Computer Technology

Topic 2

The Processor: A Simple Implementation Scheme

[*Computer Organization and Design, 4th Edition*, Patterson & Hennessy, © 2008, MK]

Overview

- Review
  - Memory elements: flip-flops, registers and memory 4.2
  - Processor components
  - Instruction cycle 4.1
- A simple implementation scheme: single-cycle processor
  - Building a datapath
  - Building a control unit
  - Clock cycle

Memory elements

OVERVIEW

Memory elements: flip-flops

- Circuit used to store 1-bit data
- Triggering:
  - How the output values change
  - SR, JK, D and T
- Synchronous- Asynchronous
  - Synchronous circuits with asynchronous inputs
- Triggering classes:
  - When does the input value change
  - Edge-triggered and level-triggered

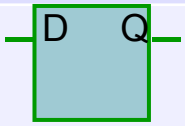
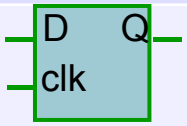
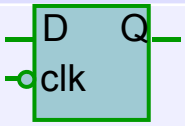
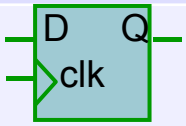
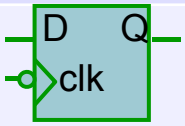
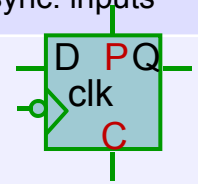
S	R	Q_{t+1}
0	0	Q_t
0	1	0
1	0	1
1	1	-

J	K	Q_{t+1}
0	0	Q_t
0	1	0
1	0	1
1	1	Q_t

D	Q_{t+1}
0	0
1	1

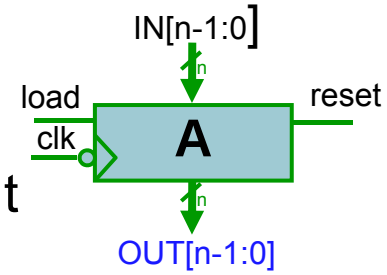
T	Q_{t+1}
0	Q_t
1	$\overline{Q}_t$

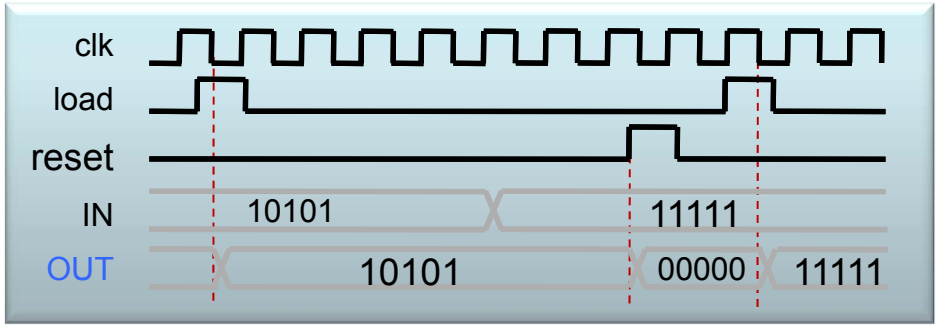
D flip-flops

Asynchronous	High-level triggered sync.	Low-level triggered sync.	rising-edge-triggered sync.	falling-edge-triggered sync.	… with other async. inputs
					

Memory elements: registers

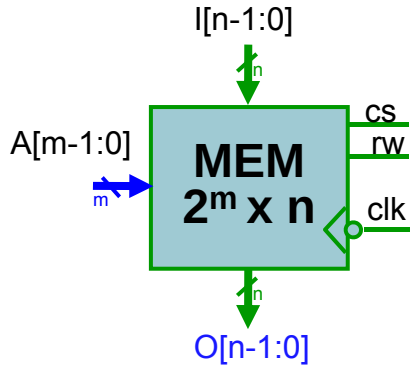
- Circuit used to store n-bit data
- Parallel in/out register (example)
  - The stored value is read at the output OUT[n-1:0]
  - The value to be loaded is at the input IN[n-1:0]
  - The value IN is put into the register when the input load is asserted and the falling edge occurs at clk
  - When the input reset is asserted, the register value is cleared (0 is stored)

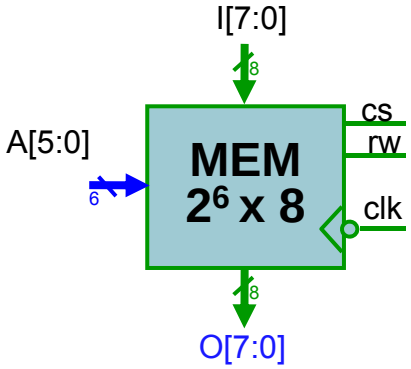


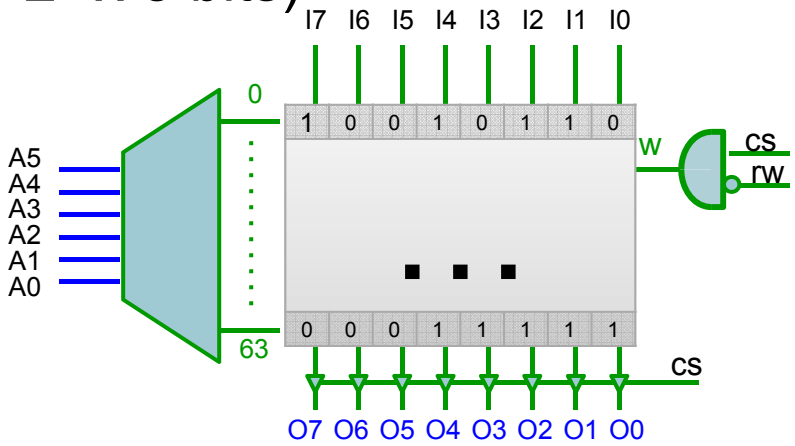


Memory elements: memory

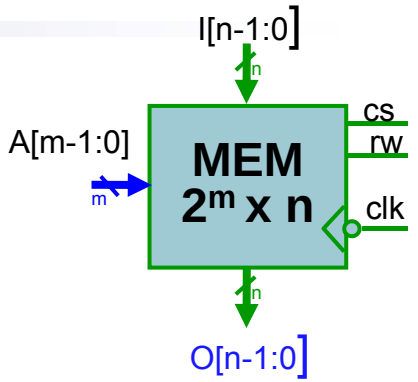
- Circuit to store M (depth) words of n-bit (width) data ($M = 2^m$)
- Example:
  - cs: chip select signal
  - rw: read (1) and write (0) signal
  - clk: clock signal (to synchronize writing)
  - IN[n-1:0]: input data
  - A[m-1:0]: input addresses
  - O[n-1:0]: output data
- 64 Bytes memory (2^6 x 1 Byte = 2^6 x 8 bits)







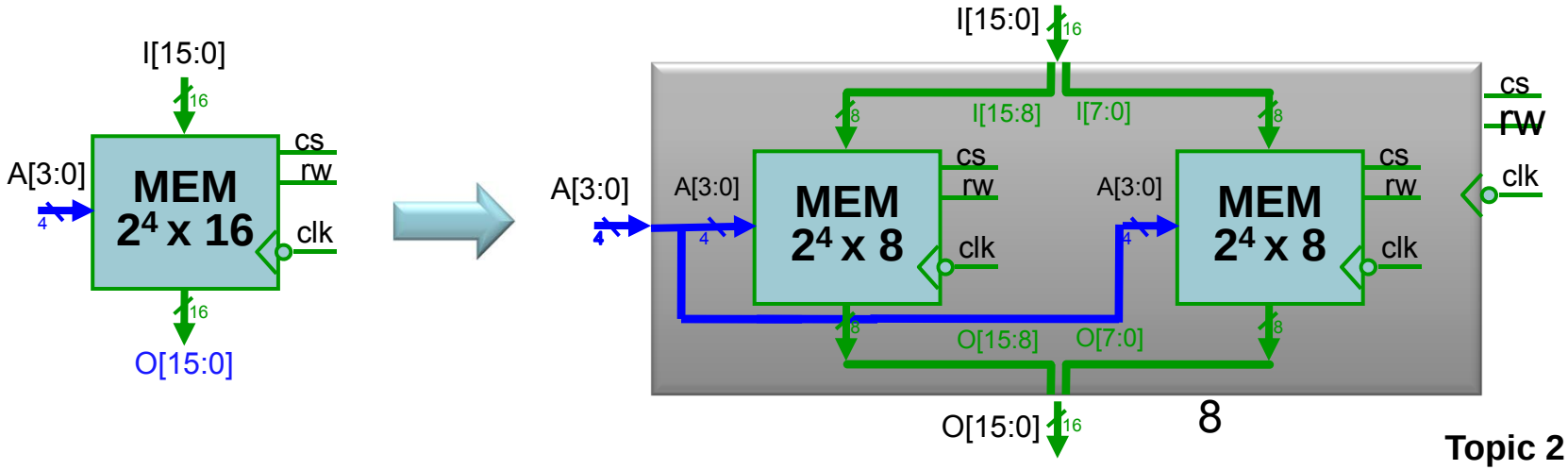
Memory elements: memory



- Reading:
  - A[m-1:0] is an address A to be read (in binary)
  - Assert rw and cs to 1
  - After some delay, in OUT[n-1:0] the stored data appears in address A
- Writing:
  - A[m-1:0] is an address A to write into (in binary)
  - Data T to be written (in binary) is placed into IN[m-1:0]
  - Assert rw to 0 and cs to 1
  - After the falling clk edge, data T is stored in memory address A

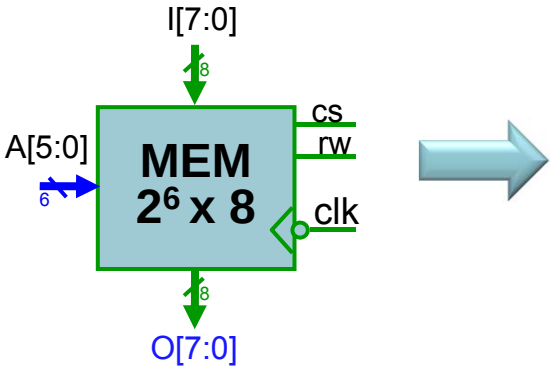
Memory elements: memory

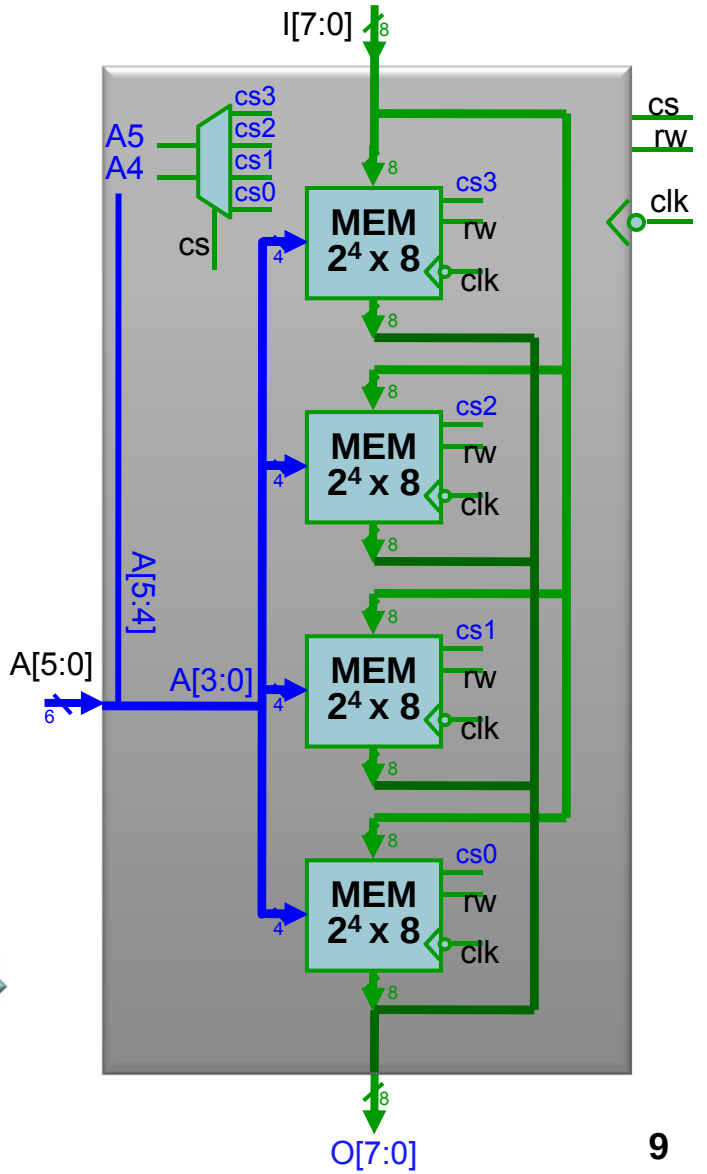
- How to obtain memory expansion in width:
  - Expanding the width, word size ⟺
    - Design a 2^mxq memory chip using 2^mxn memory chips (blocks)
    - Nr blocks for each entrance: lowest natural number p, with $p \geq q/n$
    - To store q bits, split them into p sets of n bits and distribute them over the p blocks of n
  - Example: design a 2^4x16 bits memory using 2^4x8 memory blocks



Memory elements: memory

- Memory expansion in depth:
  - Memory capacity (num. of words)
    - Design a larger 2^qxn memory with smaller 2^mxn memory blocks
    - Take $p = 2^{q-m}$ memories of 2^mxn
    - Distribute the 2^q locations over p blocks of 2^m locations and store each block in a 2^mxn memory
  - Example: design a 2^6x8 bits memory using 2^4x8 memory blocks

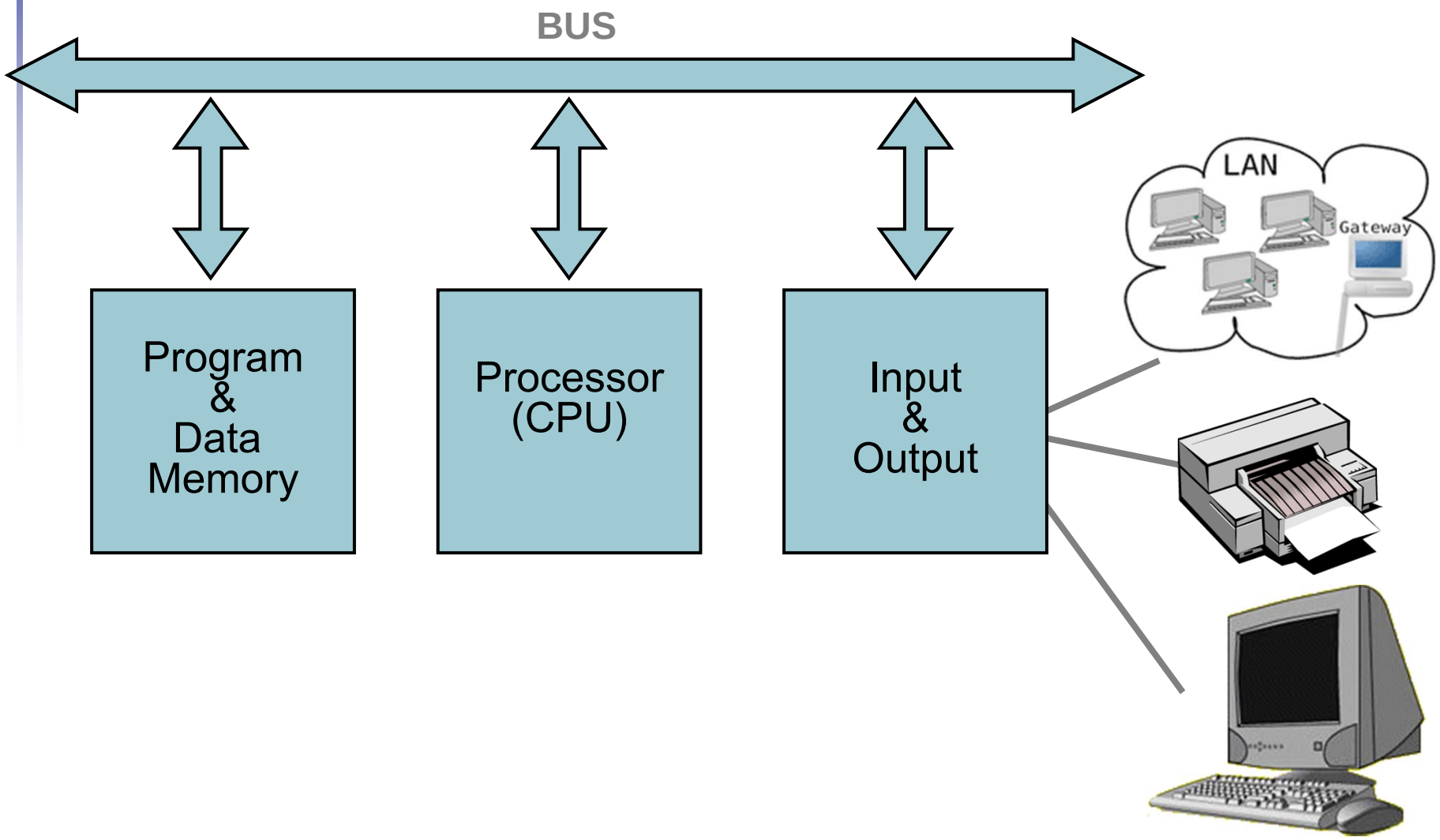




Processor components

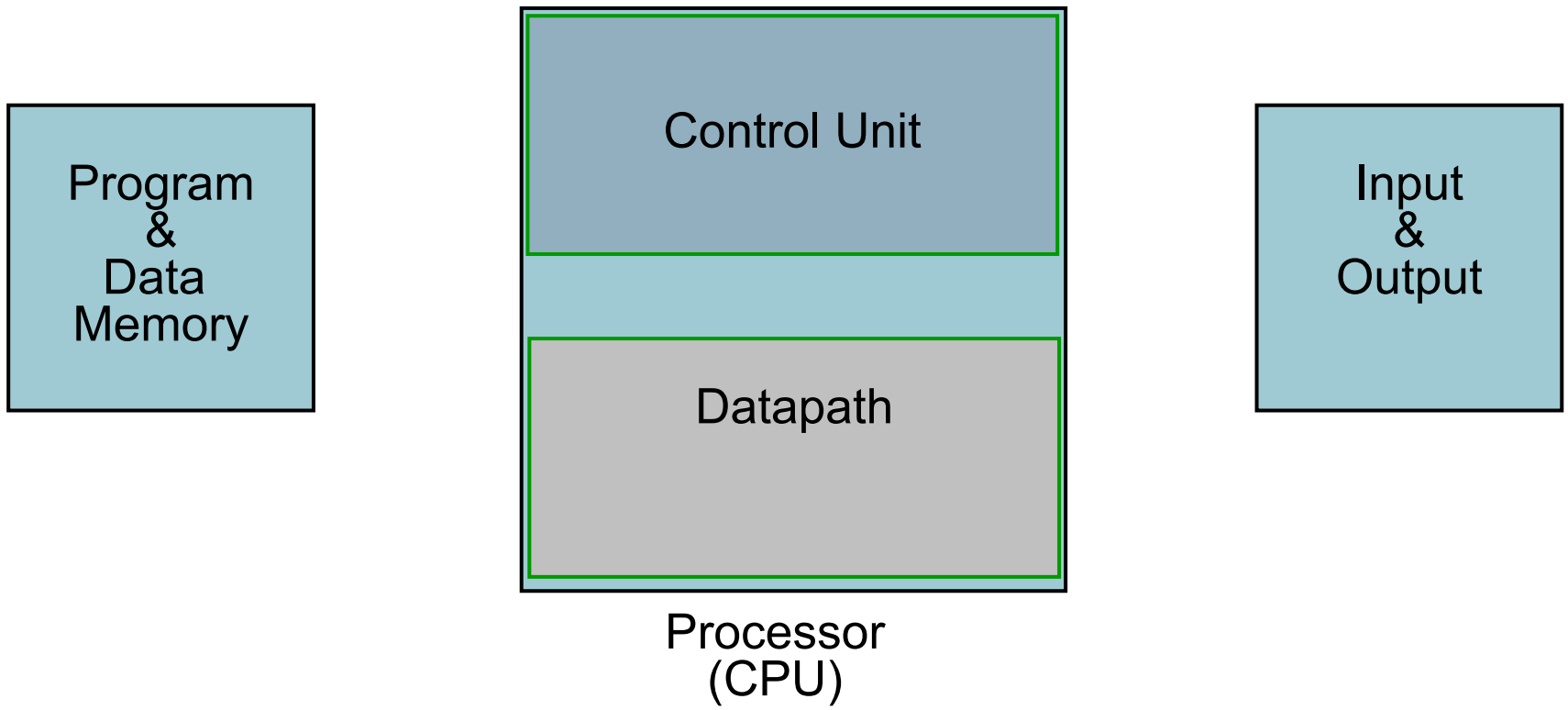
OVERVIEW

Components of a computer



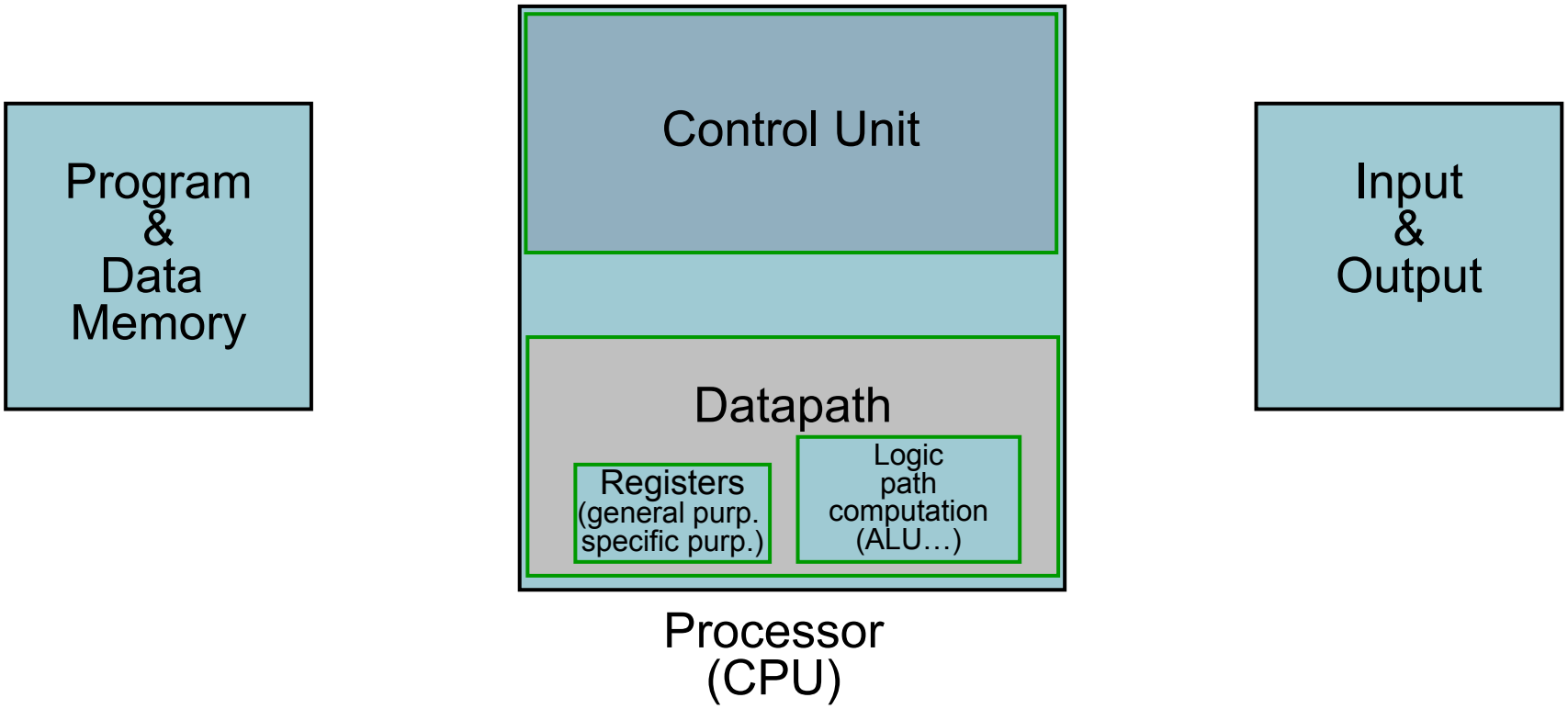
Components of a computer

- Focus on the CPU



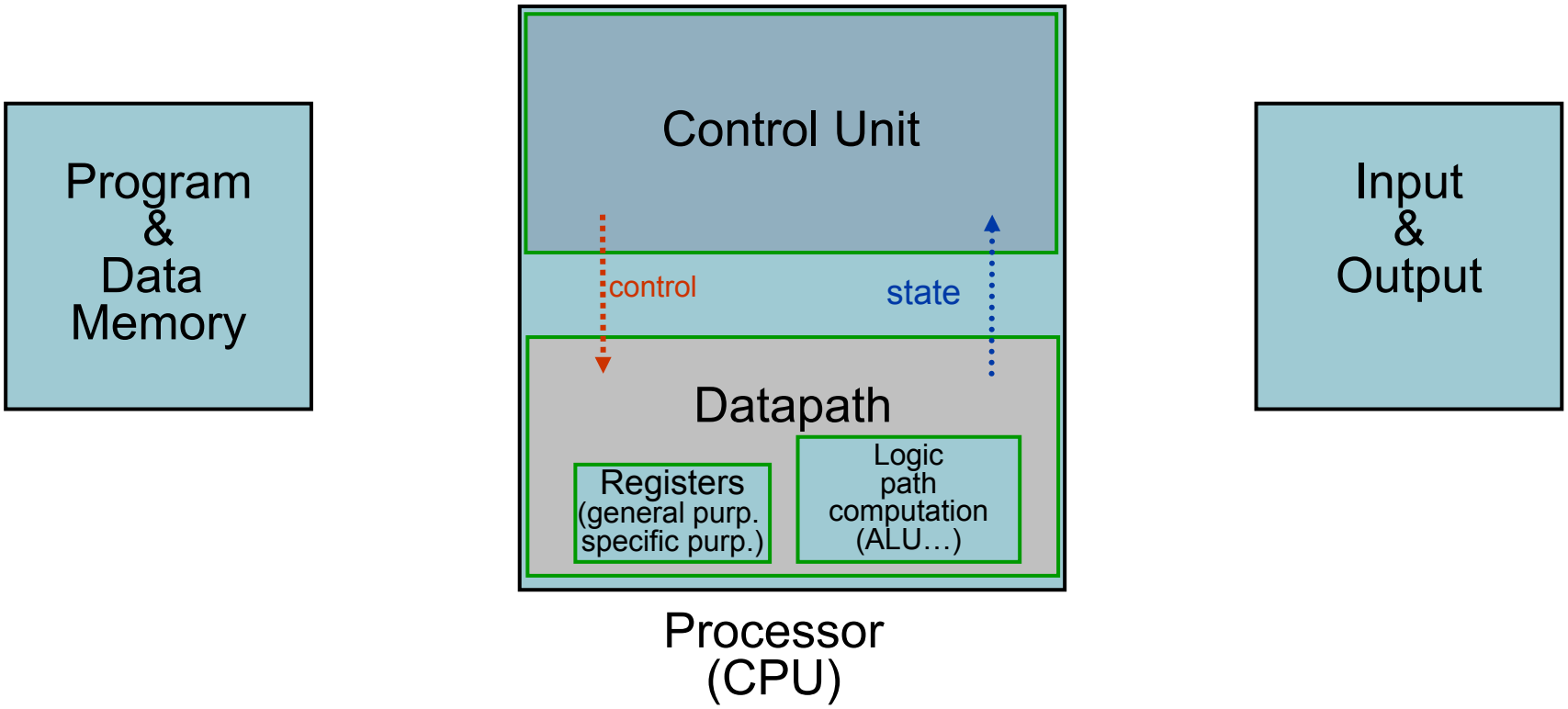
Components of a computer

- Focus on the CPU



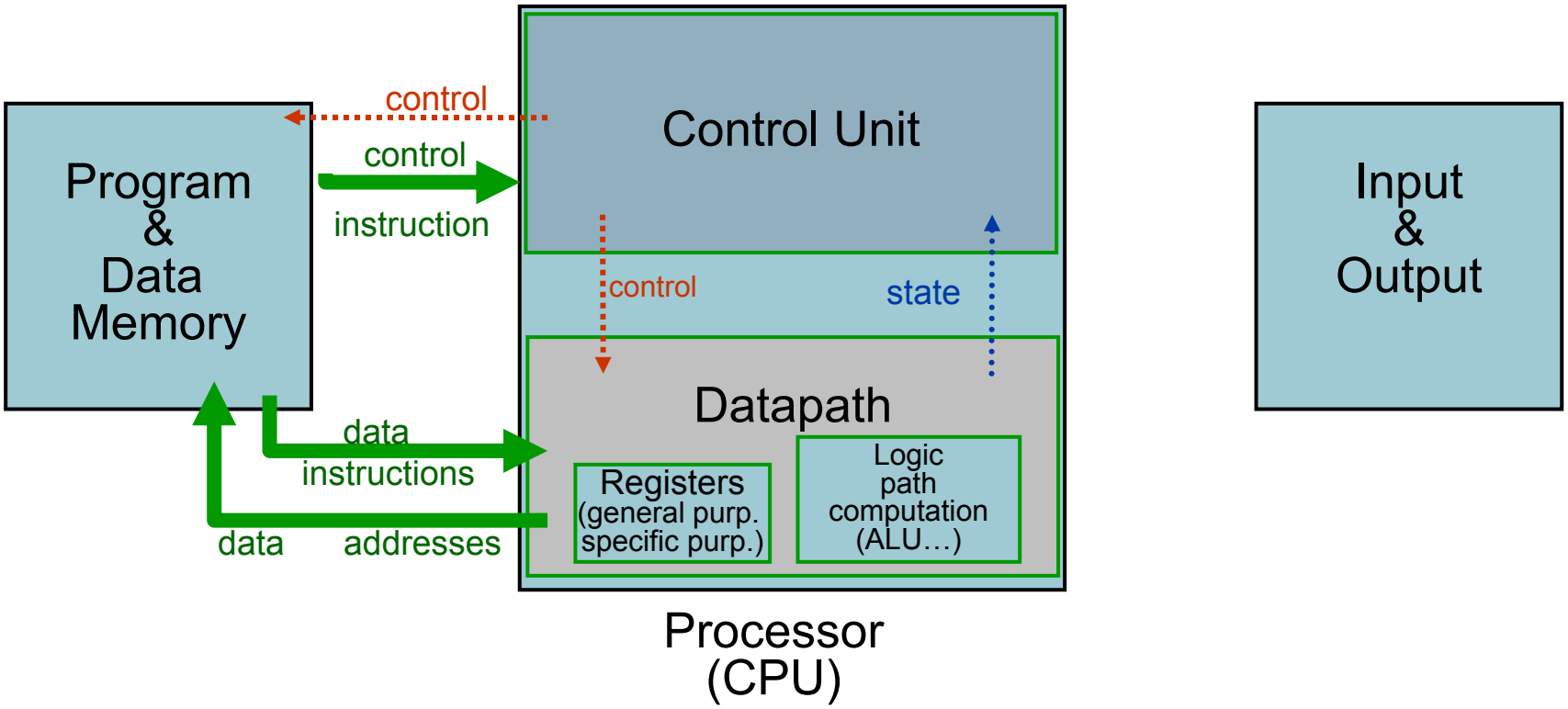
Components of a computer

- Focus on the CPU



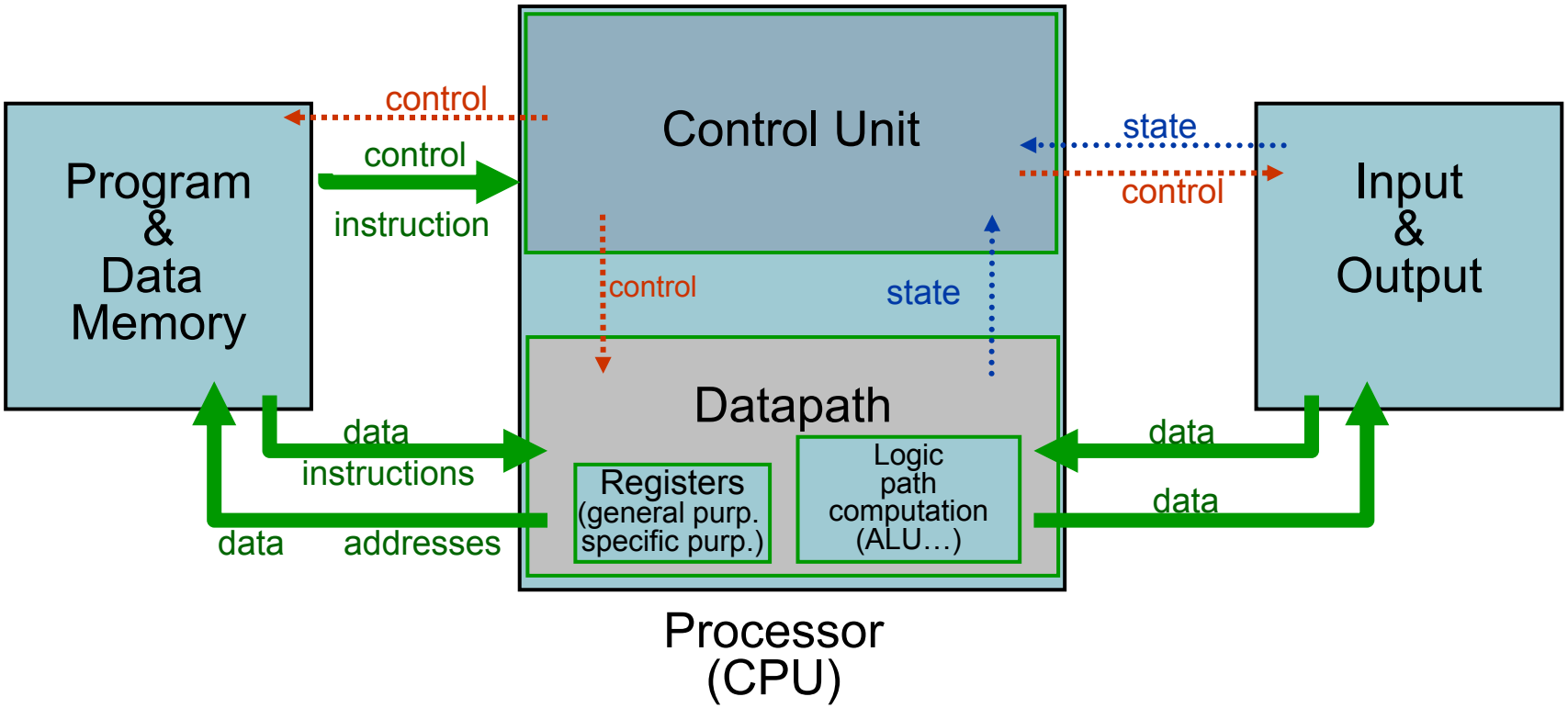
Components of a computer

- Focus on the CPU



Components of a computer

- Focus on the CPU



Instruction cycle (see also Section 4.1 of HP)

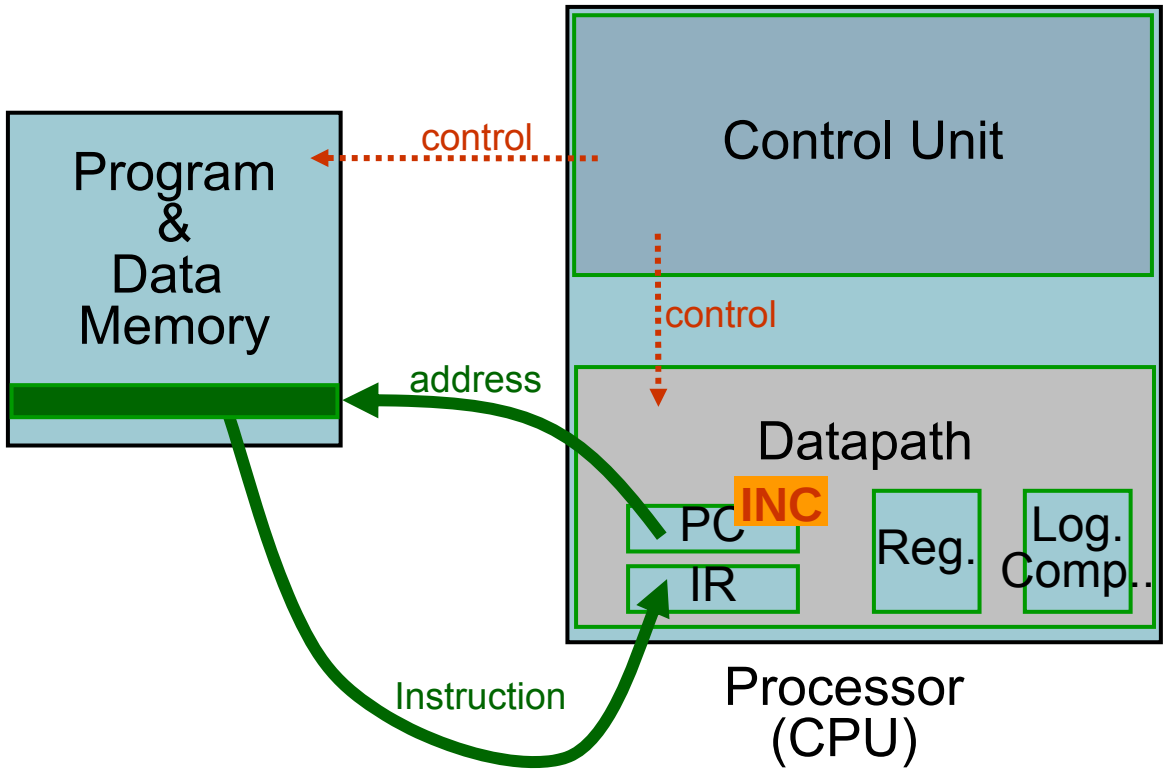
OVERVIEW

Instruction cycle

- Instruction execution steps
- Instruction cycle (general):
  - Fetching: the instruction is read from memory using the address in the Program Counter (PC)
  - Decoding: the type of instruction and the operands are determined
  - Execution: the operation specified by the instruction is executed
  - Do something with the result; write the result into register or memory or decide to jump

Instruction cycle

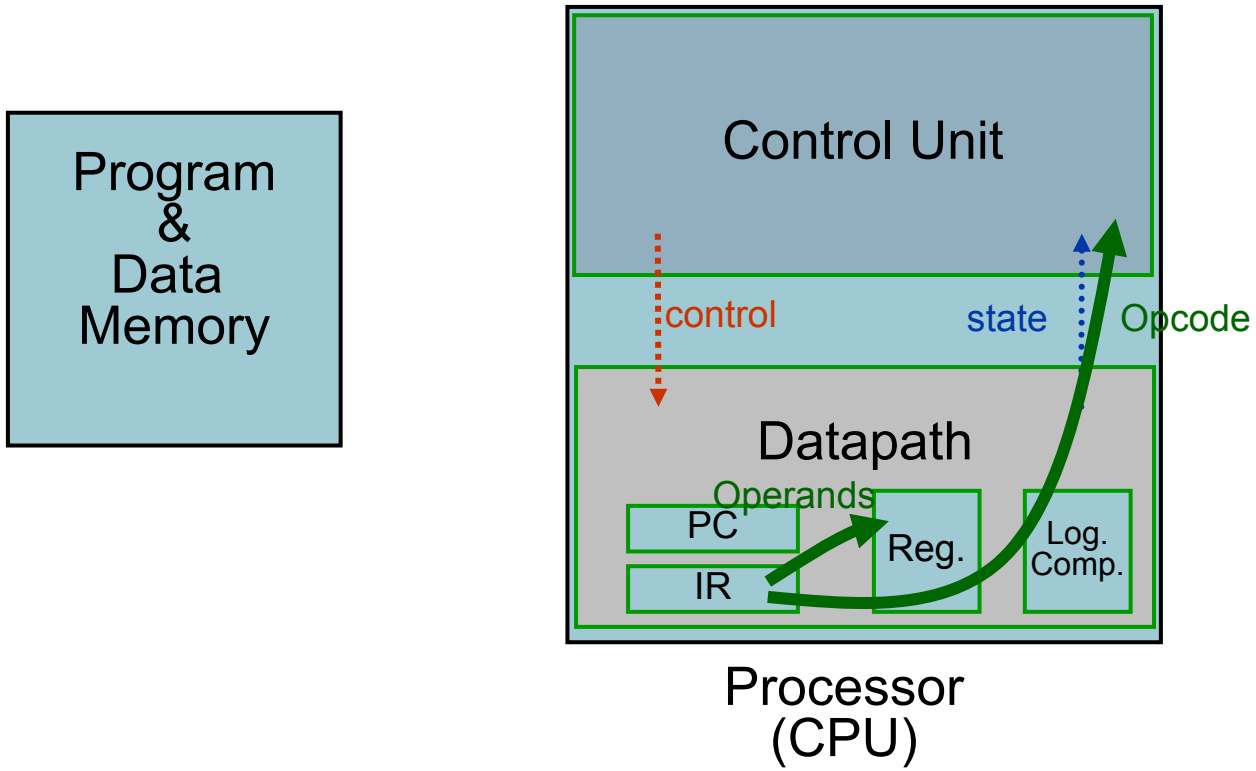
- Fetching



What is INC ?

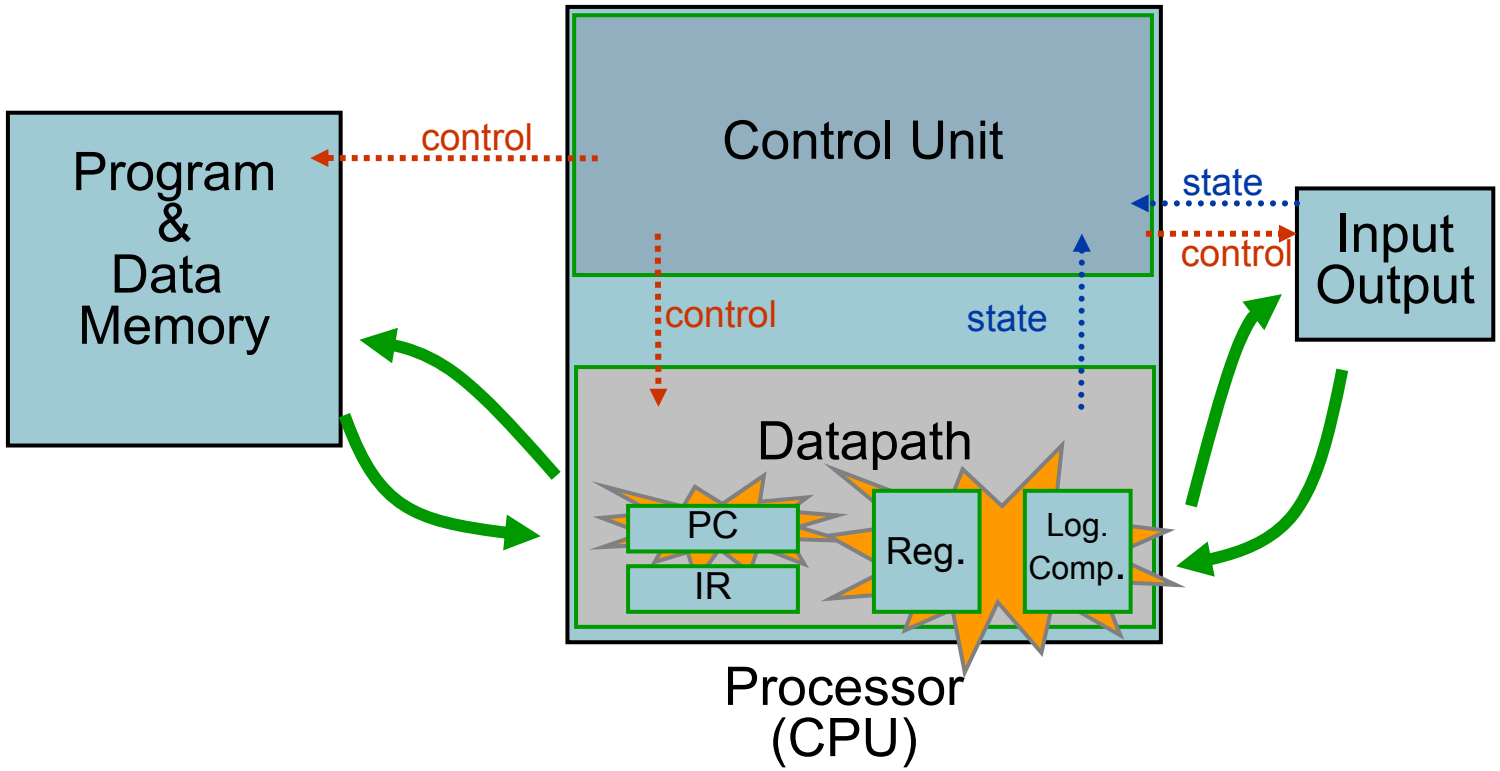
Instruction cycle

- Decoding



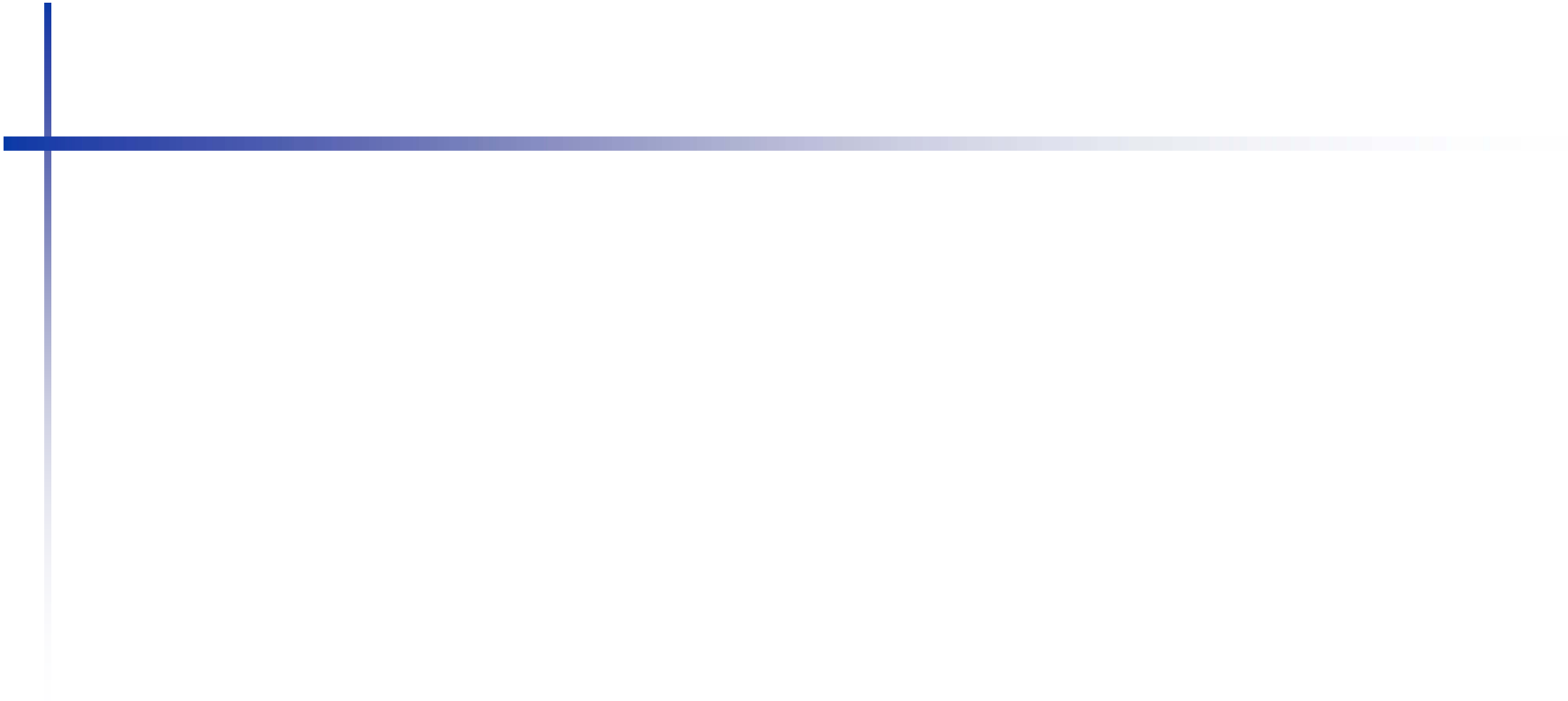
Instruction cycle

- Execution



Overview

- Review
  - Memory elements: flip-flops, registers and memory
  - Processor components
  - Instruction cycle
- A simple implementation scheme: single-cycle processor
  - Building a datapath 4.3
  - Building a control unit
  - Clock cycle



Building a Datapath, Section 4.3 HP

SINGLE-CYCLE PROCESSOR IMPLEMENTATION

Introduction

- MIPS (RISC) Design Principles
  - Simplicity favors regularity
    - fixed size instructions
    - small number of instruction formats
    - opcode always the first 6 bits
  - Smaller is faster
    - limited instruction set
    - limited number of registers in register file
    - limited number of addressing modes
  - Make the common case fast
    - arithmetic operands from the register file (load-store machine)
    - allow instructions to contain immediate operands
  - Good design requires good compromises
    - three instruction formats

The Processor: Datapath & Control

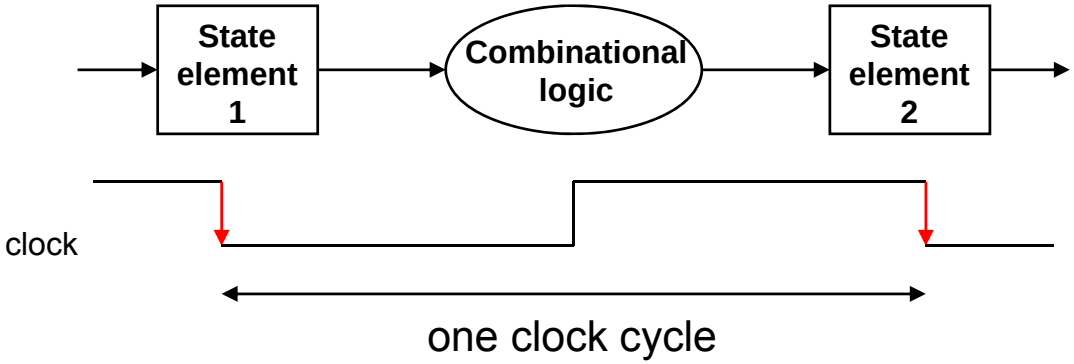
- MIPS implementation is simplified
  - All instructions are executed in one clock cycle
- Simple subset, shows most aspects
  - memory-reference instructions: **`lw, sw`**
  - arithmetic-logical instructions: **`add, sub, and, or, slt`**
  - control flow instructions: **`beq, j`**

Single-cycle implementation

- Features:
  - Register file of 32 registers of 32 bits
  - Register Program Counter (PC)
  - 2 memory blocks of 2^{32}x 8 bits (instruction memory and data memory)
  - ALU performs 8 operations
- All state elements are written on every clock cycle (falling edge CLK)

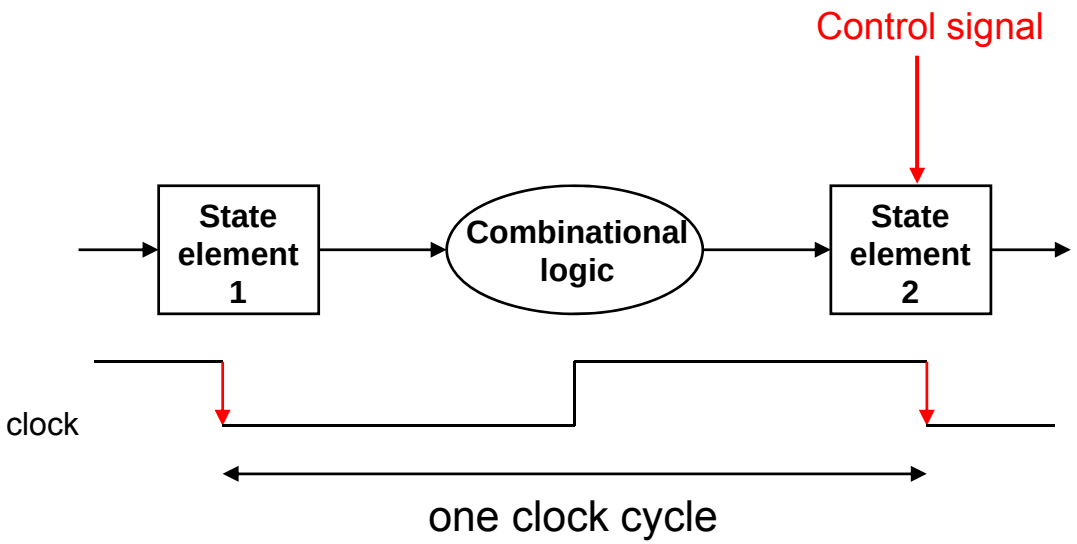
Aside: Clocking Methodologies

- The clocking methodology defines when data in a state element is valid and stable relative to the clock
  - State elements - a memory element such as a register
  - Edge-triggered – all state changes occur on a clock edge
- Typical execution
  - read contents of state elements -> send values through combinational logic -> write results to one or more state elements



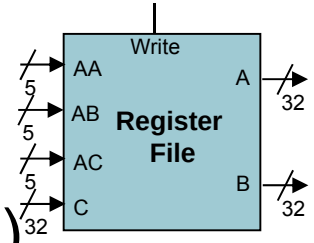
Aside: Clocking Methodologies

- Assumes state elements are written on every clock cycle; if not, need explicit write control signal
  - write occurs only when both the write control is asserted and the clock edge occurs

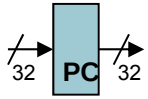


Single-cycle implementation

- Register file
  - 32 registers of 32 bits
  - Two read ports (A, B) and one write port (C)
    - Two registers can be read and one written at the same time
  - Interface:
    - AA: address port of register to be read through A
    - AB: address port of register to be read through B
    - AC: address port of register to be written through C



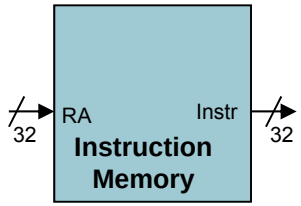
- PC
  - 32-bit synchronous parallel in/out register

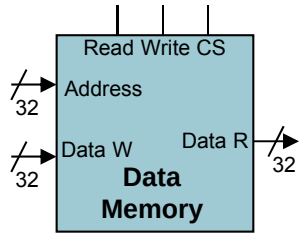


Single-cycle implementation

- Memory blocks:
  - 2 memory blocks of 2^{32}x 8 bits
  - Instruction memory:
    - RA: read address port of data to be read
    - Instr: instruction read port
    - Only read
  - Data memory:
    - Address: address port of data to be read and written
    - Data W: write data
    - Data R: data to be read from memory
    - Read: read; Write: write; CS: chip select

control of reading/writing and where

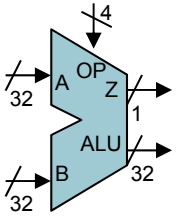




Single-cycle implementation

- ALU:
  - Arithmetic-logic unit with two 32-bit inputs
  - Flag Z (zero): output state
  - ALU operations:

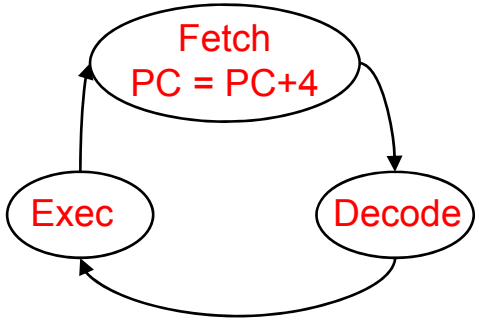
OP	Mnemo.	Operation
0000	AND	ALU = A AND B
0001	OR	ALU = A OR B
0010	ADD	ALU = A+B
0110	SUB	ALU = A-B
0111	SLT	IF (A < B) ALU = 1 else ALU = 0
1100	NOR	ALU = ~(A|B)



Why is OP 4 bits and not 6?
What does it actually mean?

Instruction cycle

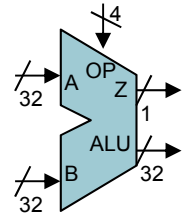
- Generic implementation (instruction cycle):
  - use the address in the program counter (PC) to fetch the instruction from memory and update PC
  - decode the instruction and read registers
  - execute the instruction



- All instructions (except `j`) use the ALU after reading the registers

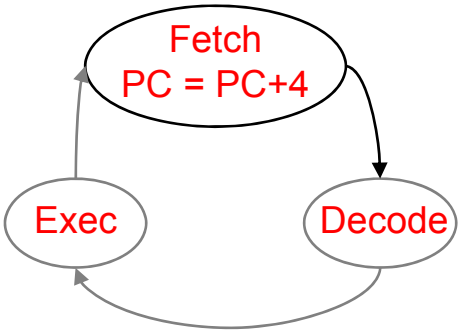
Instruction cycle

- Depending on instruction class (R,I,J-format):
  - Use ALU to calculate
    - Arithmetic result
    - Memory address for load/store
    - Branch target address
  - Access data memory for load/store
  - PC ← target address or PC + 4



Fetching instruction

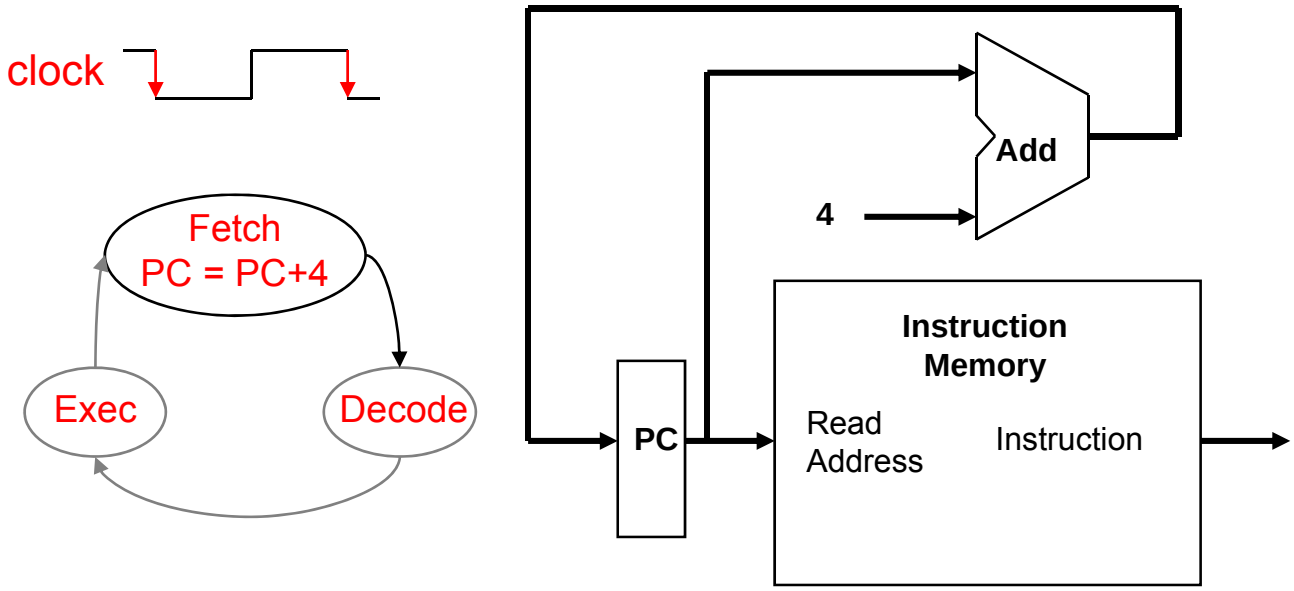
- Fetching instruction involves
  - reading the instruction from the Instruction Memory
  - updating the PC value to be the address of the next (sequential) instruction



Hardware?

Fetching instruction

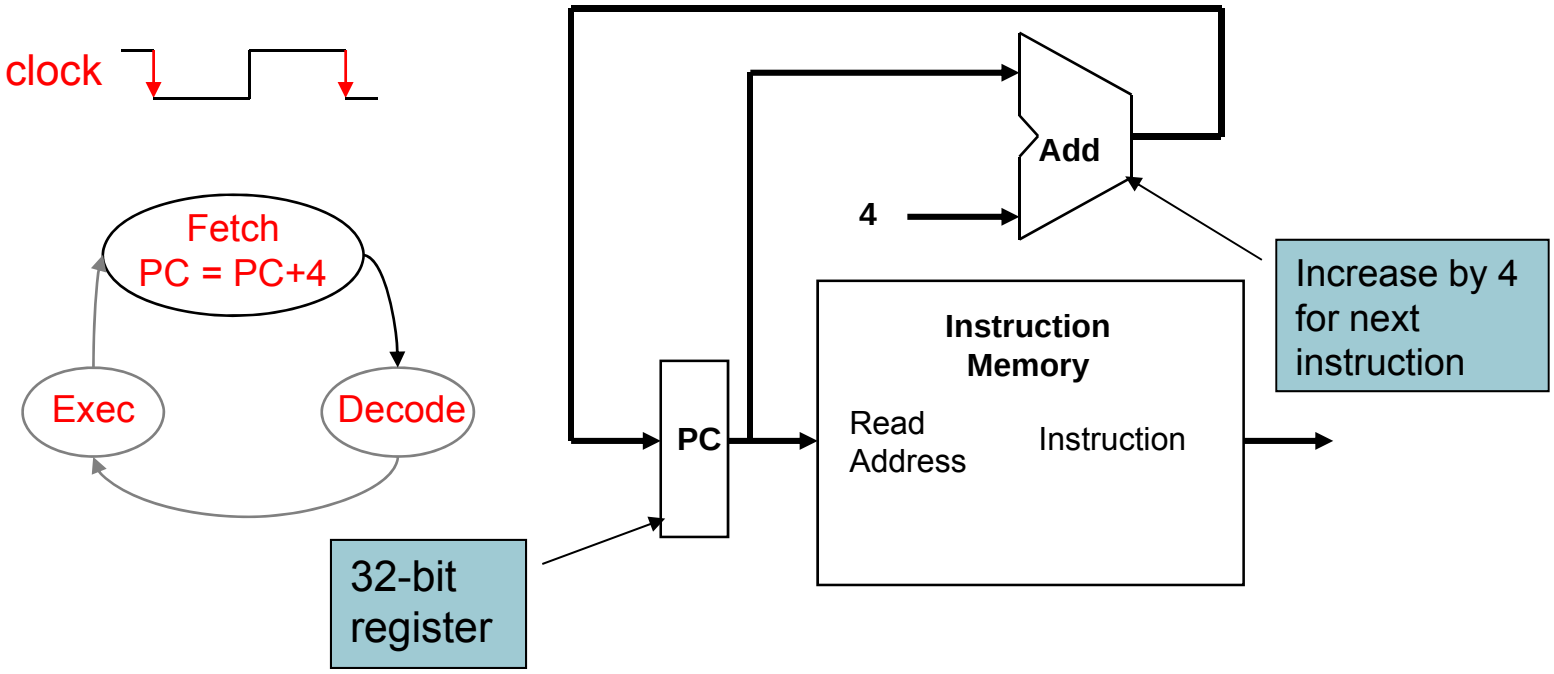
- Fetching instructions involves
  - reading the instruction from the Instruction Memory
  - updating the PC value to be the address of the next (sequential) instruction



Hardware corresponding to fetch

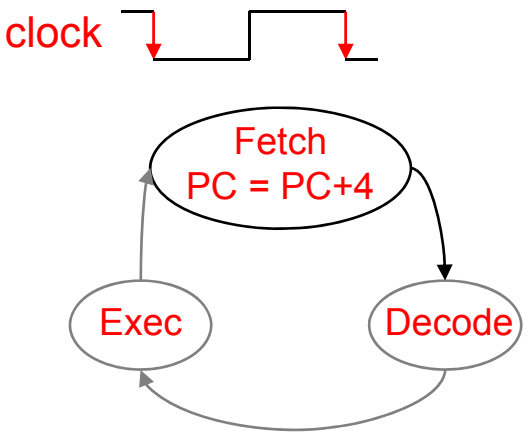
Fetching instruction

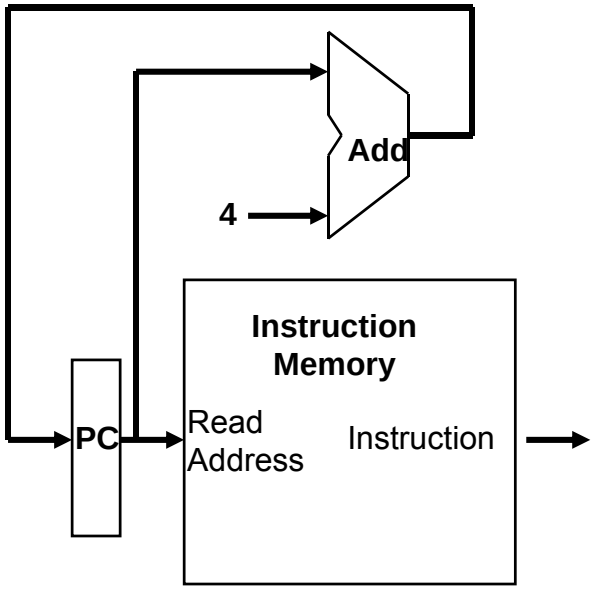
- Fetching instructions involves
  - reading the instruction from the Instruction Memory
  - updating the PC value to be the address of the next (sequential) instruction



Fetching instruction control

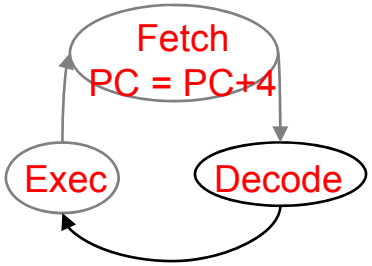
- PC is updated each clock cycle; no need for an explicit write control signal just a clock signal
- Reading from the Instruction Memory is a combinational activity; no explicit read control needed





Decoding instruction

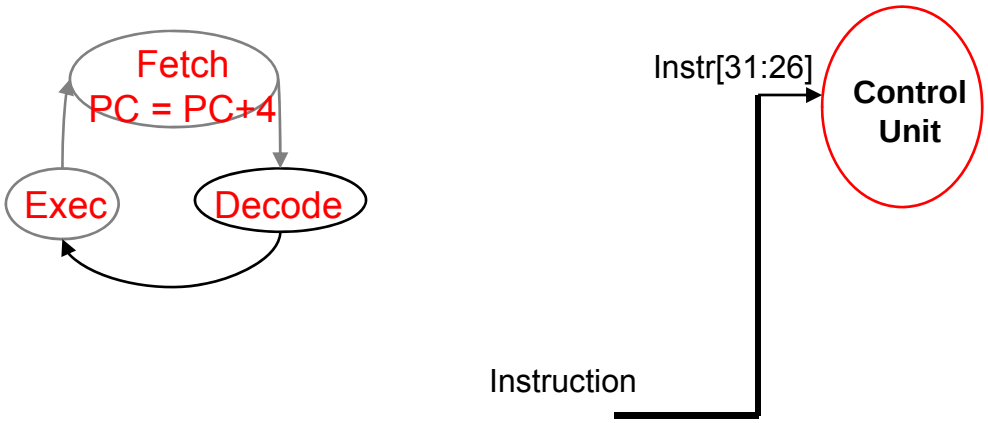
Send fetched instruction opcode and function field bits to the control unit



Hardware?

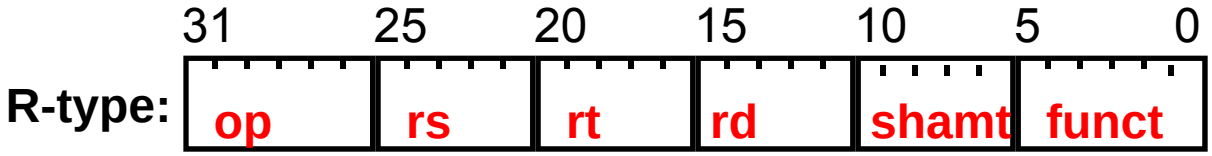
Decoding instruction

Send fetched instruction opcode and function field bits to the control unit

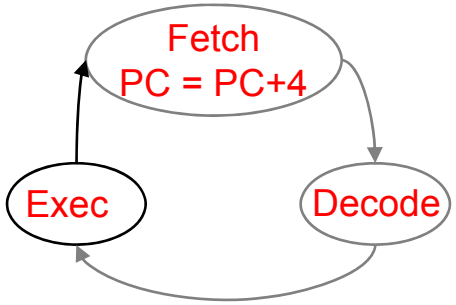


Executing R format instruction

- R format instructions (add, sub, slt, and, or)



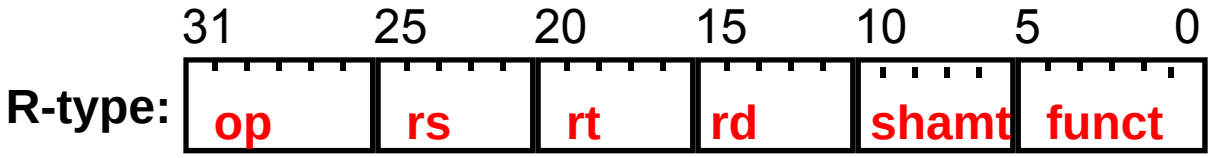
  - perform operation (op, funct) on values in rs and rt
  - store the result into the Register File location rd



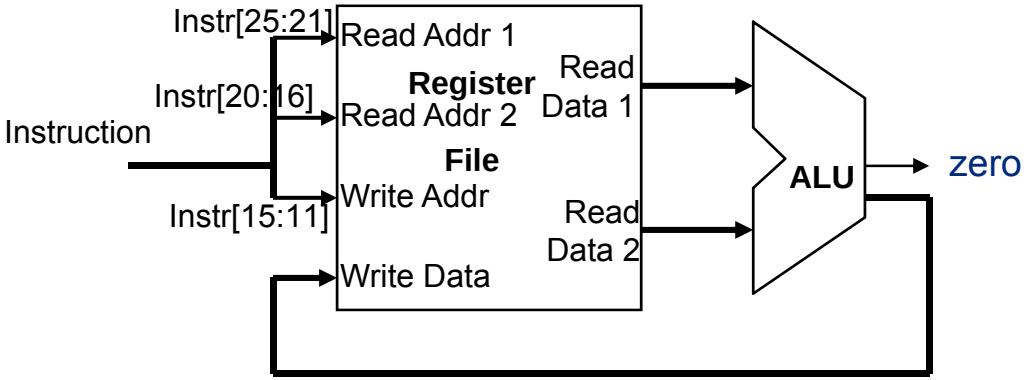
Hardware?

R format Instruction datapath

- R format instructions (add, sub, slt, and, or)



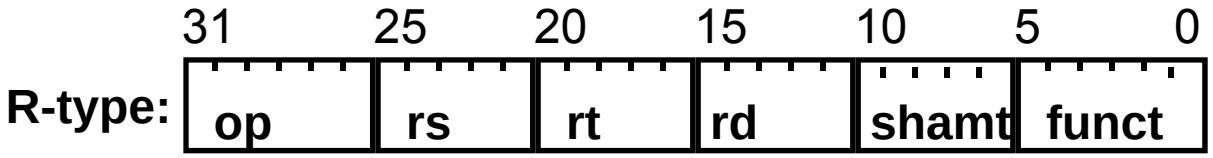
  - perform operation (op,funct) on values in rs and rt
  - store the result into the Register File location rd



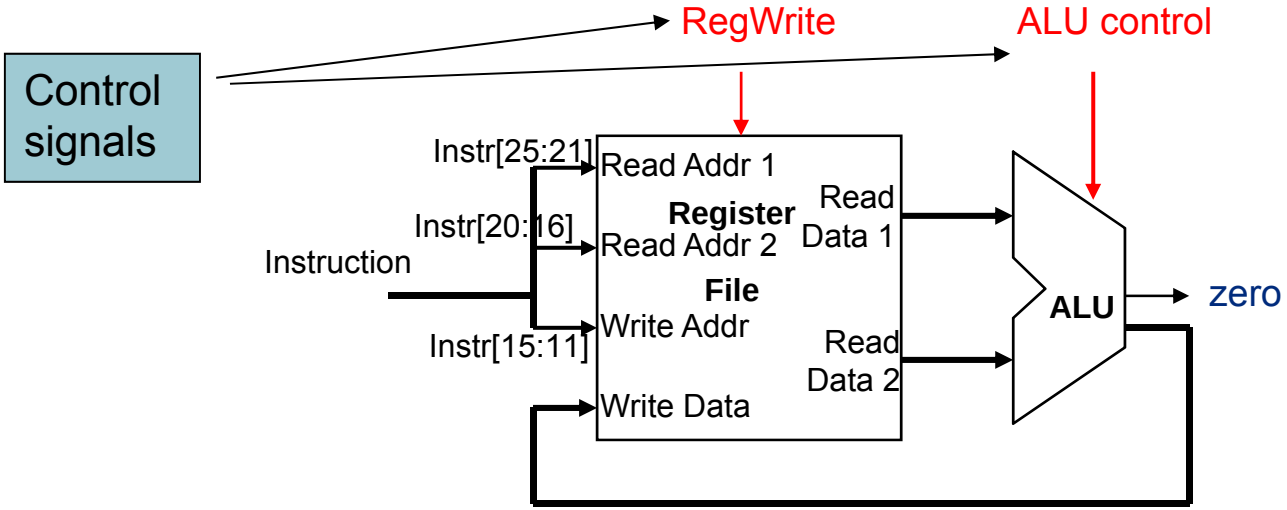
Datapath R-type instruction

R format instruction control

- R format instructions (add, sub, slt, and, or)

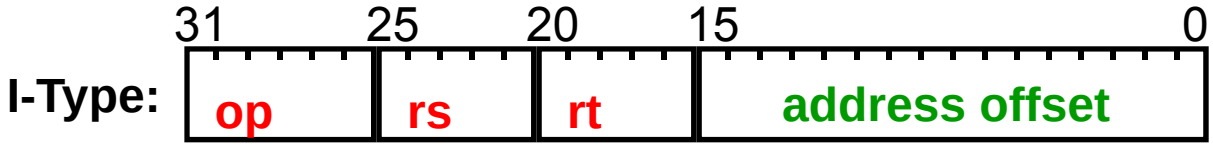


  - perform operation (op, funct) on values in rs and rt
  - store the result into Register File location rd



Executing Load and Store instructions

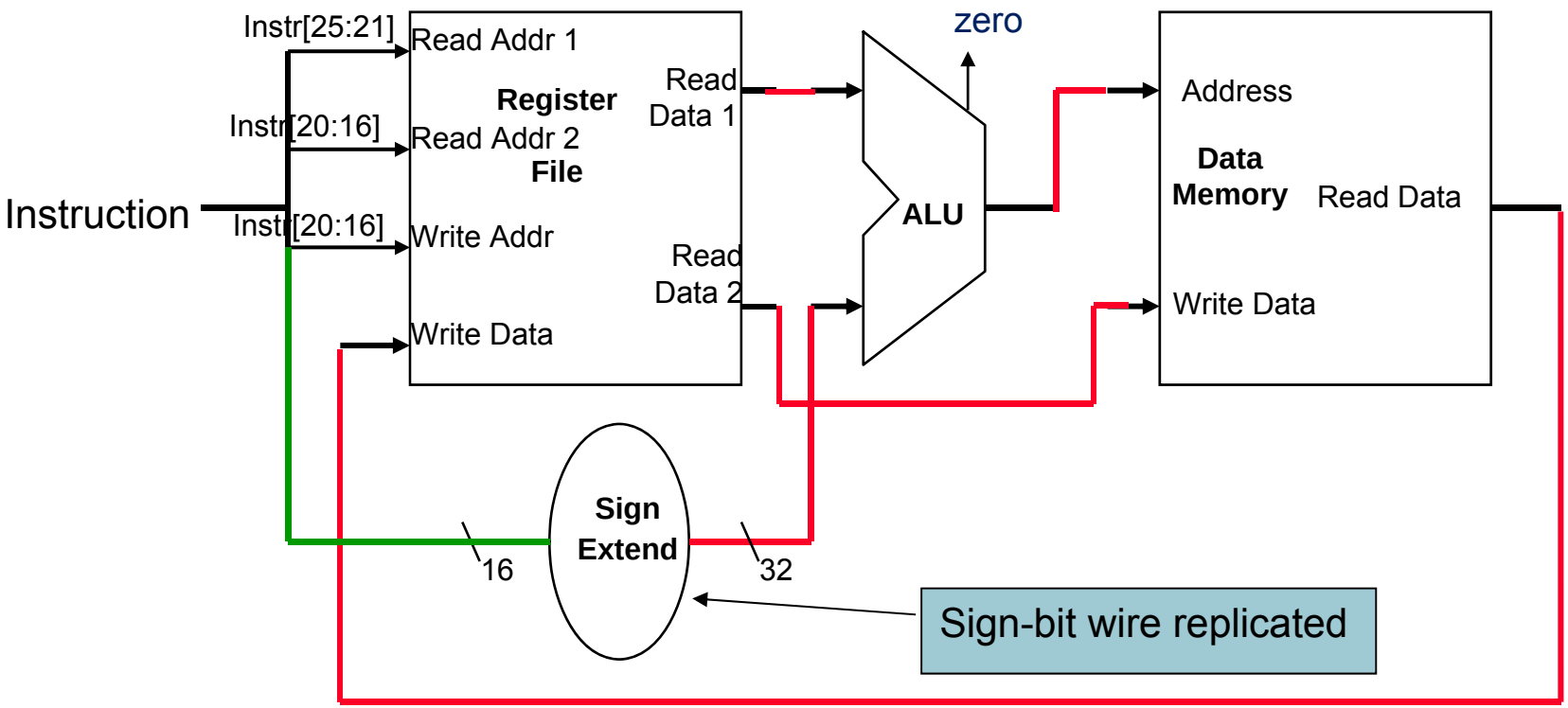
- Load and store instructions format:

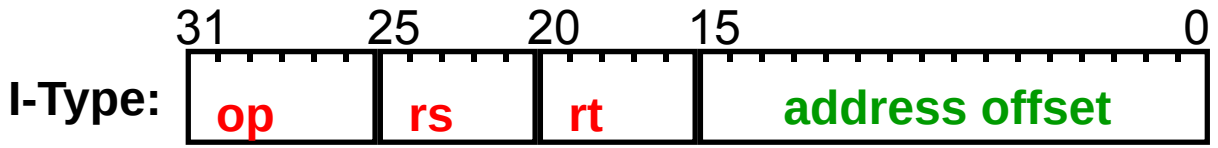


  - compute memory address by adding base register value to 16-bit signed-extended value in offset field
  - store value read from the Register File and write it into the Data Memory
  - load value, read from the Data Memory, and write it into the Register File

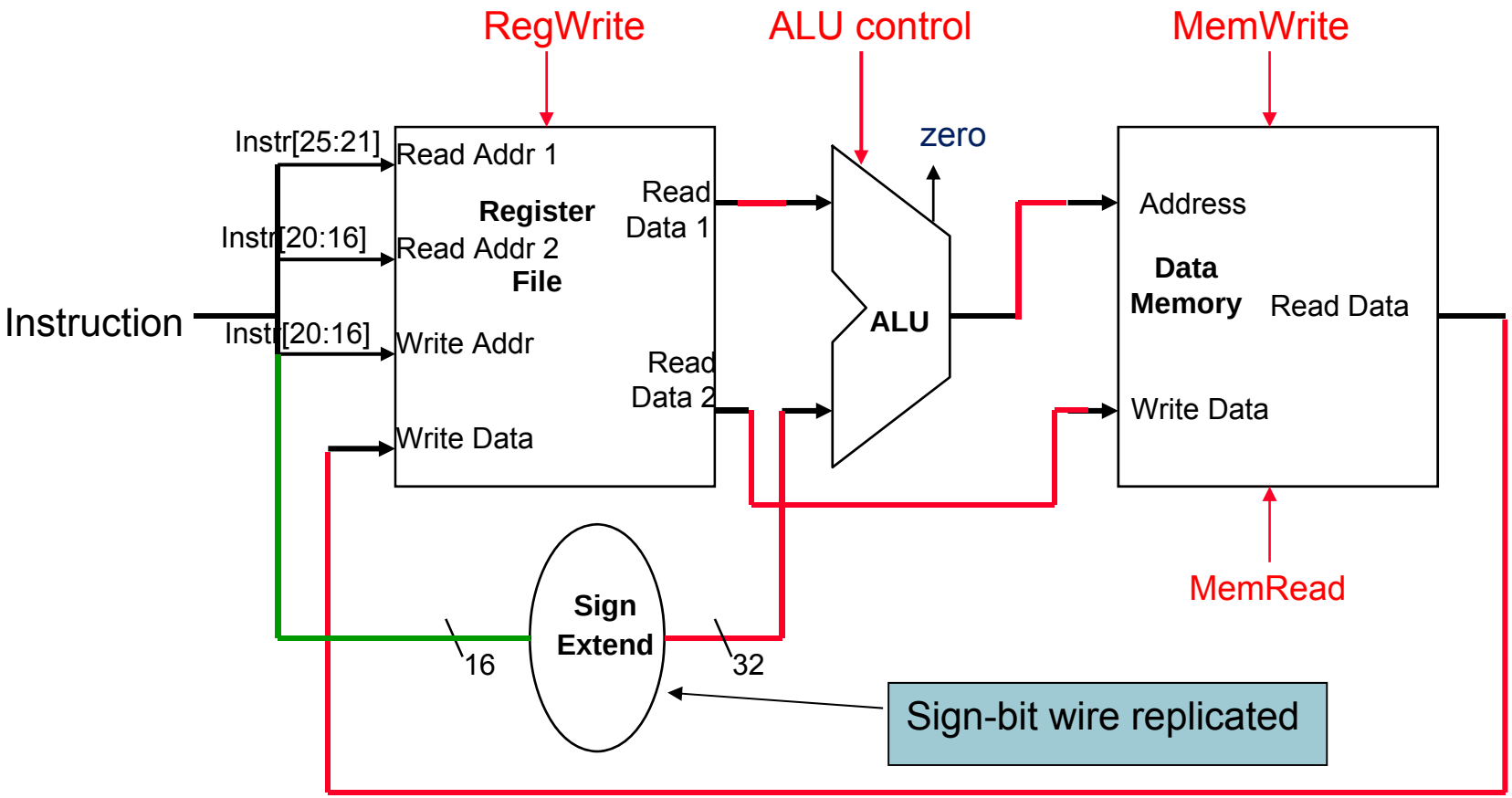
What is sign extension?

Load and Store instruction hardware



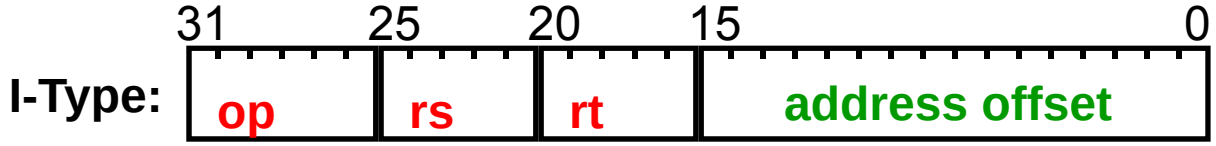


Load and Store instruction control



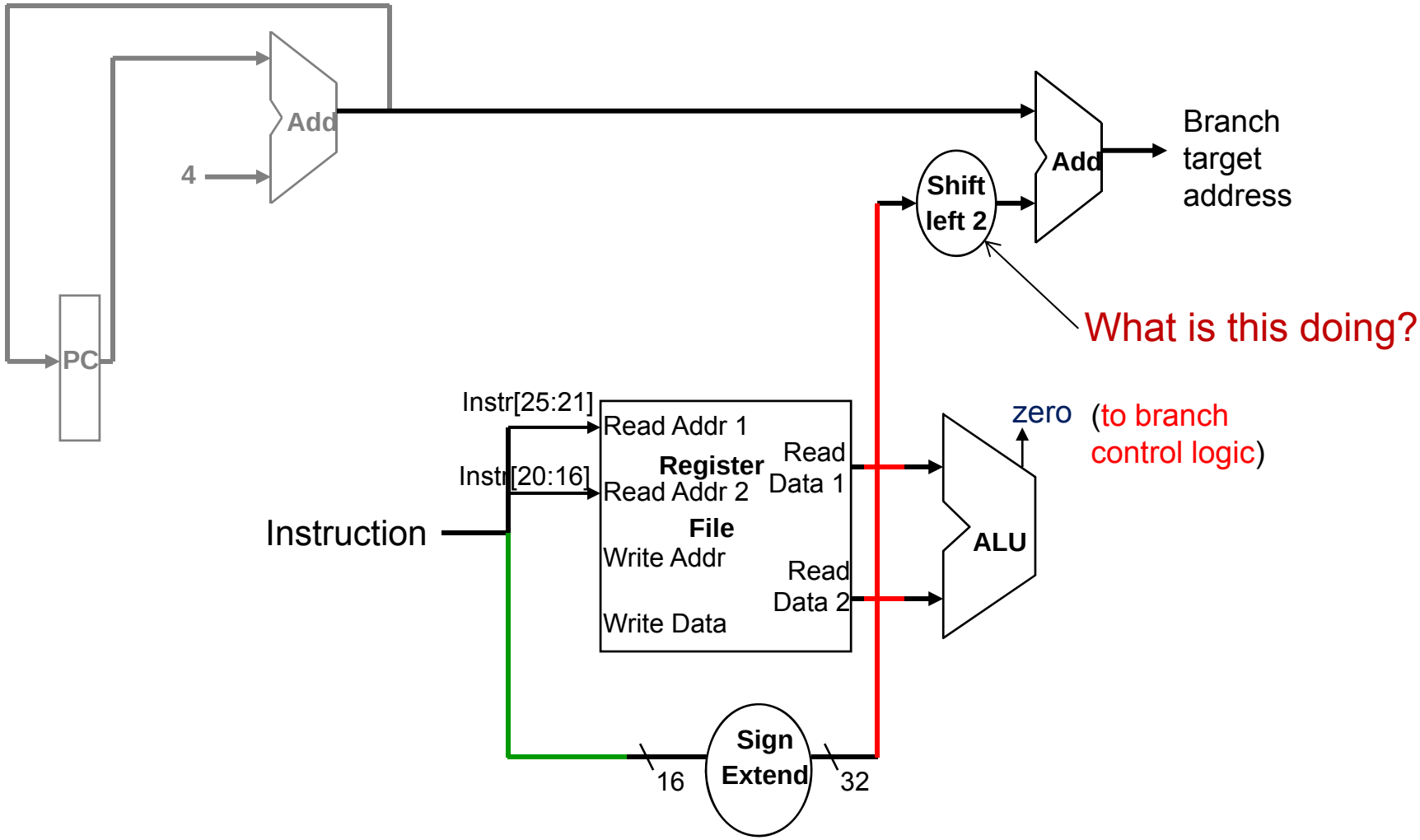
Executing Branch Instructions

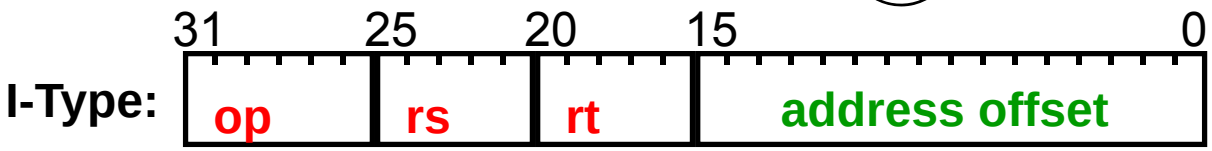
- Branch instruction format:



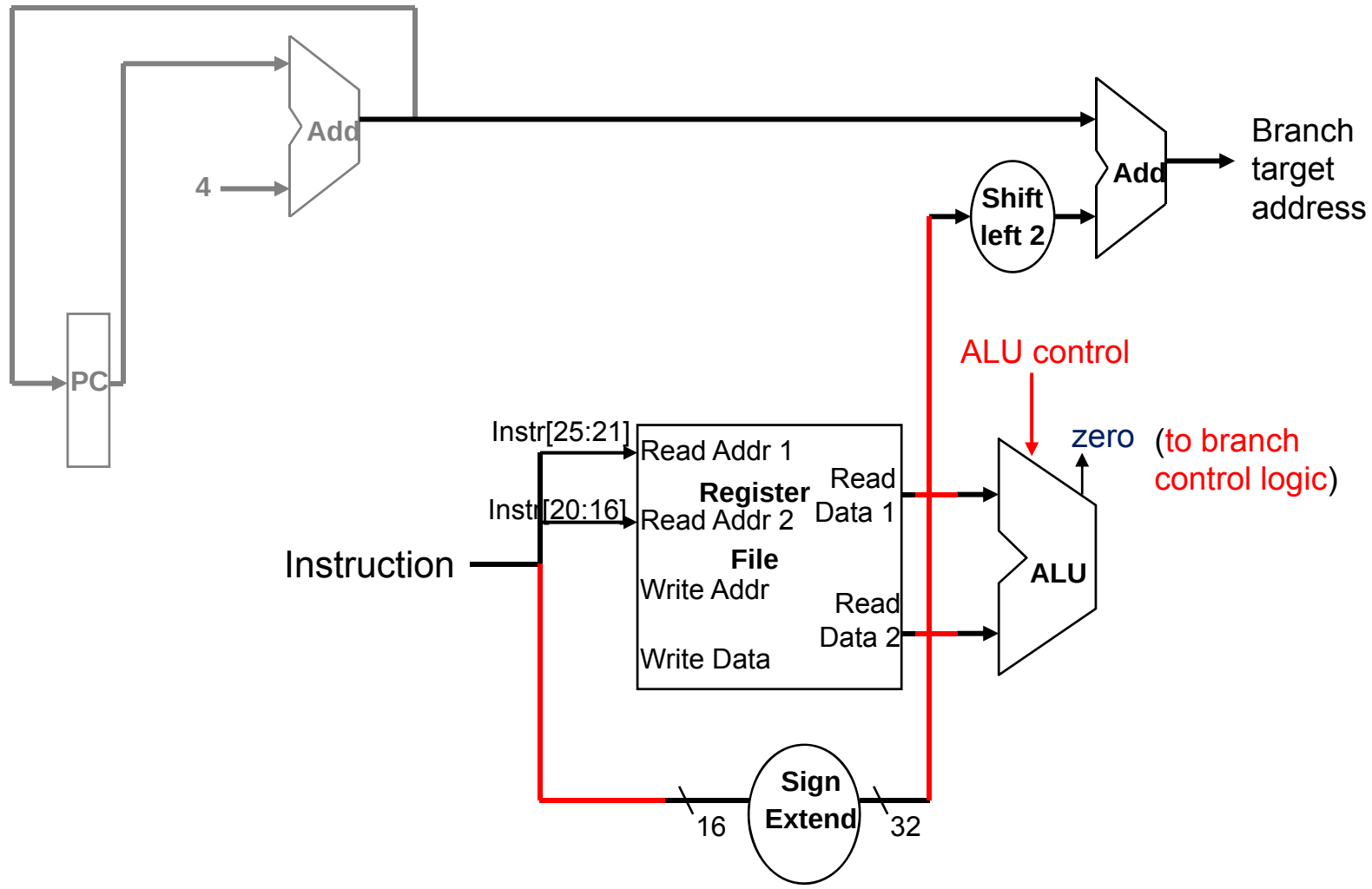
  - compare the operands read from Register File adresses rs and rt on equality (**`zero`** ALU output)
  - compute branch target address by adding the updated PC ← PC+4 to16-bit sign-extended value in offset field

Branch instruction hardware





Branch instruction control



No writing in register file and other guys do not require more control.

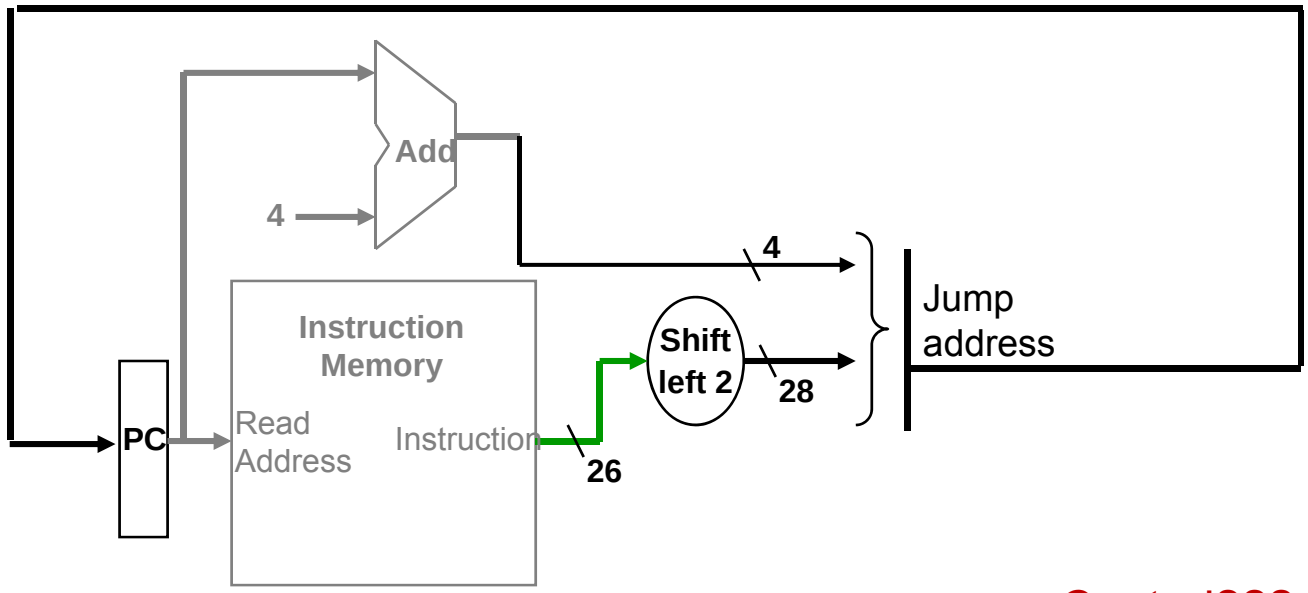
Executing Jump instructions

- Jump instruction format:

take 26 lower bits of fetched instruction, shift left by 2 bits, connect to 4 most significant bits of PC

Jump instruction hardware

take 26 lower bits of fetched instruction, shift left by 2 bits, connect to 4 most significant bits of PC

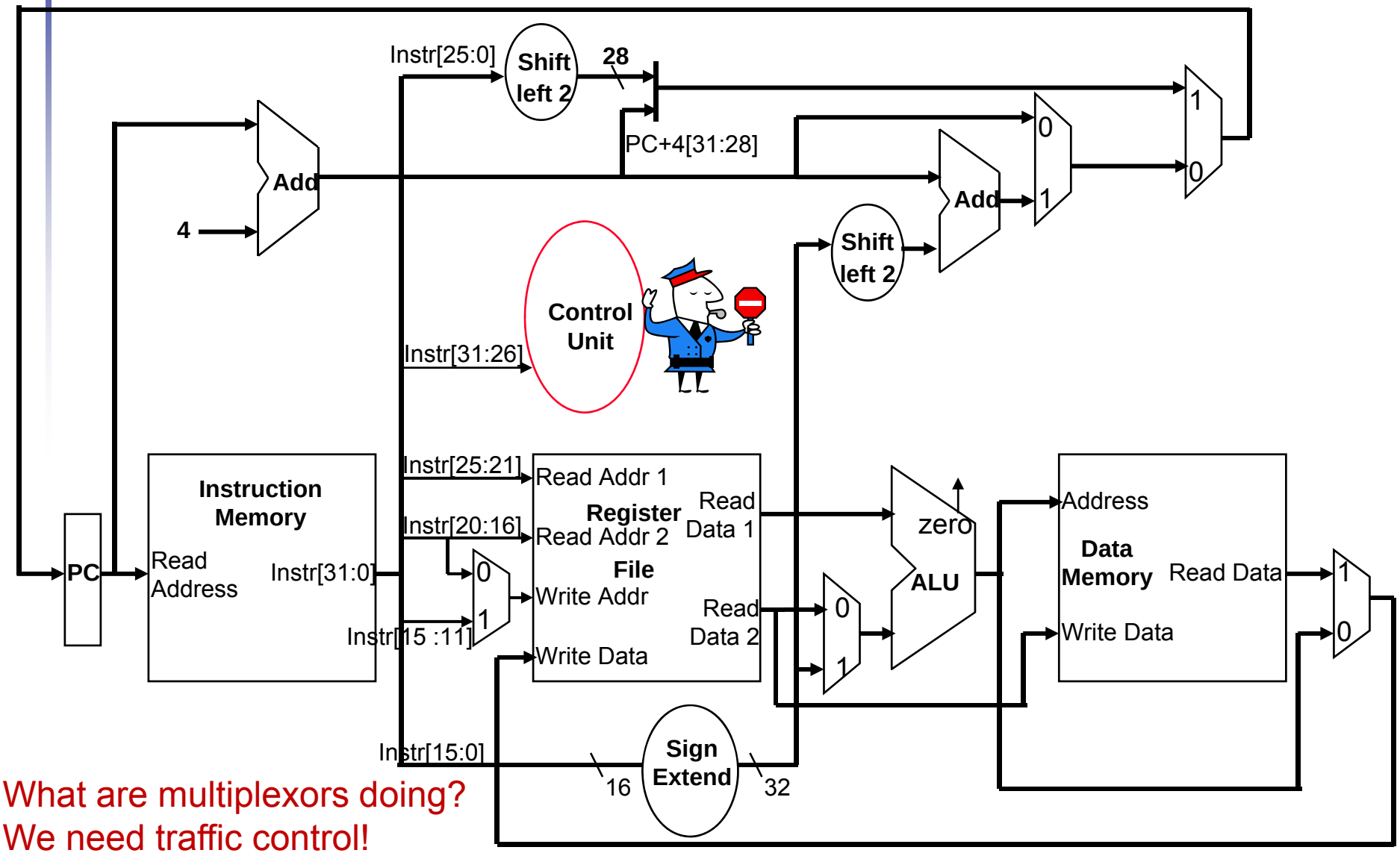


Control???

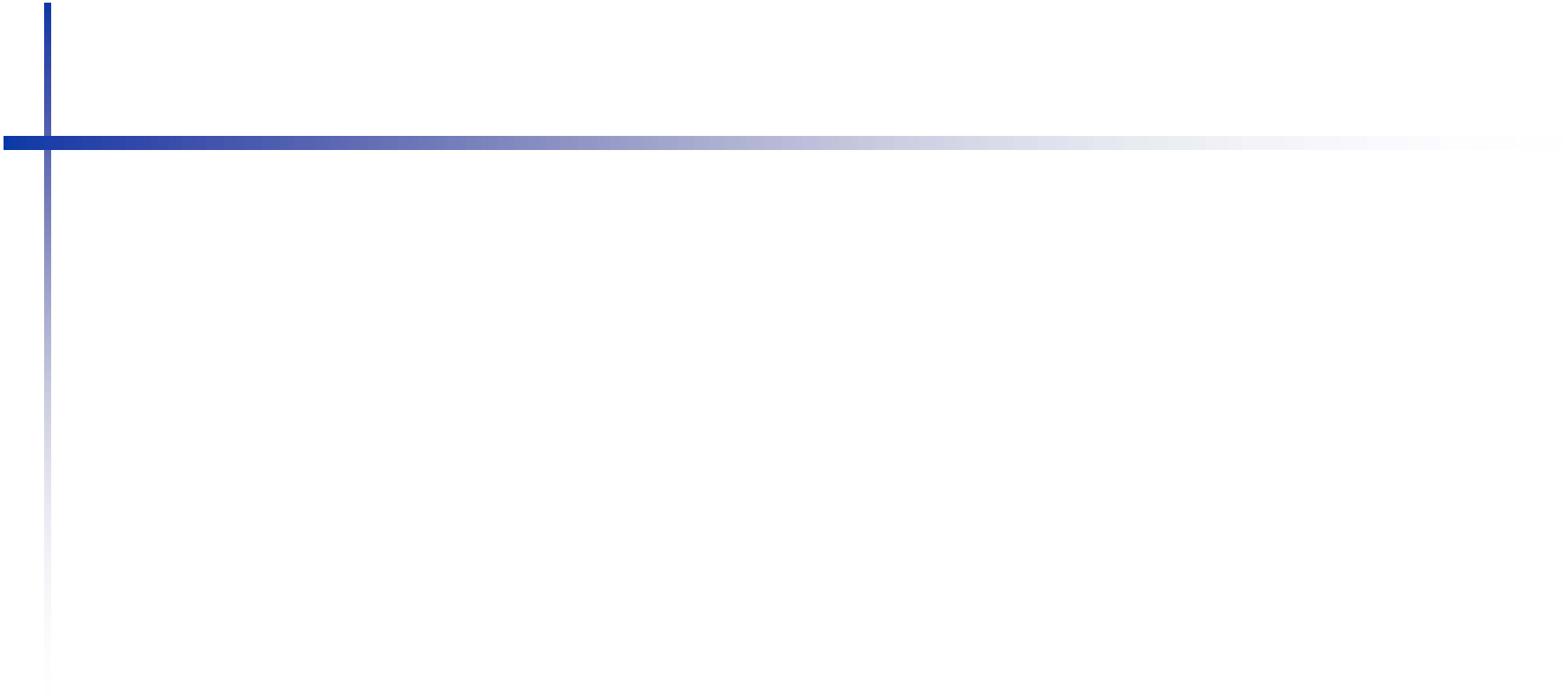
Creating a single datapath

- Assemble datapath segments and add control lines and multiplexors when needed
- Single cycle design – fetch, decode and execute each instruction in one clock cycle
  - no datapath resource can be used more than once per instruction, so some must be duplicated, e.g. separate Instruction Memory and Data Memory, several adders
  - multiplexors needed at the input of shared elements with control lines to do the selection
  - write signals to control writing to Register File and Data Memory
- Cycle time is determined by length of the longest path, alternatively: by duration slowest instruction

Full datapath



What are multiplexors doing?
We need traffic control!



Building the control unit, Section 4.4

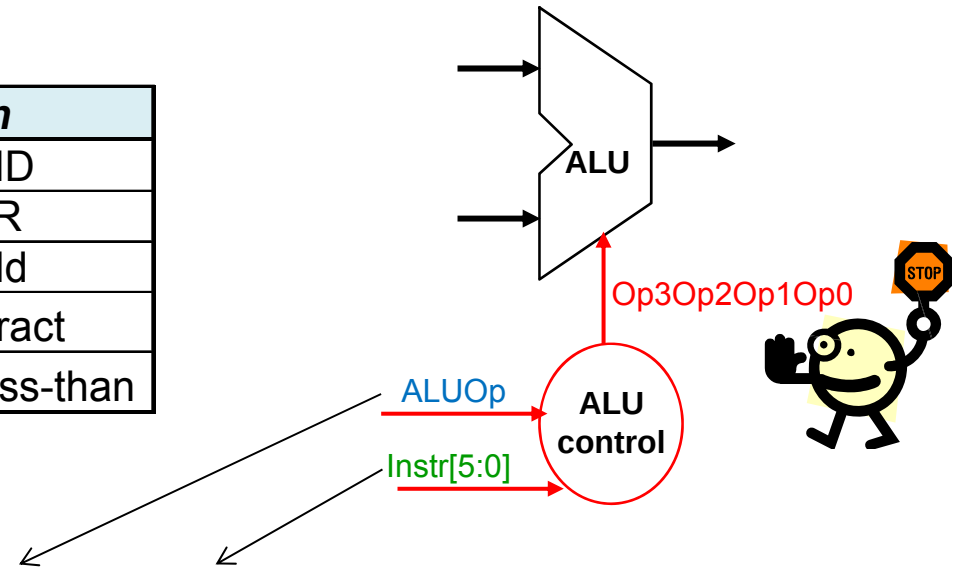
SINGLE-CYCLE PROCESSOR IMPLEMENTATION

The Control

- The main control unit
  - selects the operations to perform (ALU, Register File and Memory read/write)
  - generates the signals to control the flow of data (multiplexor inputs)
- ALU control used for
  - Load/Store: Alu operation → add
  - Branch: Alu operation → subtract
  - R-type: Alu op. depends on funct field

ALU Control

ALU Control	*Function*
0000	AND
0001	OR
0010	Add
0110	subtract
0111	set-on-less-than



Instruction opcode	*ALUOp*	*Field Funct.*	*Control input ALU*	*Operation ALU*
LW and SW	00	xxxxxx	0010	Add
Branch equal	01	xxxxxx	0110	Subtract
R-type	10	100000	0010	Add
		100010	0110	Subtract
		100100	0000	AND
		100101	0001	OR
		101010	0111	Set on less than

ALU Control

INPUTS								OUTPUTS
ALUOp		Field Function						ALU input
ALU Op1	ALU Op0	F5	F4	F3	F2	F1	F0	OP OP3 OP2 OP1 OP0
0	0	x	x	x	x	x	x	0 0 1 0
0	1	x	x	x	x	x	x	0 1 1 0
1	x	x	x	0	0	0	0	0 0 1 0
1	x	x	x	0	0	1	0	0 1 1 0
1	x	x	x	0	1	0	0	0 0 0 0
1	x	x	x	0	1	0	1	0 0 0 1
1	x	x	x	1	0	1	0	0 1 1 1

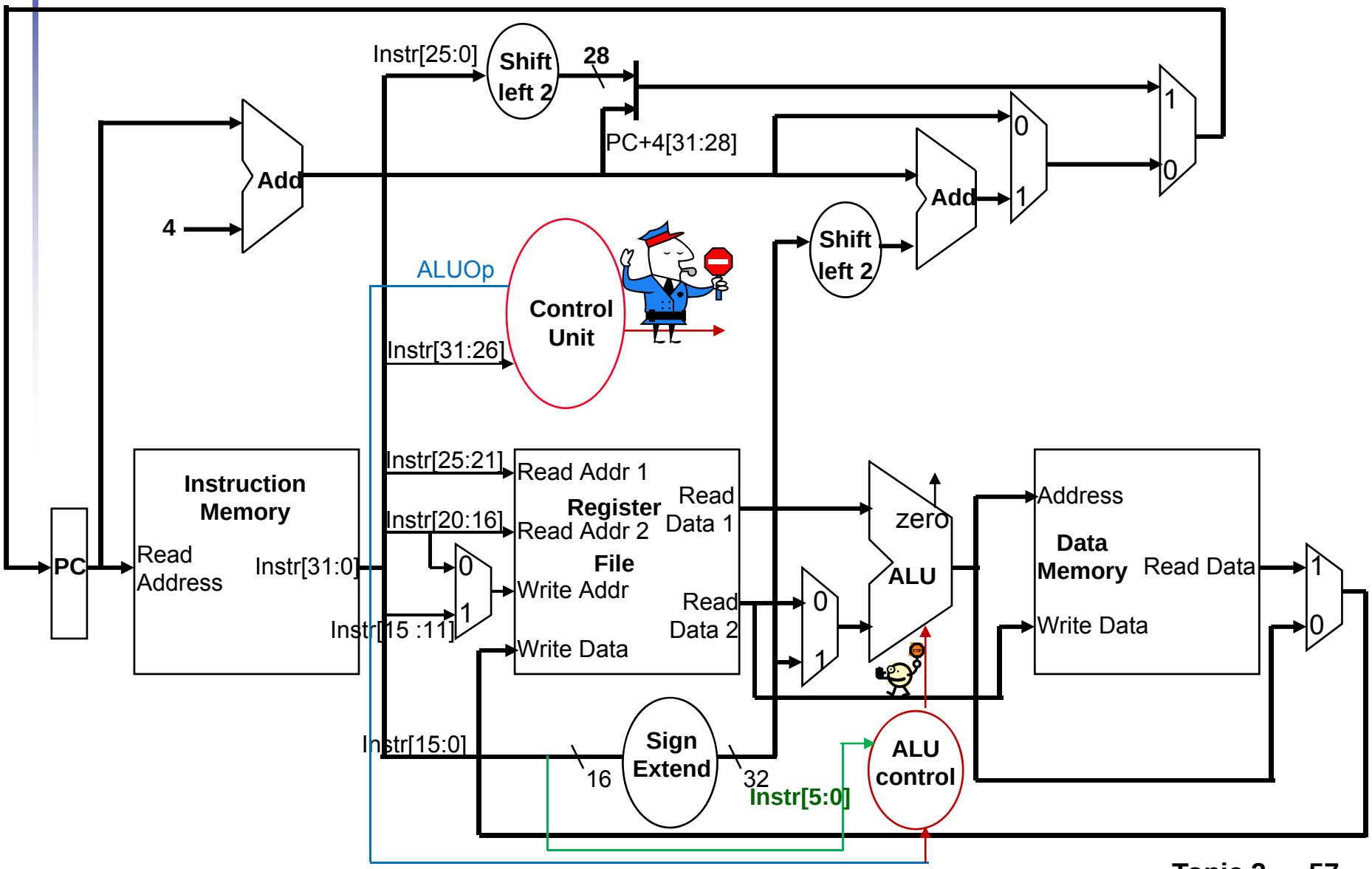
Truth table analysis

$$OP0 = ALUOp1 \cdot (F3+F0)$$

$$OP1 = \overline{ALUOp1} + \overline{F2}$$

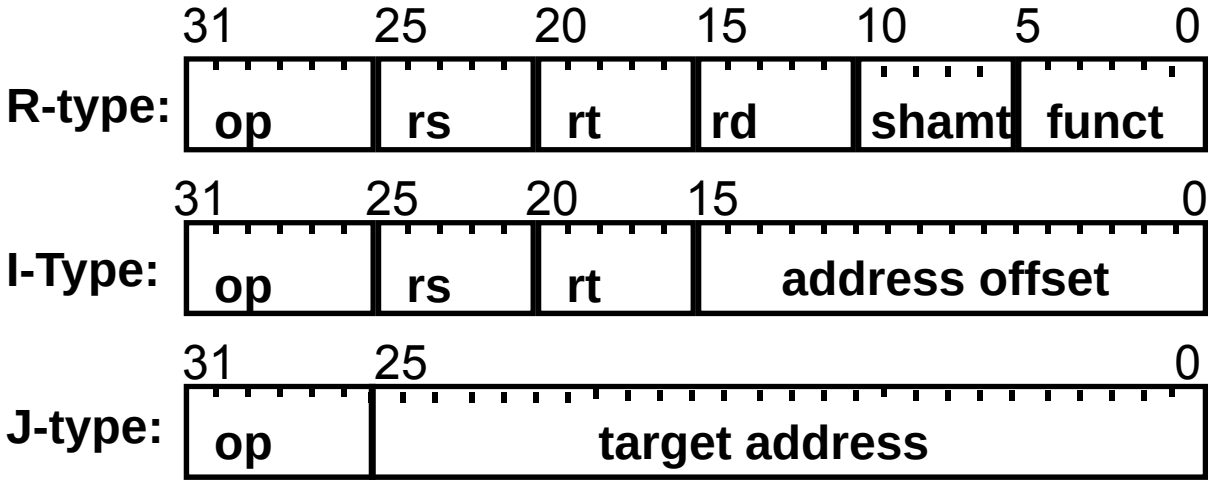
$$OP2 = ALUOp0 + ALUOp1 \cdot F1$$

Relation with main control unit



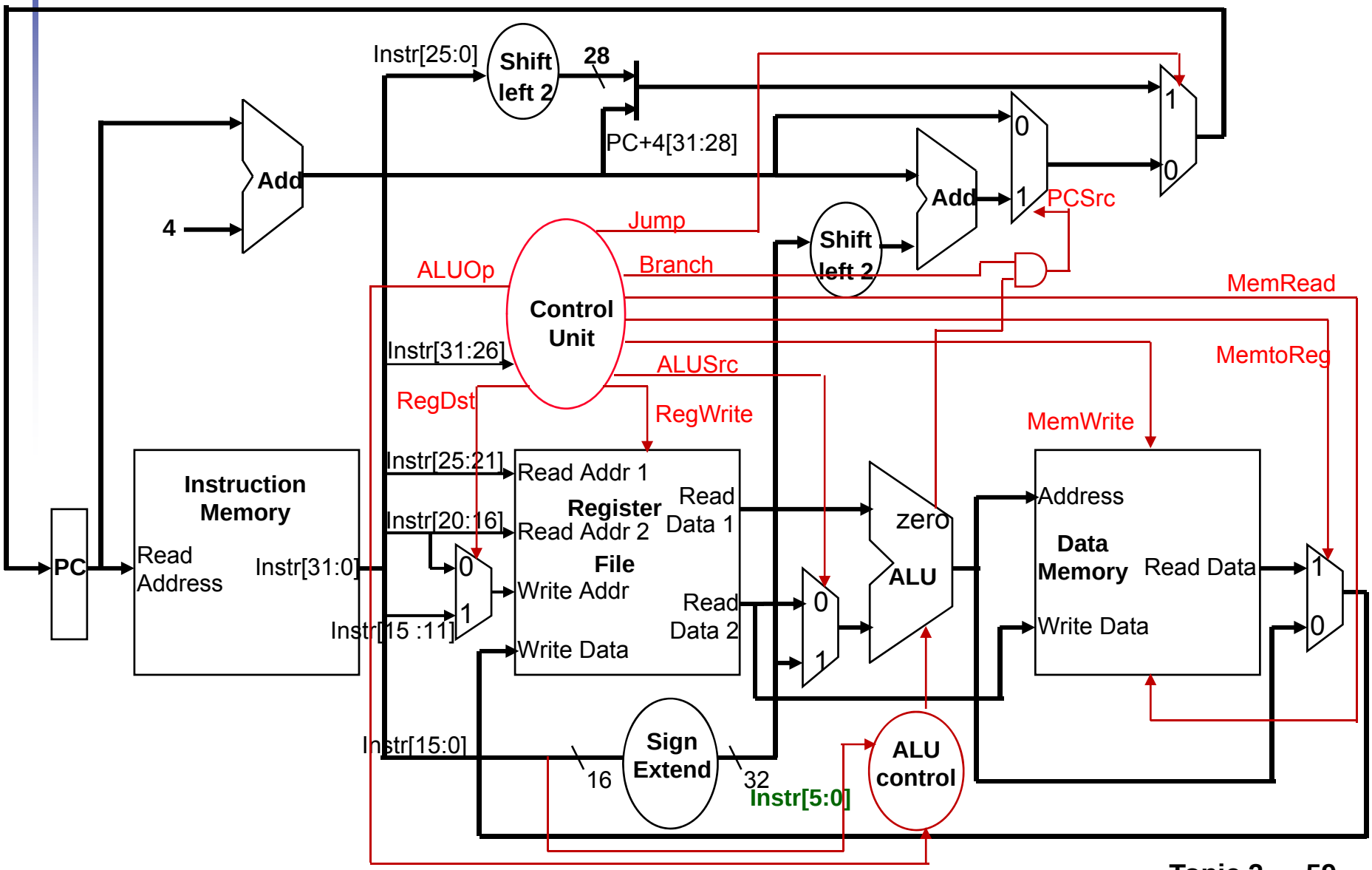
Adding the Control

- Selecting the signals to be activated (ALU, Register File and Memory read/write)
- Controlling the flow of data (multiplexor inputs)

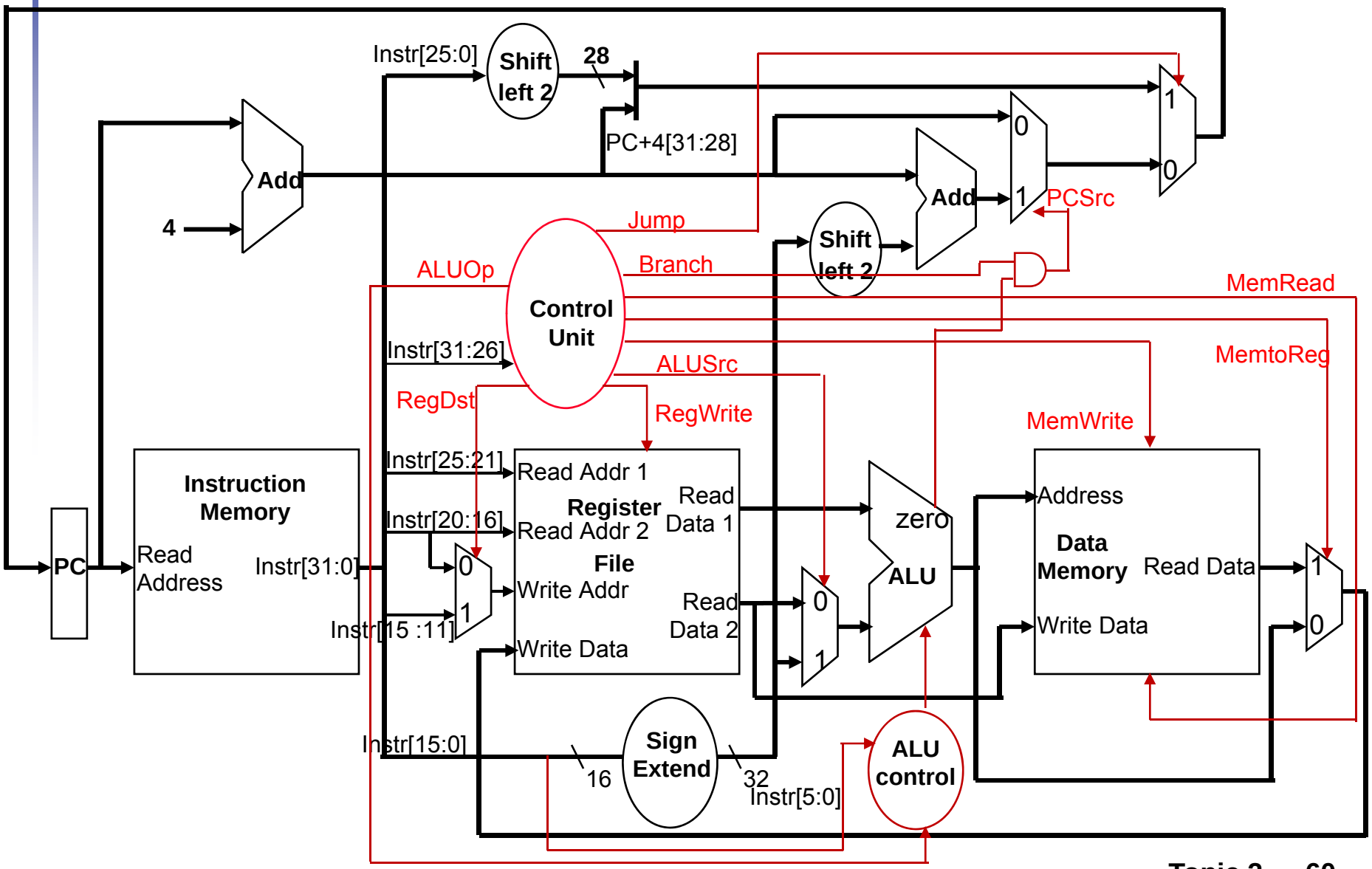


- Observations
  - op field always in bits 31-26
  - addr of registers to be read are in rs field (bits 25-21) and rt field (bits 20-16)
  - addr. of register to be written in rt (bits 20-16) for lw or in rd (bits 15-11) for R-type instructions
  - offset for beq, lw, and sw always in bits 15-0

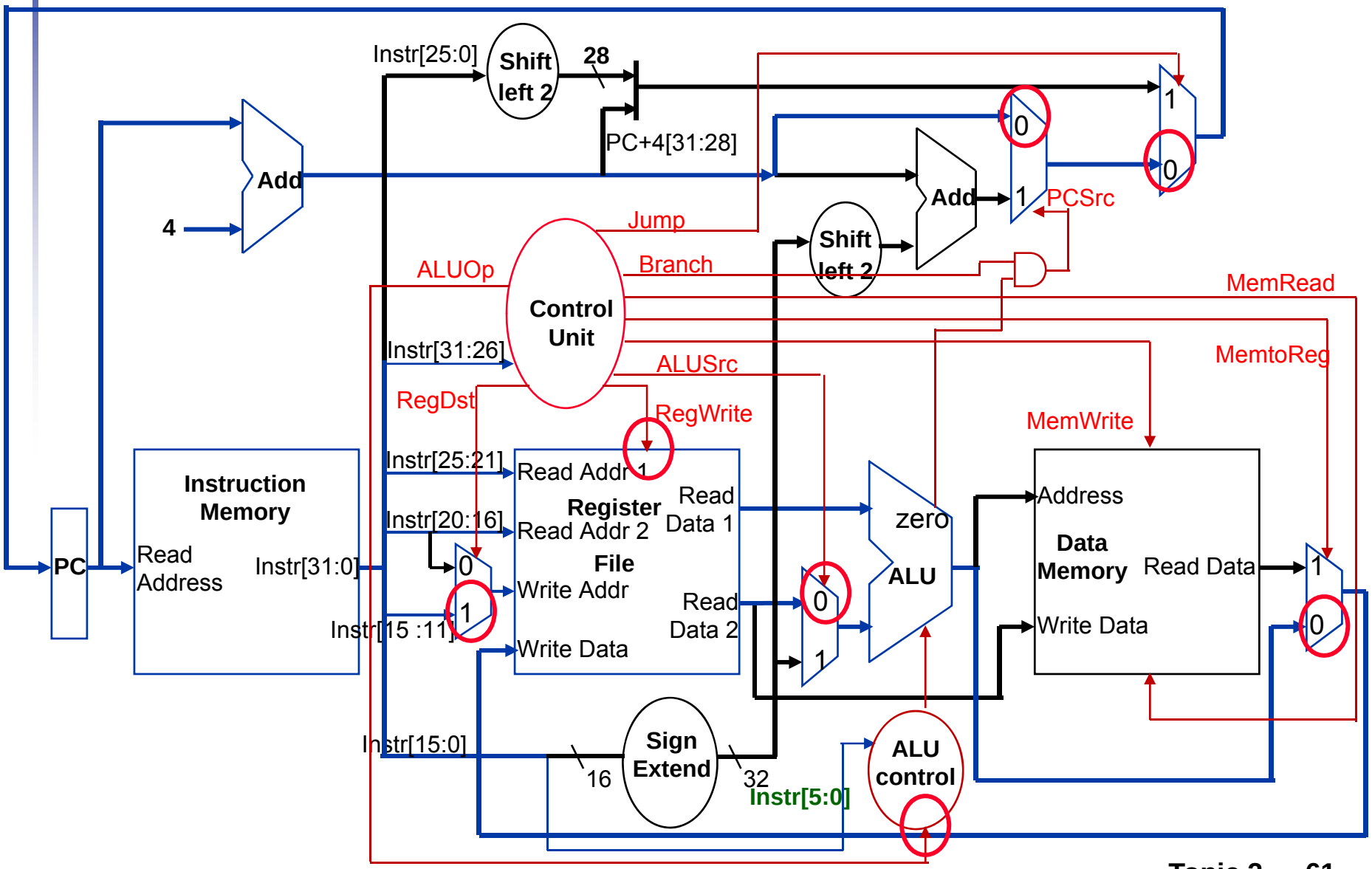
Single Cycle Datapath with Control Unit



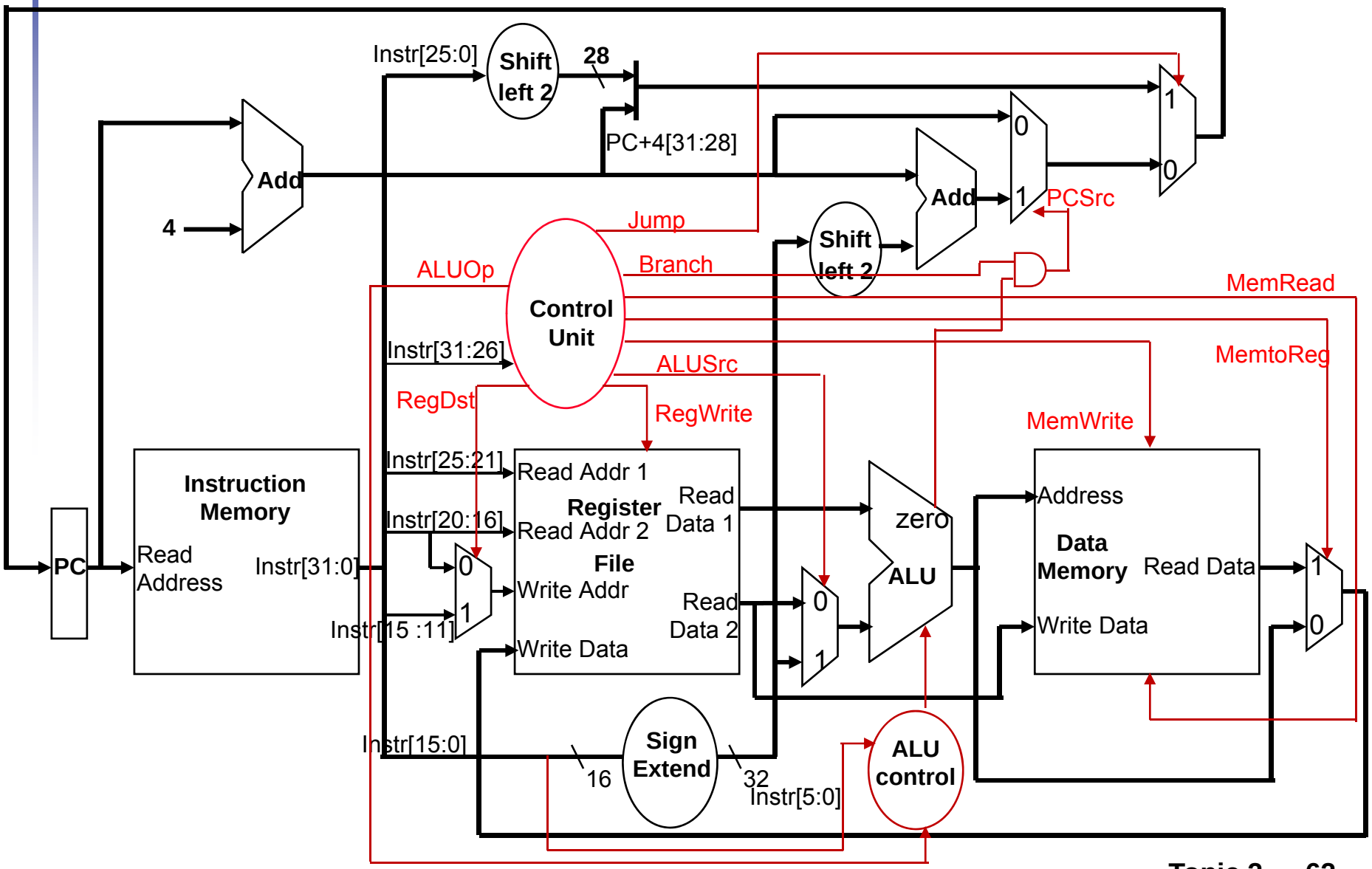
R-type instruction which signals activated?



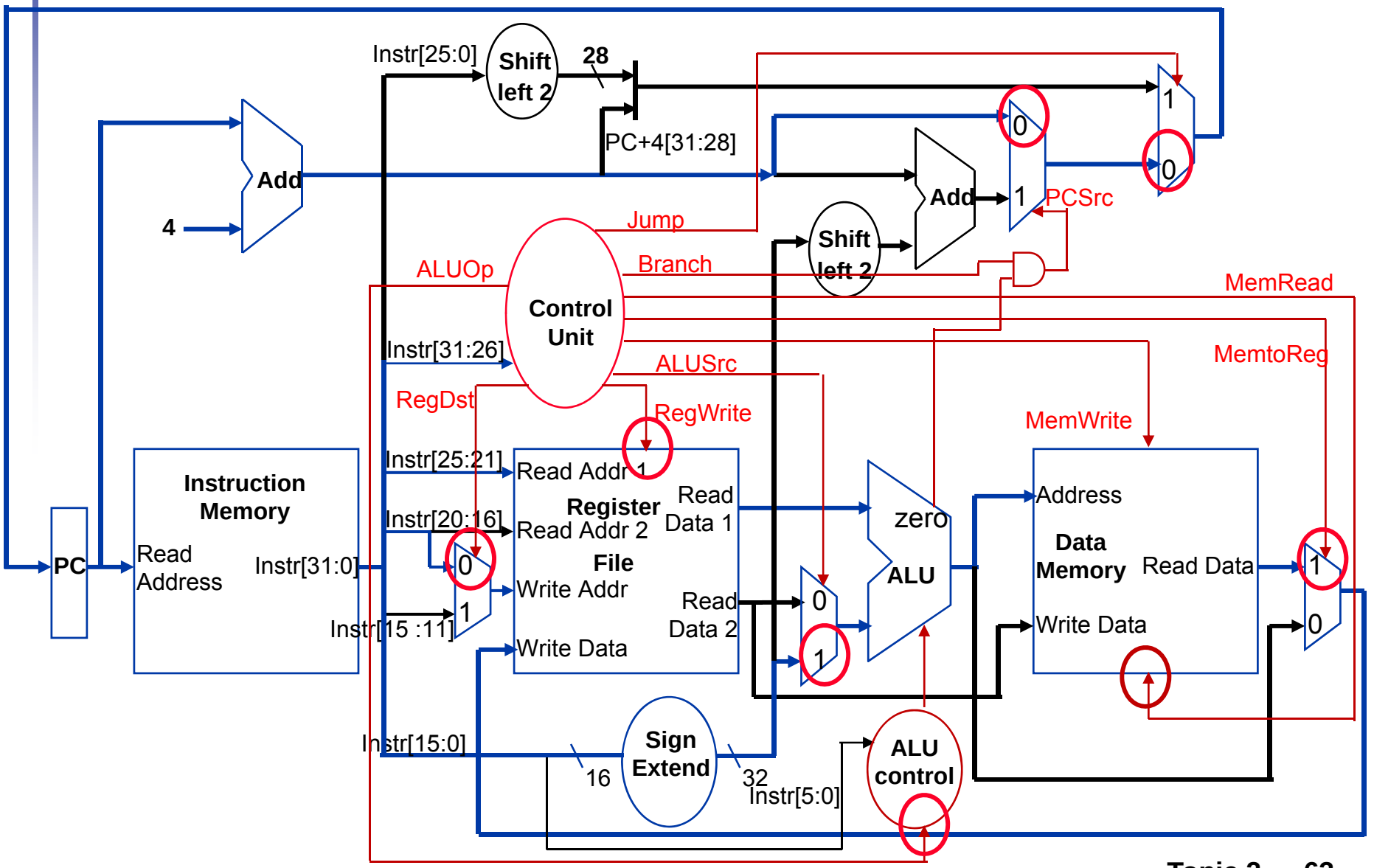
R-type instruction data/control flow



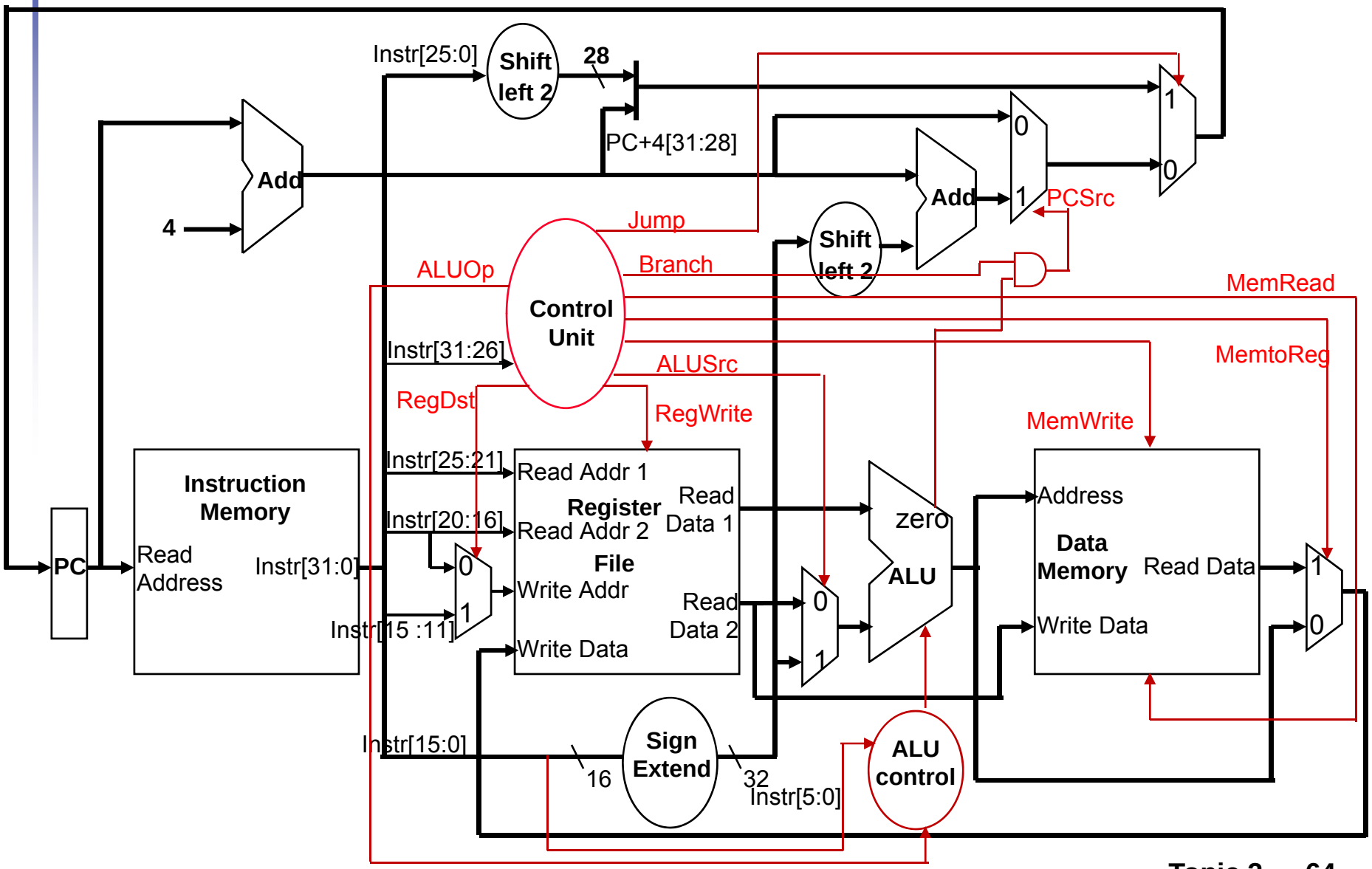
Load Word instruction data/control flow



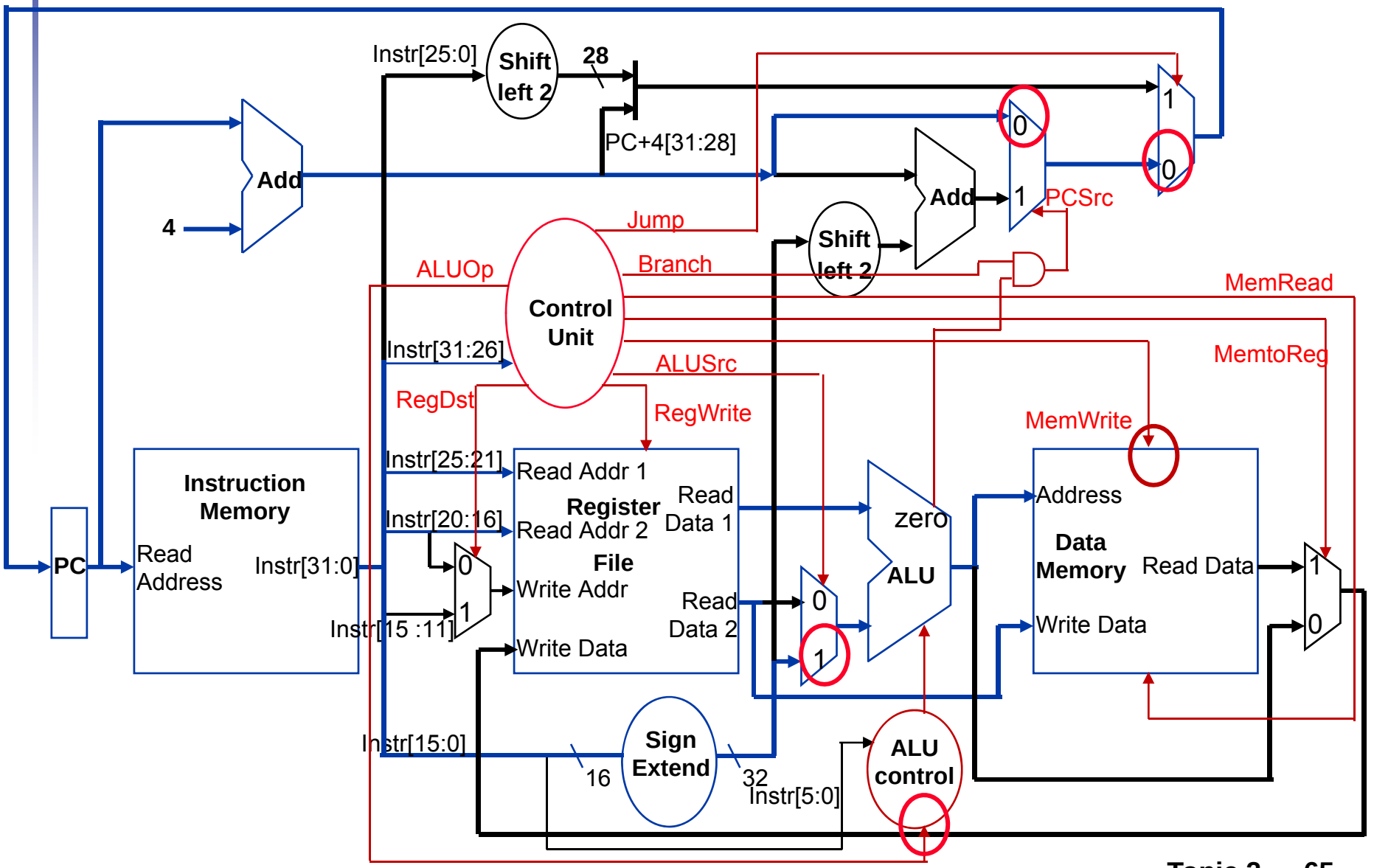
Load Word instruction data/control flow



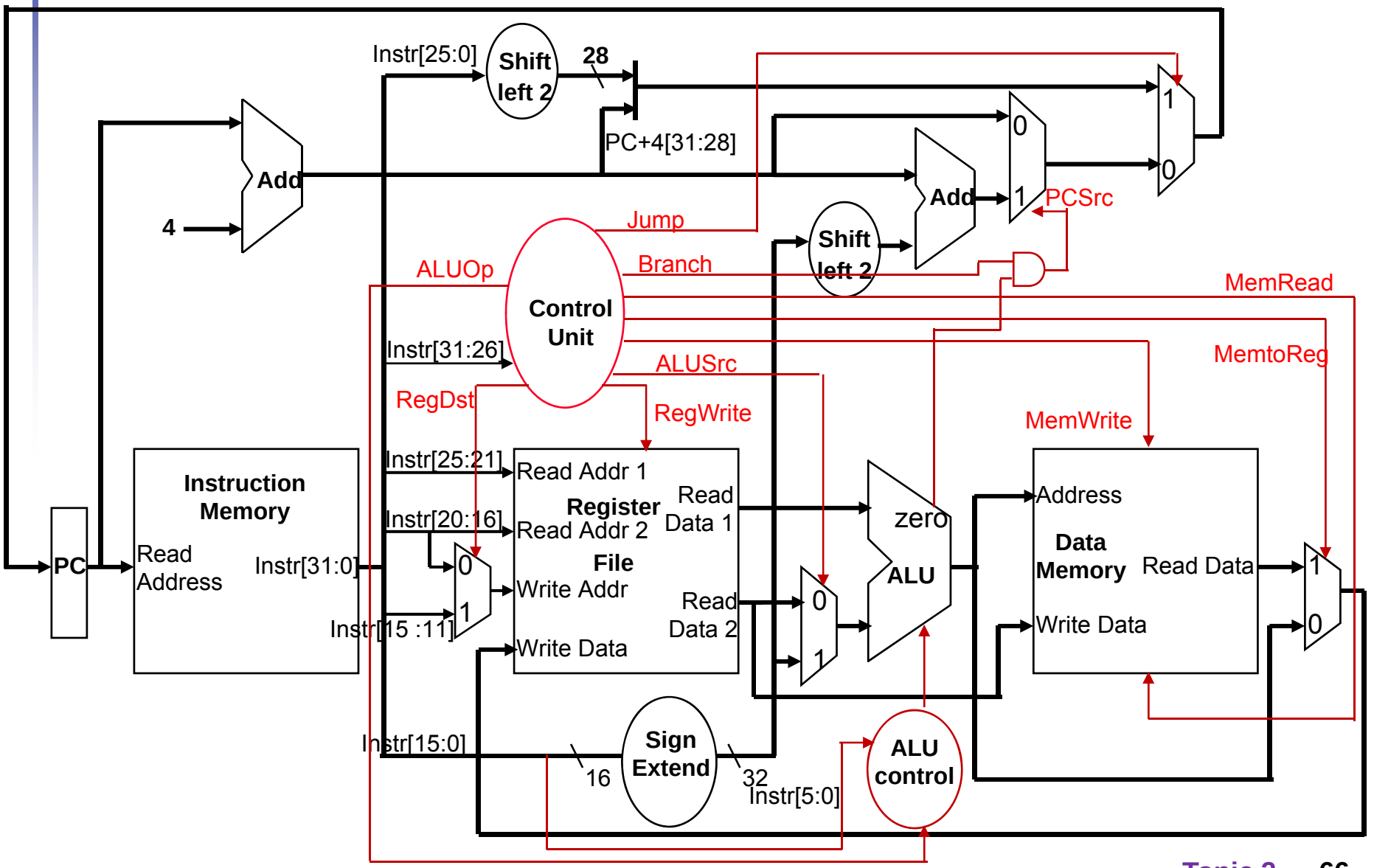
Store Word instruction data/control flow



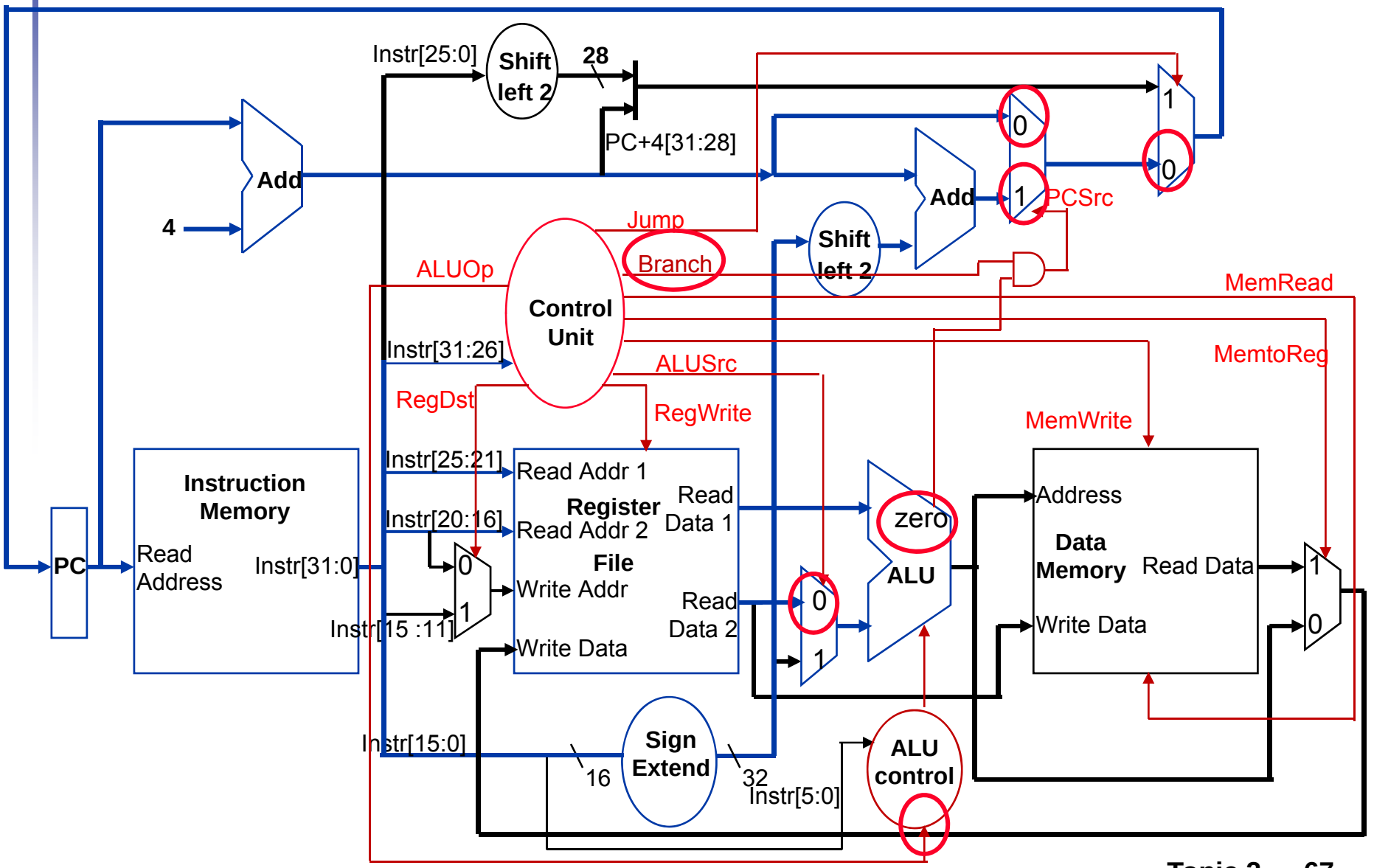
Store Word instruction data/control flow



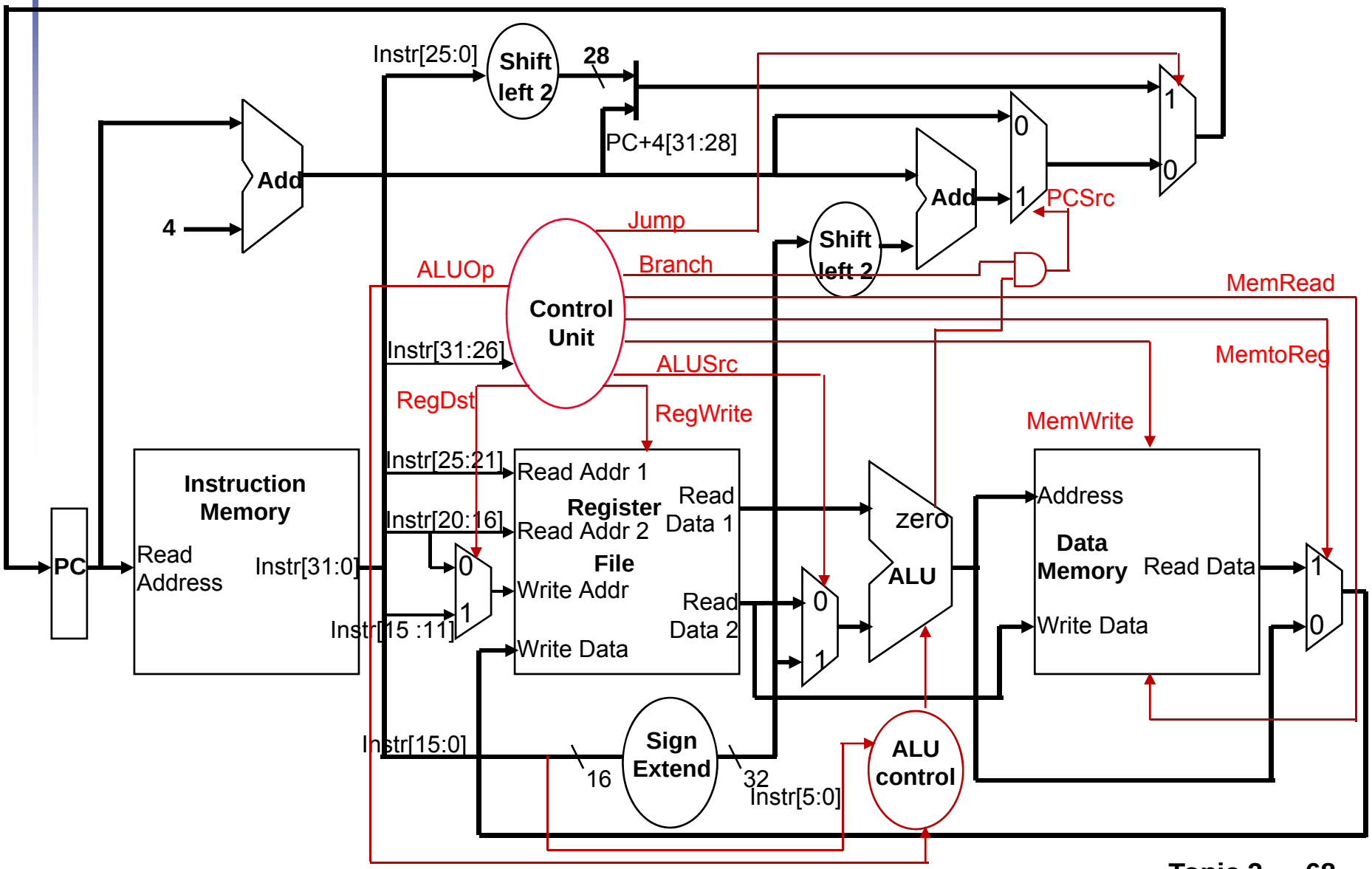
Branch instruction data/control flow



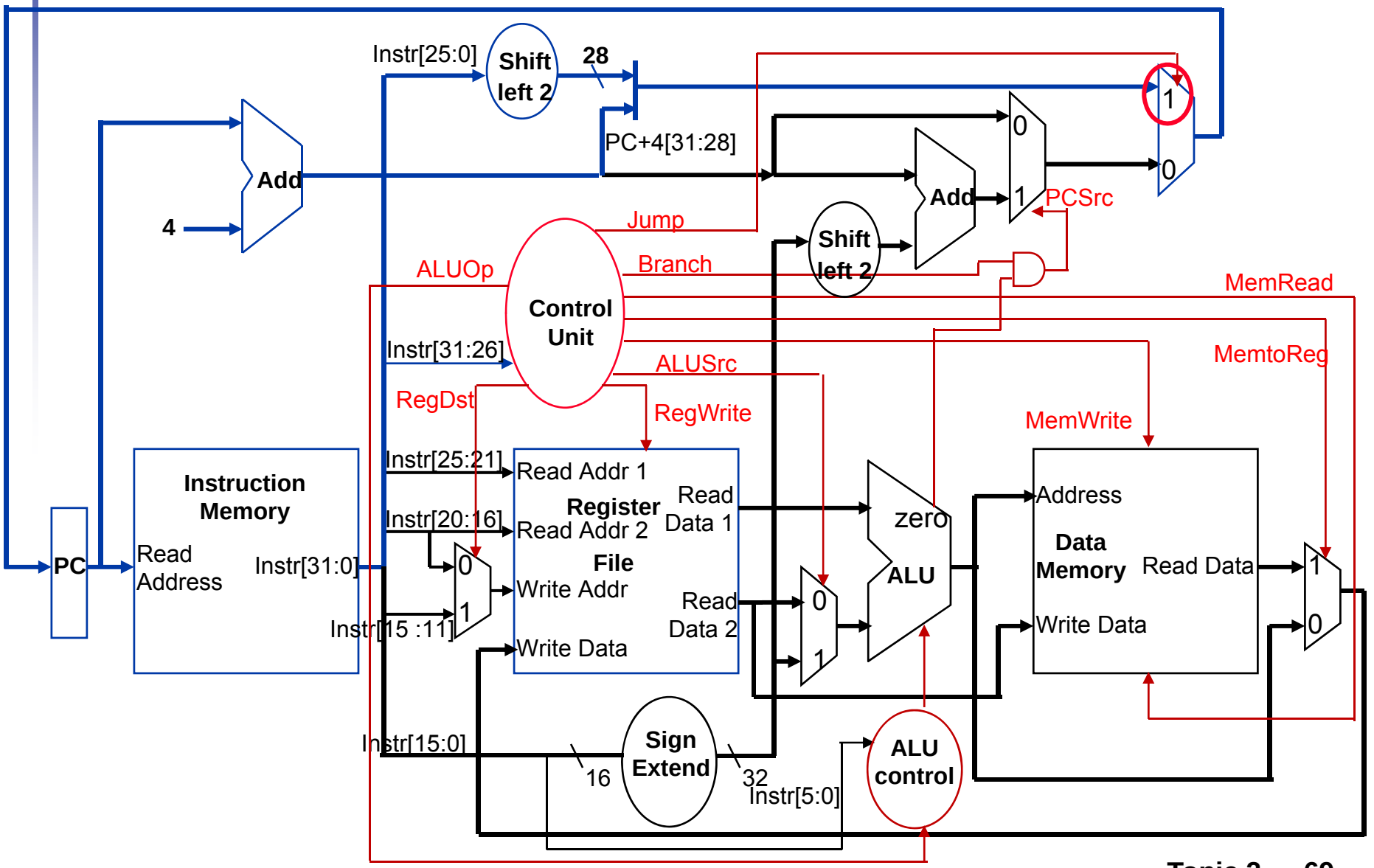
Branch instruction data/control flow



Jump instruction data/control flow



Jump instruction data/control flow



Control signal table

Instr.	ALUOp	RegDst	RegWrite	ALUSrc	MemWrite	MemRead	MemtoReg	PCSrc	Jump
R-type									
load									
store									
beq									
jump									

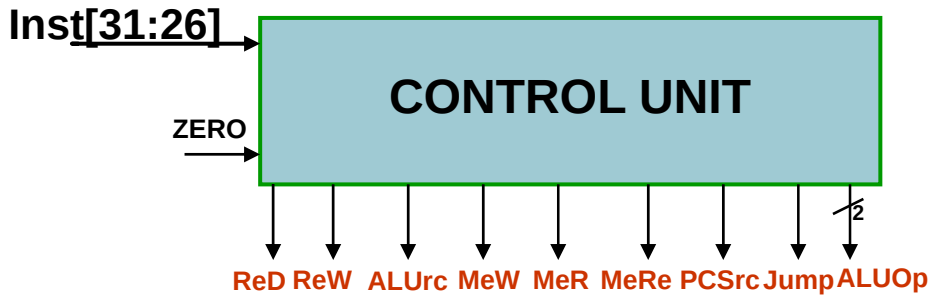
Decoding

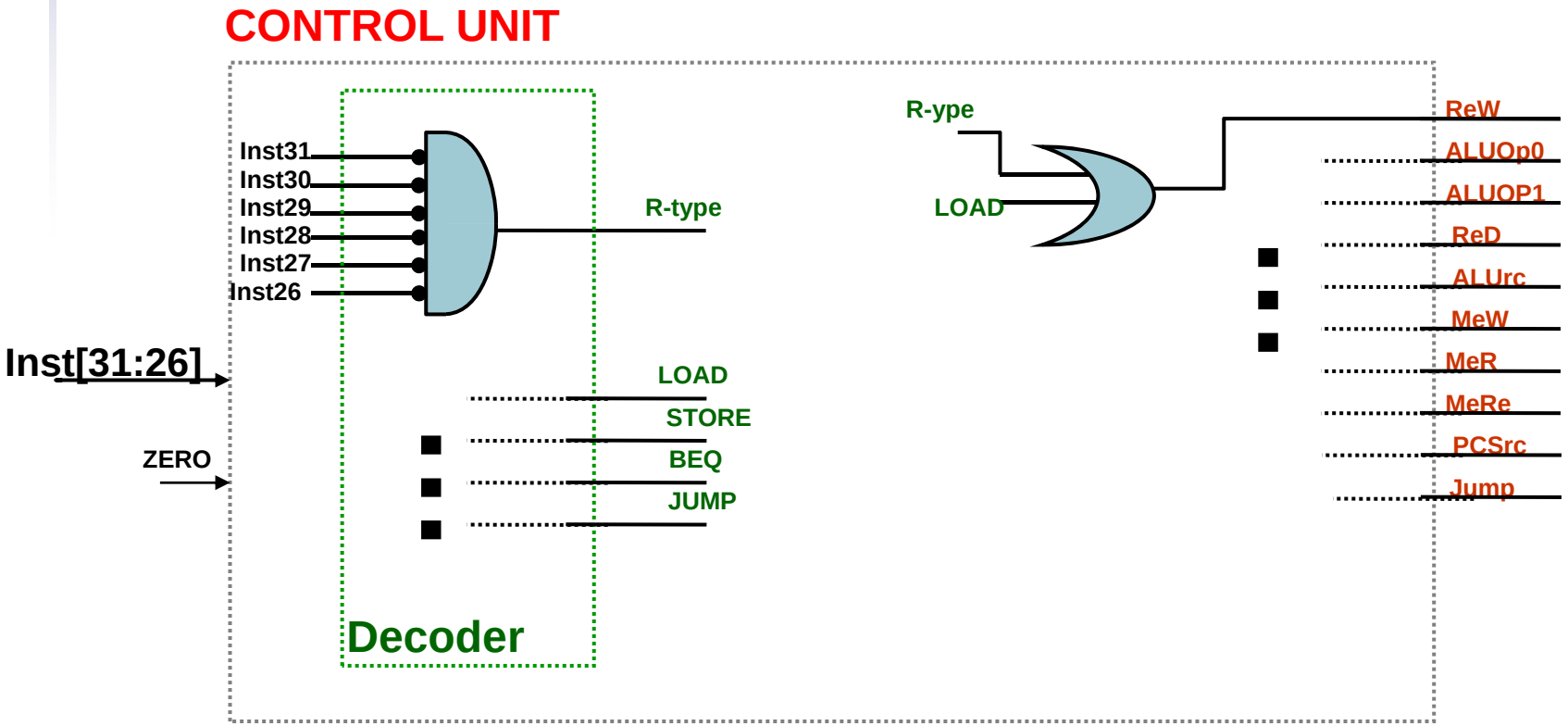
- Instruction names are signals generated by CU from the opcode Inst[31:26] Bits

Instruction	Opcode	Decoding
R-type	000000	R-type = ¬IB31 ∧ ¬IB30 ∧ ¬IB29 ∧ ¬IB28 ∧ ¬IB27 ∧ ¬IB26
load	100011	load (LW) = IB31 ∧ ¬IB30 ∧ ¬IB29 ∧ ¬IB28 ∧ IB27 ∧ IB26
store	101011	store (SW) = IB31 ∧ ¬IB30 ∧ IB29 ∧ ¬IB28 ∧ IB27 ∧ IB26
beq	000100	BEQ = ¬IB31 ∧ ¬IB30 ∧ ¬IB29 ∧ IB28 ∧ ¬IB27 ∧ ¬IB26
jump	000010	JUMP = ¬IB31 ∧ ¬IB30 ∧ ¬IB29 ∧ ¬IB28 ∧ IB27 ∧ ¬IB26

Control Unit

Combinational circuit





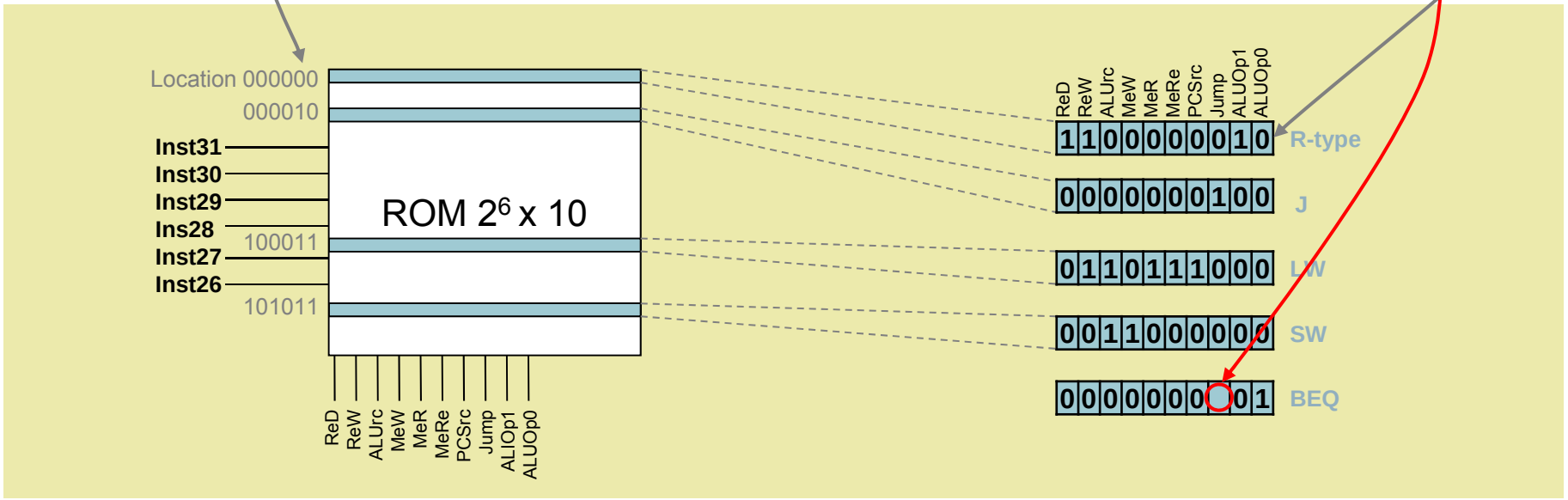
Control unit: ROM memory

- ROM used to store control signals

Inst[31:26]		ReD	ReW	ALUrc	MeW	MeR	MeRe	PCSrc	Jump	ALUOp
000000	R-type	1	1	0	0	0	0	x	0	10
100011	LW	0	1	1	0	1	1	1	0	00
101011	SW	x	0	1	1	0	x	0	0	00
000100	BEQ	x	0	0	0	0	x	x	branch_z	01
000010	Jump	x	0	x	0	0	x	x	1	00

- Input = 6 bits (opcode)
- Output = 10 bits (control signals)

- ROM 2^6 x 10



Clock cycle

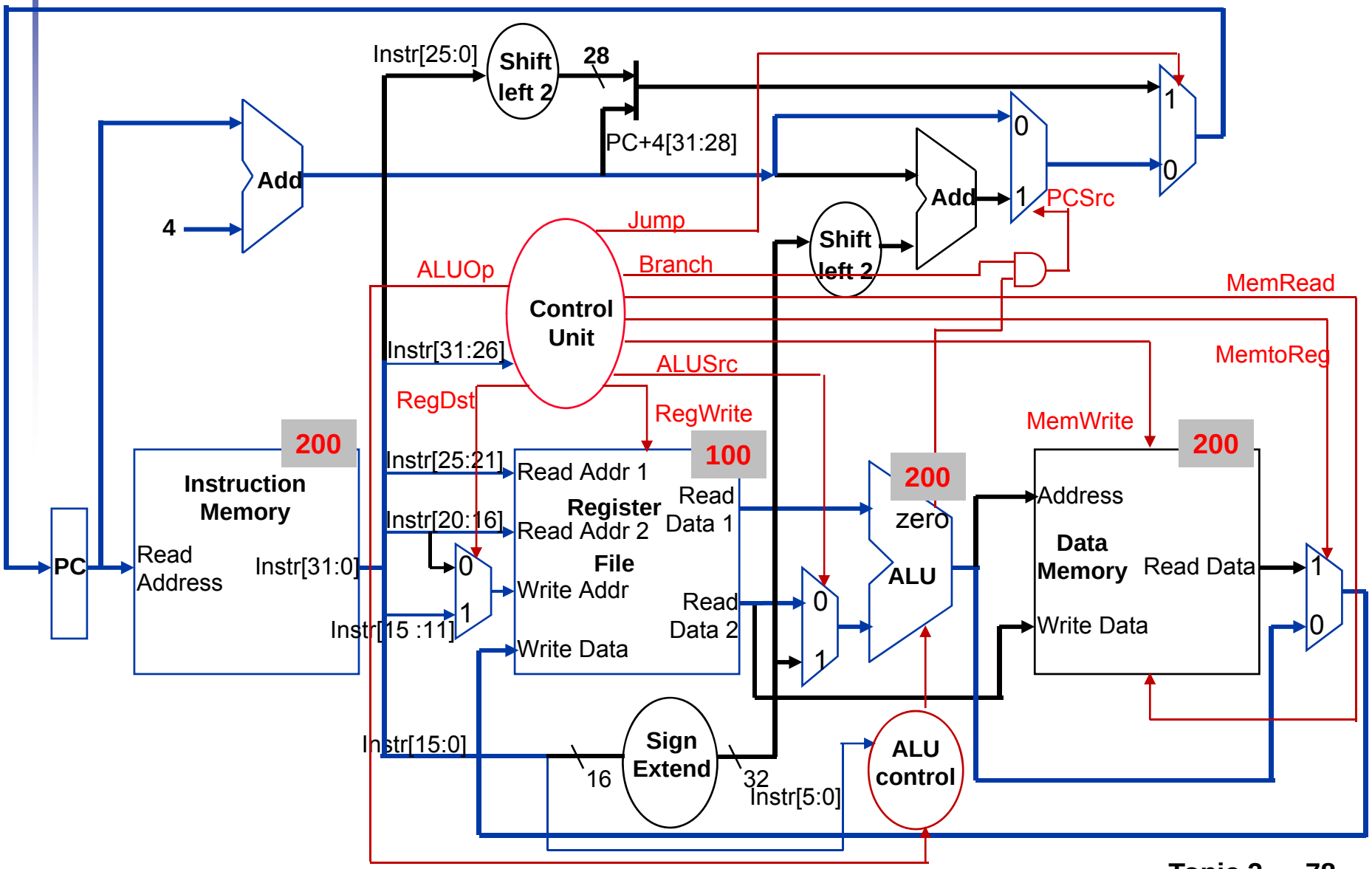
SINGLE-CYCLE PROCESSOR IMPLEMENTATION

Instruction Time (Critical Path)

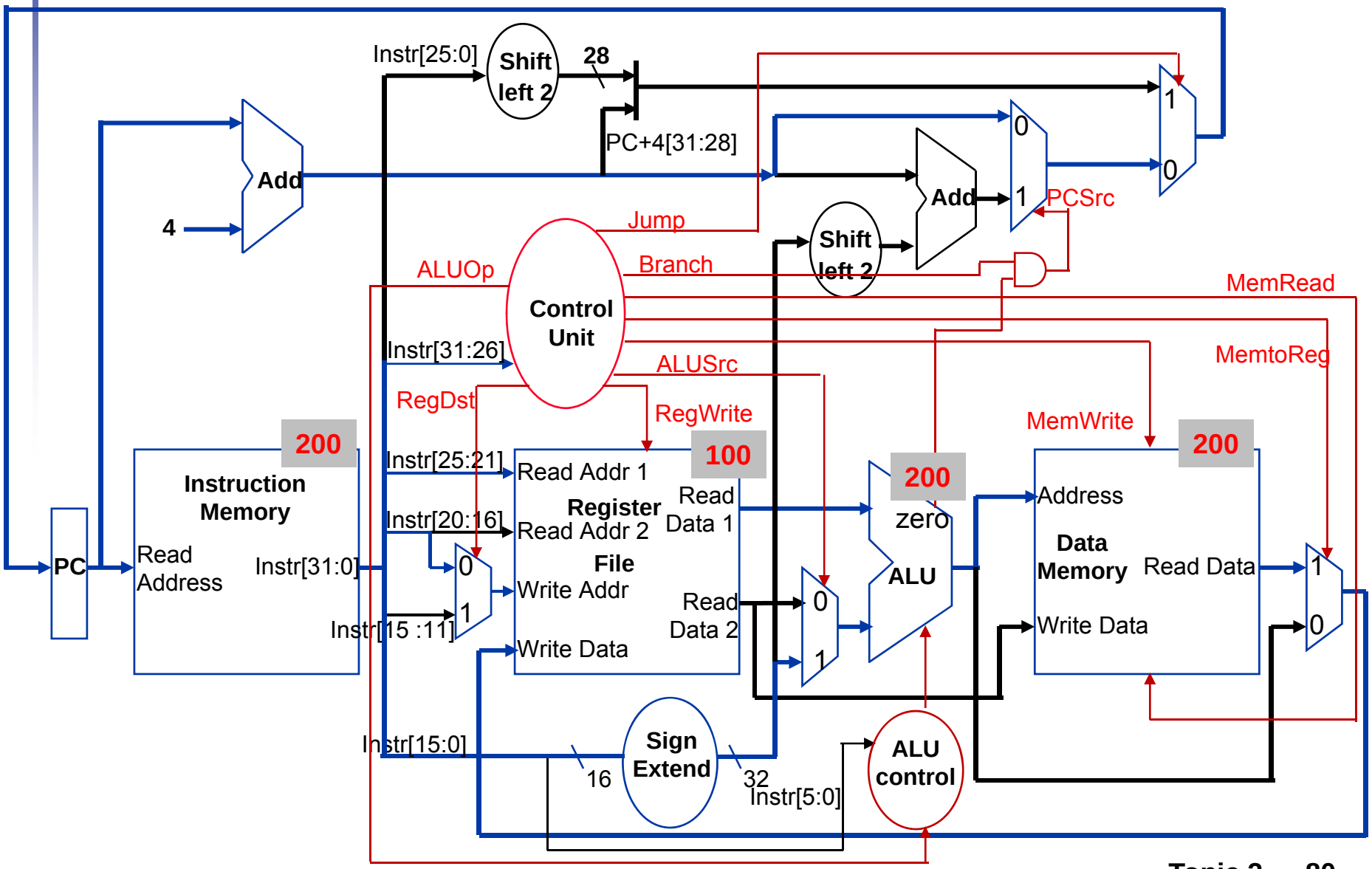
- Determine the clock cycle time assuming negligible delays for: muxes, control unit, sign extend, PC access, shift left 2, wires, setup and hold times. However
  - Instruction and Data Memory requires 200 ps
  - ALU takes 200 ps
  - Register File access (read or write) requires 100 ps

Instr.	I Mem	Reg Rd	ALU Op	D Mem	Reg Wr	Total
R-type						
load						
store						
beq						
jump						

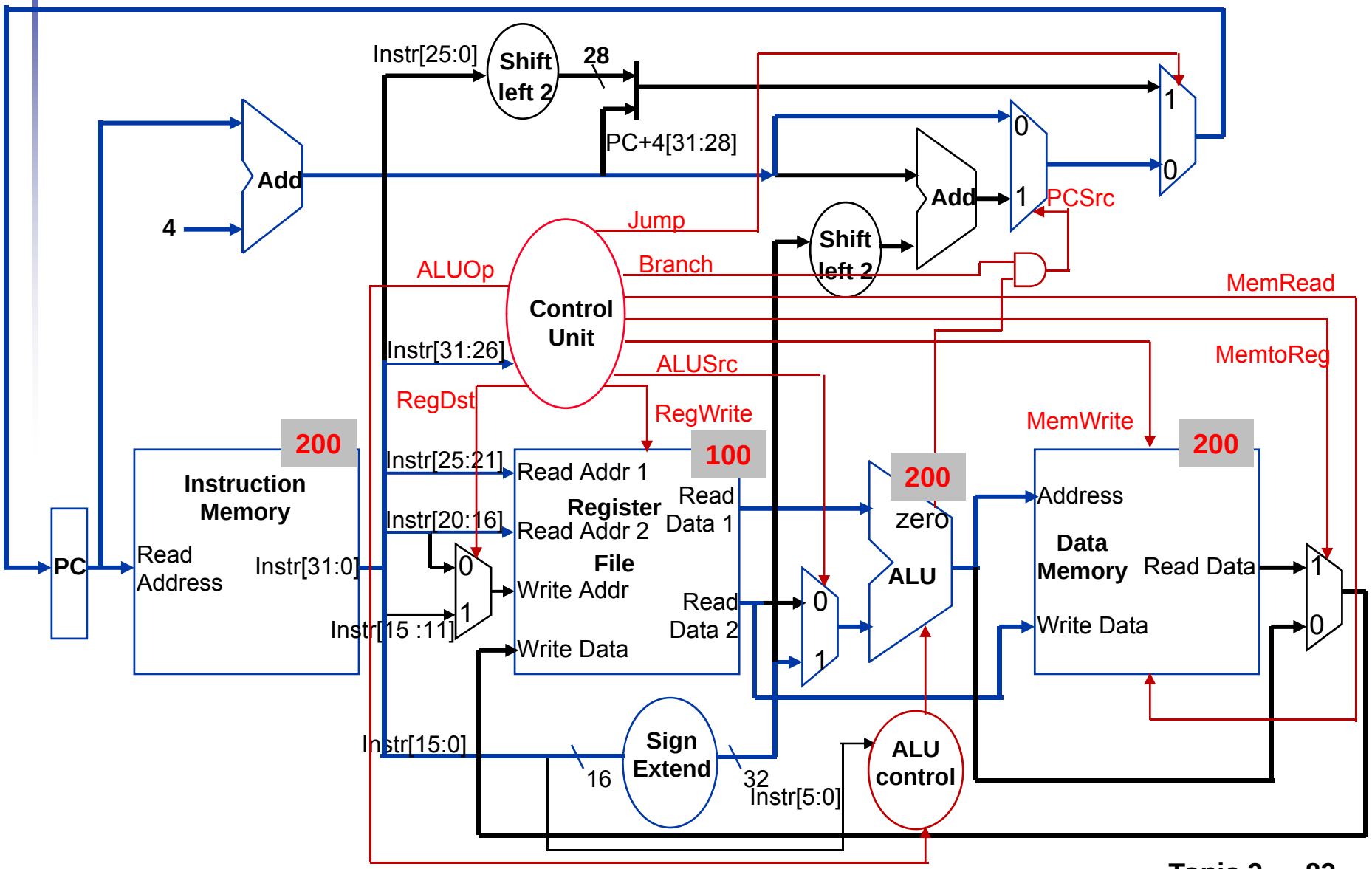
Critical path for R-type Instruction



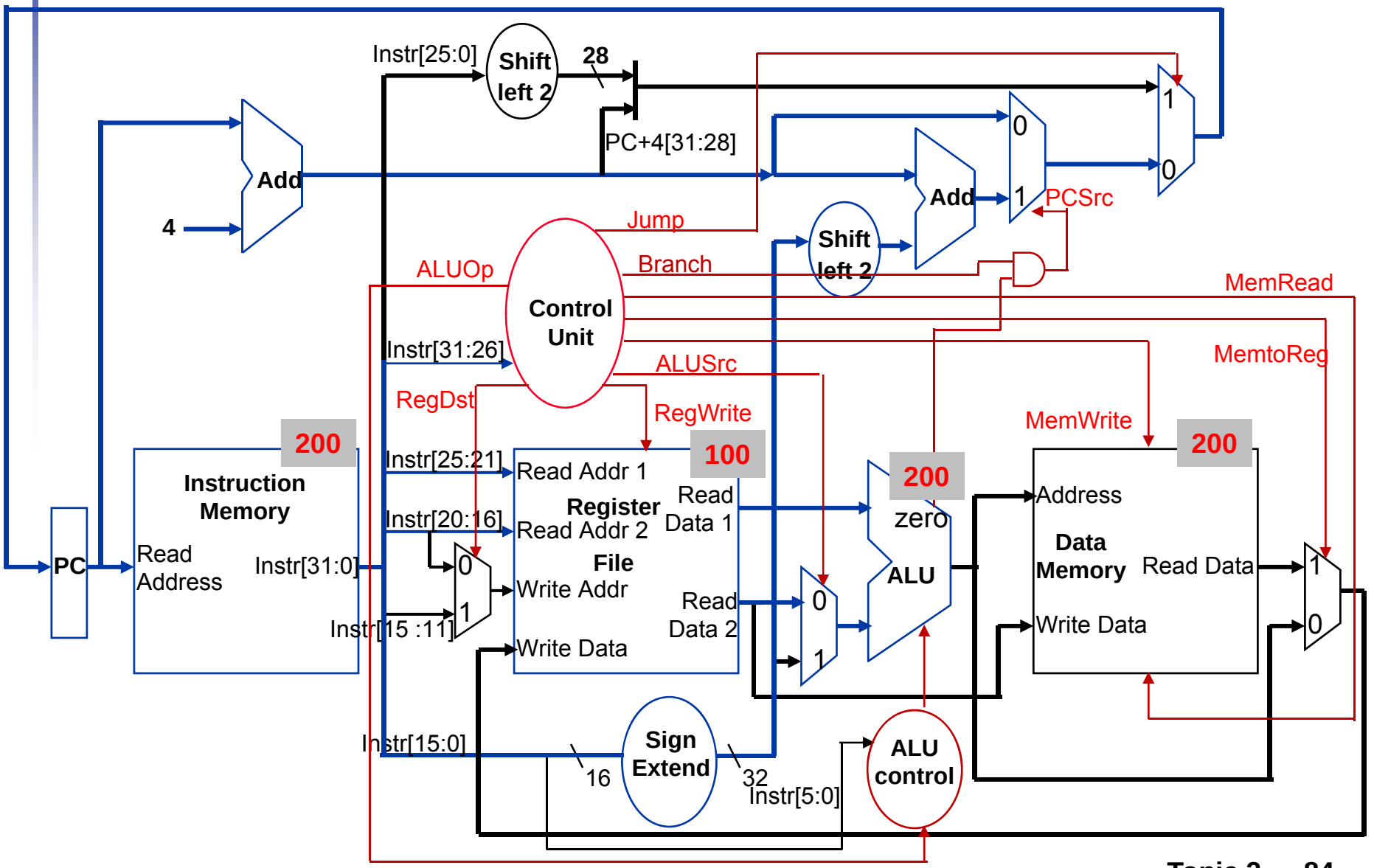
Critical path for Load Word Instruction



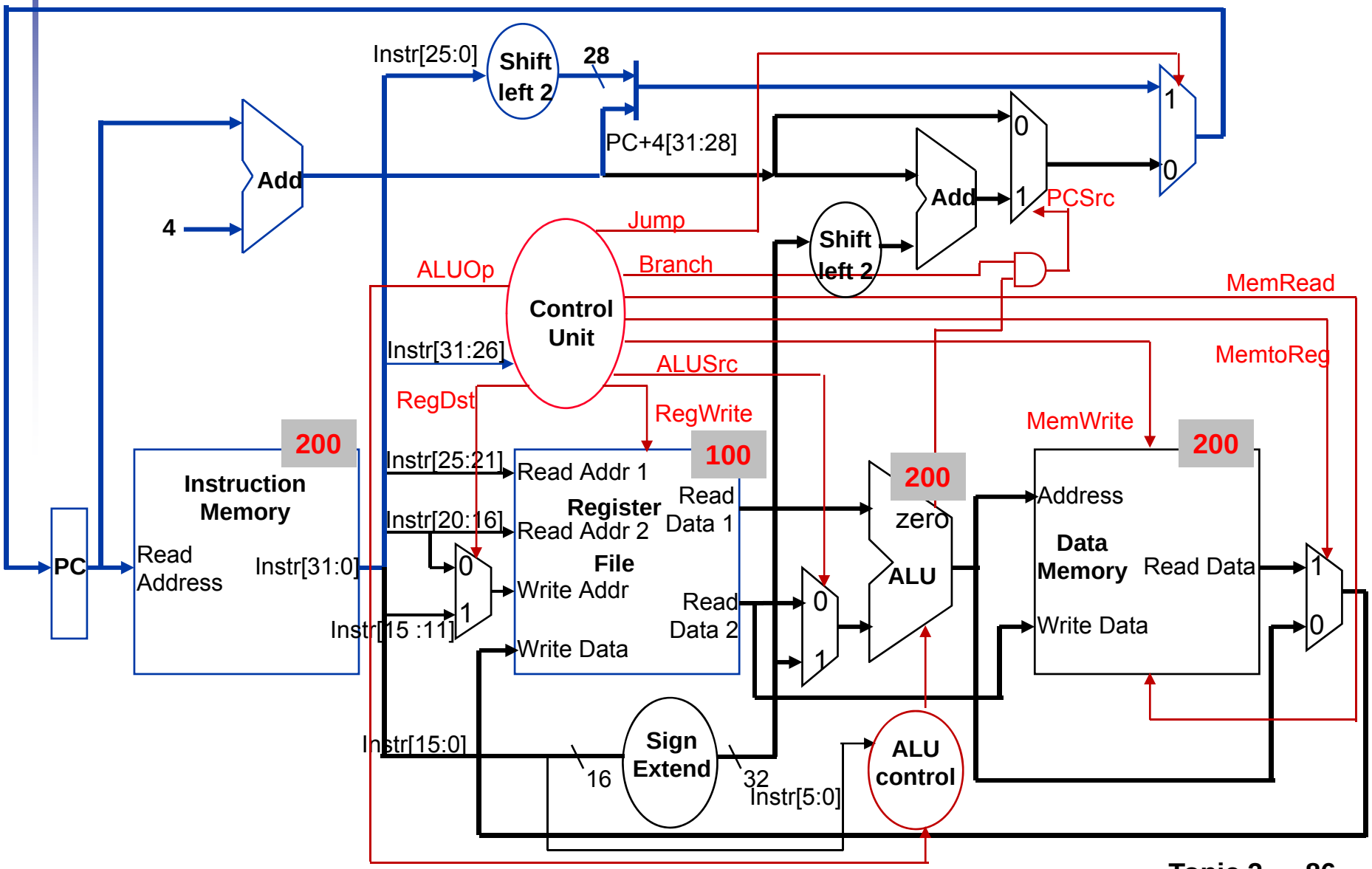
Critical path for Store Word Instruction



Critical path for Branch Instruction



Critical path for Jump Instruction



Instruction Times (Critical Paths)

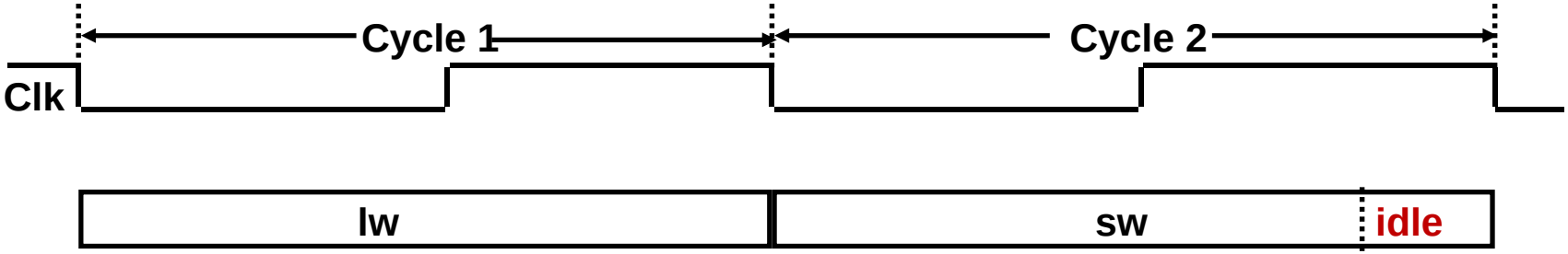
- Determine the clock cycle time assuming negligible delays for: muxes, control unit, sign extend, PC access, shift left 2, wires, setup and hold times
  - Instruction and Data Memory (200 ps)
  - ALU (200 ps)
  - Register File access (read or write) (100 ps)

Unrealistic assumption.

Instr.	I Mem	Reg Rd	ALU Op	D Mem	Reg Wr	Total
R-type	200	100	200		100	600
load	200	100	200	200	100	800
store	200	100	200	200		700
beq	200	100	200			500
jump	200					200

Single Cycle Disadvantages & Advantages

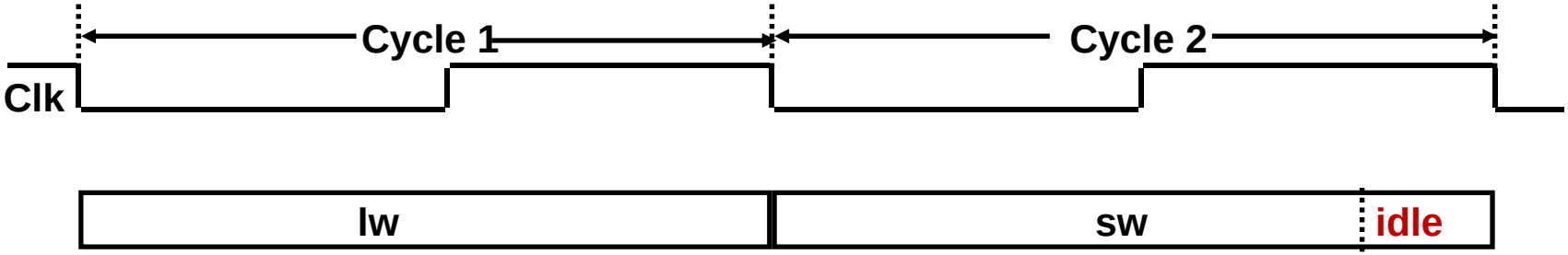
- Uses the clock cycle inefficiently – the clock cycle is determined by the slowest instruction



- Inefficient use of area – some functional units like adders must be duplicated since they can not be shared during a clock cycle

Single Cycle Disadvantages & Advantages

- ☒ Violates design principle: *"Make the common case fast"*



- ☑ Is simple and easy to understand

Bibliography

- D.A. PATTERSON and J.L. HENNESSY (2009). Computer organization and design: The hardware / software interface. 4th edition. Morgan Kaufmann
  - Chapter 4.
- G. BANDERA, F.J. CORBERA et al. (2000). Tecnología de computadores. 1st edition. Málaga. Servicio de publicaciones de la UMA.
  - Chapter 1 and 2.