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 14 (19 April 2013)

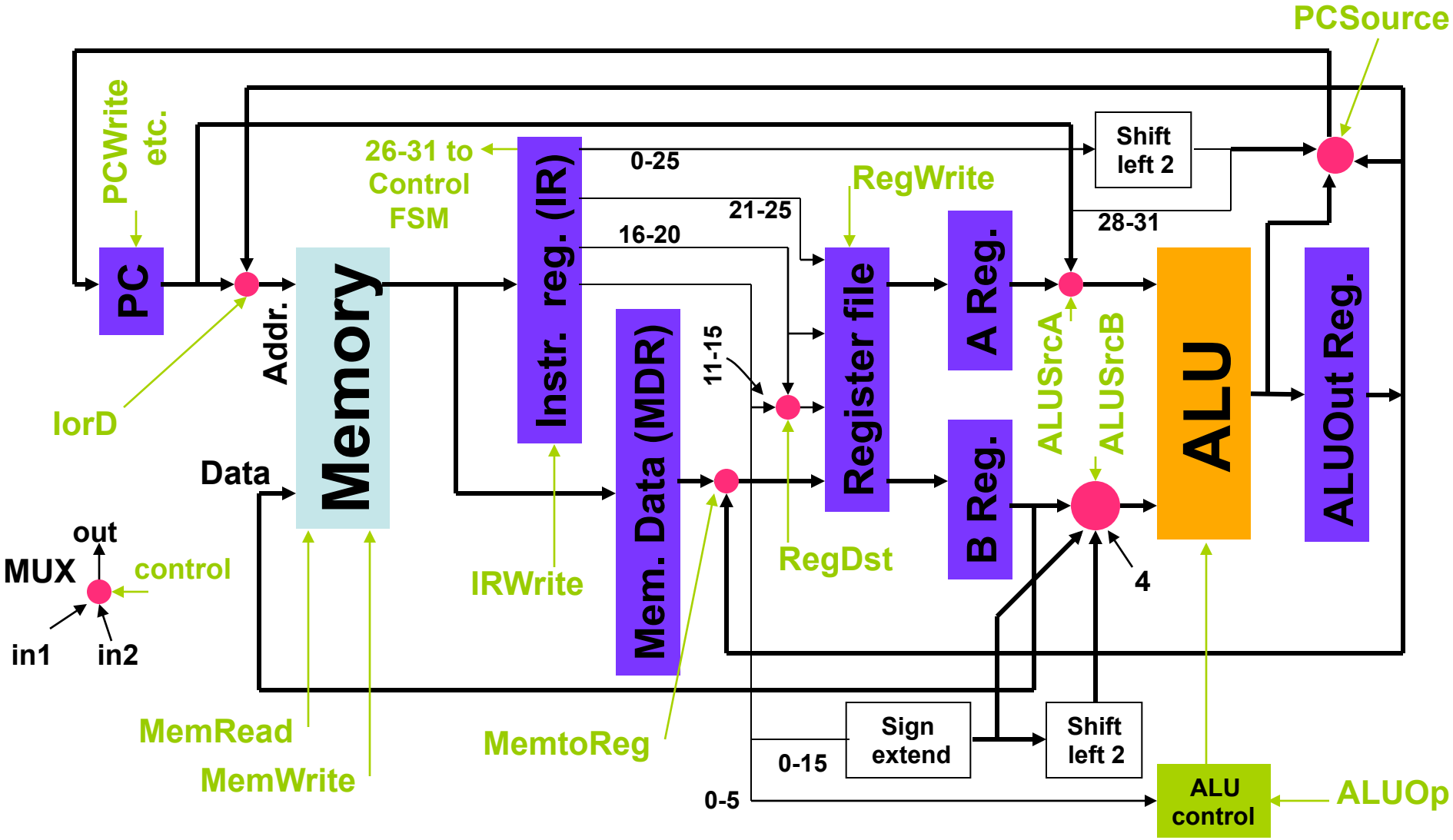
CADSL

Example Processor MIPS subset

MIPS Instruction – Subset

- Arithmetic and Logical Instructions
  - add, sub, or, and, slt
- Memory reference Instructions
  - lw, sw
- Branch
  - beq, j

Multicycle Datapath



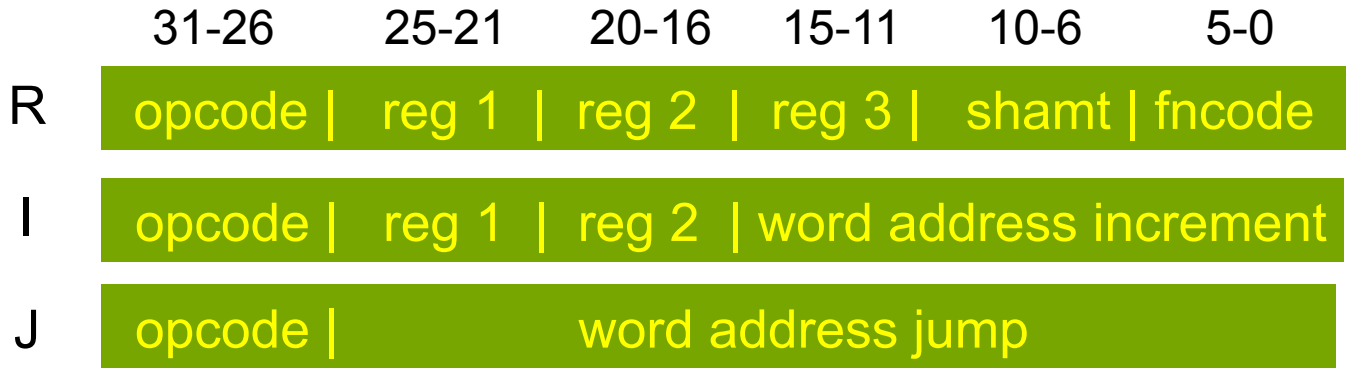
3 to 5 Cycles for an Instruction

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

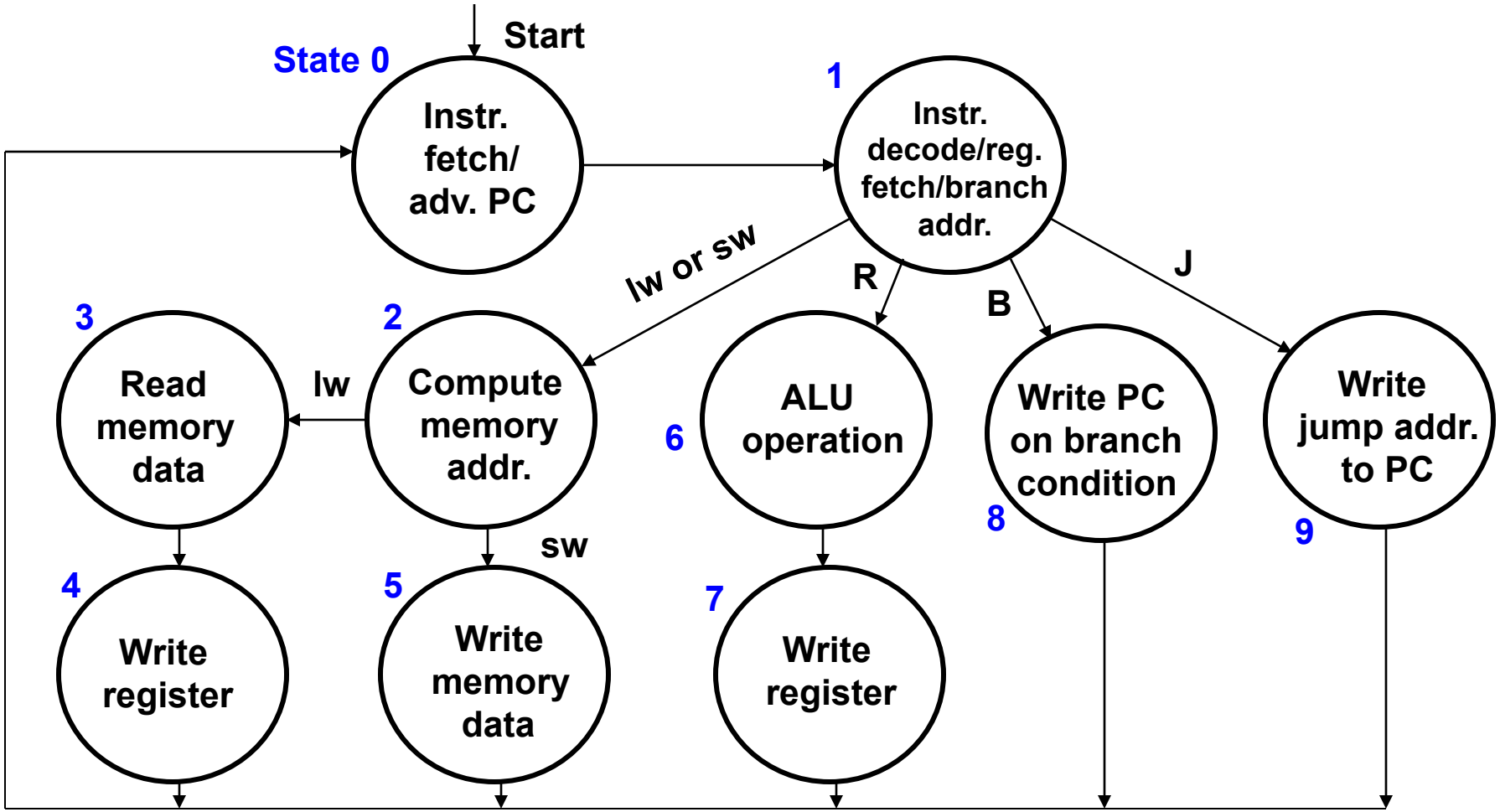
- R type, write destination register from ALUOut
  - Control signals used:
    - » **RegDst = 1 Instr. Bits 11-15 specify reg.**
    - » **MemtoReg = 0 ALUOut into reg.**
    - » **RegWrite = 1 write register**
    - » **Instruction complete, go to IF**
- I type, lw: read M[ALUOut] into MDR
  - Control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

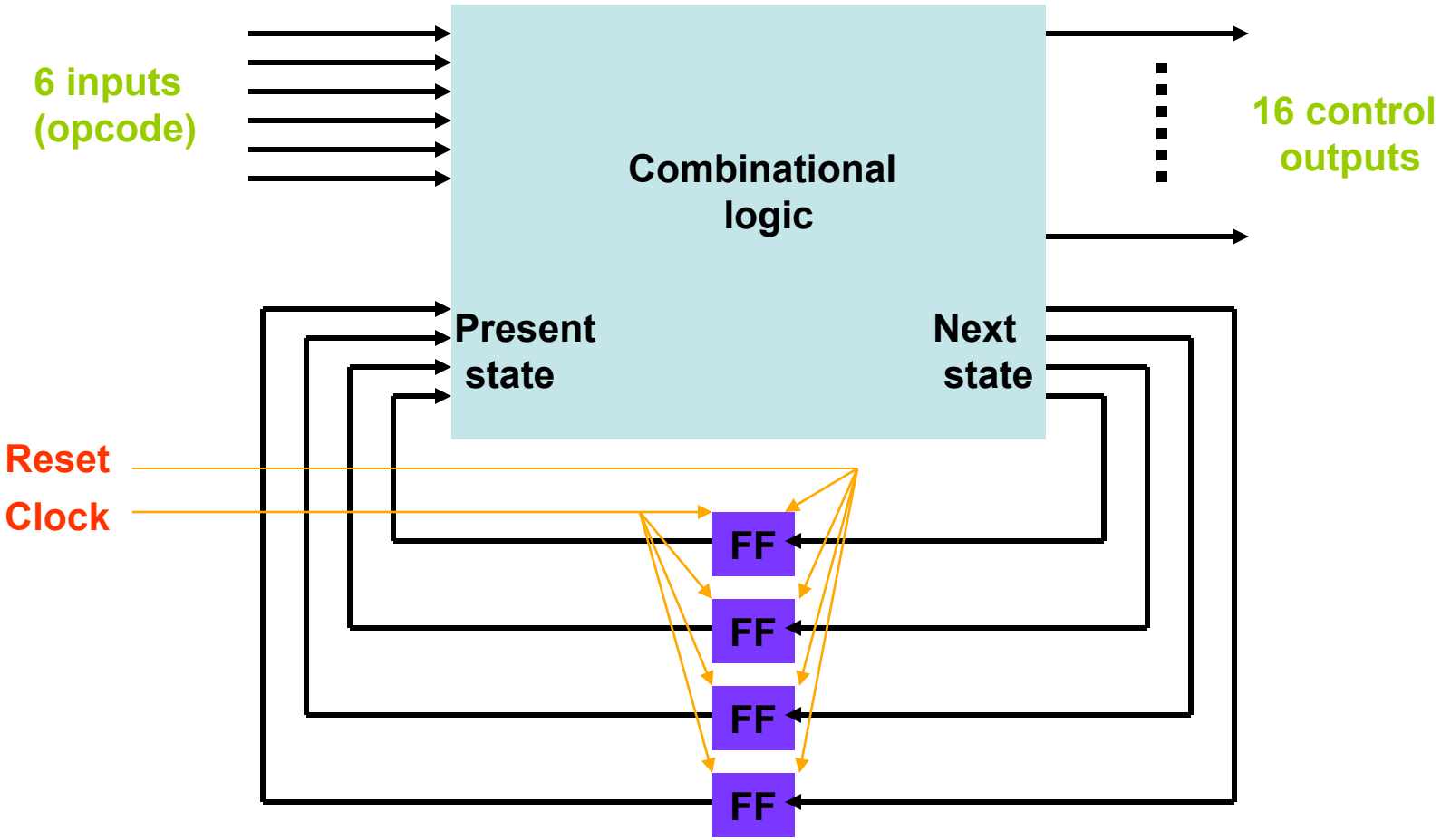
- I type, lw: write MDR to reg[IR(16-20)]
  - Control signals used:
    - » RegDst = 0 instr. Bits 16-20 are write reg
    - » MemtoReg = 1 MDR to reg file write input
    - » RegWrite = 1 read memory to MDR
    - » **Instruction complete, go to IF**

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

Control FSM


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

Control FSM (Controller)



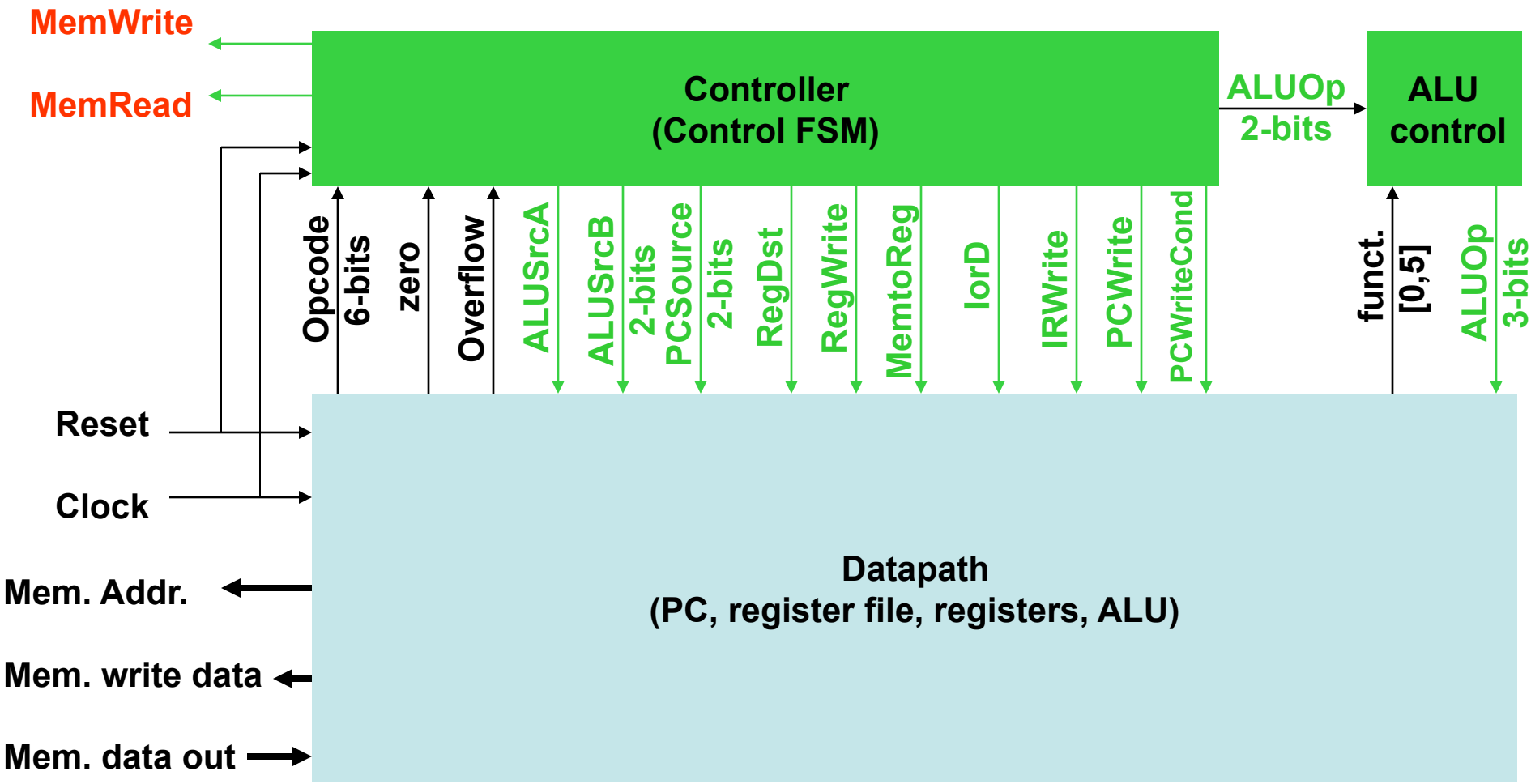
Designing the Control FSM

- 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.

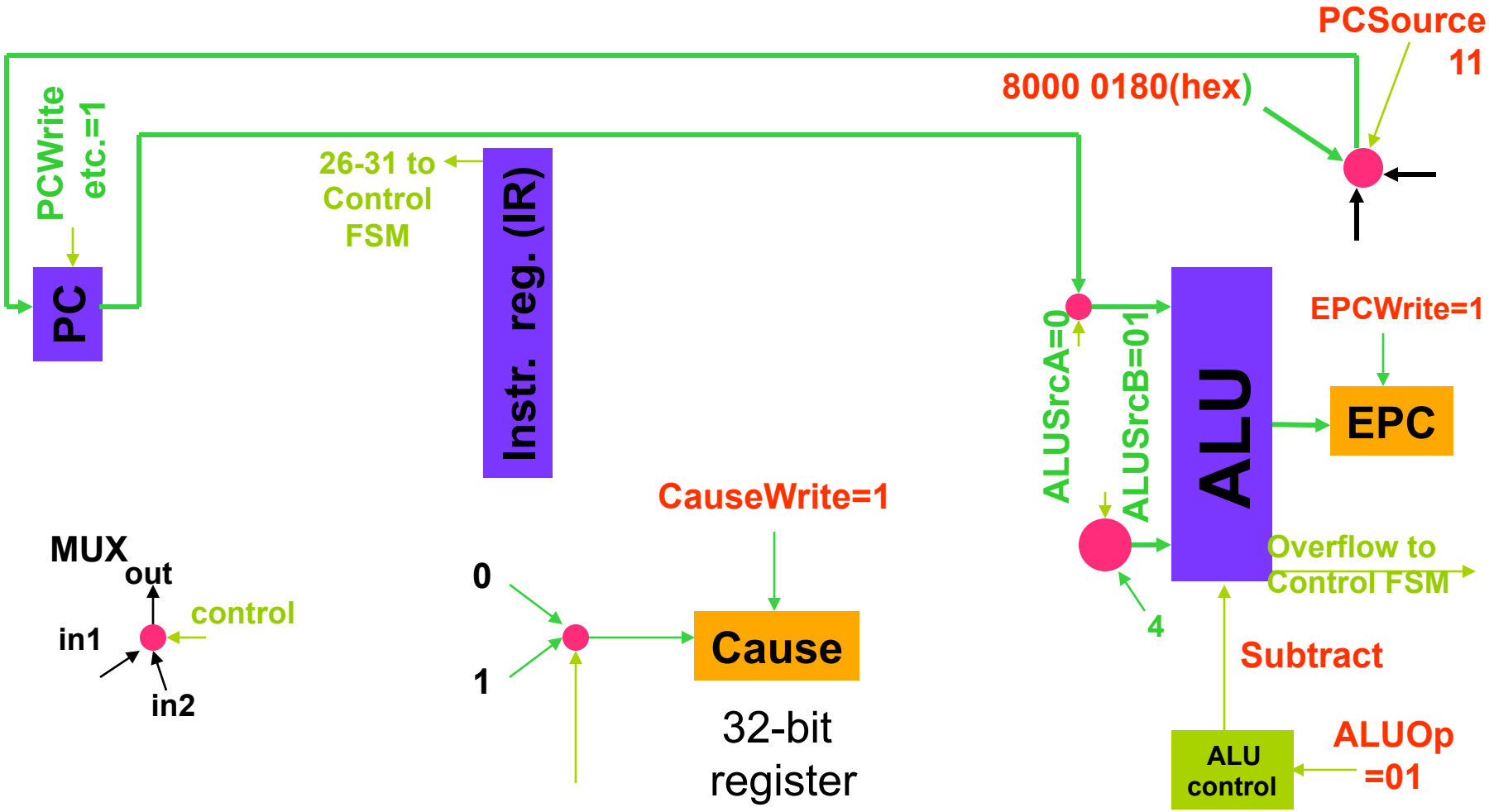
Block Diagram of a Processor



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

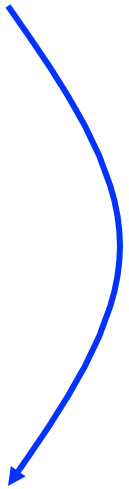


How Long Does It Take? Again

- 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 = 0.25×5 + 0.10×4 + 0.11×3 + 0.02×3 + 0.52×4

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

Multicycle vs. Single-Cycle

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

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

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

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

Traffic Flow

ILP: Instruction Level Parallelism

- Single-cycle and multi-cycle datapaths execute one instruction at a time.
- How can we get better performance?
- Answer: Execute multiple instruction at a time:
  - Pipelining – Enhance a multi-cycle datapath to fetch one instruction every cycle.
  - Parallelism – Fetch multiple instructions every cycle.

Thank You