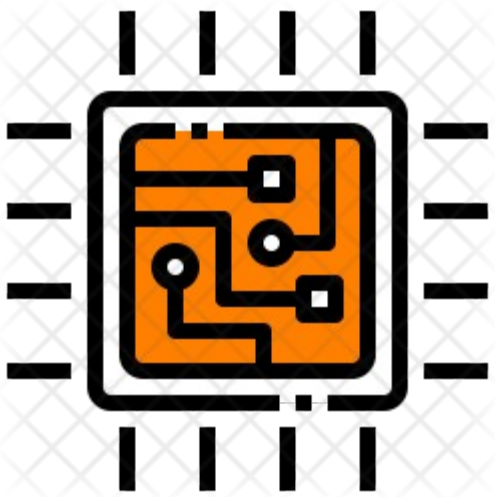
Discussion

- Explain what is the meant by the design principal “make the common case fast.” Considering MIPS processor, identify a design choice that uses this principal. Identify a special case (an uncommon case) and explain how MIPS handle it.
- Floating point number representation involves splitting the data unit (e.g., word) into multiple fields. What are these fields? How the stored value is calculated? How would changing the size of these fields affect the number precision and range?
- The principal of “performance by prediction” is used in computer design. Identify one case for which this principal is used to improve the performance. Explain how this principal is used to improve the performance.
- It is well-known that the design of computer has a cost-performance tradeoff. Identify two scenarios that confirm this tradeoff and explain the tradeoff aspects.



MIPS Processor

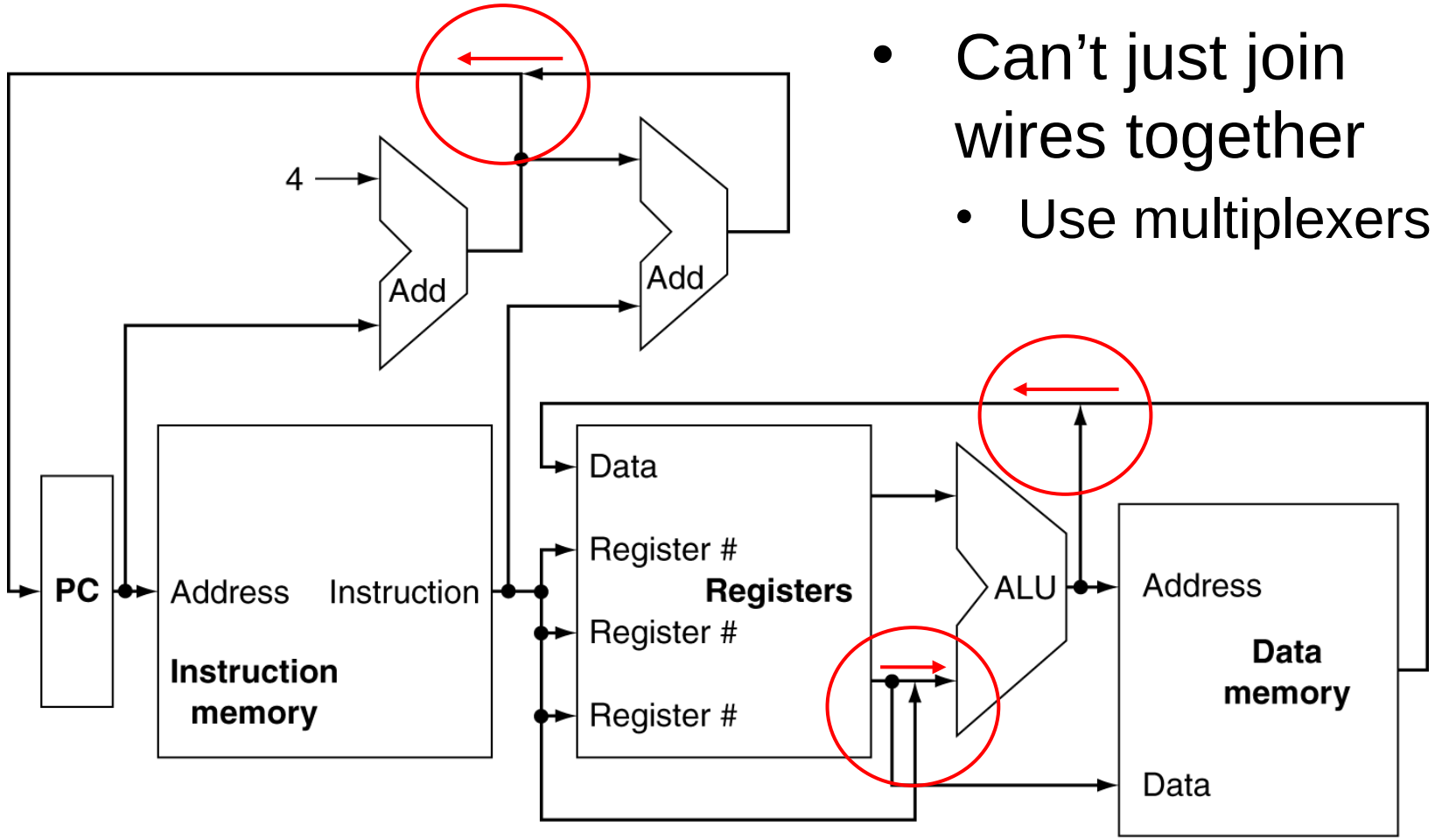
How a processor is designed?

- We explore the ***processor HW design*** using simple instruction subset (shows most aspects)
  a) Memory reference: lw, sw
  b) Arithmetic/logical: add, sub, and, or, slt
  c) Control transfer: beq, j

- We will examine two MIPS implementations
  - A simplified **single clock cycle** processor
    - the whole instruction is processed (fetch + decode+ execute) in one clock cycle
  - A **pipelined** version
    - A new design that speeds **program** execution but should handle **other complexities**

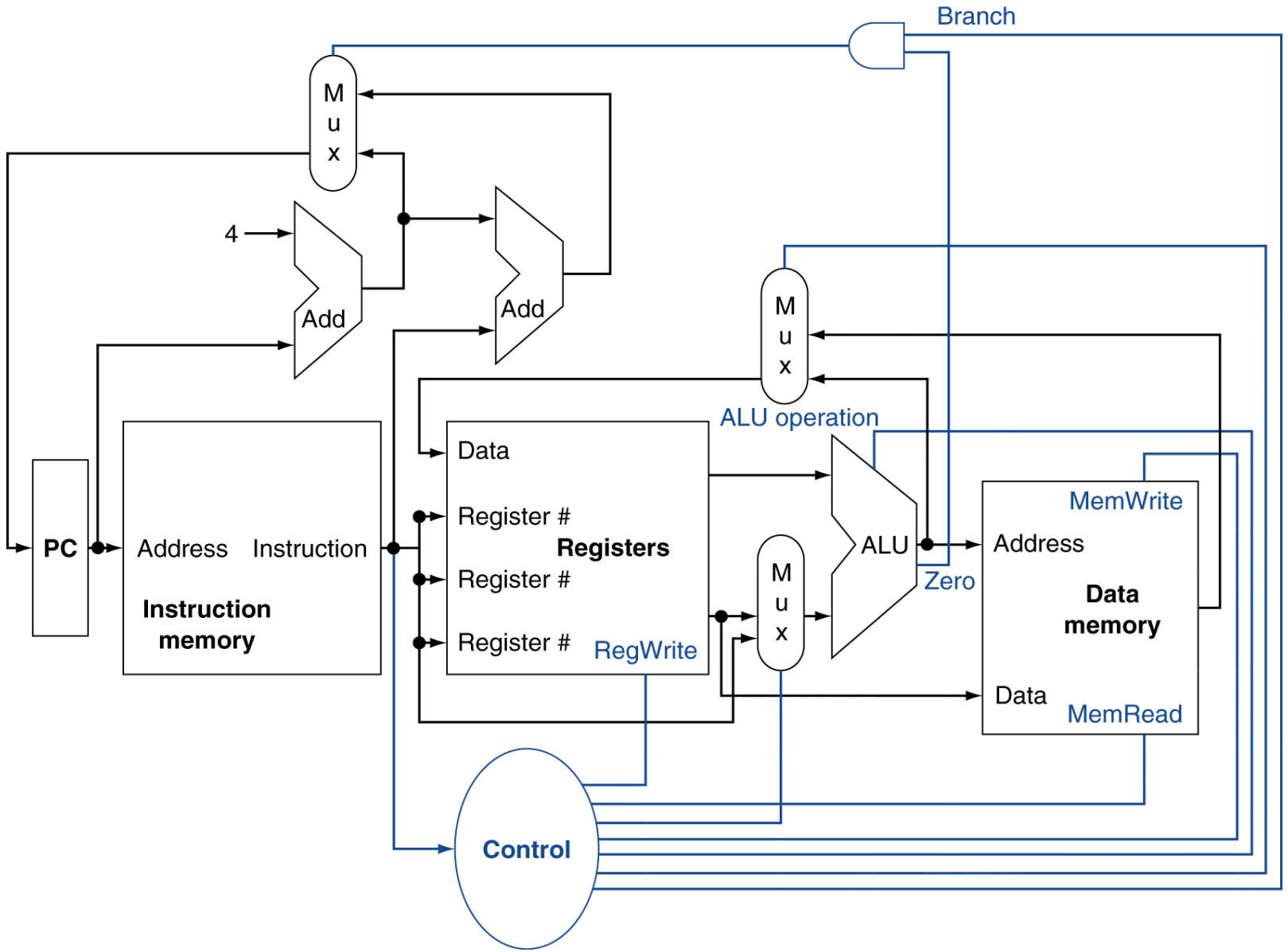
Processors are designed to *execute binary (machine) instructions*

CPU Overview



- Can't just join wires together
  - Use multiplexers

Control

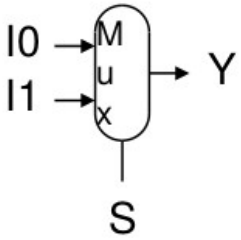


Multiplexers enable selecting an input from a group of sources

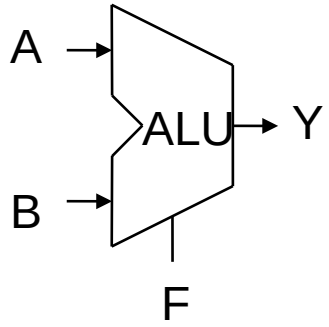
Each processor has two key parts: datapath and control

Processor Components

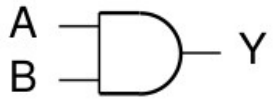
- Multiplexer
  - Y = S ? I1 : I0



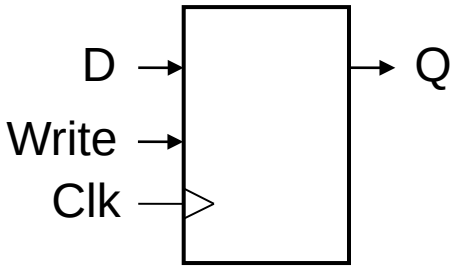
- Arithmetic/Logic Unit
  - Y = F(A, B)

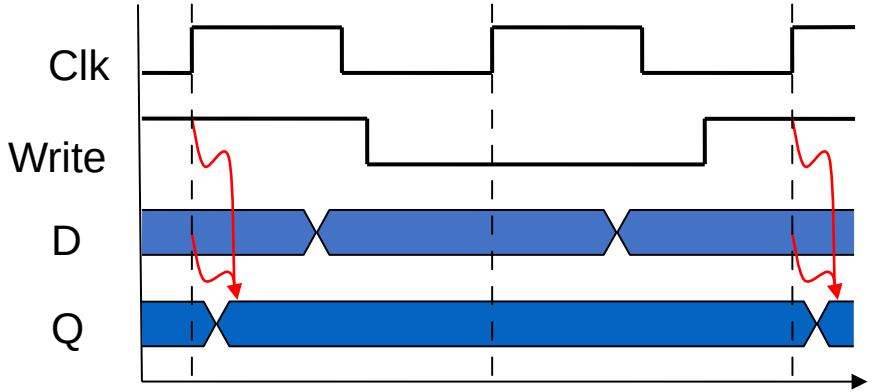


- Logic-gate
  - Y = A & B



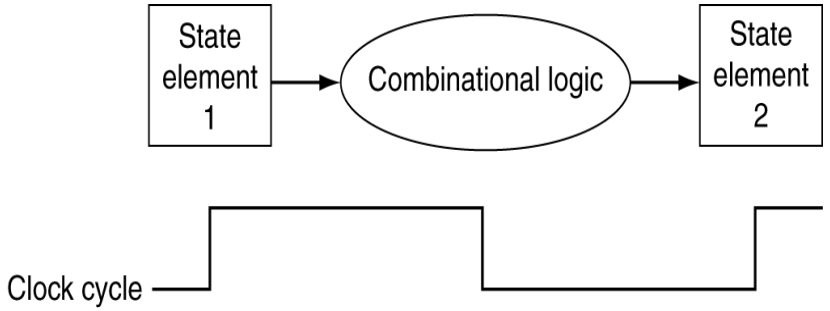
- ***Sequential*** elements

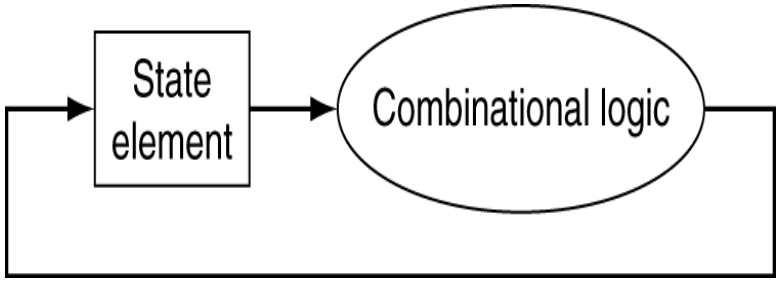




Circuit Operation

- ***Sequential*** circuits process (I/O) data at clock edges
- ***Combinational*** logic transforms data during clock cycles
  - Between clock edges
  - Input from state elements, output to state element
  - Longest delay determines clock period





MIPS
Datapath

Datapath:
Components that process data and addresses in the CPU (e.g., Registers, ALUs, mux's, memories,

Reminder!

Building the Datapath

- We will build a MIPS datapath incrementally
  - ***Identify*** and ***connect*** logic ***components*** required by different processor functions

- Each processor performs two key operations

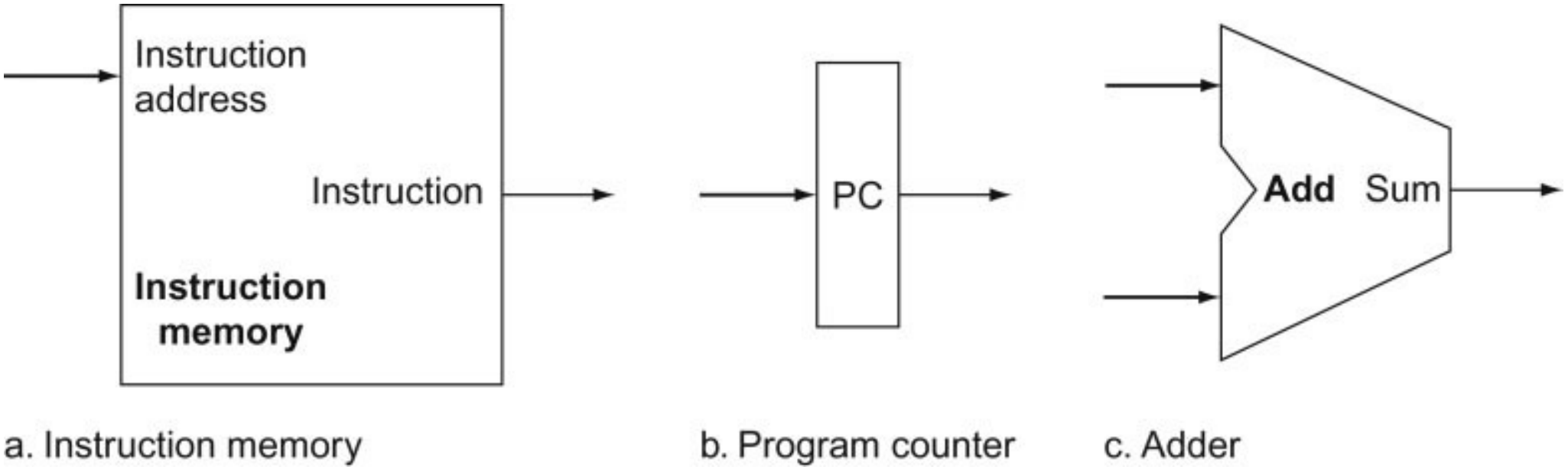
1- Instruction fetch

1) PC → instruction memory
2) PC → target address or PC + 4
   - could be different in case of branching

2- Instruction execution

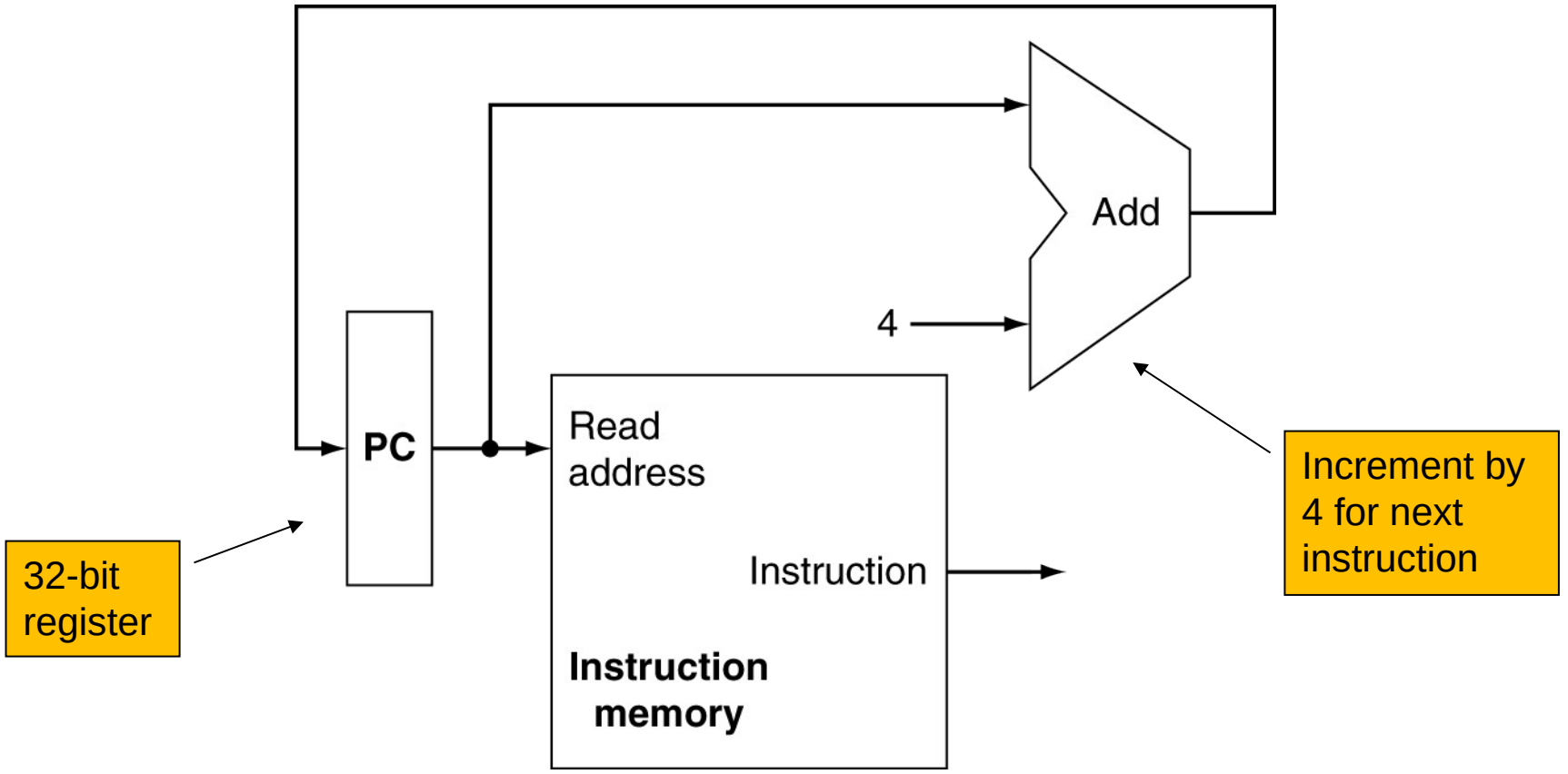
A. Read and/or write registers
B. Use ALU to calculate
   - Arithmetic result
   - Memory address for load/store
   - Branch target address

Instruction Fetch

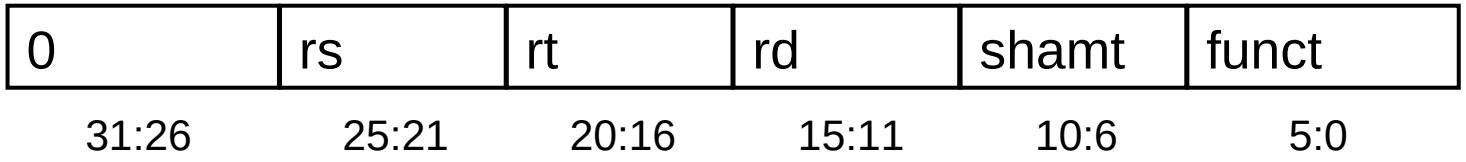


Requires two state elements
Instruction memory and PC

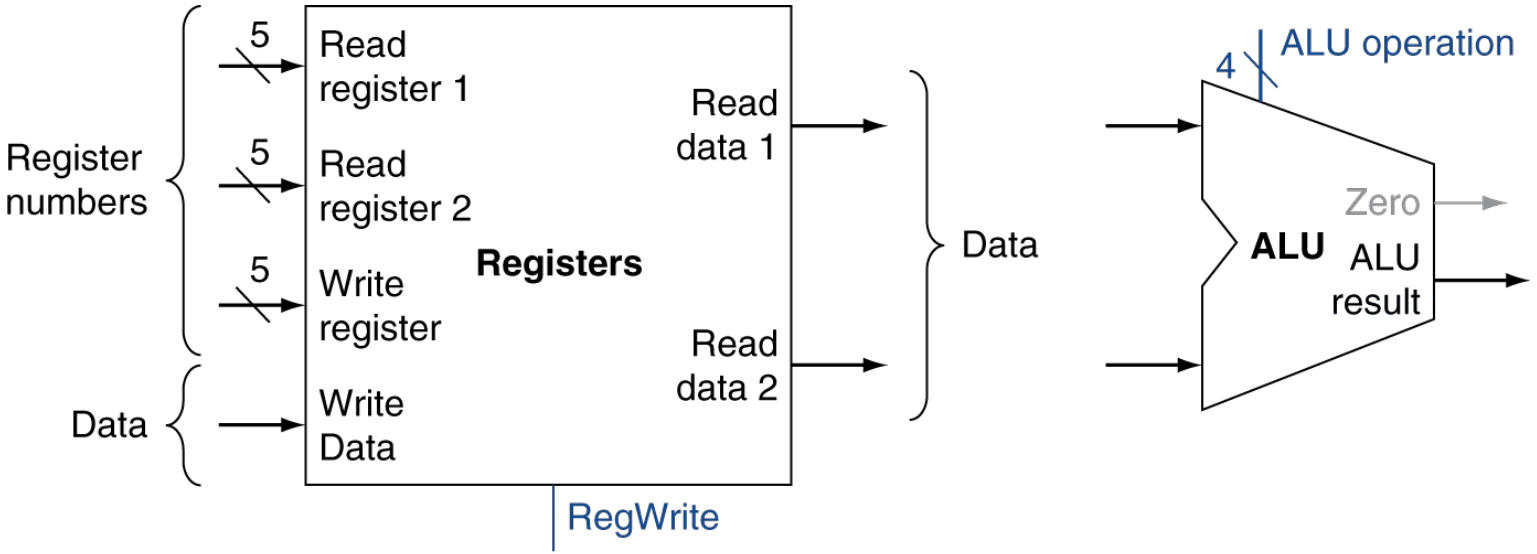
Connecting Instruction Fetch Component



R-Format Instructions



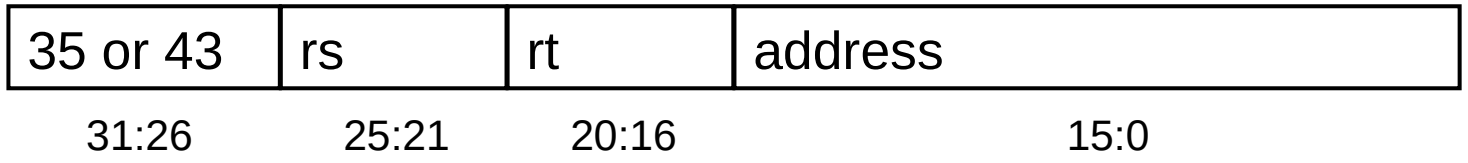
- Read two register operands
- Perform arithmetic/logical operation



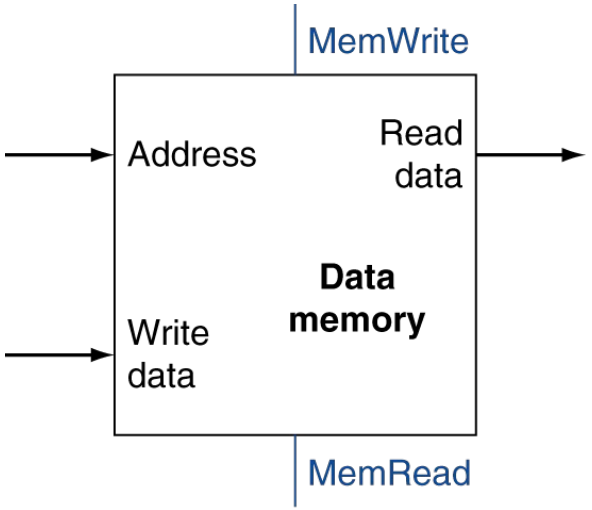
a. Registers

b. ALU

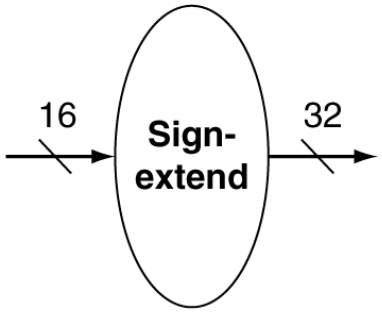
Load/Store Instructions



- Read register operands
- Calculate address using 16-bit offset
  - Use ALU, but sign-extend offset
- Load: Read memory and update register

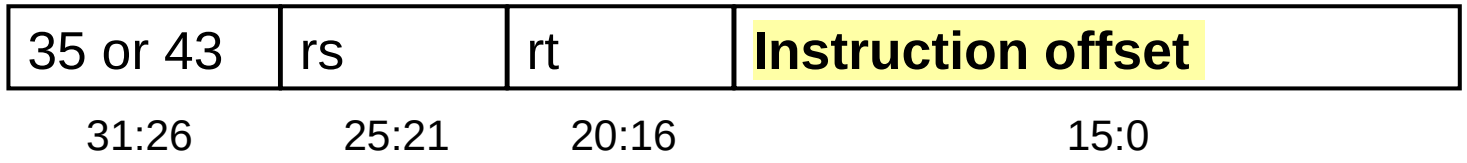


a. Data memory unit

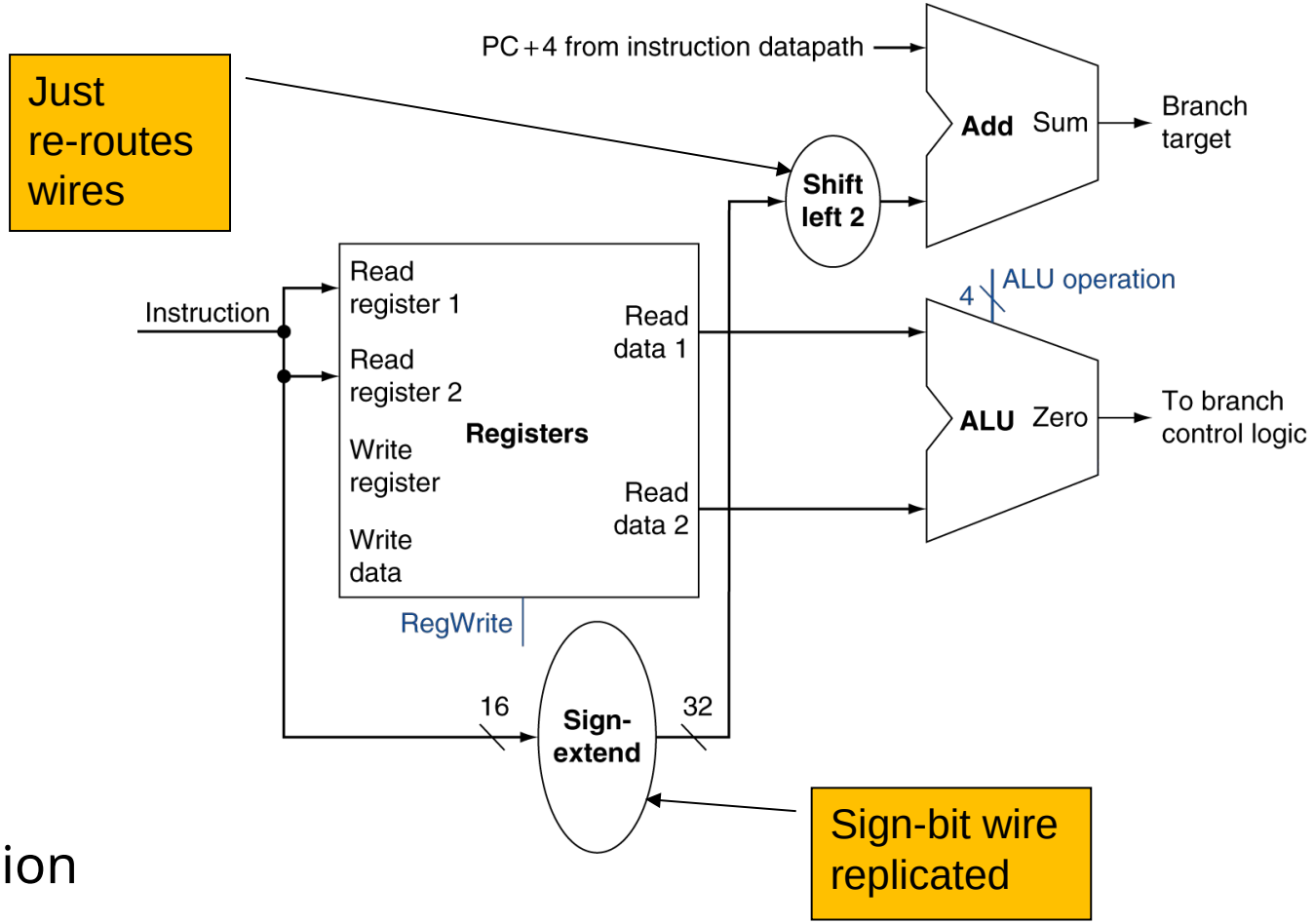


b. Sign extension unit

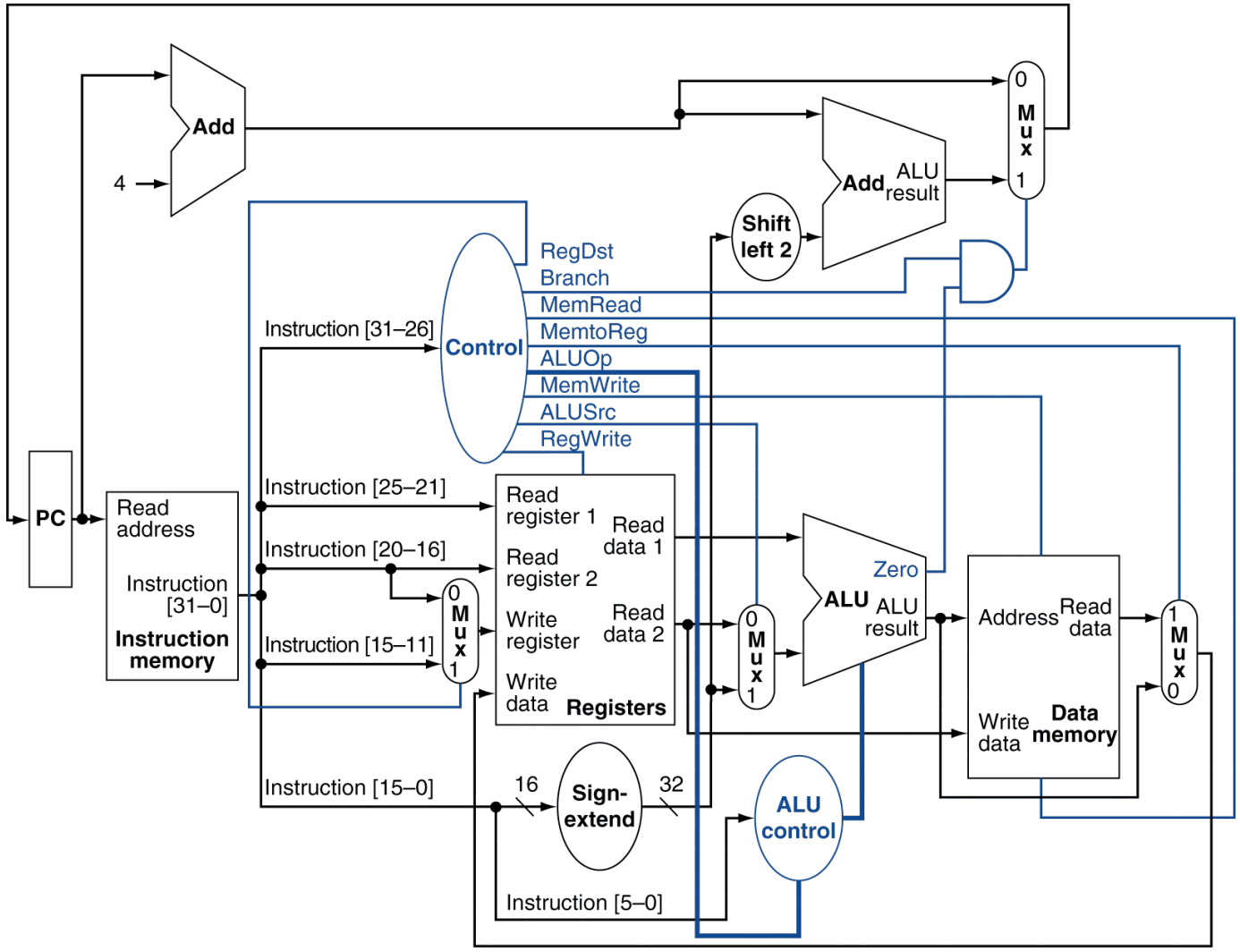
Branch Instructions



- Read register operands
- Compare operands
  - Use ALU, ***subtract*** and check Zero output
- Calculate target address
  1. Sign-extend displacement
  2. Shift left 2 places (word displacement)
  3. Add to PC + 4
     - Already calculated by instruction fetch



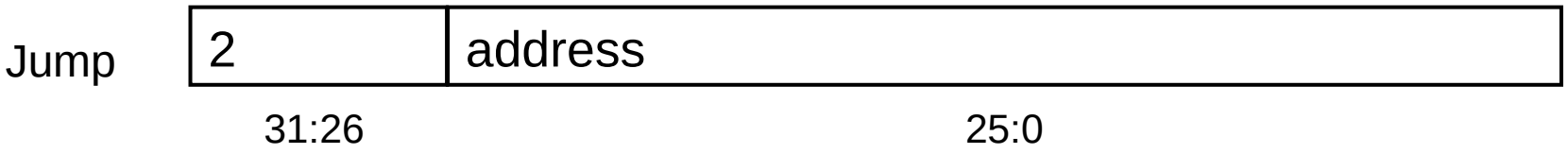
Integrated Datapath With Control



What is the role of every multiplexer in the figure?

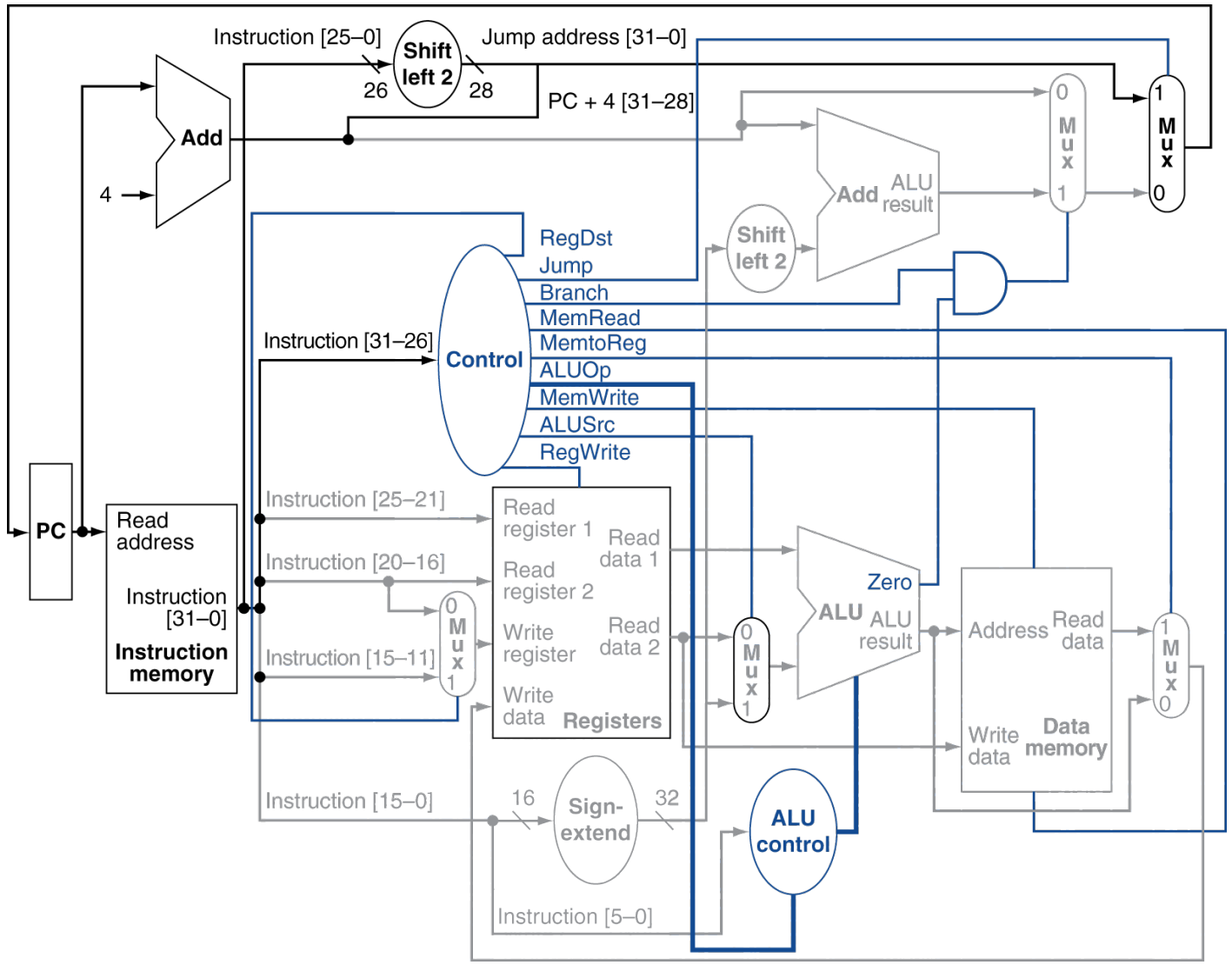
Control: the circuit that manages instruction fetch-execution by decoding instruction bits into relevant control signals

Implementing Jumps



- Jump uses word address
- Update PC with concatenation of
  - Top 4 bits of old PC
  - 26-bit jump address
  - 00
- Need an extra control signal decoded from opcode
  - Called a jump control
  - Asserted (set) only when opcode has the value of 2

Datapath With Jumps Added

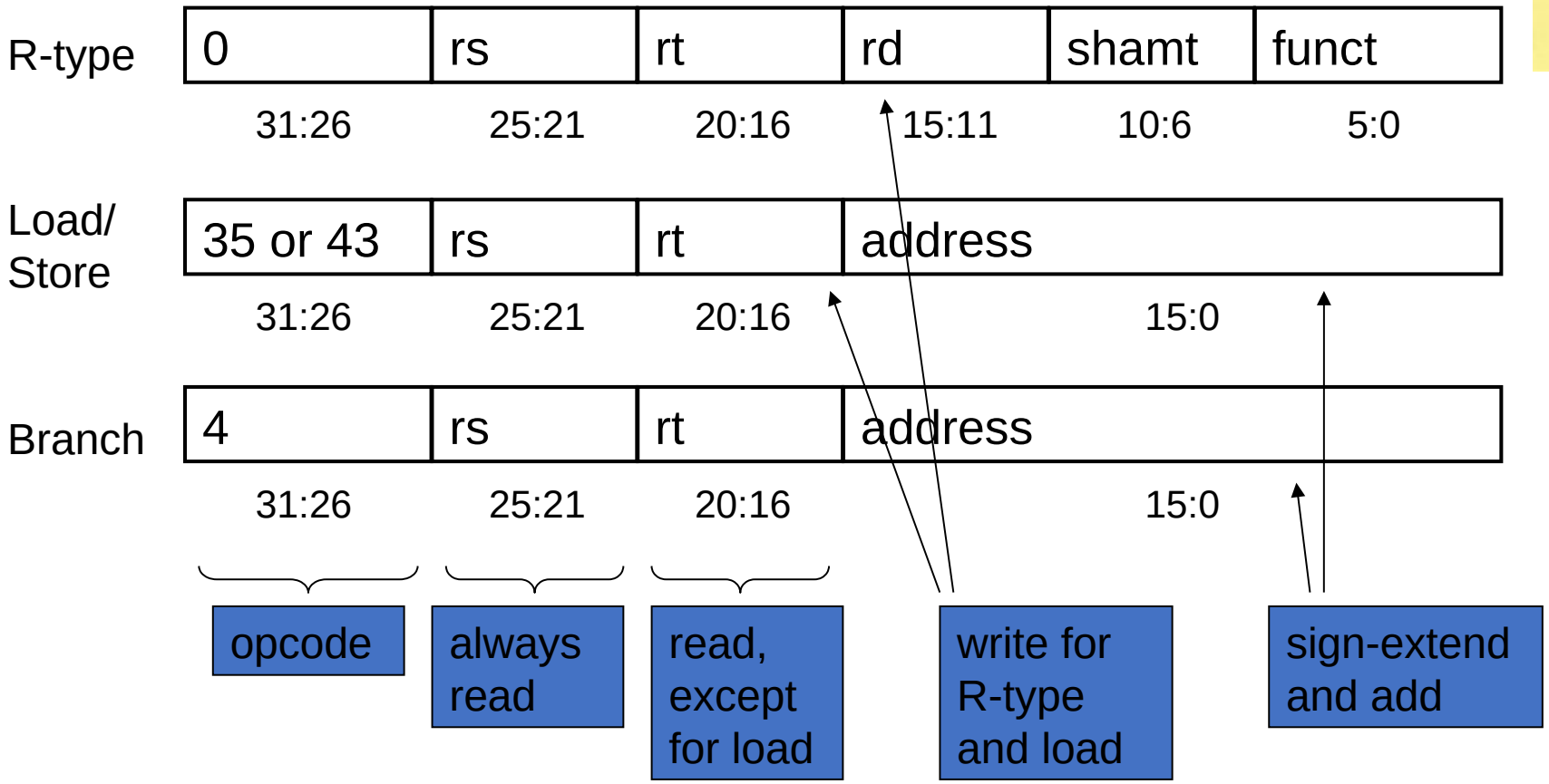


MIPS
Control

Control signals are derived from instruction

The Main Control Unit

Simplicity favors regularity



ALU Control

ALU is used for various instructions

- Load/Store: F = add
- Branch: F = subtract
- R-type: F depends on funct field

ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set-on-less-than
1100	NOR

Assume 2-bit ALUOp derived from opcode

- Combinational logic derives ALU control

opcode	ALUOp	Operation	funct	ALU function	ALU control
lw	00	load word	XXXXXX	add	0010
sw	00	store word	XXXXXX	add	0010
beq	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
		subtract	100010	subtract	0110
		AND	100100	AND	0000
		OR	100101	OR	0001
		set-on-less-than	101010	set-on-less-than	0111

Performance Issues

- Different instructions needs different execution times, why?
- **How long should be the clock period of a single clock cycle processor?**
  - ***Long enough to cover the time needed for the execution of the longest instruction (Critical path)***
  - What is MIPS critical path? load instruction
  - Load instruction uses all main stages (Instruction memory, register file, ALU, data memory, register file)
- Performance issues [Single Clock Cycle]
- All instructions operates at the speed of the slowest one (Violates Making the common case fast)
  - Not feasible to vary period for different instructions
- A solution to improve performance is by ***pipelining***

The Pipelined Processor

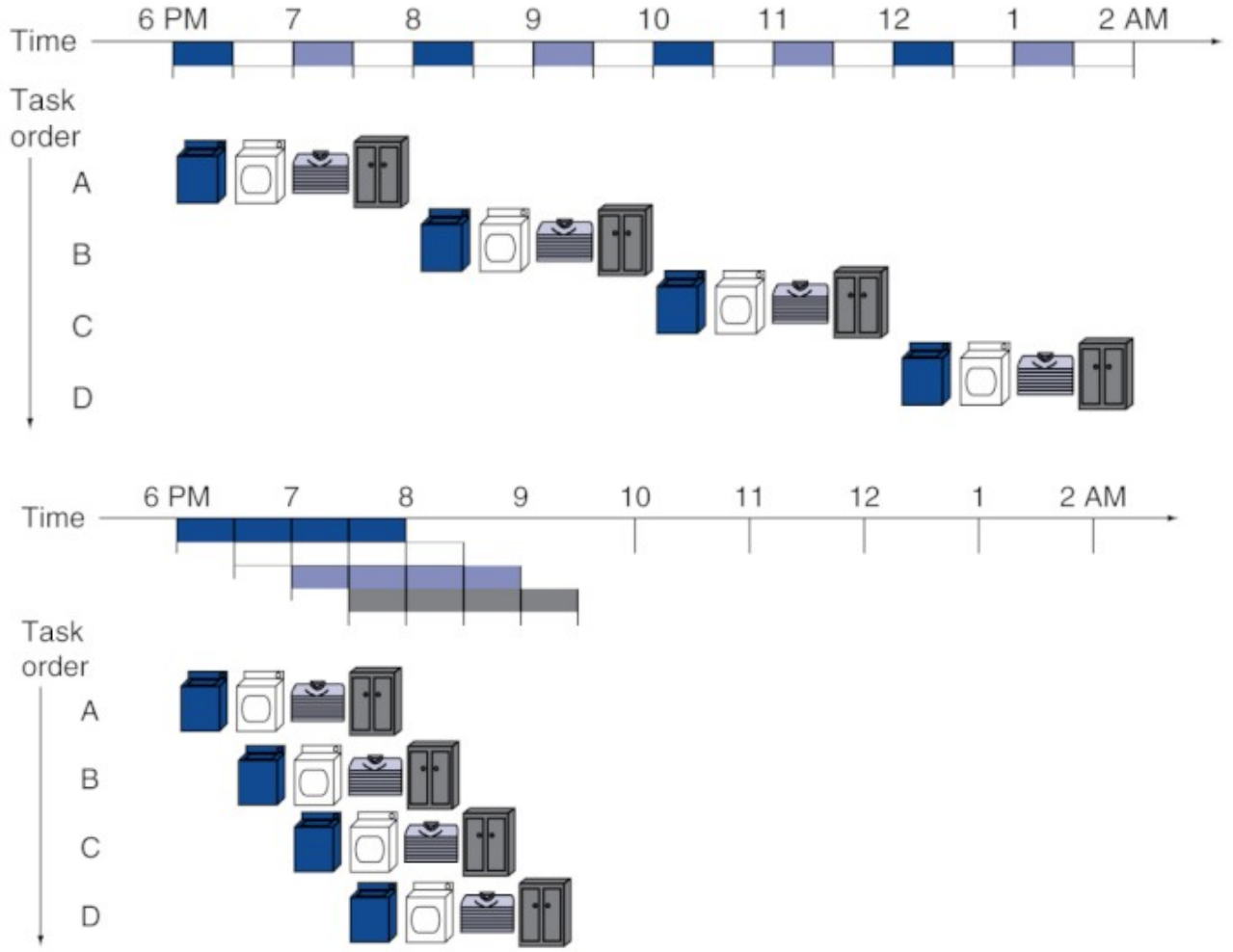
Sections 4.4, 4.5

Pipelining Analogy

- Pipelined laundry: overlapping execution
  - Parallelism improves performance

- Four loads:
  - Sequential execution

- Non-stop:
  - Speedup achieved with pipelining



MIPS Pipeline

- Five stages, one step per stage
  1. **IF**: Instruction fetch from memory
  2. **ID**: Instruction decode & register read
  3. **EX**: Execute operation or calculate address
  4. **MEM**: Access memory operand [not always]
  5. **WB**: Write result back to register [not always]

Pipelining and MIPS ISA Design

- MIPS ISA designed for pipelining
  - One instruction size [32-bits]
    - Easier to fetch in one cycle
    - c.f. x86: 1- to 17-byte instructions
  - Few and regular instruction formats
    - Can decode and read registers in one step
  - Simple Load/store addressing
    - Can calculate address in 3rd stage, access memory in 4th stage
  - Alignment of memory operands

Pipeline Performance

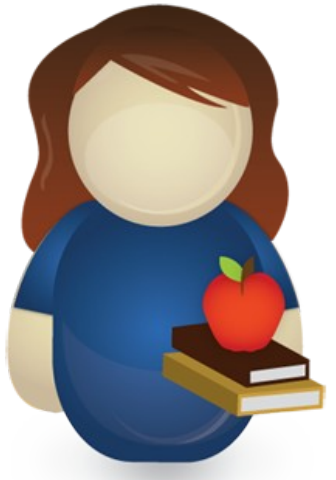
- Assume time for stages is
  - 100ps for register read or write
  - 200ps for other stages
- Table below shows the **single-cycle datapath** instruction execution time

Instr	Instr fetch	Reg read	ALU op	Memory access	Reg write	Total time
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

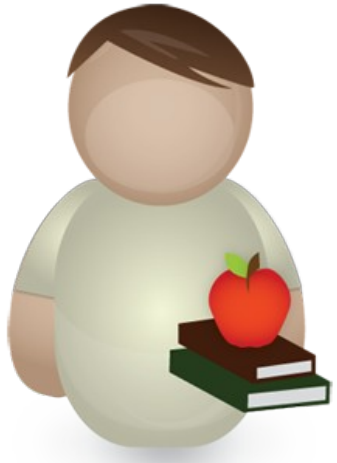
What is the clock period of this single-clock-cycle processor?

Critical path is the instruction that has the longest execution time --> clock cycle = 800ps
Load instruction in MIPS

Hi Jack, I missed the last lecture.
What did Ahmed teach?



Ahmed taught us how to do our laundry!



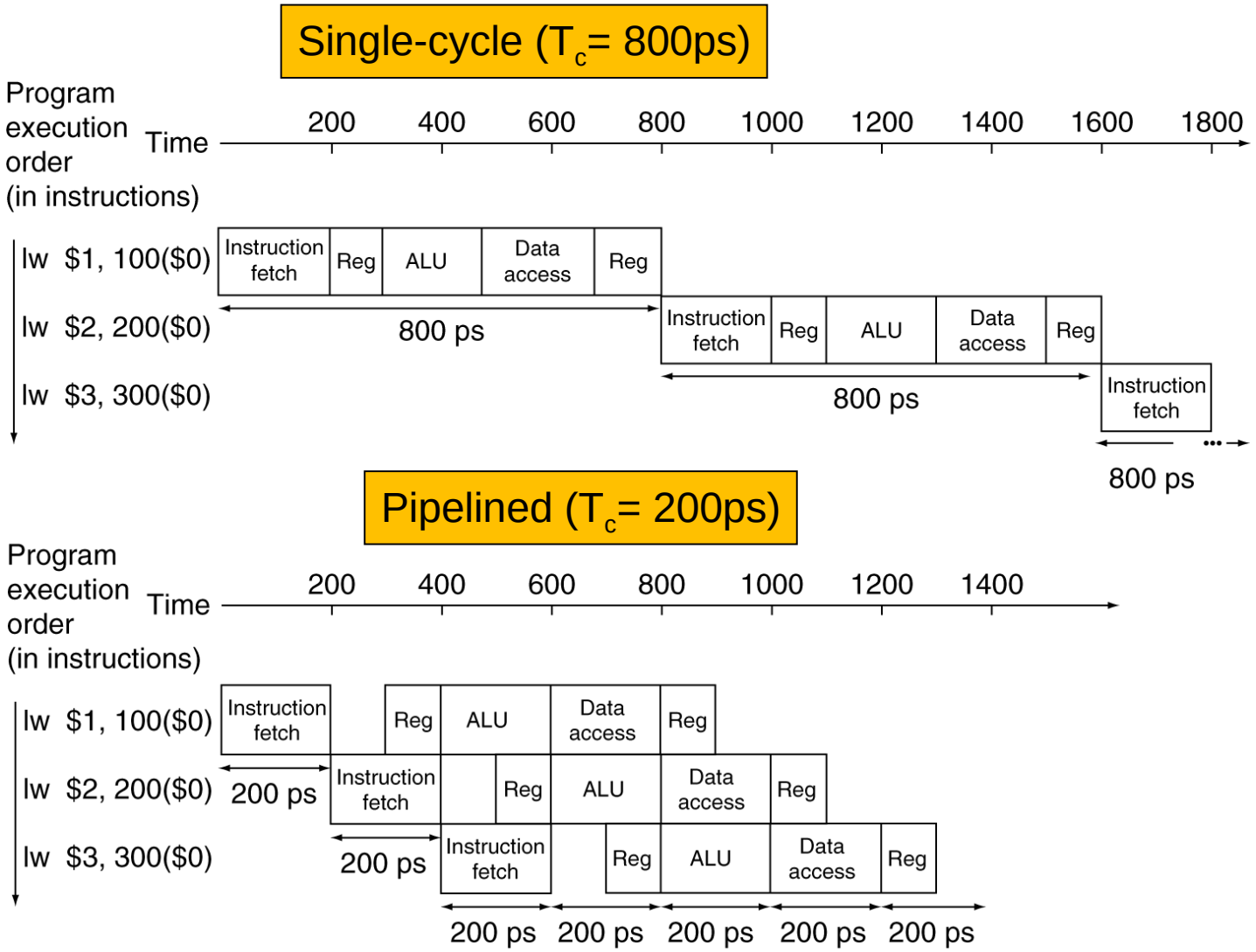
Question

Assume that individual stages of the MIPS processor datapath have the following latencies

IF	ID	EX	MEM	WB
250ps	350ps	150ps	300ps	200ps

(a) What is the clock cycle time in a pipelined and non-pipelined processor?
(b) What is the total latency of an LW instruction in a pipelined and non-pipelined processor?

Pipeline Performance Comparison



What is the clock cycle of this pipelined processor?

General rule: ***The clock cycle of pipelined architectures has to be larger than the time needed by the longest stage***

Pipeline Speedup

- If all stages are balanced
  - i.e., all **N** pipeline stages take the same time **T** and we have **I** instructions
  - Single clock cycle execution time = I * (NT)
  - Pipelined procesor execution time = NT + (I-1)T
  - speedup factor = I*N*T / [(N + I -1)T] ~N

- If not balanced, speedup is less
- Speedup due to increased throughput
  - Instruction Latency (time for each instruction) does not **decrease**

So, Pipelining is Good, right?