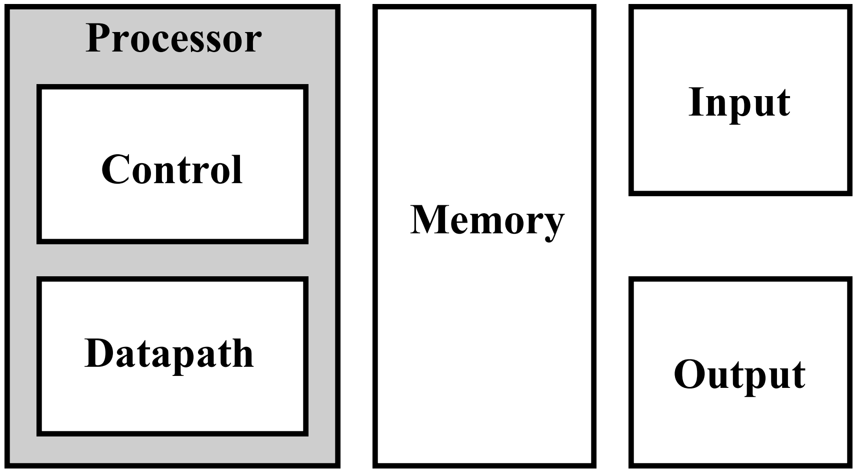
Single-Cycle CPU
Datapath Design

"The Do-It-Yourself CPU Kit"

The Big Picture: Where are We Now?

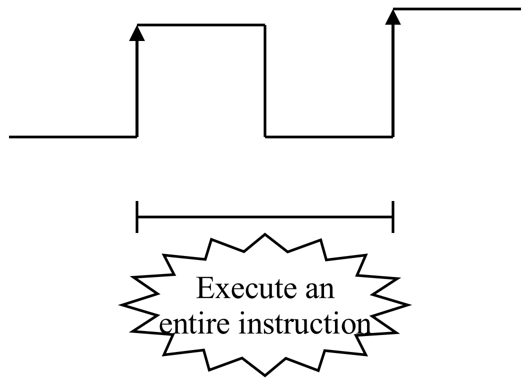
- The Five Classic Components of a Computer



- Today’s Topic: Datapath Design, then Control Design

The Big Picture: The Performance Perspective

- Processor design (datapath and control) will determine:
  - Clock cycle time
  - Clock cycles per instruction
- Starting today:
  - Single cycle processor:
    - Advantage: One clock cycle per instruction
    - Disadvantage: long cycle time



- ET = Insts * CPI * Cycle Time

The Processor: Datapath & Control

- We're ready to look at an implementation of the MIPS simplified to contain only:
  - memory-reference instructions: `lw, sw`
  - arithmetic-logical instructions: `add, sub, and, or, slt`
  - control flow instructions: `beq`
- Generic Implementation:
  - use the program counter (PC) to supply instruction address
  - get the instruction from memory
  - read registers
  - use the instruction to decide exactly what to do
- All instructions use the ALU after reading the registers
  memory-reference? arithmetic? control flow?

Review: The MIPS Instruction Formats

- All MIPS instructions are 32 bits long. The three instruction formats:

R-type

31 26	21	16	11	6	0
op	**rs**	**rt**	**rd**	**shamt**	**funct**
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

I-type

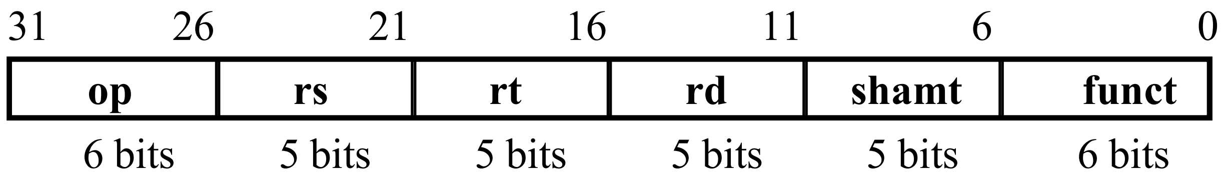
31 26	21	16	0
op	**rs**	**rt**	**immediate**
6 bits	5 bits	5 bits	16 bits

J-type

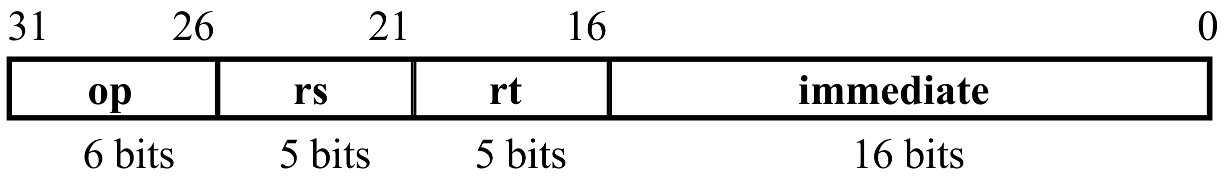
31 26	0
op	**target address**
6 bits	26 bits

The MIPS Subset

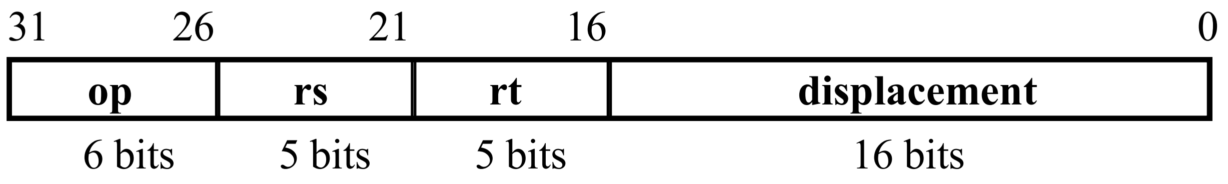
- R-type
  - *add rd, rs, rt*
  - sub, and, or, slt



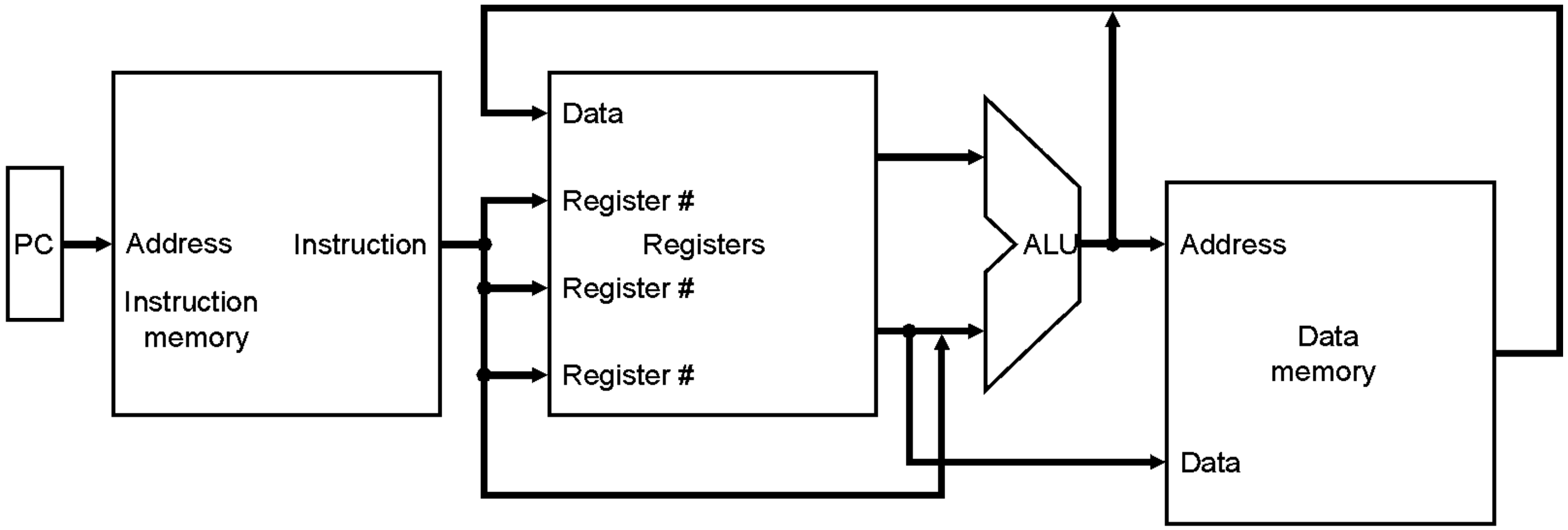
- LOAD and STORE
  - lw rt, rs, imm16
  - sw rt, rs, imm16



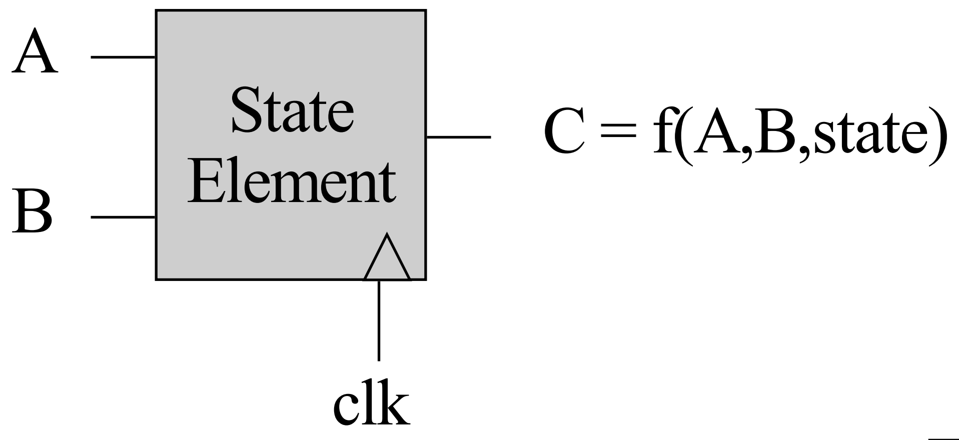
- BRANCH:
  - beq rs, rt, imm16

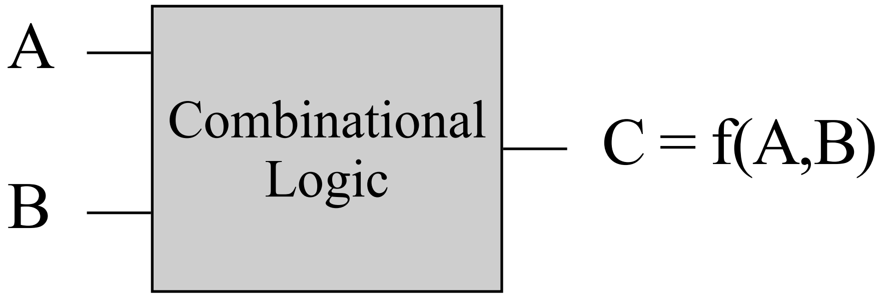


Where We're Going – The High-level View

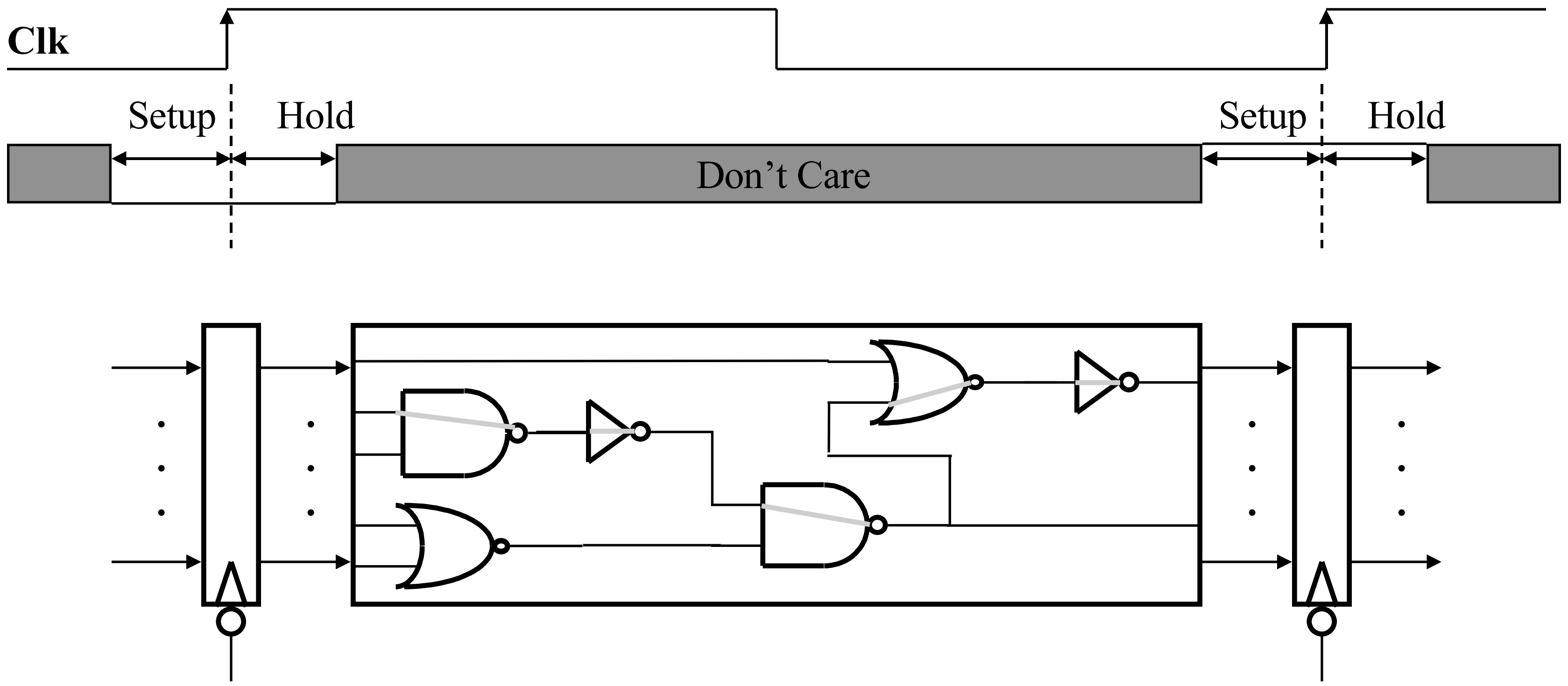


Review: Two Types of Logic Components





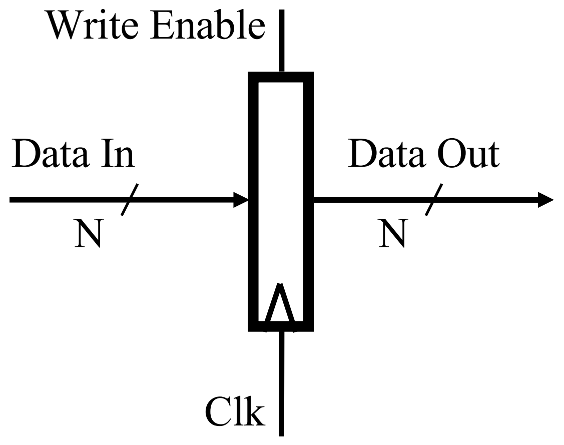
Clocking Methodology



- All storage elements are clocked by the same clock edge

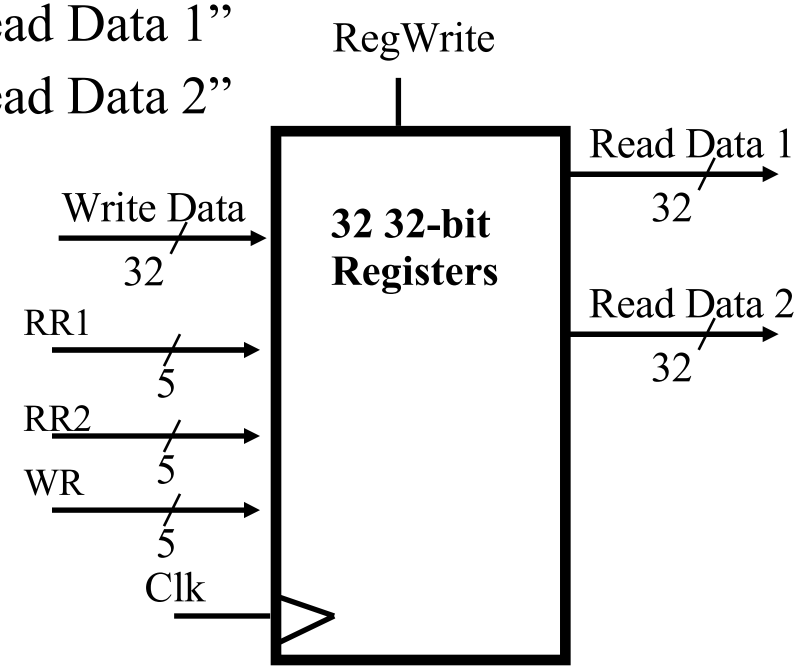
Storage Element: Register

- Register
  - Similar to the D Flip Flop except
    - N-bit input and output
    - Write Enable input
  - Write Enable:
    - 0: Data Out will not change
    - 1: Data Out will become Data In (on the clock edge)

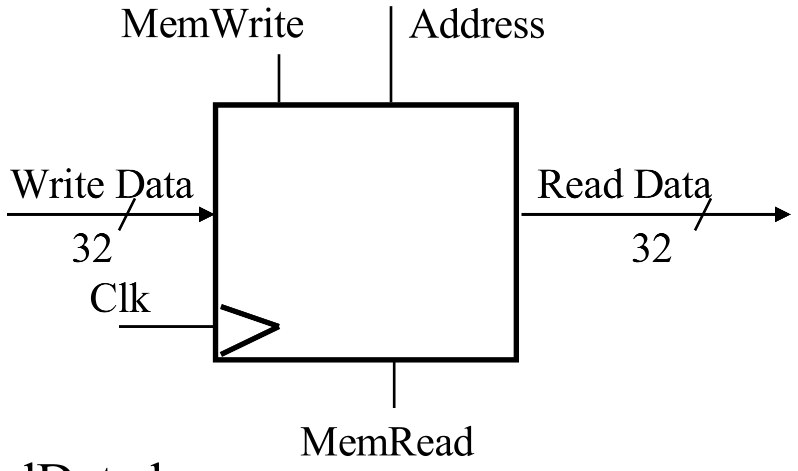


Storage Element: Register File

- Register File consists of (32) registers:
  - Two 32-bit output buses:
  - One 32-bit input bus: busW
- Register is selected by:
  - RR1 selects the register to put on bus "Read Data 1"
  - RR2 selects the register to put on bus "Read Data 2"
  - WR selects the register to be written via WriteData when RegWrite is 1
- Clock input (CLK)



Storage Element: Memory



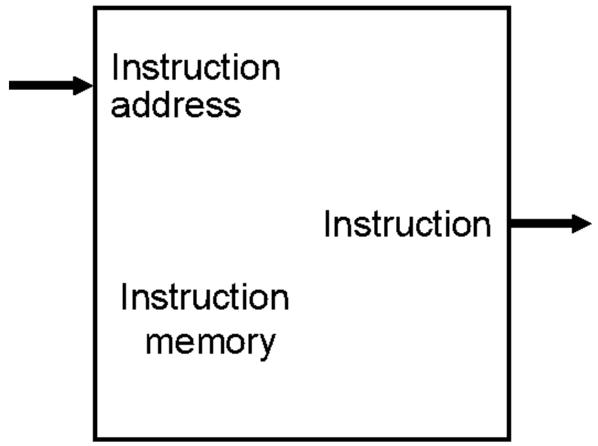
- Memory
  - Two input buses: WriteData, Address
  - One output bus: ReadData
- Memory word is selected by:
  - Address selects the word to put on ReadData bus
  - MemWrite = 1: address selects the memory word to be written via the WriteData bus
- Clock input (CLK)
  - The CLK input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block:
    - Address valid => ReadData valid after "access time."

Register Transfer Language (RTL)

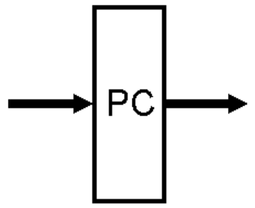
- is a mechanism for describing the movement and manipulation of data between storage elements:

```
R[3] <- R[5] + R[7]
PC <- PC + 4 + R[5]
R[rd] <- R[rs] + R[rt]
R[rt] <- Mem[R[rs] + immed]
```

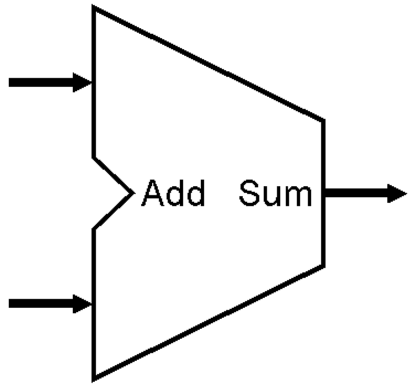
Instruction Fetch and Program Counter Management



a. Instruction memory



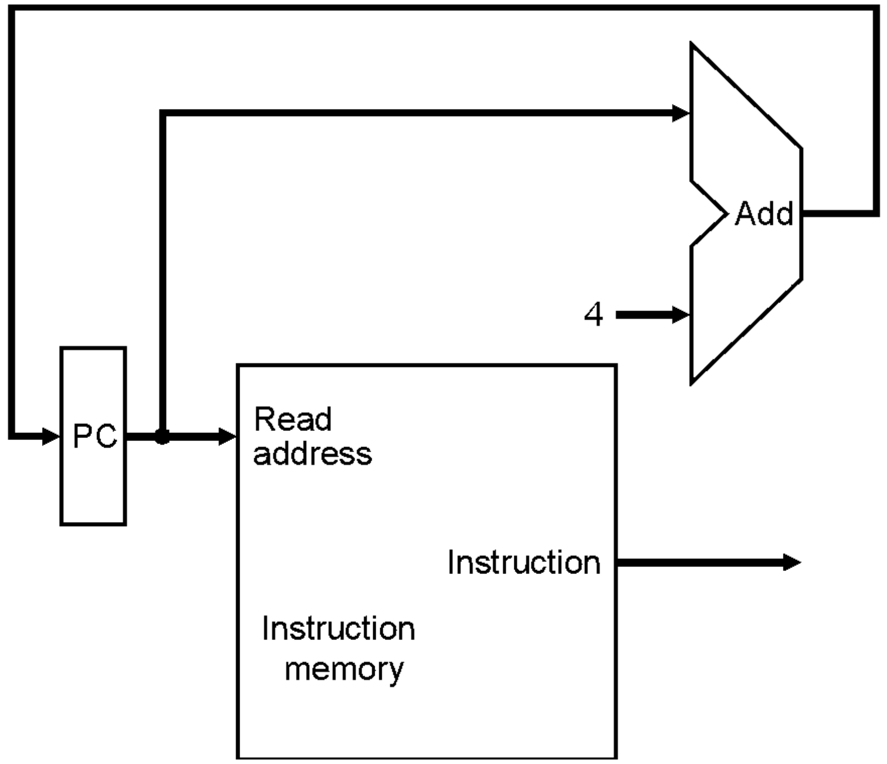
b. Program counter



c. Adder

Overview of the Instruction Fetch Unit

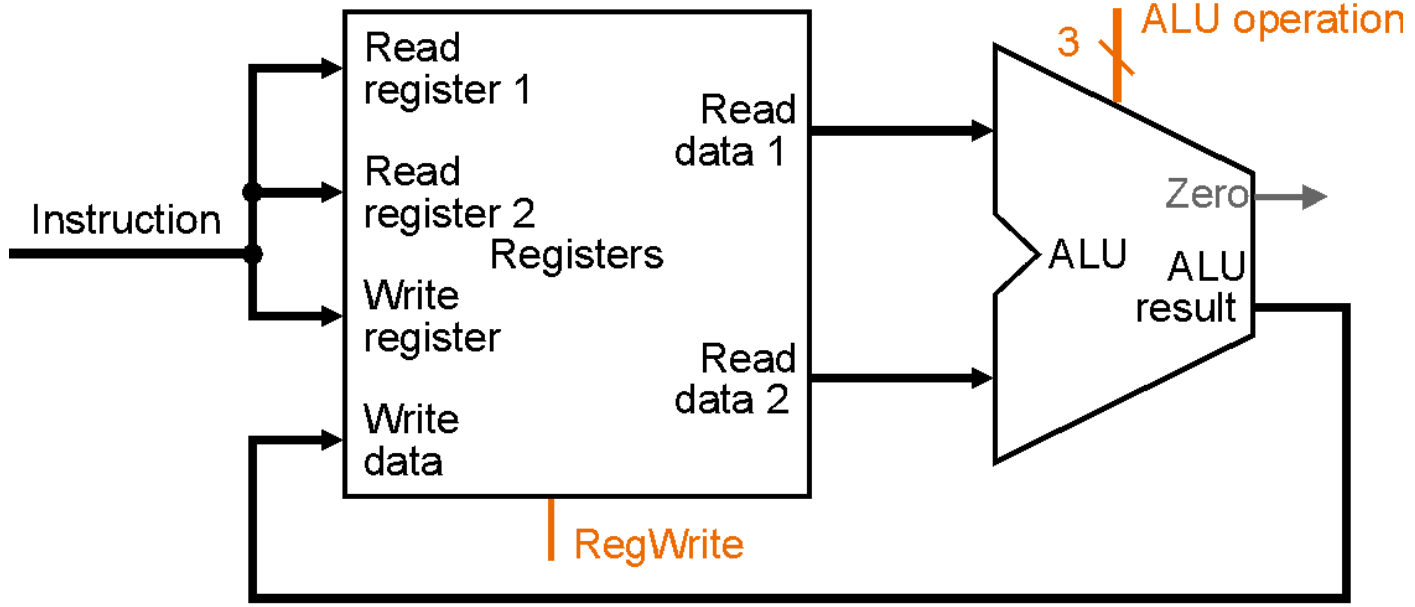
- The common RTL operations
  - Fetch the Instruction: inst <- mem[PC]
  - Update the program counter:
    - Sequential Code: PC <- PC + 4
    - Branch and Jump   PC <- "something else"



Datapath for Register-Register Operations

- R[rd] <- R[rs] op R[rt]   Example: *add   rd, rs, rt*
  - RR1, RR2, and WR comes from instruction's rs, rt, and rd fields
  - *ALUoperation* and *RegWrite*: control logic after decoding instruction

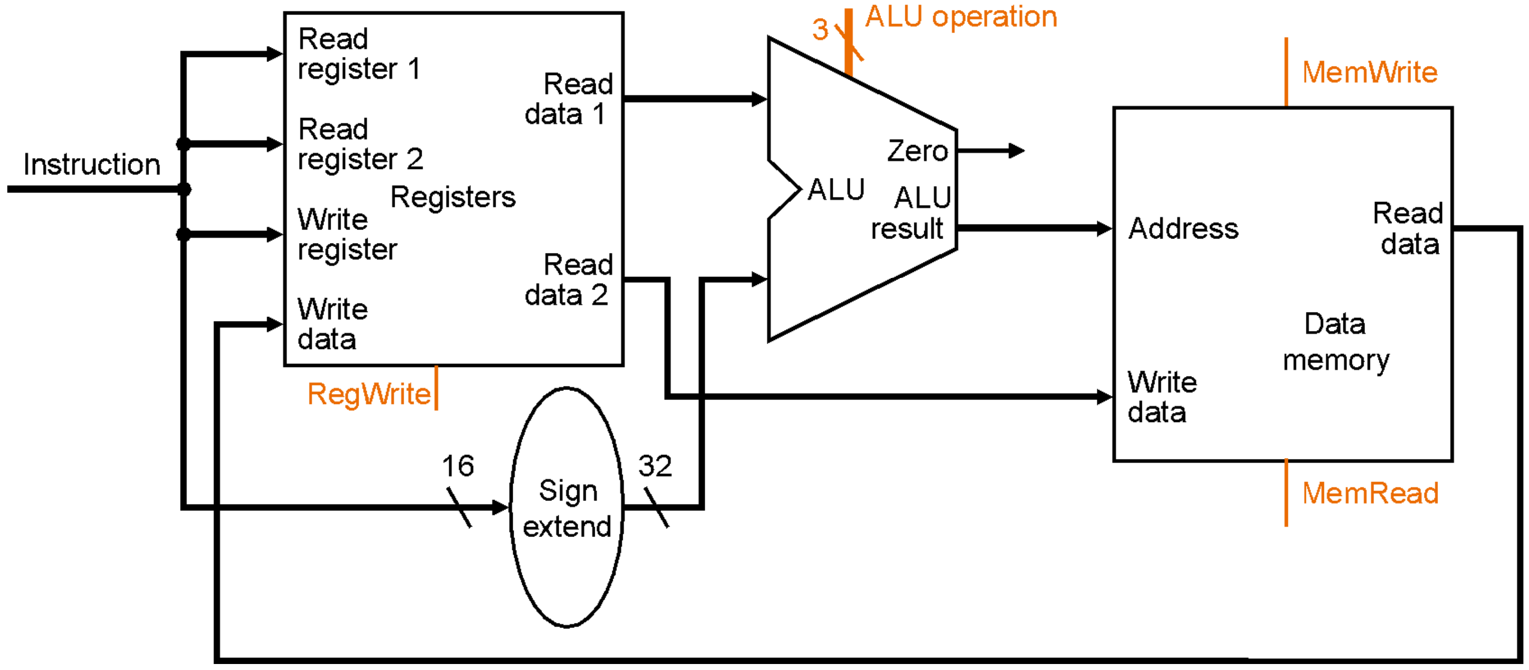
31 … 26	26 … 21	21 … 16	16 … 11	11 … 6	6 … 0
op	**rs**	**rt**	**rd**	**shamt**	**funct**
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits



Datapath for Load Operations

R[rt] <- Mem[R[rs] + SignExt[imm16]] Example: *lw rt, rs, imm16*

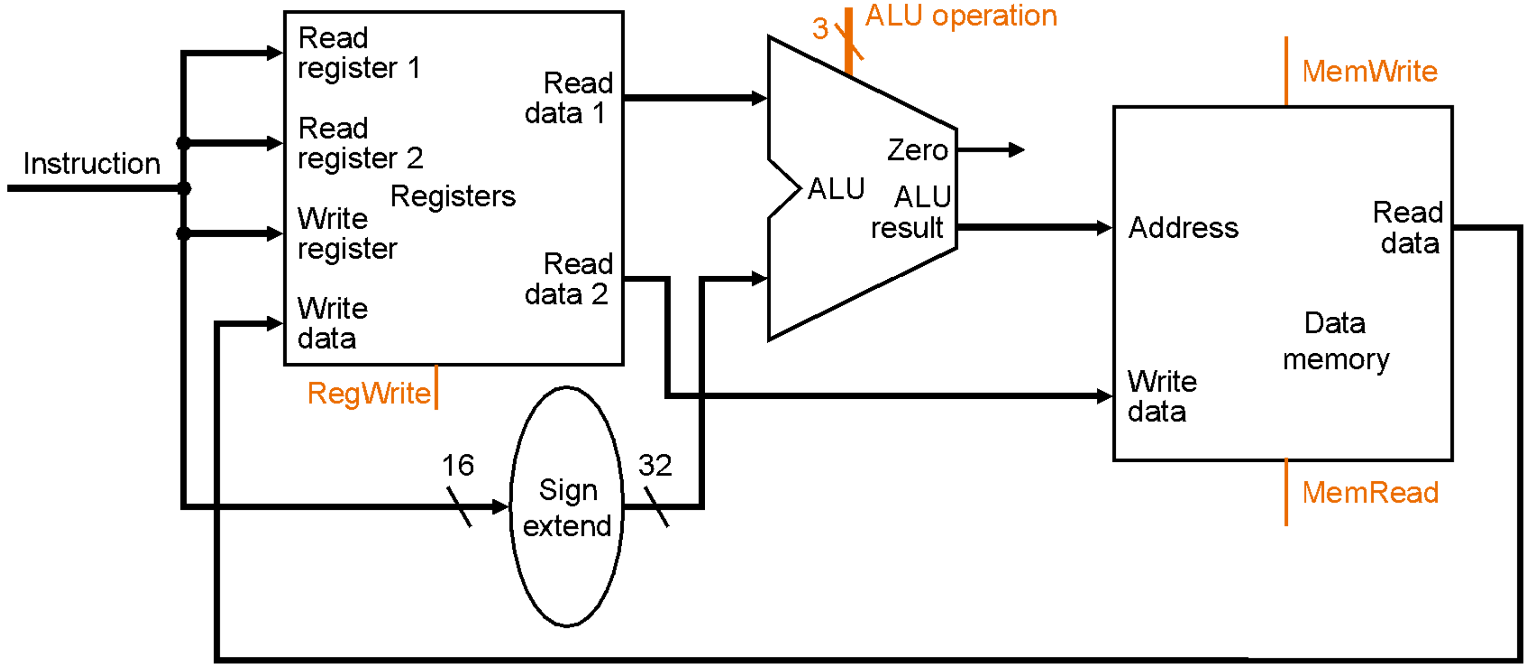
31–26	25–21	20–16	15–0
op	**rs**	**rt**	**immediate**
6 bits	5 bits	5 bits	16 bits



Datapath for Store Operations

Mem[R[rs] + SignExt[imm16]] <- R[rt]   Example: *sw   rt, rs, imm16*

31 26	21	16	0
op	**rs**	**rt**	**immediate**
6 bits	5 bits	5 bits	16 bits



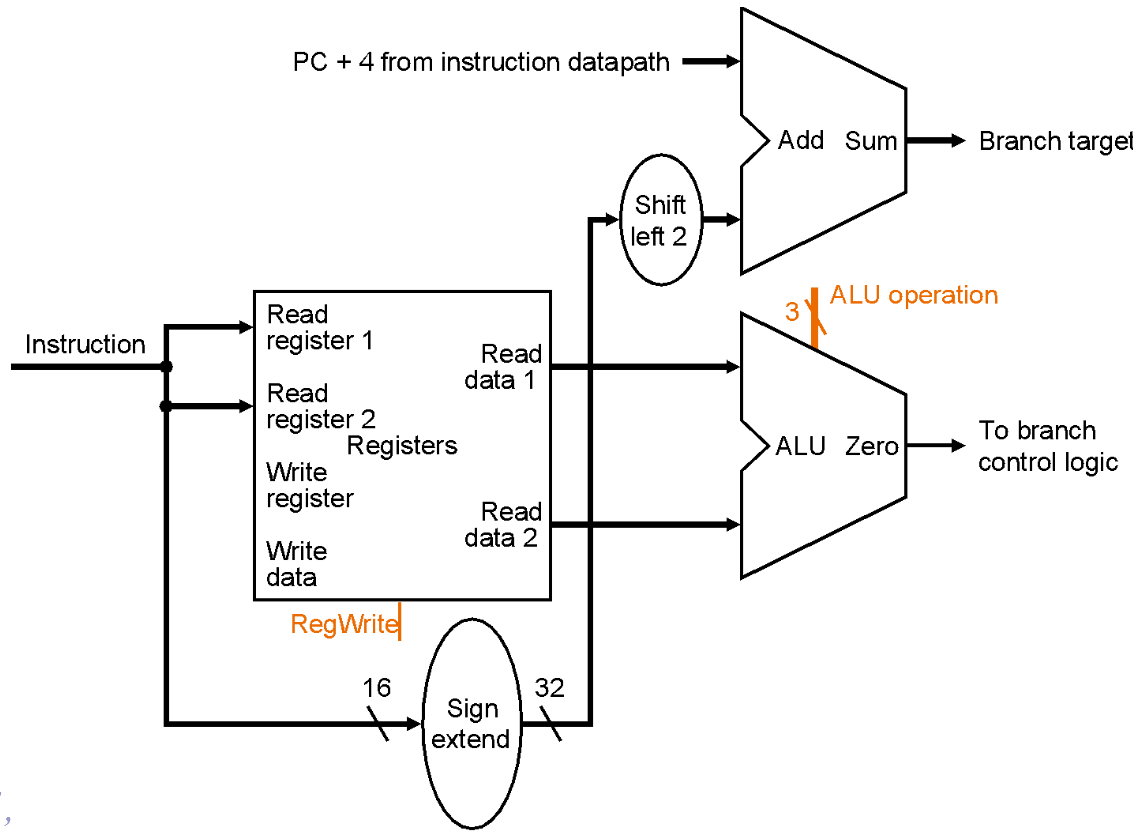
Datapath for Branch Operations

Z <- (rs == rt); if Z, PC = PC+4+imm16; else PC = PC+4

beq rs, rt, imm16

31 26 21 16 0

op	**rs**	**rt**	**immediate**
6 bits	5 bits	5 bits	16 bits

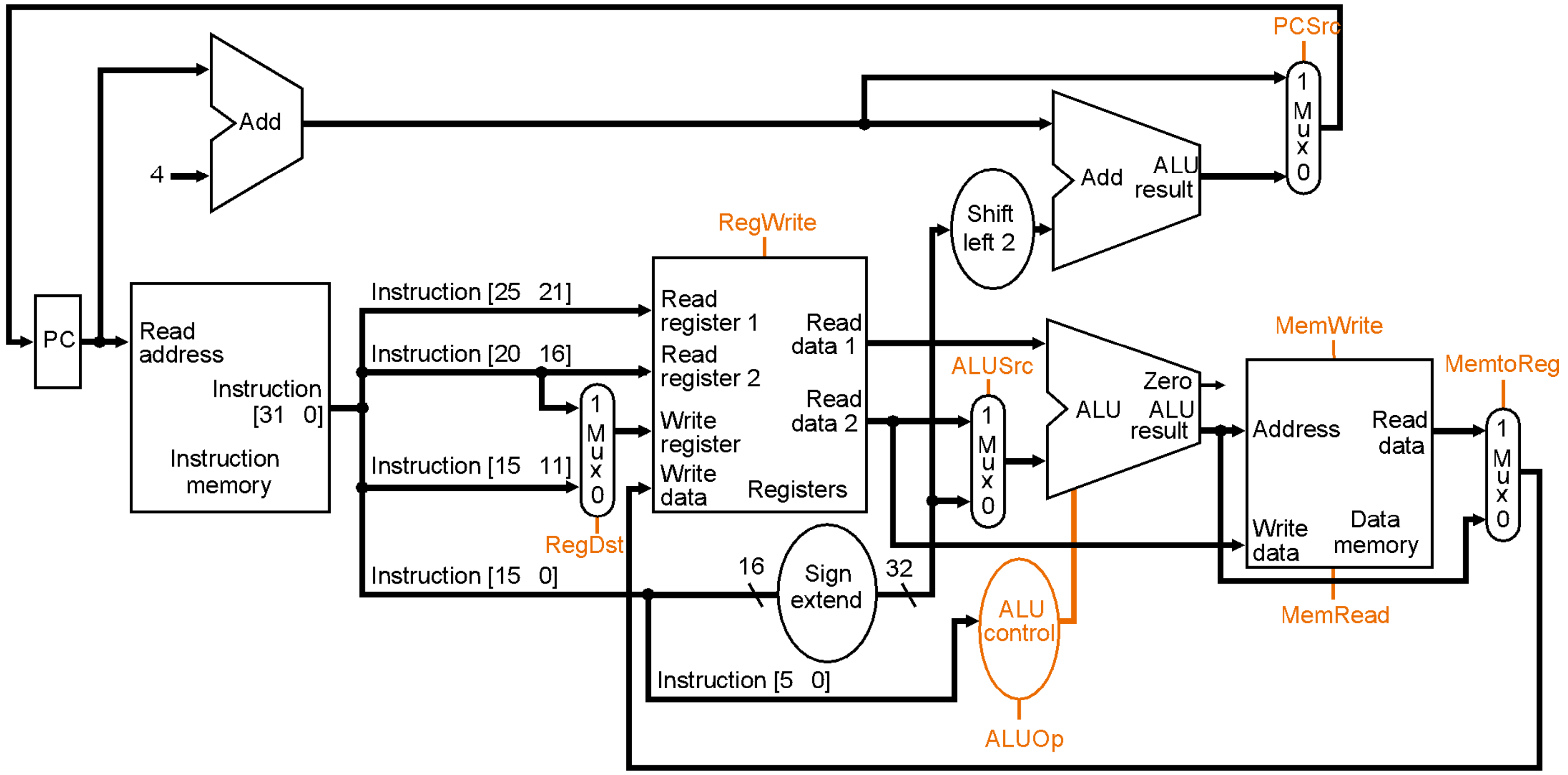


Binary Arithmetic for the Next Address

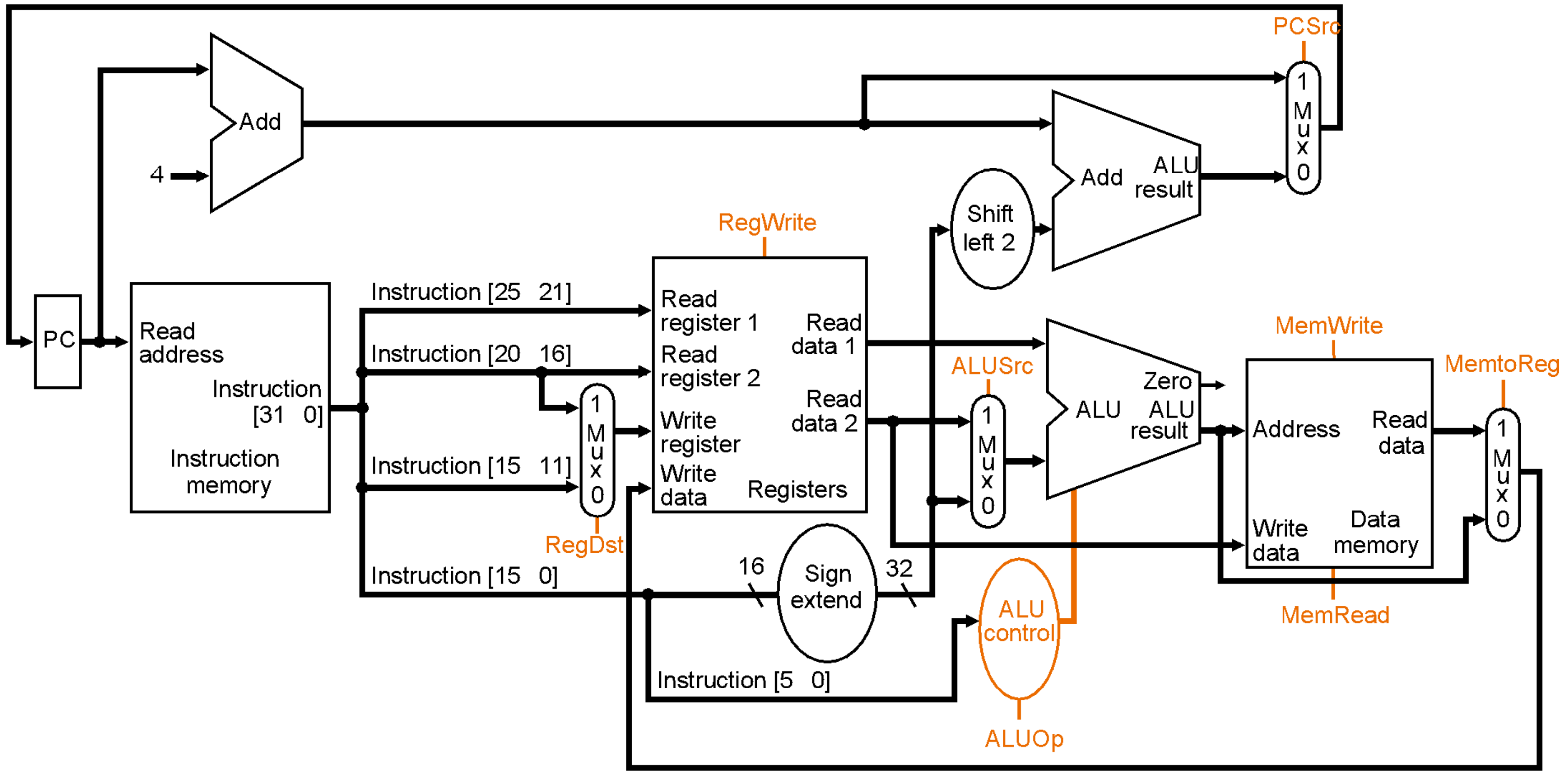
- In theory, the PC is a 32-bit byte address into the instruction memory:
  - Sequential operation: PC<31:0> = PC<31:0> + 4
  - Branch operation: PC<31:0> = PC<31:0> + 4 + SignExt[Imm16] * 4
- The magic number "4" always comes up because:
  - The 32-bit PC is a byte address
  - And all our instructions are 4 bytes (32 bits) long
  - The 2 LSBs of the 32-bit PC are always zeros
  - There is no reason to have hardware to keep the 2 LSBs
- In practice, we can simplify the hardware by using a 30-bit PC<31:2>:
  - Sequential operation: PC<31:2> = PC<31:2> + 1
  - Branch operation: PC<31:2> = PC<31:2> + 1 + SignExt[Imm16]
  - In either case: Instruction Memory Address = PC<31:2> concat "00"

Putting it All Together: A Single Cycle Datapath

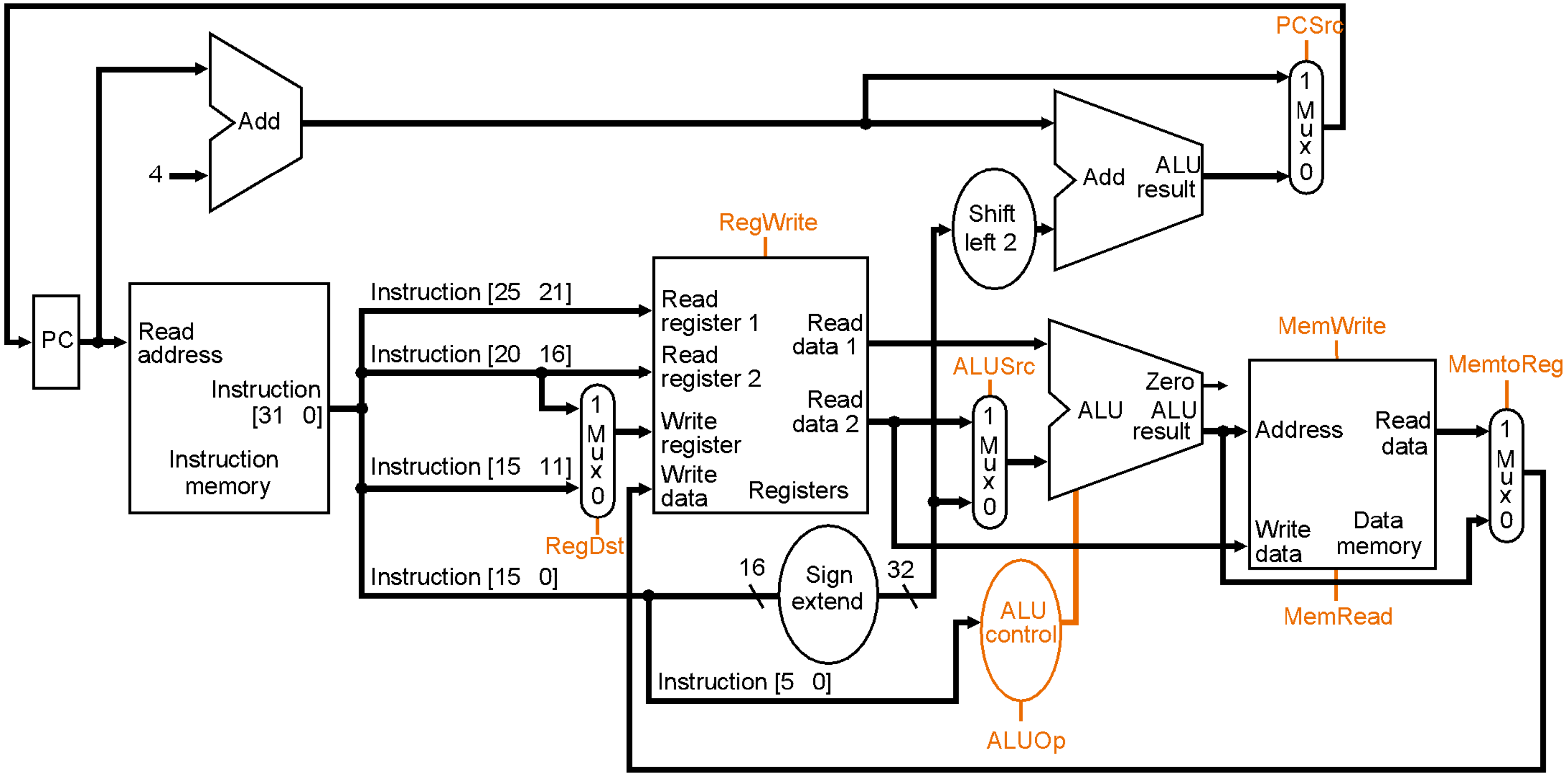
- We have everything except control signals



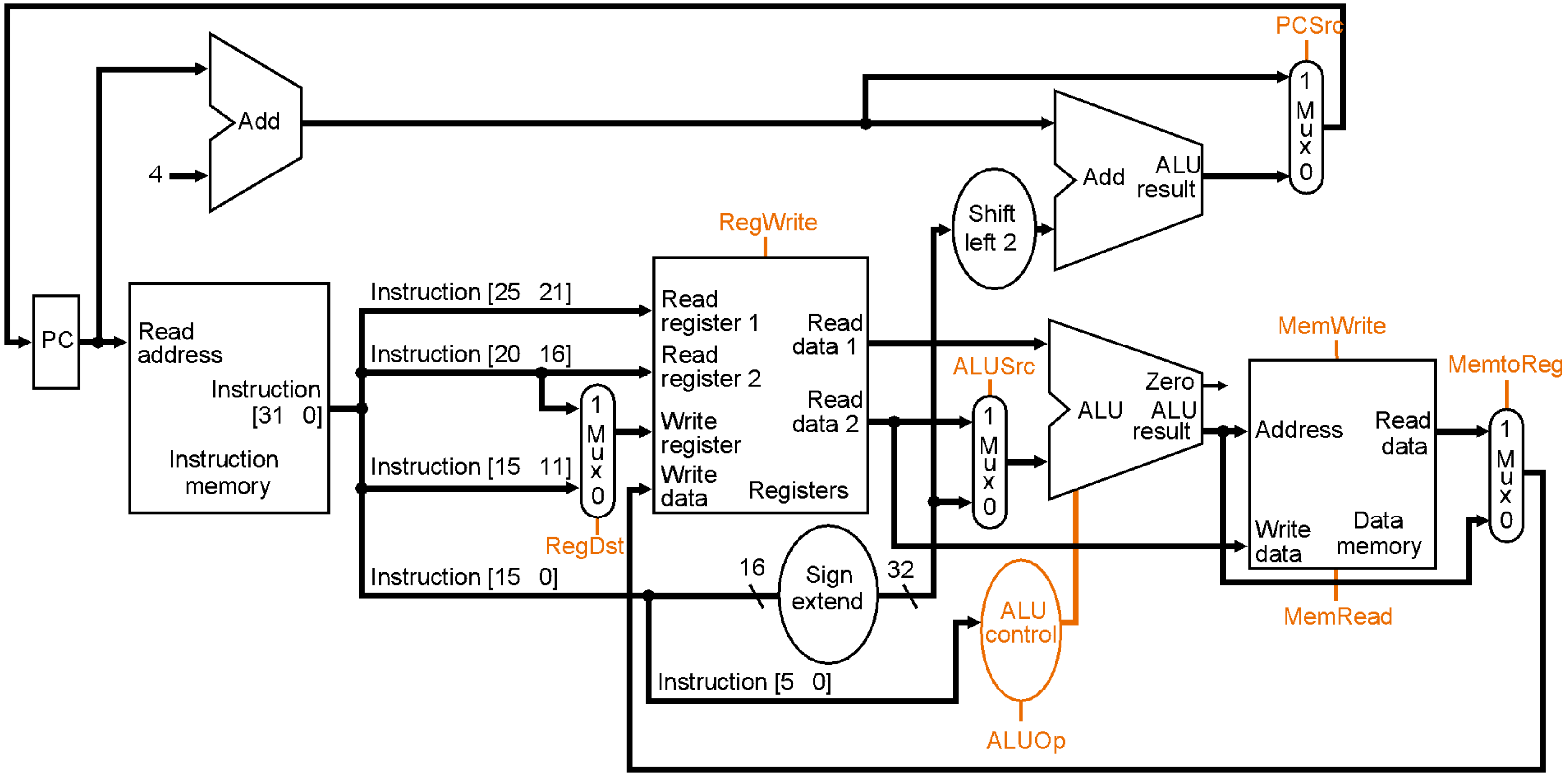
The R-Format (e.g. *add*) Datapath



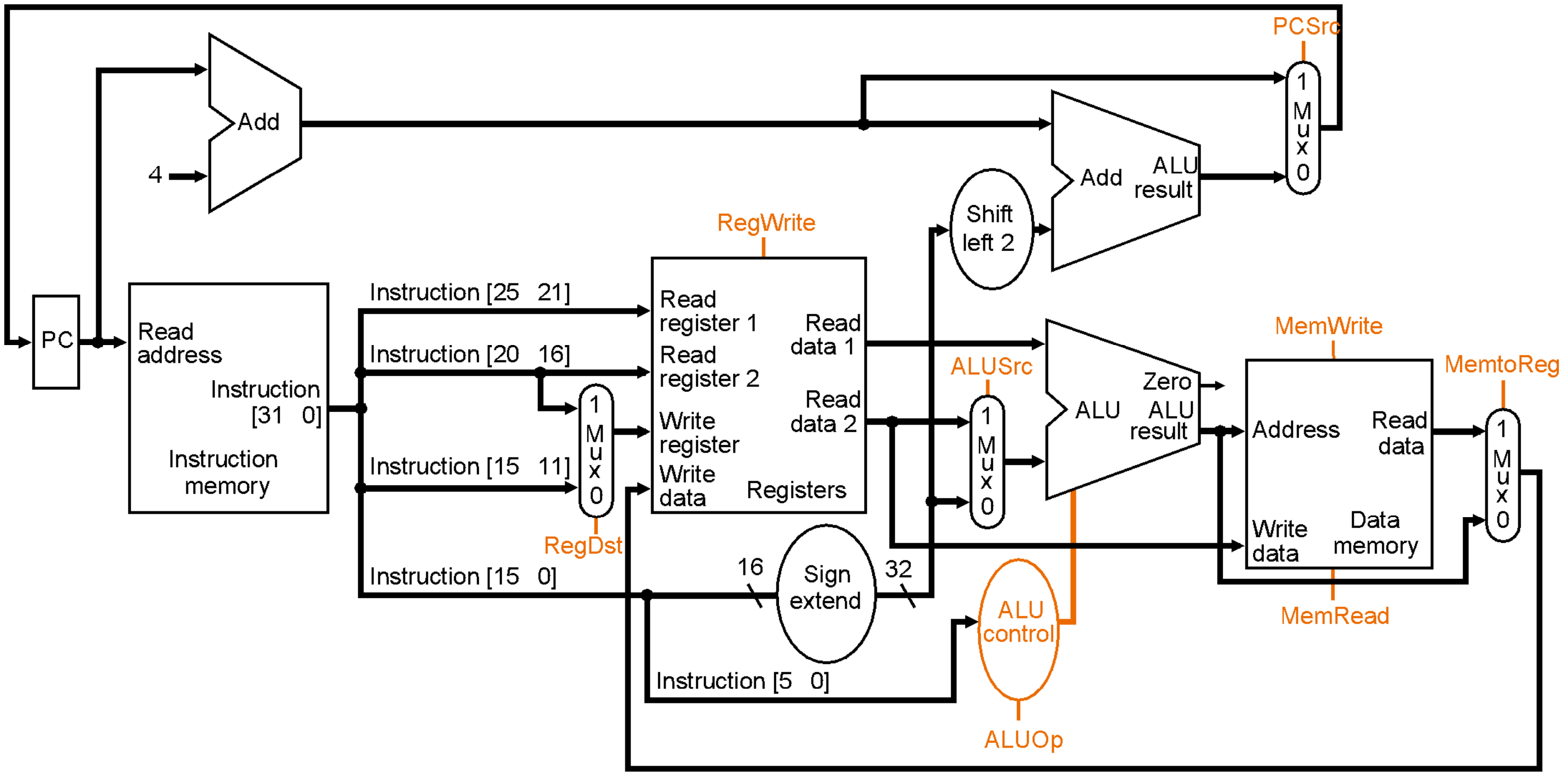
The Load Datapath



The store Datapath



The beq Datapath



Key Points

- CPU is just a collection of state and combinational logic
- We just designed a very rich processor, at least in terms of functionality
- Performance = Insts * CPI * Cycle Time
  - where does the single-cycle machine fit in?