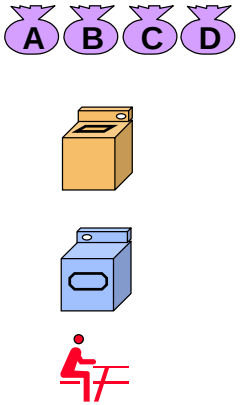
L10: Introduction to Pipelining

Adapted from:

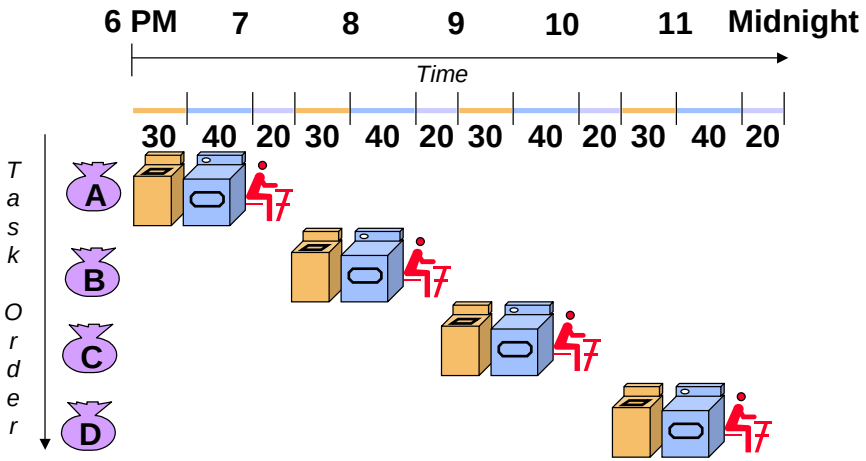
CS152: Computer Architecture and Engineering
Dave Patterson (www.cs.berkeley.edu/~pattrsn)

Pipelining is Natural!

- Laundry Example
- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold
- Washer takes 30 minutes
- Dryer takes 40 minutes
- "Folder" takes 20 minutes

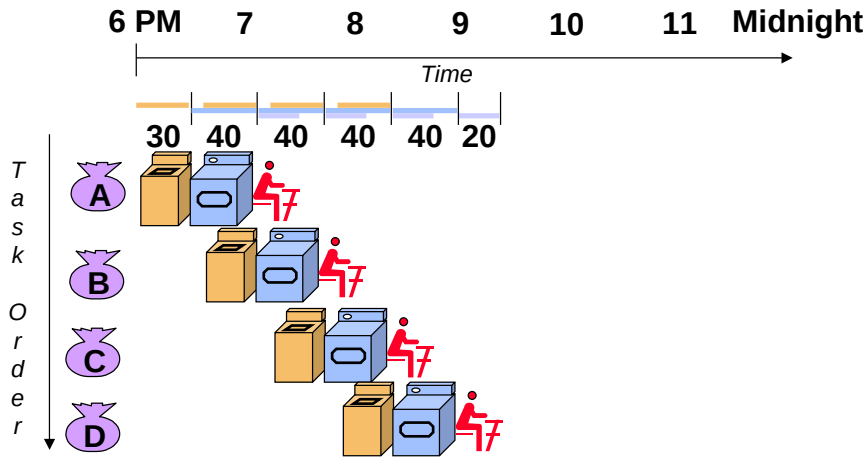


Sequential Laundry



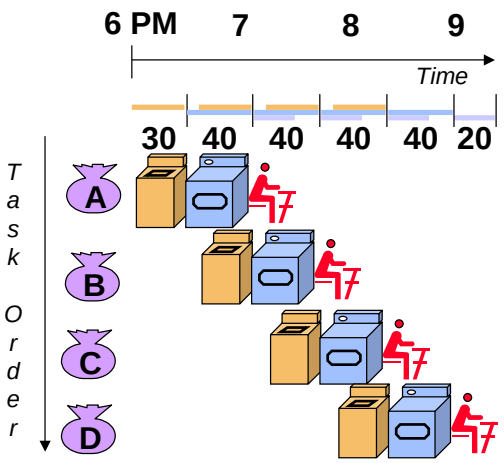
- Sequential laundry takes 6 hours for 4 loads
- How long would laundry take if they learnt pipelining?

Pipelined Laundry: Start work ASAP



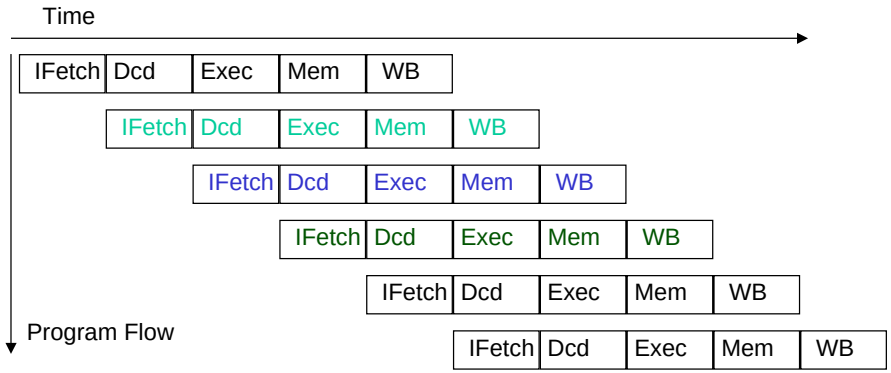
- Pipelined laundry takes 3.5 hours for 4 loads

Pipelining Lessons



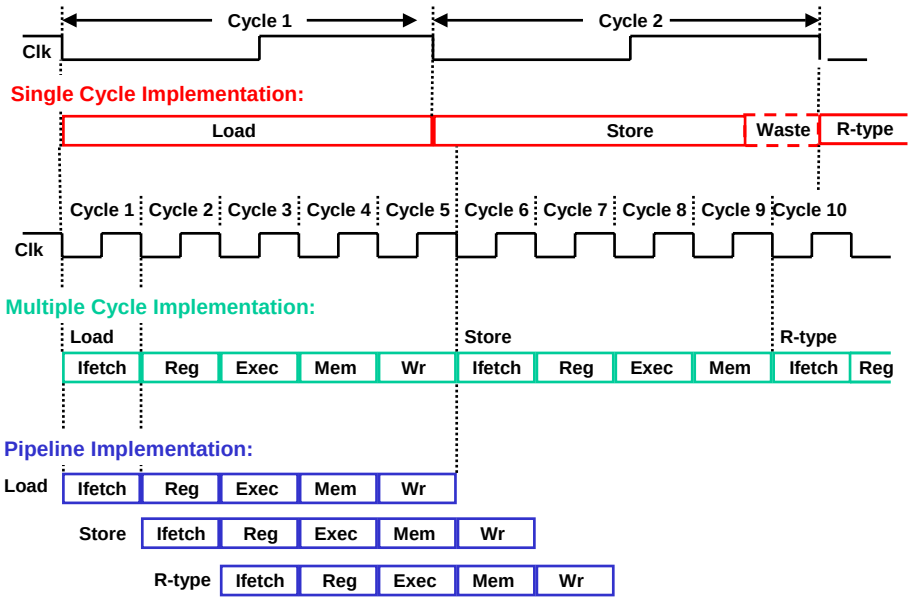
- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Pipeline rate limited by slowest pipeline stage
- Multiple tasks operating simultaneously using different resources
- Potential speedup = Number pipe stages
- Unbalanced lengths of pipe stages reduces speedup
- Time to "fill" pipeline and time to "drain" it reduces speedup
- Stall for Dependences

Pipelined Processor Execution



- Utilization?
- Now we just have to make it work

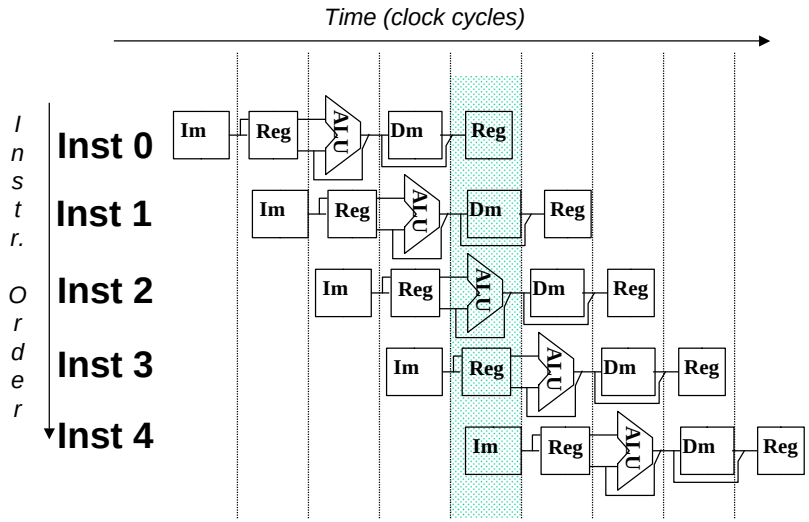
Single Cycle, Multiple Cycle, vs. Pipeline



Why Pipeline?

- Suppose we execute 100 instructions
- Single Cycle Machine
  - 45 ns/cycle x 1 CPI x 100 inst = 4500 ns
- Multicycle Machine
  - 10 ns/cycle x 4.1 CPI (due to inst mix) x 100 inst = 4100 ns
- Ideal pipelined machine
  - 10 ns/cycle x (1 CPI x 100 inst + 4 cycle drain) = 1040 ns

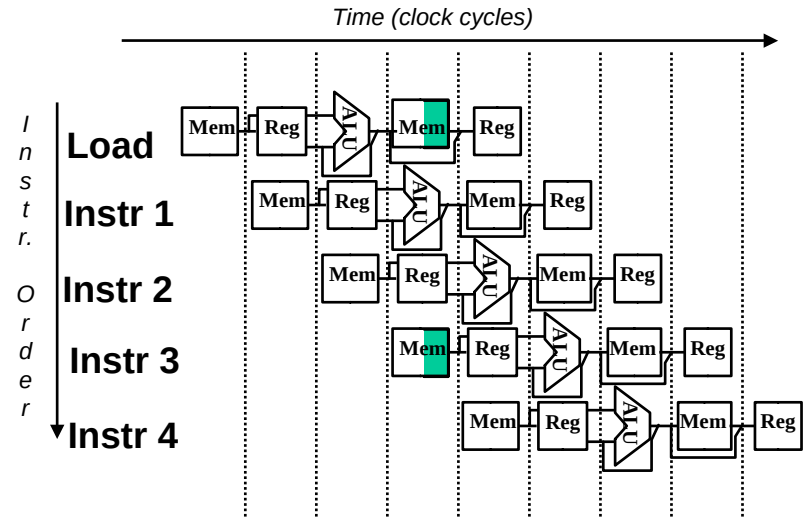
Why Pipeline? Because the resources are there!



Can pipelining get us into trouble?

- Yes: Pipeline Hazards
  - structural hazards: attempt to use the same resource two different ways at the same time
    - E.g., combined washer/dryer would be a structural hazard or folder busy doing something else (watching TV)
    - single memory for instructions and data cannot be accessed simultaneously
  - data hazards: attempt to use item before it is ready
    - E.g., one sock of pair in dryer and one in washer; can't fold until get sock from washer through dryer
    - instruction depends on result of prior instruction still in the pipeline
  - control hazards: attempt to make a decision before condition is evaluated
    - E.g., washing football uniforms and need to get proper detergent level; need to see after dryer before next load in
    - branch instructions
- Can always resolve hazards by waiting
  - pipeline control must detect the hazard
  - take action (or delay action) to resolve hazards

Single Memory is a Structural Hazard



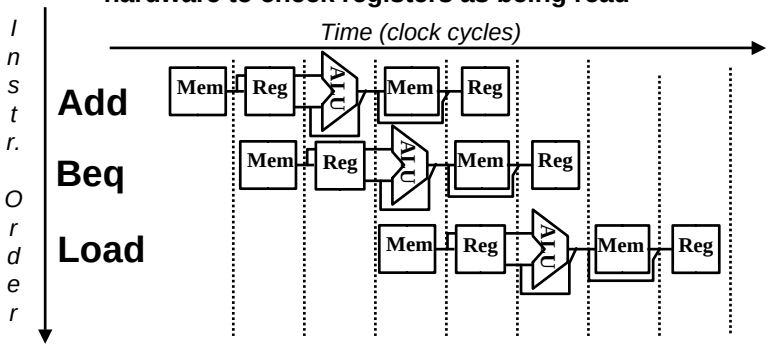
Detection is easy in this case! (right half highlight means read, left half write)

Structural Hazards limit performance

- Example: if 1.3 memory accesses per instruction and only one memory access per cycle then
  - average CPI ≥ 1.3
  - otherwise resource is more than 100% utilized
- Solution is to provide two memories

Control Hazard Solutions

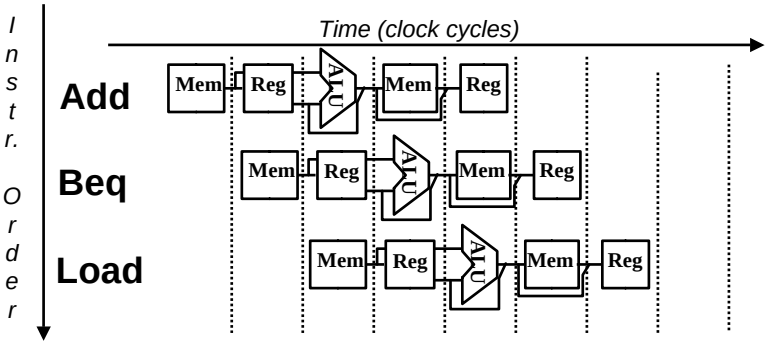
- **Stall: wait until decision is clear**
  - **It's possible to move up decision to 2nd stage by adding hardware to check registers as being read**



- **Impact: 2 clock cycles per branch instruction => increases CPI & slows down overall performance**

Control Hazard Solutions

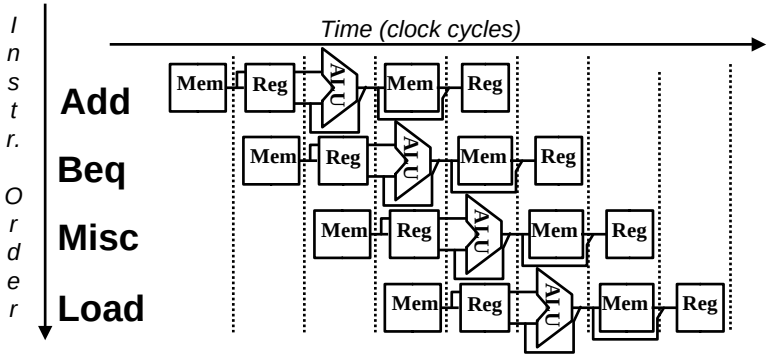
- **Predict: guess one direction then back up if wrong**
  - **Predict not taken**



- **Impact: 1 clock cycle per branch instruction if right, 2 if wrong (right - 50% of time)**
- **More dynamic scheme: history of 1 branch (~90%√)**

Control Hazard Solutions

- **Redefine branch behavior (takes place after next instruction) "delayed branch"**



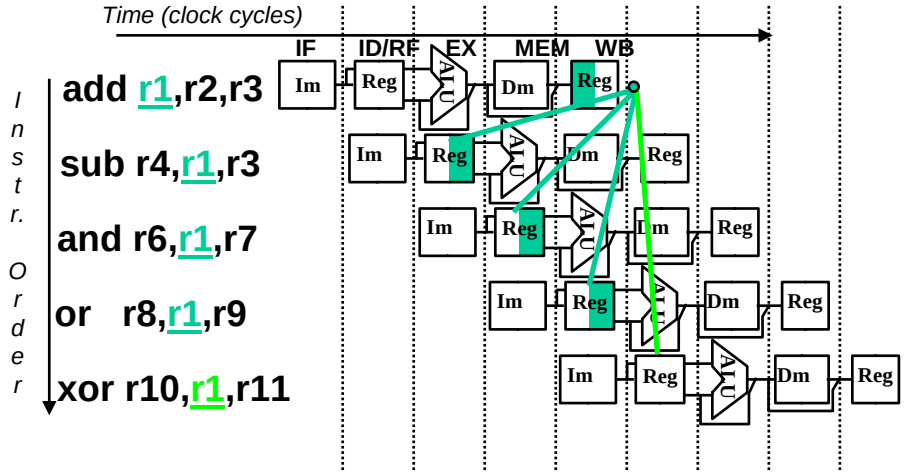
- **Impact: 1 clock cycles per branch instruction if can find instruction to put in "slot" (- 50% of time)**
- **As launch more instruction per clock cycle, less useful**

Data Hazard on r1

```
add r1 ,r2,r3      ; store result in reg r1
sub r4, r1 ,r3     ; read operand in reg r1
and r6, r1 ,r7
or  r8, r1 ,r9
xor r10, r1 ,r11
```

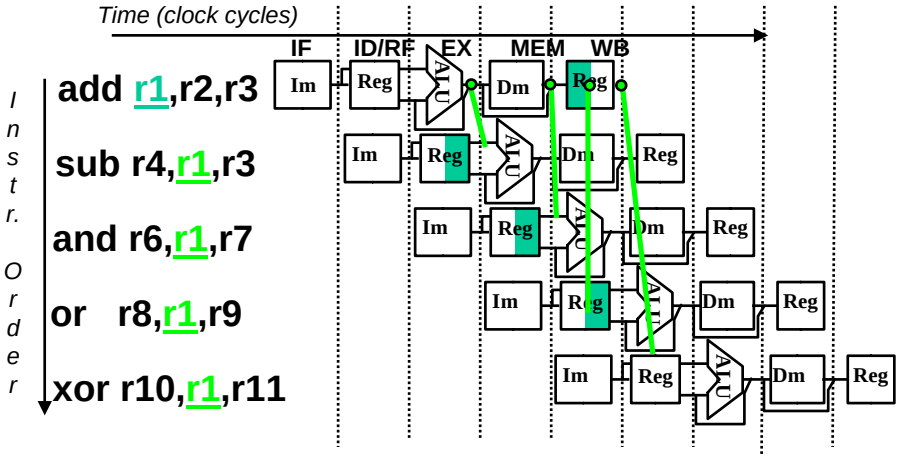
Data Hazard on r1:

Dependencies backwards in time are hazards



Data Hazard Solution:

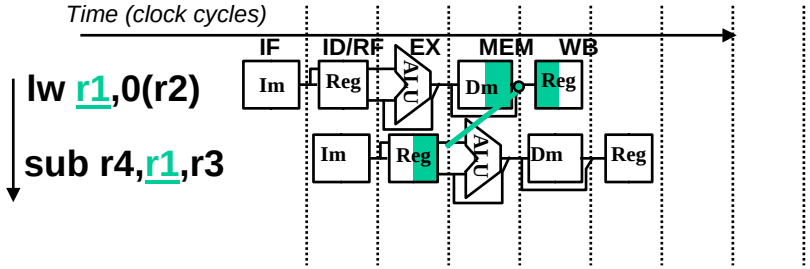
- **"Forward" result from one stage to another**



- **"OR" OK if define read/write properly**

Forwarding (or Bypassing): What about Loads

- **Dependencies backwards in time are hazards**



- **Can't solve with forwarding:**
  - **Must delay/stall instruction dependent on loads**

Next…

- **Look at design and control of pipelined datapath…**
- **Read Patterson & Hennessy, Ch 6**