Major CPU Design Steps

Datapath

1. Analyze instruction set operations using independent RTN

   ISA => RTN => datapath requirements.

   – This provides the the required datapath components and how they are connected to meet ISA requirements.

2. Select required datapath components, connections & establish clock methodology (e.g clock edge-triggered).

   + Determine number of cycles per instruction and operations in each cycle.

3. Assemble datapath meeting the requirements.

Control

4. Identify and define the function of all control points or signals needed by the datapath.

   – Analyze implementation of each instruction to determine setting of control points that affects its operations and register transfer.

   For each cycle of the instruction

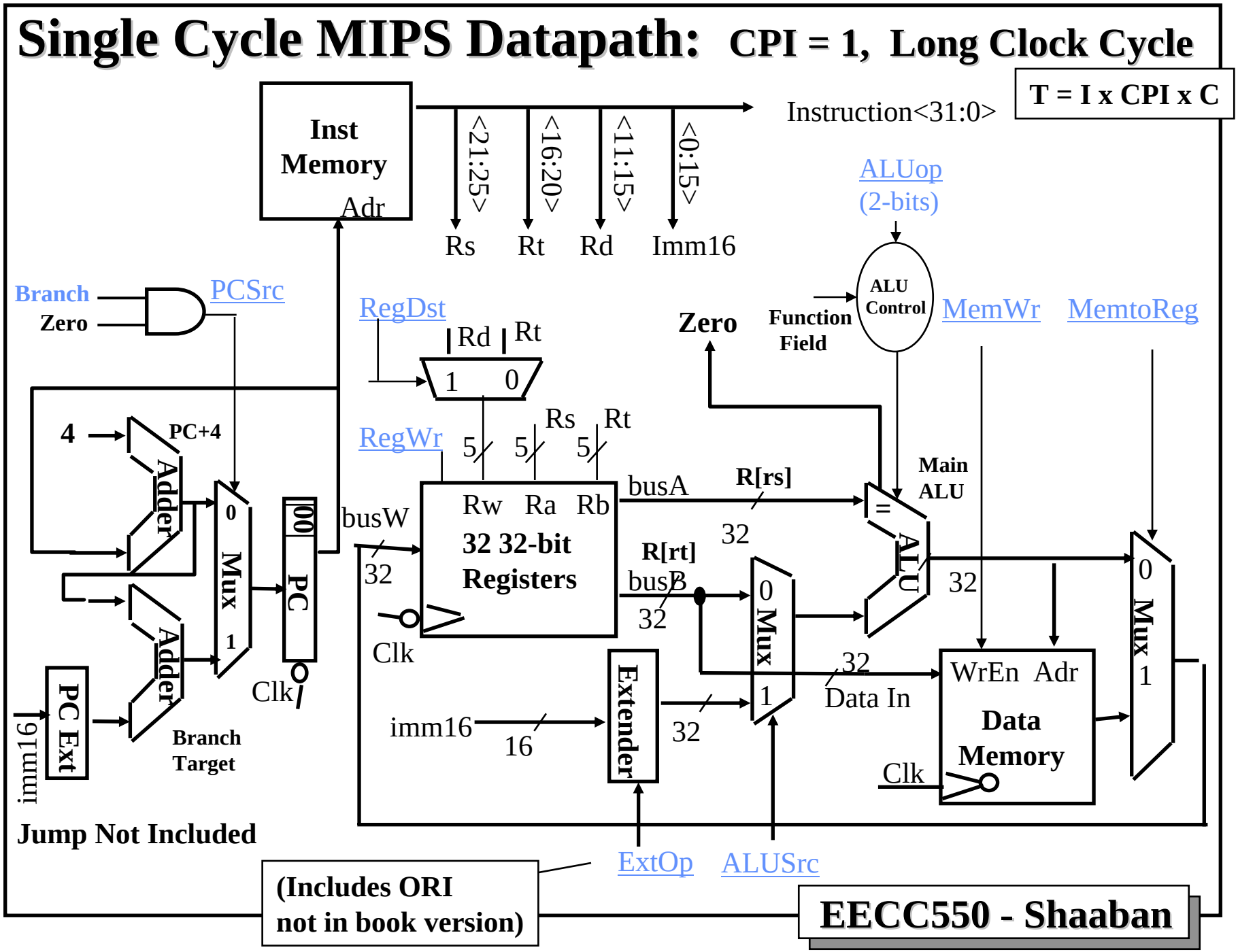
5. Design & assemble the control logic.

   – Hard-Wired: Finite-state machine implementation.

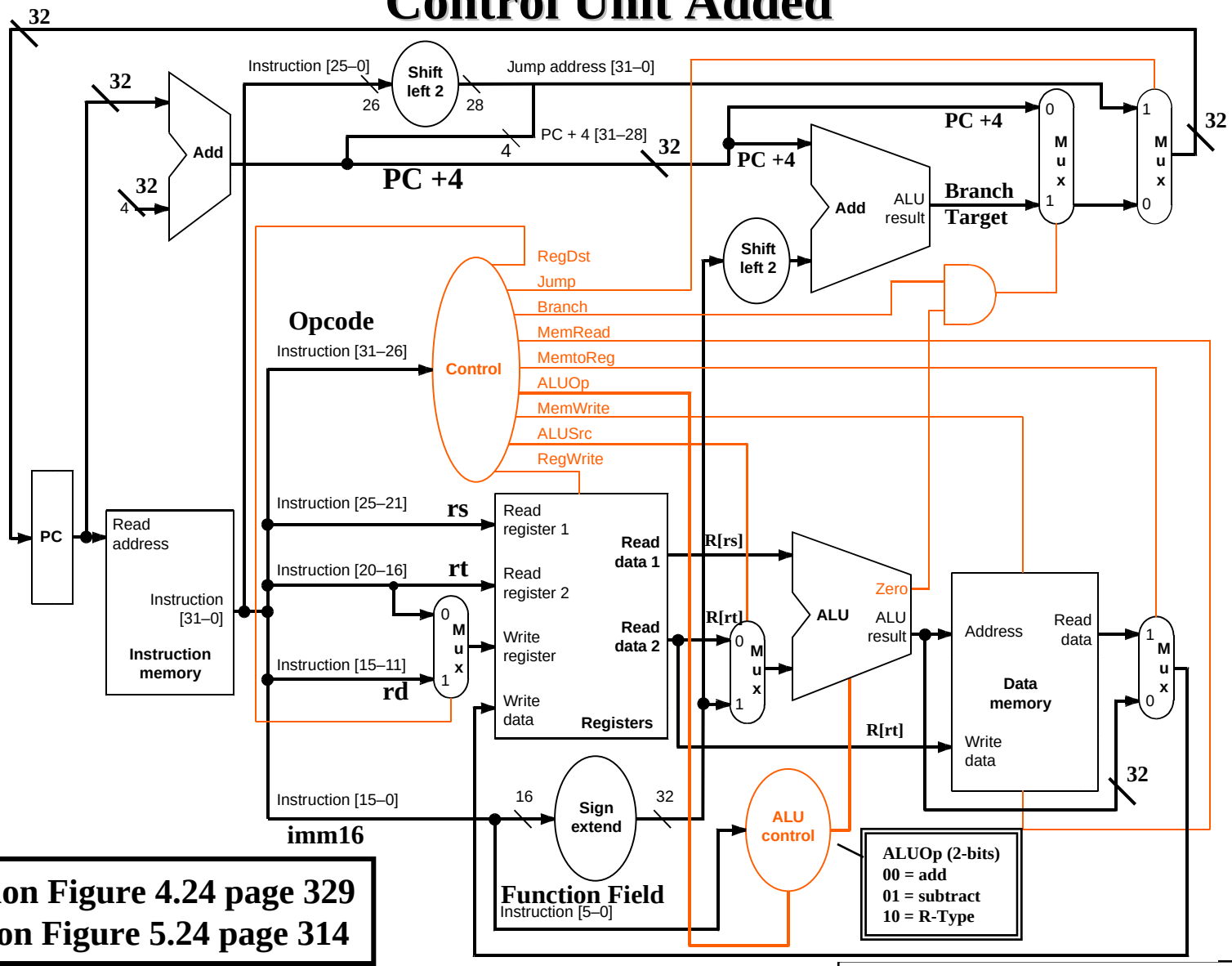
   – Microprogrammed. (i.e using a control program)

3rd Edition Chapter 5.5 – See Handout – Not in 4th Edition

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



Single Cycle MIPS Datapath Extended To Handle Jump with Control Unit Added



4th Edition Figure 4.24 page 329
3rd Edition Figure 5.24 page 314

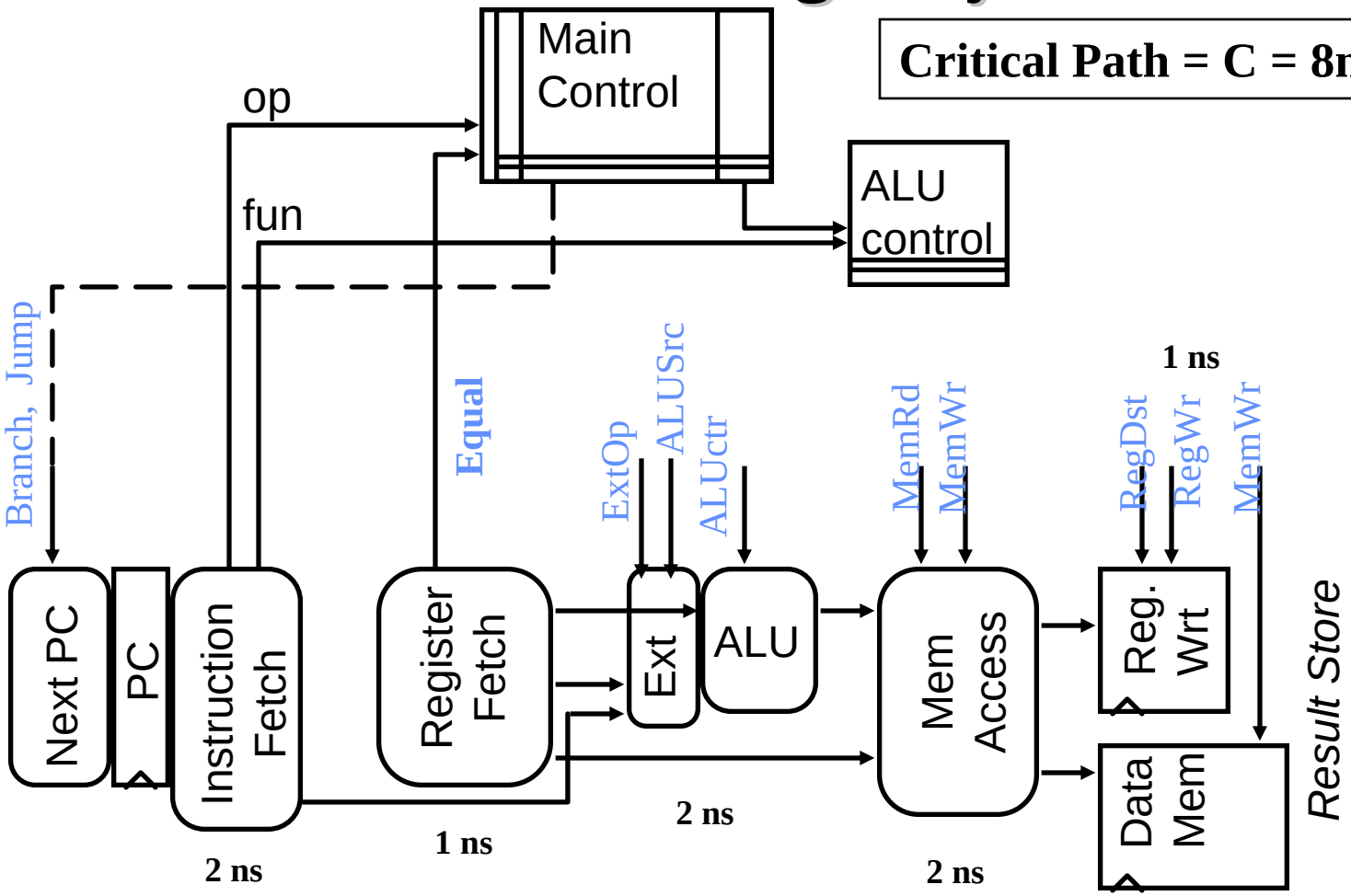
In this book version, ORI is not supported—no zero extend of immediate needed.

Drawbacks of Single-Cycle Processor

CPI = 1

1. 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.
2. Impossible to implement complex, variable-length instructions and complex addressing modes in a single cycle.
     - e.g indirect memory addressing.
3. High and duplicate hardware resource requirements
   - Any hardware functional unit cannot be used more than once in a single cycle (e.g. ALUs).
4. Cannot pipeline (overlap) the processing of one instruction with the previous instructions.
   - (instruction pipelining, 4th edition chapter 4 – 3rd edition ch. 6).

Abstract View of Single Cycle CPU



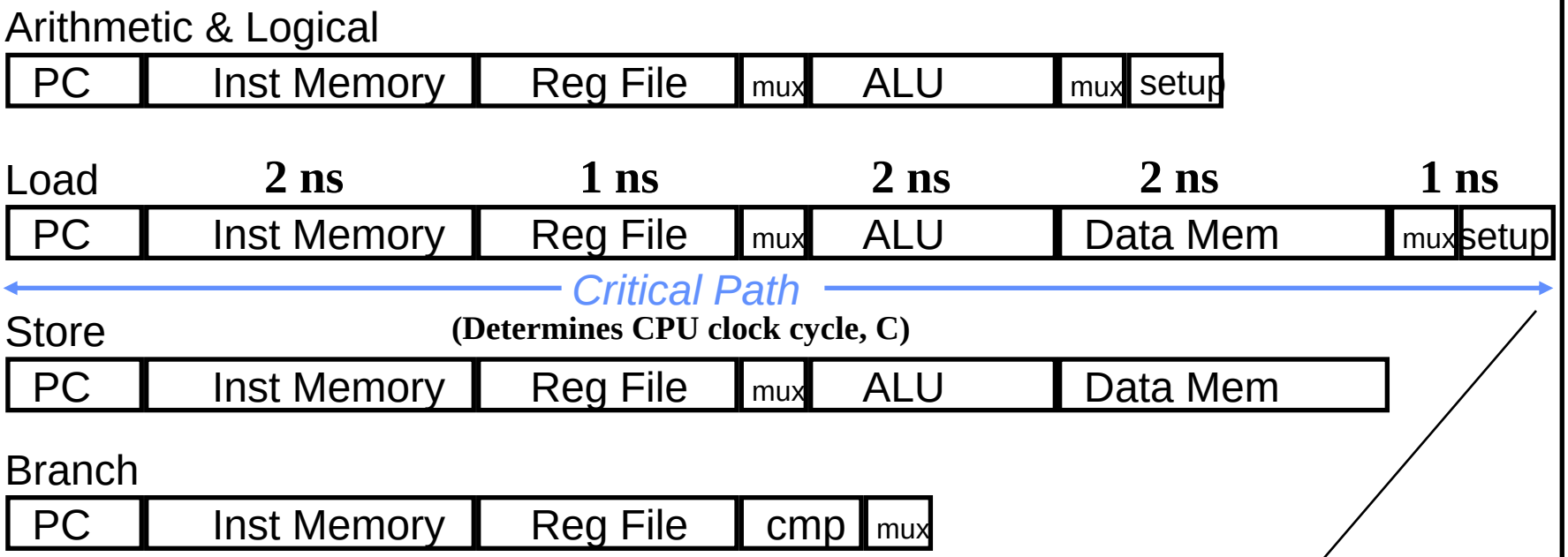
Critical Path = C = 8ns (LW)

One CPU Clock Cycle
Duration C = 8ns

One instruction per cycle CPI = 1

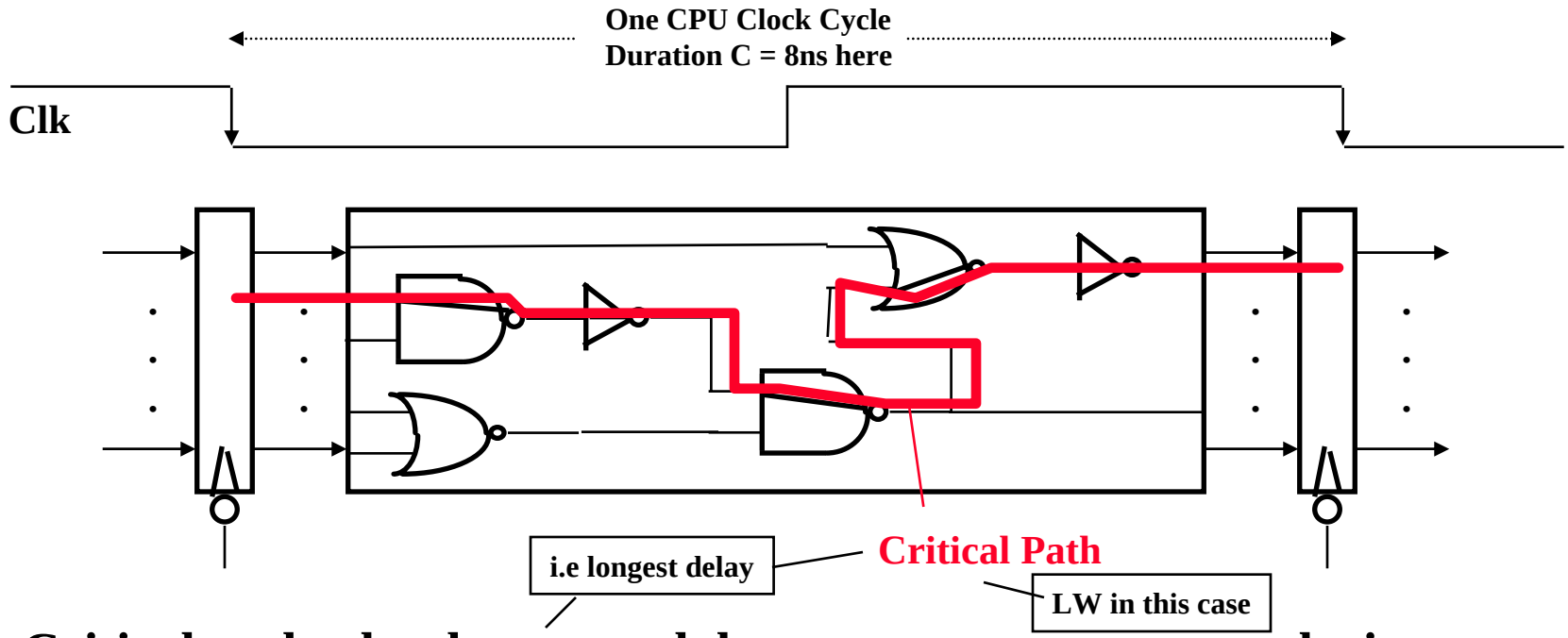
Assuming the following datapath/control hardware components delays:
Memory Units: 2 ns ALU and adders: 2 ns
Register File: 1 ns Control Unit < 1 ns

Single Cycle Instruction Timing



Critical Path: Load - LW (e.g C = 8 ns)

Clock Cycle Time & Critical Path

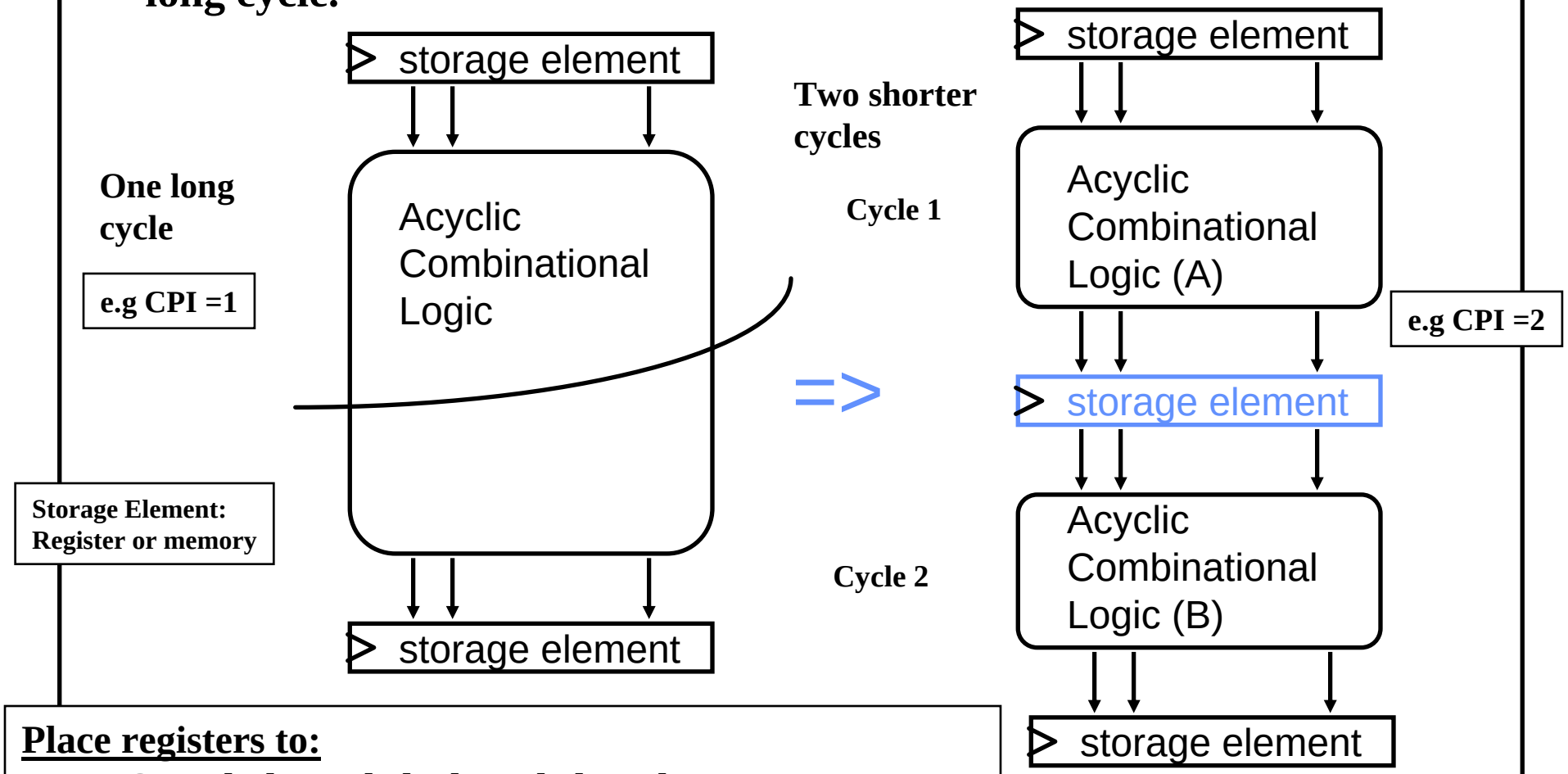


- **Critical path: the slowest path between any two storage devices**
- **Clock Cycle time is a function of the critical path, and must be greater than:**
  - **Clock-to-Q + Longest Delay Path through the Combination Logic + Setup + Clock Skew**

Assuming the following datapath/control hardware components delays:
Memory Units: 2 ns ALU and adders: 2 ns
Register File: 1 ns Control Unit < 1 ns

Reducing Cycle Time: Multi-Cycle Design

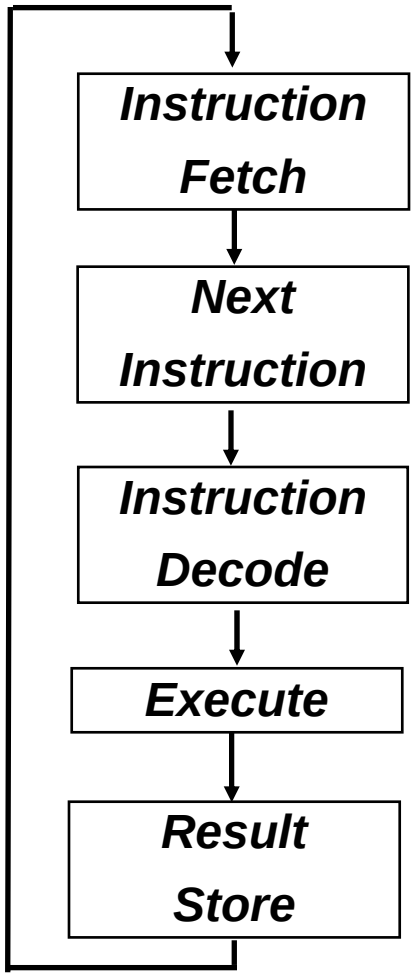
- **Cut combinational dependency graph by <u>inserting registers / latches.</u>**
- **The same work is done in two or more shorter cycles, rather than one long cycle.**



<u>Place registers to:</u>

- **Get a balanced clock cycle length**
- **Save any results needed for the remaining cycles**

Basic MIPS Instruction Processing Steps



Instruction Memory

Obtain instruction from program storage

Instruction ← Mem[PC]

Update program counter to address of next instruction

PC ← PC + 4

Common steps for all instructions

Determine instruction type

Obtain operands from registers

Done by Control Unit

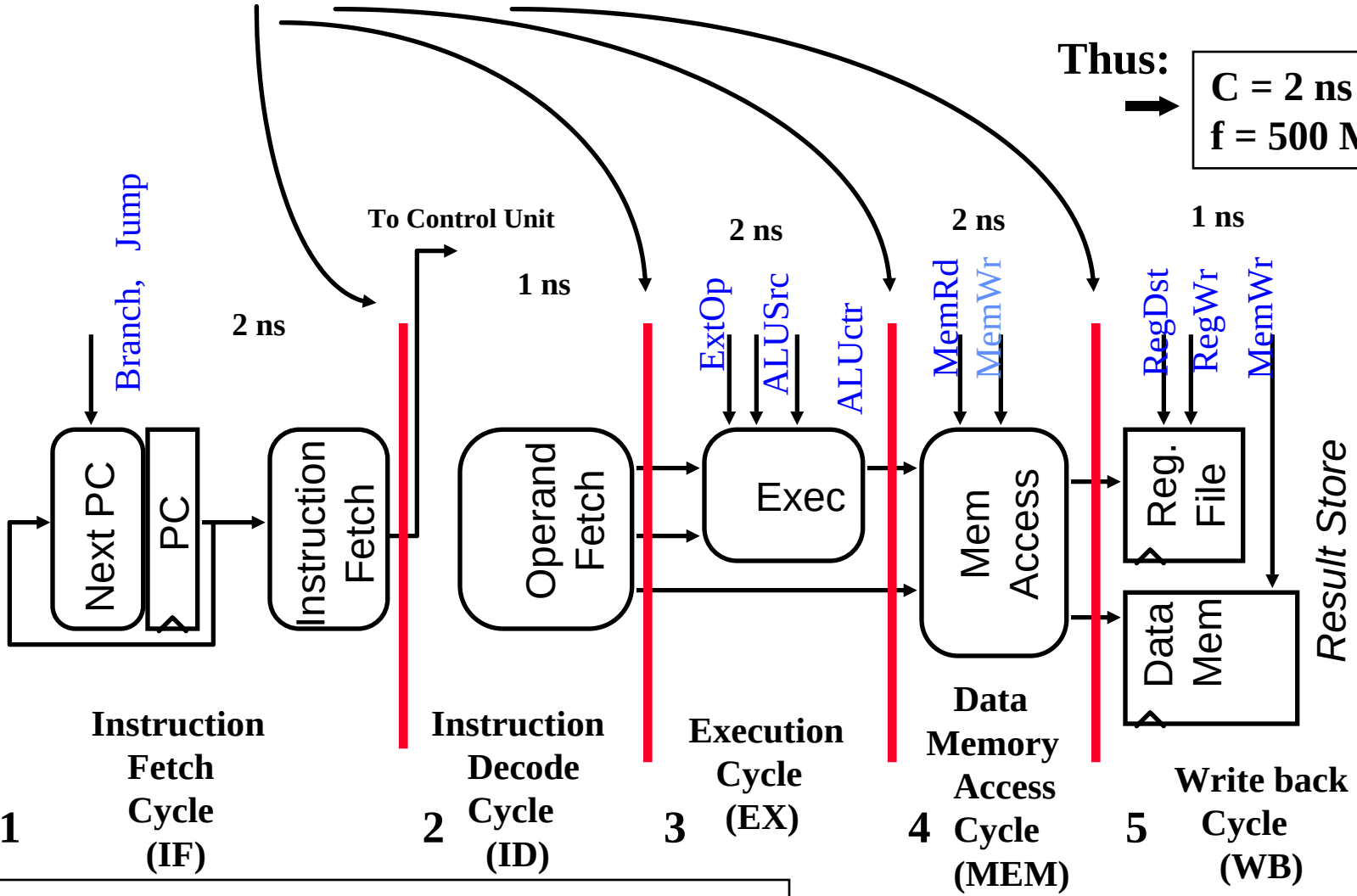
Compute result value or status

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

T = I x CPI x C

Partitioning The Single Cycle Datapath

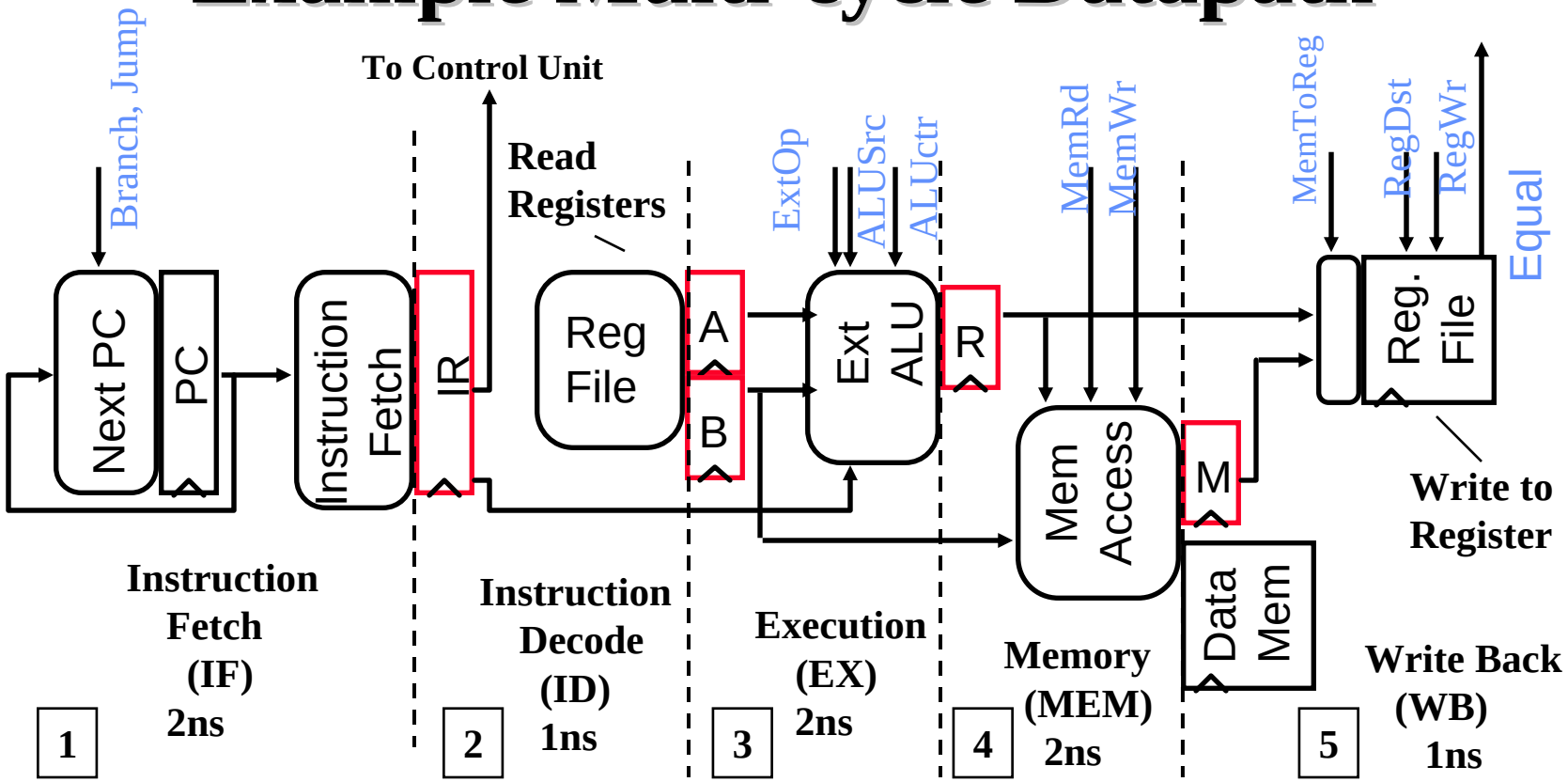
Add registers between steps to break into cycles



Place registers to:

- Get a balanced clock cycle length
- Save any results needed for the remaining cycles

Example Multi-cycle Datapath



Registers added: All clock-edge triggered (not shown register write enable control lines)

IR: Instruction register

A, B: Two registers to hold operands read from register file. i.e R[rs], R[rt]

R: or ALUOut, holds the output of the main ALU ALU result

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

CPU Clock Cycle Time: Worst cycle delay = C = 2ns (ignoring MUX, CLK-Q delays)

Assuming the following datapath/control hardware components delays:
Memory Units: 2 ns ALU and adders: 2 ns
Register File: 1 ns Control Unit < 1 ns

Thus Clock Rate:
f = 1 / 2ns = 500 MHz

Operations (Dependant RTN) for Each Cycle

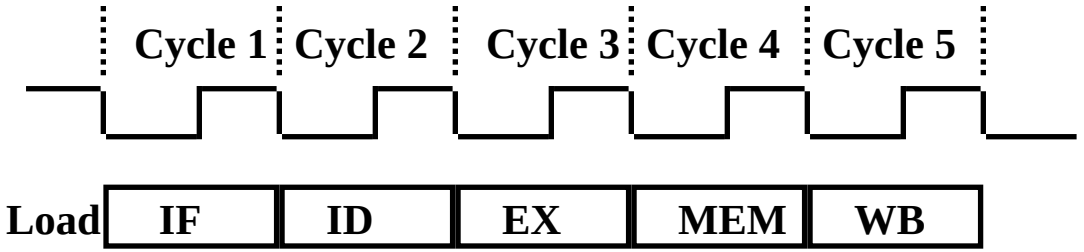
		R-Type	Logic Immediate	Load	Store	Branch
IF	Instruction Fetch	IR ← Mem[PC]	IR ← Mem[PC]	IR ← Mem[PC]	IR ← Mem[PC]	IR ← Mem[PC]
ID	Instruction Decode	A ← R[rs] B ← R[rt]	A ← R[rs] B ← R[rt	A ← R[rs] B ← R[rt	A ← R[rs] B ← R[rt]	A ← R[rs] B ← R[rt]
EX	Execution	R ← A funct B	R ← A OR ZeroExt[imm16]	R ← A + SignEx(Im16)	R ← A + SignEx(Im16)	Zero ← A - B If Zero = 1: PC ← PC + 4 + (SignExt(imm16) x4) else (i.e Zero =0): 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[rt] ← M PC ← PC + 4		

Instruction Fetch (IF) & Instruction Decode cycles are common for all instructions

MIPS Multi-Cycle Datapath:

Five Cycles of Load

CPI = 5



1- **Instruction Fetch (IF):**

Fetch the instruction from instruction Memory.

2- **Instruction Decode (ID):**

Operand Register 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 loaded data to the register file. Update PC.

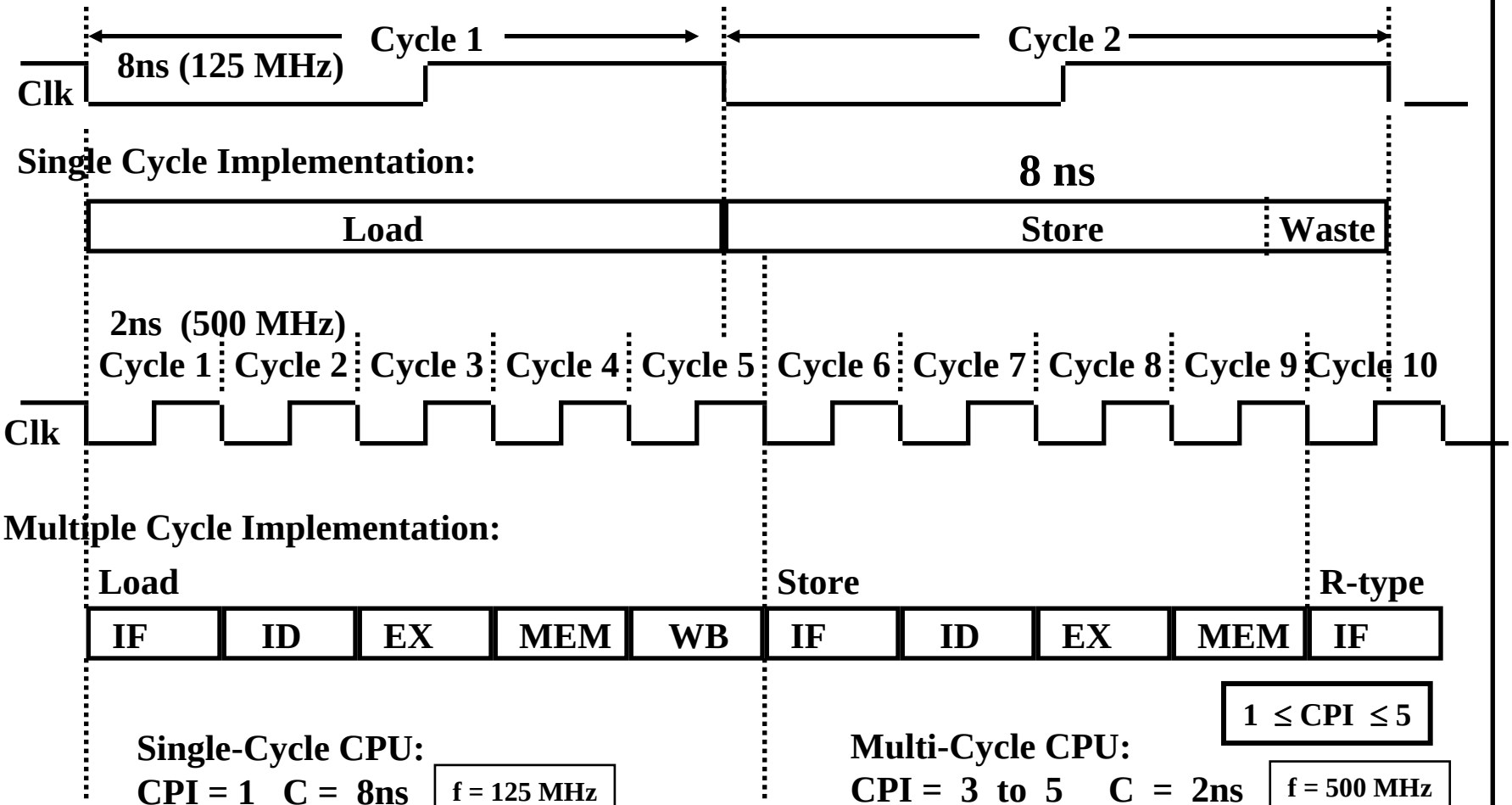
Multi-cycle Datapath Instruction CPI

- **R-Type/Immediate: Require four cycles, CPI = 4**
  - **IF, ID, EX, WB**
- **Loads: Require five cycles, CPI = 5**
  - **IF, ID, EX, MEM, WB**
- **Stores: Require four cycles, CPI = 4**
  - **IF, ID, EX, MEM**
- **Branches/Jumps: Require three cycles, CPI = 3**
  - **IF, ID, EX**

- **Average or effective program CPI: $3 \leq CPI \leq 5$ depending on program profile (instruction mix).**

C = 2 ns f = 500 MHz

Single Cycle Vs. Multi-Cycle CPU



$1 \leq CPI \leq 5$

Single-Cycle CPU:
CPI = 1 C = 8ns f = 125 MHz
One million instructions take =
$I \times CPI \times C = 10^6 \times 1 \times 8 \times 10^{-9} = 8$ msec

Multi-Cycle CPU:
CPI = 3 to 5 C = 2ns f = 500 MHz
One million instructions take from
$10^6 \times 3 \times 2 \times 10^{-9} = 6$ msec
to $10^6 \times 5 \times 2 \times 10^{-9} = 10$ msec
depending on instruction mix used.

$T = I \times CPI \times C$

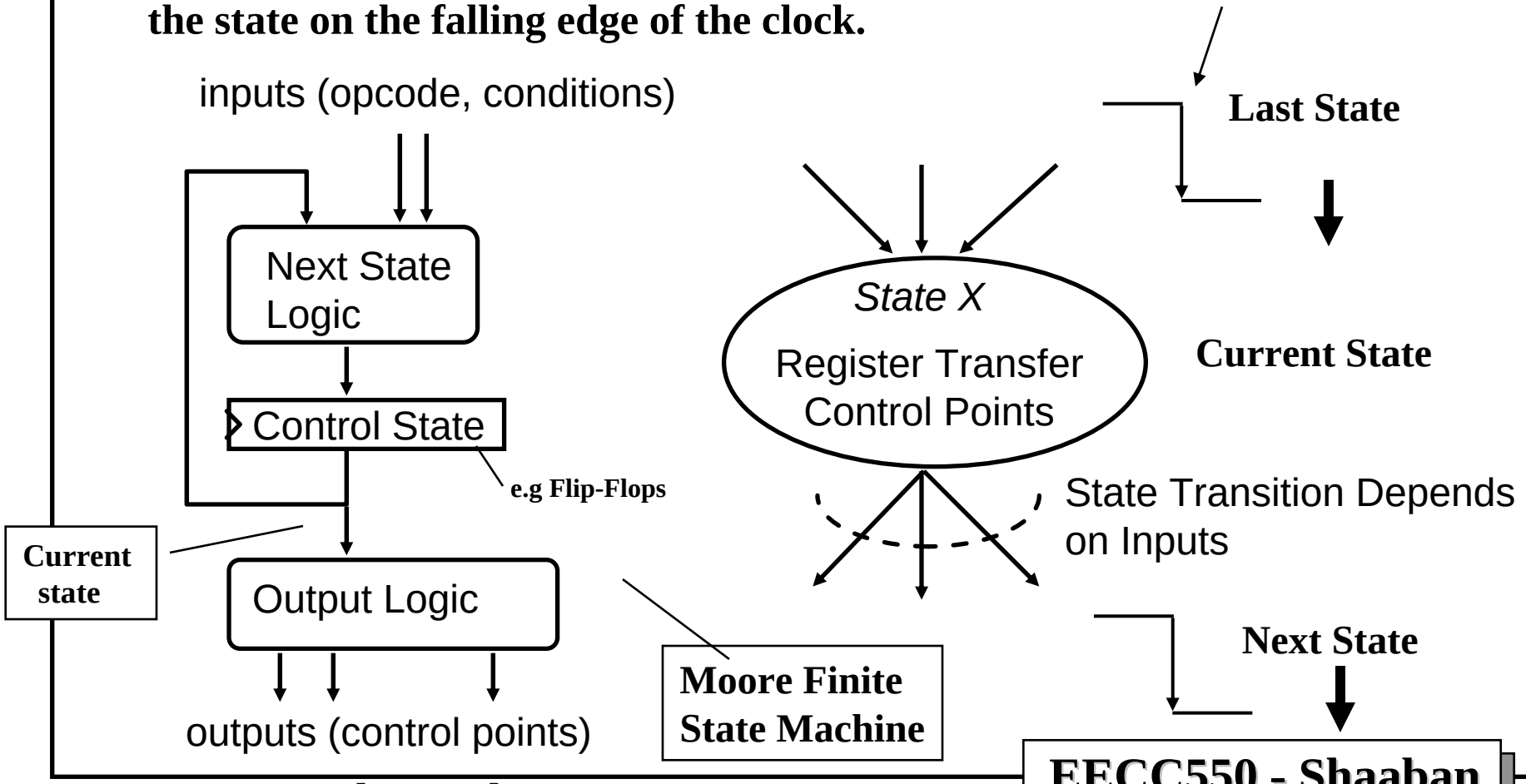
Assuming the following datapath/control hardware components delays:
Memory Units: 2 ns ALU and adders: 2 ns
Register File: 1 ns Control Unit < 1 ns

Control Unit Design:

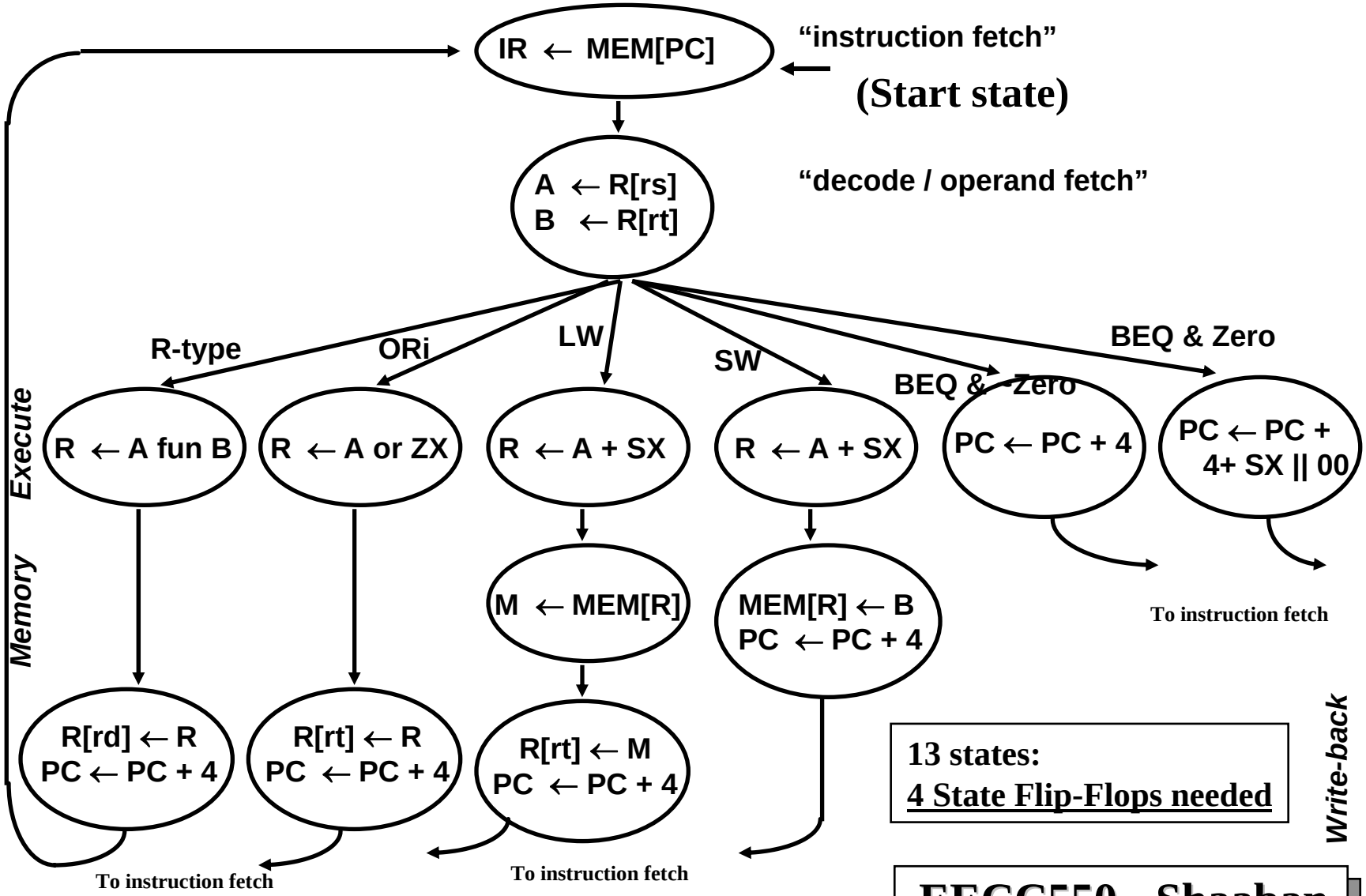
Finite State Machine (FSM) Control Model

AKA Hardwired Control

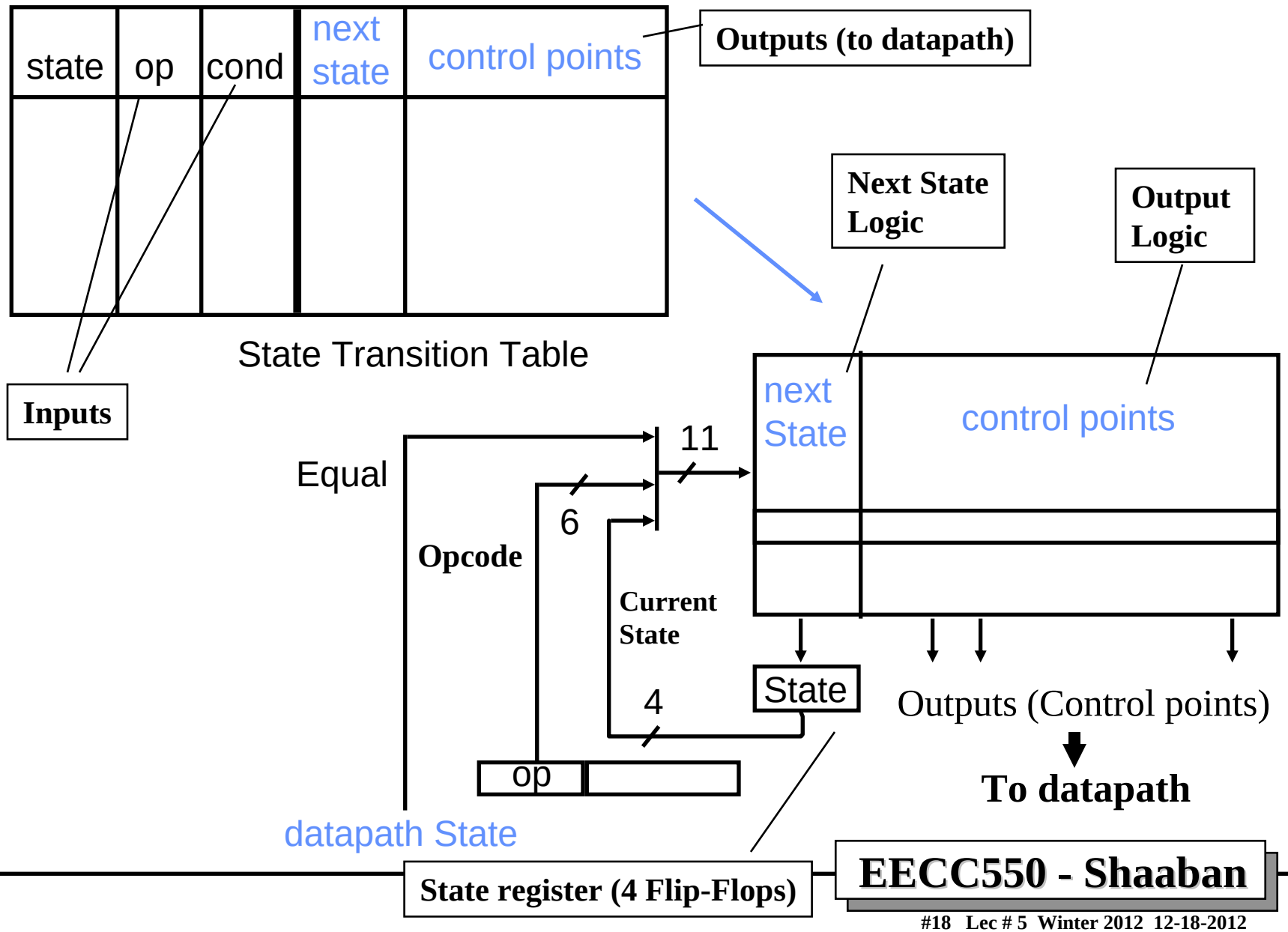
- **State specifies control points (outputs) for Register Transfer.**
- **Control points (outputs) are assumed to depend only on the current state and not inputs (i.e. Moore finite state machine)**
- **Transfer (register/memory writes) and state transition occur upon exiting the state on the falling edge of the clock.**



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



Traditional FSM Controller



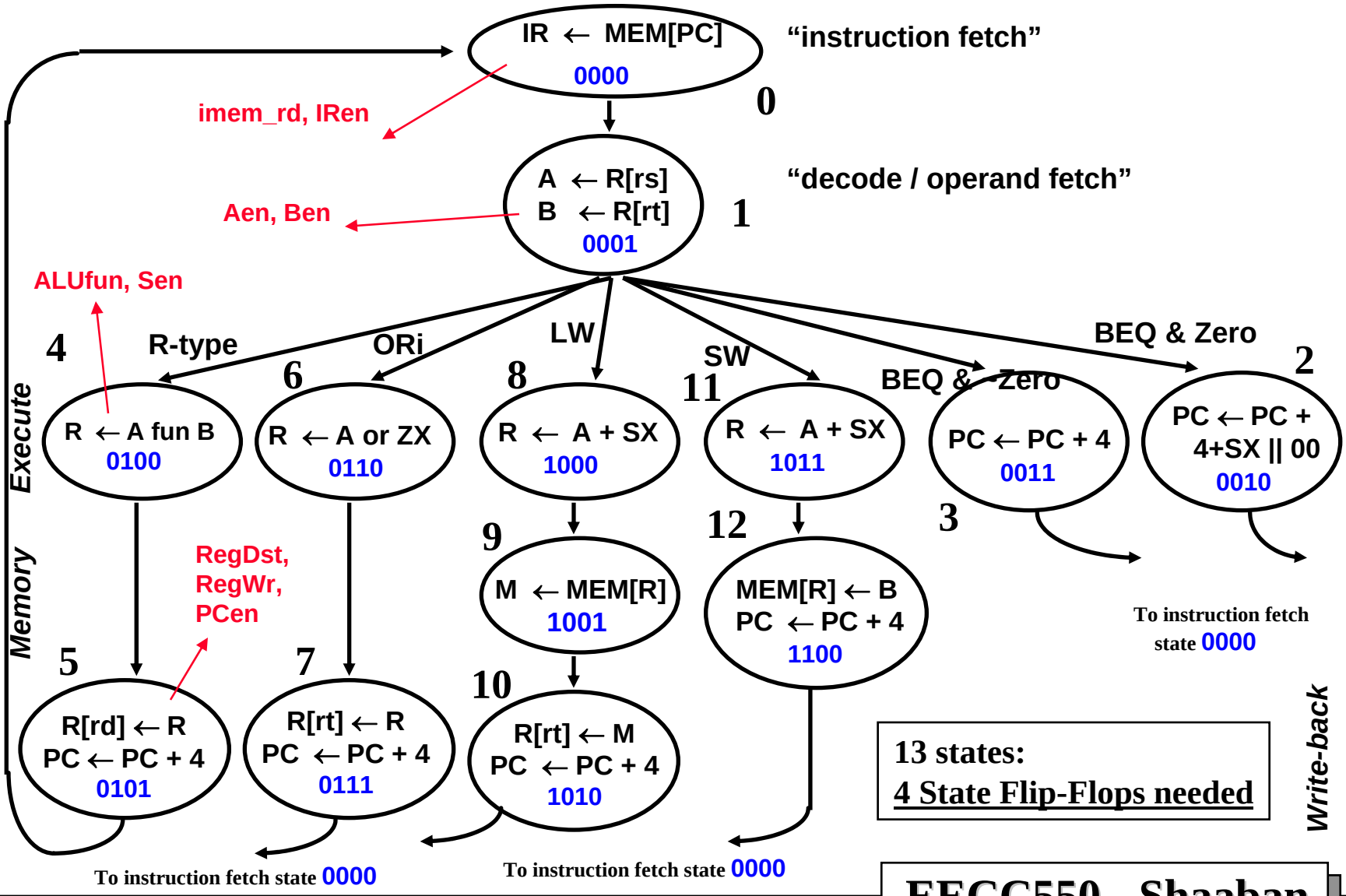
Traditional FSM Controller

datapath + state diagram => control

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

More on FSM controller implementation in Appendix C

Mapping RTNs To Control Points Examples & State Assignments



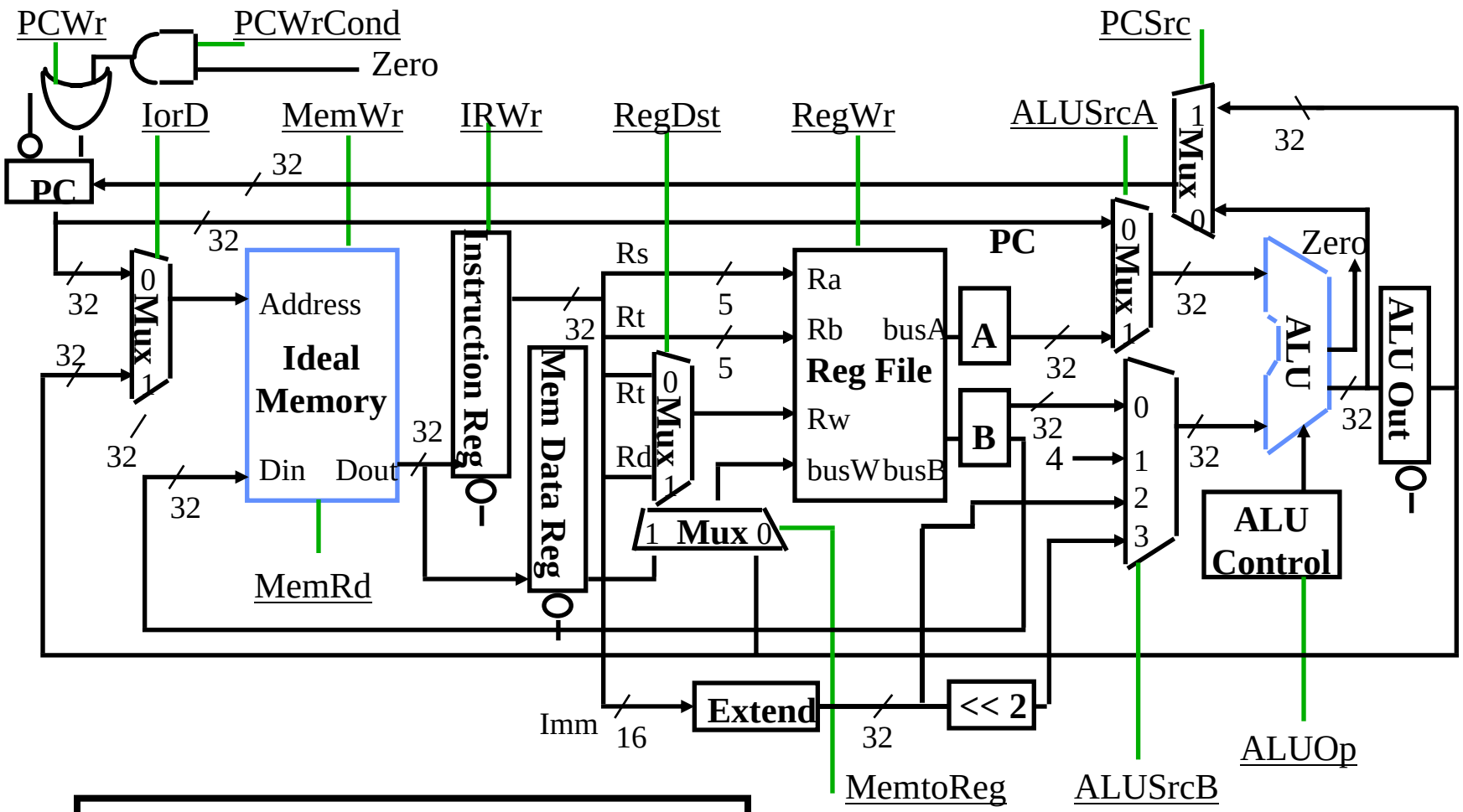
Detailed Control Specification – (Partial) State Transition Table

	Current State	Op field	Z	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
IF	0000	??????	?	0001	1														
ID	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	Can be combined in one state											
	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	1			
	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				

More on FSM controller implementation in Appendix C

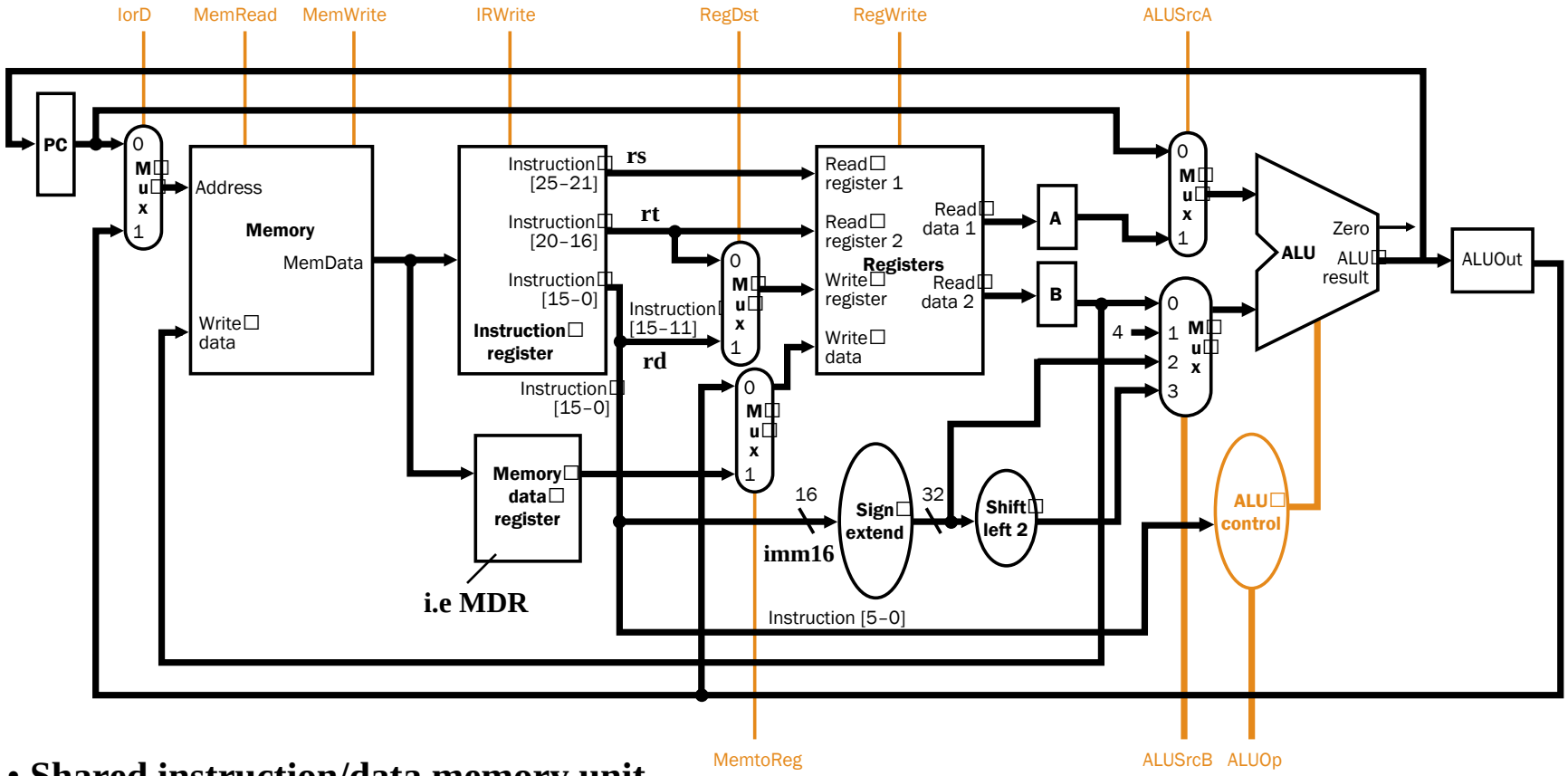
Alternative Multiple Cycle Datapath (In Textbook)

- **Minimizes Hardware: 1 memory, 1 ALU**



3rd Edition Chapter 5.5 (see handout) – Not in 4th Edition

Alternative Multiple Cycle Datapath (In Textbook)

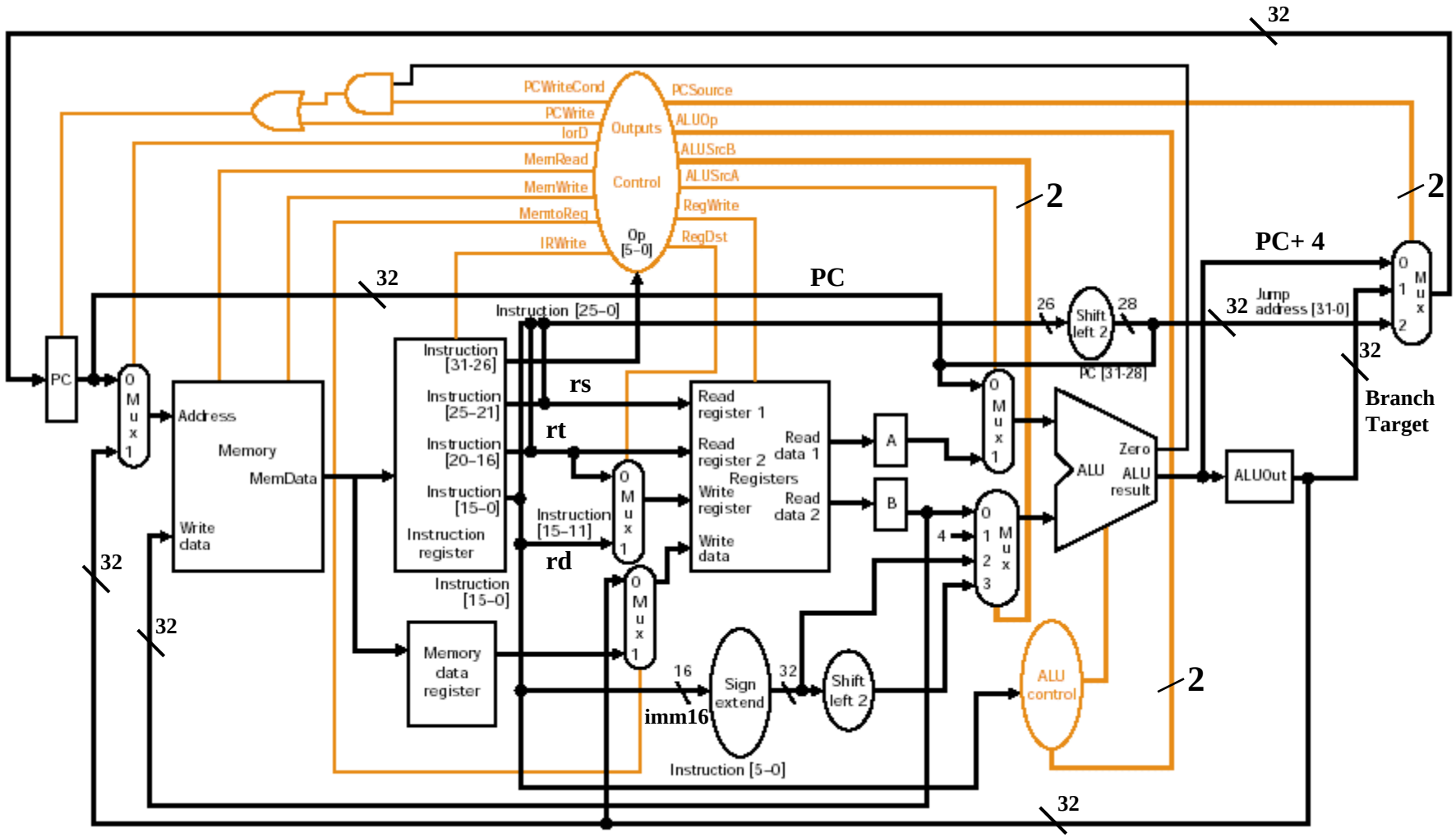


- **Shared instruction/data memory unit**
- **A single ALU shared among instructions**
- **Shared units require additional or widened multiplexors**
- **Temporary registers to hold data between clock cycles of the instruction:**
  - **Additional registers:**

    Instruction Register (IR), Memory Data Register (MDR), A, B, ALUOut

(Figure 5.27 page 322)

Alternative Multiple Cycle Datapath With Control Lines (Fig 5.28 In Textbook)



(ORI not supported, Jump supported)

(Figure 5.28 page 323)

The Effect of The 1-bit Control Signals

Signal Name	Effect when deasserted (=0)	Effect when asserted (=1)
RegDst	The register destination number for the write register comes from the rt field (instruction bits 20:16).	The register destination number for the write register comes from the rd field (instruction bits 15:11).
RegWrite	None	The register on the write register input is written with the value on the Write data input.
ALUSrcA	The first ALU operand is the PC	The First ALU operand is register A (i.e R[rs])
MemRead	None	Content of memory specified by the address input are put on the memory data output.
MemWrite	None	Memory contents specified by the address input is replaced by the value on the Write data input.
MemtoReg	The value fed to the register write data input comes from ALUOut register.	The value fed to the register write data input comes from data memory register (MDR).
IorD	The PC is used to supply the address to the memory unit.	The ALUOut register is used to supply the the address to the memory unit.
IRWrite	None	The output of the memory is written into Instruction Register (IR)
PCWrite	None	The PC is written; the source is controlled by PCSource
PCWriteCond i.e. Branch	None	The PC is written if the Zero output of the ALU is also active.

(Figure 5.29 page 324)

The Effect of The 2-bit Control Signals

Signal Name	Value (Binary)	Effect
ALUOp	00	The ALU performs an add operation
	01	The ALU performs a subtract operation
	10	The funct field of the instruction determines the ALU operation (R-Type)
ALUSrcB	00	The second input of the ALU comes from register B (i.e R[rs])
	01	The second input of the ALU is the constant 4
	10	The second input of the ALU is the sign-extended 16-bit immediate (imm16) field of the instruction in IR
	11	The second input of the ALU is is the sign-extended 16-bit immediate field of IR shifted left 2 bits (for branches)
PCSource	00	Output of the ALU (PC+4) is sent to the PC for writing
	01	The content of ALUOut (the branch target address) is sent to the PC for writing
	10	The jump target address (IR[25:0] shifted left 2 bits and concatenated with PC+4[31:28] is sent to the PC for writing i.e jump address

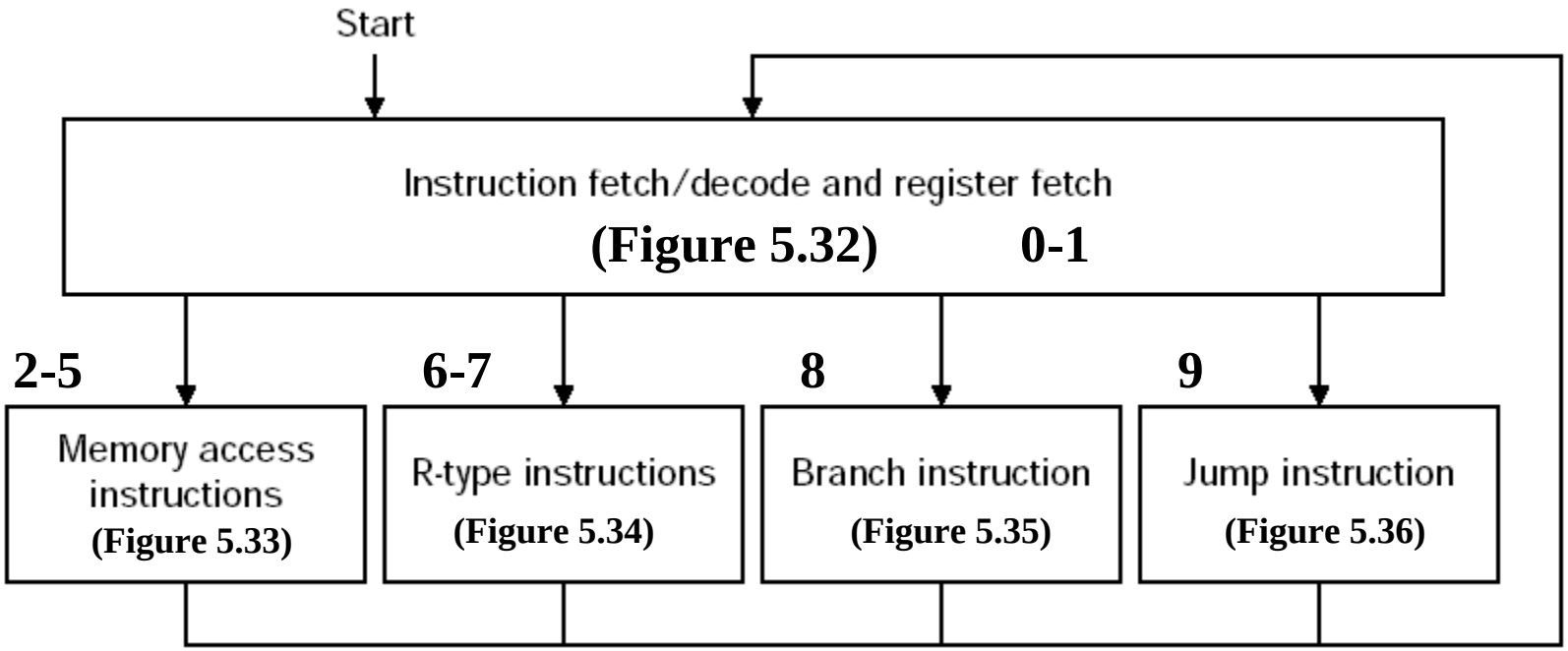
(Figure 5.29 page 324)

Operations (Dependant RTN) for Each Cycle

		R-Type	Load	Store	Branch	Jump
IF	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
ID	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)
EX	Execution	ALUout ← A funct B	ALUout ← A + SignEx(Imm16)	ALUout ← A + SignEx(Imm16)	Zero ← A - B Zero: PC ← ALUout	PC ← Jump Address
MEM	Memory		MDR ← Mem[ALUout]	Mem[ALUout] ← B		
WB	Write Back	R[rd] ← ALUout	R[rt] ← MDR			

Instruction Fetch (IF) & Instruction Decode (ID) cycles are common for all instructions

High-Level View of Finite State Machine Control

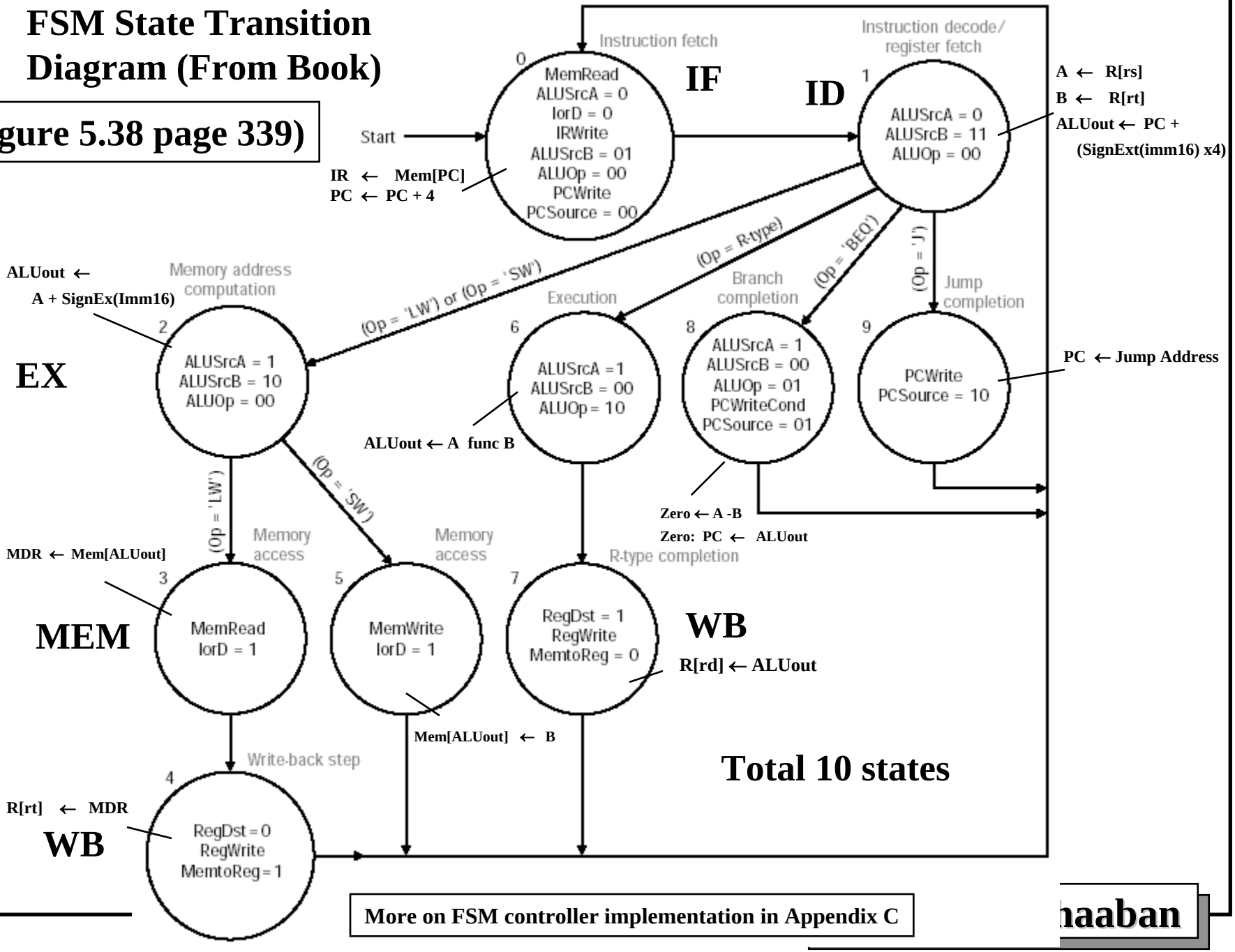


- First steps are independent of the instruction class
- Then a series of sequences that depend on the instruction opcode
- Then the control returns to fetch a new instruction.
- Each box above represents one or several state.

(Figure 5.31 page 332)

FSM State Transition Diagram (From Book)

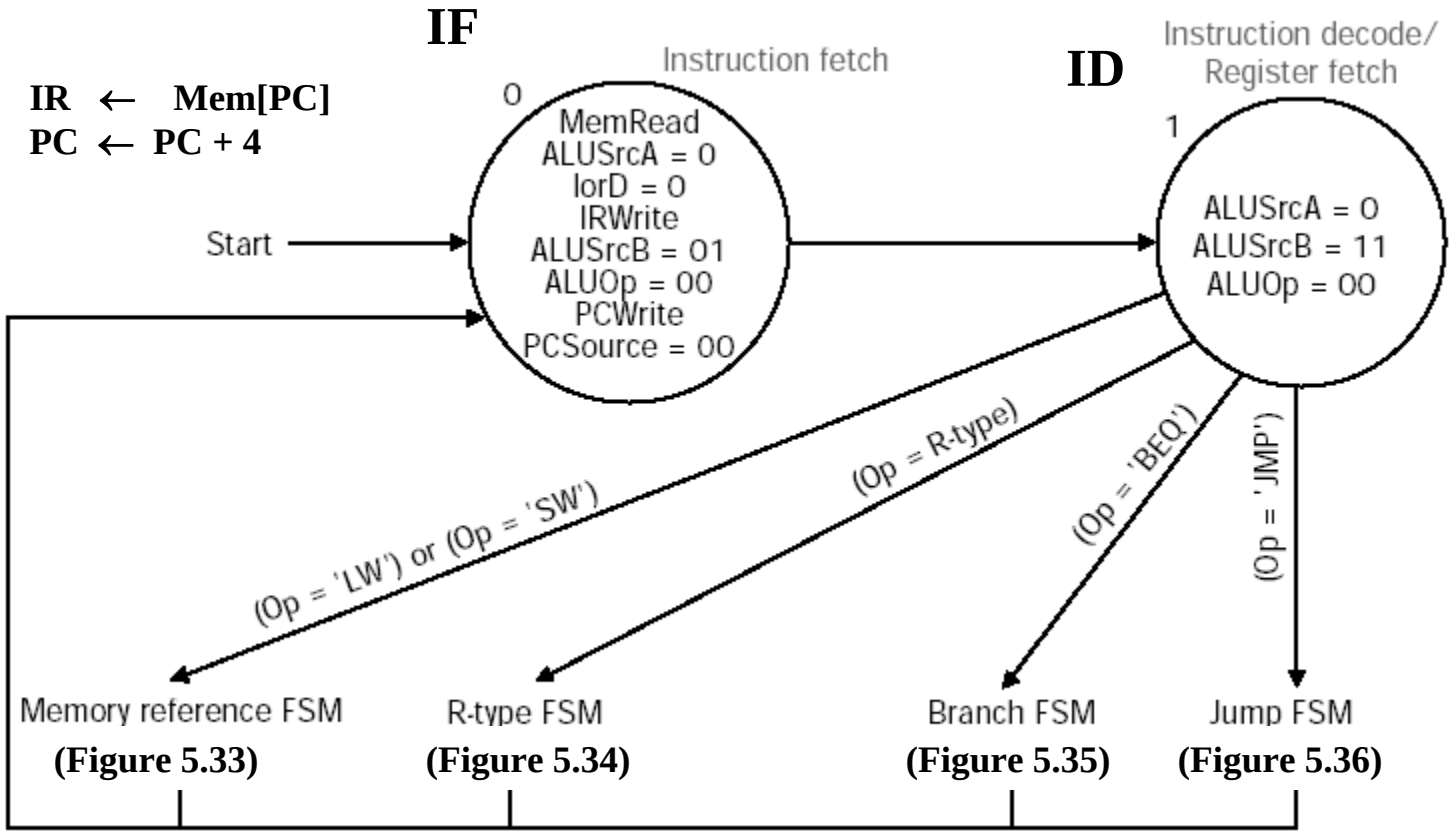
(Figure 5.38 page 339)



More on FSM controller implementation in Appendix C

Instruction Fetch (IF) and Decode (ID) FSM States

A ← R[rs]
B ← R[rt]
ALUout ← PC + (SignExt(imm16) x4)

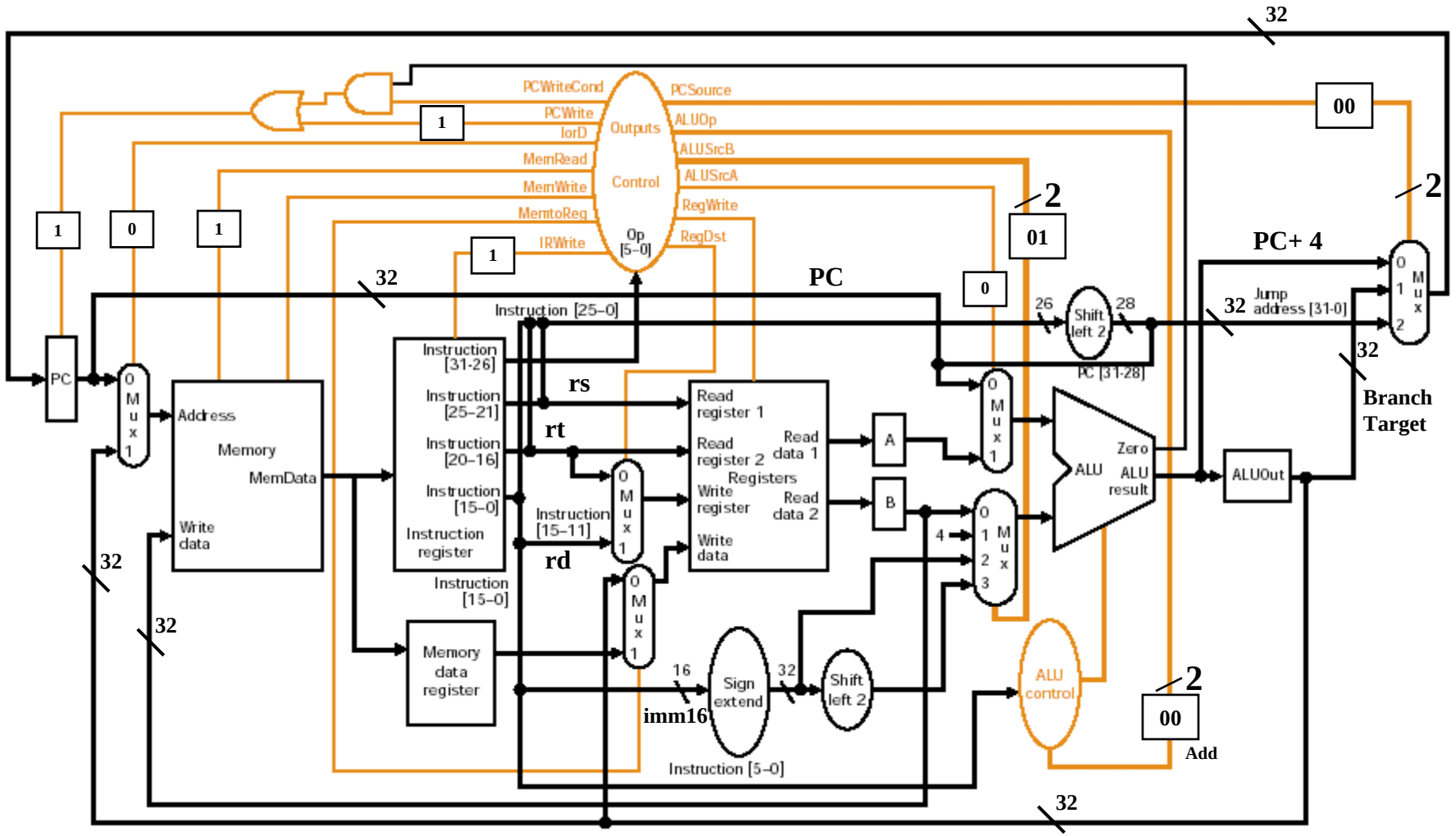


(Figure 5.32 page 333)

Instruction Fetch (IF) Cycle (State 0)

IR ← Mem[PC]
PC ← PC + 4

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



(ORI not supported, Jump supported)

(Figure 5.28 page 323)

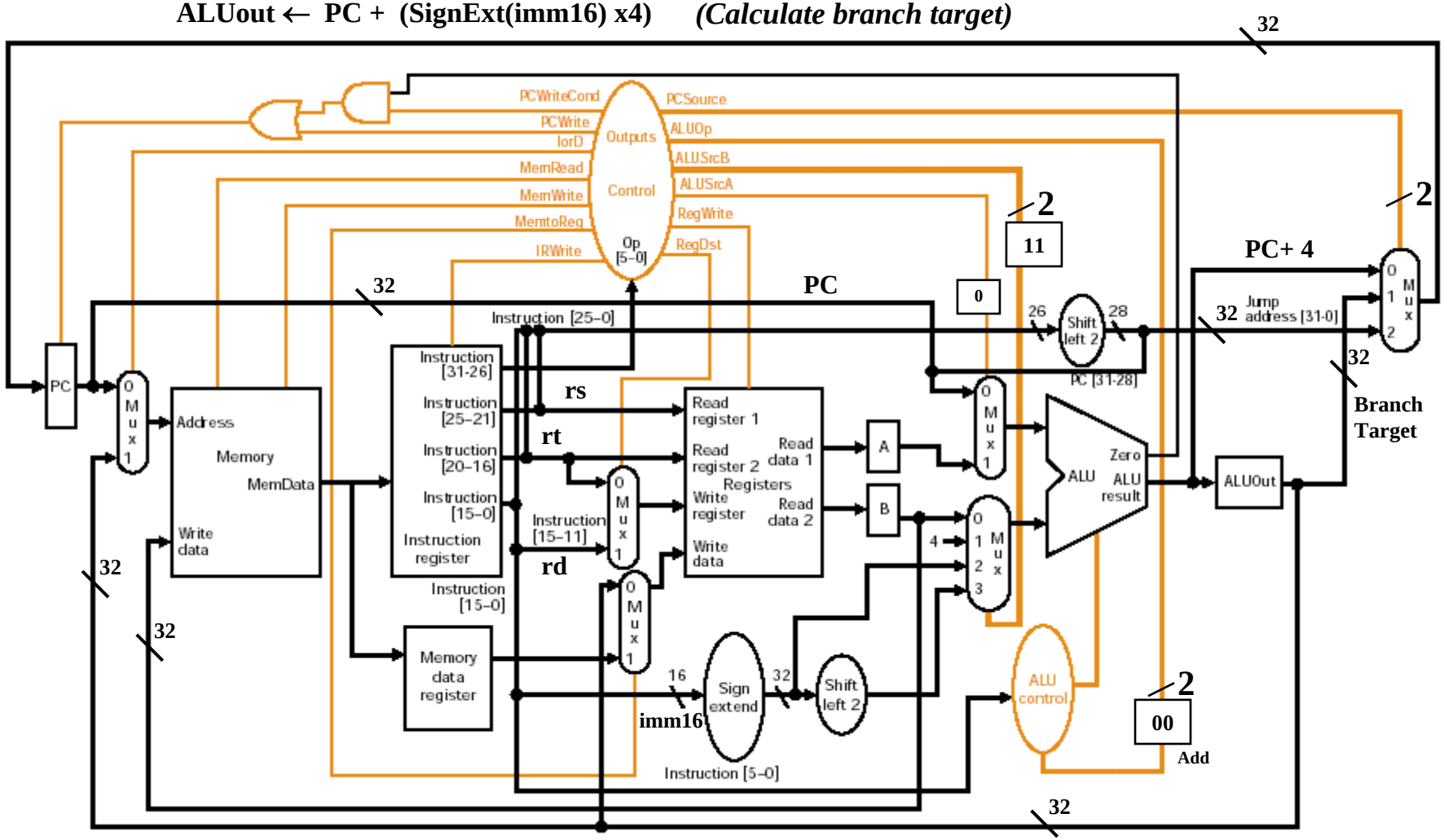
Instruction Decode (ID) Cycle (State 1)

A ← R[rs]

B ← R[rt]

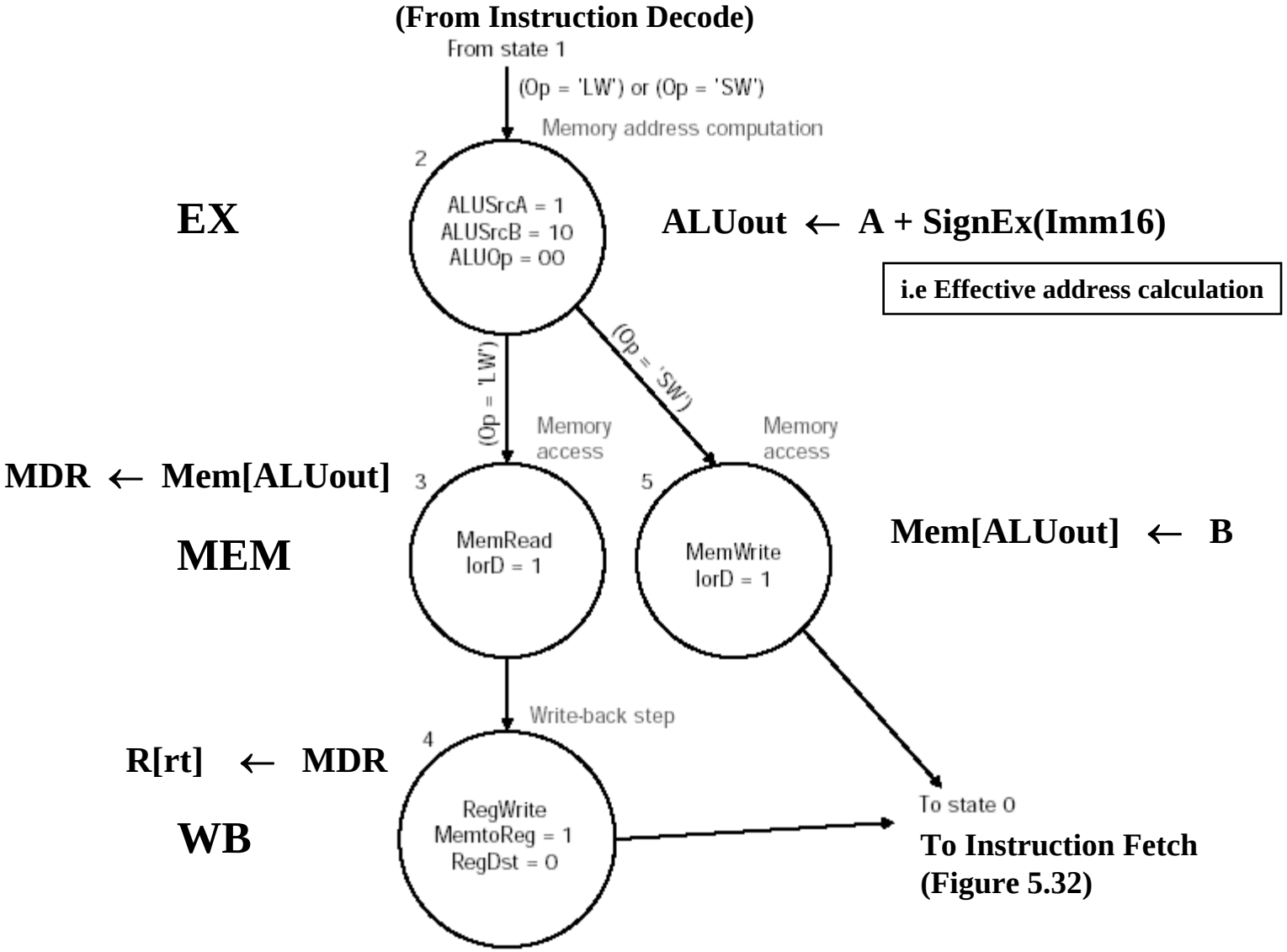
ALUSrcA = 0	ALUSrcB = 11	ALUOp = 00 (add)

ALUout ← PC + (SignExt(imm16) x4) *(Calculate branch target)*



(ORI not supported, Jump supported)

Load/Store Instructions FSM States



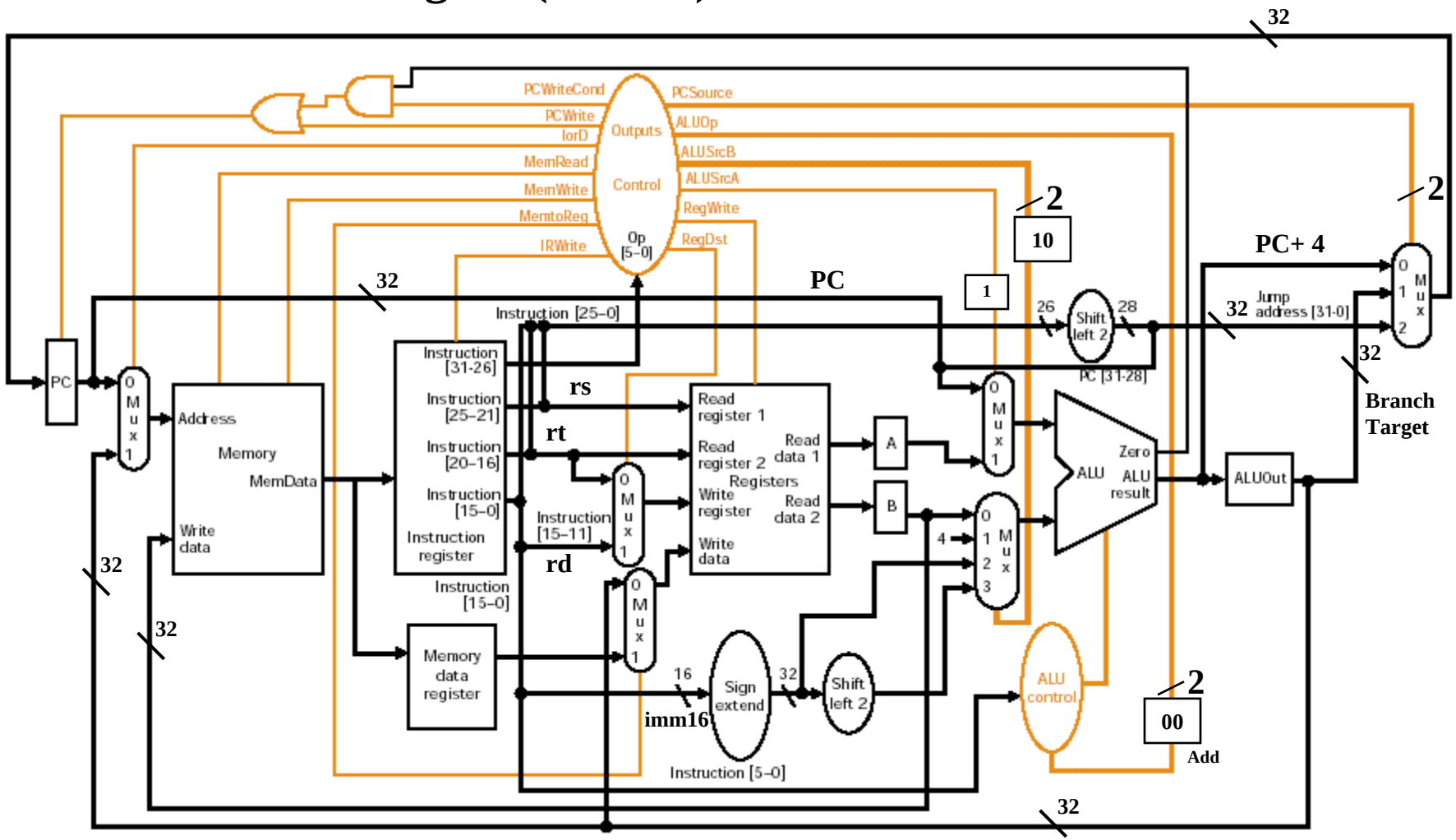
(Figure 5.33 page 334)

Load/Store Execution (EX) Cycle (State 2)

Effective address calculation

ALUout ← A + SignEx(Imm16)

ALUSrcA = 1 **ALUSrcB = 10**
ALUOp = 00 (add)

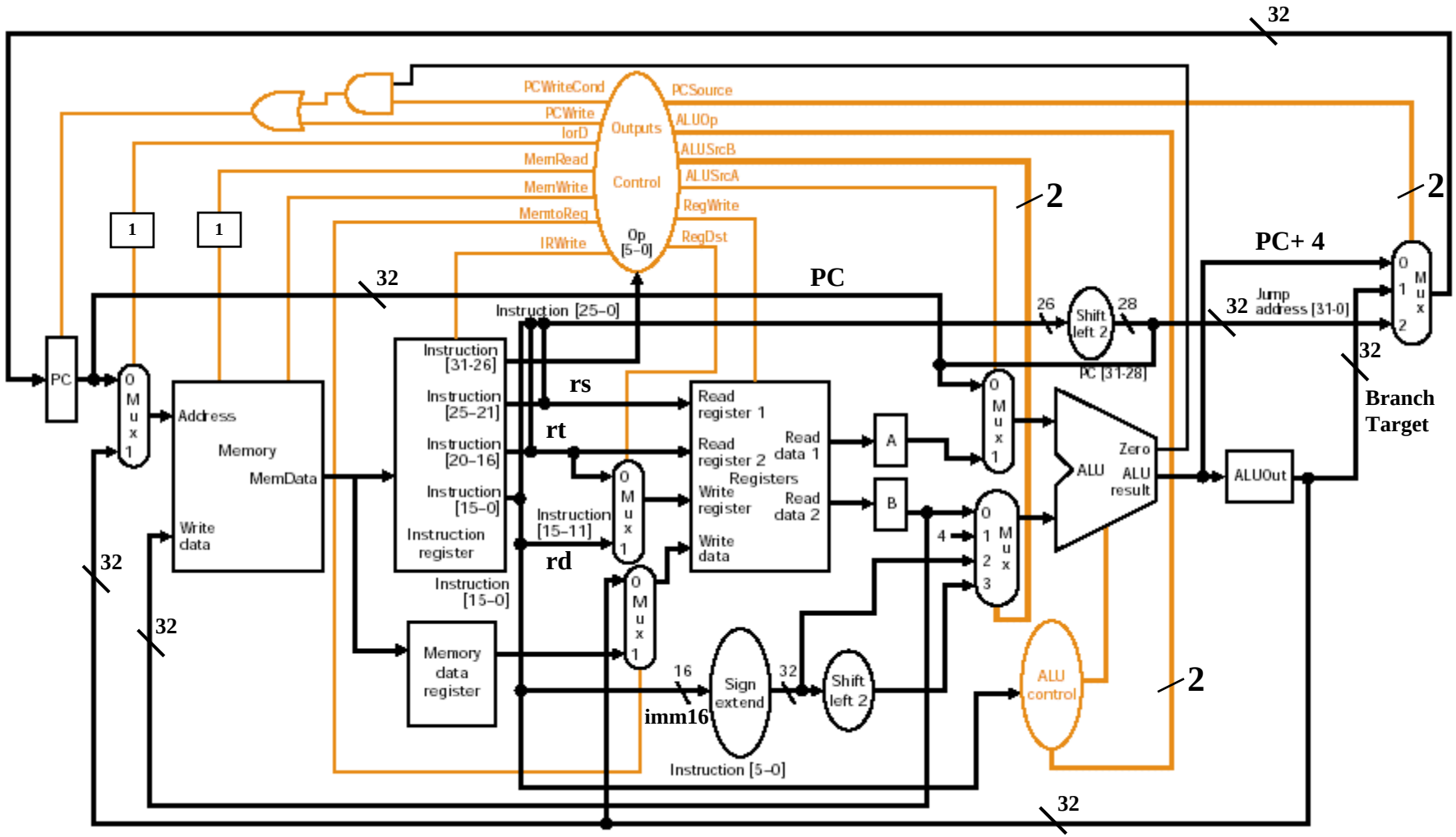


(ORI not supported, Jump supported)

Load Memory (MEM) Cycle (State 3)

MDR ← Mem[ALUout]

MemRead = 1 IorD = 1



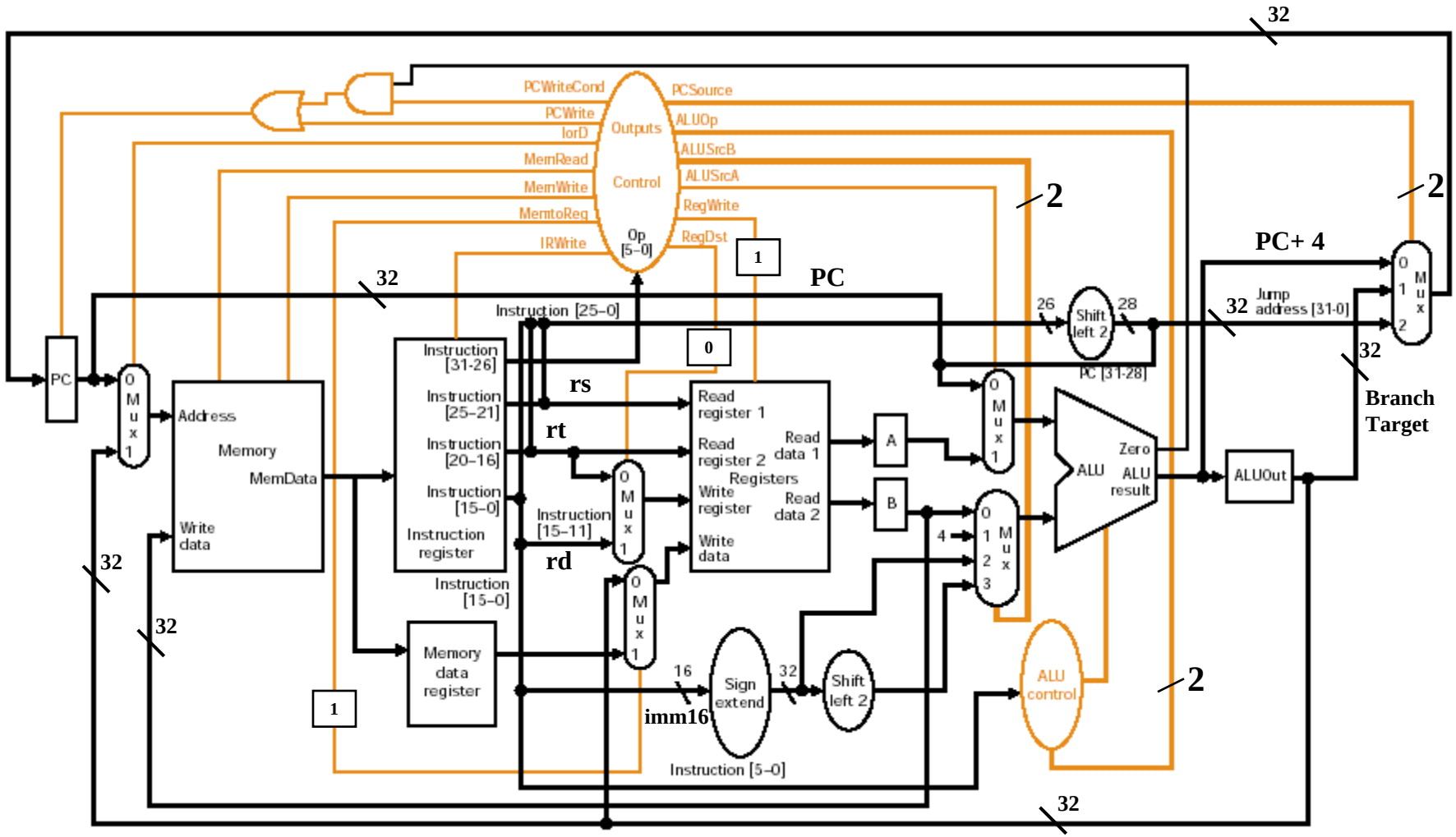
(ORI not supported, Jump supported)

(Figure 5.28 page 323)

Load Write Back (WB) Cycle (State 4)

R[rt] ← MDR

RegWrite = 1 MemtoReg = 1 RegDst = 0



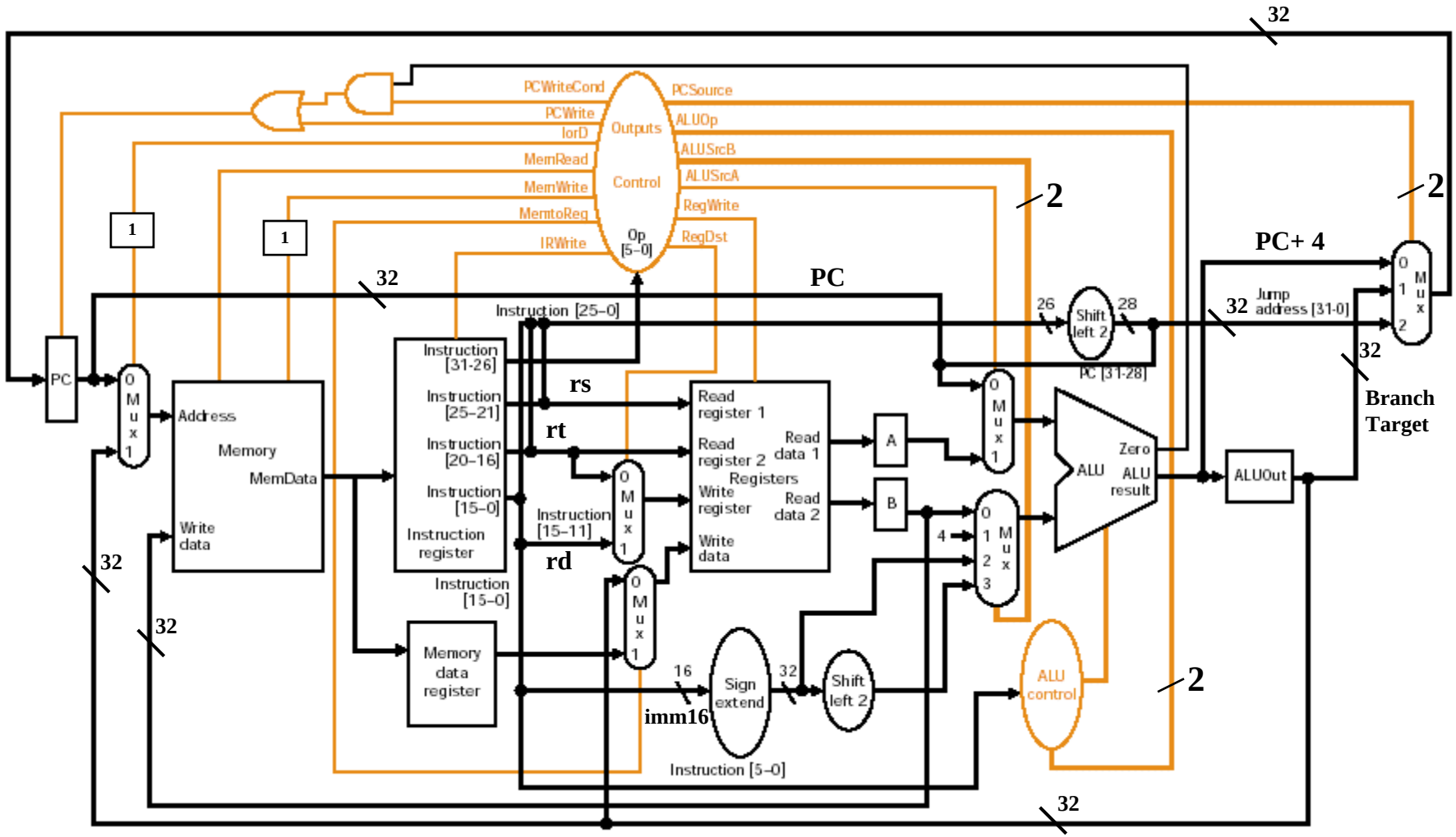
(ORI not supported, Jump supported)

(Figure 5.28 page 323)

Store Memory (MEM) Cycle (State 5)

Mem[ALUout] ← B

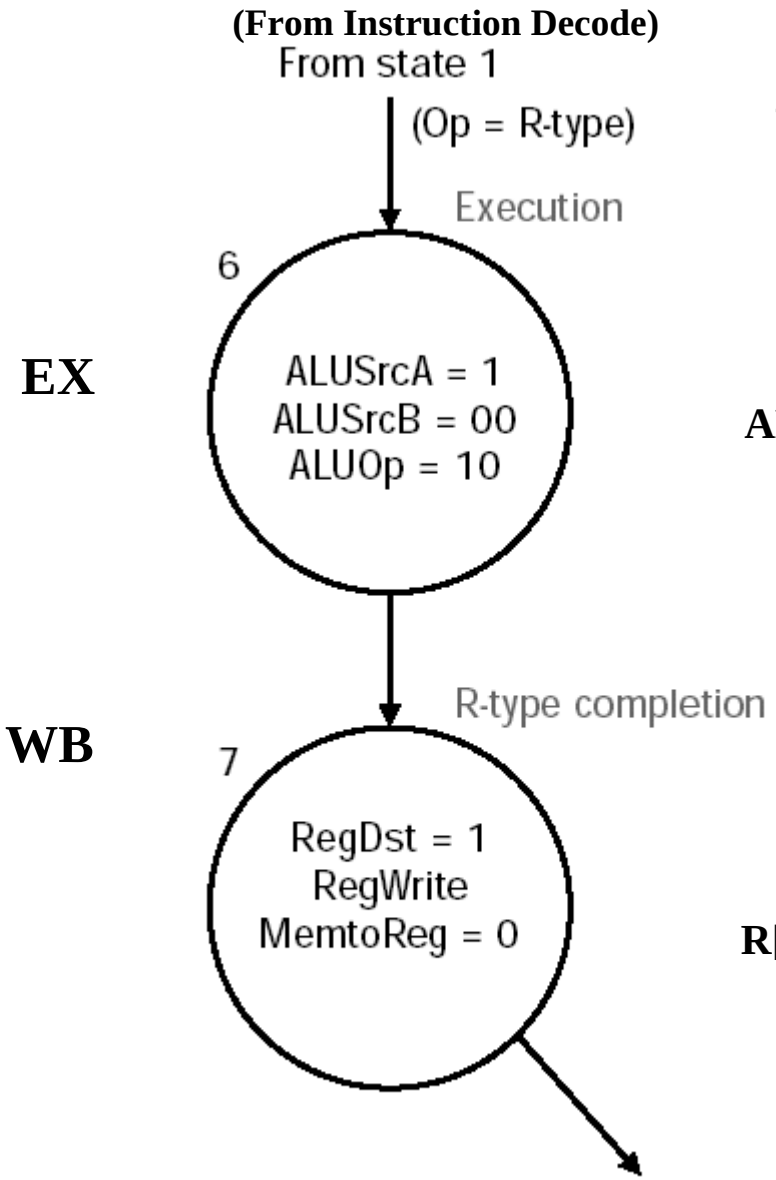
MemWrite = 1 IorD = 1



(ORI not supported, Jump supported)

(Figure 5.28 page 323)

R-Type Instructions FSM States



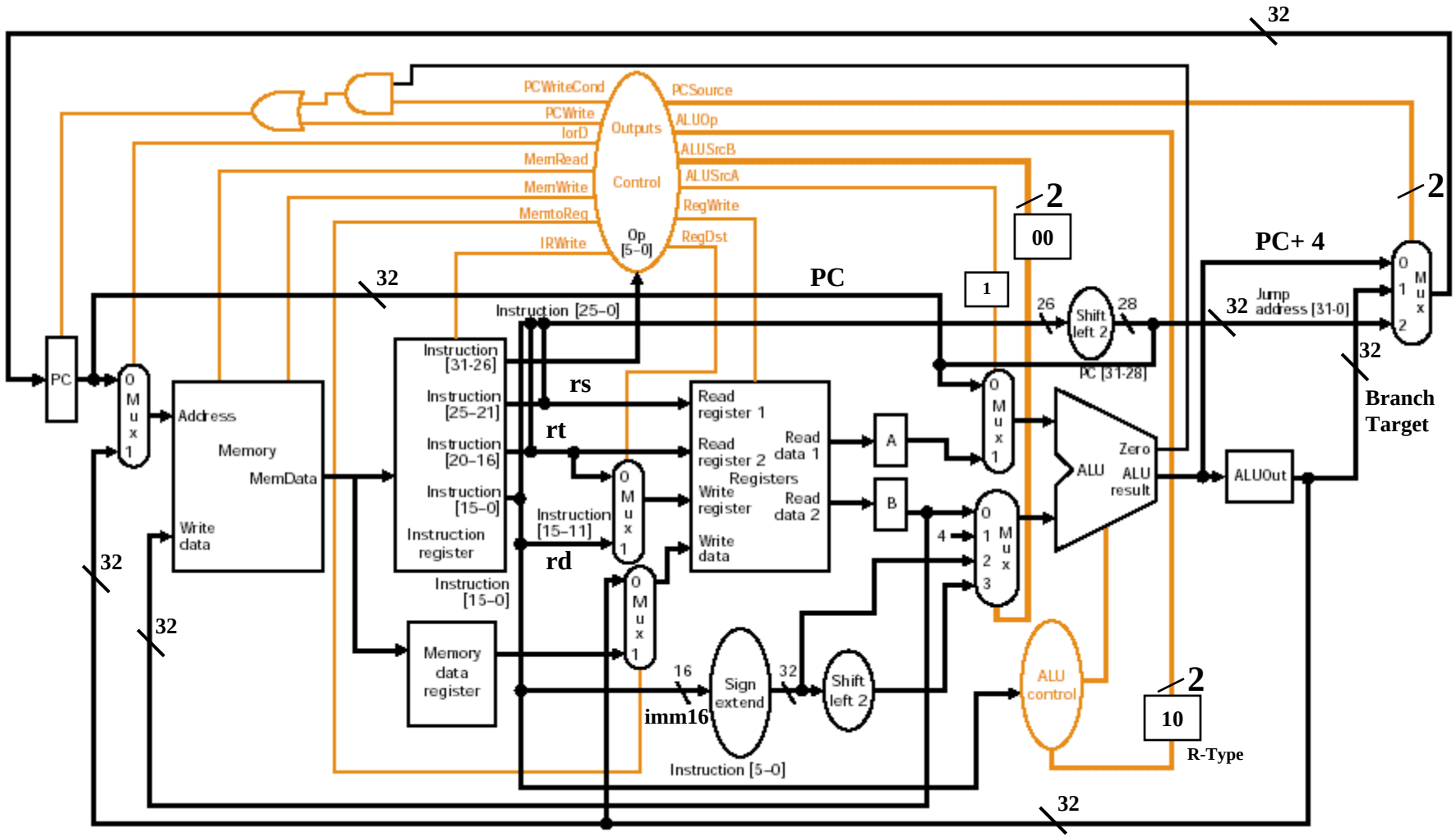
ALUout ← A funct B

R[rd] ← ALUout

(Figure 5.34 page 335)

R-Type Execution (EX) Cycle (State 6)

ALUout ← A funct B | **ALUSrcA = 1** | **ALUSrcB = 00** | **ALUOp = 10 (R-Type)**



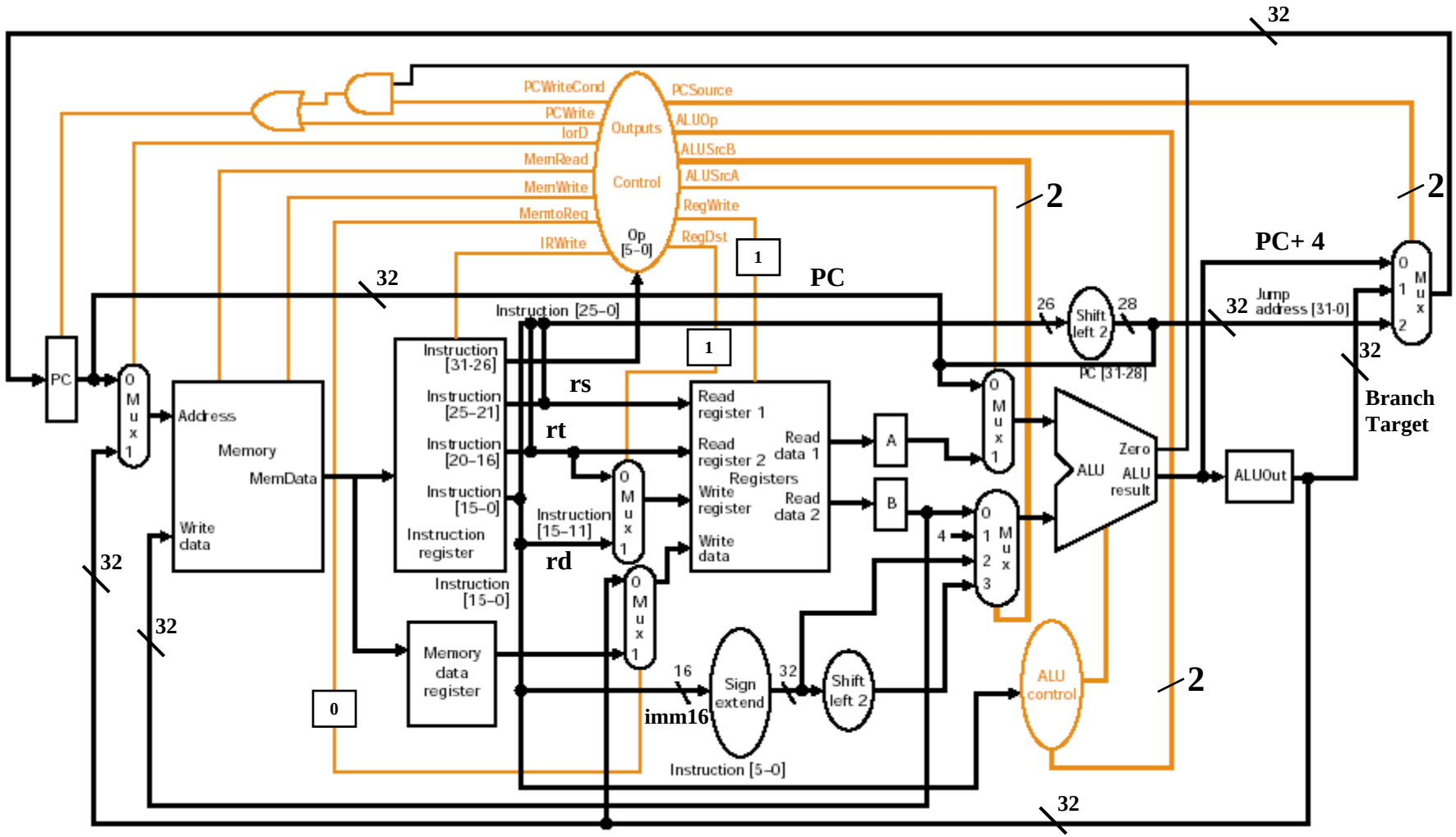
(ORI not supported, Jump supported)

(Figure 5.28 page 323)

R-Type Write Back (WB) Cycle (State 7)

R[rd] ← ALUout

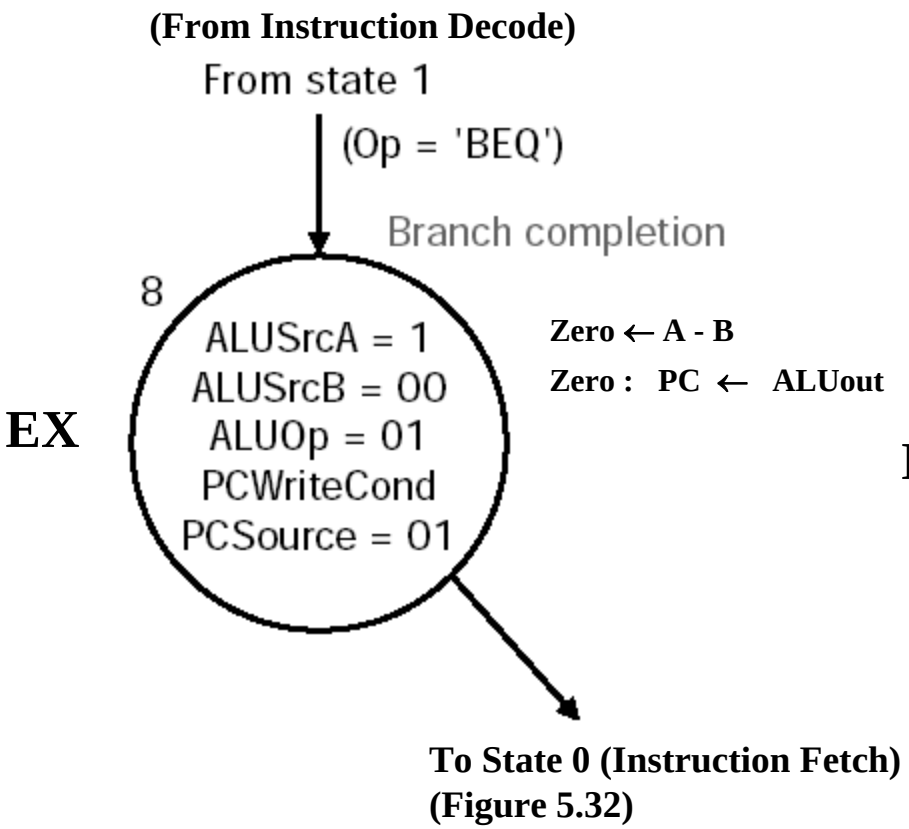
RegWrite = 1 MemtoReg = 0 RegDst = 1



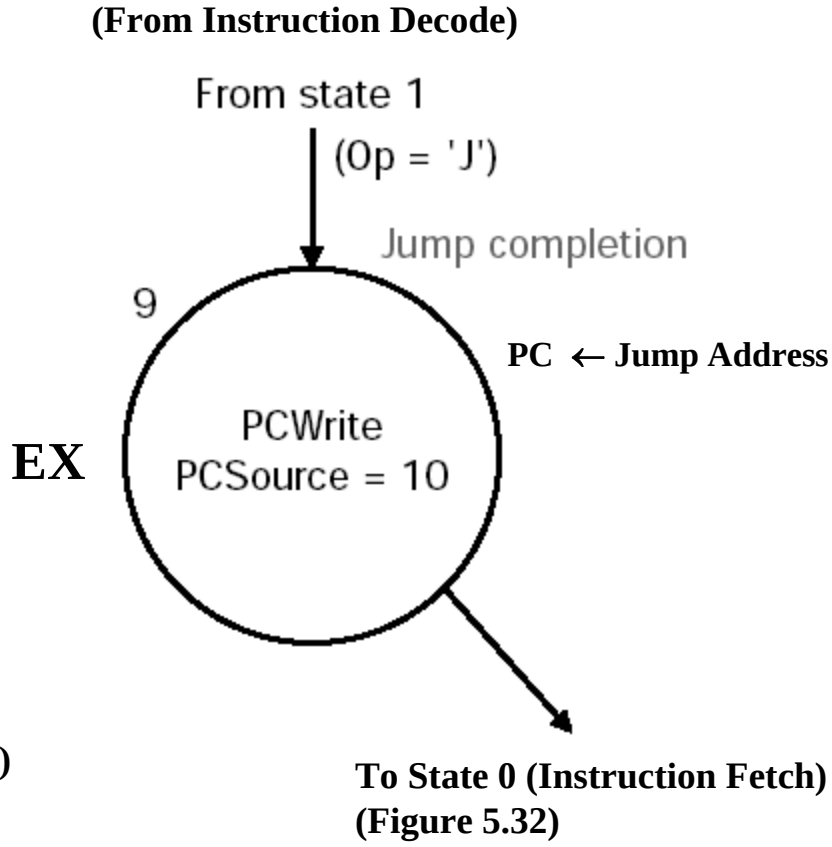
(ORI not supported, Jump supported)

(Figure 5.28 page 323)

Branch Instruction Single EX State



Jump Instruction Single EX State



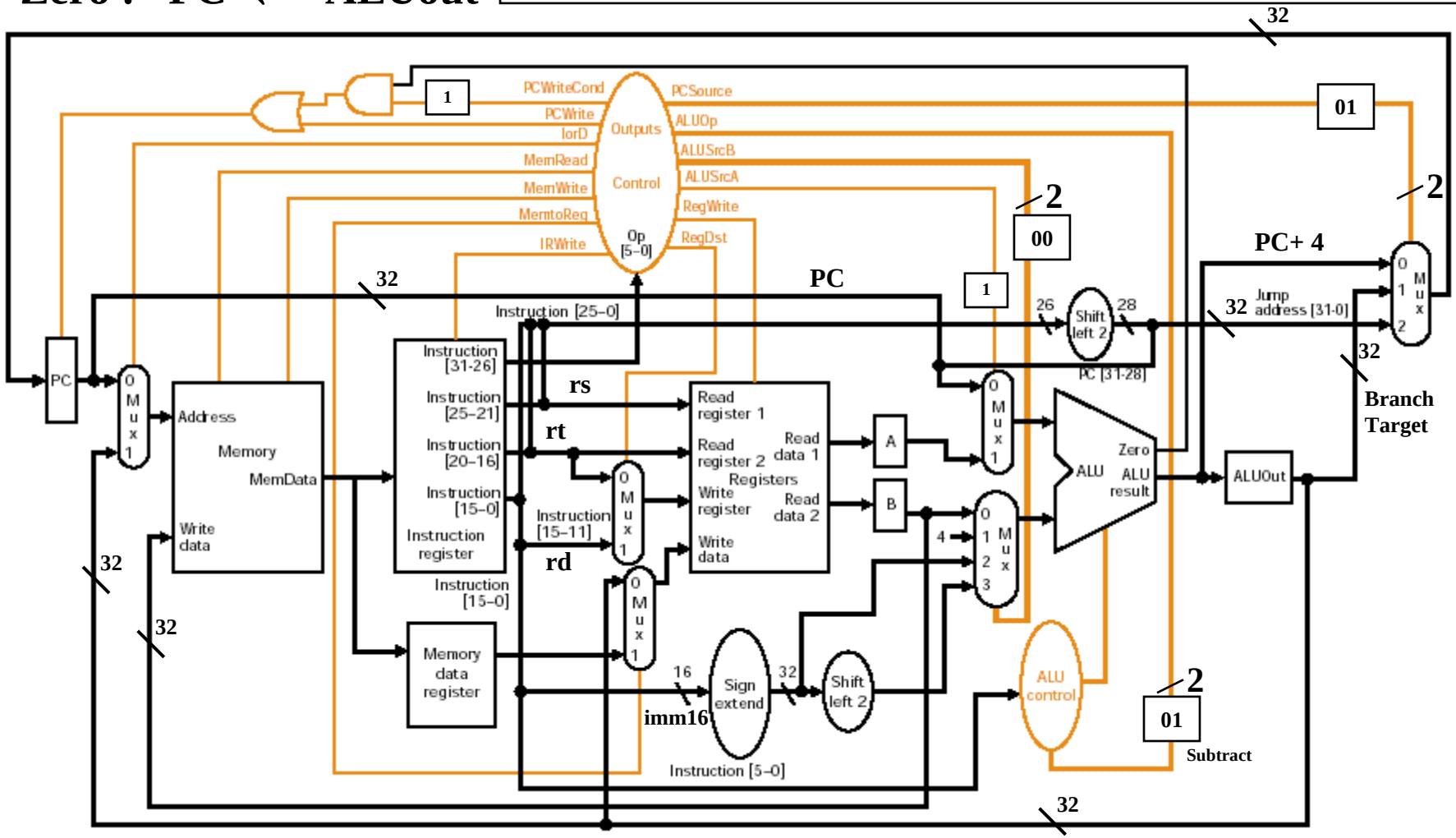
(Figures 5.35, 5.36 page 337)

Branch Execution (EX) Cycle (State 8)

Zero ← A - B

Zero : PC ← ALUout

ALUSrcA = 1	**ALUSrcB = 00**	**ALUOp = 01 (Subtract)**
PCWriteCond = 1	**PCSource = 01**	



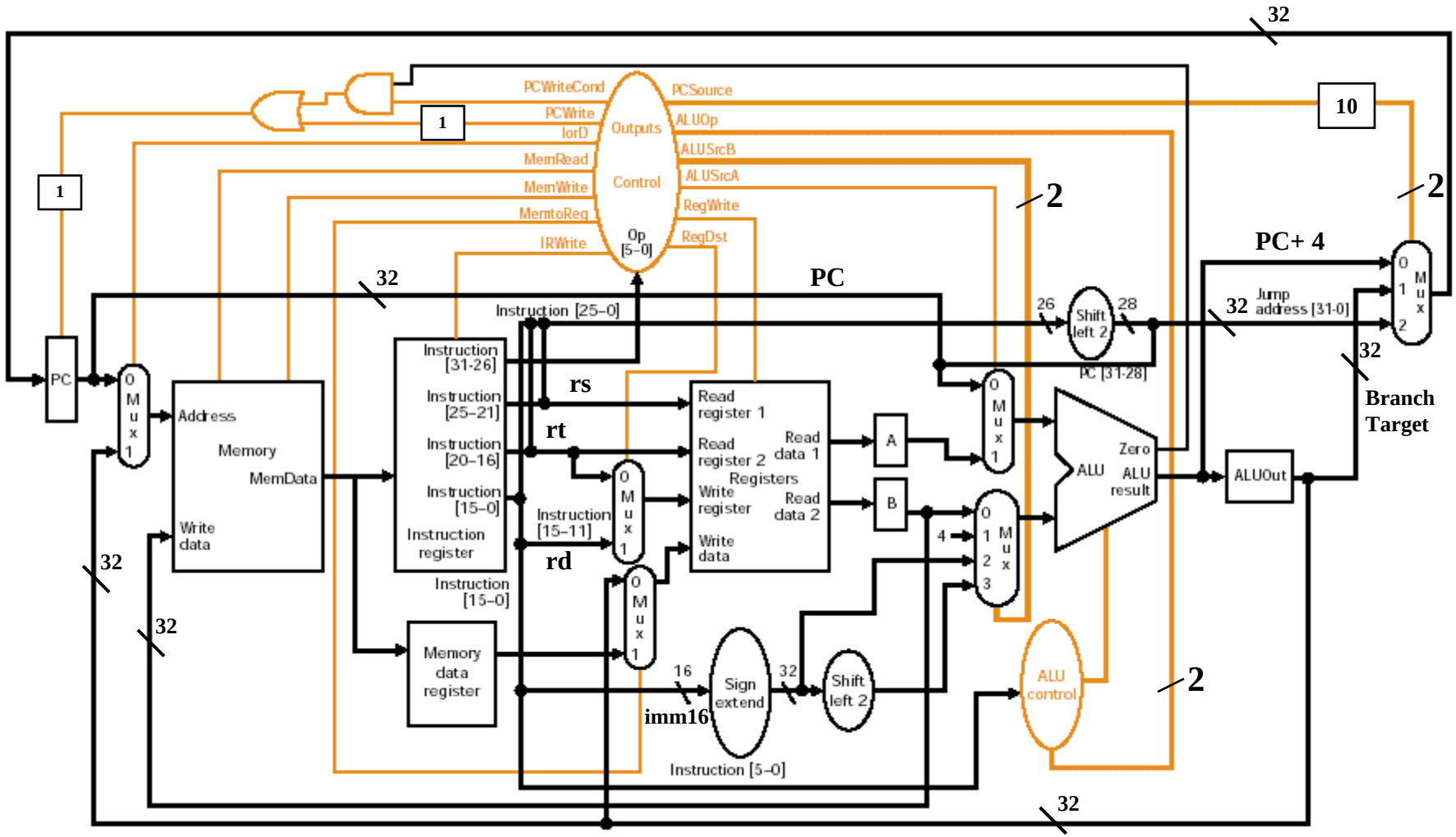
(ORI not supported, Jump supported)

(Figure 5.28 page 323)

Jump Execution (EX) Cycle (State 9)

PC ← Jump Address

PCWrite = 1 PCSource = 10



(ORI not supported, Jump supported)

EECC550 - Shaaban

(Figure 5.28 page 323)

MIPS Multi-cycle Datapath Performance Evaluation

$1 \leq CPI \leq 5$

- **What is the average CPI?**
  - **State diagram gives CPI for each instruction type.**
  - **Workload (program) 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).

Adding Support for swap to Multi Cycle Datapath

- **You are to add support for a new instruction, swap that exchanges the values of two registers to the MIPS multicycle datapath of Figure 5.28 on page 232**

  swap $rs, $rt

  i.e. **R[rt] ← R[rs]**
  R[rs] ← R[rt]

- **Swap used the R-Type format with:**

  the value of field rs = the value of field rd

- **Add any necessary datapaths and control signals to the multicycle datapath. Find a solution that minimizes the number of clock cycles required for the new instruction <u>without modifying the register file</u>. Justify the need for the modifications, if any.**

  i.e No additional register write ports

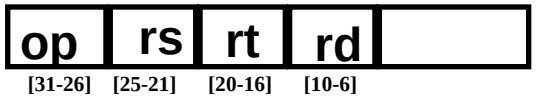
- **Show the necessary modifications to the multicycle control finite state machine of Figure 5.38 on page 339 when adding the swap instruction. For each new state added, provide the dependent RTN and active control signal values.**

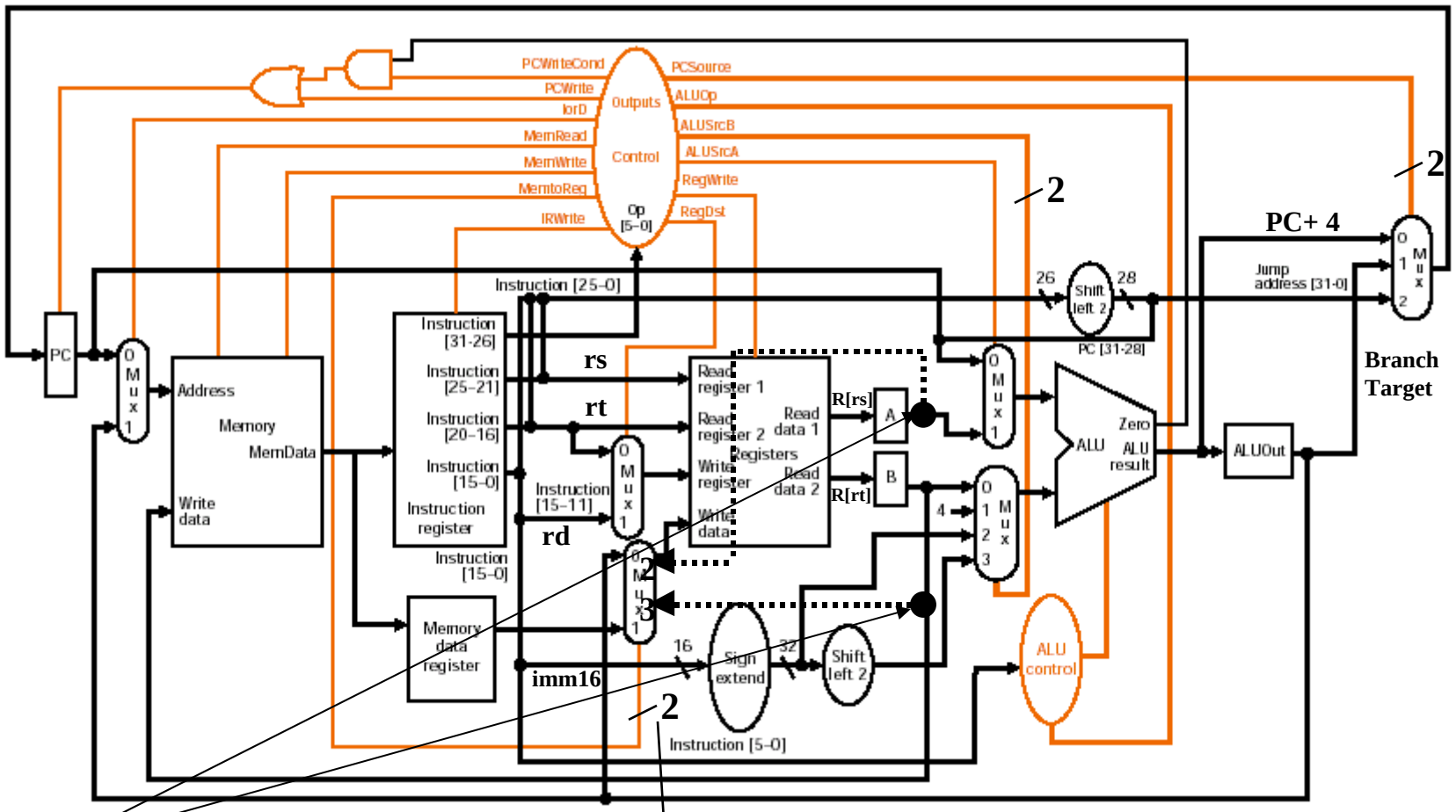
Adding swap Instruction Support to Multi Cycle Datapath

Swap $rs, $rt **R[rt] ← R[rs]**

R[rs] ← R[rt]

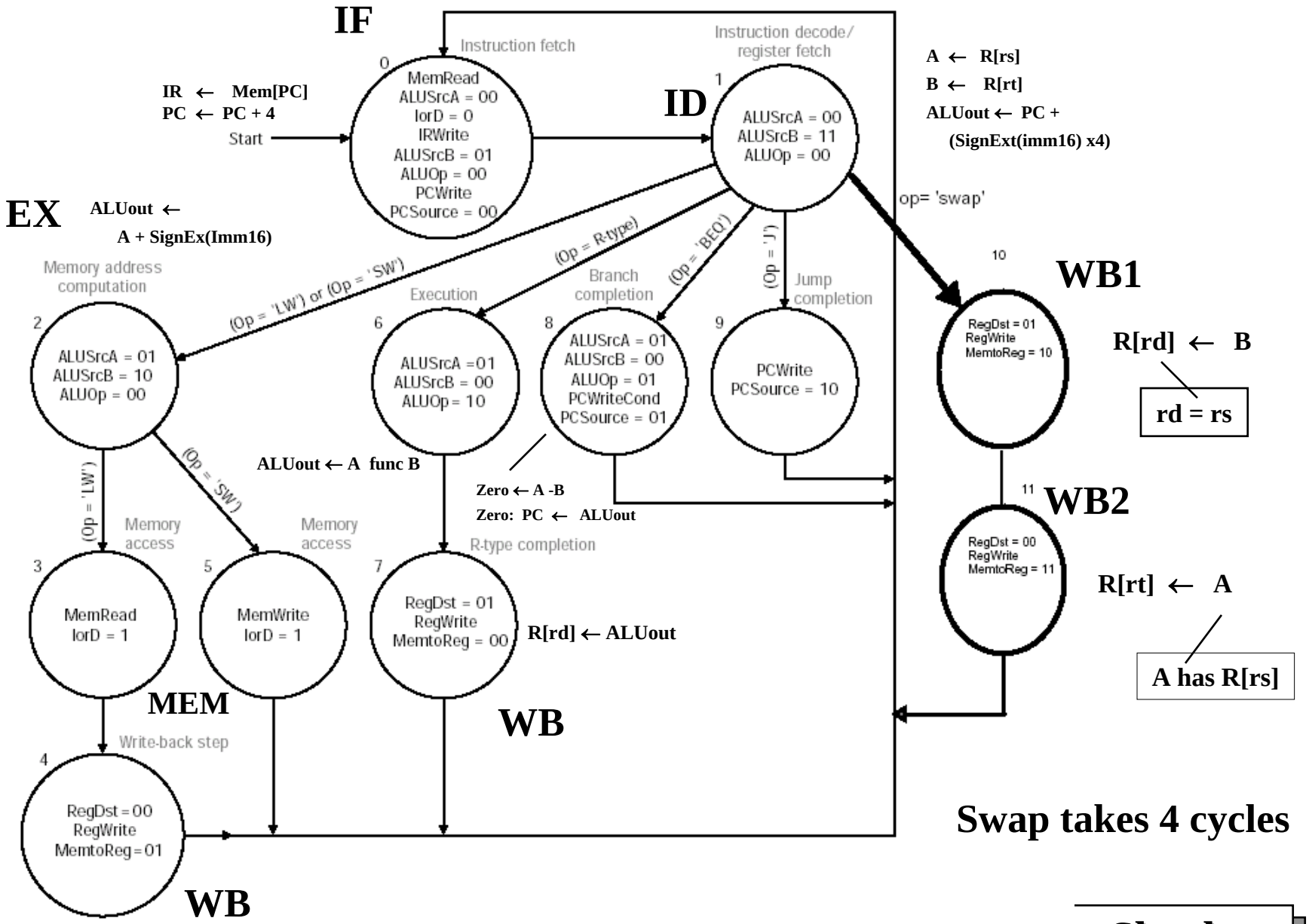
We assume here rs = rd in instruction encoding





The outputs of A and B should be connected to the multiplexor controlled by MemtoReg if one of the two fields (rs and rd) contains the name of one of the registers being swapped. The other register is specified by rt. The MemtoReg control signal becomes two bits.

Adding swap Instruction Support to Multi Cycle Datapath



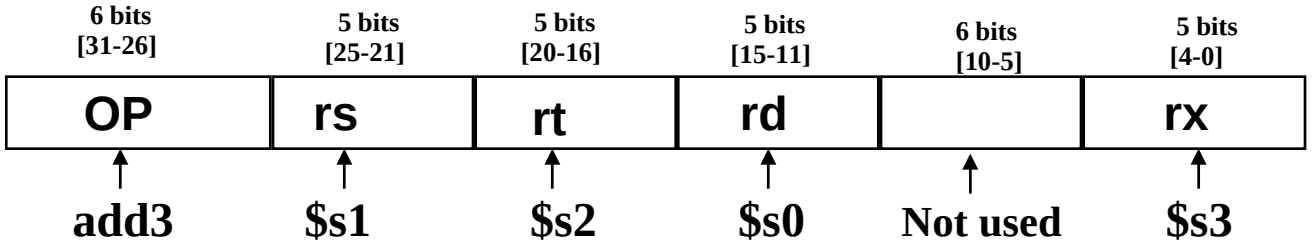
Adding Support for add3 to Multi Cycle Datapath

- **You are to add support for a new instruction, add3, that adds the values of three registers, to the MIPS multicycle datapath of Figure 5.28 on page 232**
  For example:

  add3 $s0,$s1, $s2, $s3

  Register $s0 gets the sum of $s1, $s2 and $s3.

  The instruction encoding uses a modified R-format, with an additional register specifier rx added replacing the five low bits of the "funct" field.



- **Add necessary datapath components, connections, and control signals to the multicycle datapath <u>without modifying the register bank or adding additional ALUs</u>. Find a solution that minimizes the number of clock cycles required for the new instruction. Justify the need for the modifications, if any.**
- **Show the necessary modifications to the multicycle control finite state machine of Figure 5.38 on page 339 when adding the add3 instruction. For each new state added, provide the dependent RTN and active control signal values.**

add3 instruction support to Multi Cycle Datapath

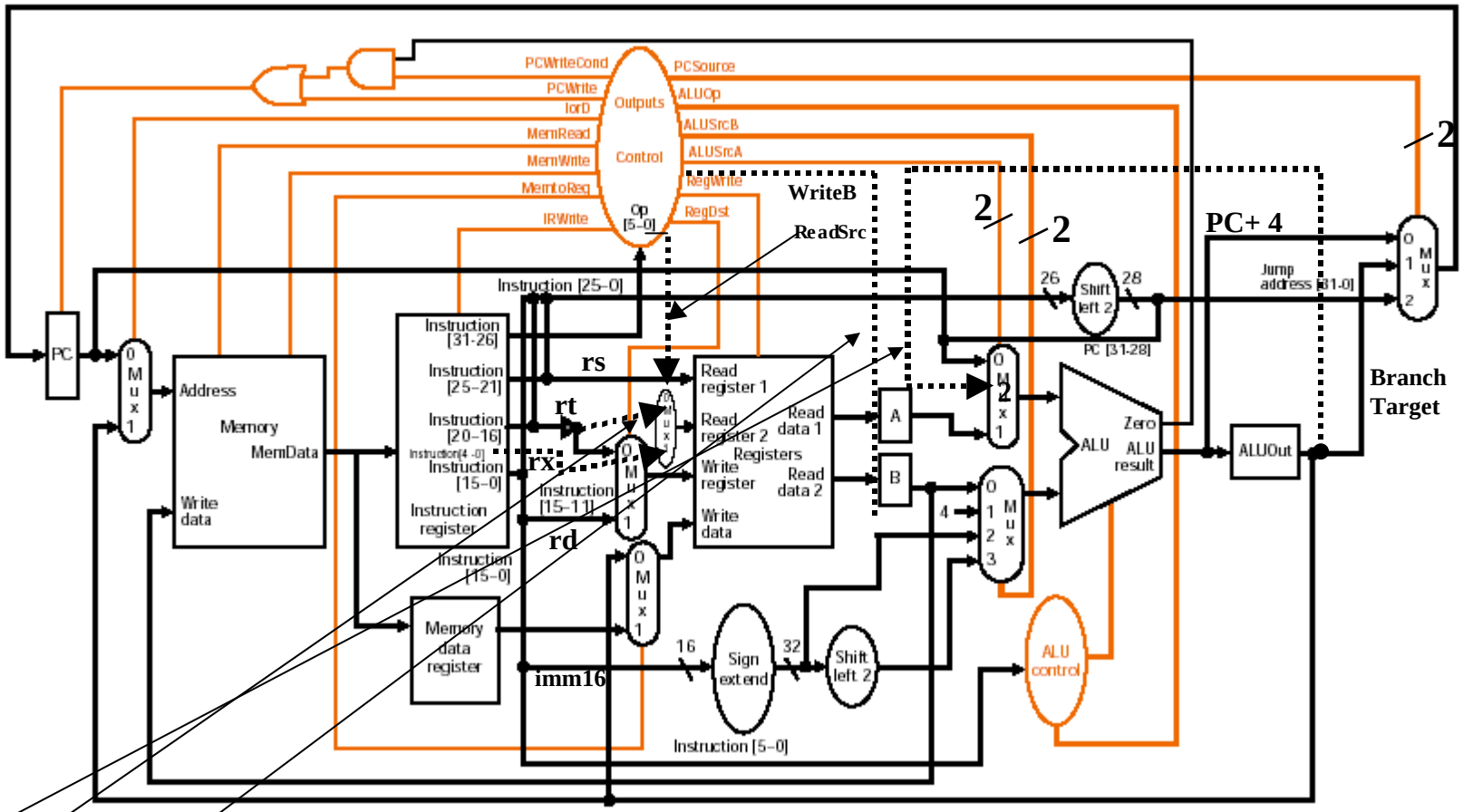
Add3 $rd, $rs, $rt, $rx

R[rd] ← R[rs] + R[rt] + R[rx]

rx is a new register specifier in field [0-4] of the instruction
No additional register read ports or ALUs allowed

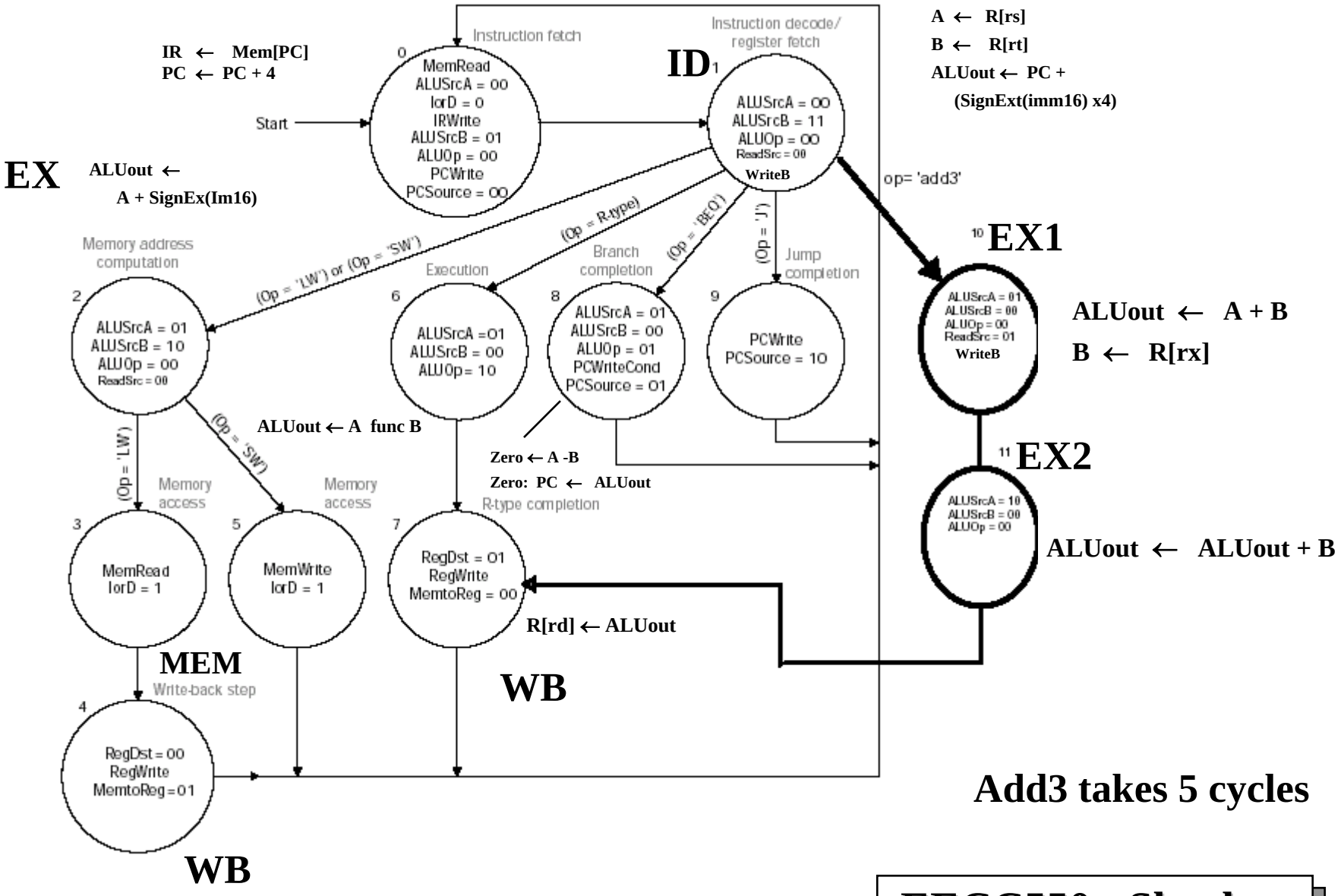
Modified R-Format

op	rs	rt	rd		rx
[31-26]	[25-21]	[20-16]	[10-6]		[4-0]



1. ALUout is added as an extra input to first ALU operand MUX to use the previous ALU result as an input for the second addition.
2. A multiplexor should be added to select between rt and the new field rx containing register number of the 3rd operand (bits 4-0 for the instruction) for input for Read Register 2.
This multiplexor will be controlled by a new one bit control signal called ReadSrc.
3. WriteB control line added to enable writing R[rx] to B

add3 instruction support to Multi Cycle Datapath



Add3 takes 5 cycles