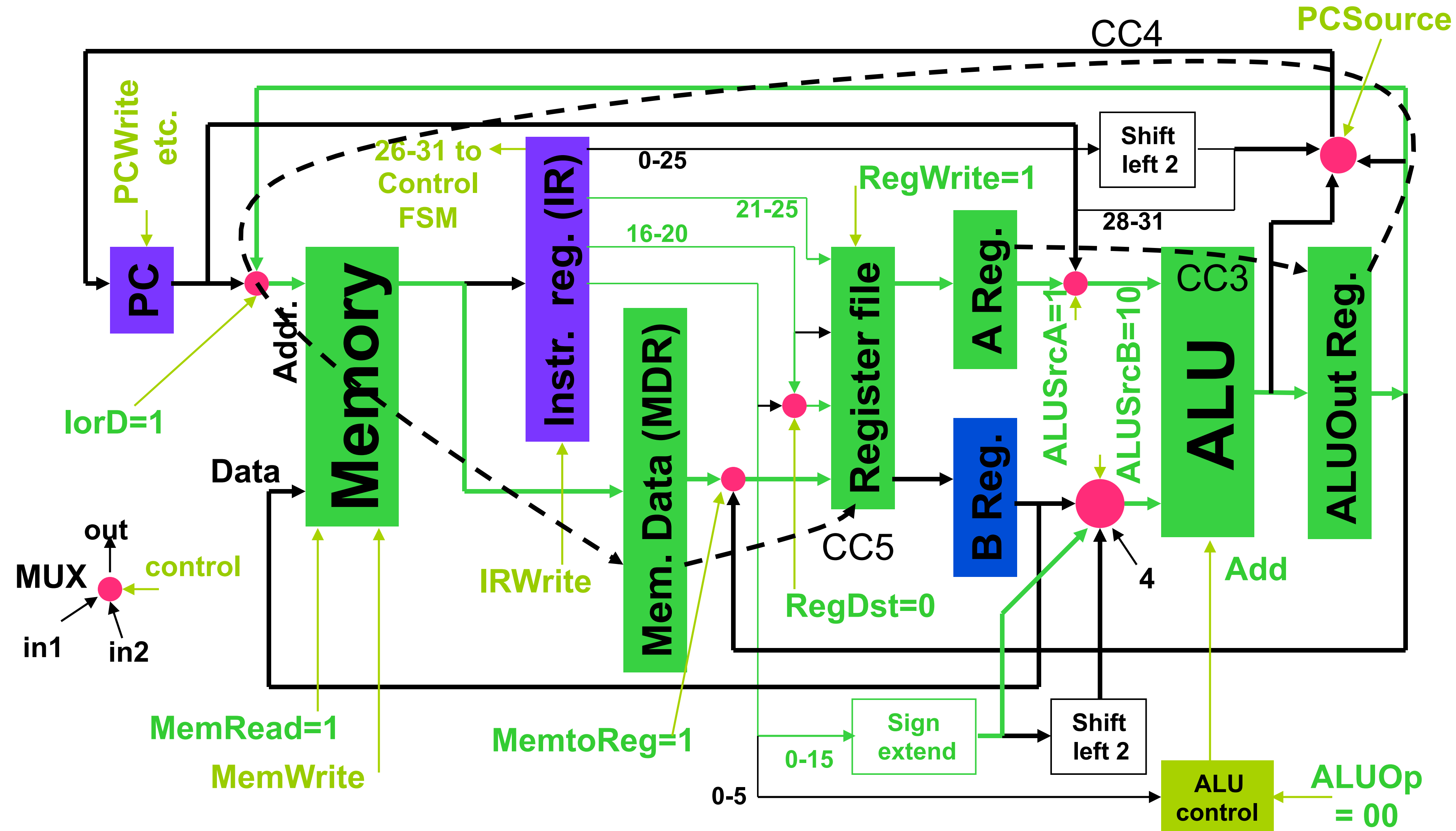
State 1
Instruction decode (ID) /
Register fetch /
Branch addr.

ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00

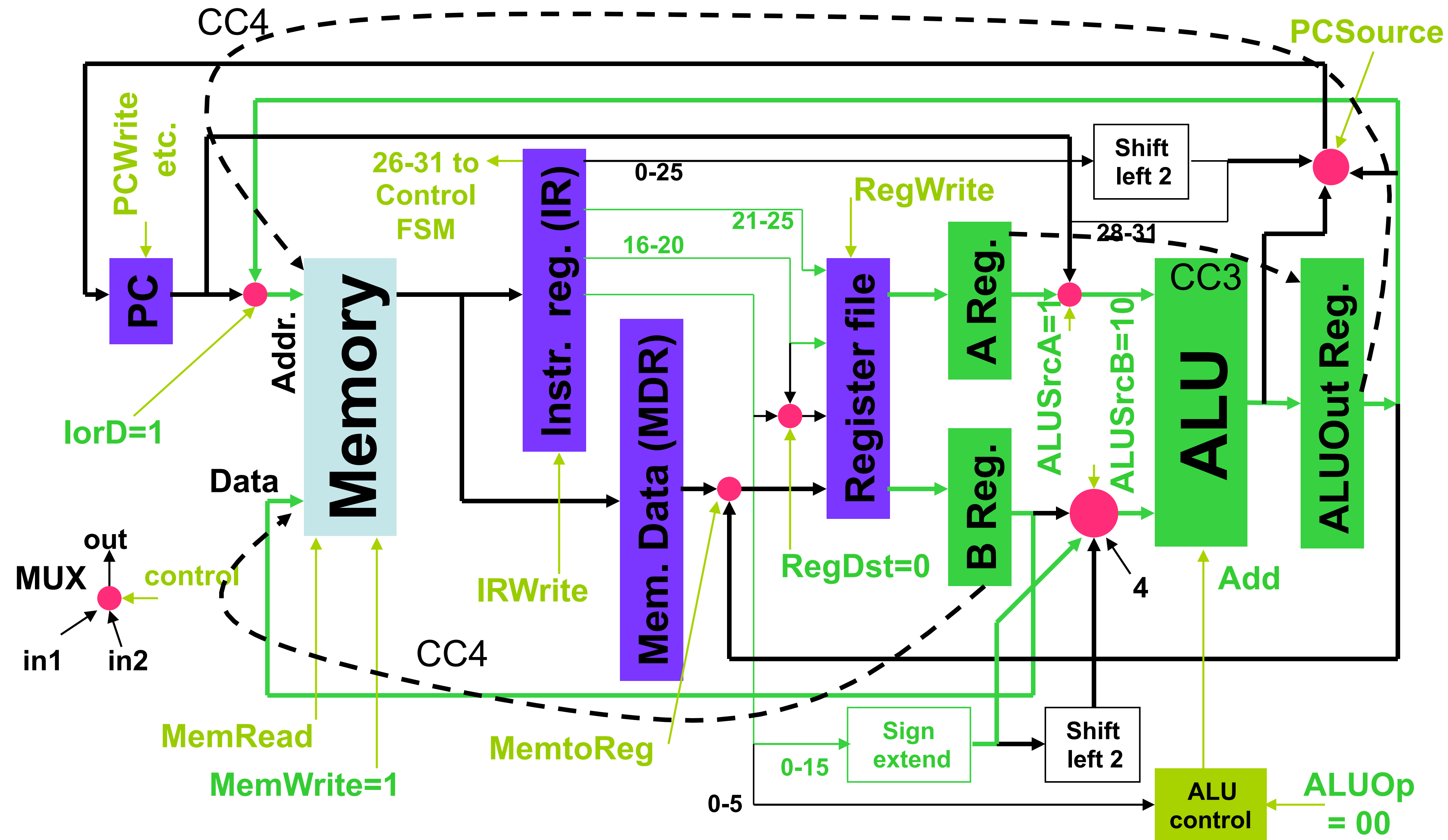
Opcode = lw, sw
Opcode = R-type
Opcode = BEQ
Opcode = J-type

FSM-M
FSM-R
FSM-B
FSM-J

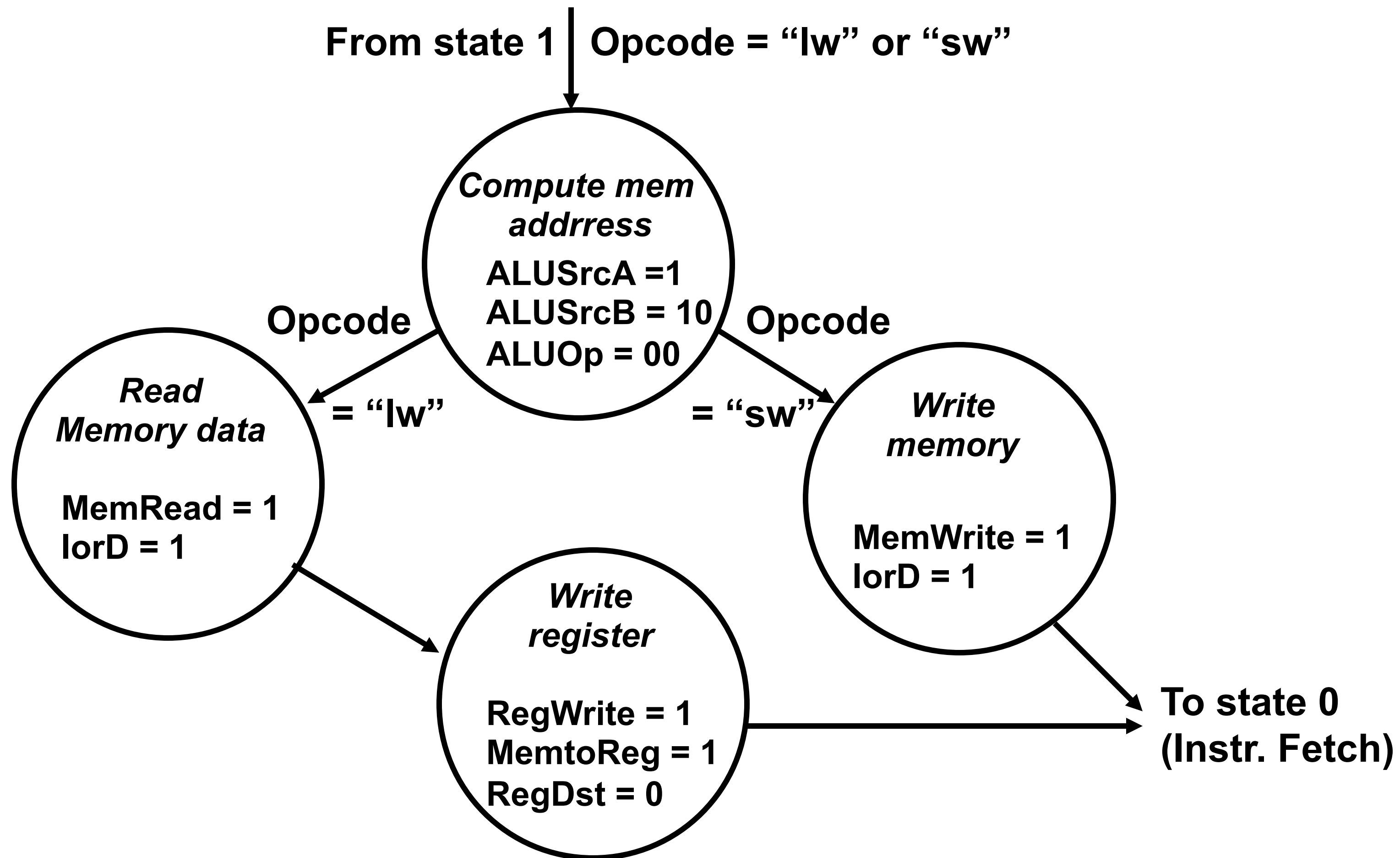
State 1 (Opcode = lw) → FSM-M (CC3-5)



State 1 (Opcode= sw)→FSM-M (CC3-4)



FSM-M (Memory Access)

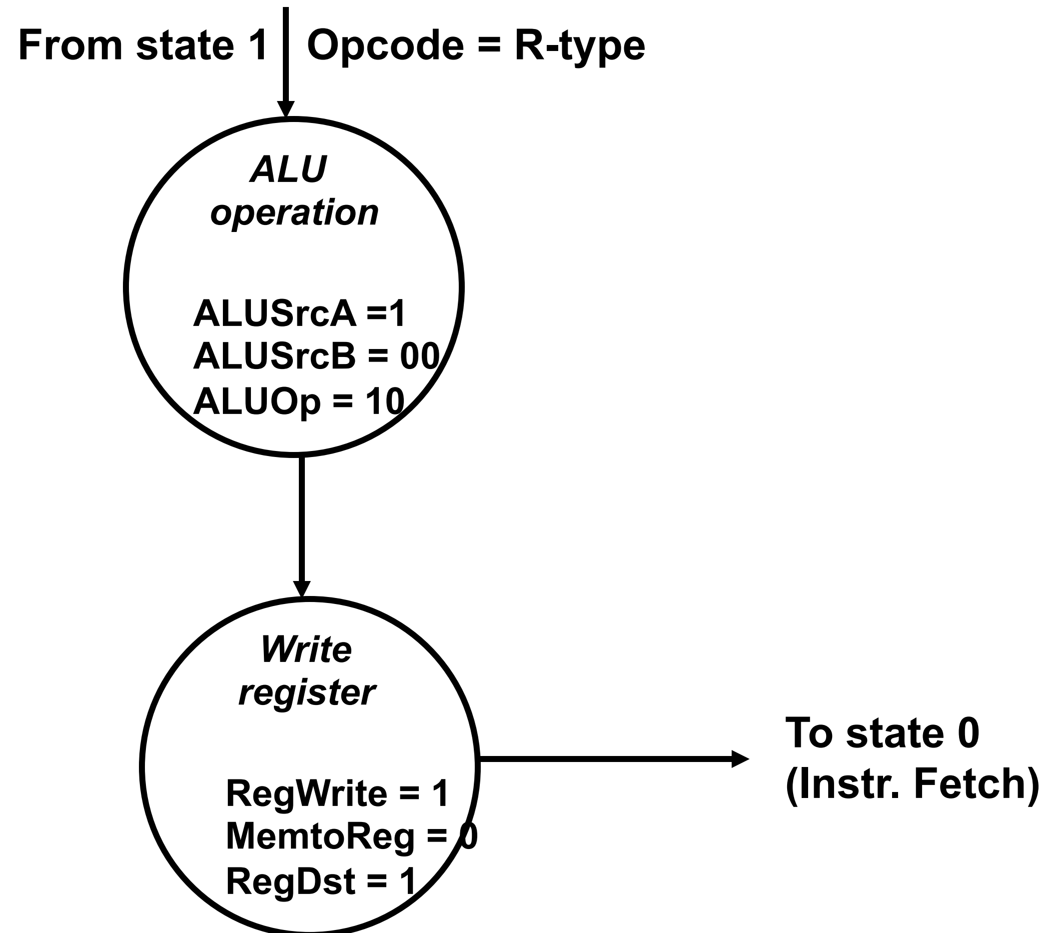


State 1(Opcode=R-type)→FSM-R (CC3-4)

PCSource
PCWrite
etc.
26-31 to
Control
FSM
0-25
Shift
left 2
RegWrite
21-25
16-20
28-31
PC
Memory
Addr.
Data
Instr. reg. (IR)
Register file
A Reg.
B Reg.
ALUSrcA=1
ALUSrcB=00
ALU
ALUOut Reg.
CC3
CC4
IorD
11-15
Mem. Data (MDR)
RegDst=0
4
"funct. code"
out
MUX
control
in1
in2
IRWrite
MemRead
MemWrite
MemtoReg=0
0-15
Sign
extend
Shift
left 2
0-5
ALU
control
ALUOp
= 10

CADSL

FSM-R (R-type Instruction)

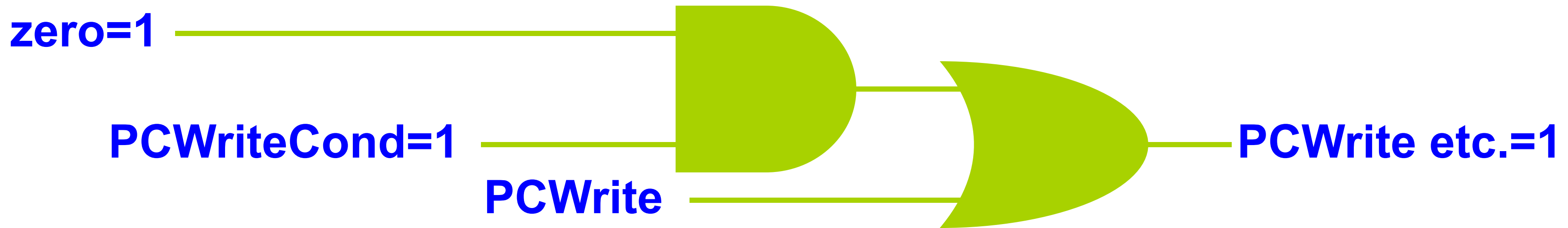


State 1 (Opcode = beq) → FSM-B (CC3)

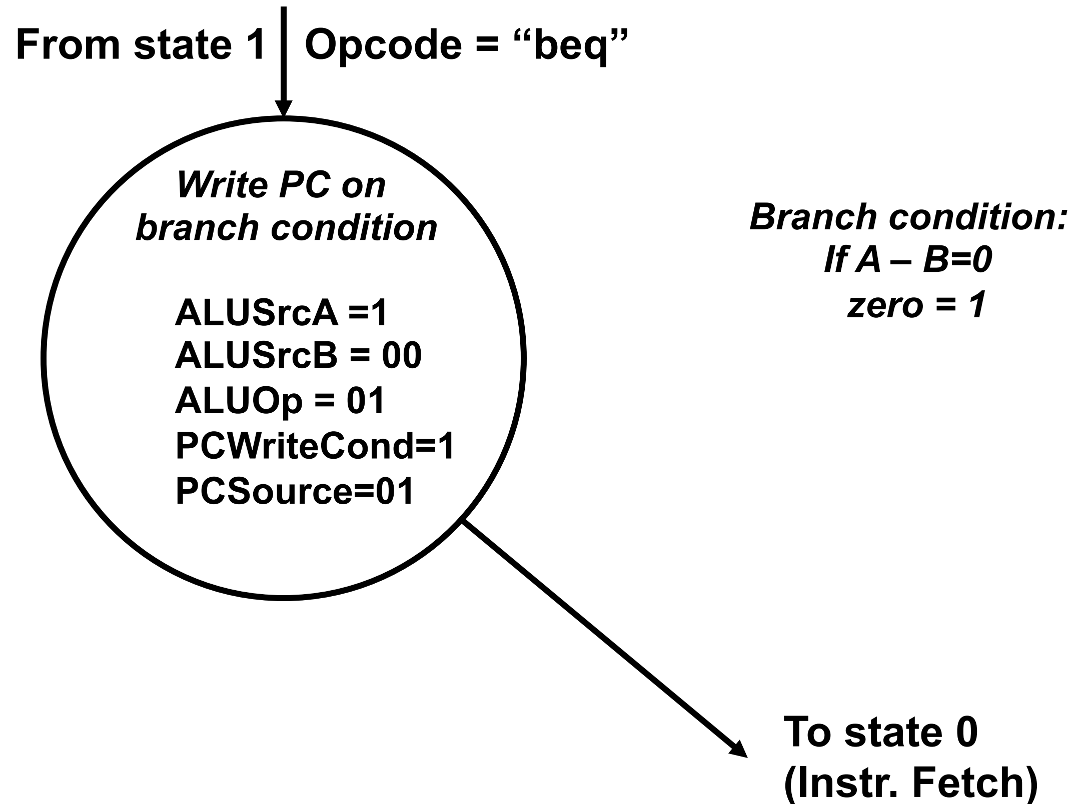
If(zero)
PCSource
01
PCWrite
etc.=1
26-31 to
Control
FSM
0-25
21-25
16-20
RegWrite
Shift
left 2
28-31
PC
Addr.
Memory
Instr. reg. (IR)
Register file
A Reg.
B Reg.
ALUSrcA=1
ALUSrcB=00
ALU
ALUOut Reg.
zero
CC3
IorD
Data
Mem. Data (MDR)
11-15
MUX
out
control
in1
in2
IRWrite
RegDst
4
subtract
MemRead
MemWrite
MemtoReg
Sign
extend
Shift
left 2
0-15
0-5
ALU
control
ALUOp
= 01

CADSL

Write PC on “zero”



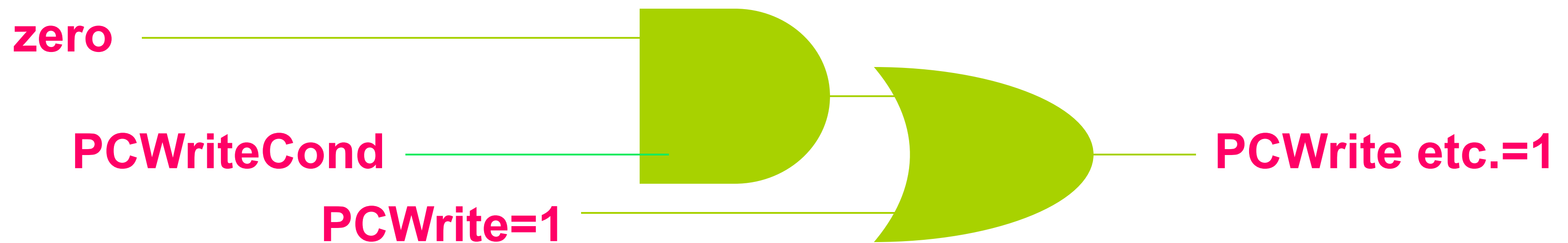
FSM-B (Branch)



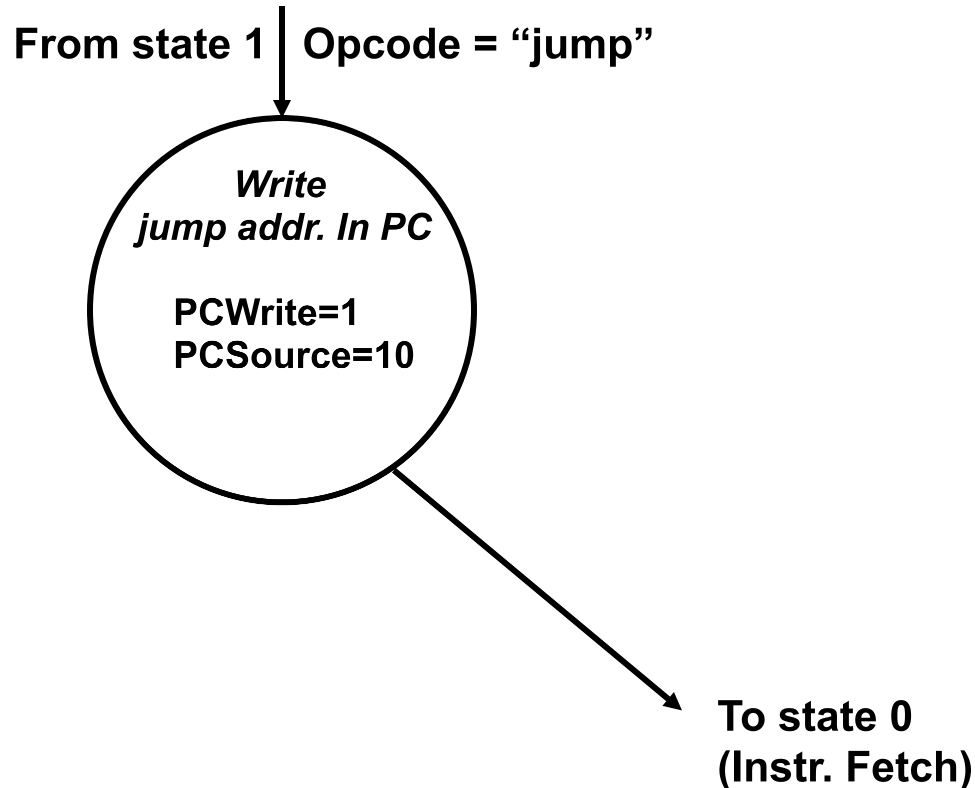
State 1 (Opcode = j) → FSM-J (CC3)

CC3
PCSource
10
PCWrite
etc.
26-31 to
Control
FSM
0-25
Shift
left 2
RegWrite
21-25
28-31
16-20
PC
Addr.
Memory
Instr. reg. (IR)
Mem. Data (MDR)
Register file
A Reg.
B Reg.
ALUSrcA
ALUSrcB
ALU
zero
ALUOut Reg.
IorD
11-15
Dat
a
out
MUX
control
in1
in2
IRWrite
RegDst
4
MemRead
MemWrite
MemtoReg
Sign
extend
0-15
Shift
left 2
ALU
control
ALUOp
0-5

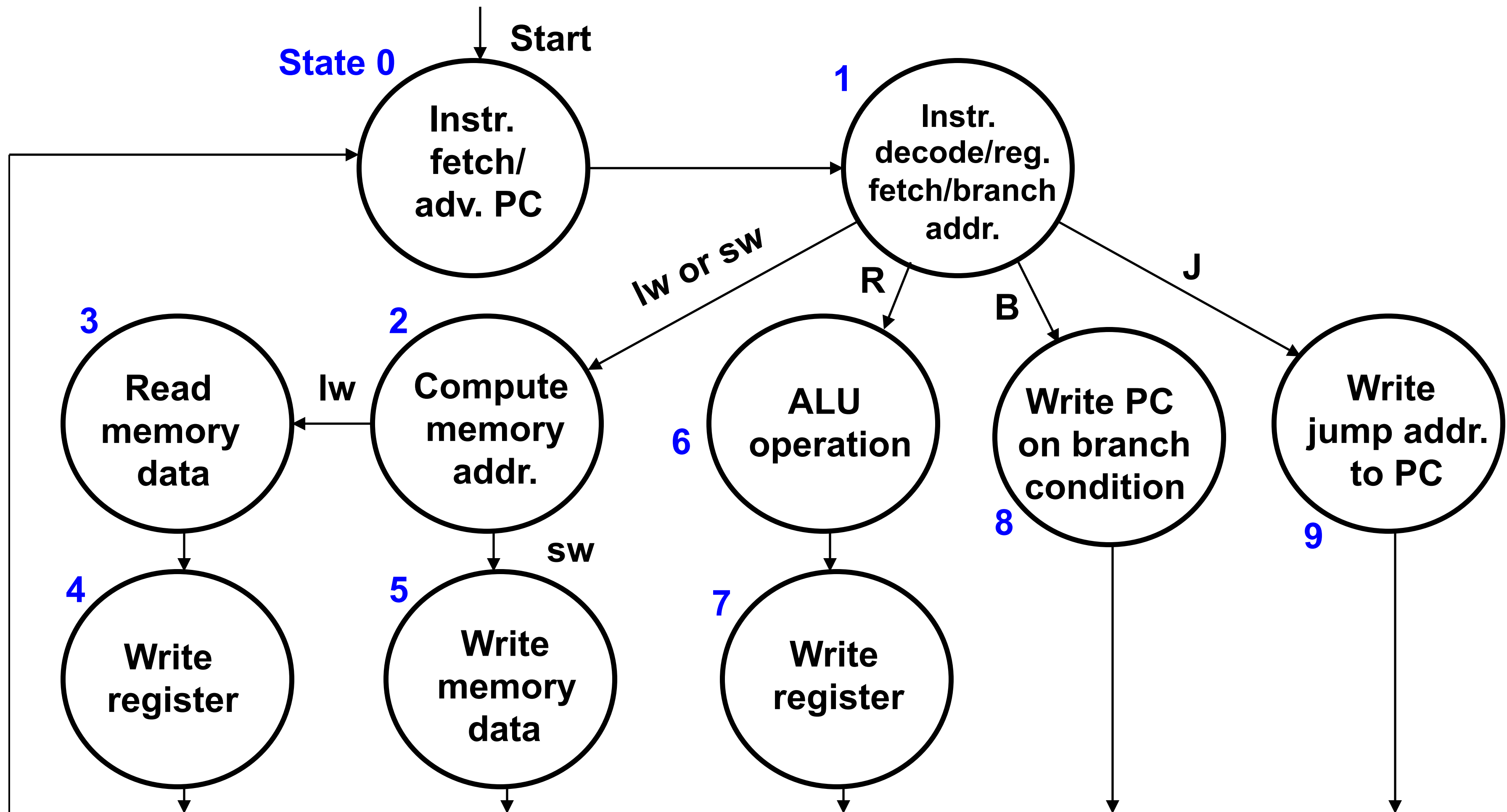
Write PC



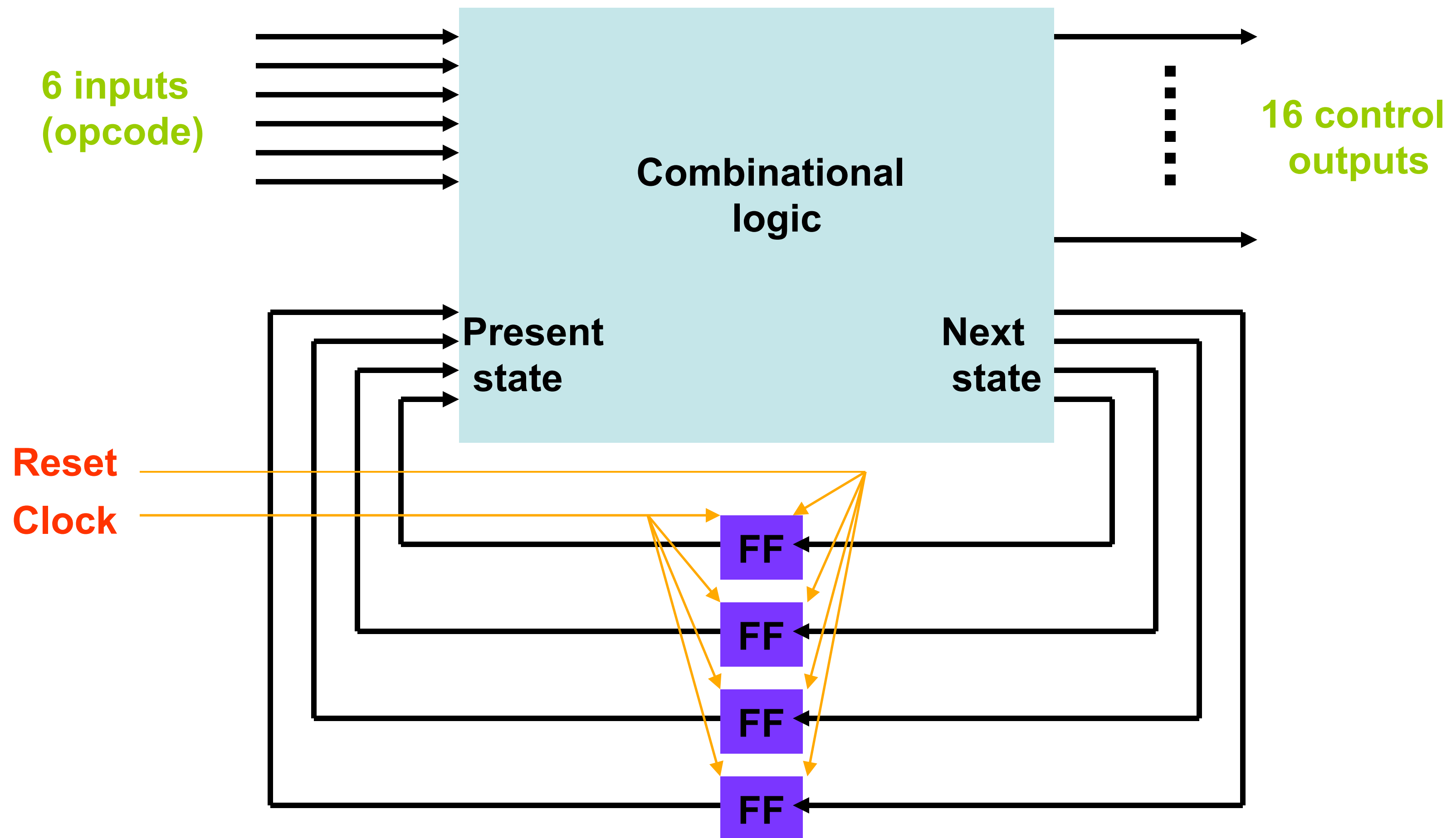
FSM-J (Jump)



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
. . . .	. . . .	. . . .	

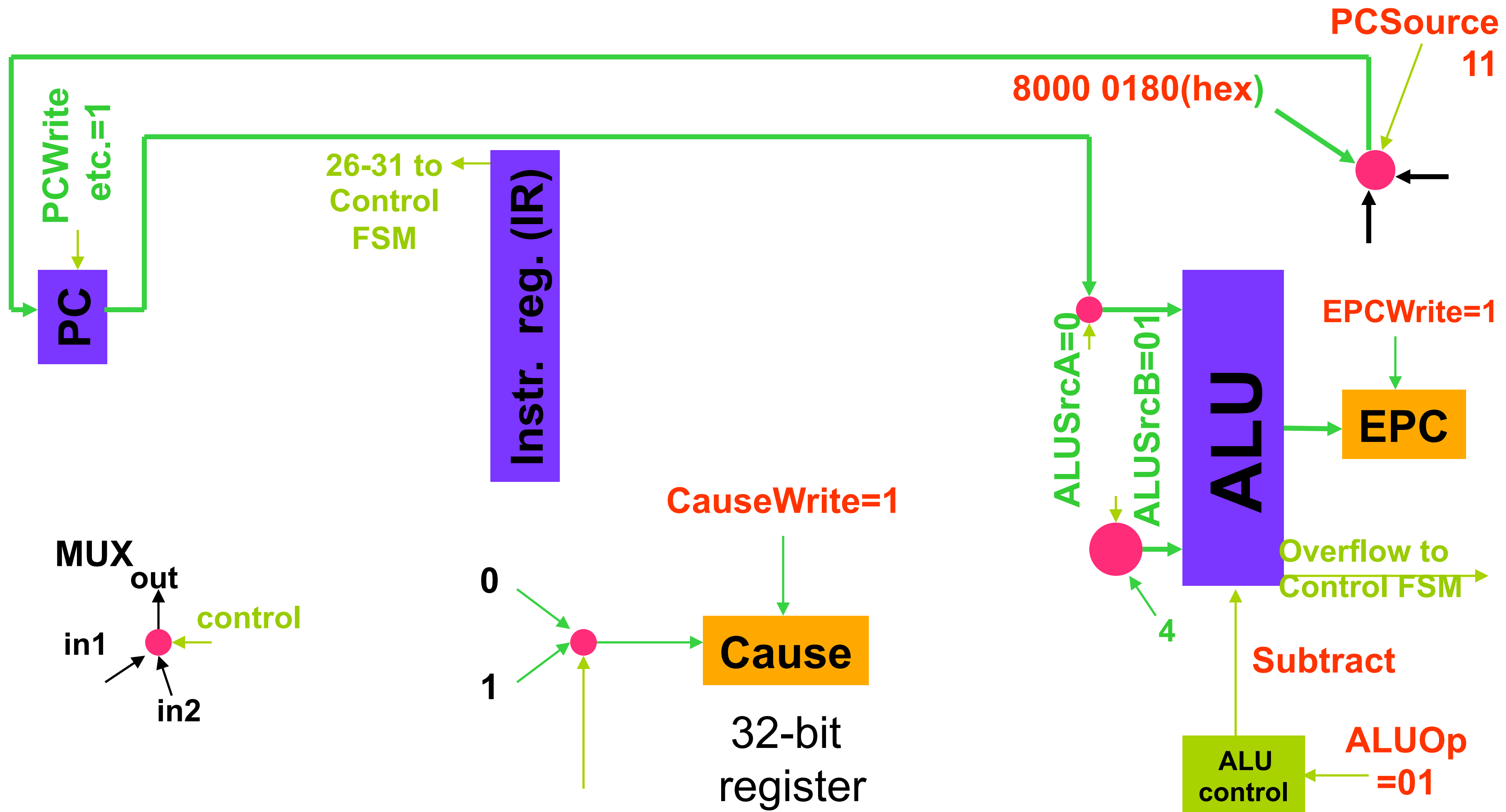
- Synthesize a logic circuit from the truth table.
- Connect four flip-flops between the next state outputs and present state inputs.

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

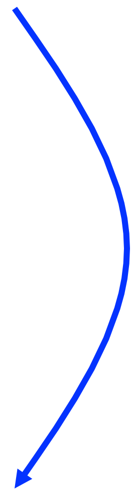


How Long Does It Take? Again

- 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

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

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%
- CPI = $0.25\times5 + 0.10\times4 + 0.11\times3 + 0.02\times3 + 0.52\times4$

  = 4.12 for multicycle datapath
- CPI = 1.00 for single-cycle datapath

Multicycle vs. Single-Cycle

Performance ratio = 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.

Thank You