CSE 3421
Computer Architecture

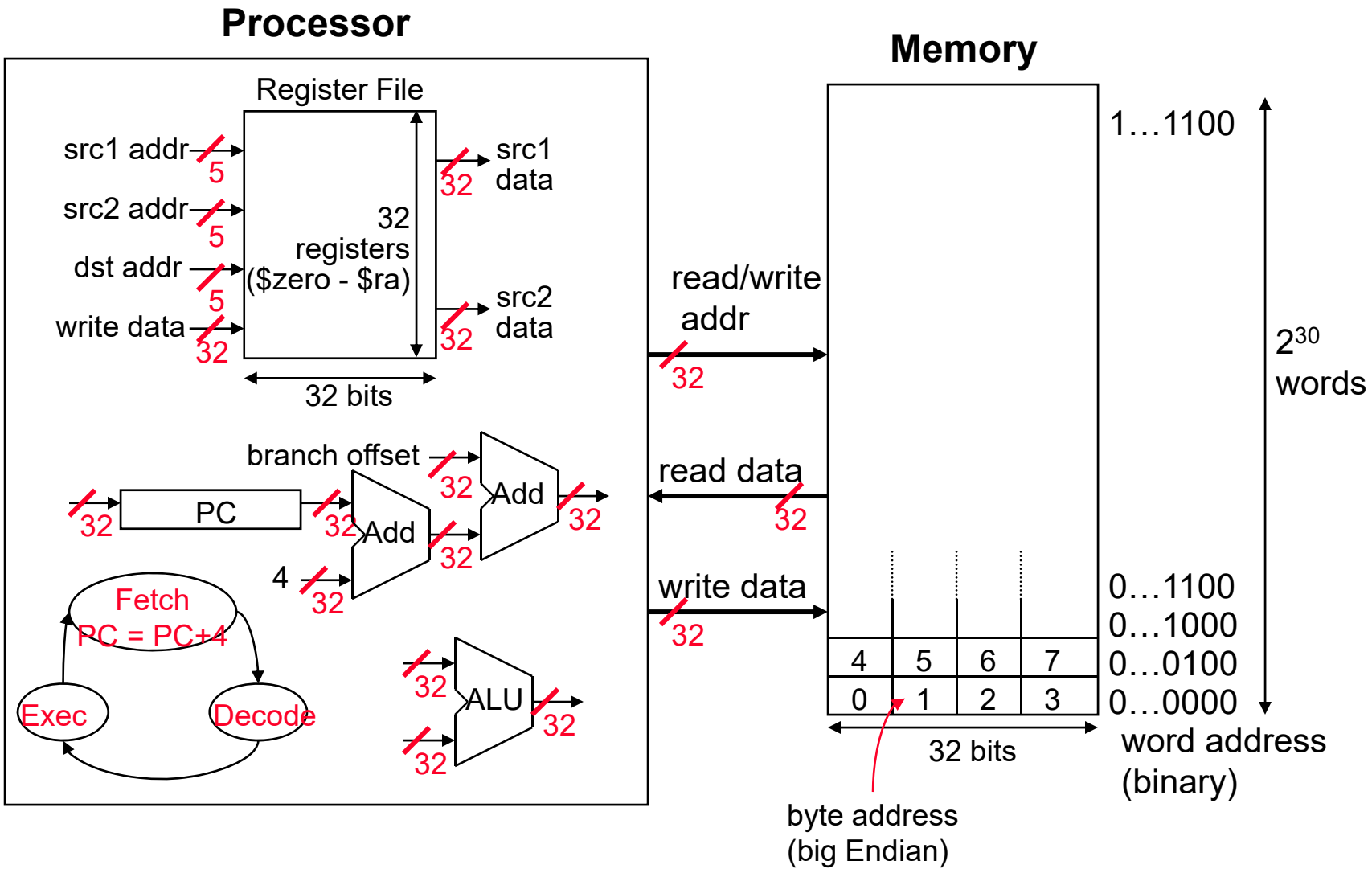
Chapter 4: The Processor
- Single Cycle Datapath -

Xiaodong Zhang

Review: MIPS (RISC) Design Principles

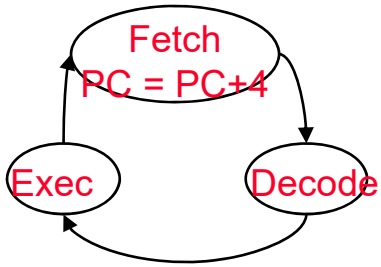
- Simplicity favors regularity
  - fixed size instructions
  - small number of instruction formats
  - opcode always the first 6 bits
- Smaller is faster
  - limited instruction set
  - limited number of registers in register file
  - limited number of addressing modes
- Make the common case fast
  -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

MIPS Organization So Far



The Processor: Datapath & Control

- Our implementation of the MIPS is simplified
  - memory-reference instructions: `lw, sw`
  - arithmetic-logical instructions: `add, sub, and, or, slt`
  - control flow instructions: `beq, j`
- Generic implementation (three steps in common)
  1. use the program counter (PC) to supply the instruction address and fetch the instruction from memory (and update the PC)
  2. decode the instruction (and read registers)
  3. execute the instruction

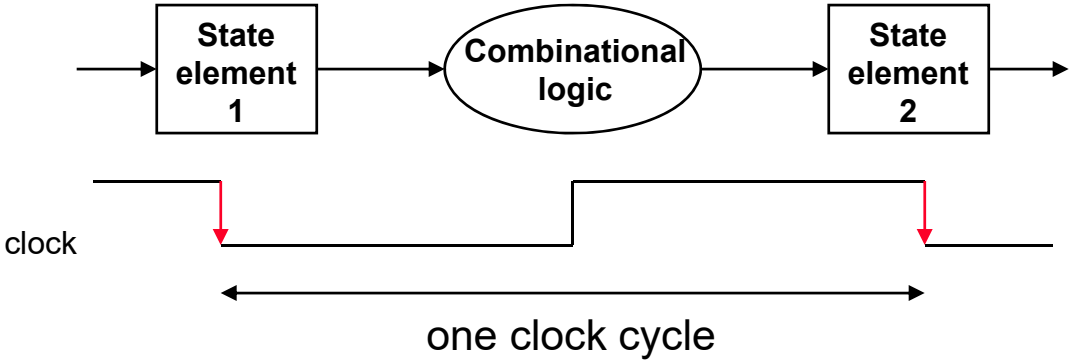


- All instructions (except `j`) use the ALU after reading the registers ("j add" is formed by shifting and concatenation)

How? memory-reference? arithmetic? control flow?

Aside: Clocking Methodologies

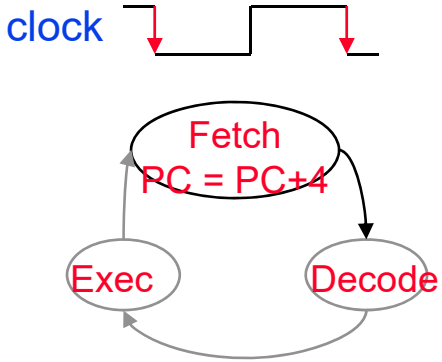
- The clocking methodology defines when data in a state element is valid and stable relative to the clock
  - State elements - a memory element such as a register
  - Edge-triggered – all state changes occur on a clock edge
- Typical execution
  - read contents of state elements -> send values through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

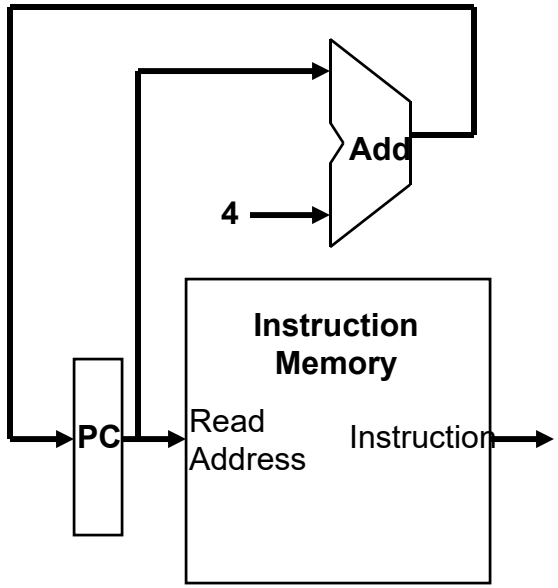


- State elements, such as registers can be written on every clock cycle
  - write occurs only when both the write control is asserted and the clock edge occurs

Fetching Instructions

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

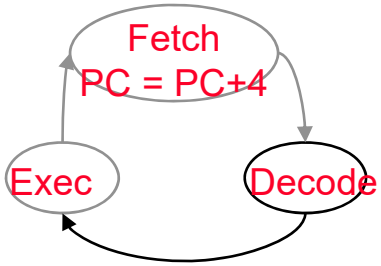




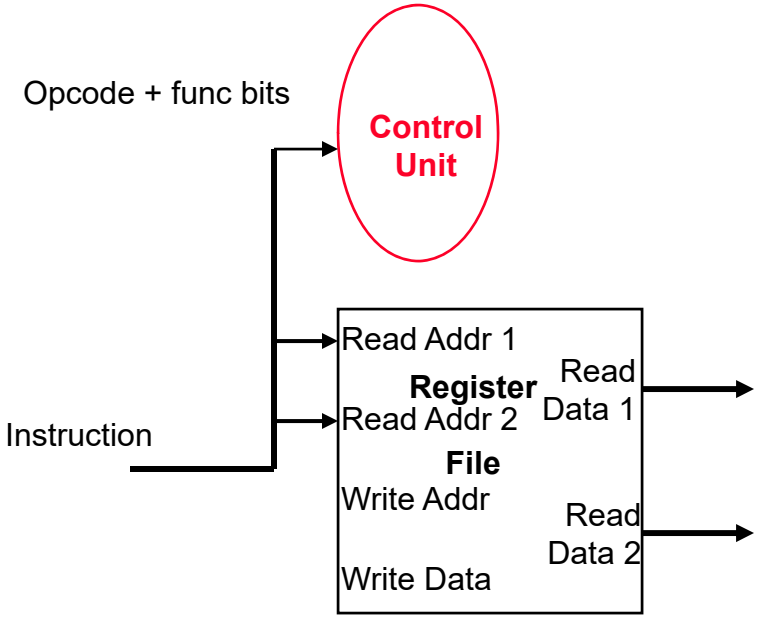
- PC is updated every clock cycle, so it does not need an explicit write control signal just a clock signal
- Reading from the Instruction Memory is a combinational activity, so it doesn't need an explicit read control signal

Decoding Instructions

- Decoding instructions involves
  - sending the fetched instruction's opcode and function field bits to the control unit
  - Signals from the control unit like the gas/break pedals



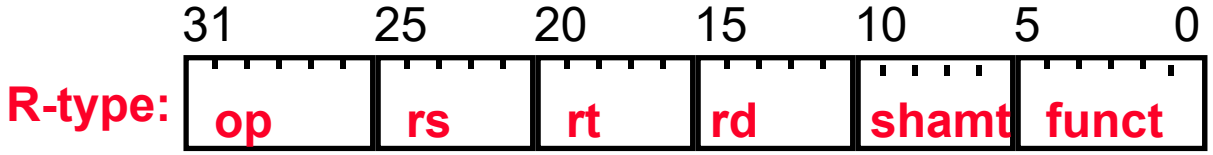
and



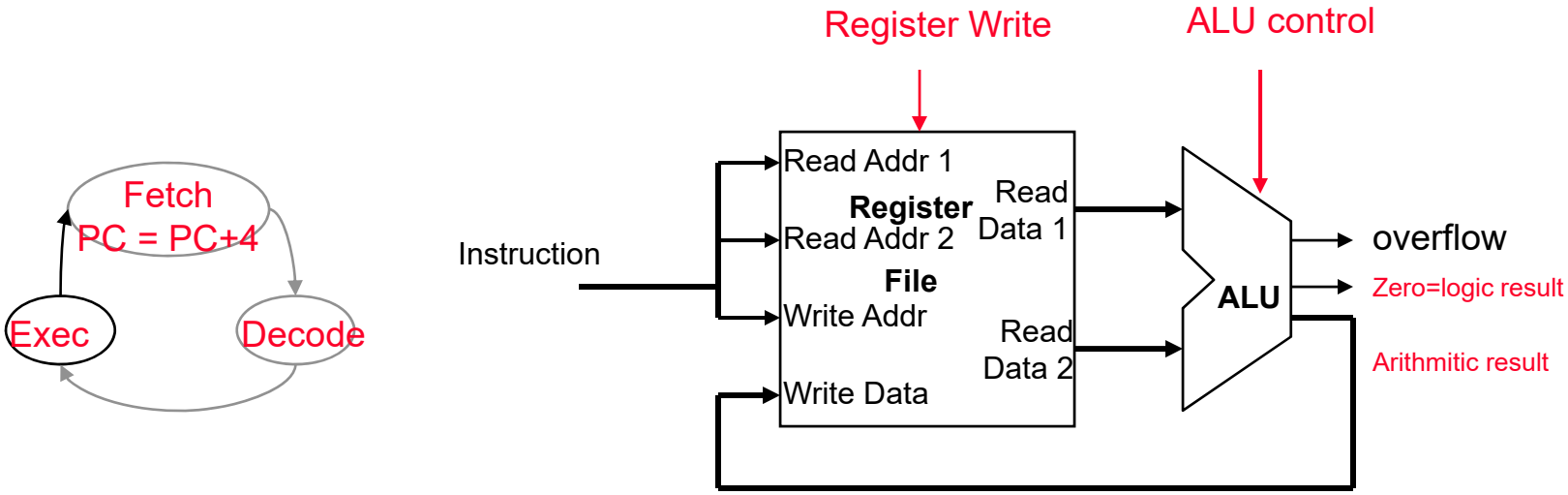
  - reading two values from the Register File
    - Register File addresses are contained in the instruction

Executing R Format Operations

- R format operations (`add, sub, slt, and, or`)



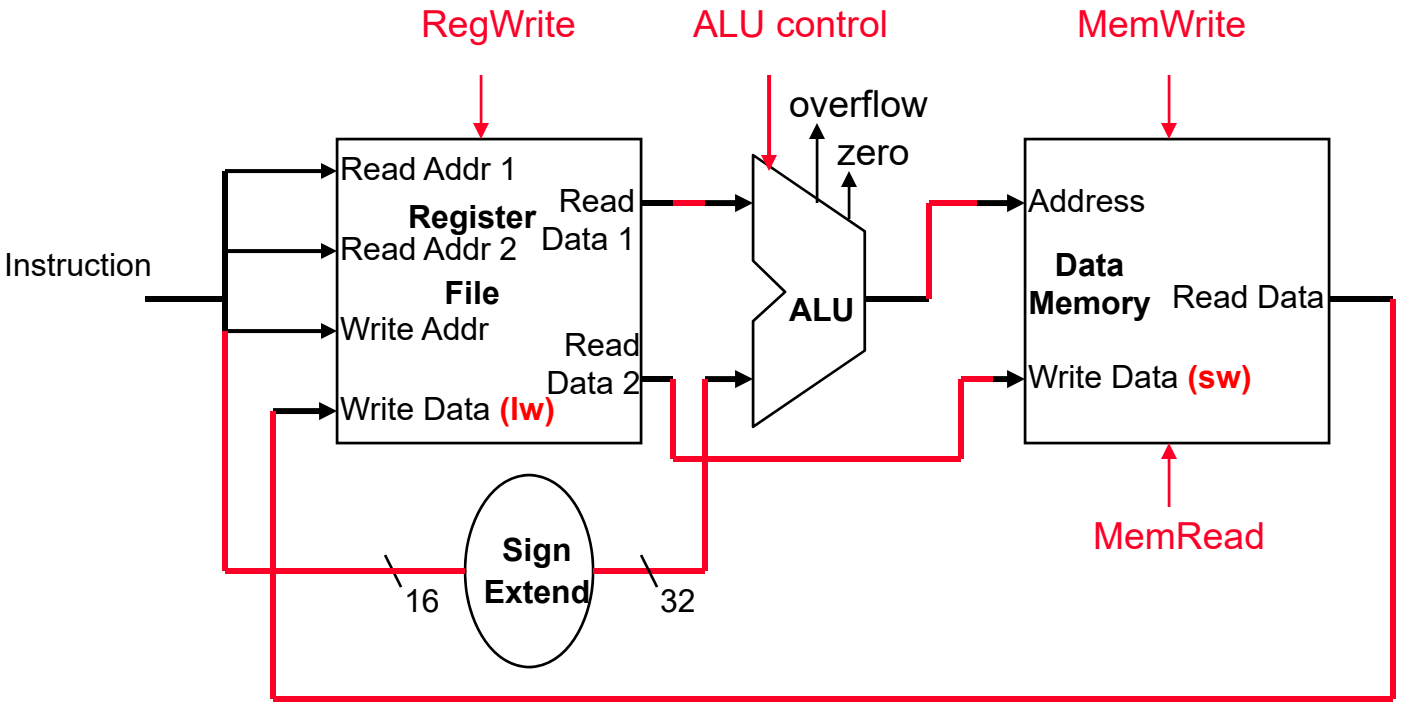
- perform operation (op and funct) on values in rs and rt
- store the result back into the Register File (into location rd)



- Note that Register File is not written every cycle (e.g. `lw`), so we need Register Write control signal for the Register File

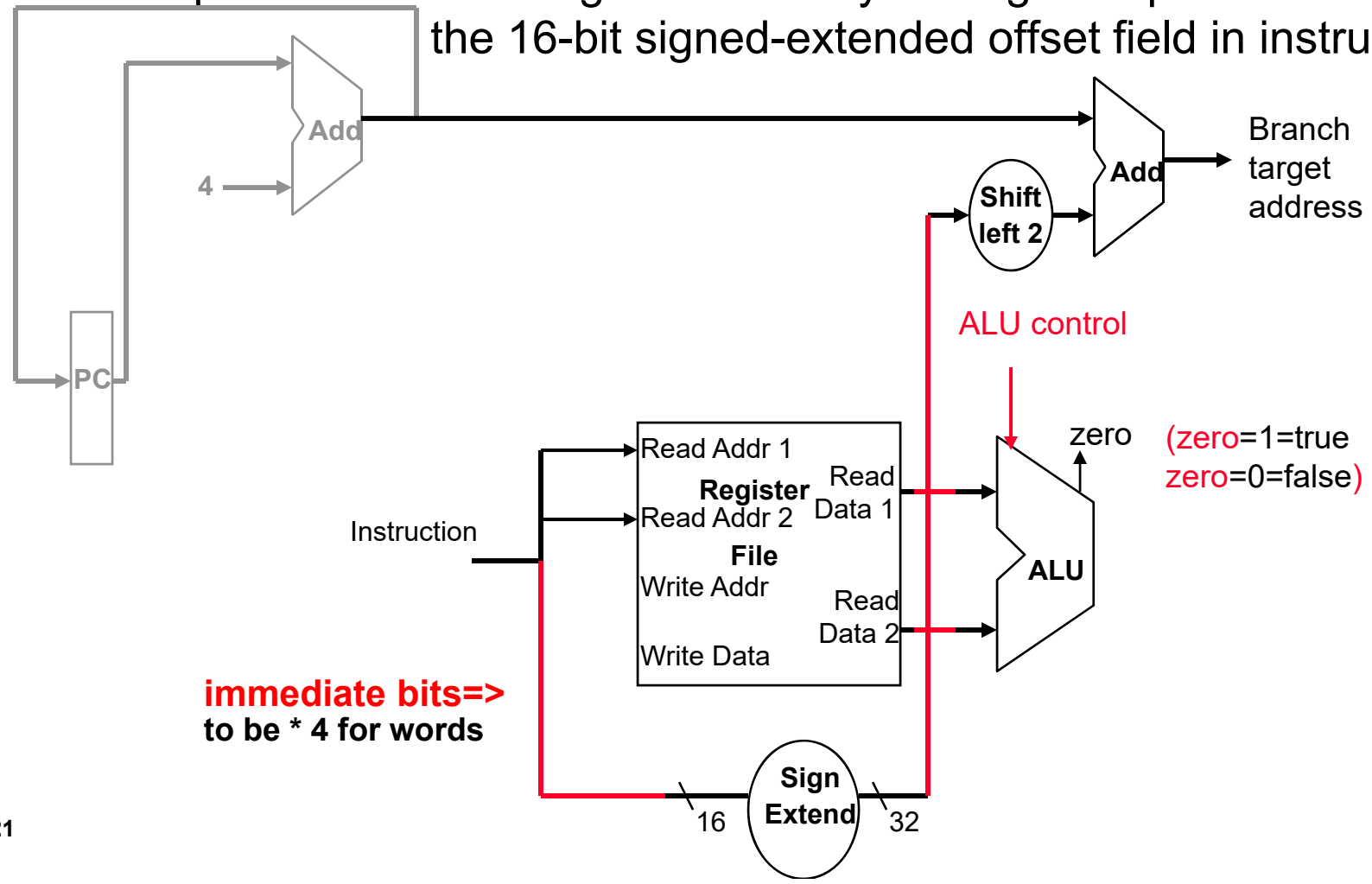
Executing Load and Store Operations

- Load and store operations involves
  - compute memory address by adding the base register (read from the Register File during decode) to the 16-bit signed-extended offset field in the instruction
  - store value from register $s1 to Memory (**sw** $s1, 20($s2))
  - load value from Memory to the Register $s1 (**lw** $s1, 20($s2))



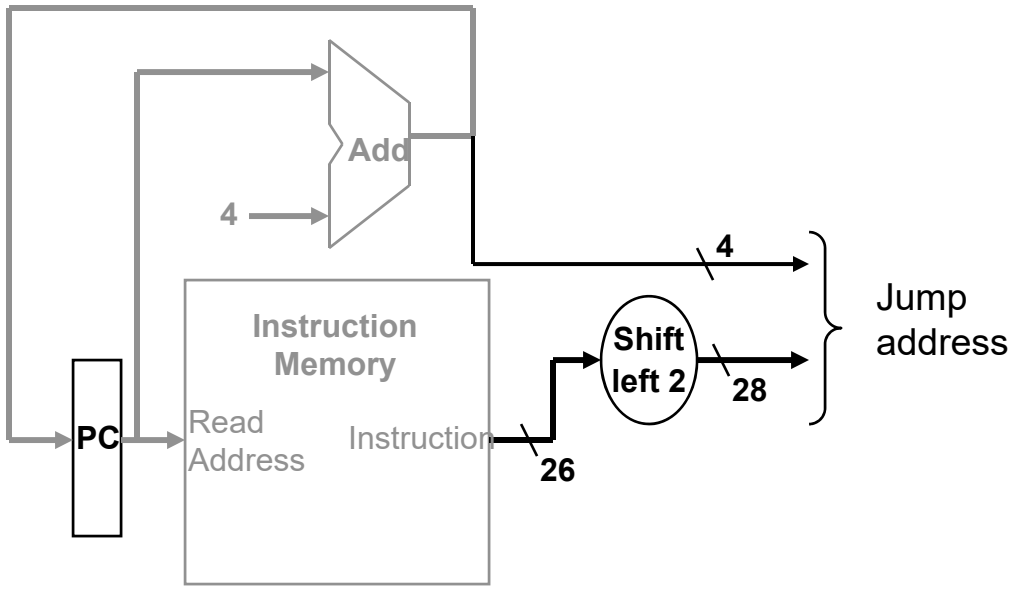
Executing Branch Operations

- Branch operations involves
  - compare the operands read from the Register File during decode for equality (`zero` ALU output, e.g. beq $t1, $zero, immediate)
  - compute the branch target address by adding the updated PC to the 16-bit signed-extended offset field in instruction



Executing Jump Operations

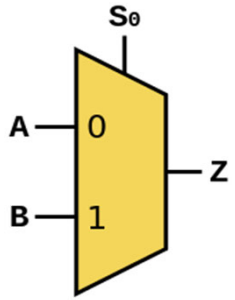
- Jump operation involves
  - replace the lower 28 bits of the PC with the lower 26 bits of the fetched instruction shifted left by 2 bits (times 4 for word address)
  - The most significant 4-bits of PC is kept for the program block



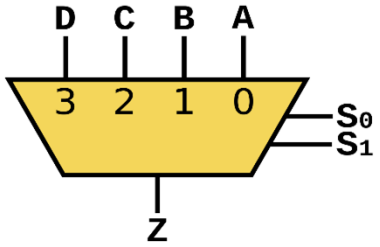
Multiplexor is often used in the data path

- A multiplexor is a device that takes multiple signals and outputs one signal by a specific selection

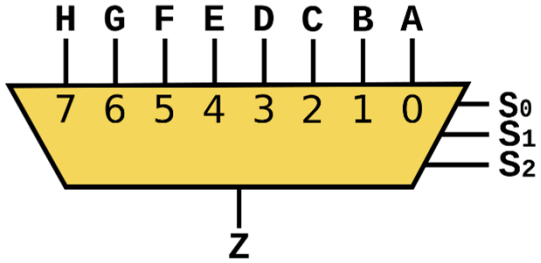
- A 2-to-1 multiplexor



- A 4-1 multiplexor



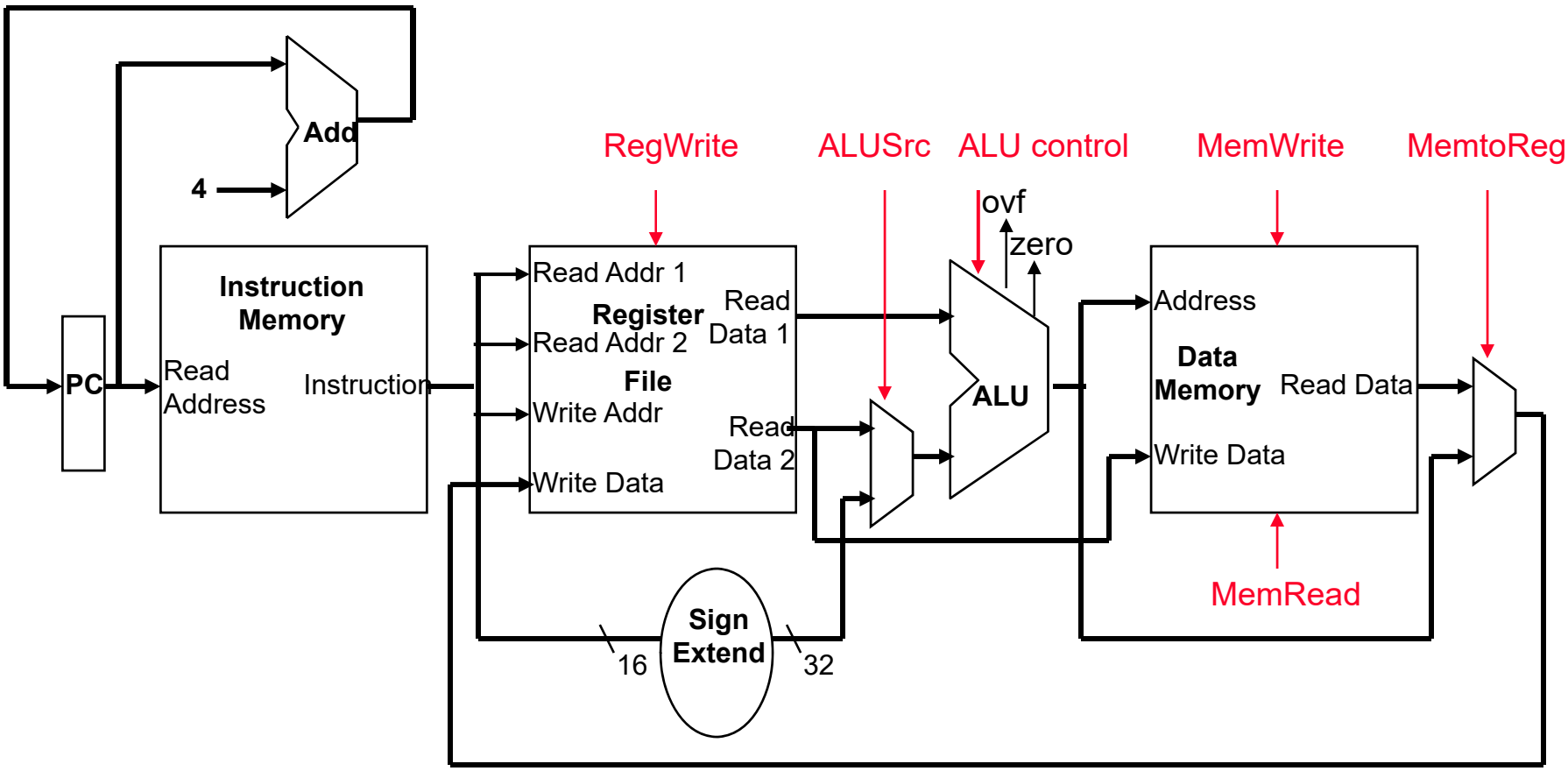
- A 8-to-1 multiplexor



Creating a Single Datapath from the Parts

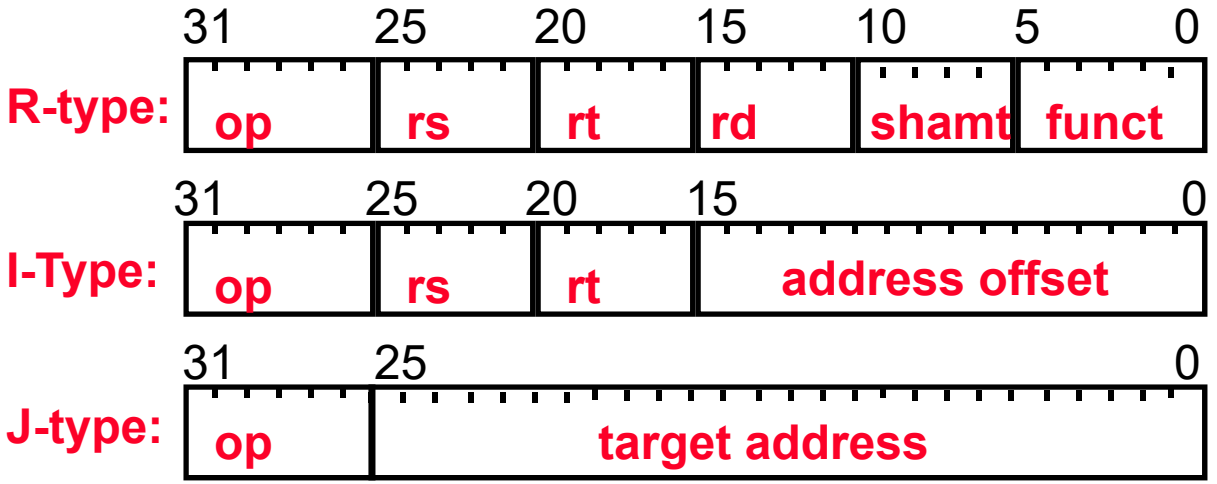
- Assemble the datapath segments and add control lines and multiplexors as needed
- Single cycle design – fetch, decode and execute each instructions in one clock cycle
  - no datapath resource can be used more than once per instruction, so some must be duplicated (e.g., separate Instruction Memory and 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 the Register File and Data Memory
- Cycle time is determined by length of the longest path

Fetch, R, and Memory Access Portions



Adding the Control

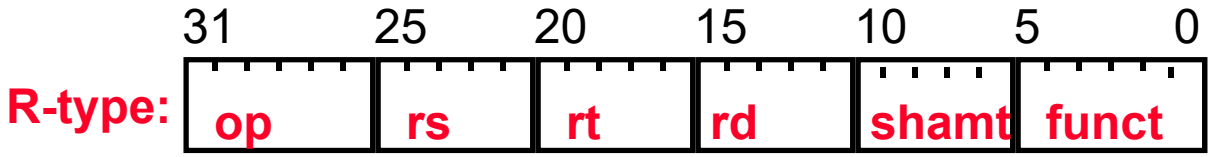
- Selecting the operations to perform (ALU, Register File and Memory read/write)
- Controlling the flow of data (multiplexor inputs)



- **Regulated regions**
  - op field always in bits 31-26
  - addr of registers to be read are always specified by the rs field (bits 25-21) and rt field (bits 20-16); for lw and sw rs is the base register
  - addr. of register to be written is in one of two places – in rt (bits 20-16) for lw; in rd (bits 15-11) for R-type instructions
  - offset for beq, lw, and sw always in bits 15-0

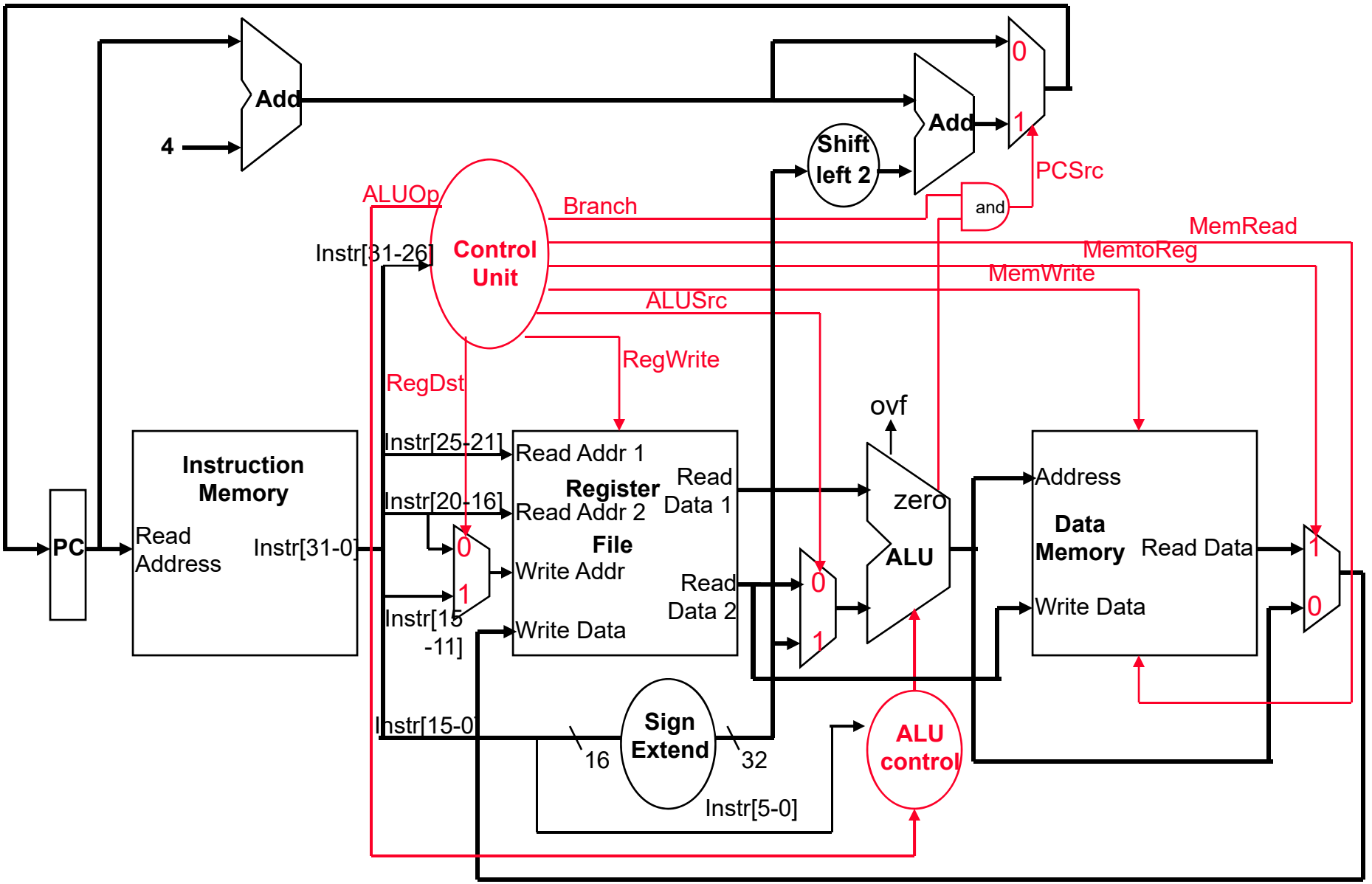
Decoding R-type from the fetched instruction

- For R-type, an operation is done between registers rs (bits 25-21) and rt (bits 20-16) by ALU and the result is written to register rd (bits 15-11)

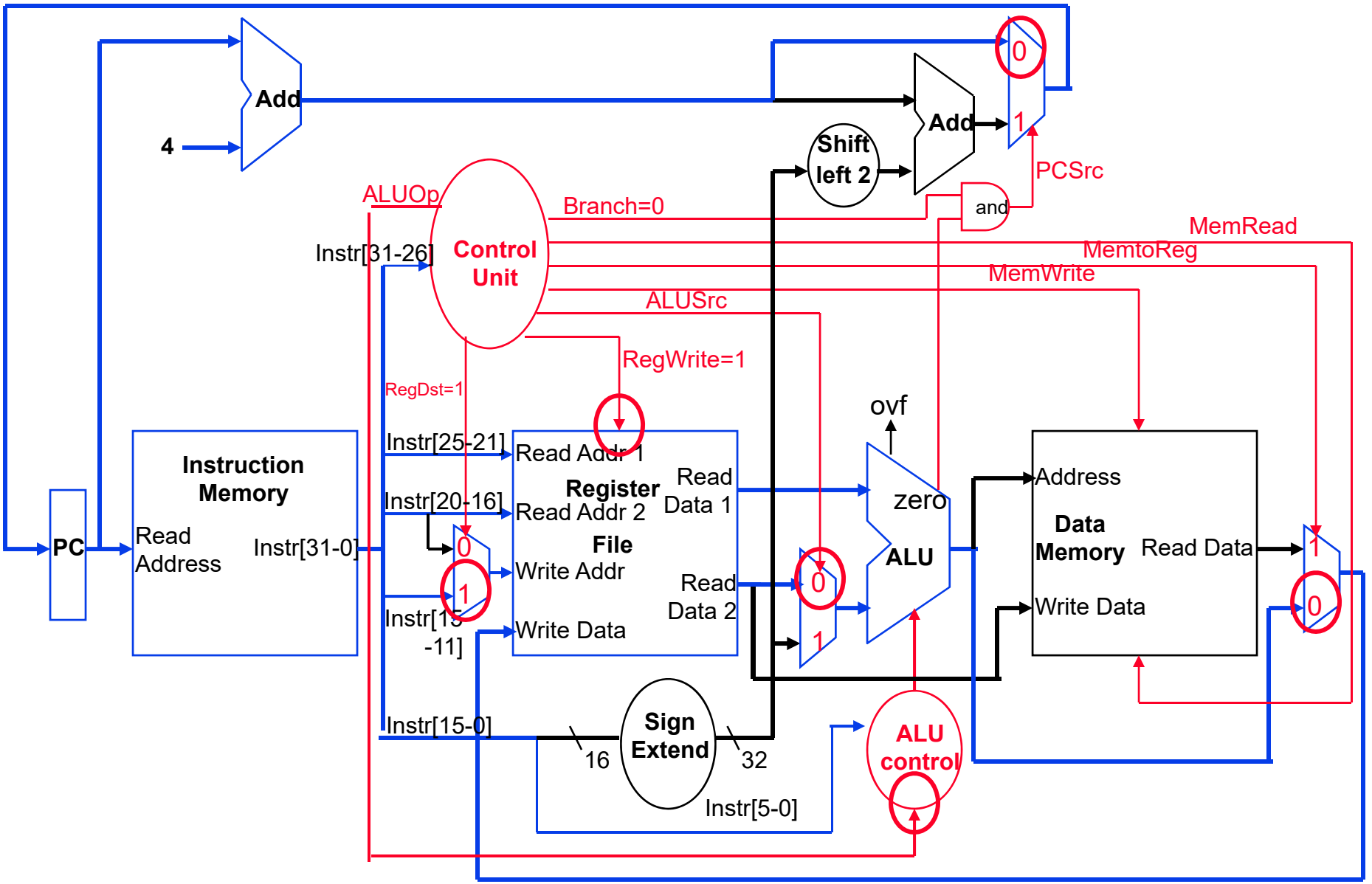


- Control signals
  - RegDST=1
  - Branch=0
  - ALUSrc=0
  - ALUOp = needed operations, e.g. add, sub, …
  - MemtoReg=0
  - RegWrite =1

Single Cycle Datapath with Control Unit



R-type Instruction Data/Control Flow



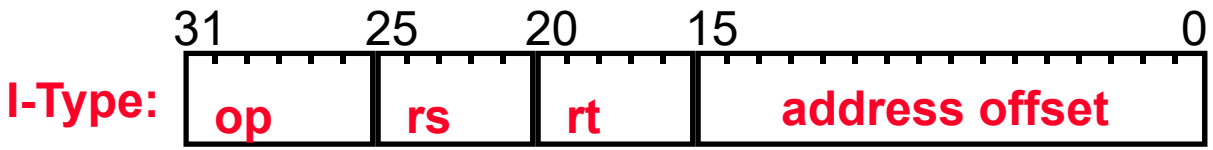
Explanation of R-type Instruction Signals

- PC+4 or go to branch address is controlled by the output of AND gate:
  - Two inputs: branch = 0/1, logic Zero=0 (no logic op) from ALU
  - In this case, branch = 0, multiplexor selection = 0, thus PC+4
- Two ALU inputs:
  - Read Data 1 and the other one is ALUSrc signal (0 for data 2)
- What ALU Operation is selected by signal ALUOp
- MemtoReg = 0, the ALU result is selected to Register file
- The destination register is selected by RegDst (1, bits 15-11)
- RegWrite triggers the write to register
- **Only the blue line circuits are affected**

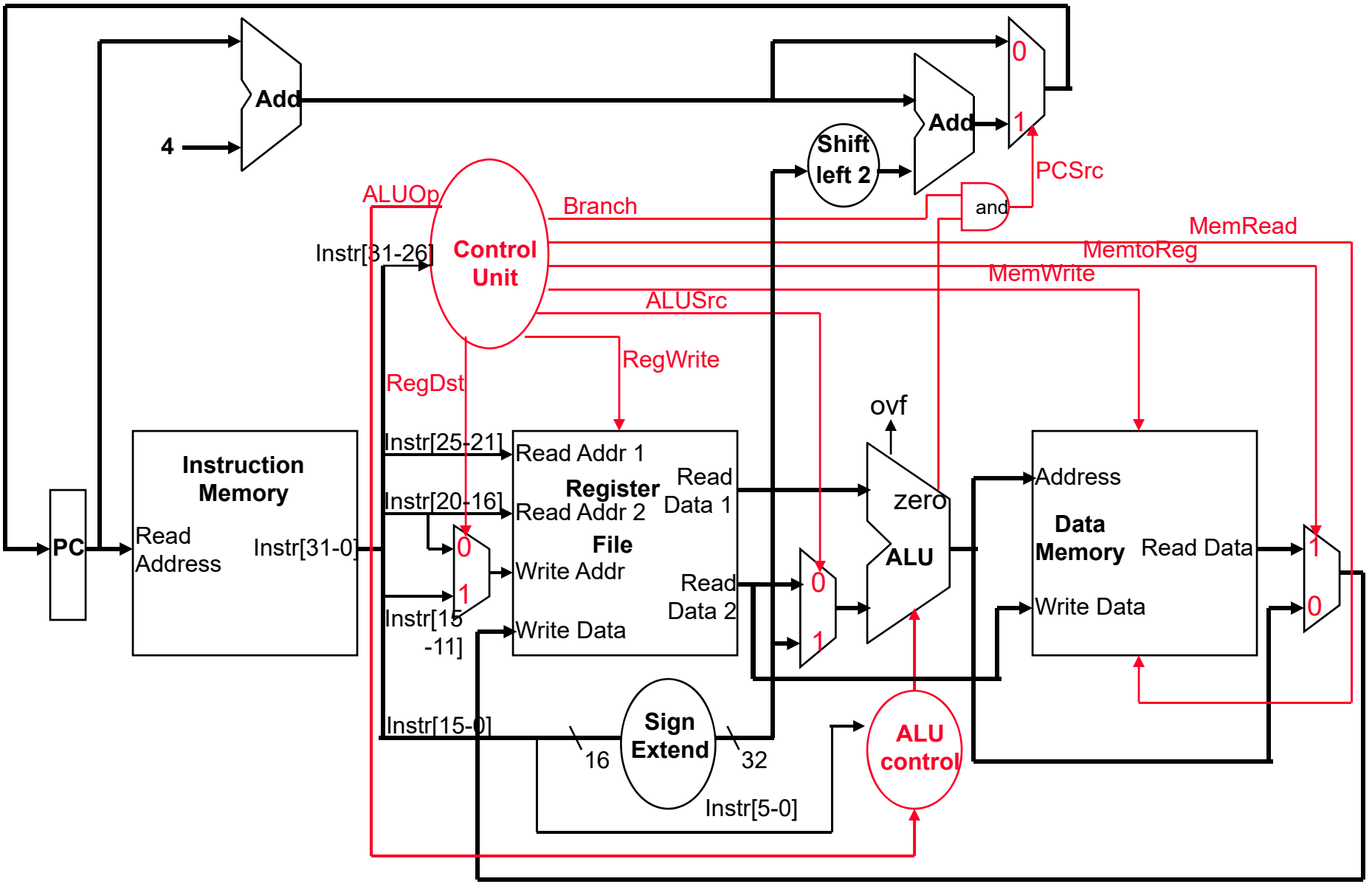
Decoding fetched instruction “lw”

- This is an I-type. Base memory address is in register rs (bits 25-21) and offset comes from bits 15-0. An addition between the two gives the memory address. The memory word is loaded to register rt (bits 20-16)

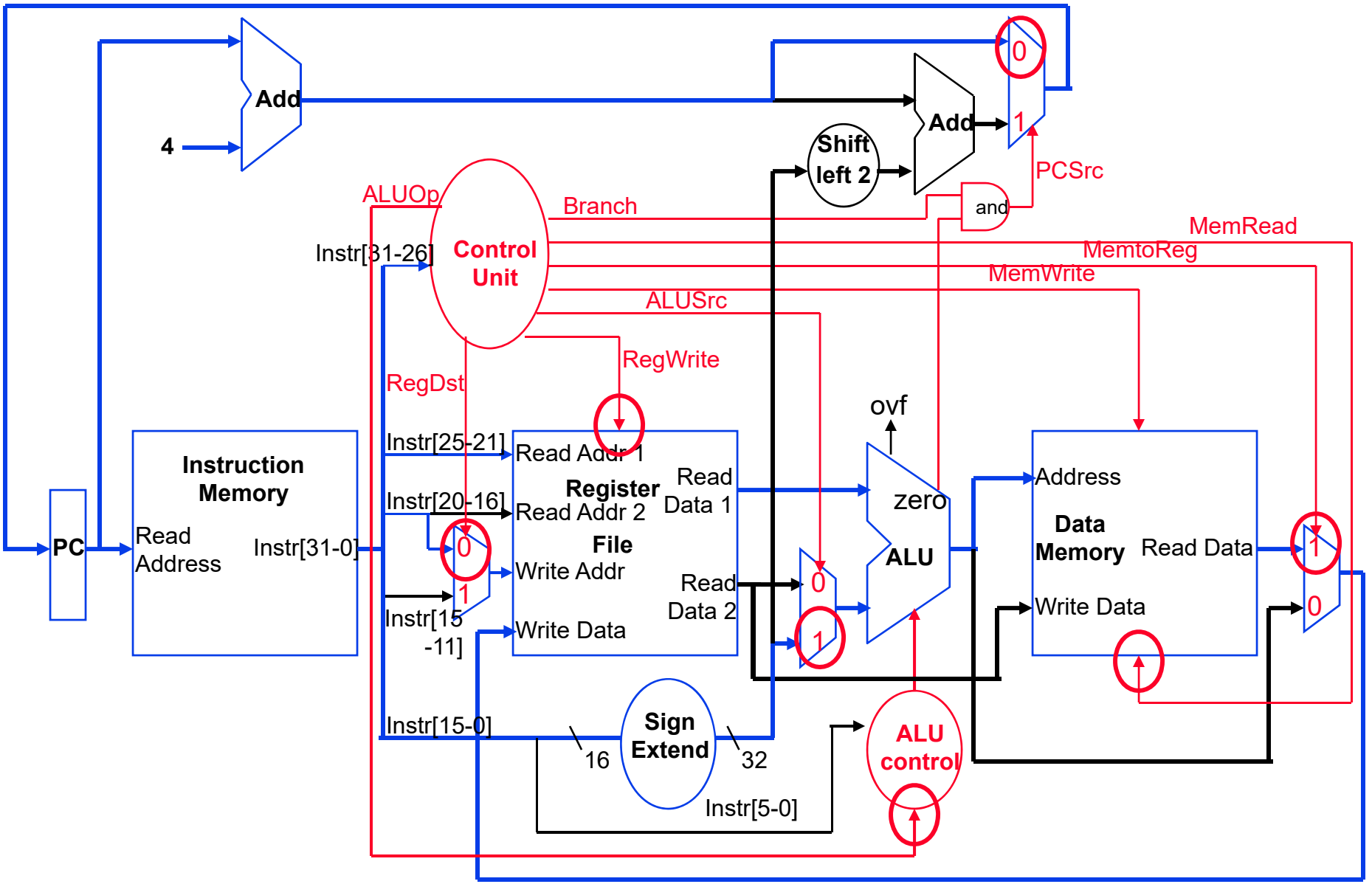
- Control signals
  - RegDST=0
  - Branch=0
  - ALUSrc=1
  - ALUOp = add,
  - MemtoReg=1
  - MemRead = 1
  - RegWrite = 1



Load Word Instruction Data/Control Flow



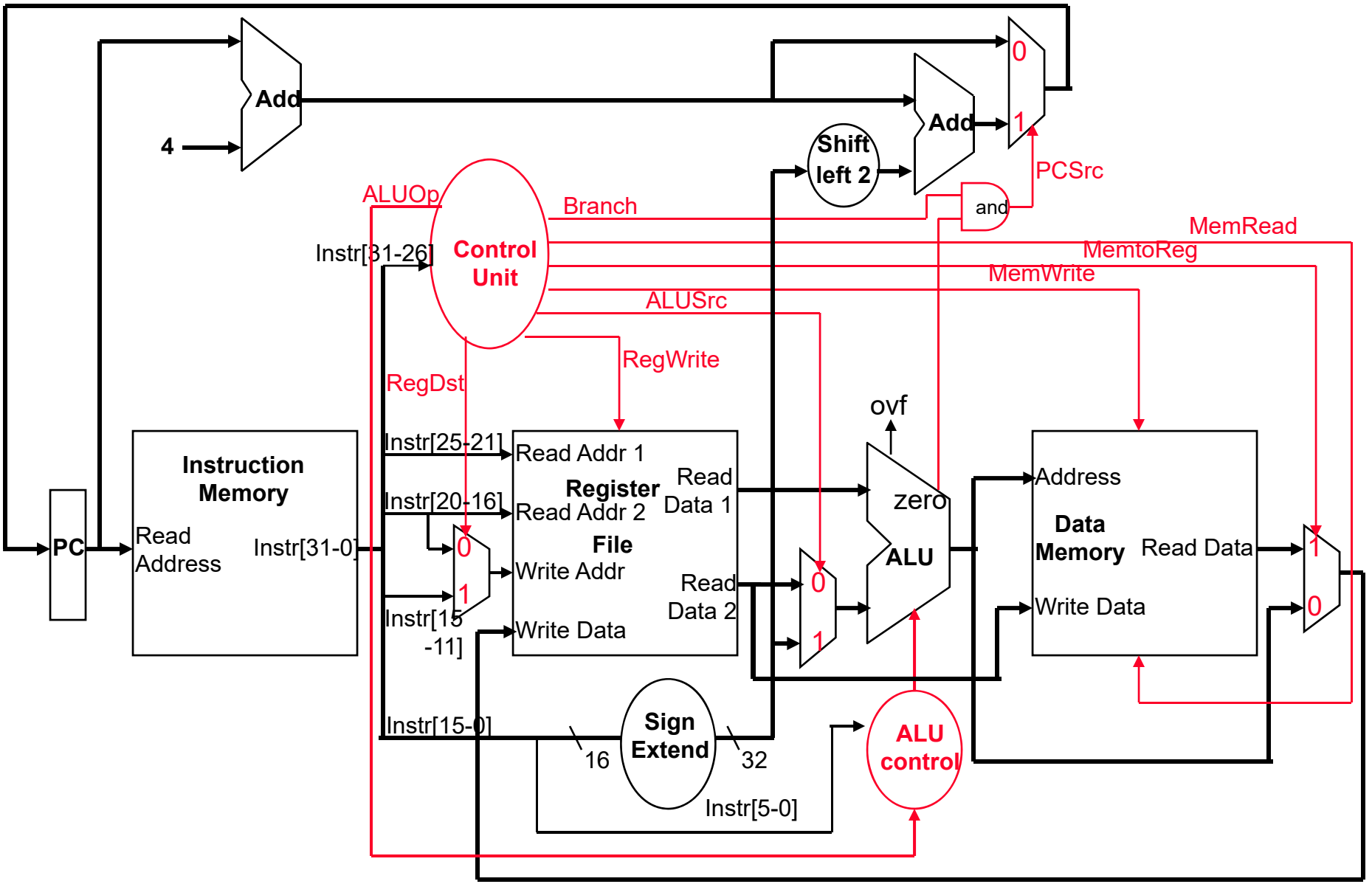
Load Word Instruction Data/Control Flow



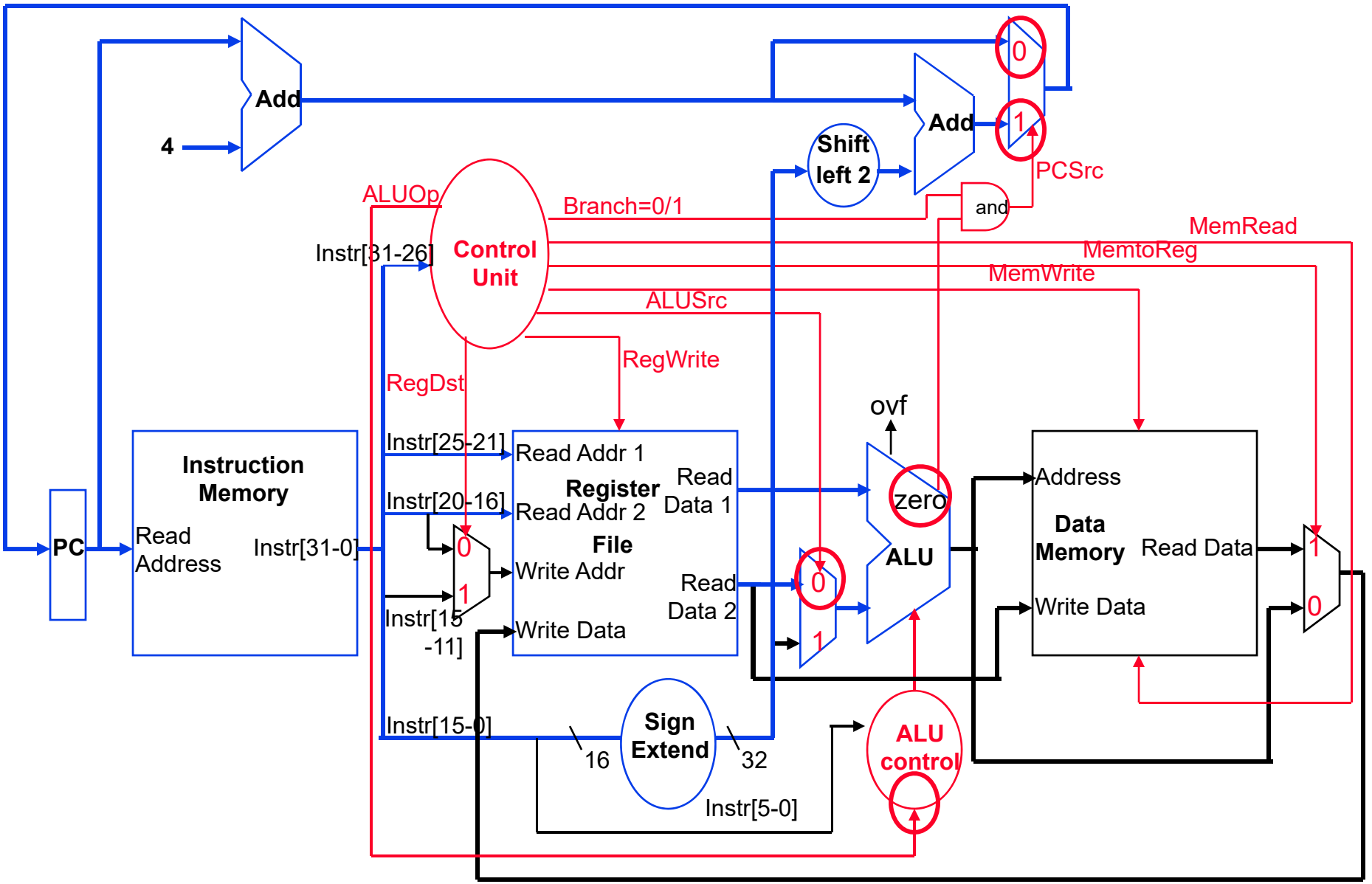
Explanation of Load Word Instruction Signals

- PC+4 or go to branch address is controlled by the output of AND gate:
  - Two inputs: branch = 0/1, logic Zero=0 (not logic op) from ALU
  - In this case, branch = 0, multiplexor selection = 0, thus PC+4
- Two ALU inputs:
  - Read Data 1 and the other one is ALUSrc signal (1 for immediate bits of offset value)
- Which ALU Operation is selected by signal ALUOp (+ in this case)
- The destination register is selected by RegDst (=0, bits 20-16)
- RegWrite =1 triggers the write to register
- MemtoReg and MemRead deliver data from memory to R
- **Only the blue line circuits are affected**

Branch Instruction Data/Control Flow



Branch Instruction Data/Control Flow



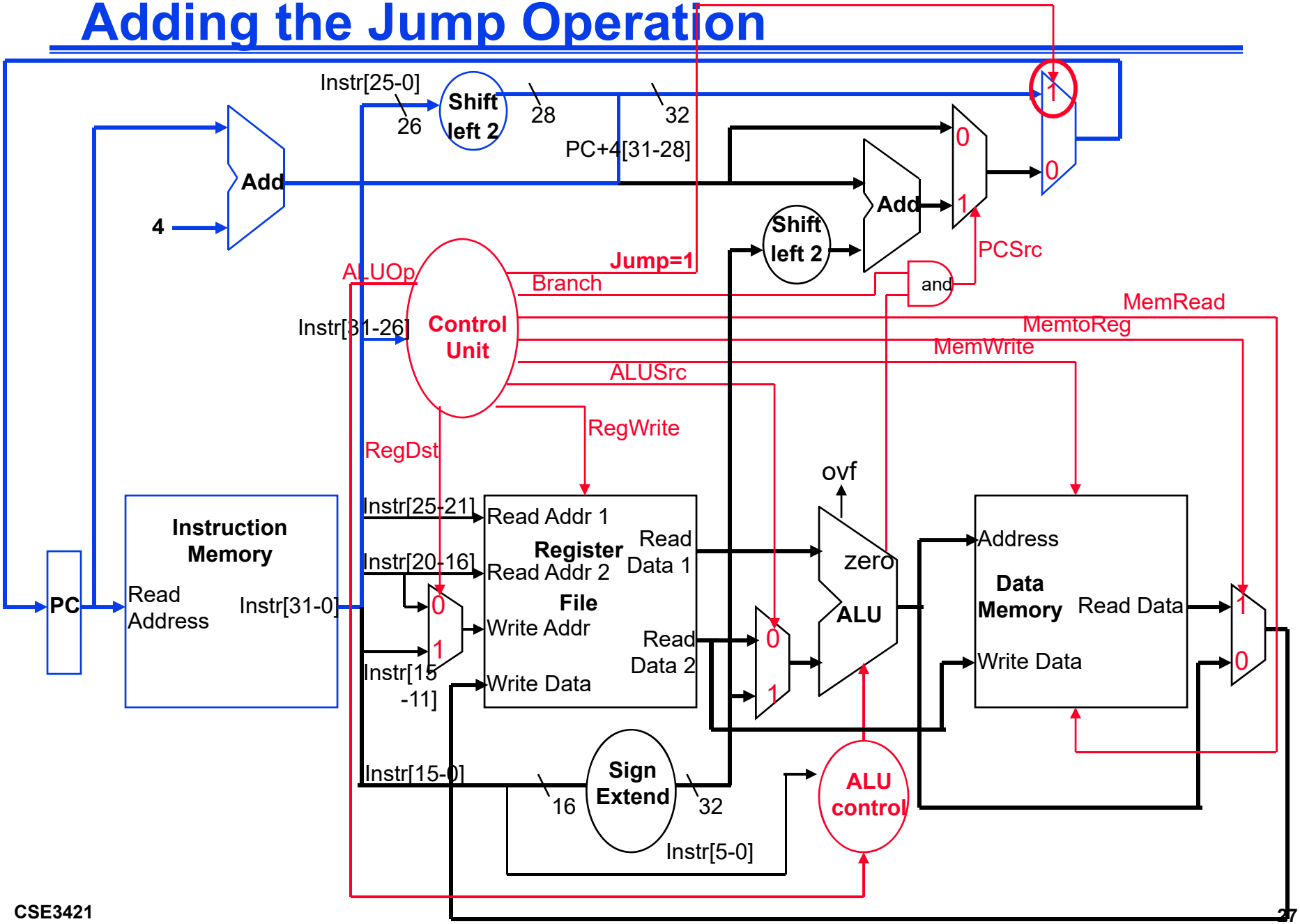
Explanation of Branch prediction

- PC+4 or go to branch address is controlled by the output of AND gate:
  - Two inputs: branch = 0/1, logic Zero = 0/1 after logic op from ALU
  - In this case, branch =1. If the comparison operation is true Zero=1, we prepare the branch address; otherwise, Zero=0.

- Two ALU inputs:
  - Read Data 1 and the other one is ALUSrc signal (0 for data 2)

- A logic ALU Operation is selected by signal ALUOp
  - eq, lt, gt, …
  - The result set Zero to 0 (false) or 1 (true).

Adding the Jump Operation



Explanation of Jump

- Jump signal selects new address
  - Shifting the two bits of 26-bit destination address => 28 bits
  - Concatenating the 4 bits in PC [31-28] with the above 28 bits
  - Replacing the 32 bits to the current PC to be the new address

Key points to read the circuit

- Reading the at the component level:
  1. Instruction memory (fetches a instruction based on PC)
  2. Register file (connects to one or more registers)
  3. ALU (determines what the two inputs are, and what ALU does)
  4. Memory (for a given memory address, read/write a word)
  5. A register (rd in R-type, rt in I-type) gets the result

- Control signals
  - Generated by the decoder to select the input/output of the above components, and signals to perform various operations in each component, e.g. ALUOp, MemRead, RegWrite, …

- Multiplexors are used for different selections.

Truth Table for (Main) Control Unit

Input Output

	Op-code	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-type	000000	1	0	0	1	d	0	0	1	0
lw	100011	0	1	1	1	1	0	0	0	0
sw	101011	d	1	d	0	0	1	0	0	0
beq	000100	d	0	d	0	d	0	1	0	1

Truth Table of ALU Control Unit

Input | Output

ALUOp		Funct field						ALU Control	
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0		
0	0	d	d	d	d	d	d	0 0 10	add
0	1	d	d	d	d	d	d	0 1 10	sub
1	0	1	0	0	0	0	0	0 0 10	add
1	0	1	0	0	0	1	0	0 1 10	sub
1	0	1	0	0	1	0	0	0 0 00	and
1	0	1	0	0	1	0	1	0 0 01	or
1	0	1	0	1	0	1	0	0 1 11	slt
1	0	1	0	0	1	1	1	1 1 00	nor

Ainvert Bivert Operation

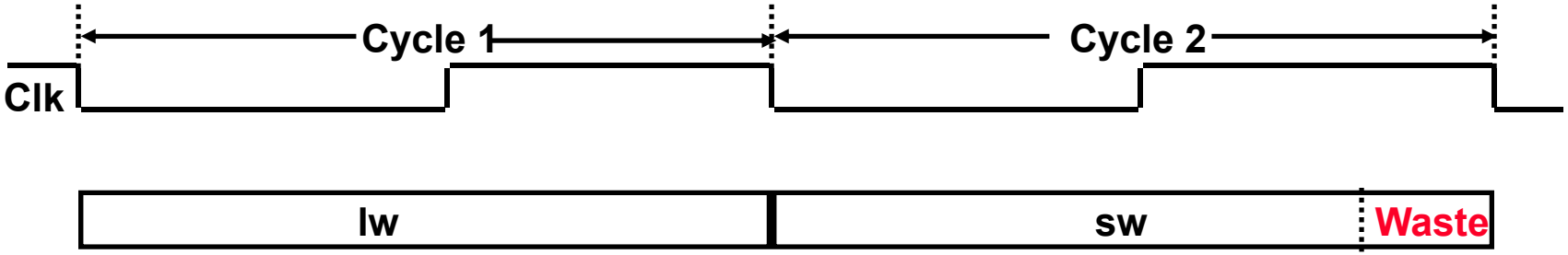
Instruction Critical Paths

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

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 must be timed to accommodate the slowest instruction
  - especially problematic for more complex instructions like floating point multiply



- May be wasteful of area since some functional units (e.g., adders) must be duplicated since they can not be shared during a clock cycle

but

- Is simple and easy to understand

How Can We Make It Faster?

- Start fetching and executing the next instruction before the current one has completed
  - Pipelining – (all?) modern processors are pipelined for performance
  - Remember *the* performance equation:

    CPU time = CPI * CC * IC
    - CPI: cycle per instruction, CC: clock cycle time, IC: instruction count
- Under *ideal* conditions and with a large number of instructions, the speedup from pipelining is approximately equal to the number of pipe stages
  - A five stage pipeline is nearly five times faster because the CC is shorten nearly five times compared with a single cycle design
- Fetch (and execute) more than one instruction at a time