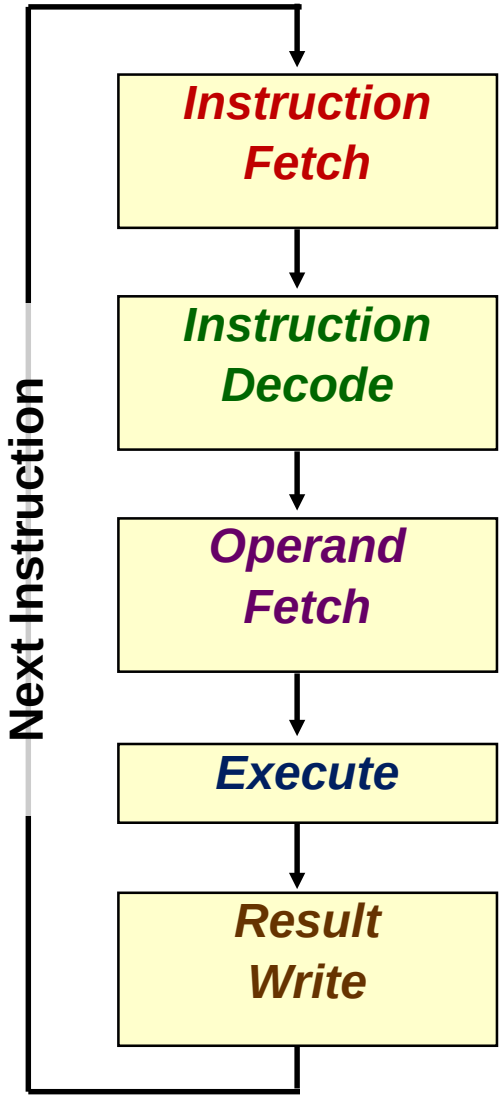
Chap. 4 The Processor

Part A – Simple Implementation

** NUS, Aaron Tan 교수의 강의자료를 일부 포함하고 있습니다. **

Instruction Execution Cycle (Basic)



Fetch:

- Get instruction from memory
- Address is in **P**rogram **C**ounter (PC) Register

Decode:

- Find out the operation required

Operand Fetch:

- Get operand(s) needed for operation

Execute:

- Perform the required operation

Result Write (Store):

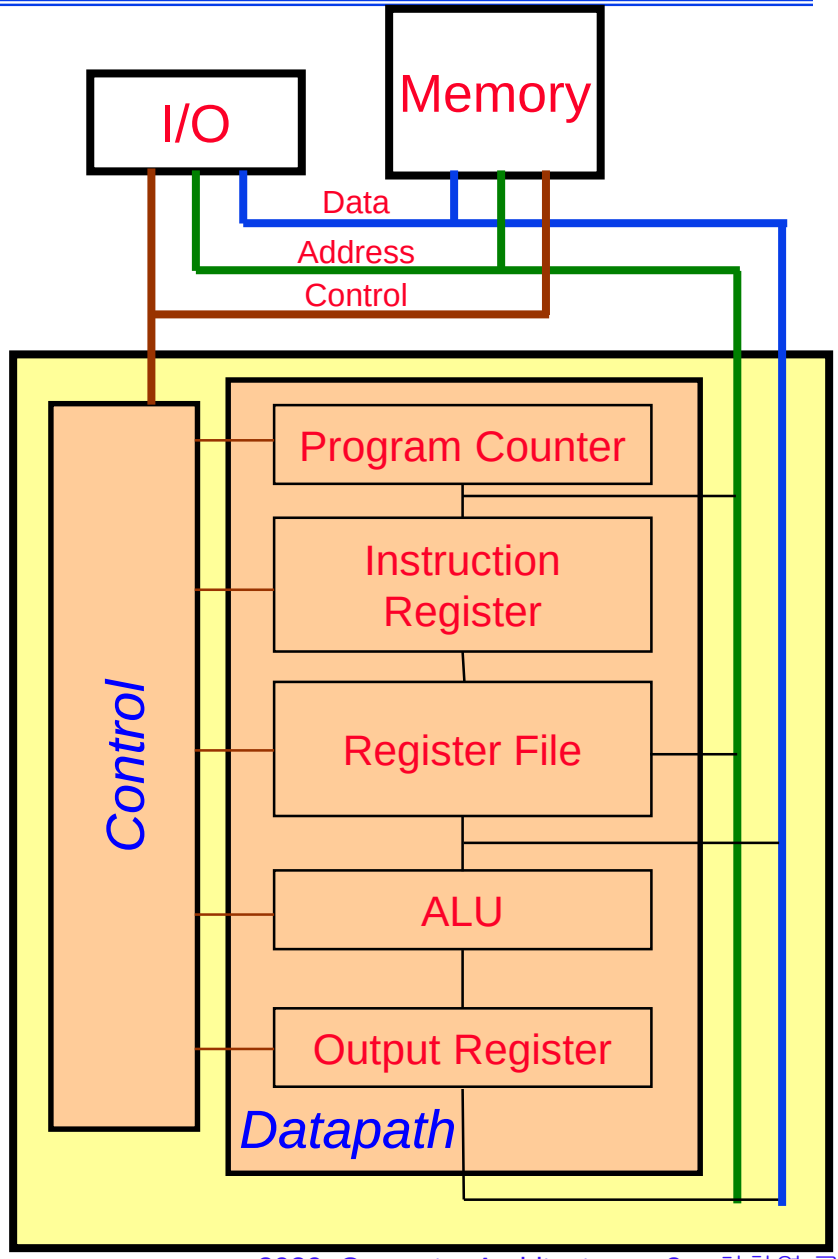
- Store the result of the operation

The Processor

- 2 Major Components for a processor
  - Datapath
    - Performs the arithmetic, logical and memory operations
    - Collection of components that process data

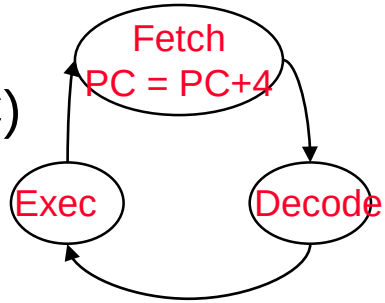
      Arithmetic Logic Unit(ALU),

      Shifters, Registers, ultipliers
  - Control
    - Tells the datapath, memory and I/O devices what to do according to program instructions



The Processor (Datapath & Control) Implementation

- Our implementation of the MIPS is simplified
  - arithmetic and logical instructions: **`add, sub, and, or, slt`**
  - memory-reference instructions: **`lw, sw`**
  - control flow instructions: **`beq, bne, j`**
- Generic implementation
  - use the PC to supply the instruction address and fetch the instruction from memory (and update the PC)
  - decode the instruction (and read registers)
  - execute the instruction



- All instructions (except **`j`**) use the ALU after reading the registers
  - *Arithmetic result : add, addi, sub, and, or*
  - *Memory address for load/store*
  - *Comparison result for branches*

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

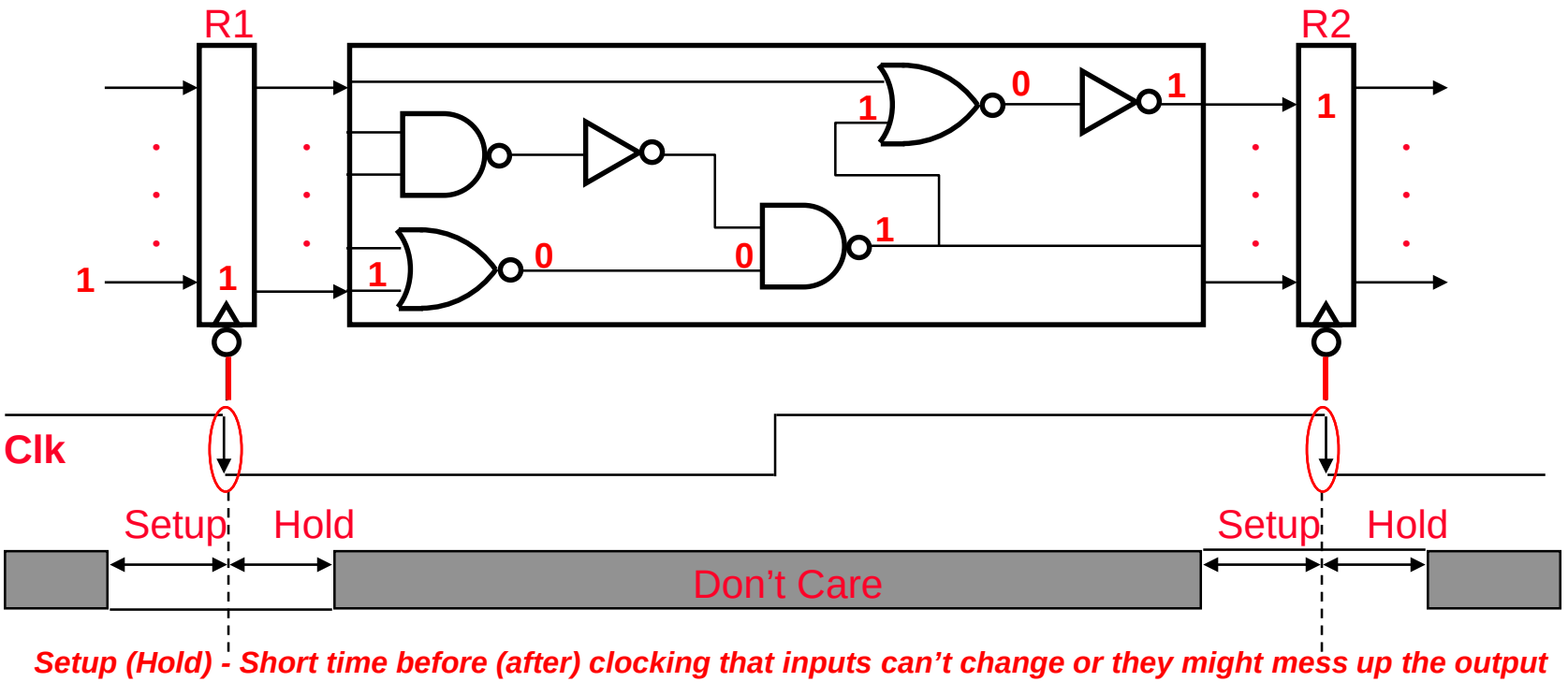
- Typical execution
  - read contents of state elements
  - send values through combinational logic
  - write results to one or more state elements

Register Transfer Language and Clocking

Register transfer in RTL:

R2 ← f(R1)

What Really Happens Physically



Two possible clocking methodologies : positively triggered or negatively triggered. This class uses the negatively-triggered.

Building a Datapath

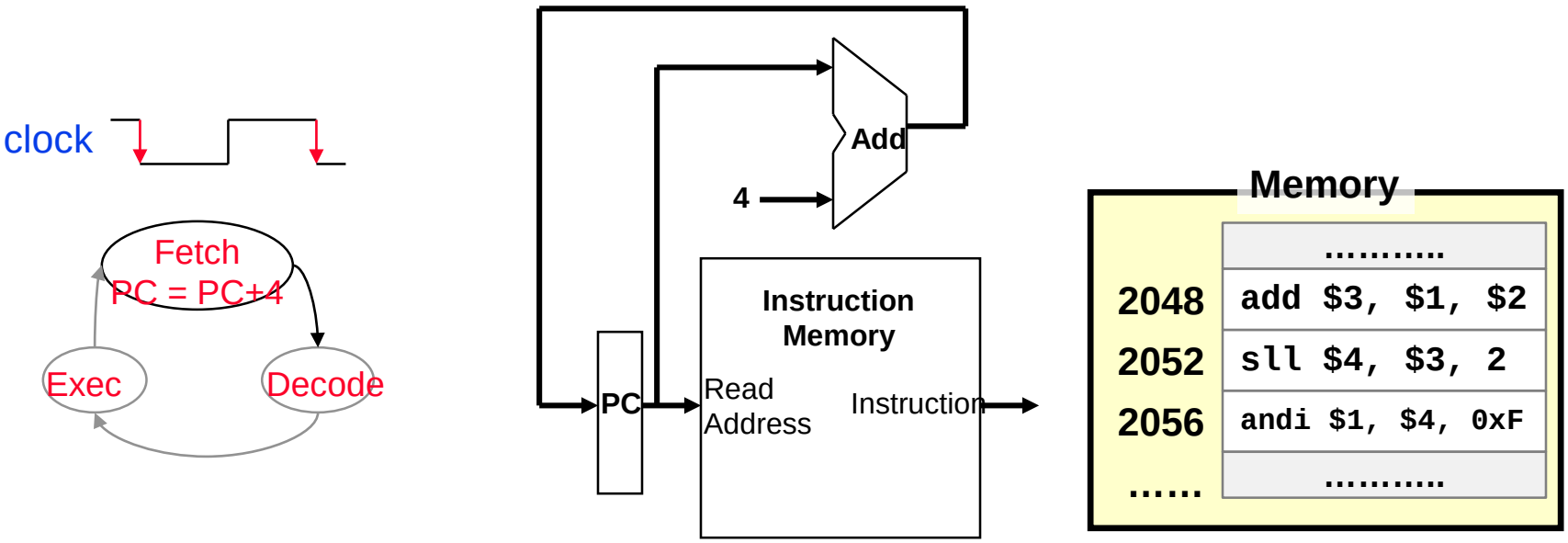
- MIPS Instruction Execution
- Design changes:
  - Merge *Decode* and *Operand Fetch* – Decode is simple for MIPS
  - Split *Execute* into ***ALU*** (Calculation) and ***Memory Access***

	`add $3, $1, $2`	`lw $3, 20( $1 )`	`beq $1, $2, label`
Fetch	Read inst. at [PC]	Read inst. at [PC]	Read inst. at [PC]
Decode & Operand Fetch	o Read [**$1**] as *opr1* o Read [**$2**] as *opr2*	o Read [**$1**] as *opr1* o Use ***20*** as *opr2*	o Read [**$1**] as *opr1* o Read [**$2**] as *opr2*
ALU	*Result* = *opr1* + *opr2*	*MemAddr* = *opr1* + *opr2*	*Taken* = (*opr1* == *opr2*)? *Target* = (**PC**+4) + **ofst×4**
Memory Access		Use *MemAddr* to read from memory	
Result Write	*Result* stored in **$3**	*Memory* data stored in **$3**	if (*Taken*) **PC** = *Target*

opr = operand **MemAddr** = Memory Address **ofst** = offset

Fetching Instructions

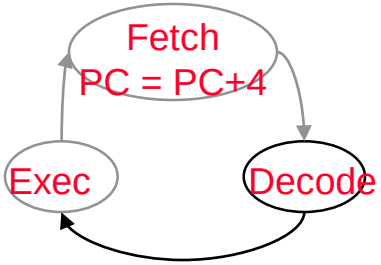
- Fetching instructions involves
  - reading the instruction from the Instruction Memory
  - updating the PC value to be the address of the next (sequential) instruction



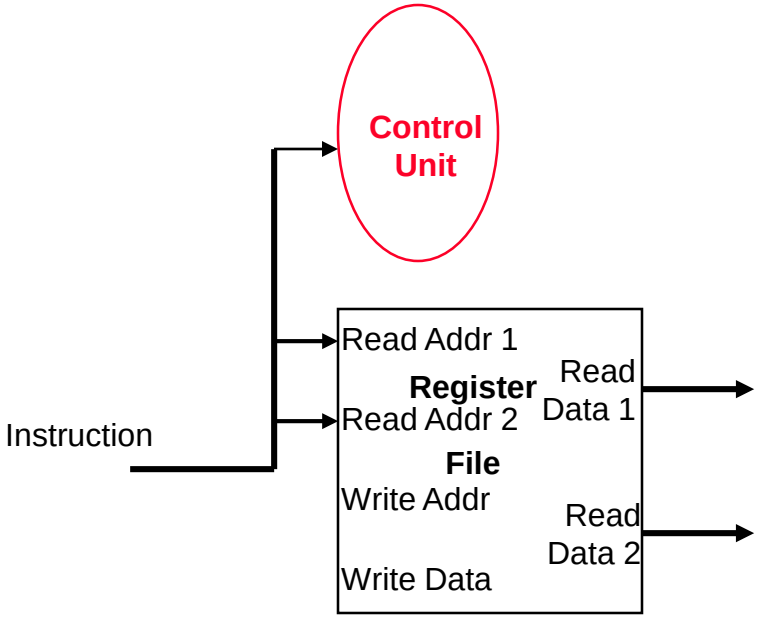
- PC is updated every clock cycle, so it does not need an explicit write control signal just a clock signal
- Reading from the Instruction Memory is a combinational activity, so it doesn't need an explicit read control signal

Decoding Instructions

- Decoding instructions involves
  - sending the fetched instruction's opcode and function field bits to the control unit

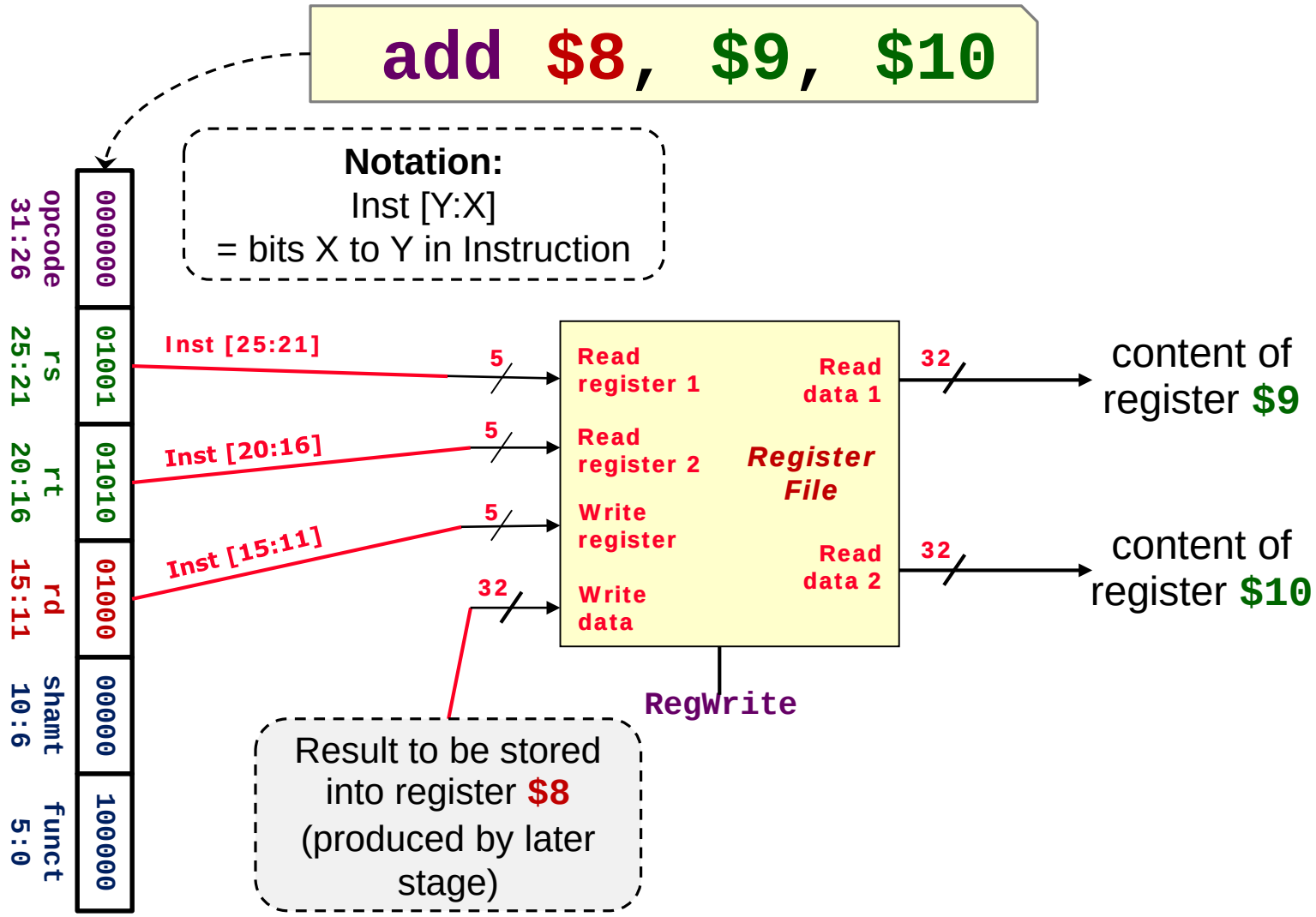


and

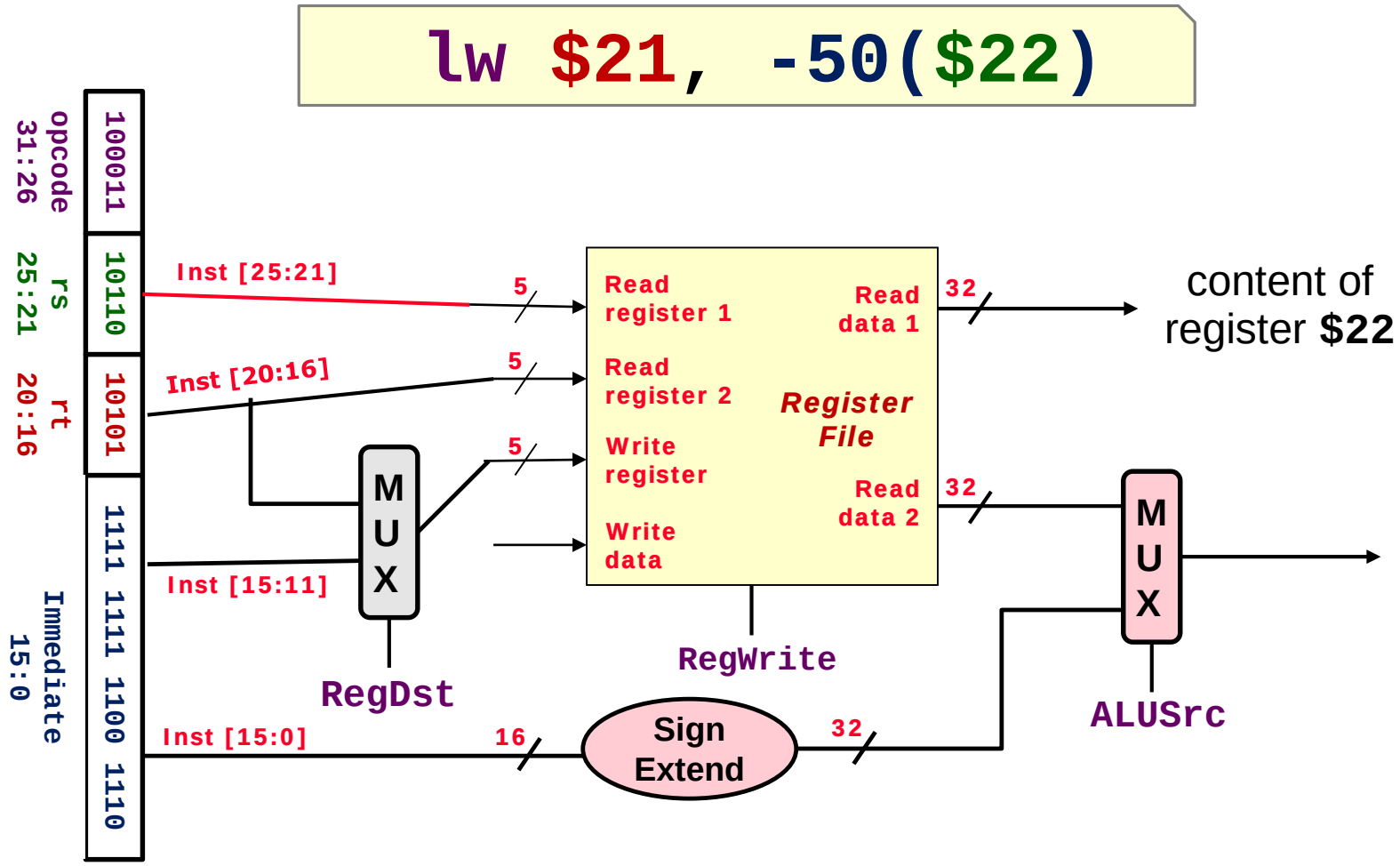


  - reading two values from the Register File
    - Register File addresses are contained in the instruction

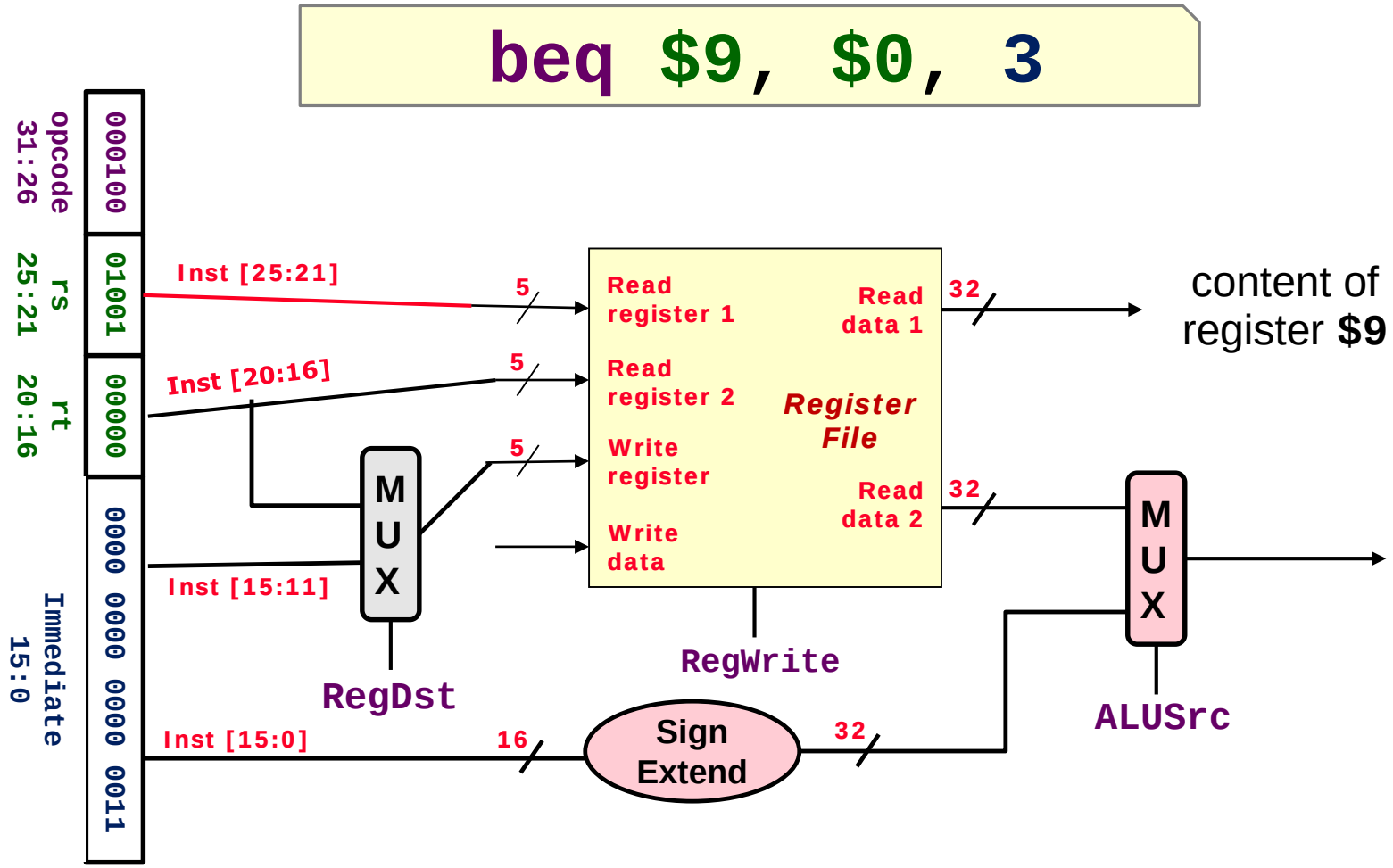
Decode Stage: **R-Format Instruction**



Decode Stage: **Load Word Instruction**

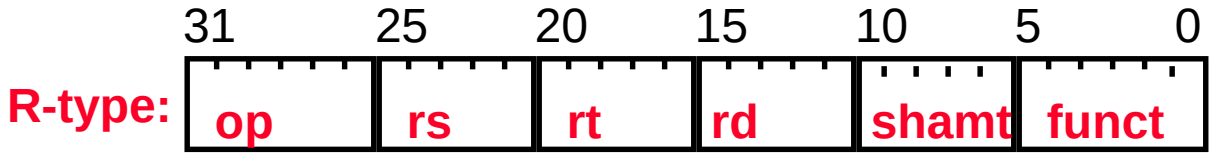


Decode Stage: **Branch Instruction**

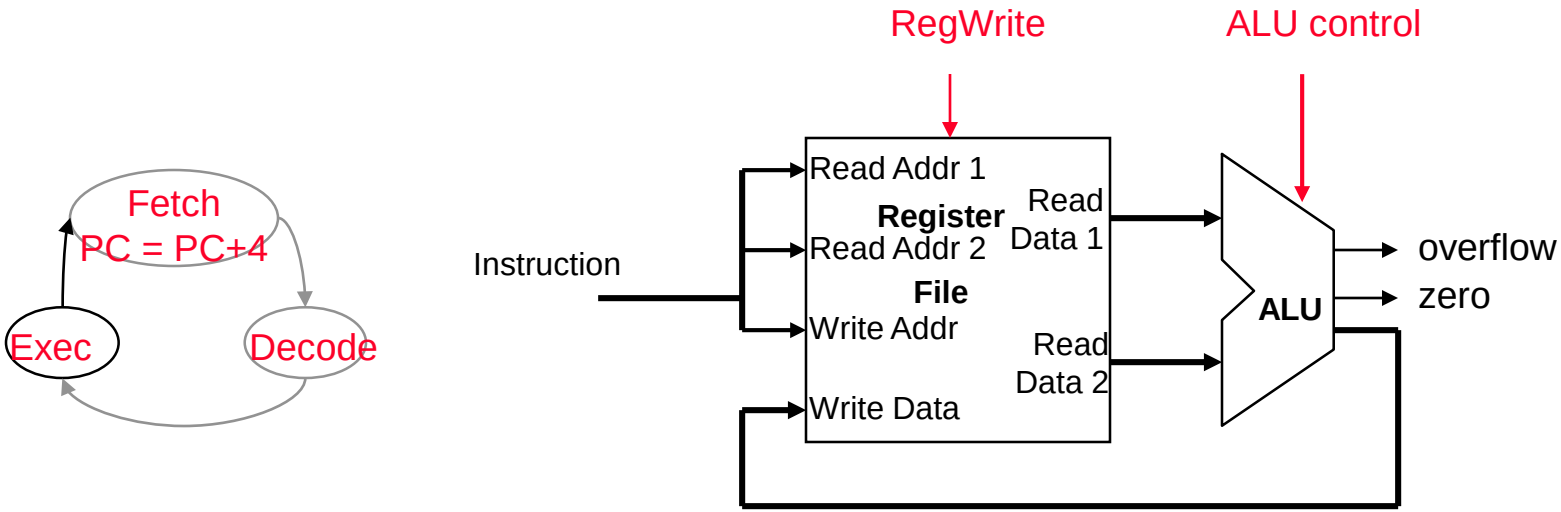


Executing R Format Operations

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



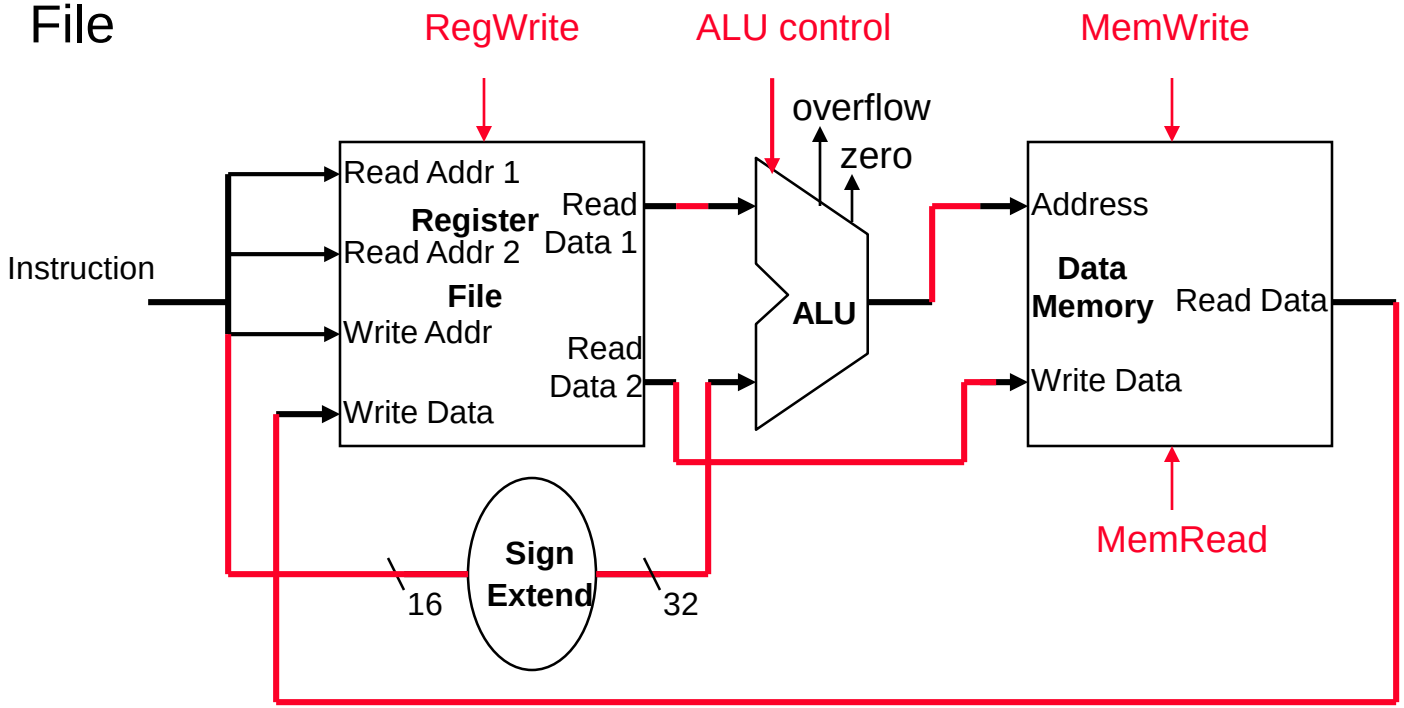
- perform operation (op and funct) on values in rs and rt
- store the result back into the Register File (into location rd)



- Note that Register File is not written every cycle (e.g. **`sw`**), so we need an explicit write control signal for the Register File

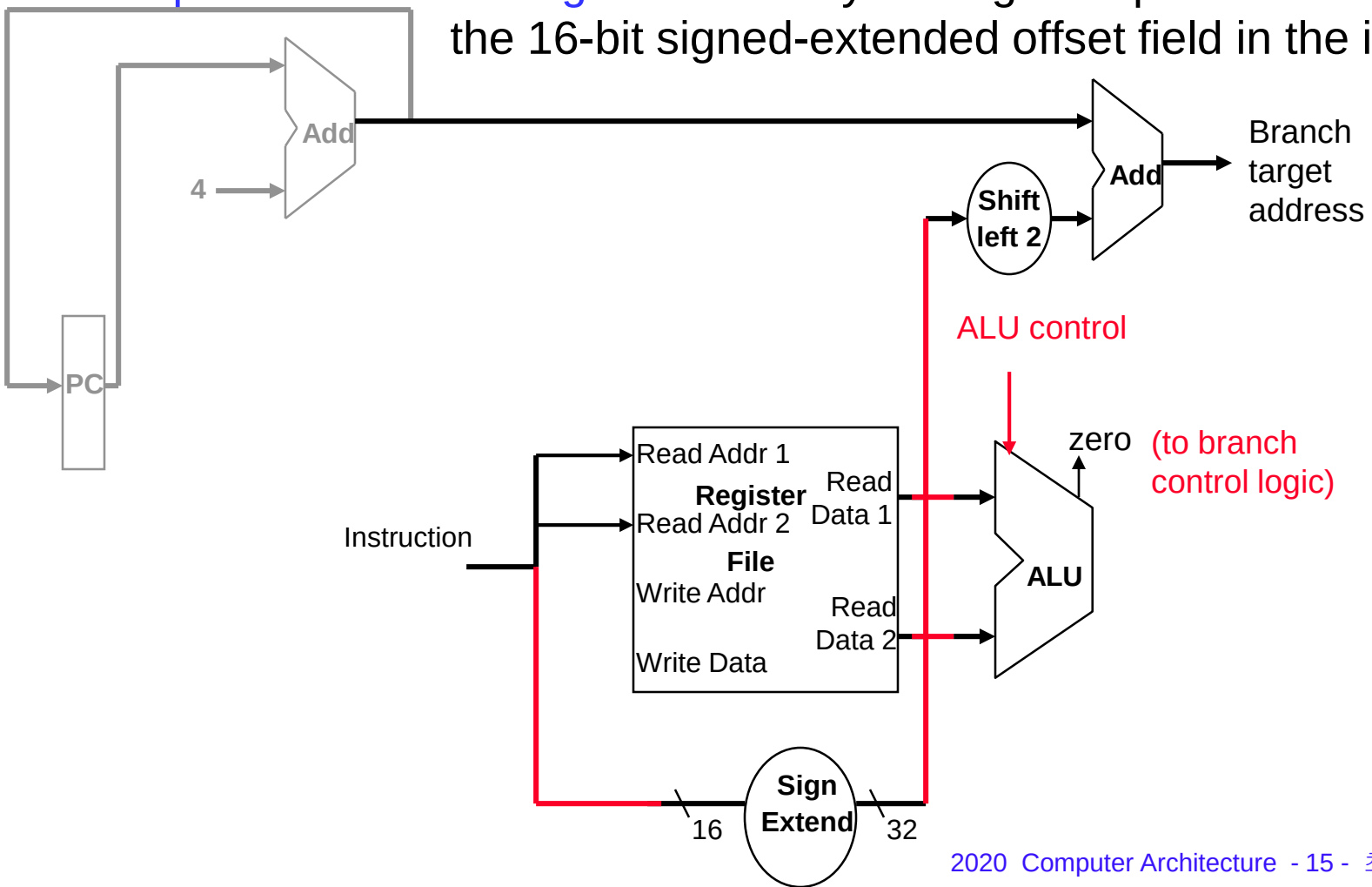
Executing Load and Store Operations

- 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 (read from the Register File during decode) written to the Data Memory
  - load value, read from the Data Memory, written to the Register File



Executing Branch Operations

- Branch operations involves
  - compare the operands read from the Register File during decode for equality (**`zero`** ALU output)
  - compute the branch target address by adding the updated PC to the 16-bit signed-extended offset field in the instr



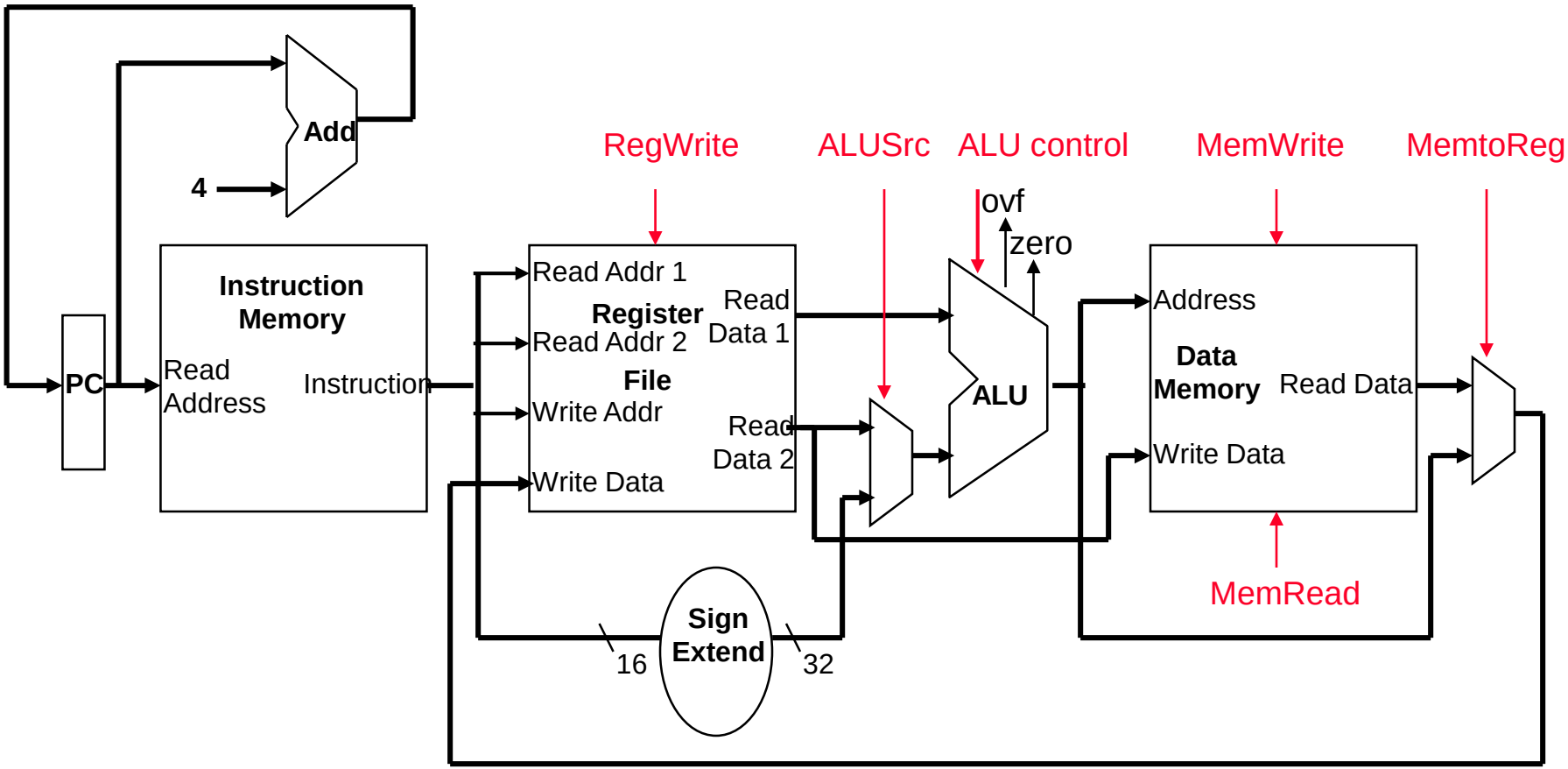
Creating a Single Datapath from the Parts

- 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



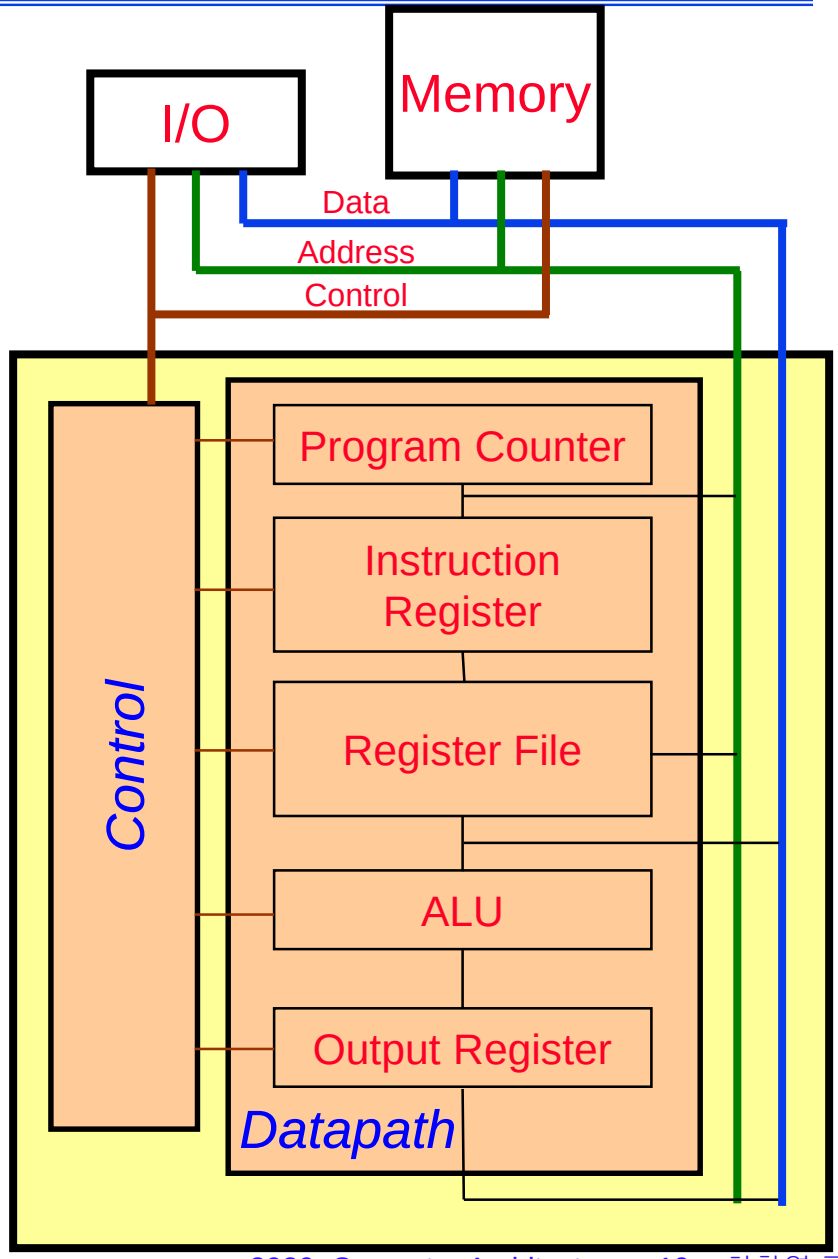
- Summary
  - Single cycle datapath

Adding the Control

Selecting the operations to perform

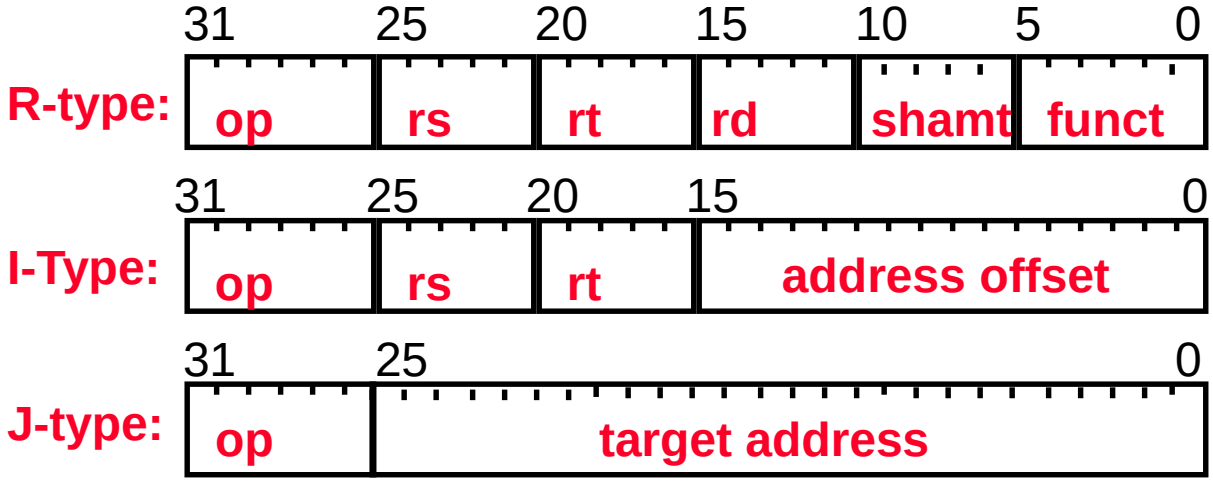
- ALU
- Register File
- and Memory read/write

Controlling the flow of data (multiplexor inputs)



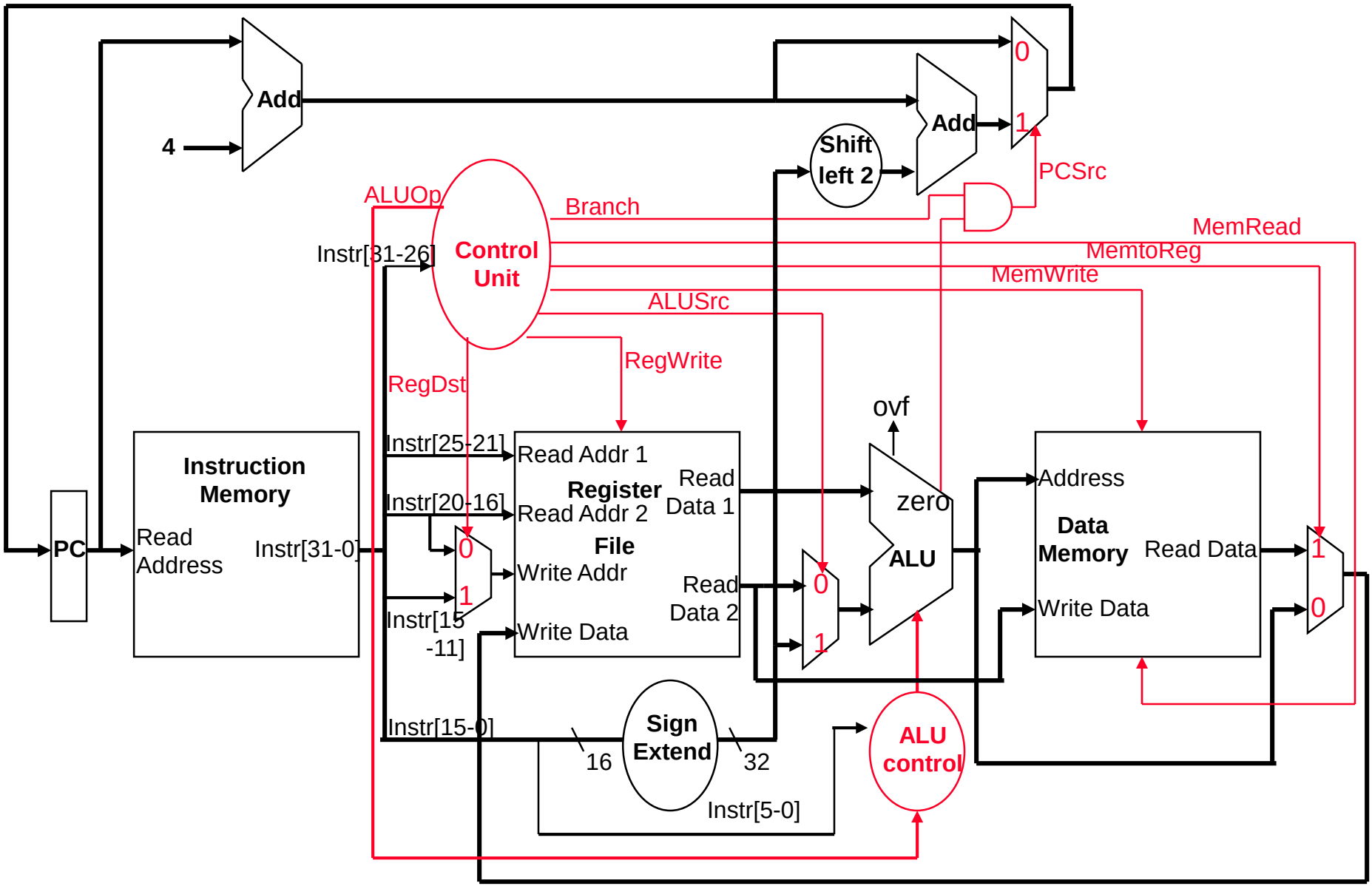
Adding the Control

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

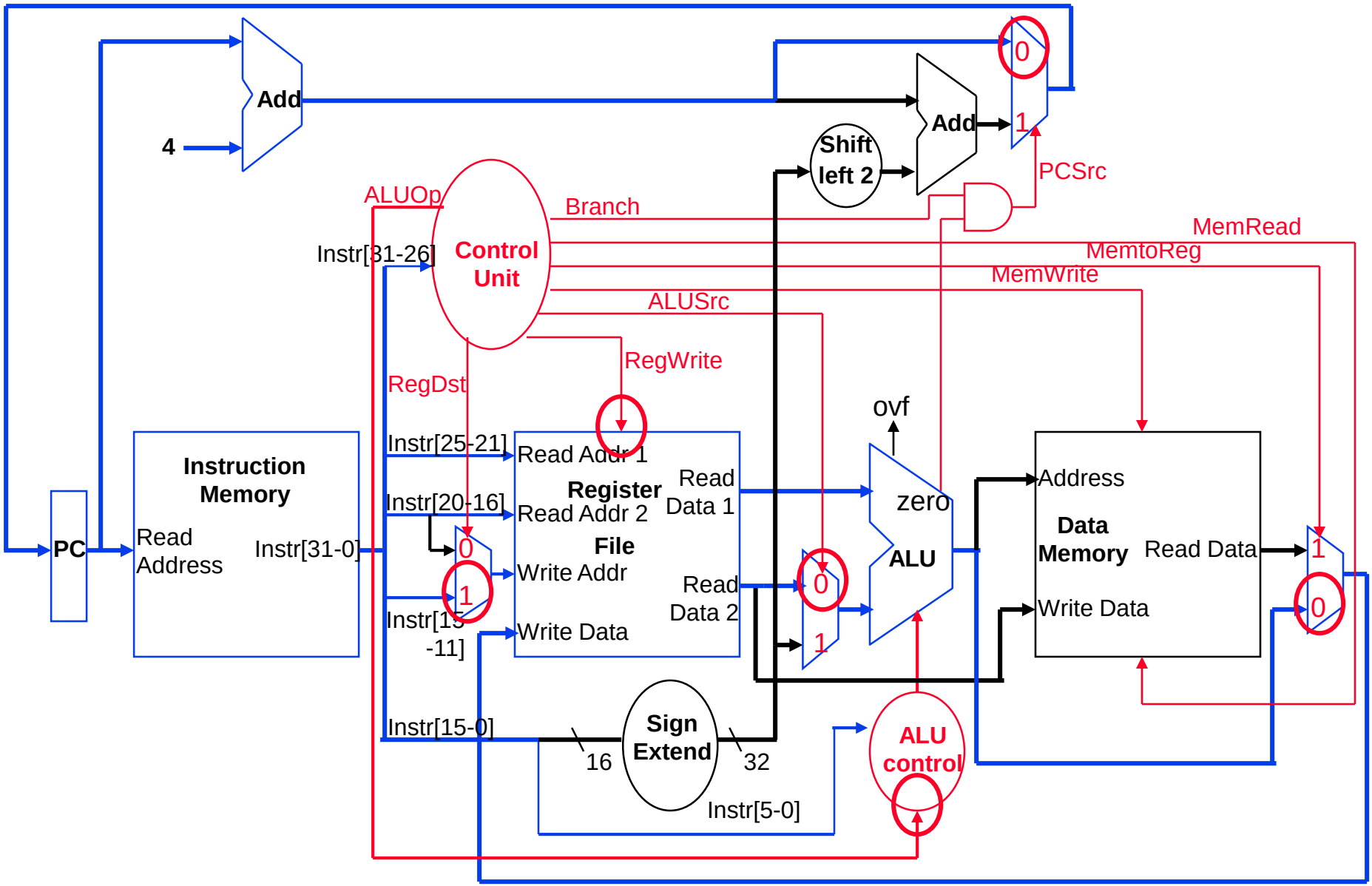


- Observations
  - op field always in bits 31-26
  - addr of registers to be read are always specified by the rs field (bits 25-21) and rt field (bits 20-16); for lw and sw rs is the base register
  - addr. of register to be written is in one of two places – in rt (bits 20-16) for lw; in rd (bits 15-11)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

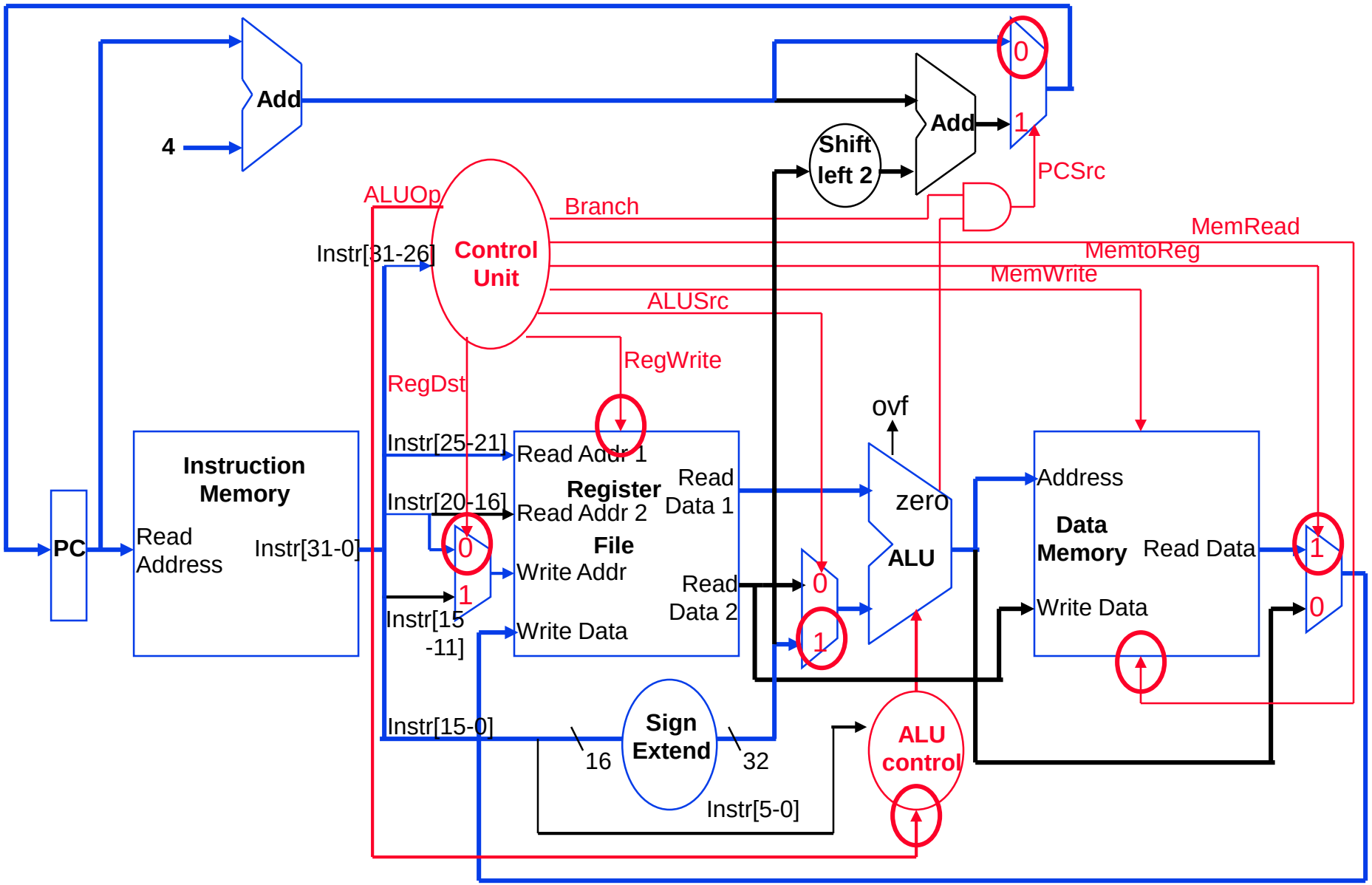
Single Cycle Datapath with Control Unit



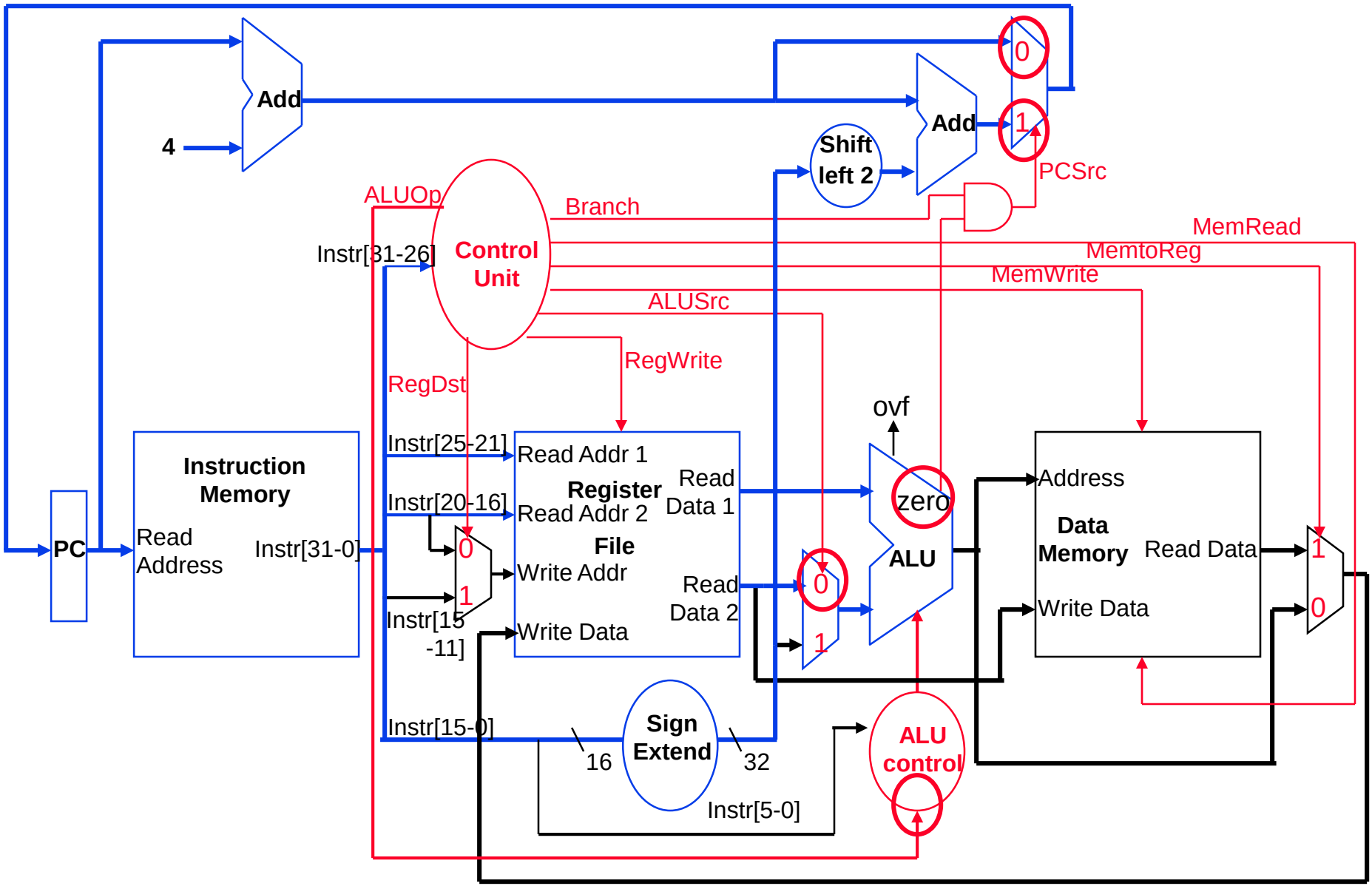
R-type Instruction Data/Control Flow



Load Word Instruction Data/Control Flow



Branch Instruction Data/Control Flow



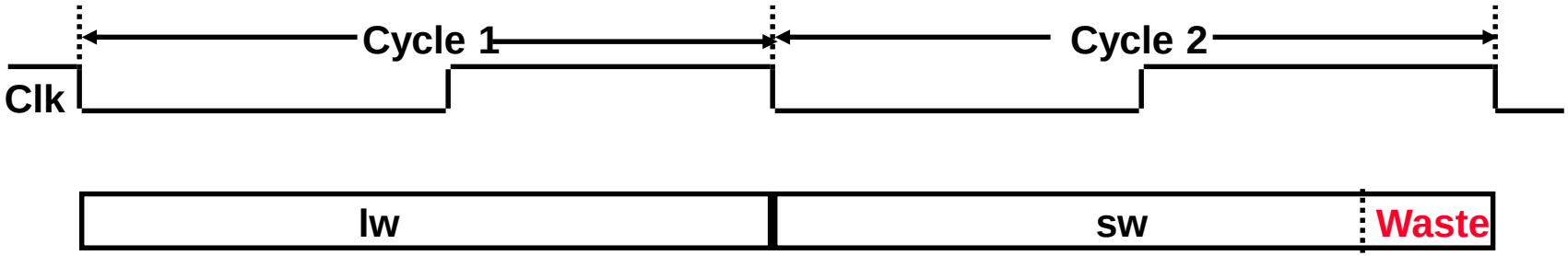
Instruction Critical Paths

- 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 (Clock) Cycle Implementation

- 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

Summary