# EEE304 – Digital Design with HDL (II)

## Lecture 6

Dr. Ming Xu

Dept of Electrical & Electronic Engineering

XJTLU



# In This Session

- The Processor



# Introduction

§ 4.1 Introduction

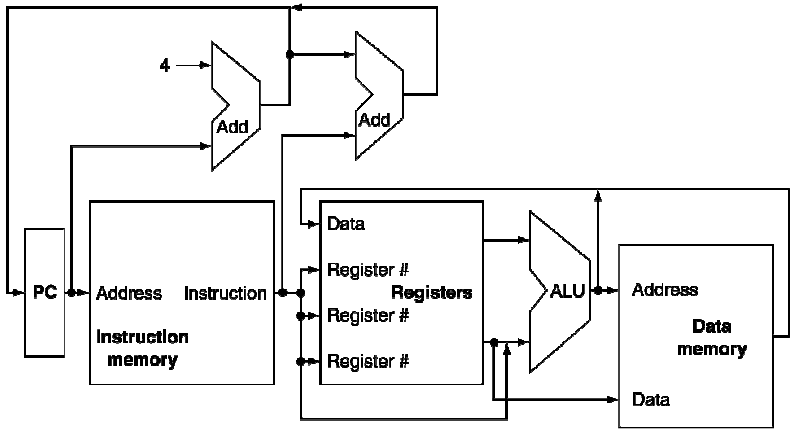
- We will examine two MIPS implementations
  - A simplified version
  - A more realistic pipelined version
- Simple subset, shows most aspects
  - Memory reference: `lw`, `sw`
  - Arithmetic/logical: `add`, `sub`, `and`, `or`, `slt`
  - Control transfer: `beq`, `j`



# Instruction Execution

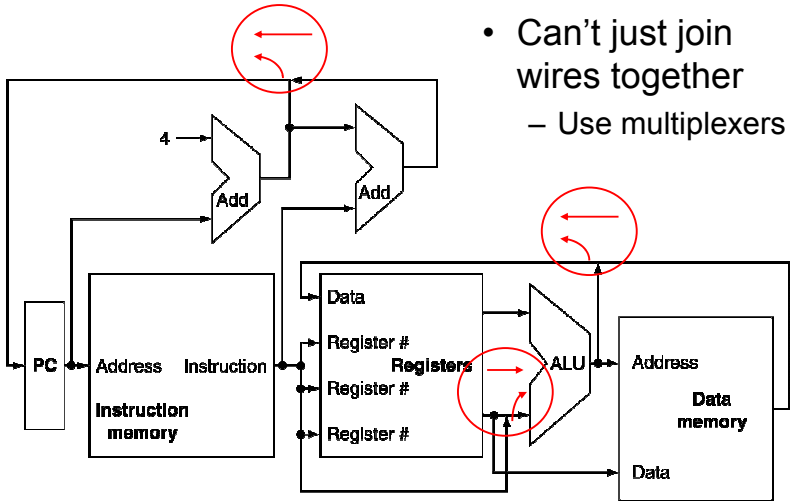
- PC → instruction memory, fetch instruction
- Register numbers → register file, read registers
- Depending on instruction class
  - Use ALU to calculate
    - Arithmetic result
    - Memory address for load/store
    - Branch target address
  - Access data memory for load/store
  - PC ← target address or PC + 4

# CPU Overview





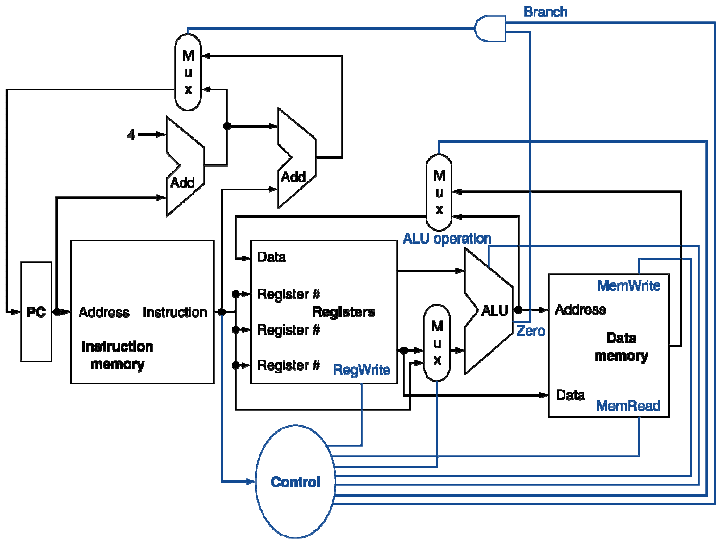

# Multiplexers




- Can't just join wires together
  - Use multiplexers



# Control



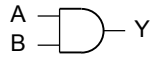



# Logic Design Basics

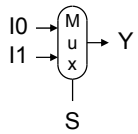
- Information encoded in binary
  - Low voltage = 0, High voltage = 1
  - One wire per bit
  - Multi-bit data encoded on multi-wire buses
- Combinational element
  - Operate on data
  - Output is a function of input
- State (sequential) elements
  - Store information

# Combinational Elements

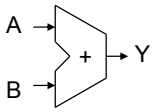
- AND-gate
  - Y = A & B



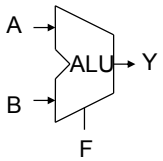

- Multiplexer
  - Y = S ? I1 : I0




- Adder
  - Y = A + B



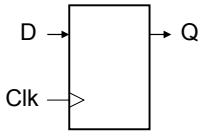

- Arithmetic/Logic Unit
  - Y = F(A, B)

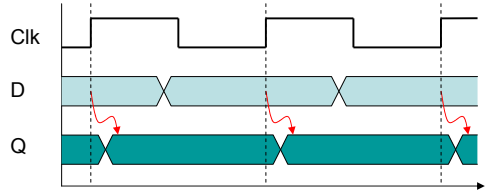





# Sequential Elements

- Register: stores data in a circuit
  - Uses a clock signal to determine when to update the stored value
  - Edge-triggered: update when Clk changes from 0 to 1

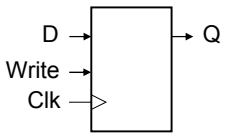


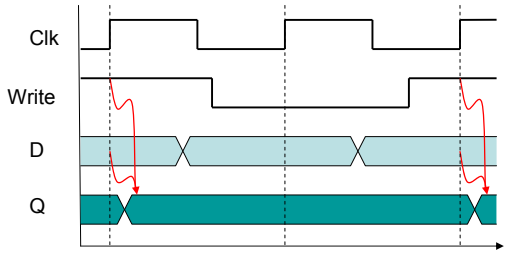






# Sequential Elements

- Register with write control
  - Only updates on clock edge when write control input is 1
  - Used when stored value is required later



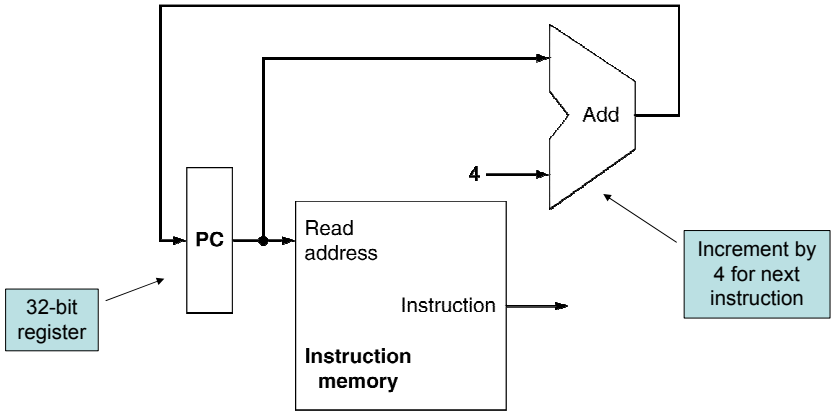



# Building a Datapath



- Datapath
  - Elements that process data and addresses in the CPU
    - Registers, ALUs, mux's, memories, …
- We will build a MIPS datapath incrementally
  - Refining the overview design

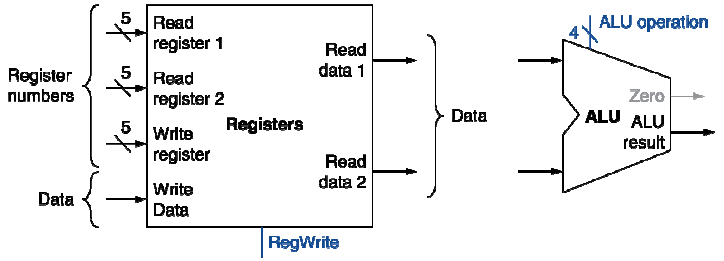
# Instruction Fetch






# R-Format Instructions

- Read two register operands
- Perform arithmetic/logical operation
- Write register result

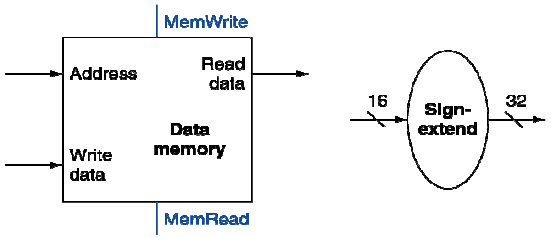



a. Registers b. ALU



# Load/Store Instructions

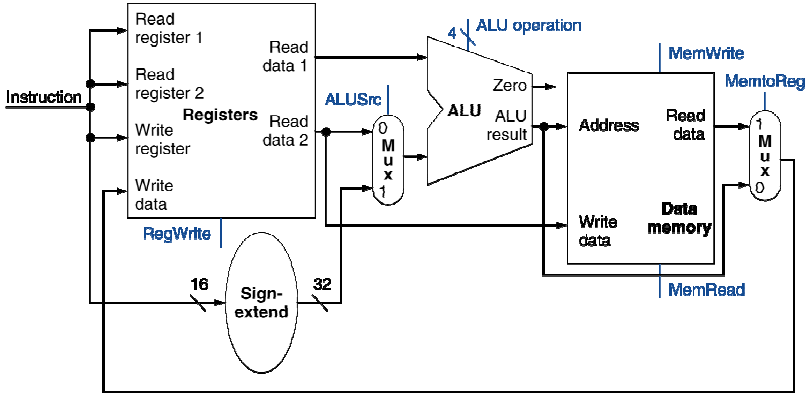
- Read register operands
- Calculate address using 16-bit offset
  - Use ALU, but sign-extend offset
- Load: Read memory and update register
- Store: Write register value to memory




a. Data memory unit b. Sign extension unit



# R-Type/Load/Store Datapath

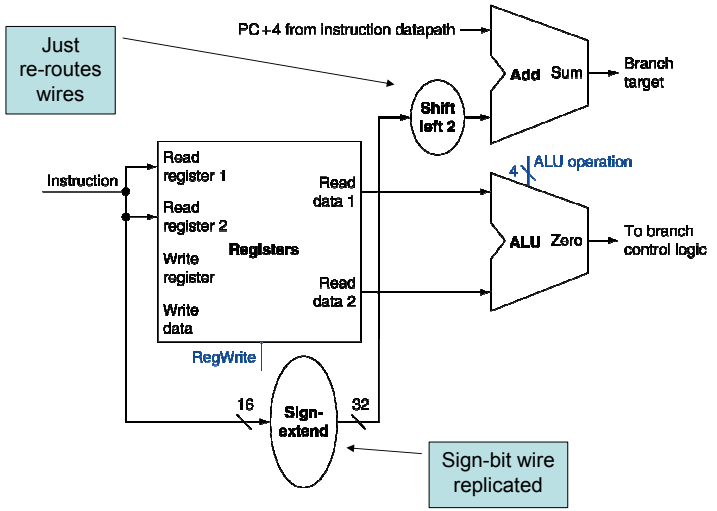


# Branch Instructions

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

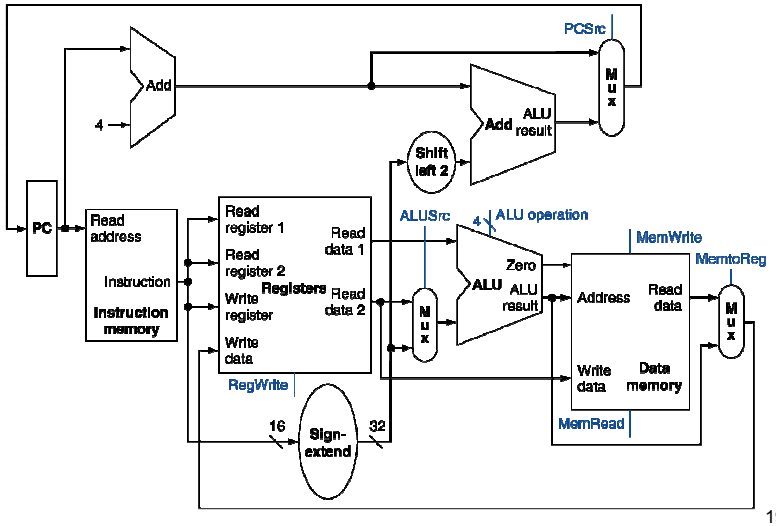


# Branch Instructions






# Full Datapath






# Composing the Elements

- First-cut data path does an instruction in one clock cycle
  - Each datapath element can only do one function at a time
  - Hence, we need separate instruction and data memories
- Use multiplexers where alternate data sources are used for different instructions