Lecture 8 – Introduction to Pipelining

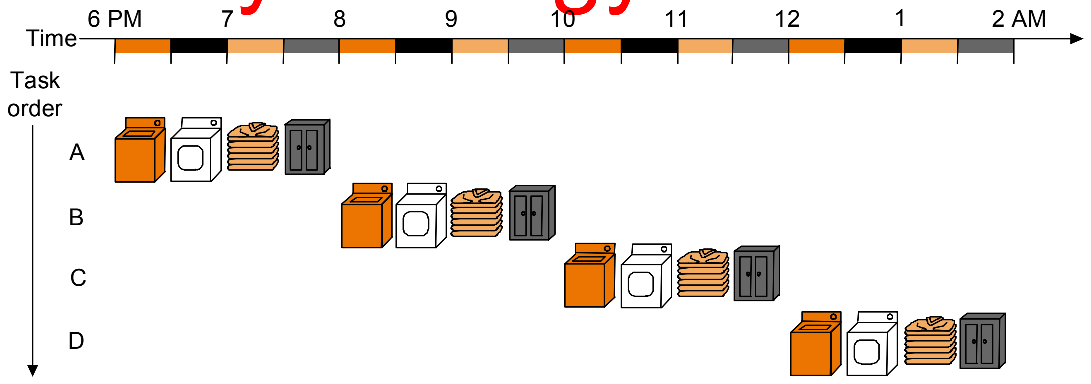
Karl R. Wilcox
Karl@cs.rhul.ac.uk

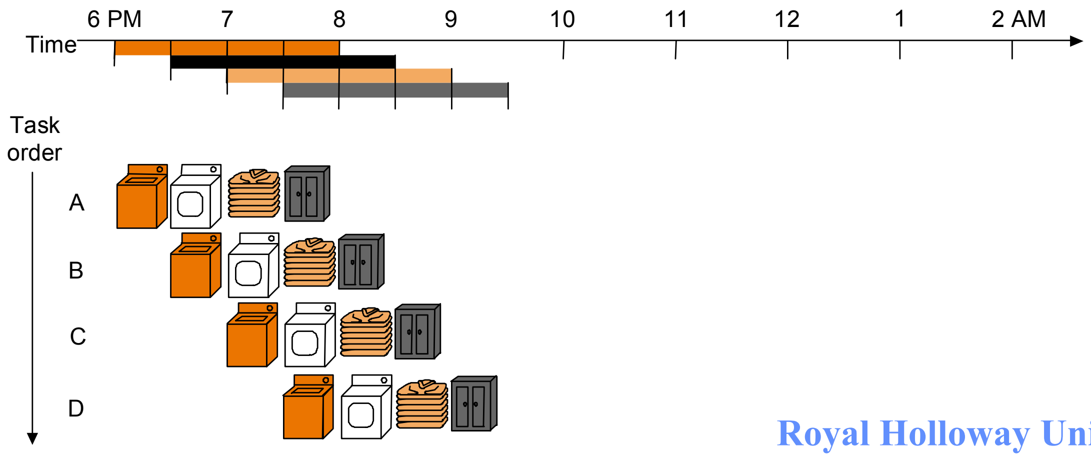
Objectives

- In this lecture we will cover
  - An overview of pipelining
  - Benefits and problems of pipelining
  - Instruction Sets for pipelining
  - Implementing a pipelined datapath
  - (diagrams are from Patterson & Hennessy)

Royal Holloway University of London

The Laundry Analogy





Royal Holloway University of London

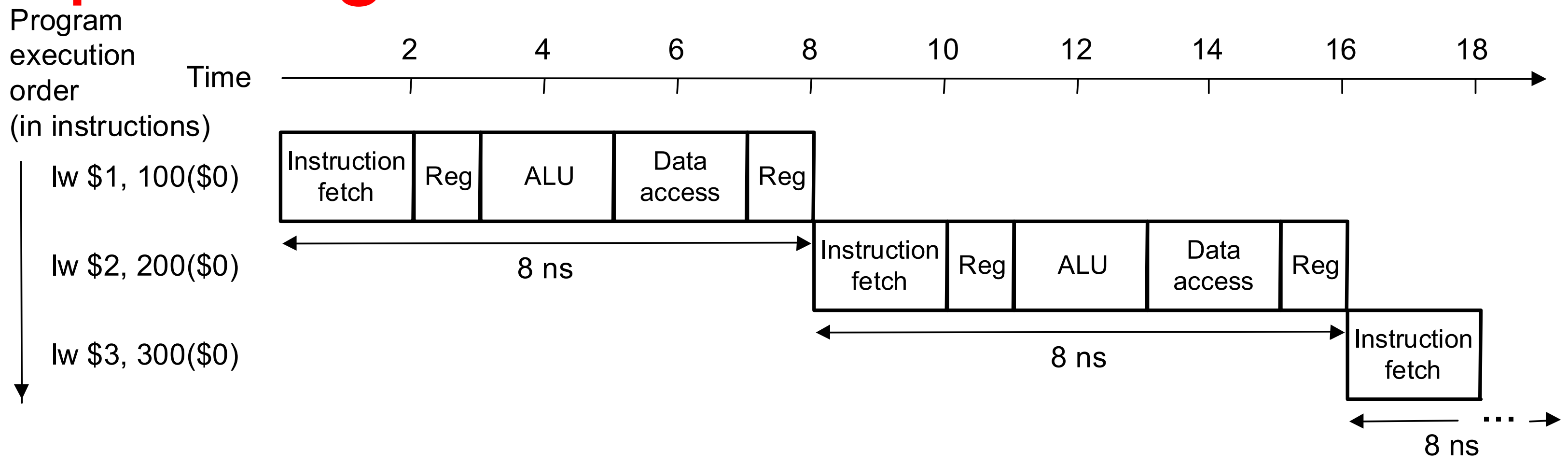
Points To Note

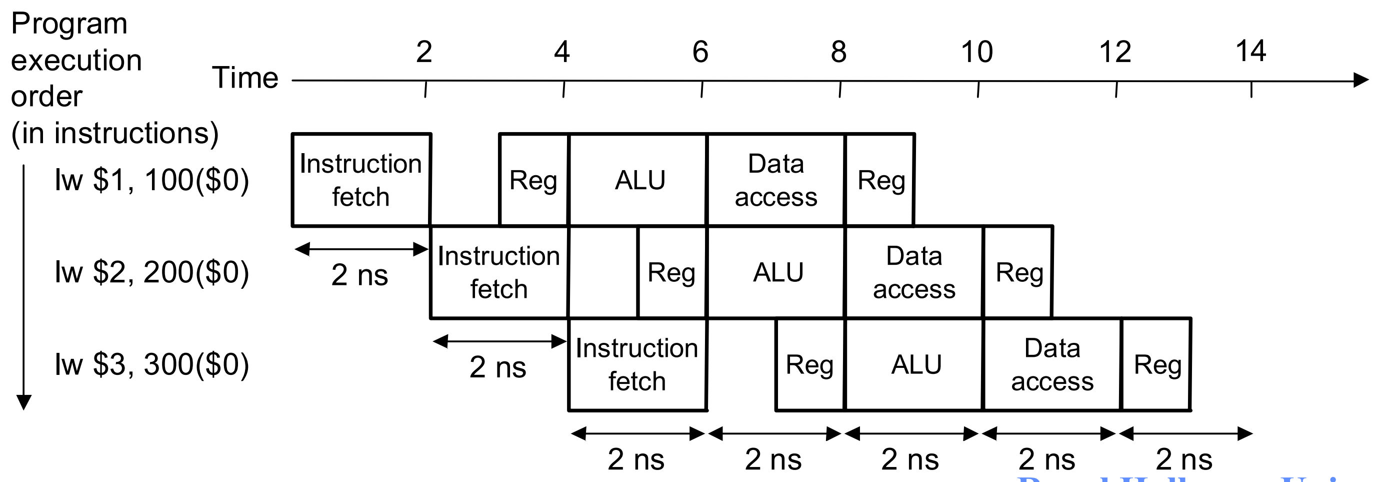
- The diagrams show machine usage over time
  - There is only one washing machine, used repeatedly
- Pipelining does NOT speed up activities
  - It still takes 2 hours to launder clothes
- Pipelining increases <u>throughput</u>
  - In a given time we can launder more clothes

Stages in MIPS Instruction Execution

1. **Fetch instruction from memory**
2. **Read registers and decode instruction type**
3. **Execute Operation or calculate address**
4. **Access operand in data memory**
5. **Write result to register**

Pipelining MIPS Instructions





Points To Note

- Each step in the execution must be the same length of time
  - That of the longest step
- In general, the longer the pipeline the greater the increase in speedup, but:
  - Each step needs to be “balanced”, (see above)
  - Pipelining (e.g. registers) imposes some overhead
  - (see also information on hazards later)

Problems with pipelines

- **We can only execute the next instruction overlapped with the current one if:**
  - **That concurrent instructions do not need the same hardware at the same time**
  - **We know what the next instruction will be**
  - **That all the data (operands) we require are available**
- **A <u>Hazard</u> exists if any of these conditions not met**

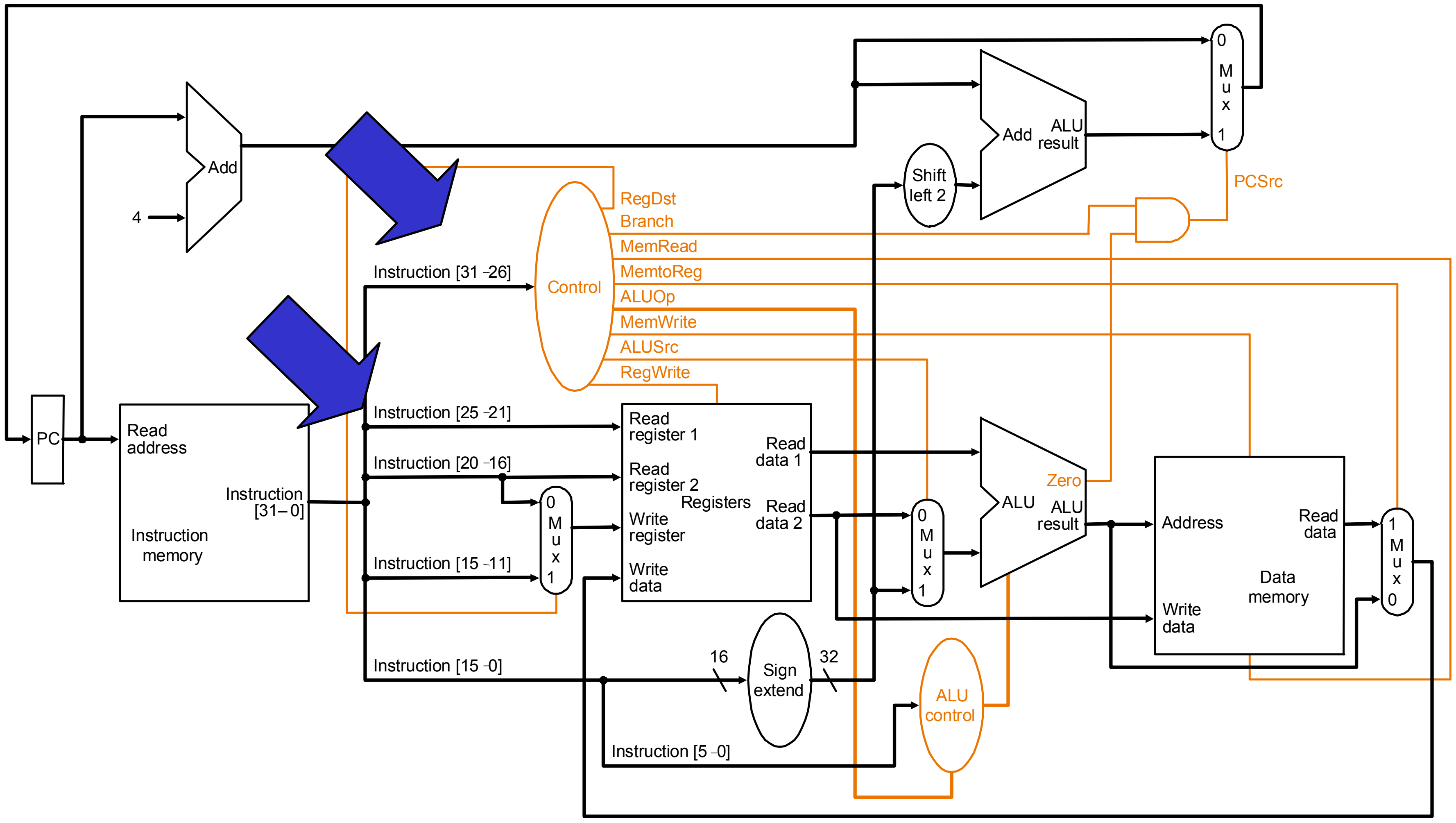
Structural Hazards

- **A structural hazard exists if the hardware cannot support the combination of instructions we wish to overlap**
  - **For example, if we need the ALU to calculate an address offset at the same time as another instruction needs it to add two registers**
- **Structural hazards can be minimised by careful design of the instruction set**

MIPS Instruction Design

- **The MIPS instruction set has been designed to aid pipelining by:**
  - Having all instructions the same length
    - Helps balance pipeline stages
  - Simple (and consistent) instruction formats
    - Register addresses always in the same place (see next slide)
    - Can select registers at the same time as instruction decode
  - Cannot operate directly on memory operands
    - Must load into registers first, store later
    - Execute stage calculates memory address, load or store later
  - Operands must be aligned on a 4 byte boundary
    - No need for double memory access to get 32 bits

Register and Instruction Decode

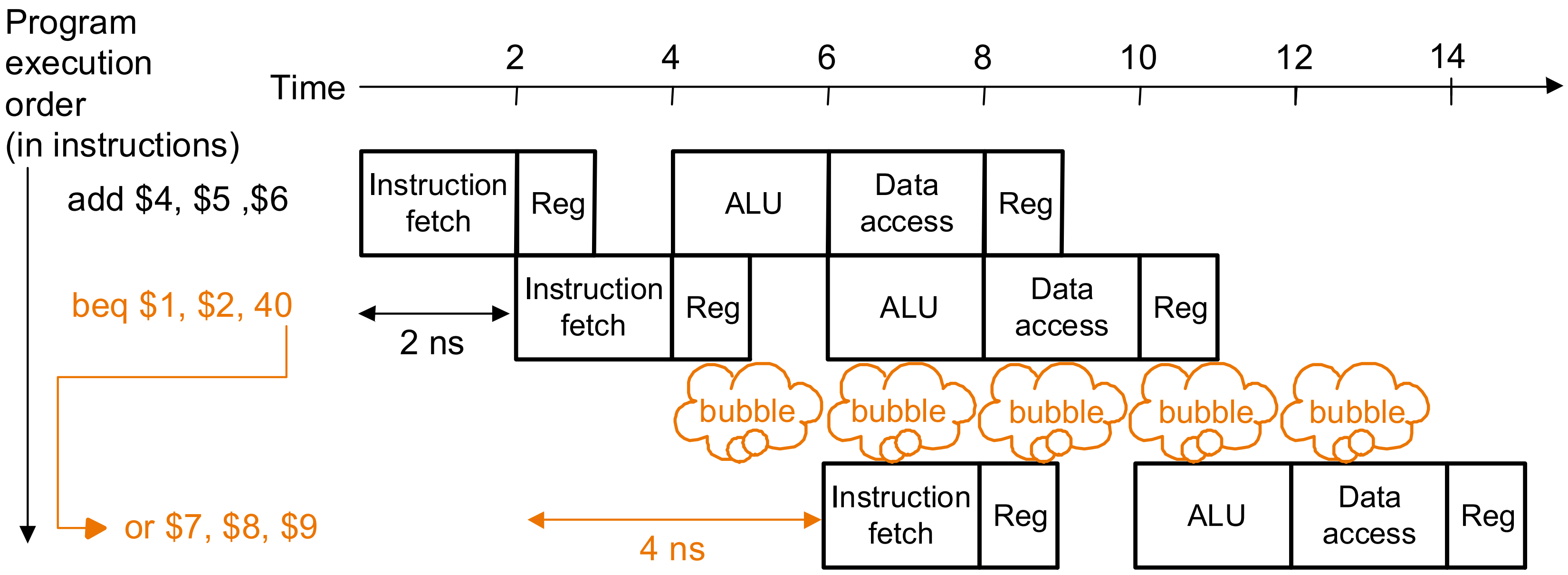


Control Hazards

- How do we handle jumps?
  - We do not know where to jump to until the instruction has been executed
  - The next instruction to execute is NOT sequential
- How do we handle conditional branches?
  - The next instruction MAY BE sequential
  - It may be equivalent to a jump
  - We do not know until instruction completion
- Two approaches – STALL and PREDICT

Stall on Branch

- If we encounter a branch, insert a delay into the pipeline
  - i.e. suspend overlapping until we know the outcome
  - Also known as putting a "bubble" in the pipeline



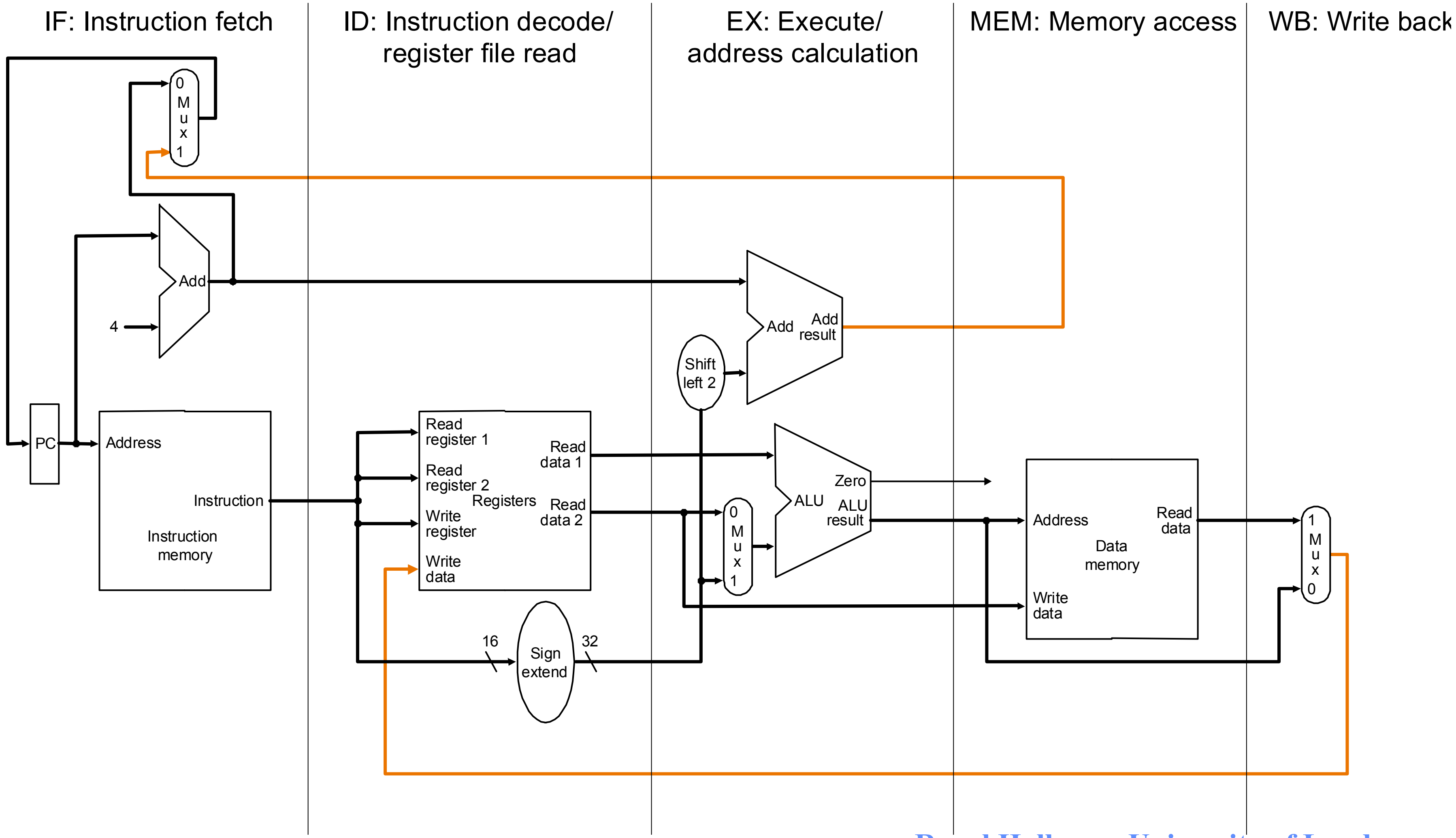
Predict Branch Outcome

- Assume we will always take branch
  - If we are correct we can carry on at full speed
  - If we are wrong we will have to discard some instructions
    - Known as “flushing” the pipelining
  - We must make sure we undo everything properly!
- Alternatively assume we never take branch
- Or try to predict branch outcome
  - Assume always take backwards jumps (bottom of loop)
  - Maintain history of recent jumps

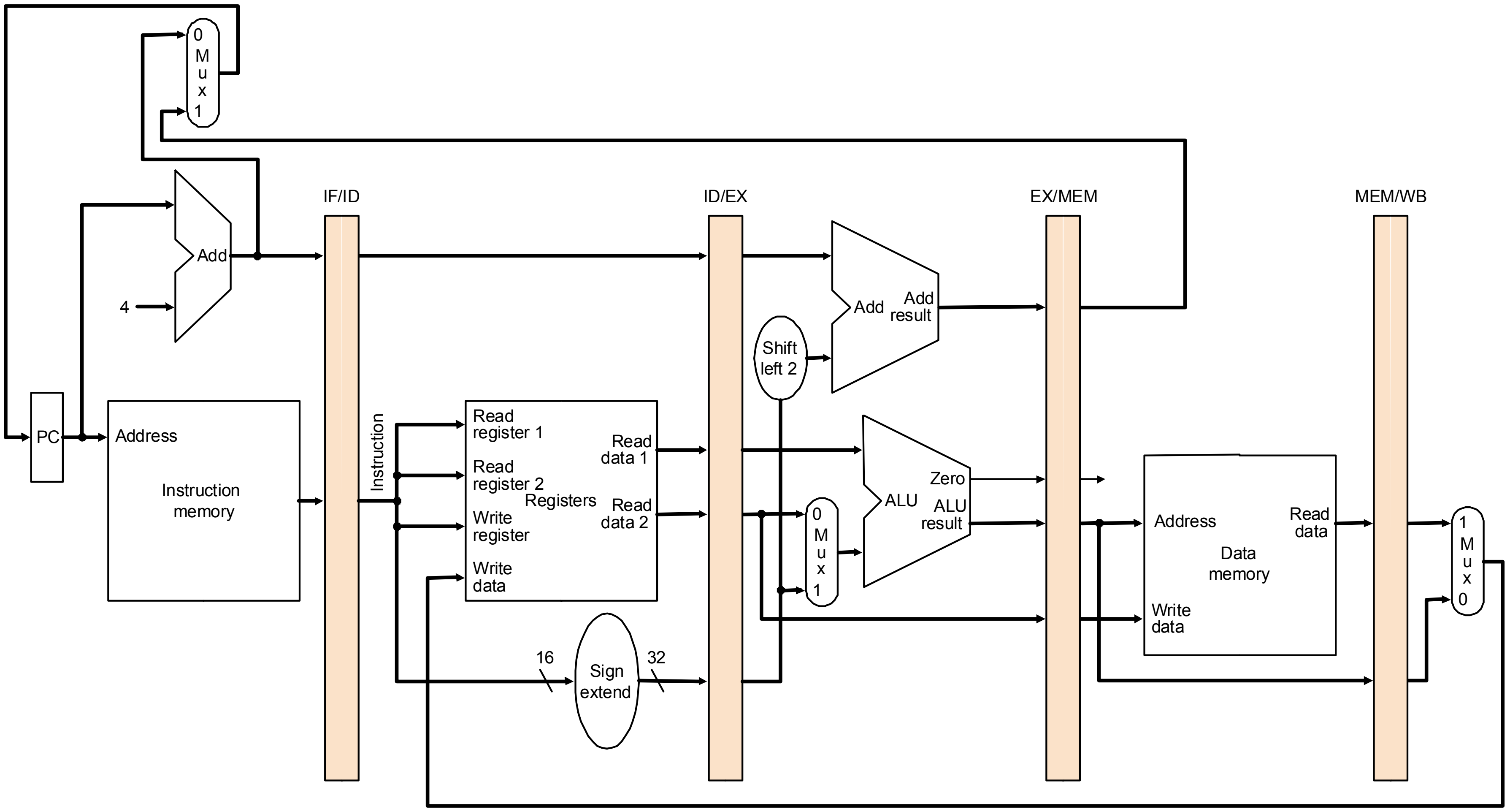
Data Hazards

- To execute an instruction we need all of the operands available
  - What if one operand is being calculated by the previous instruction?
- Two possible approaches
  - Stall the pipeline until data becomes available
  - Use <u>Forwarding</u>, move data between pipeline stages
  - (more on this next week)

Recall - Single Cycle Datapath



A Pipelined Datapath



Summary

- Pipelining is the process of overlapping instruction execution
  - Break instructions into smaller (balanced) steps
  - We need registers between each step + more control logic
- Pipelining increases <u>throughput</u>, it does NOT reduce execution time
- Pipeline <u>Hazards</u> exist where we cannot overlap instructions
  - Structural, control and data hazards

Next Lecture

- Pipelines in action!
- More detail on managing hazards
  - The MIPS branch delay slot
- Applicability of pipelining techniques