CSCI-564 Advanced Computer Architecture

Lecture 6: Pipelining Review

Bo Wu
Colorado School of Mines
Wake up! Time to do laundry!
The Laundry Analogy

- Place one dirty load of clothes in the washer
- When the washer is finished, place the wet load in the dryer
- When the dryer is finished, take out the dry clothes and fold
- When folding is finished, ask your roommate (?) to put the clothes away
Pipelining Multiple Loads of Laundry

- 4 loads of laundry in parallel
- no additional resources
- throughput increased by 4
- latency per load is the same
Question: We have a four-stage pipeline. Every stage takes 1 hour. How long does it take to finish 100 loads?
Question: We have a four-stage pipeline. Every stage takes 1 hour. How long does it take to finish 100 loads?

Question: We have a four-stage pipeline. Every stage takes 1 hour. How long does it take to finish $N$ loads?
In Reality, maybe...

the slowest step decides throughput
Pipelining is Everywhere
Pipelining is Everywhere
An Ideal Pipeline

- All objects go through the same stages
- No sharing of resources between any two stages
- Propagation delay through all pipeline stages is equal
- Scheduling of a transaction entering the pipeline is not affected by the transactions in other stages
The Instruction Execution Cycle

1. Instruction fetch (IF)
2. Instruction decode and register operand fetch (ID/RF)
3. Execute/Evaluate memory address (EX/AG)
4. Memory operand fetch (MEM)
5. Store/writeback result (WB)
Unpipelined Datapath for MIPS

Diagram showing the unpipelined datapath for MIPS, including components such as PCSrc, Add, RegWrite, MemWrite, WBSrc, 0x4 Add, addr, inst, Memory, RegDst, ExtSel, OpCode, Ext, Imm, ALU, Control, ALU result, we, rs1, rs2, rd1, ws, wd, rd2, GPRs, Memory, Data, wdata, we, addr, zero?, br, rind, jabs, pc+4.
Simplified Unpipelined Datapath

Diagram showing the flow of data through various components like memory, ALU, and registers.
Clock period can be reduced by dividing the execution of an instruction into multiple cycles

\[ t_C > \max \{t_{\text{IM}}, t_{\text{RF}}, t_{\text{ALU}}, t_{\text{DM}}, t_{\text{RW}}\} \ (= t_{\text{DM}} \text{ probably}) \]
Clock period can be reduced by dividing the execution of an instruction into multiple cycles:

\[ t_C > \max \{ t_{IM}, t_{RF}, t_{ALU}, t_{DM}, t_{RW} \} \ (= t_{DM} \text{ probably}) \]
Clock period can be reduced by dividing the execution of an instruction into multiple cycles

\[ t_C > \max \{ t_{IM}, t_{RF}, t_{ALU}, t_{DM}, t_{RW} \} \quad ( = t_{DM} \text{ probably}) \]

However, CPI will increase unless instructions are pipelined
Technology Assumptions

- A small amount of very fast memory (caches) backed up by a large, slower memory
- Fast ALU (at least for integers)
- Multiported Register files (slower!)

Thus, the following timing assumption is reasonable

\[ t_{IM} \approx t_{RF} \approx t_{ALU} \approx t_{DM} \approx t_{RW} \]

A 5-stage pipeline will be the focus of our detailed design

- some commercial designs have over 30 pipeline stages to do an integer add!
We need some way to show multiple simultaneous transactions in both space and time.
Pipeline Diagrams: Transactions vs. Time

- **fetch phase**
  - PC
  - 0x4
  - Add
  - Inst. Memory
  - addr
  - rdata
  - IR

- **decode & register-fetch phase**
  - Hardwired Controller
  - rs1
  - rs2
  - rd1
  - ws
  - wd
  - rd2
  - GPRs
  - Imm Ext

- **execute phase**
  - ALU
  - we
  - wdata
  - rdata

- **memory phase**
  - Addr
  - Memory
  - wdata

- **write-back phase**
  - write

### Instruction Timeline

- **time**
  - instruction1
  - IF₁
  - t0
  - ID₁
  - t1
  - EX₁
  - t2
  - MA₁
  - t3
  - WB₁
  - t4
  - instruction2
  - IF₂
  - t5
  - ID₂
  - t6
  - EX₂
  - t7
  - MA₂
  - t8
  - WB₂
  - t9
  - instruction3
  - IF₃
  - t10
  - ID₃
  - t11
  - EX₃
  - t12
  - MA₃
  - t13
  - WB₃
  - t14
  - instruction4
  - IF₄
  - t15
  - ID₄
  - t16
  - EX₄
  - t17
  - MA₄
  - t18
  - WB₄
  - t19
  - instruction5
  - IF₅
  - t20
  - ID₅
  - t21
  - EX₅
  - t22
  - MA₅
  - t23
  - WB₅
  - t24
  - ....