

# Pipelining and Instruction Pipelining

---

- ◆ **Last Time**
  - Midterm Exam, Grading in progress
- ◆ **Today**
  - Pipelining: performance thru concurrency
  - Applying Pipelining to Instruction Execution
  - Hazards and when they are problems
- ◆ **Reminders/Announcements**
  - Read P&H Chapter 6.1-6.7, Pipelining

CS 141 Chien

1

Feb 22, 2000

# Complete Basic Computer Implementation

---

- ◆ **Constituents**
  - Instruction Fetch
  - Decode
  - Read Registers
  - Execute
  - Write Registers
- ◆ **Single Cycle Control (slow...)**
- ◆ **Multiple Cycle Control (Control FSM)**
- ◆ **Exceptions**
- ◆ **So, how can we make it go faster?**

CS 141 Chien

2

Feb 22, 2000

## Concepts of Pipelining

---

- ◆ **Latency** = Time from initiation of an operation until its results are available
- ◆ **Examples**
  - Adder: time from inputs valid to output valid
  - Memory: time from addresses valid, read strobe, to data out
  - Control logic: time from stable inputs to stable outputs
  - Others?
- ◆ **Macroscopic examples? (real life)**

## Latency Examples (real life)

---

- ◆ **Line at McDonalds: 10 mins** from entry to “have food”
- ◆ **Car ride SD -> LAX: 2.5 hours** (or faster)
- ◆ **Homework grading: time from turn-in, to handed back** (hopefully, < 1.5 week)
- ◆ **Turning tap on until water comes out of hose**
- ◆ **Others?**

## Throughput

---

- ◆ **Throughput** = rate at which something happens or gets done. Generally the initiation rate or completion rate is fine. Usually OPs / unit time
- ◆ **Examples**
  - Instructions per second (instruction processing throughput)
  - Floating point operations per second (floating point throughput)
  - Megabits per second (network throughput)
  - Megawords per second (memory read throughput)
- ◆ **Macroscopic examples?**

## Throughput Examples

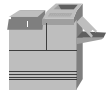
---

- ◆ **McDonalds, 10 people served per minute,**
    - 600 people get food in 1 hour!
    - Latency = 10 mins, how many people served while you wait?
  - ◆ **Car Drive SD -> LAX, one car, 5 people, 2 people per hour**
    - 2.5 hours latency
    - Rate NOT same as latency
  - ◆ **Homework grading, 100 homeworks per week**
    - 16.7 homeworks per day, but latency is still 1 week.
- =>Rate NOT same as latency, no direct relationship.**

## Contrasting Latency and Throughput

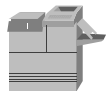
---

### ◆ Multiple Resources (Parallelism)



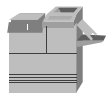
5 seconds to copy

12 copies / minute



5 seconds to copy

12 copies / minute



5 seconds to copy

12 copies / minute

**All 3 machines: 5 secs to copy, 36 copies / minute**

## Latency versus Throughput

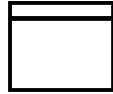
---

- ◆ **Replication *only increases throughput*, not latency.**  
Throughput is additive, latency is  $\min(x, y)$ .
- ◆ **Latency is a critical commodity, and very expensive to improve.**
- ◆ **Pipelining and replication only improve throughput.**
- ◆ **Pipelining: the basic idea**
  - Think “assembly line”
  - Breaking total work into small components
  - Each component can be “busy” doing useful work

## Pipelining

---

### ◆ Washing Laundry: Washer + Dryer



30 minutes



50 minutes

Latency for a wash =  $30 + 50 = 80$  minutes

2 loads ==> 160 minutes = 2 hrs, 40 minutes?

Pipelined: start 1 wash, start second when first goes into the dryer.

CS 141 Chien

9

Feb 22, 2000

## Pipelining

---

### ◆ Washer and Dryer



30 minutes



50 minutes

Latency = 80 minutes

Overlapped execution allows us to achieve....

2 loads in  $30 + 50 + 50 = 130$  minutes, much faster!

Throughput = 1 load / 50 minutes = 1.2 loads / hour

Latency and Throughput are not reciprocals.

CS 141 Chien

10

Feb 22, 2000

## Doing Pipelining Well

---

- ◆ **Issues for good performance?**
  - Unrelated activities
  - Equal pipeline stages (all match clock)
  - Fast issue rate (short clock period)
- ◆ **What limits the performance improvement in our simple washer/dryer case?**
  - granularity
  - moving the clothes
  - others?
- ◆ **How do these apply to computers? Instruction execution**

CS 141 Chien

11

Feb 22, 2000

## Pipelining Instruction Execution

---

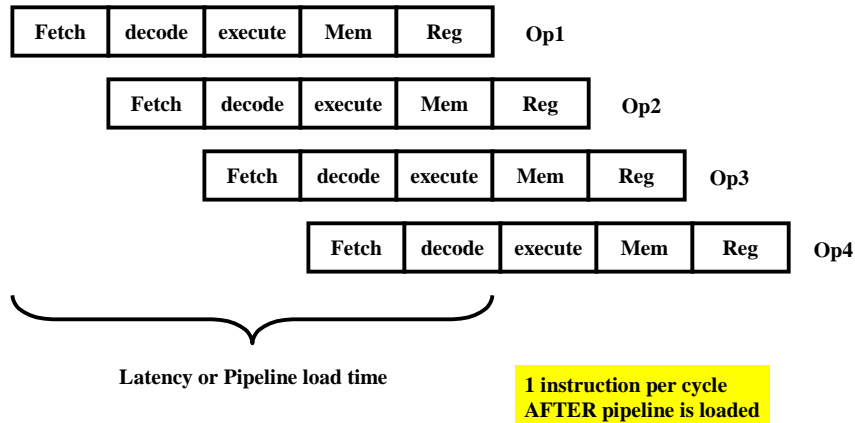
- ◆ **Parts: Fetch, Decode/Rd, Exec, WriteResults**
  - Overlap the parts
  - Overlap execution of several instructions
  - Increase instruction throughput
  - Doesn't reduce instruction latency?
  - How does this relate to performance measures?
    - » MIPS, Execution Time
- ◆ **When does the next instruction depend on the current one?**
  - Uses value produced by current instruction
  - Current instruction is a branch
  - Otherwise, No problem!

CS 141 Chien

12

Feb 22, 2000

## Pipelined Execution



CS 141 Chien

13

Feb 22, 2000

## Pipeline Hazards

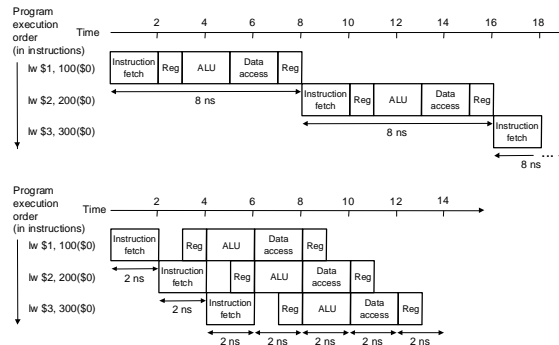
- ◆ *There must be problems and pitfalls*
- ◆ *Structural Hazard*
  - Hardware cannot support two instructions simultaneously
    - » e.g. either wash or dry, but not both
    - » read or write memory, but not both
- ◆ *Control Hazard*
  - Decision made by partially executed instruction affects currently loading instruction
    - » May execute wrong code if branch taken
    - » Later: Branch Prediction, Delayed Branching
- ◆ *Data Hazard*
  - Current instruction depends on output of incomplete instruction ahead of it in the pipeline

CS 141 Chien

14

Feb 22, 2000

# Pipelining



- ◆ **Improve performance by increasing instruction throughput**  
*Ideal speedup is number of stages in the pipeline. Do we achieve this?*

CS 141 Chien

15

Feb 22, 2000

## Pipelining: Basics

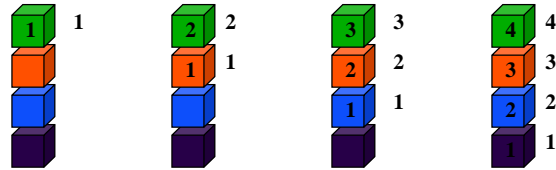
- ◆ **What makes it easy**
  - all instructions are the same length
  - just a few instruction formats
  - memory operands appear only in loads and stores
- ◆ **What makes it hard?**
  - structural hazards: suppose we had only one memory
  - control hazards: need to worry about branch instructions
  - data hazards: an instruction depends on a previous instruction
- ◆ **We'll build a pipeline and follow instructions through. Start to think about hazards**

CS 141 Chien

16

Feb 22, 2000

## Pipelining: Successive clock cycles



- ◆ Units of work (instructions) move forward at each clock cycle
- ◆ Start and finish in order
- ◆ *Throughput* =
- ◆ *Latency* =

CS 141 Chien

17

Feb 22, 2000

## Pipelining Instruction Execution

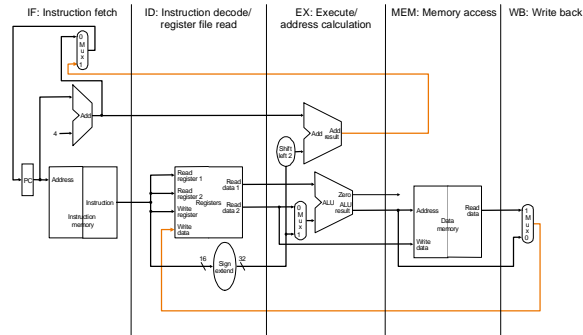
- ◆ Same idea as before
- ◆ Datapath pipeline will go left-> right
- ◆ Don't worry about dependencies between instructions for now
- ◆ Extension of the basic single cycle datapath from before
- ◆ Memory operations, Each "large" hardware unit is a pipeline stage
- ◆ Stages: IFetch, Decode/RR, Exec, Memory, WriteRegs

CS 141 Chien

18

Feb 22, 2000

# Single Cycle Control - Review



*What do we need to add to actually split the datapath into stages?*

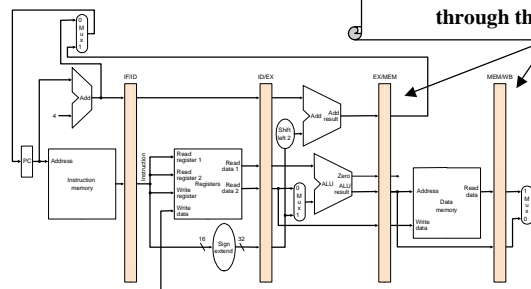
*Which reverse flows cause data/control hazards?*

CS 141 Chien

19

Feb 22, 2000

# Pipelined Datapath



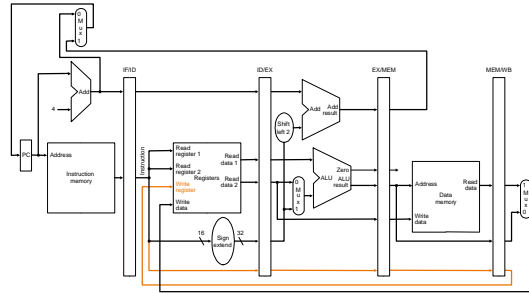
*Can you find a problem even if there are no dependencies? (Think about the registers?)*

CS 141 Chien

20

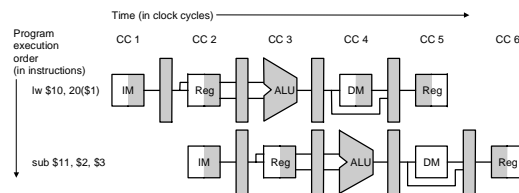
Feb 22, 2000

# Corrected Datapath



The *source/write registers* needs to be kept track of for each instruction.

# Graphically Representing Pipelines (Multiple-clock-cycle diagram)



Can help with answering questions like:

- how many cycles does it take to execute this code?
- what is the ALU doing during cycle 4?
- use this representation to help understand data paths

**Most recent instruction at bottom and moved right**

# Following instructions through pipelines (Single-Cycle Diagrams)

Let's follow three instructions through:

`lw $10, $20($1)`

`sub $11, $2, $3`

`add $5, $6, $7`

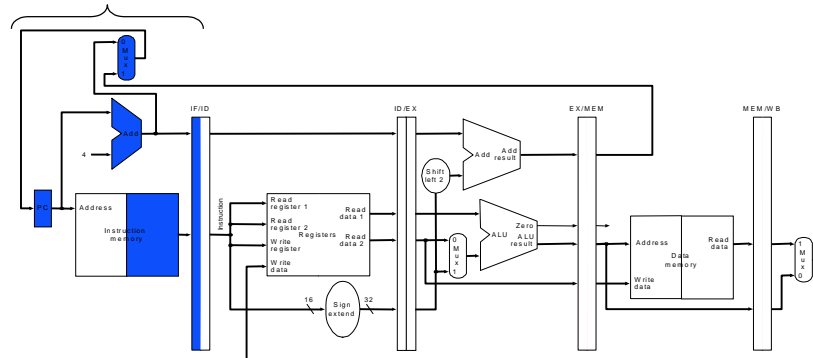
CS 141 Chien

23

Feb 22, 2000

## Clock cycle 1

Fetch: `lw $10, $20($1)`



◆ Instruction 1 in Instruction Fetch

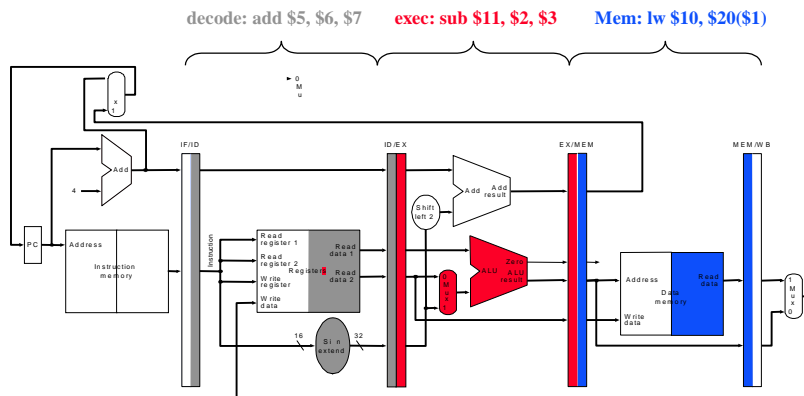
CS 141 Chien

24

Feb 22, 2000



## Clock Cycle 4



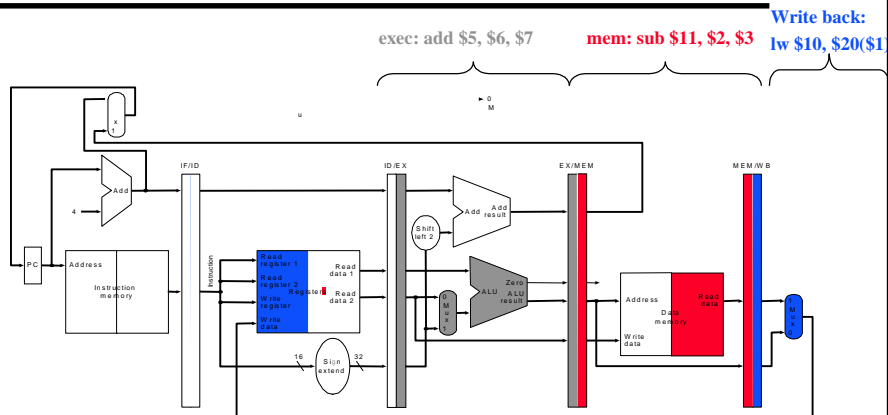
◆ 1 in memory, 2 in execute, 3 in decode

CS 141 Chien

27

Feb 22, 2000

## Clock Cycle 5



◆ 1 in write back, 2 in memory, 3 in execute

◆ Monotonous yet?

CS 141 Chien

28

Feb 22, 2000

## Notes about Pipelining

- ◆ All instructions move forward at same rate
- ◆ Start in order, complete in same order
- ◆ Clock rate, determined by longest stage
- ◆ **IF** no “dependencies” between instructions, can increase throughput rate by factor of the pipeline depth.
- ◆ Latency for each instruction is no better, probably a little worse
- ◆ Control is much more complicated...Next time.

CS 141 Chien

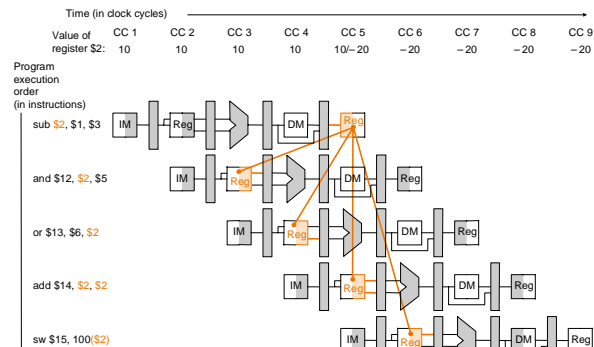
29

Feb 22, 2000

## Dependencies

- ◆ Problem with starting next instruction before first is finished

– dependencies that “go backward in time” are data hazards



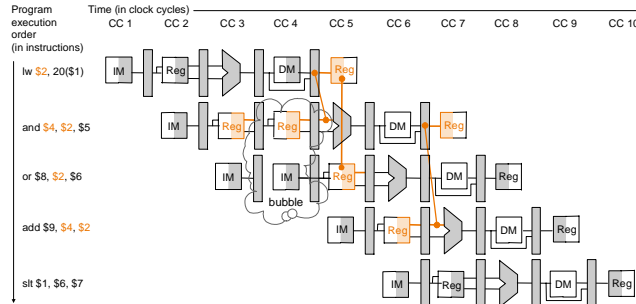
CS 141 Chien

30

Feb 22, 2000

# Stalling

- ◆ We can stall the pipeline by keeping an instruction in the same stage



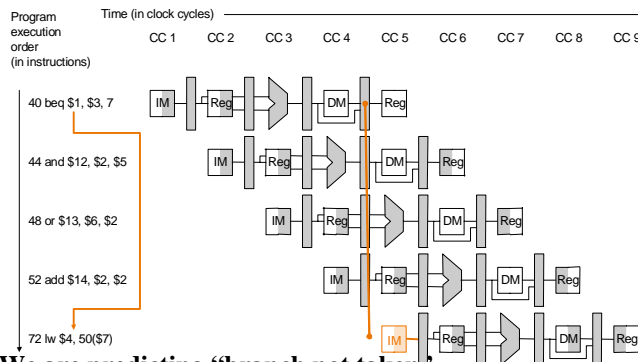
CS 141 Chien

31

Feb 22, 2000

# Branch Hazards

- ◆ When we decide to branch, other instructions are in the pipeline!



- ◆ We are predicting "branch not taken" need to add hardware for flushing instructions if we are wrong

CS 141 Chien

32

Feb 22, 2000

## Summary

---

- ◆ **Throughput and Latency**
- ◆ **Basics of Pipelining**
  - Increases throughput
  - Doesn't reduce latency
  - Can increase performance!
- ◆ **Pitfalls**
  - Structural Hazards
  - Control Hazards
  - Data Hazards
- ◆ **Next time: Pipelined Control**