COMPUTER ORGANIZATION AND DESIGN
The Hardware/Software Interface
5th Edition

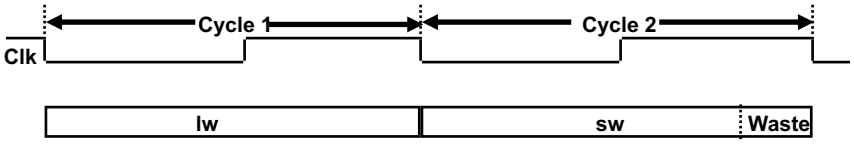
Chapter 4

The Processor

Pipelining

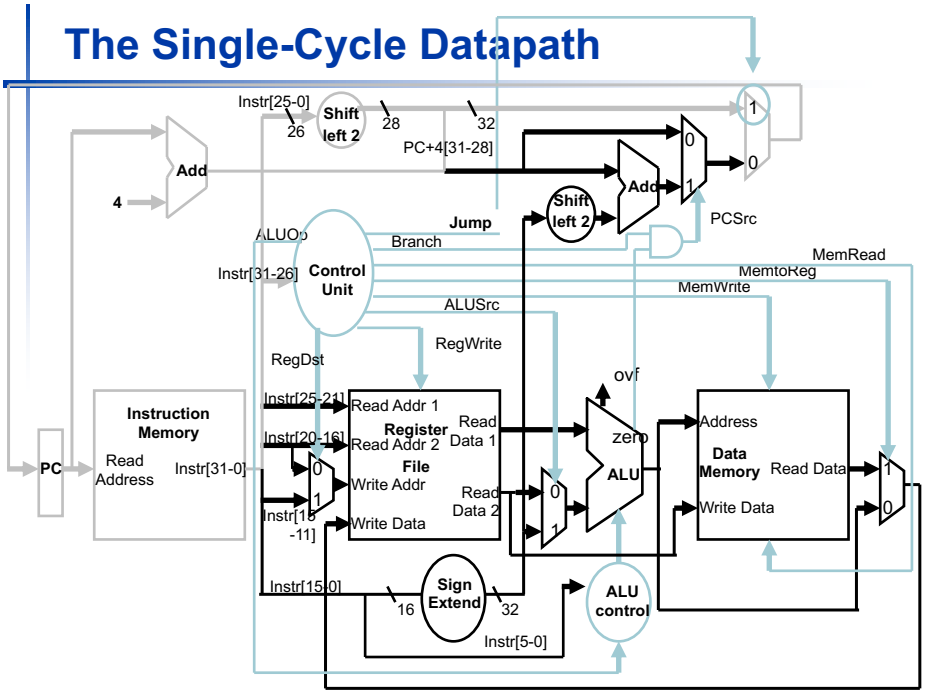
Single-Cycle Disadvantages & Advantages

- Uses the clock cycle inefficiently – the clock cycle must be timed to accommodate the ***slowest*** instruction.
  - This would be especially problematic for more complex instructions like floating point multiply.

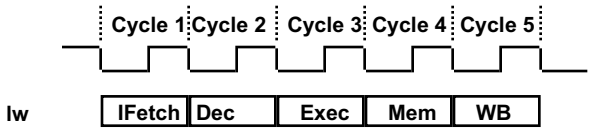


- May be wasteful of area. Some functional units (e.g., adders, memory) must be duplicated since they can not be shared during a clock cycle.
- However, the single-cycle implementation is simple and easy to understand.

The Single-Cycle Datapath



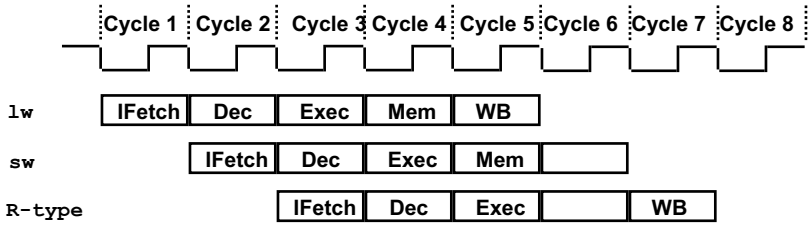
The Five Stages of the Load Instruction



- IFetch: Instruction Fetch and Update PC
- Dec: Instruction Decode and Register fetch
- Exec: Execute R-type; calculate memory address
- Mem: Read/write the data from/to the Data Memory
- WB: Write the result into the register file

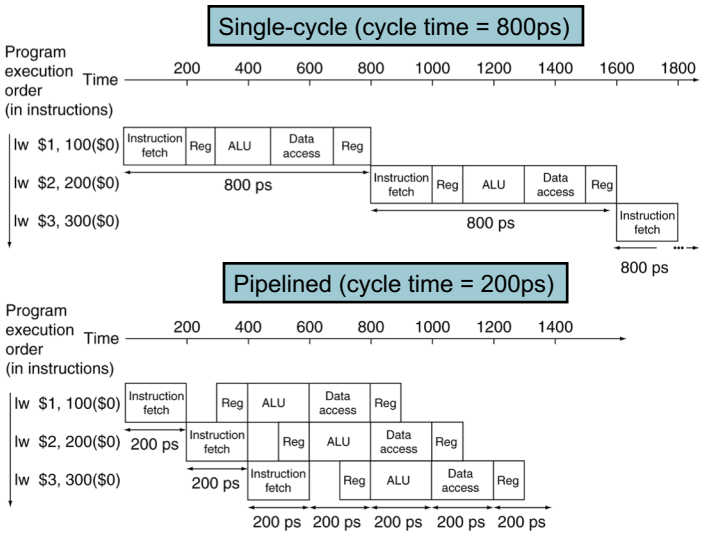
A Pipelined MIPS Processor

- Start the next instruction before the current one has completed
  - Improves ***throughput*** - total amount of work done in a given time.
  - Instruction ***latency*** (time from the start of an instruction to its completion) is *not reduced.* It is often increased due to imbalances between stages.



- Clock cycle (pipeline stage time) is limited by the **slowest** stage.
- For some stages, we don't need the whole clock cycle (e.g., WB).
- For some instructions, some stages are wasted cycles (i.e., nothing is done during that cycle for that instruction).

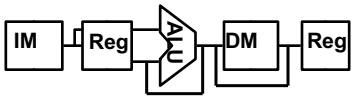
Pipeline Performance



Pipelining the MIPS ISA

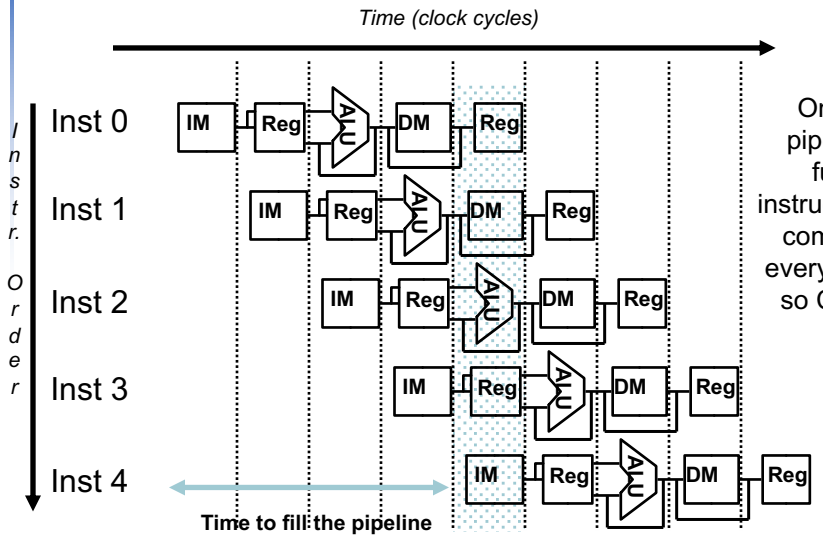
- What makes it easy
  - All instructions are the same length (32 bits)
    - Can fetch in the 1st stage and decode in the 2nd stage.
  - Few instruction formats (three) with *symmetry* across formats
    - Can begin reading register file in 2nd stage.
  - Memory operations occur only in loads and stores
    - Can use the execute stage to calculate memory addresses.
  - Each instruction writes at most one result (i.e., changes the machine state) and does it in the last pipeline stages (MEM or WB).
  - Operands must be aligned in memory so a single data transfer takes only one data memory access.

Graphically Representing MIPS Pipeline



- Can help with answering questions like:
  - How many cycles does it take to execute this code?
  - What is the ALU doing during cycle 4?
  - Is there a hazard (whatever that is), why does it occur, and how can it be fixed?

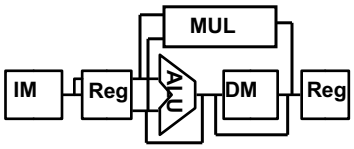
Five Instruction Sequence



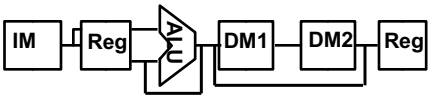
Once the pipeline is full, one instruction is completed every cycle, so CPI = 1

Other Pipeline Structures Are Possible

- What about a slow multiply instruction?
  - Make the clock twice as slow or …
  - Let it take two cycles (since it doesn't use the DM stage).

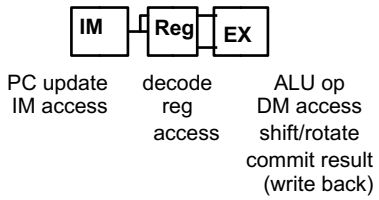


- What if the data memory access is twice as slow as the instruction memory?
  - Make the clock twice as slow or …
  - Let data memory access take two cycles (and keep the same clock rate).

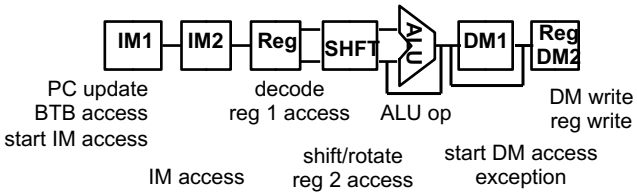


Other Pipeline Architectures

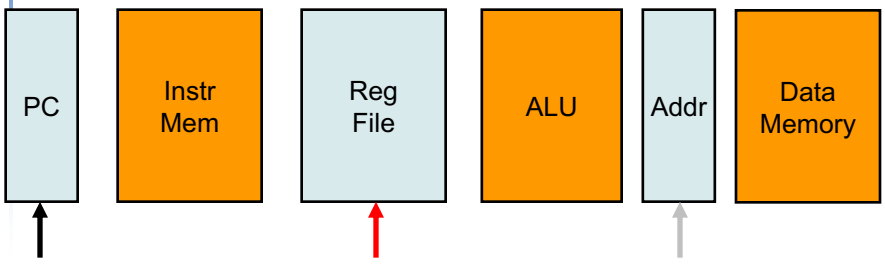
- ARM7



- Motorola XScale



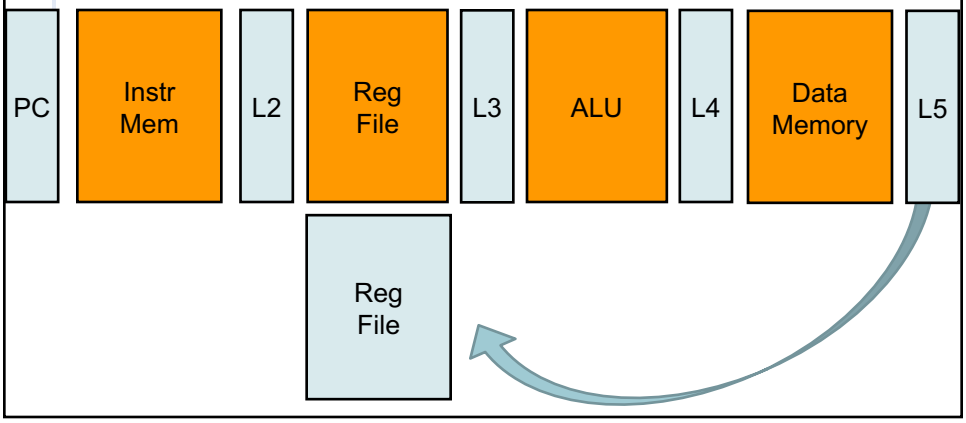
Latches and Clocks in Single-Cycle Design



- The entire instruction executes in a single cycle.
- Green blocks are latches.
- At the rising edge, a new PC is latched.
- At the rising edge, the result of the previous cycle is latched.
- At the falling edge, the address of LW/SW is latched so we can access the data memory in the 2nd half of the cycle.

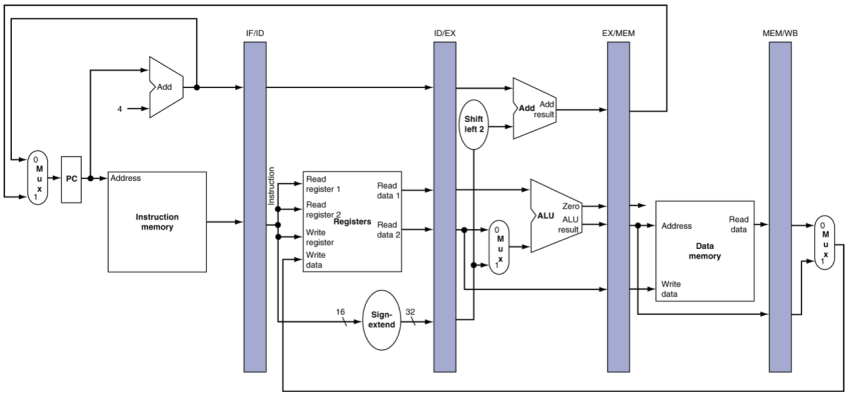
Multi-Stage Circuit

- Instead of executing the entire instruction in a single cycle (a single stage), let's break up the execution into multiple stages, each separated by a latch.



Pipeline Registers

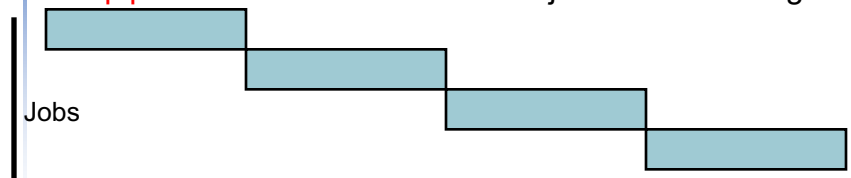
- Need registers between stages to hold information produced in previous cycle.



The Assembly Line

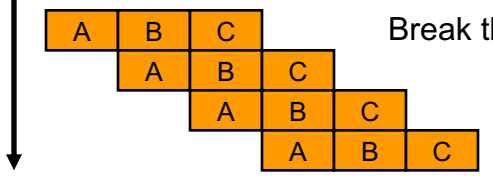
Unpipelined

Start and finish a job before moving on.



Jobs

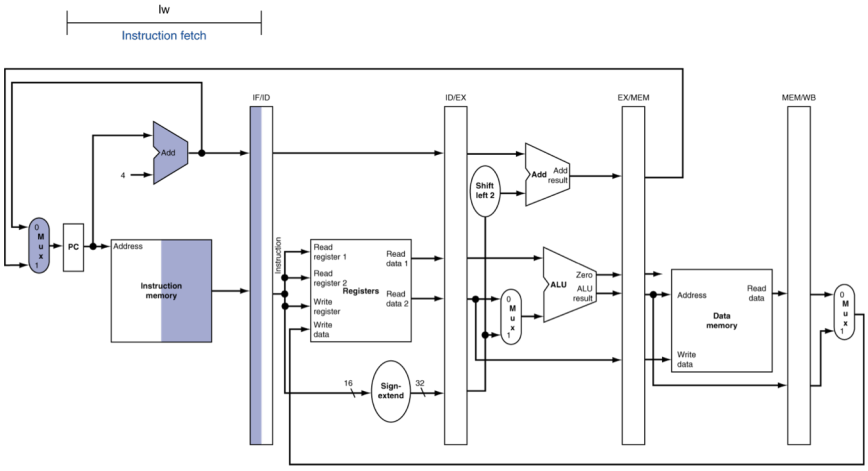
Time



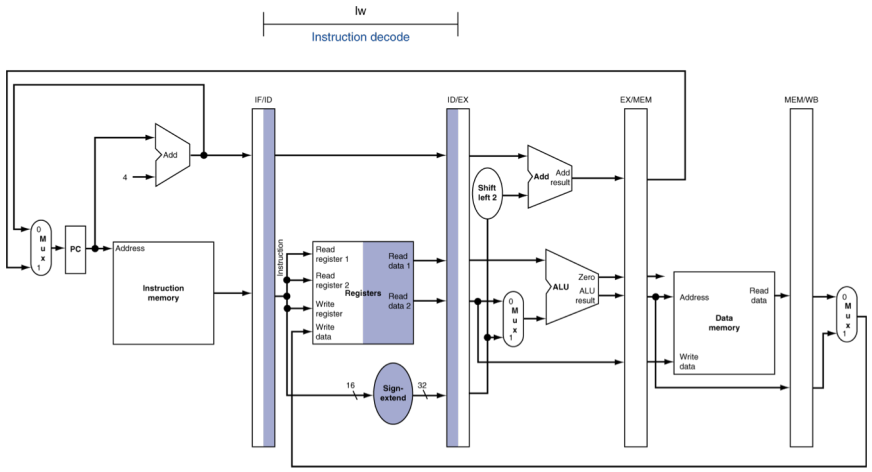
Break the job into smaller stages.

Pipelined

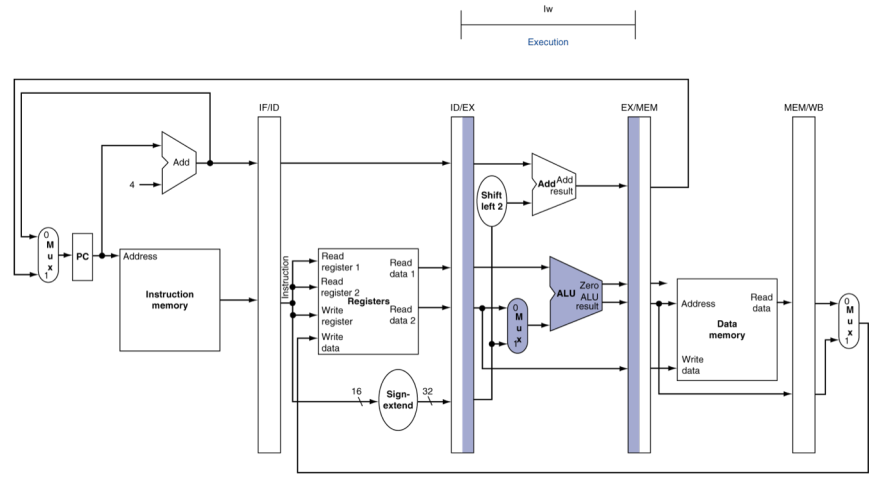
Instruction Fetch



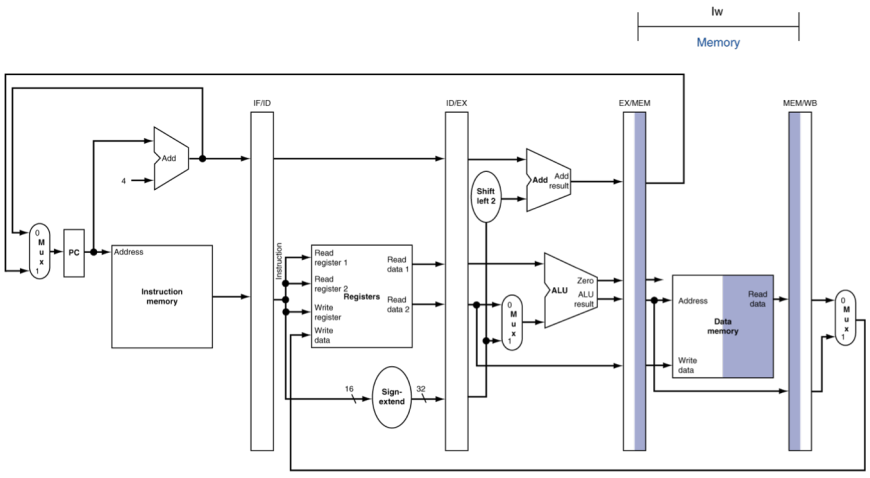
Instruction Decode and Register Fetch



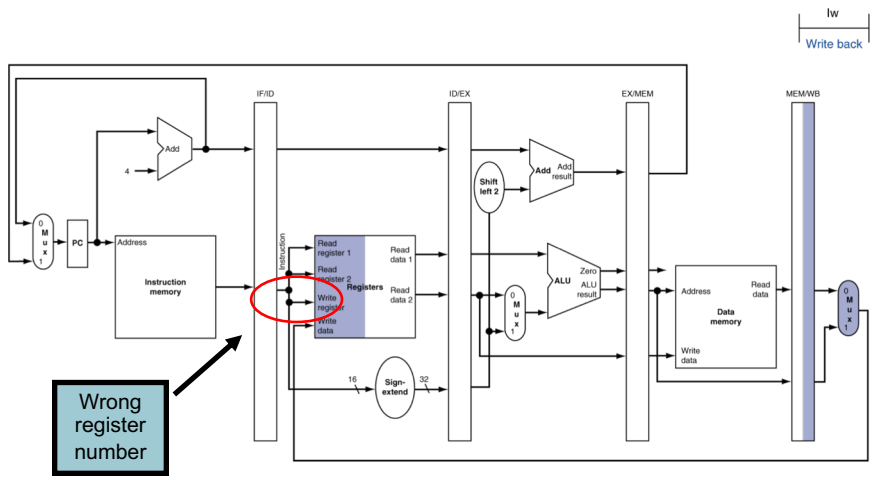
Execute



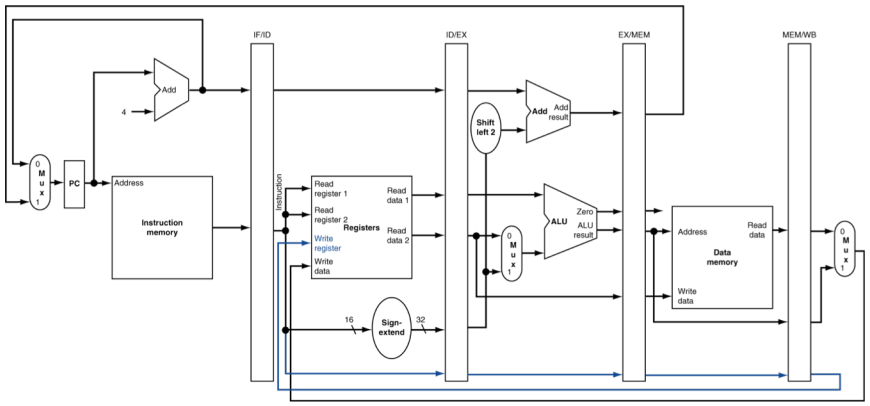
Memory Access (Load or Store)



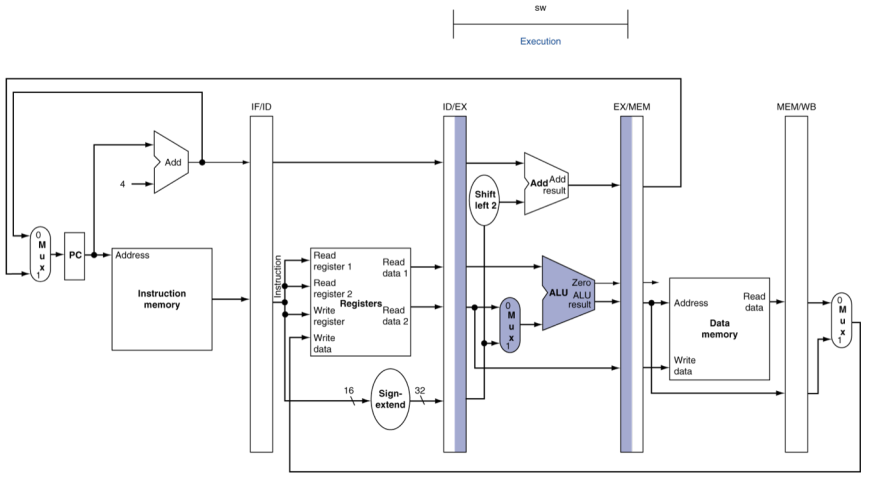
Write-back



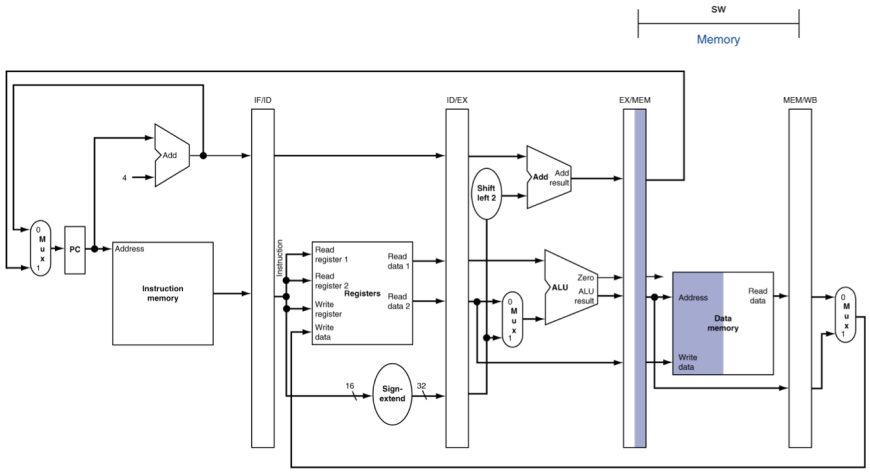
Corrected Datapath for Load Instruction



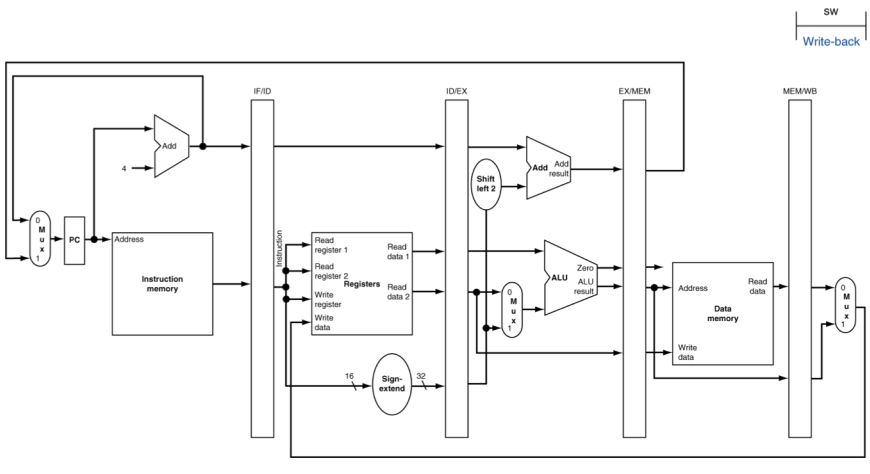
Execute Stage for Store Instruction



Memory Access Stage for Store



Write-back Stage for Store Instruction



Performance Improvements?

- Compared to single-cycle execution
  - Does it take longer to finish each individual instruction?
  - Does it take longer to finish a series of instructions?
  - Is a 10-stage pipeline better than a 5-stage pipeline?

Quantitative Effects

- As a result of pipelining:
  - Time in ns/instruction increases.
  - Each instruction takes more cycles to execute.
  - But… average CPI remains roughly the same.
  - Clock speed goes up.
  - Total execution time goes down, resulting in lower average time per instruction.
  - Under ideal conditions,
    - Speedup = ratio of *elapsed times between successive instruction completions* = number of pipeline stages = increase in clock speed

Can Pipelining Get Us Into Trouble?

- Yes: ***Pipeline Hazards***
  - **Structural hazards**: attempt to use the same resource by two different instructions at the same time.
  - **Data hazards**: attempt to use data before it is ready
    - An instruction's source operand(s) are produced by a prior instruction still in the pipeline.
  - **Control hazards**: attempt to make a decision about program control flow before the condition has been evaluated and the new PC target address calculated
    - Branch and jump instructions, exceptions.
- Hazards can usually be resolved by ***waiting.***
- Pipeline control must detect the hazard and take action to resolve it.

Summary

- All modern day processors use pipelining.
- Pipelining doesn't help *latency* of single task, it helps *throughput* of entire workload.
- Potential speedup: a CPI of 1 and faster Clock Cycle.
- Pipeline rate limited by *slowest* pipeline stage
  - Unbalanced pipe stages make for inefficiencies.
  - The time to "fill" pipeline and time to "drain" it can impact speedup for deep pipelines and short code runs.
- Must detect and resolve hazards
  - Stalling negatively affects CPI.