CPU Design Steps

1. Analyze instruction set operations using independent RTN => datapath requirements.

2. Select set of datapath components & establish clock methodology.

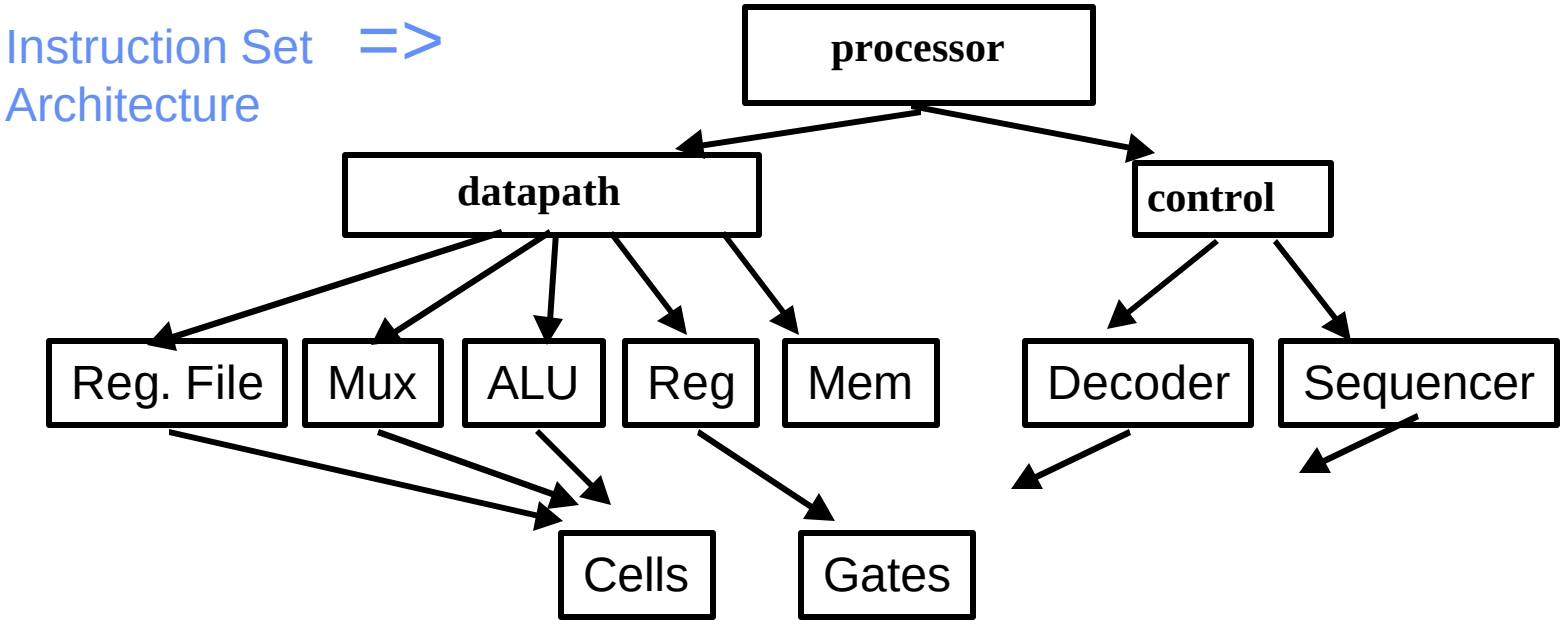
3. Assemble datapath meeting the requirements.

4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.

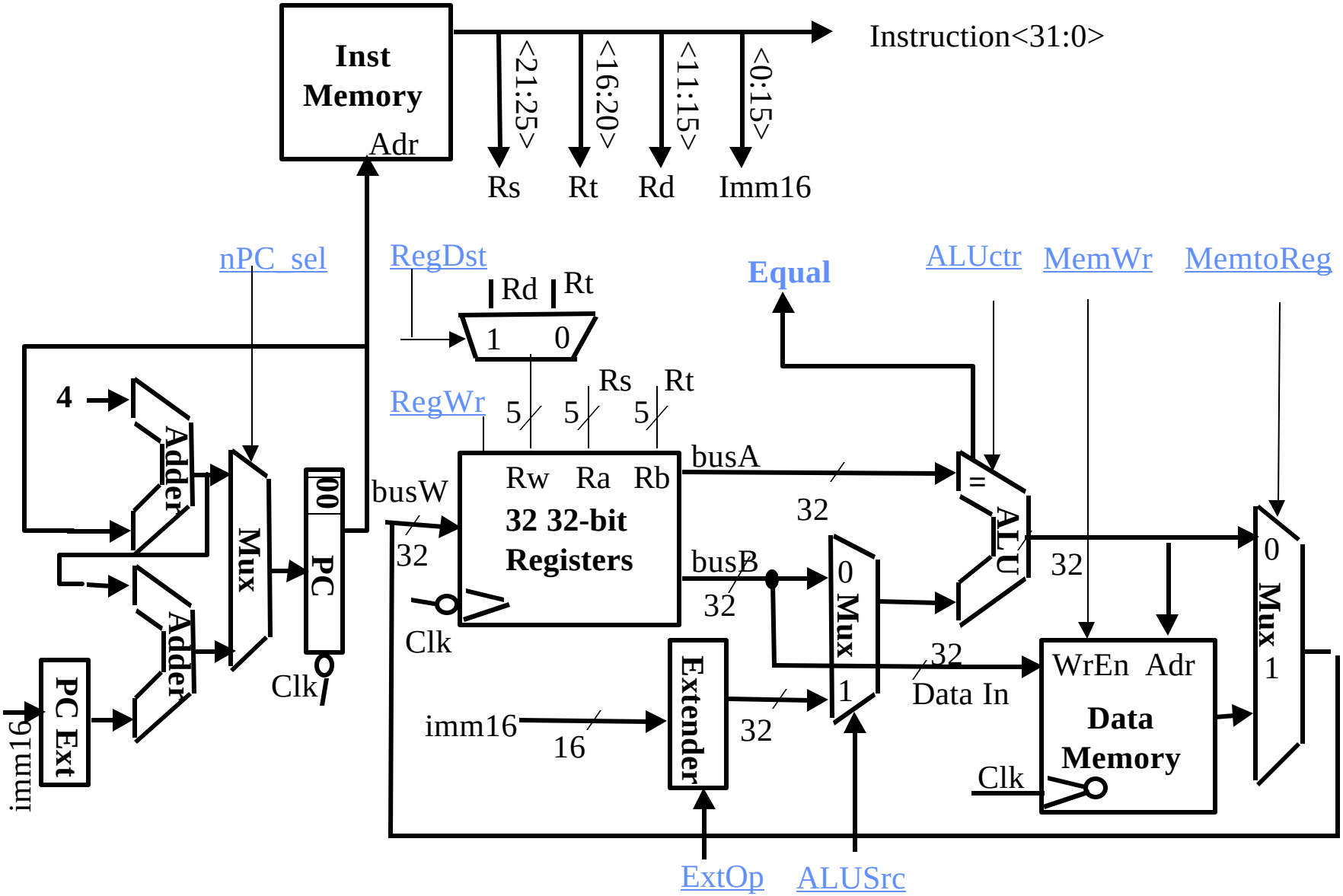
5. Assemble the control logic.

CPU Design & Implantation Process

- **Bottom-up Design:**
  - **Assemble components in target technology to establish critical timing.**
- **Top-down Design:**
  - **Specify component behavior from high-level requirements.**
- **Iterative refinement:**
  - **Establish a partial solution, expand and improve.**



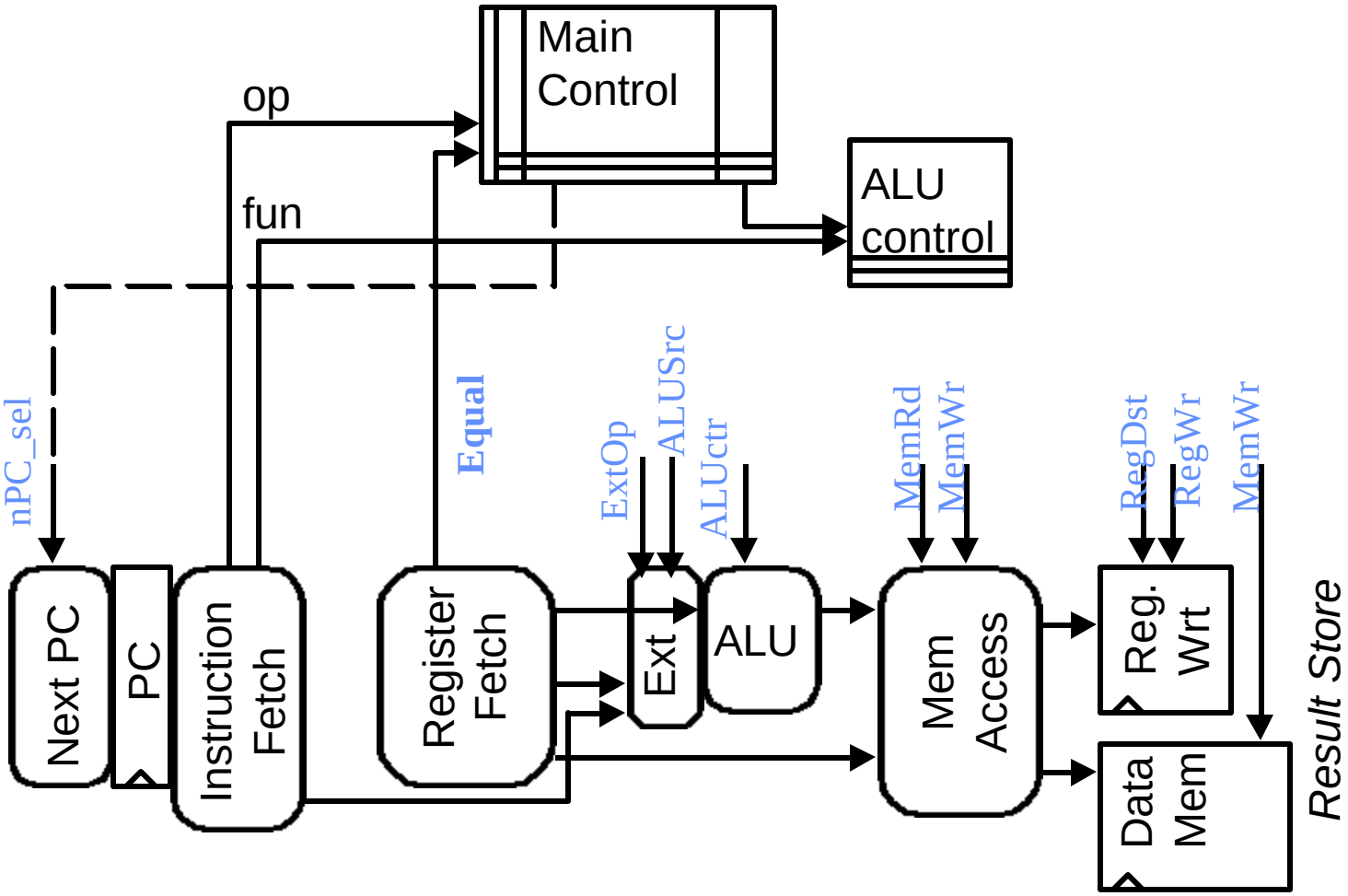
Single Cycle MIPS Datapath: CPI = 1, Long Clock Cycle



Drawback of Single Cycle Processor

- **Long cycle time.**
- **All instructions must take as much time as the slowest:**
  - **Cycle time for load is longer than needed for all other instructions.**
- **Real memory is not as well-behaved as idealized memory**
  - **Cannot always complete data access in one (short) cycle.**

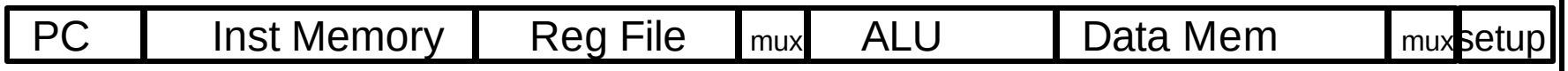
Abstract View of Single Cycle CPU



Single Cycle Instruction Timing

Arithmetic & Logical

Load



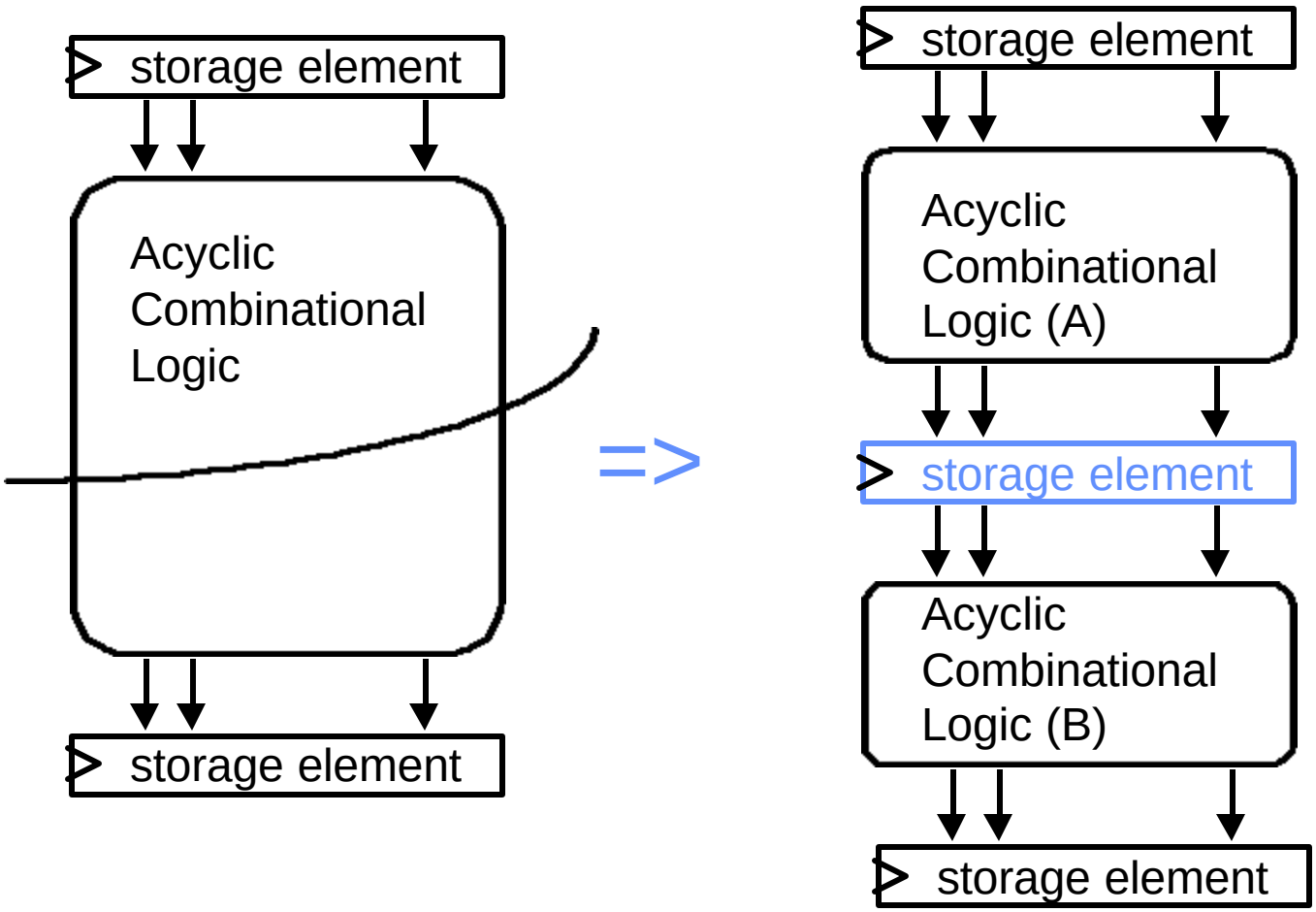
Critical Path

Store

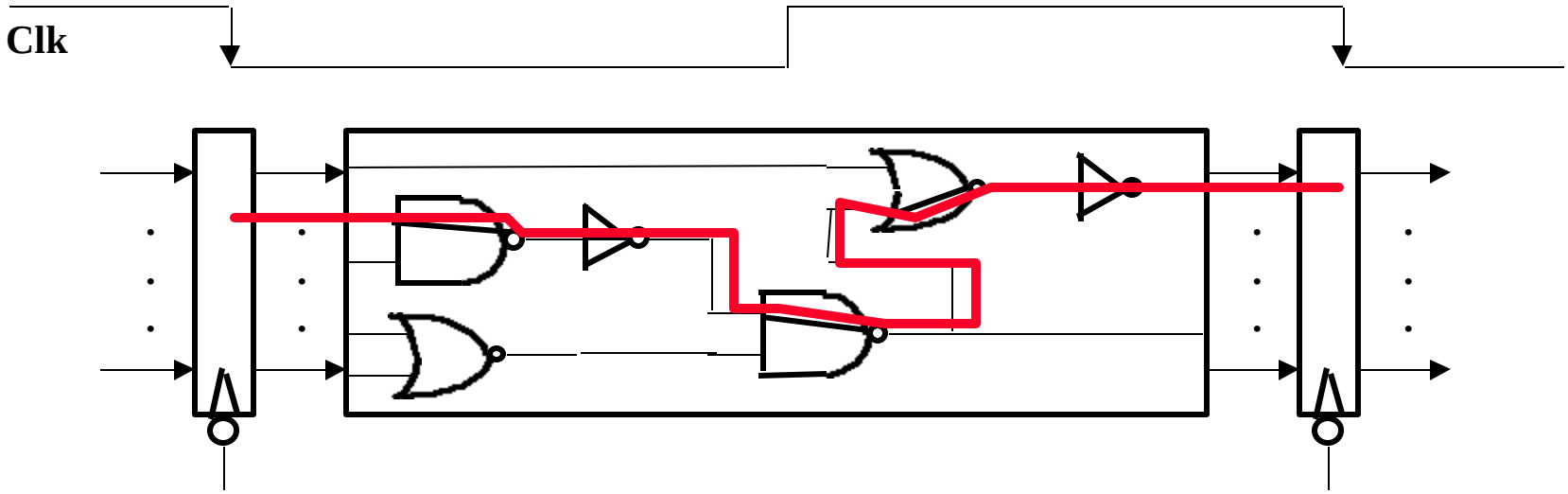
Branch

Reducing Cycle Time: Multi-Cycle Design

- **Cut combinational dependency graph by inserting registers / latches.**
- **The same work is done in two or more fast cycles, rather than one slow cycle.**

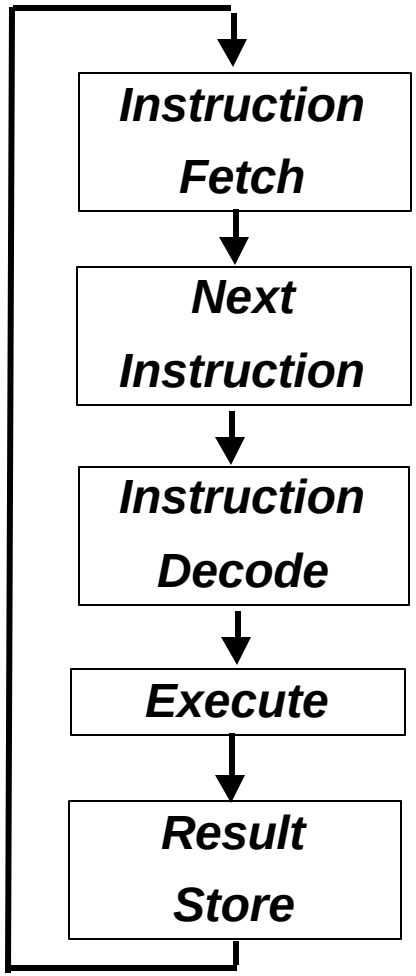


Clock Cycle Time & Critical Path



- Critical path: the slowest path between any two storage devices
- Cycle time is a function of the critical path
- must be greater than:
  - Clock-to-Q + Longest Path through the Combination Logic + Setup

Instruction Processing Cycles



Obtain instruction from program storage

Update program counter to address of next instruction

Determine instruction type

Obtain operands from registers

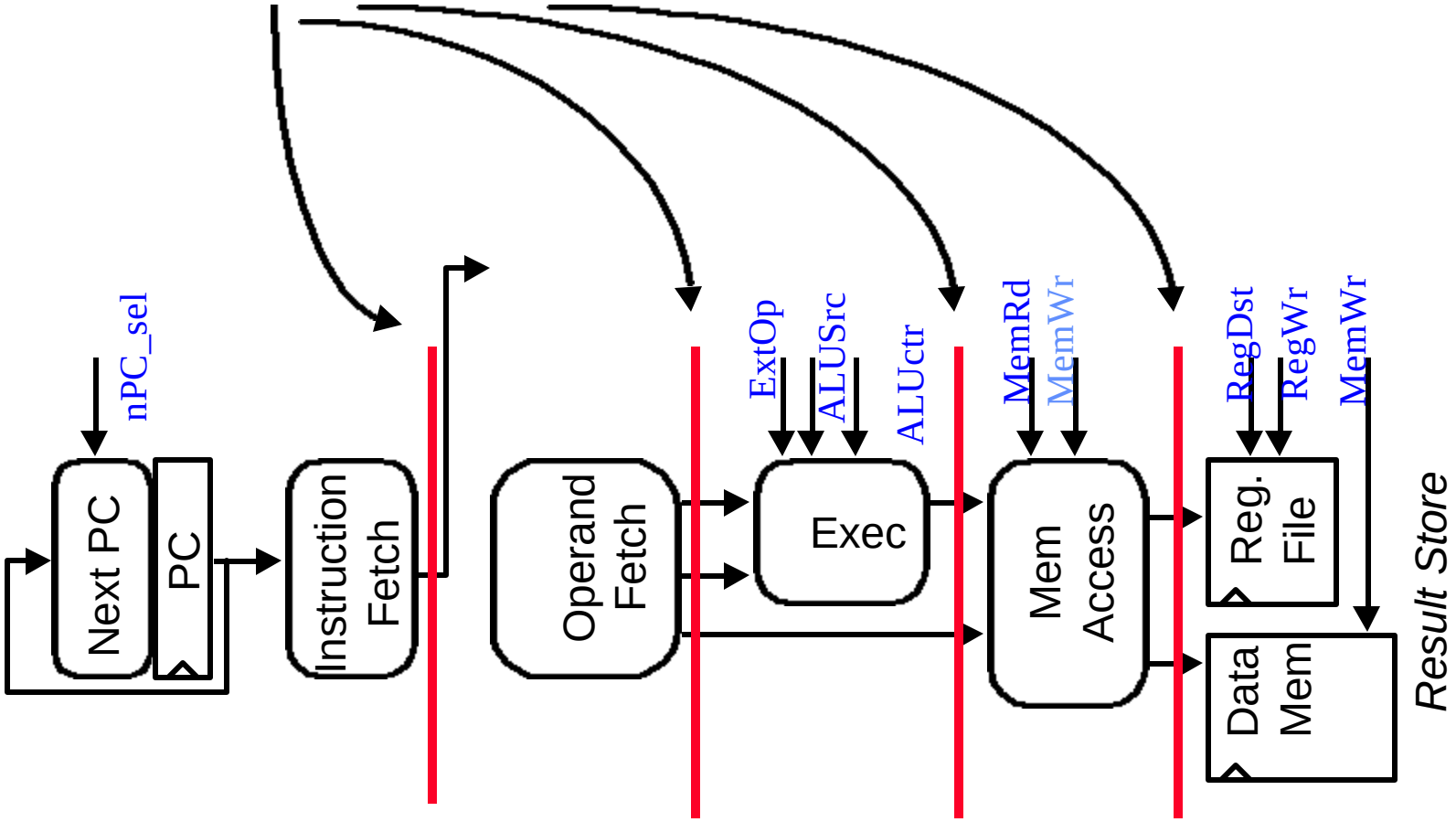
Compute result value or status

Store result in register/memory if needed (usually called Write Back).

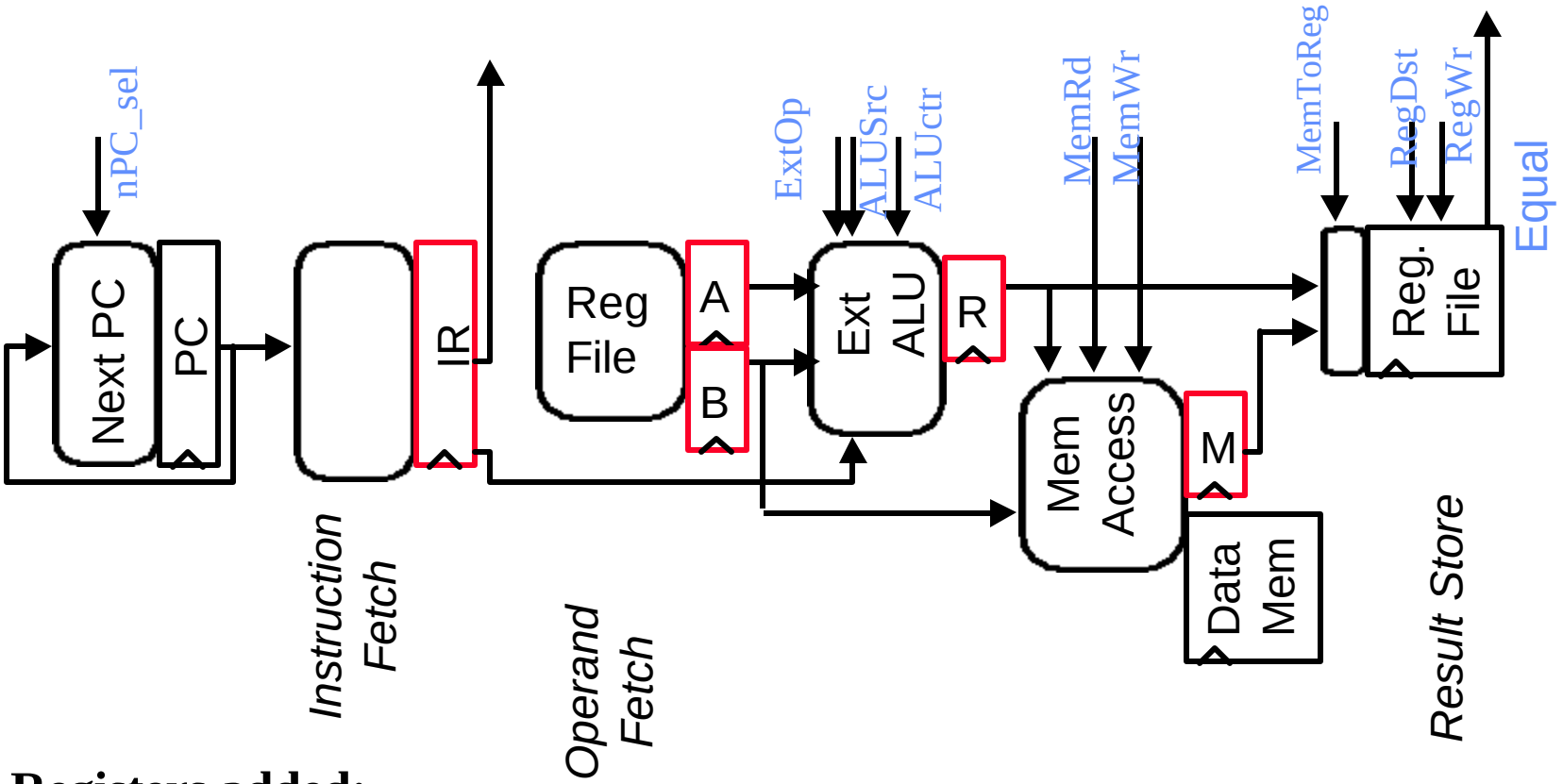
Common steps for all instructions

Partitioning The Single Cycle Datapath

Add registers between smallest steps



Example Multi-cycle Datapath



Registers added:

IR: **Instruction register**

A, B: **Two registers to hold operands read from register file.**

R: **or ALUOut, holds the output of the ALU**

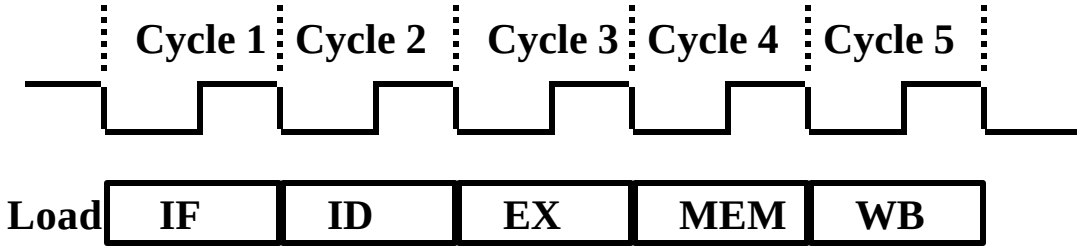
M: **or Memory data register (MDR) to hold data read from data memory**

Operations In Each Cycle

		R-Type	Logic Immediate	Load	Store	Branch
IF	Instruction Fetch	IR ¬ Mem[PC]	IR ¬ Mem[PC]	IR ¬ Mem[PC]	IR ¬ Mem[PC]	IR ¬ Mem[PC]
ID	Instruction Decode	A ¬ R[rs] B ¬ R[rt]	A ¬ R[rs]	A ¬ R[rs]	A ¬ R[rs] B ¬ R[rt]	A ¬ R[rs] B ¬ R[rt]
EX	Execution	R ¬ A + B	R ¬ A OR ZeroExt[imm16]	R ¬ A + SignEx(Im16)	R ¬ A + SignEx(Im16)	If Equal = 1 PC ¬ PC + 4 + (SignExt(imm16) x4) else PC ¬ PC + 4
MEM	Memory			M ¬ Mem[R]	Mem[R] ¬ B PC ¬ PC + 4	
WB	Write Back	R[rd] ¬ R PC ¬ PC + 4	R[rt] ¬ R PC ¬ PC + 4	R[rd] ¬ M PC ¬ PC + 4		

MIPS Multi-Cycle Datapath:

Five Cycles of Load



1- Instruction Fetch (IF) Instruction Fetch
- Fetch the instruction from the Instruction Memory.

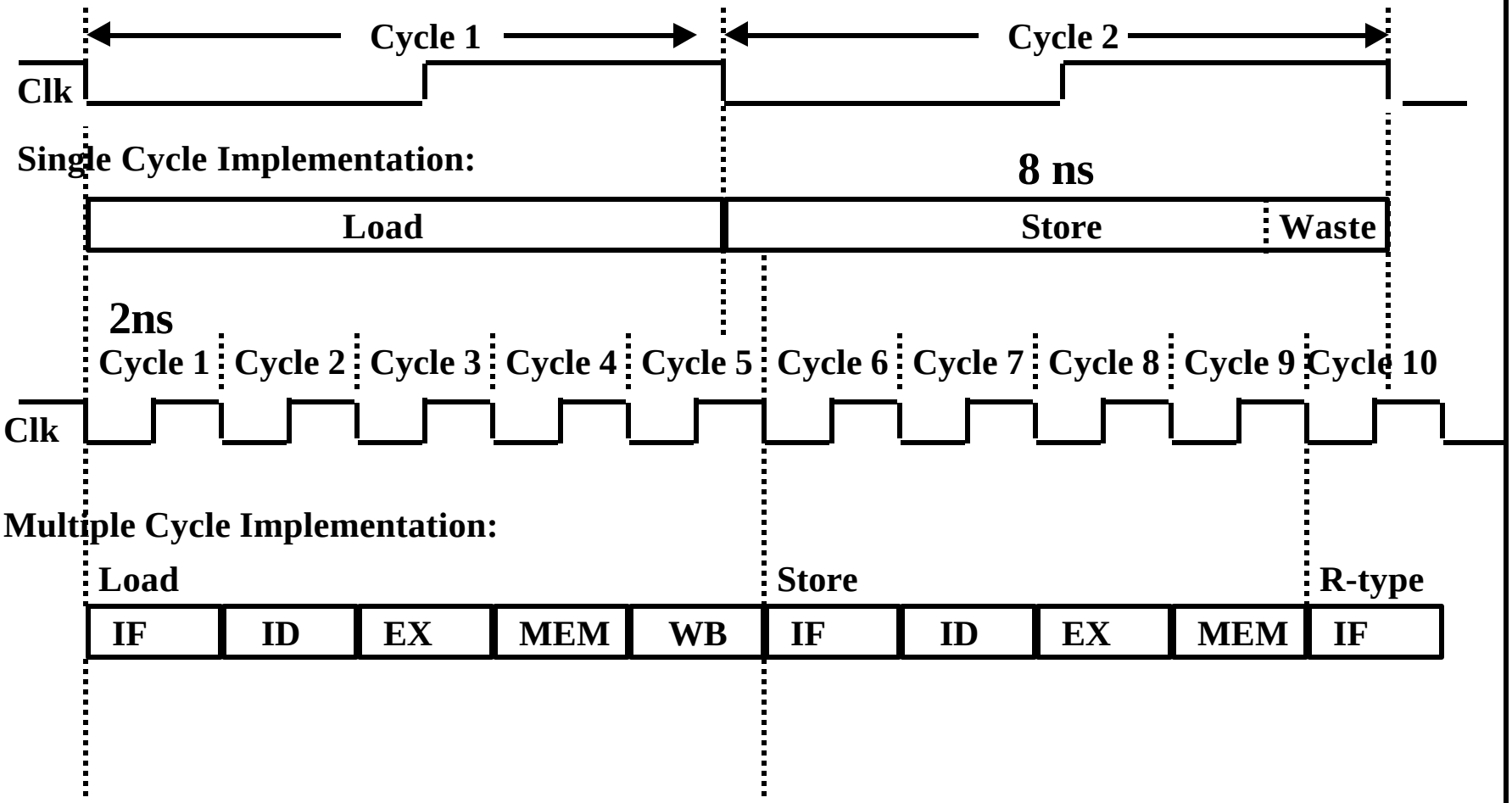
2- Instruction Decode (ID): Registers Fetch and Instruction Decode.

3- Execute (EX): Calculate the effective memory address.

4- Memory (MEM): Read the data from the Data Memory.

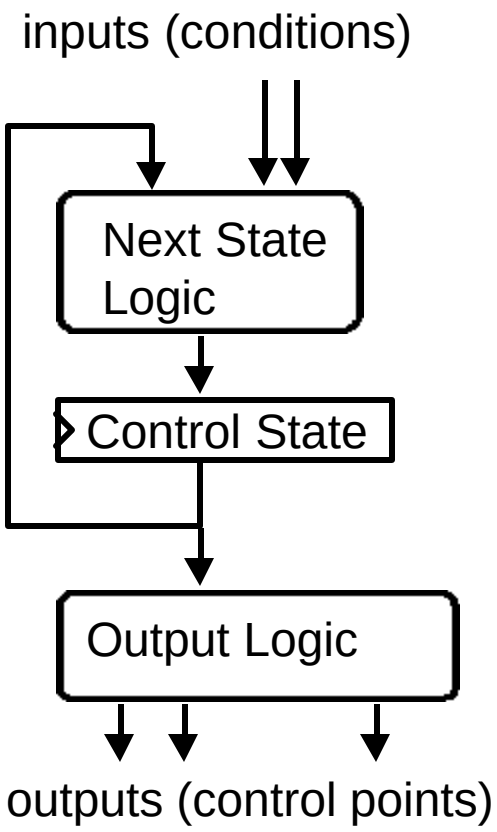
5- Write Back (WB): Write the data back to the register file. Update PC.

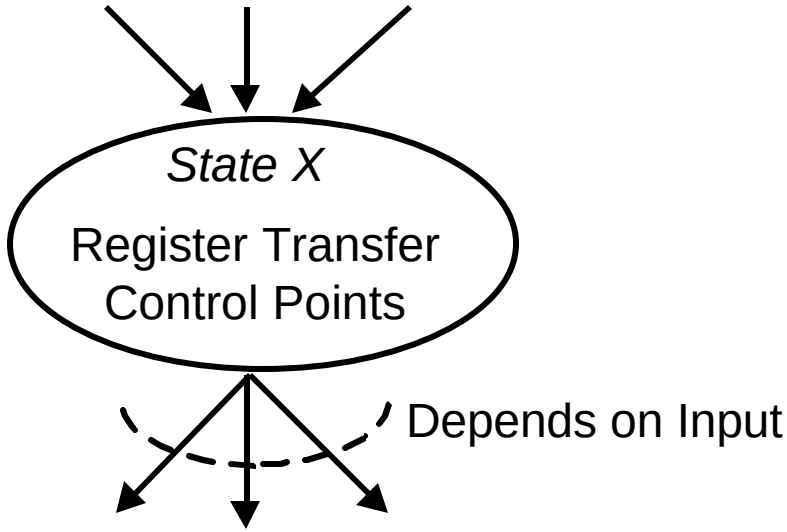
Single Cycle Vs. Multi-Cycle CPU



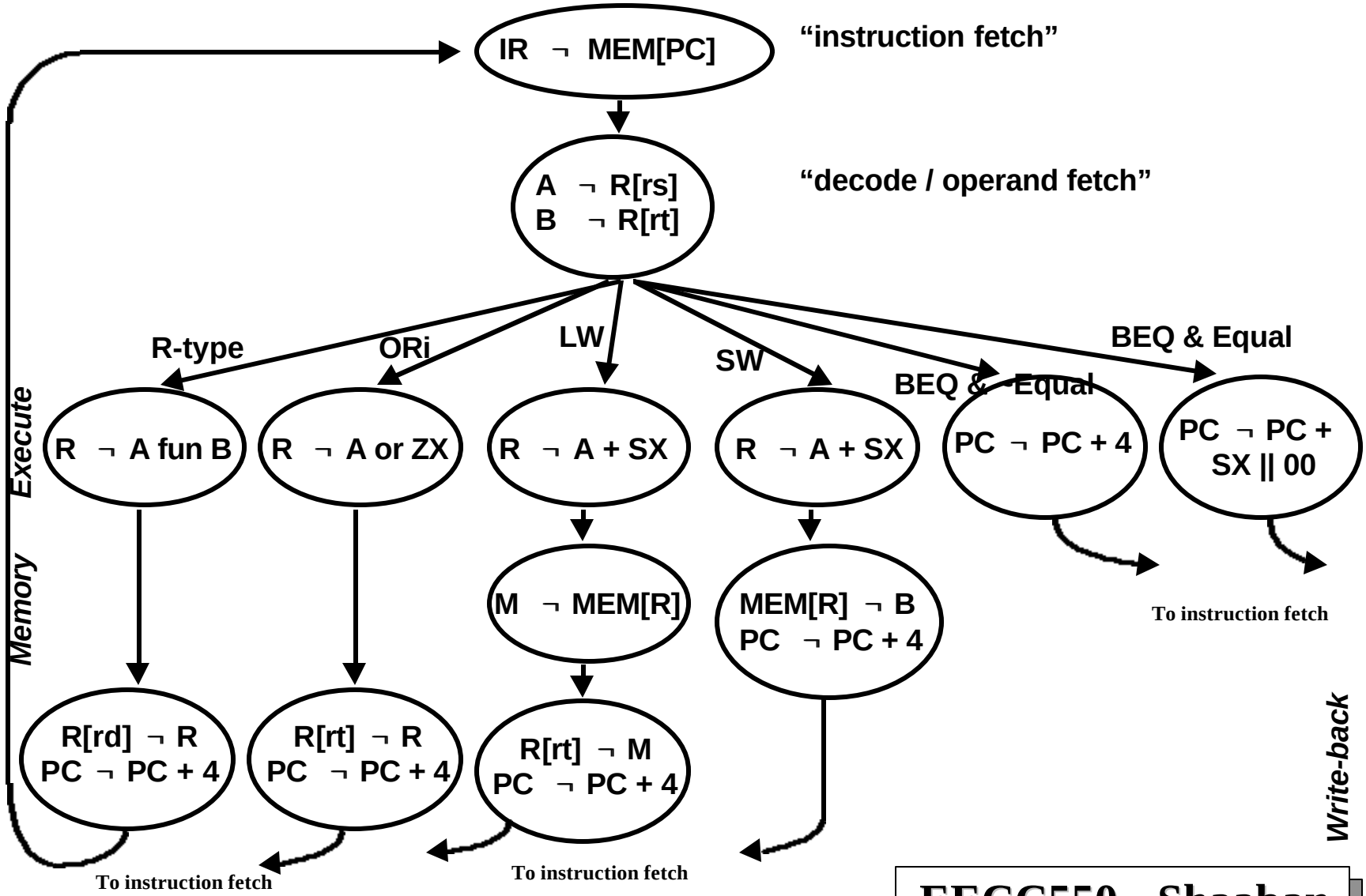
Finite State Machine (FSM) Control Model

- **State specifies control points for Register Transfer.**
- **Transfer occurs upon exiting state (same falling edge).**





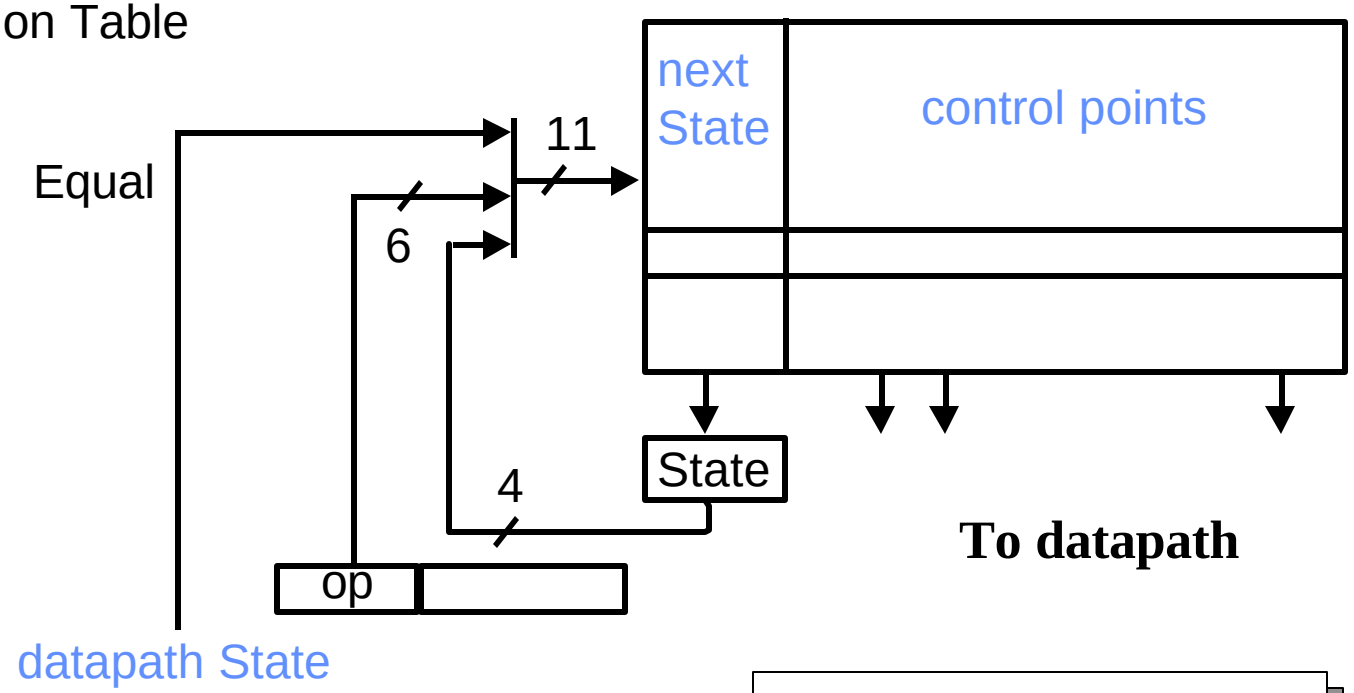
Control Specification For Multi-cycle CPU Finite State Machine (FSM)



Traditional FSM Controller

state	op	cond	next state	control points

Truth or Transition Table

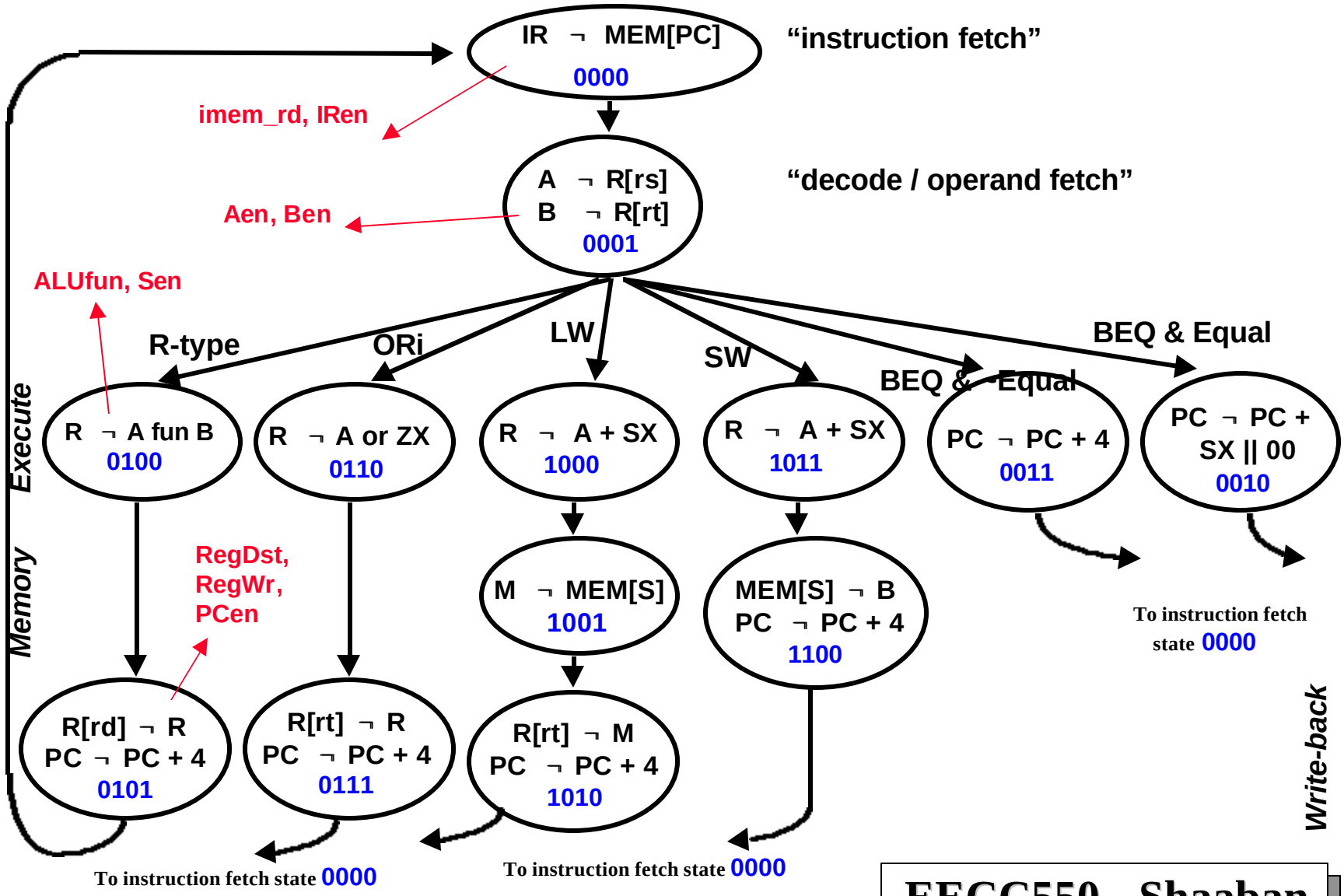


Traditional FSM Controller

datapath + state diagram => control

- **Translate RTN statements into control points.**
- **Assign states.**
- **Implement the controller.**

Mapping RTNs To Control Points Examples & State Assignments

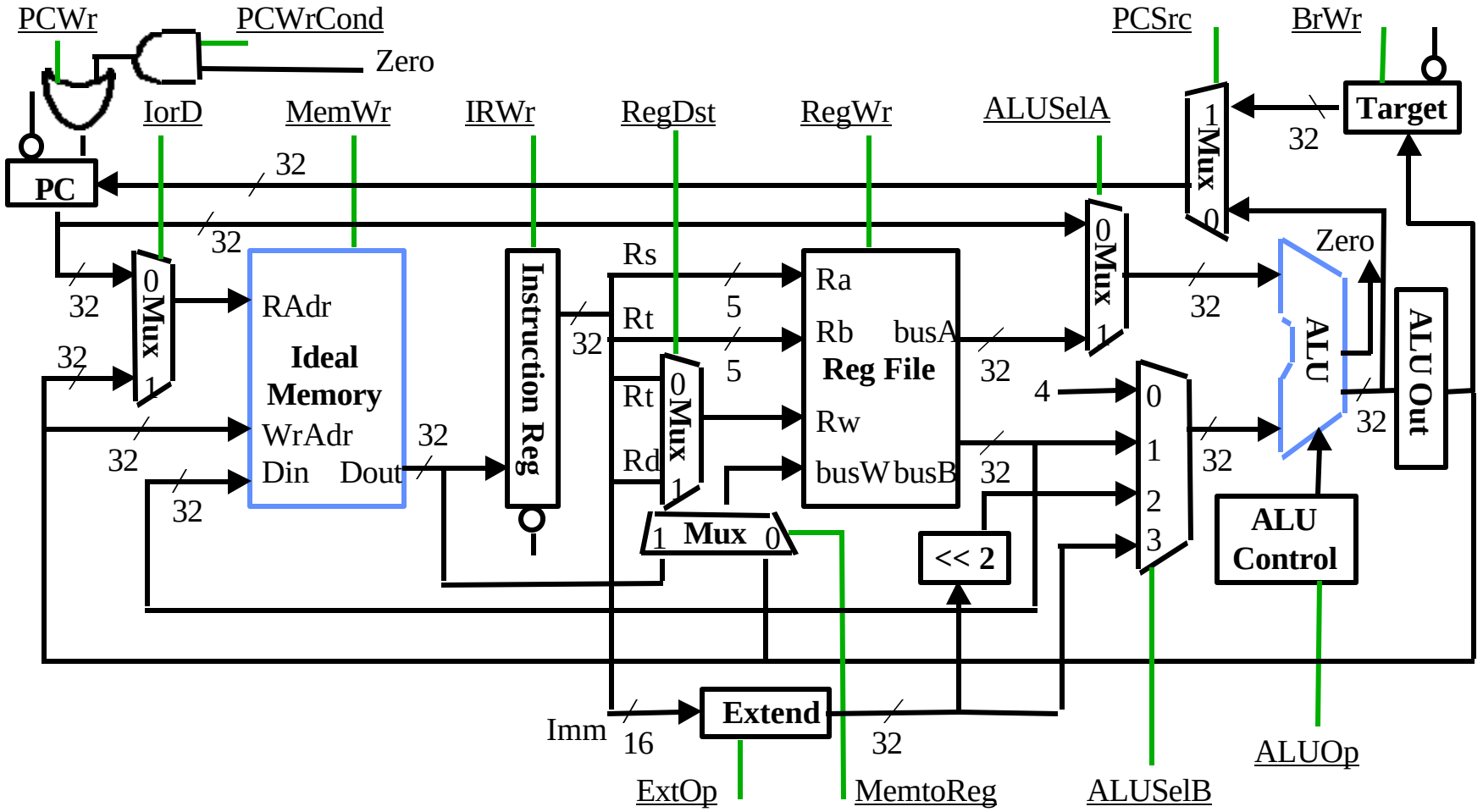


Detailed Control Specification

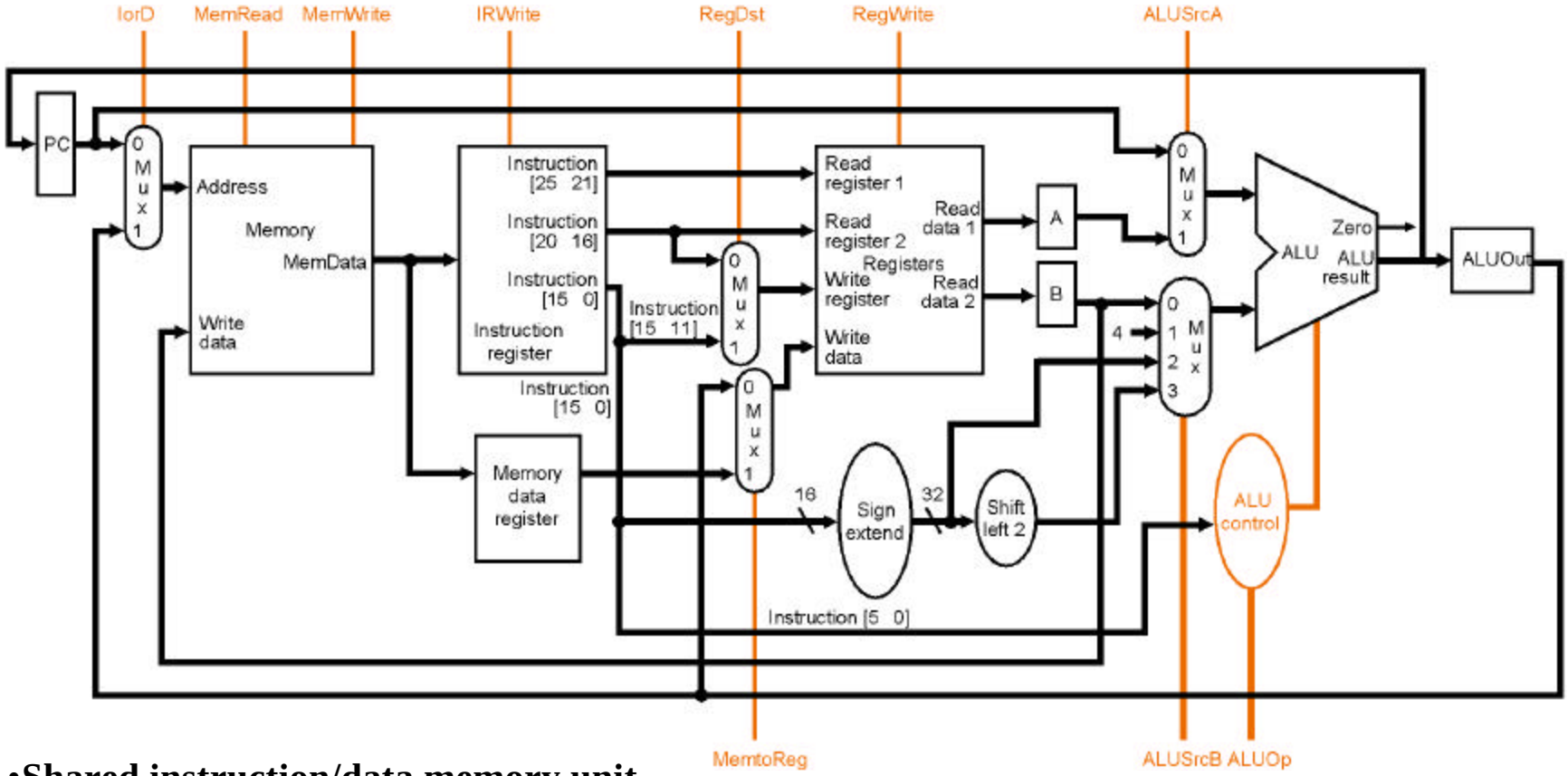
	State	Op field	Eq	Next	IR	PC en	PC sel	Ops A	Ops B	Exec Ex	Exec Sr	Exec ALU	Exec S	Mem R	Mem W	Mem M	Write-Back M-R	Write-Back Wr	Write-Back Dst
	0000	??????	?	0001	1														
	0001	BEQ	0	0011				1	1										
	0001	BEQ	1	0010				1	1										
	0001	R-type	x	0100				1	1										
	0001	orI	x	0110				1	1										
	0001	LW	x	1000				1	1										
	0001	SW	x	1011				1	1										
BEQ	0010	xxxxxx	x	0000		1	1												
	0011	xxxxxx	x	0000		1	0												
R	0100	xxxxxx	x	0101						0	1	fun	1						
	0101	xxxxxx	x	0000		1	0										0	1	1
ORI	0110	xxxxxx	x	0111						0	0	or	1						
	0111	xxxxxx	x	0000		1	0										0	1	0
LW	1000	xxxxxx	x	1001						1	0	add	1						
	1001	xxxxxx	x	1010										1	0	0			
	1010	xxxxxx	x	0000		1	0										1	1	0
SW	1011	xxxxxx	x	1100						1	0	add	1						
	1100	xxxxxx	x	0000		1	0							0	1				

Alternative Multiple Cycle Datapath (In Textbook)

- **Miminizes Hardware: 1 memory, 1 adder**



Alternative Multiple Cycle Datapath (In Textbook)

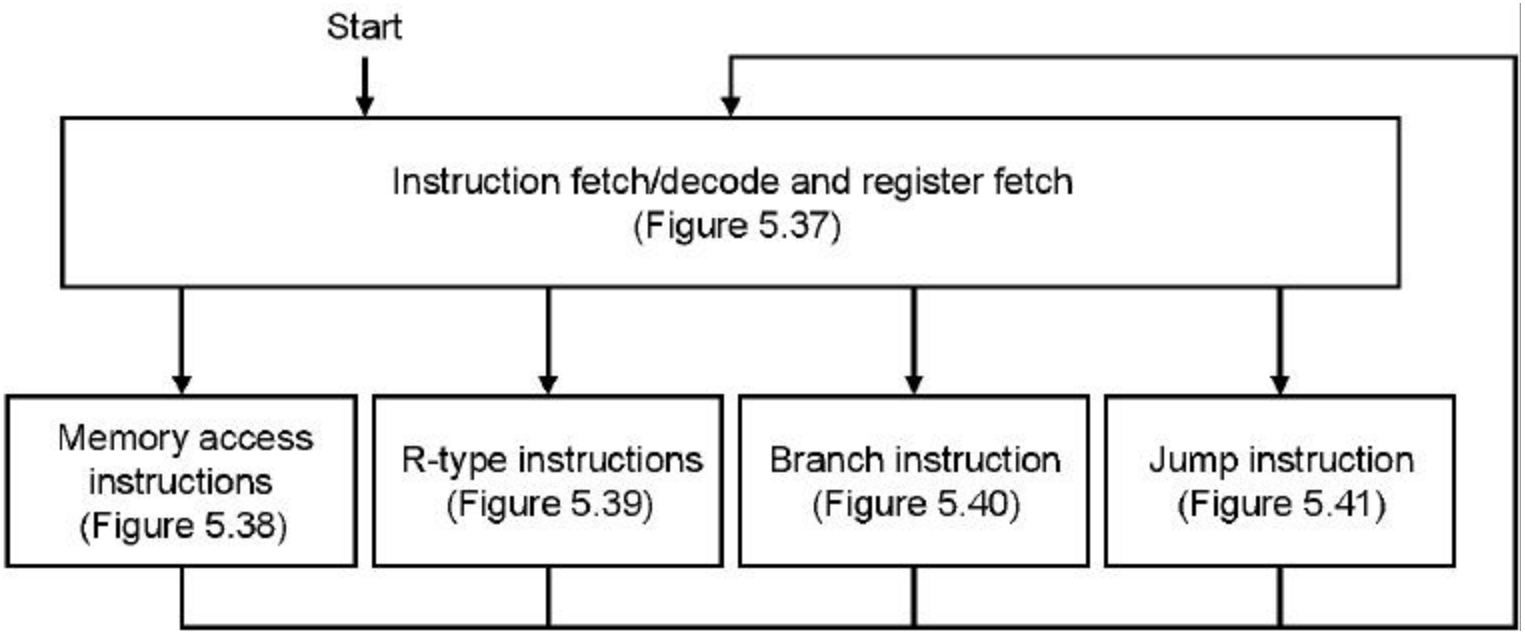


- **Shared instruction/data memory unit**
- **A single ALU shared among instructions**
- **Shared units require additional or widened multiplexors**
- **Temporary registers to hold data between clock cycles of the instruction:**
  - **Additional registers: Instruction Register (IR), Memory Data Register (MDR), A, B, ALUOut**

Operations In Each Cycle

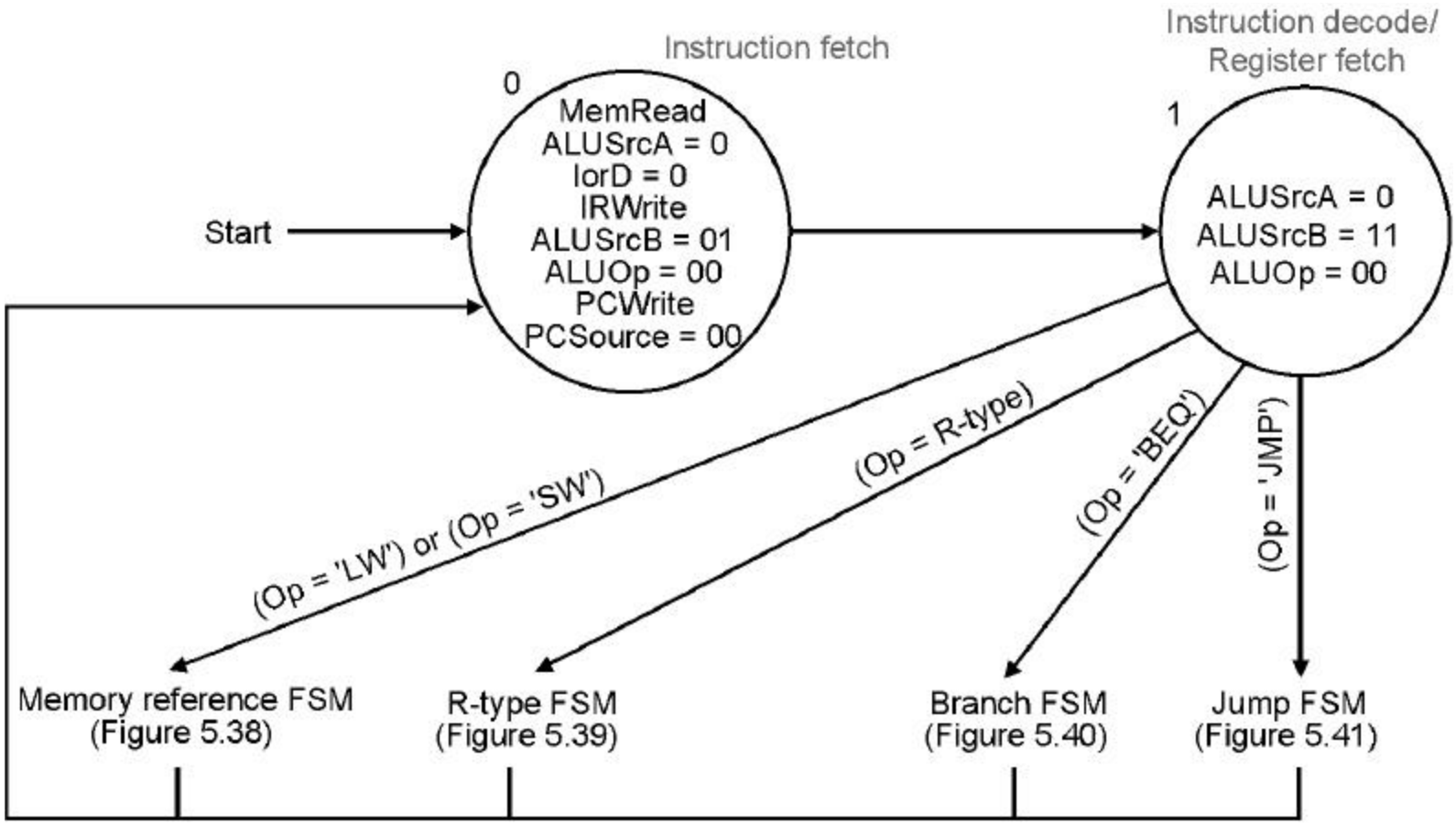
	R-Type	Logic Immediate	Load	Store	Branch
Instruction Fetch	IR ← Mem[PC] PC ← PC + 4	IR ← Mem[PC] PC ← PC + 4	IR ← Mem[PC] PC ← PC + 4	IR ← Mem[PC] PC ← PC + 4	IR ← Mem[PC] PC ← PC + 4
Instruction Decode	A ← R[rs] B ← R[rt] ALUout ← PC + (SignExt(imm16) x4)	A ← R[rs] B ← R[rt] ALUout ← PC + (SignExt(imm16) x4)	A ← R[rs] B ← R[rt] ALUout ← PC + (SignExt(imm16) x4)	A ← R[rs] B ← R[rt] ALUout ← PC + (SignExt(imm16) x4)	A ← R[rs] B ← R[rt] ALUout ← PC + (SignExt(imm16) x4)
Execution	ALUout ← A + B	ALUout ← A OR ZeroExt[imm16]	ALUout ← A + SignEx(Im16)	ALUout ← A + SignEx(Im16)	If Equal = 1 PC ← ALUout
Memory			M ← Mem[ALUout]	Mem[ALUout] ← B	
Write Back	R[rd] ← ALUout	R[rt] ← ALUout	R[rd] ← Mem		

High-Level View of Finite State Machine Control

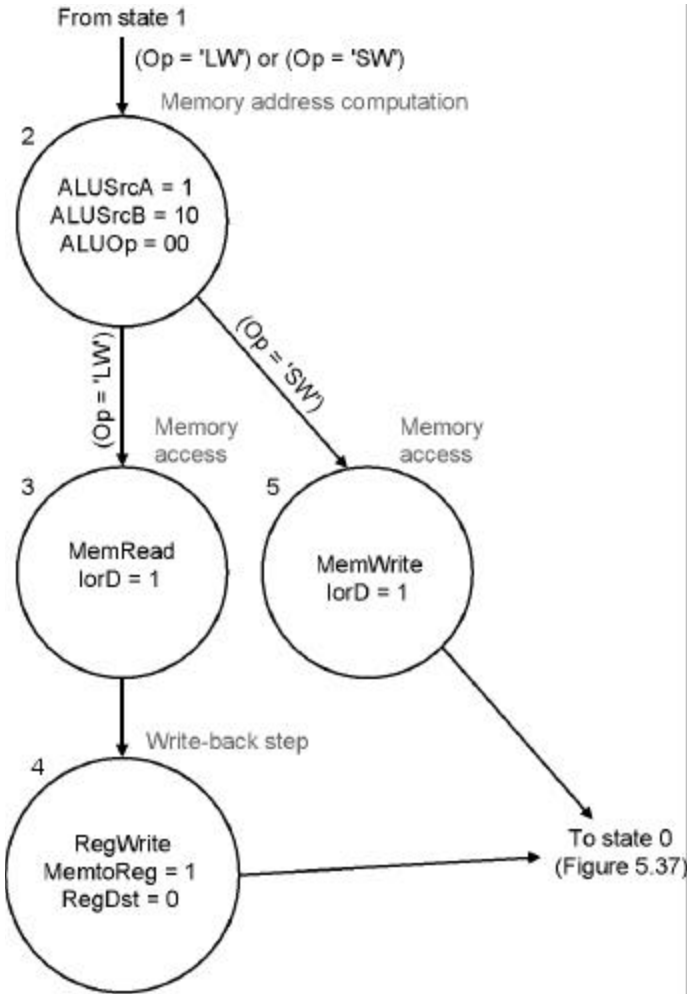


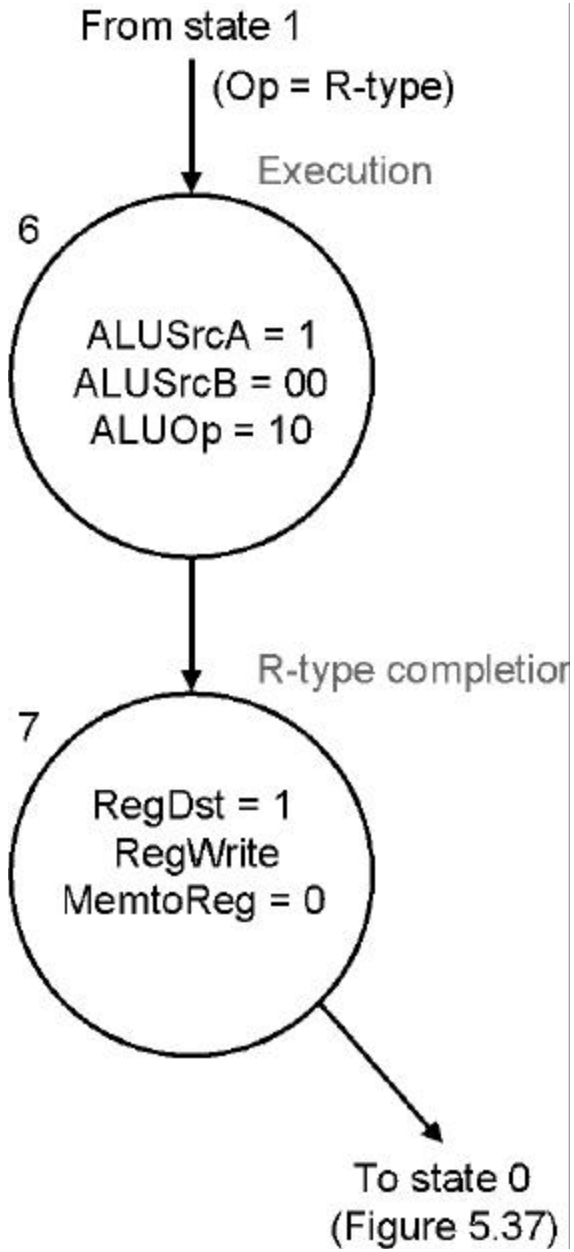
- **First steps are independent of the instruction class**
- **Then a series of sequences that depend on the instruction opcode**
- **Then the control returns to fetch a new instruction.**
- **Each box above represents one or several state.**

Instruction Fetch and Decode FSM States



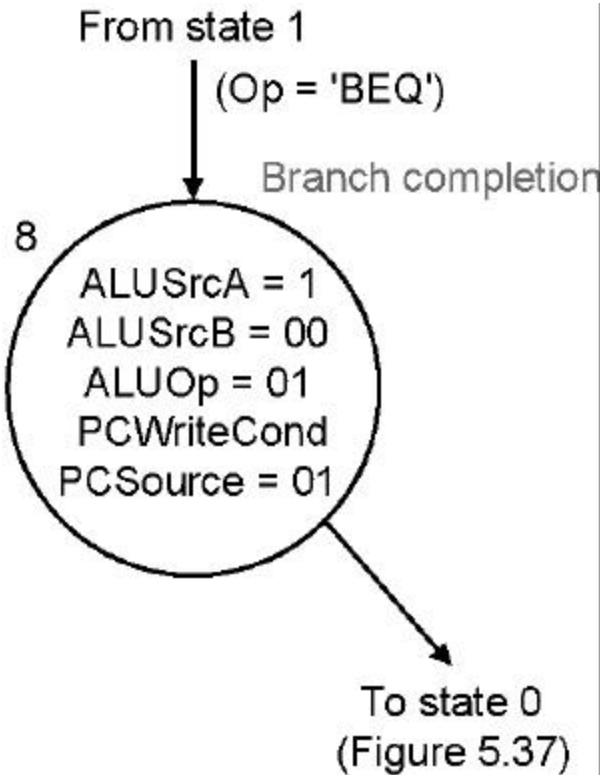
Load/Store Instructions FSM States



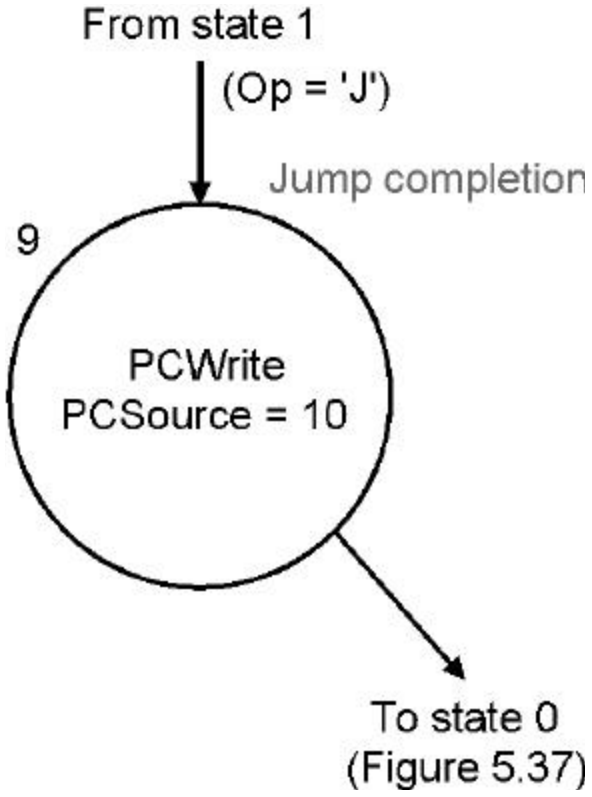


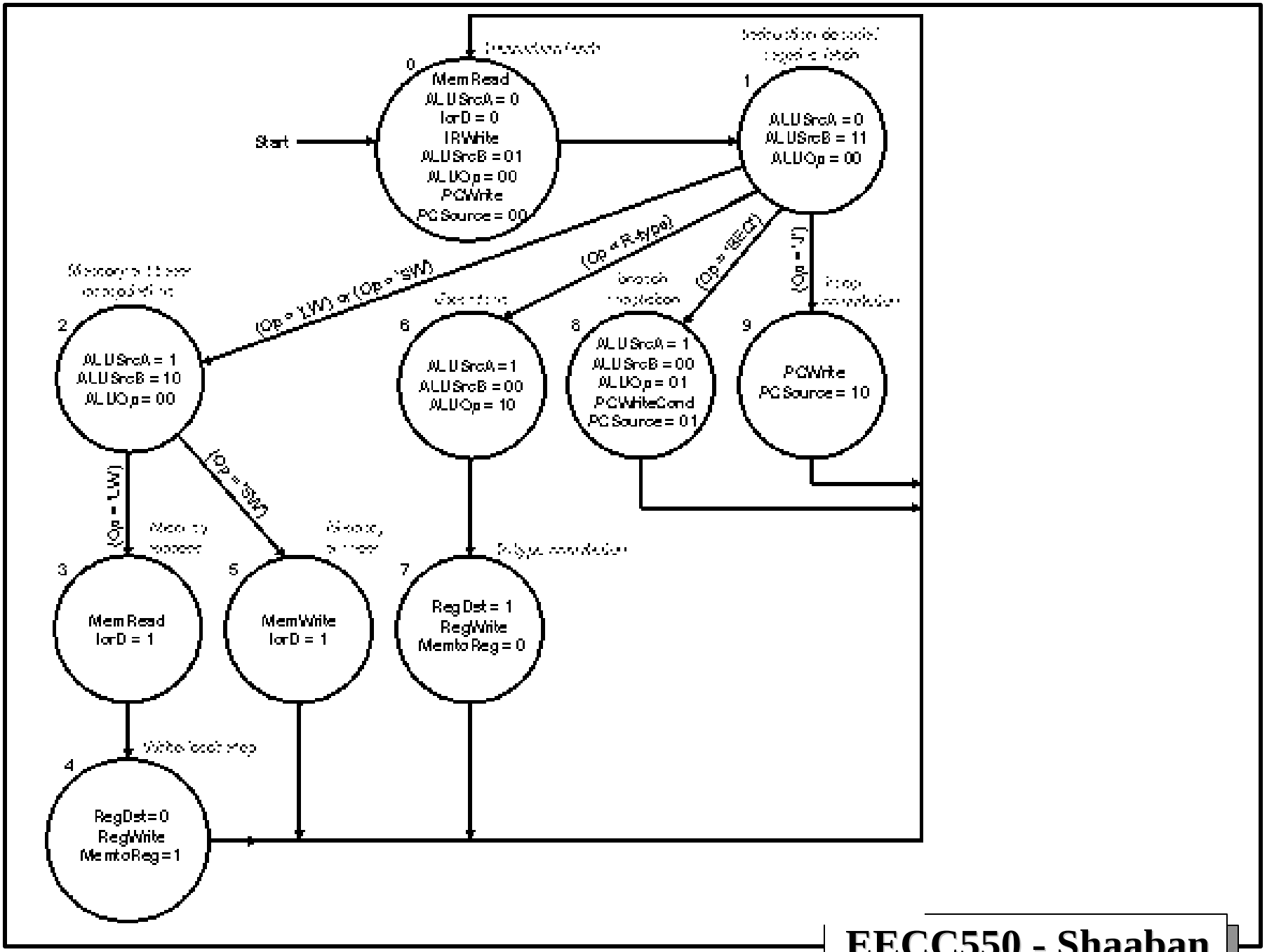
R-Type Instructions FSM States

Branch Instruction Single State

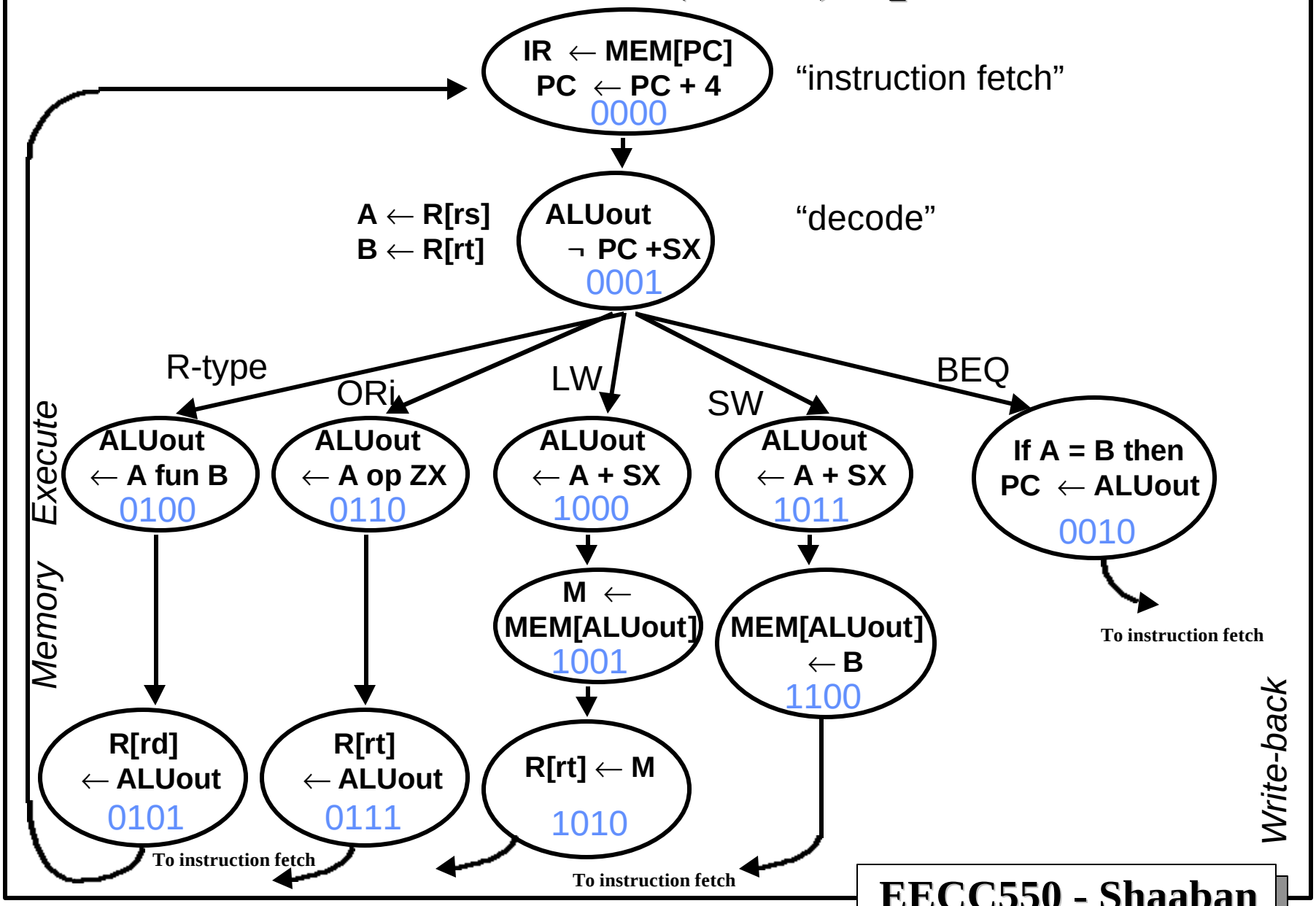


Jump Instruction Single State




Start
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
1
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
(Op = JMP)
(Op = R-type)
(Op = BEQ)
(Op = LW) or (Op = SW)
2
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
6
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10
8
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
9
PCWrite
PCSource = 10
(Op = LW)
(Op = SW)
3
MemRead
IorD = 1
5
MemWrite
IorD = 1
7
RegDst = 1
RegWrite
MemtoReg = 0
4
RegDst = 0
RegWrite
MemtoReg = 1

Finite State Machine (FSM) Specification



MIPS Multi-cycle Datapath Performance Evaluation

- **What is the average CPI?**
  - **State diagram gives CPI for each instruction type**
  - **Workload below gives frequency of each type**

Type	CPI_i for type	Frequency	CPI_i x $freqI_i$
Arith/Logic	4	40%	1.6
Load	5	30%	1.5
Store	4	10%	0.4
branch	3	20%	0.6
		Average CPI:	4.1

Better than CPI = 5 if all instructions took the same number of clock cycles (5).