CMPE 110: Computer Architecture

Week 3
Pipelining

Jishen Zhao (http://users.soe.ucsc.edu/~jzhao/)

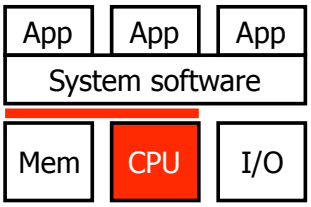
[Adapted in part from Jose Renau, Mary Jane Irwin, Joe Devietti, Onur Mutlu, and others]

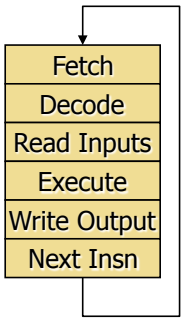
Reminder

- Quiz 1 will be posted on Monday

Review: ISA

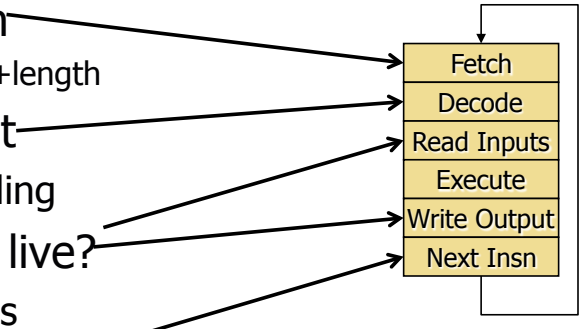
- What is ISA?
- Program execution model:
  - Compilation
  - Assembly & machine language
- Instruction execution model
  - Registers, memory, PC
  - Instruction execution
- ISA design goals
  - Programmability
  - Performance/implementability
  - Compatibility
- Aspects of ISAs





Review: Aspects of ISA

- Instruction length
  - Next instruction: PC+length
- Instruction format
  - Instruction encoding
- Where does data live?
  - Addressing modes
- Control transfers
  - How to find the next instruction
  - Branch
  - Jump

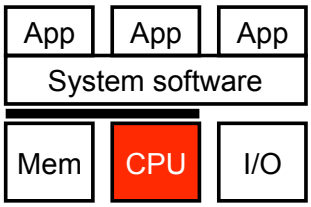


Today

- Basic concept of pipelining
- Single-cycle datapath vs. pipelined datapath
- Latency vs. throughput & performance
- Basic pipelining

Pipelining

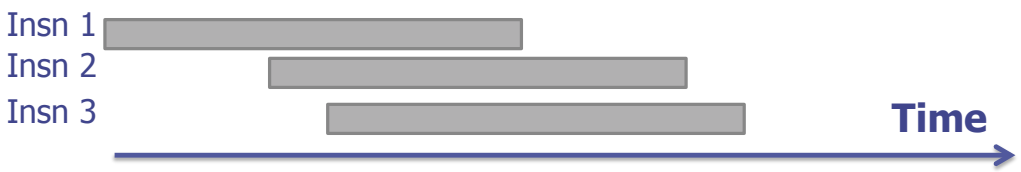
Pipelining



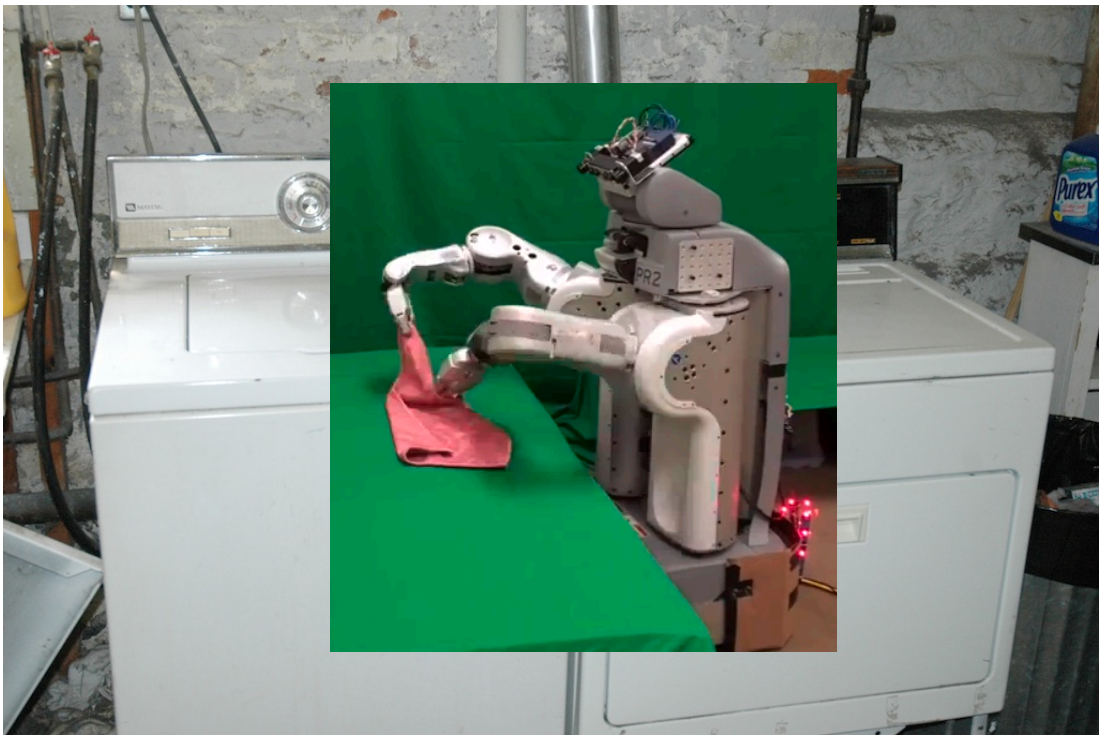
- Single-cycle datapaths
- Latency vs. throughput & performance
- Basic pipelining
- Data hazards
  - Bypassing
  - Load-use stalling
- Pipelined multi-cycle operations
- Control hazard
  - Branch prediction

Main Concept

- Instructions (C++) broken down into finite set of assembly language instructions
- Instructions (start) executing sequentially
- Pipelining method speeds up sequential execution of these instructions
  - The next instruction can get started, while the previous one is still running



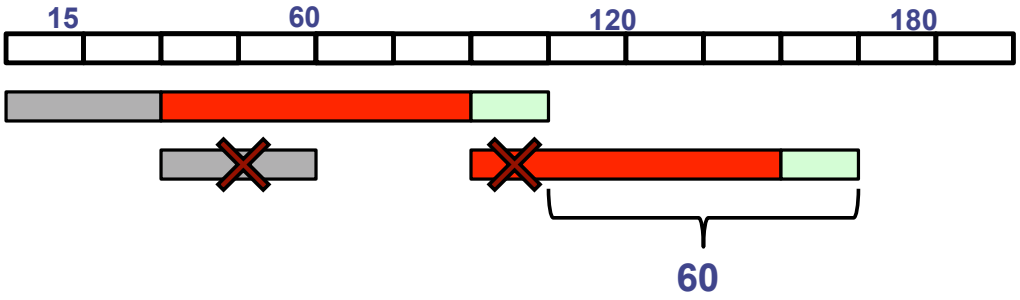
The eternal pipelining metaphor



Example

- You have a washer, dryer, and "folding robot"
  - Each takes 1 unit of time per load
- Assumptions
  - Washing takes 30 minutes, drying 60 minutes, and folding 15 min
- Questions
  - How long for one load in total?
  - How long for two loads of laundry?
  - How long for 100 loads of laundry?

Answers



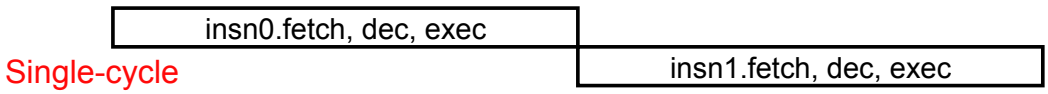
- Now assume:
  - Washing takes 30 minutes, drying 60 minutes, and folding 15 min
  - How long for one load in total? **105 minutes**
  - How long for two loads of laundry? 105 + 60 = **165 minutes**
  - How long for 100 loads of laundry? 105 + 60*99 = **6045 min**

Pipelined Datapath

Recall: Latency vs. Throughput

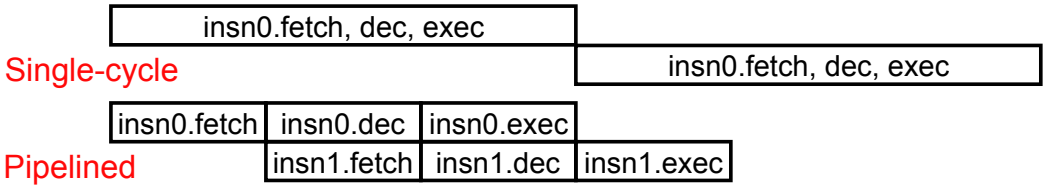
- **Latency (execution time)**: time to finish a fixed task
- **Throughput (bandwidth)**: number of tasks in fixed time
- Choose definition of performance that matches your goals
  - Scientific program? Latency, web server: throughput?

Latency vs. Throughput

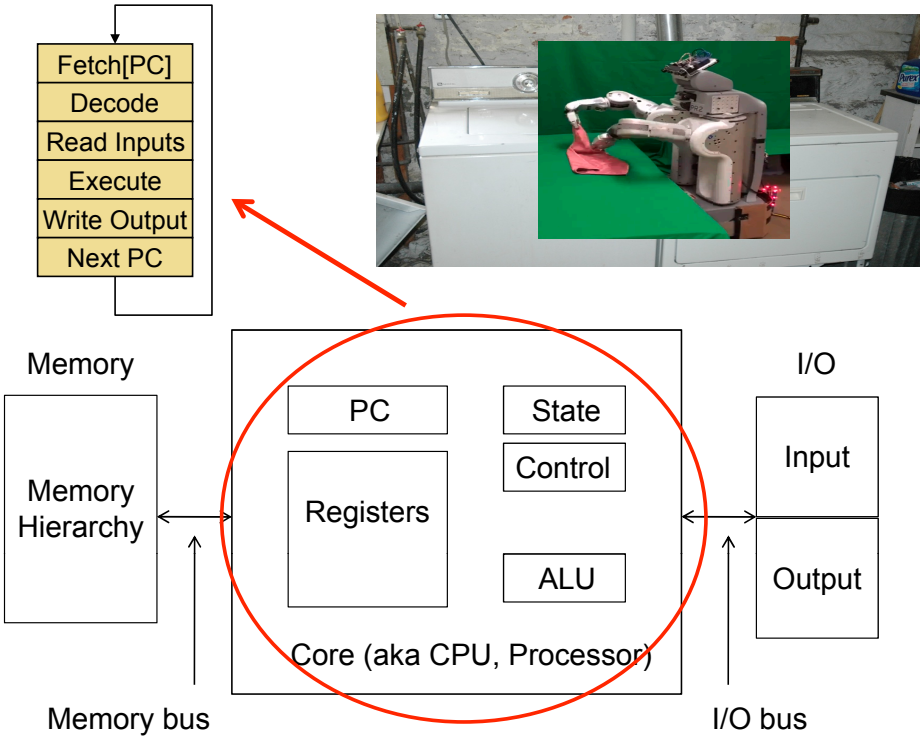


- Single instruction latency
  - Doesn't matter: programs comprised of billions of instructions
  - Difficult to reduce anyway
- Goal is to make programs, not individual insns, go faster
  - Instruction throughput → program latency
  - Key: **exploit Inter-insn Parallelism**

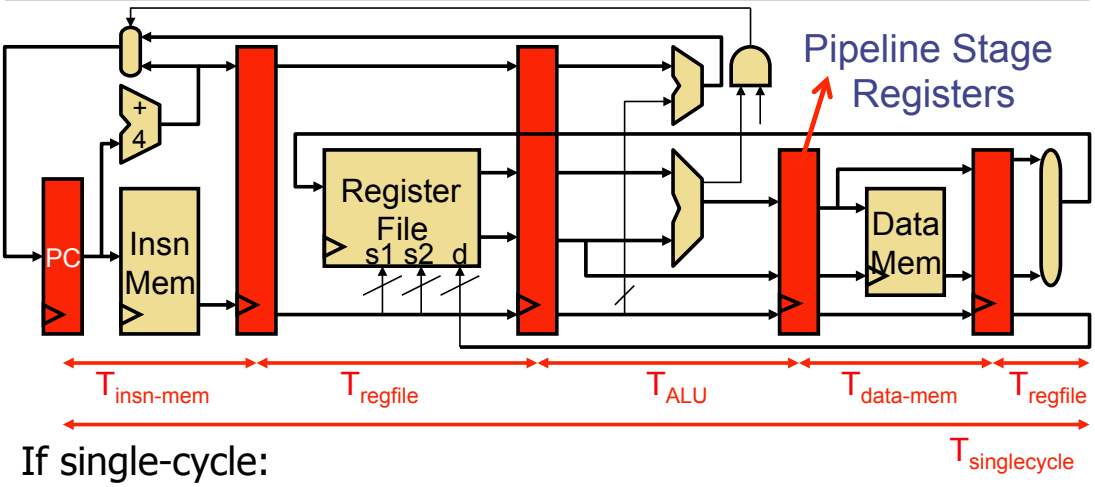
Pipelining



- Important performance technique
  - **Improves instruction throughput, not instruction latency**
- How it works
  - When insn advances from stage 1 to 2, next insn enters at stage 1
  - Maintains illusion of sequential fetch/execute loop
  - Individual instruction takes the same number of stages
  - \+ **But instructions enter and leave at a much faster rate**

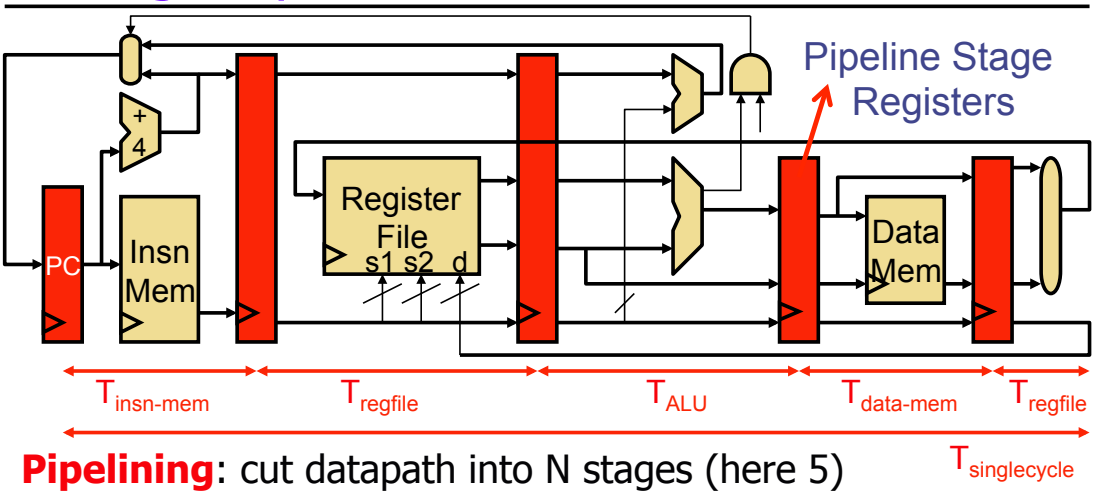


5 Stage Pipeline: Inter-Insn Parallelism

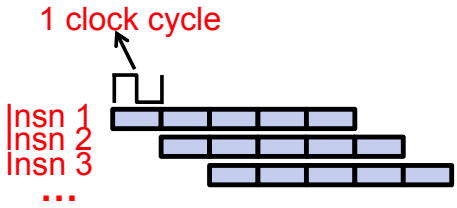


- If single-cycle:
  - Time to execute each instruction is $T_{singlecycle}$

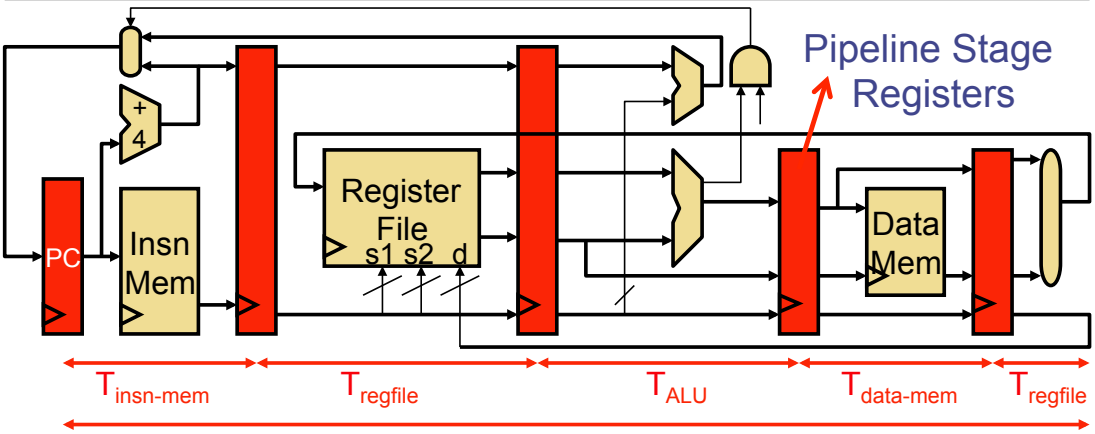
5 Stage Pipeline: Inter-Insn Parallelism



- **Pipelining**: cut datapath into N stages (here 5)
  - One insn in each stage in each cycle

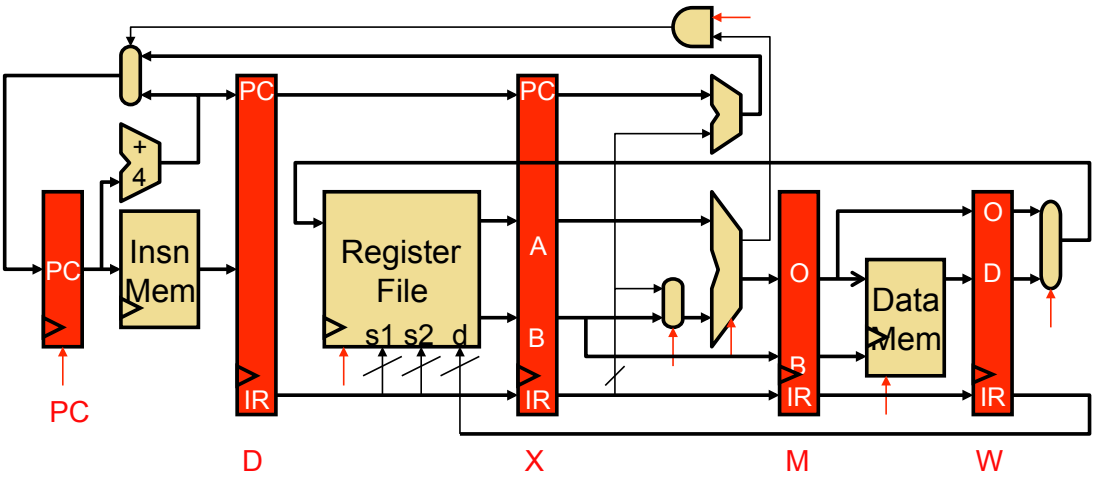


5 Stage Pipeline: Inter-Insn Parallelism



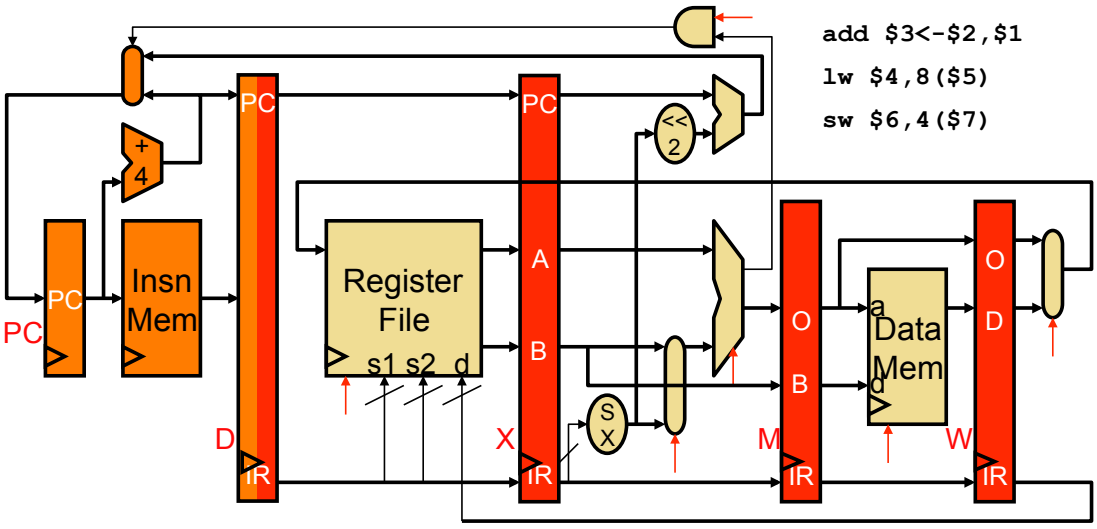
- **Pipelining**: cut datapath into N stages (here 5)
  - One insn in each stage in each cycle
  + Clock period = MAX($T_{insn-mem}$, $T_{regfile}$, T_{ALU}, $T_{data-mem}$)
  + Base CPI = 1: insn enters and leaves every cycle
    - Actual CPI > 1: pipeline must often "stall"
  - Individual insn latency increases (pipeline overhead)

5 Stage Pipelined Datapath



- Five stage: **F**etch, **D**ecode, e**X**ecute, **M**emory, **W**riteback
  - Nothing magical about 5 stages (Pentium 4 had 22 stages!)
- Latches (pipeline registers) named by stages they begin
  - **PC**, **D**, **X**, **M**, **W**

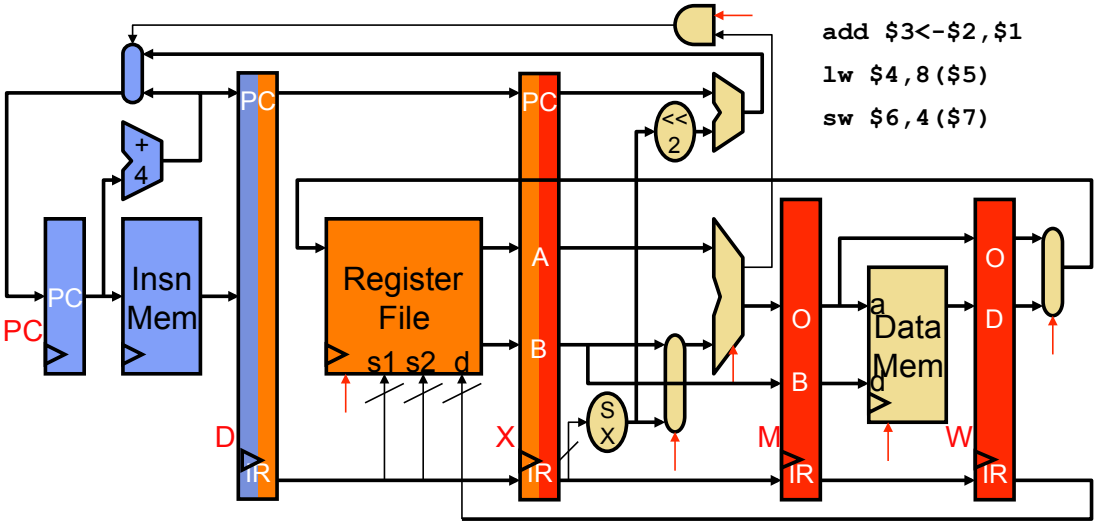
Pipeline Example: Cycle 1



`add $3<-$2,$1`

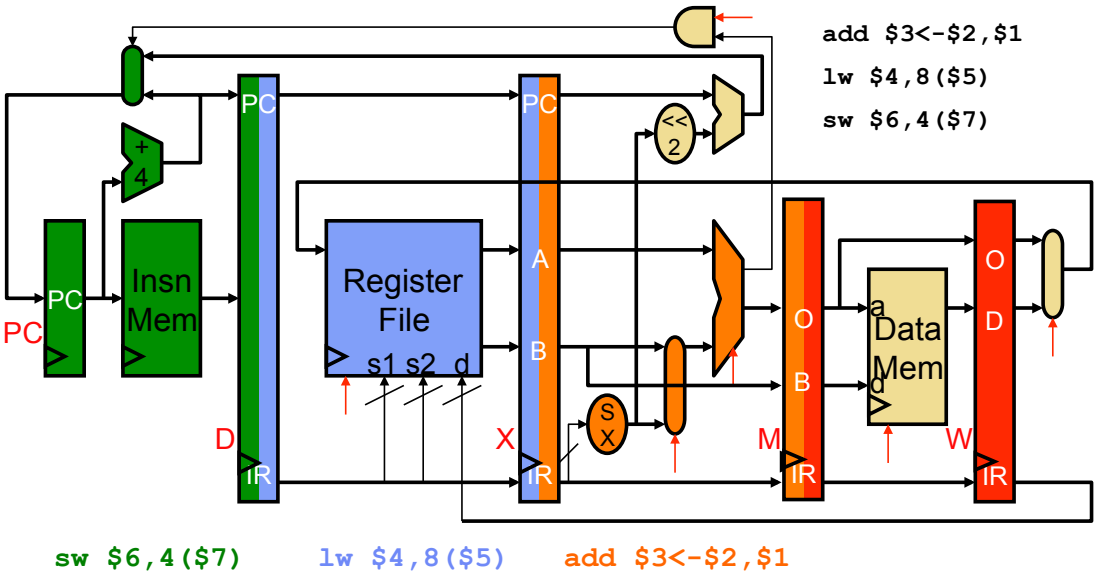
- 3 instructions

Pipeline Example: Cycle 2

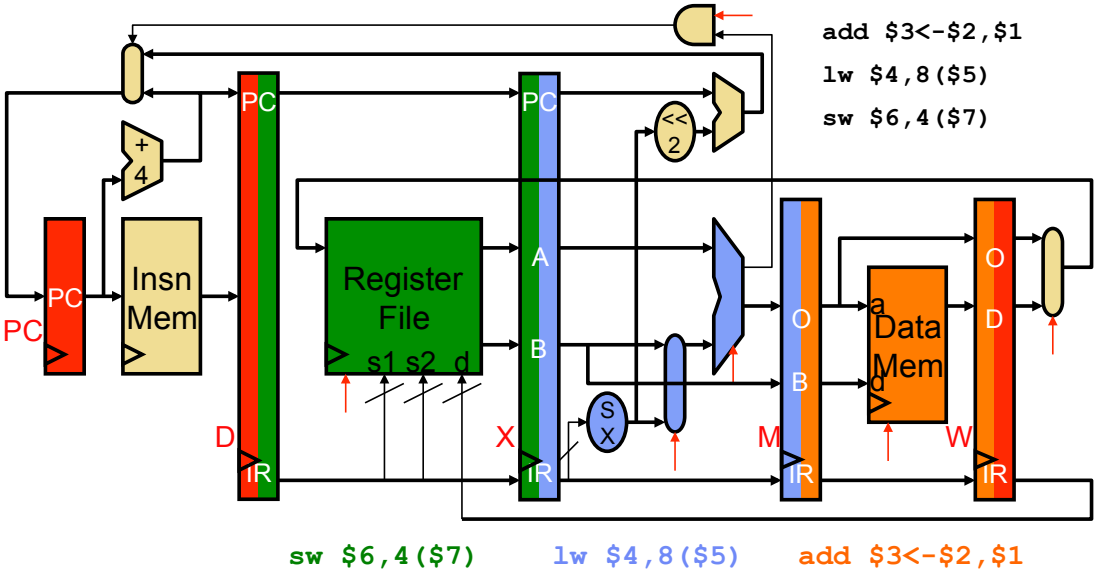


`lw $4,8($5)` `add $3<-$2,$1`

Pipeline Example: Cycle 3

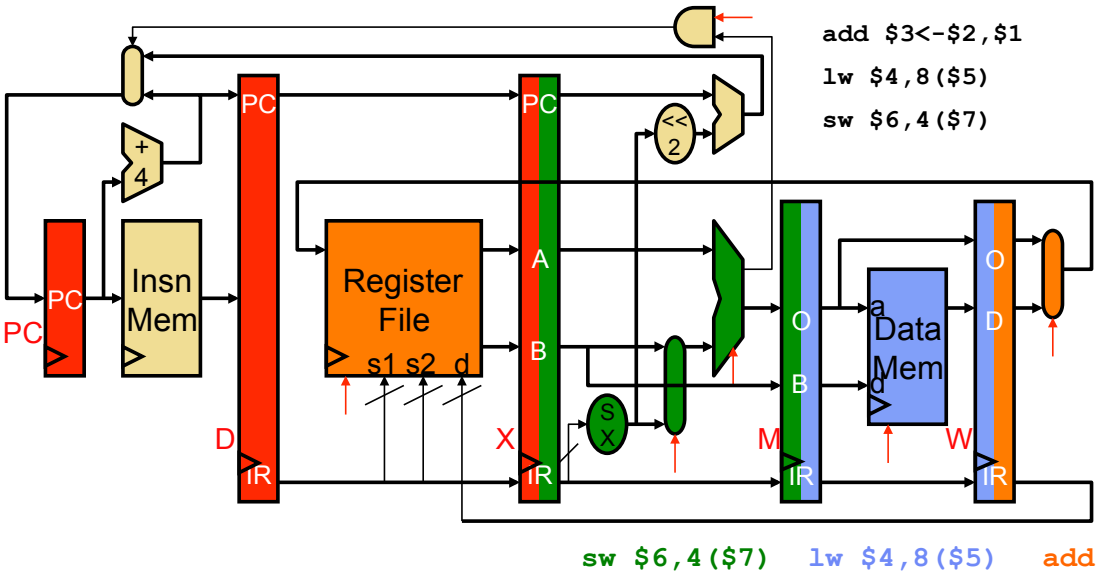


Pipeline Example: Cycle 4

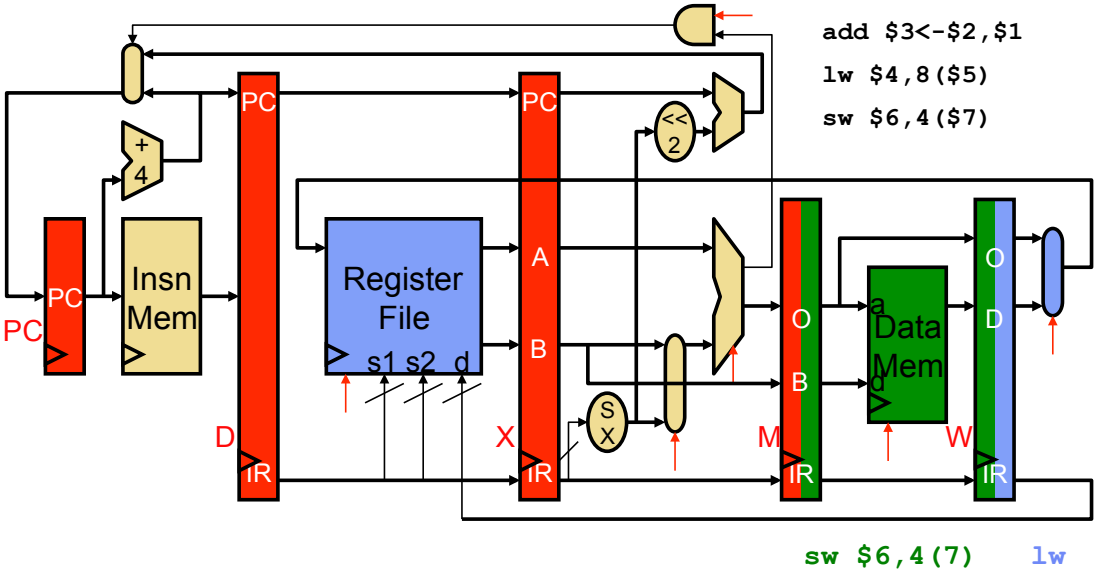


- 3 instructions

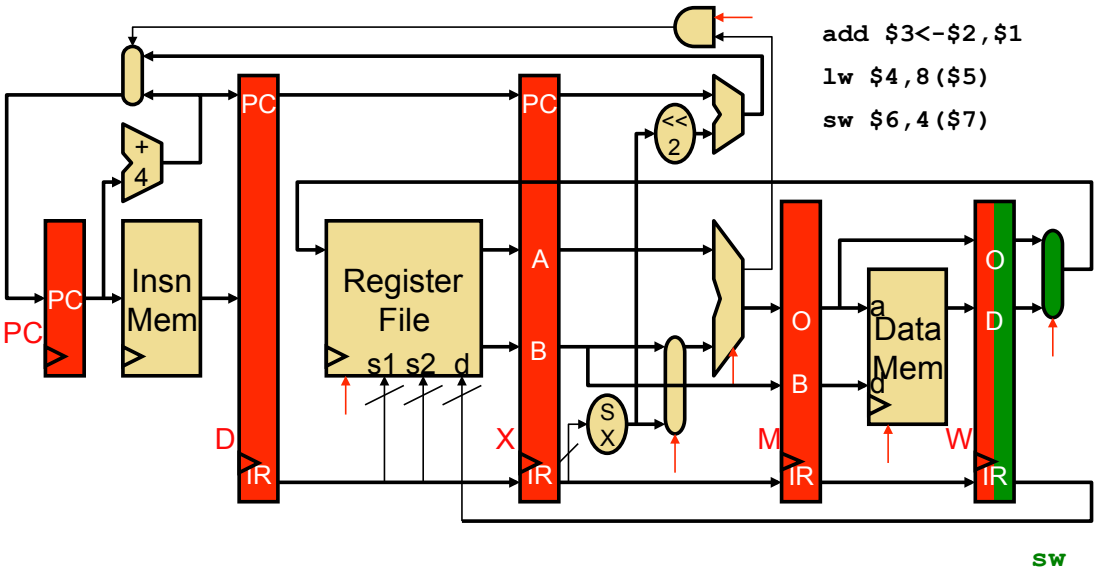
Pipeline Example: Cycle 5



Pipeline Example: Cycle 6



Pipeline Example: Cycle 7



Pipeline Diagram

- **Pipeline diagram**: shorthand for what we just saw
  - Across: cycles
  - Down: insns
  - Convention: e.g., **X** means `lw $4,8($5)` finishes e**X**ecute stage and writes into M latch at end of cycle 4

	1	2	3	4	5	6	7	8	9
`add $3<-$2,$1`	F	D	X	M	W				
`lw $4,8($5)`		F	D	**X**	M	W			
`sw $6,4($7)`			F	D	X	M	W		

Data Dependences, Pipeline Hazards, and Bypassing

Preview: Dependences and Hazards

- **Dependence**: relationship between two insns
  - **Data dep.**: two insns use same storage location
  - **Control dep.**: one insn affects whether another executes at all
  - **Enforced** by making older insn go before younger one
    - Happens naturally in single-cycle designs
    - But not in a pipeline!

- **Hazard**: dependence & possibility of wrong insn order
  - **Stall**: for order by keeping younger insn waiting in same stage
  - Hazards are a bad thing: stalls reduce performance