Lecture 18: Pipelining I

Laundry Pipelining Example
- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, fold, and put away
- Washer takes 30 minutes
- Dryer takes 30 minutes
- "Folder" takes 30 minutes
- "Stasher" takes 30 minutes to put clothes into drawers

Sequential Laundry
- 6 PM 7 8 9 10 11 12 1 2 AM
- Task Order
- Washer, Dryer, Folder, Stasher
- Each task takes 30 minutes
- Sequential laundry takes 8 hours for 4 loads

Pipelined Laundry
- 6 PM 7 8 9 10 11 12 1 2 AM
- Task Order
- Washer, Dryer, Folder, Stasher
- Each task takes 30 minutes
- Pipelined laundry takes 3.5 hours for 4 loads!

General Definitions
- **Latency**: time to completely execute a certain task
  - for example, time to read a sector from disk is disk access time or disk latency
- **Throughput**: amount of work that can be done over a period of time

Pipelining Lessons (1/2)
- Pipelining doesn’t help latency of single task, it helps throughput of entire workload
- Multiple tasks operating simultaneously using different resources
- Potential speedup = Number pipe stages
- Time to “fill” pipeline and time to “drain” it reduces speedup: 2.3X v. 4X in this example
Pipelining I (Credit to Dan Garcia, UC Berkeley) (7) 

Pipelining Lessons (2/2)

• Suppose new Washer takes 20 minutes, new Stasher takes 20 minutes. How much faster is pipeline?

• Pipeline rate limited by slowest pipeline stage

• Unbalanced lengths of pipe stages also reduces speedup

Steps in Executing MIPS

1) Ifetch: Fetch Instruction, Increment PC
2) Decode Instruction, Read Registers
3) Execute:
   Mem-ref: Calculate Address
   Arith-log: Perform Operation
4) Memory:
   Load: Read Data from Memory
   Store: Write Data to Memory
5) Write Back: Write Data to Register

Graphical Pipeline Representation

(In Reg, right half highlight read, left half write)

Time (clock cycles)

Load
Add
Store
Sub
Or

Graphical Pipeline Representation

Example

• Suppose 2 ns for memory access, 2 ns for ALU operation, and 1 ns for register file read or write; compute instr rate

• Nonpipelined Execution:
  - lw: IF + Read Reg + ALU + Memory + Write
    Reg = 2 + 1 + 2 + 2 + 1 = 8 ns
  - add: IF + Read Reg + ALU + Write Reg
    = 2 + 1 + 2 + 1 = 6 ns

• Pipelined Execution:
  - Max(IF,Read Reg,ALU,Memory,Write Reg) = 2 ns

Review: Datapath for MIPS

• Use datapath figure to represent pipeline

PipeLine Execution Representation

Time

Ifetch Dcd Exec Mem WB

• Every instruction must take same number of steps, also called pipeline “stages”, so some will go idle sometimes

PipeLine Execution Representation

©

©

Fall 2004

Pipelining I (Credit to Dan Garcia, UC Berkeley) (8)

Steps in Executing MIPS

1) Ifetch: Fetch Instruction, Increment PC
2) Decode Instruction, Read Registers
3) Execute:
   Mem-ref: Calculate Address
   Arith-log: Perform Operation
4) Memory:
   Load: Read Data from Memory
   Store: Write Data to Memory
5) Write Back: Write Data to Register

Graphical Pipeline Representation

(In Reg, right half highlight read, left half write)

Time (clock cycles)

Load
Add
Store
Sub
Or

Graphical Pipeline Representation

Example

• Suppose 2 ns for memory access, 2 ns for ALU operation, and 1 ns for register file read or write; compute instr rate

• Nonpipelined Execution:
  - lw: IF + Read Reg + ALU + Memory + Write
    Reg = 2 + 1 + 2 + 2 + 1 = 8 ns
  - add: IF + Read Reg + ALU + Write Reg
    = 2 + 1 + 2 + 1 = 6 ns

• Pipelined Execution:
  - Max(IF,Read Reg,ALU,Memory,Write Reg) = 2 ns

Review: Datapath for MIPS

• Use datapath figure to represent pipeline

PipeLine Execution Representation

Time

Ifetch Dcd Exec Mem WB

• Every instruction must take same number of steps, also called pipeline “stages”, so some will go idle sometimes
Pipelining I (Credit to Dan Garcia, UC Berkeley) (13) Fall 2004

Pipeline Hazard: Matching socks in later load

A depends on D; stall since folder tied up

Limits to pipelining

- **Hazards** prevent next instruction from executing during its designated clock cycle
  - **Structural hazards**: HW cannot support this combination of instructions (single person to fold and put clothes away)
  - **Control hazards**: Pipelining of branches & other instructions stall the pipeline until the hazard; “bubbles” in the pipeline
  - **Data hazards**: Instruction depends on result of prior instruction still in the pipeline (missing sock)

Structural Hazard #1: Single Memory (1/2)

Read same memory twice in same clock cycle

Structural Hazard #1: Single Memory (2/2)

- **Solution:**
  - infeasible and inefficient to create second memory
  - so handle this by having **two Level 1 Caches** (a temporary smaller [of usually most recently used] copy of memory)
  - have both an L1 **Instruction Cache** and an L1 **Data Cache**
  - need more complex hardware to control when both caches miss

Structural Hazard #2: Registers (1/2)

Can’t read and write to registers simultaneously

Structural Hazard #2: Registers (2/2)

- **Fact**: Register access is **VERY** fast; takes less than half the time of ALU stage
- **Solution**: introduce convention
  - always Write to Registers during first half of each clock cycle
  - always Read from Registers during second half of each clock cycle
  - Result: can perform Read and Write during same clock cycle
Structural Hazard #3: Results not ready

To be continued.....

Things to Remember

• Optimal Pipeline
  • Each stage is executing part of an instruction each clock cycle.
  • One instruction finishes during each clock cycle.
  • On average, execute far more quickly.

• What makes this work?
  • Similarities between instructions allow us to use same stages for all instructions (generally).
  • Each stage takes about the same amount of time as all others: little wasted time.