CS 61C: Great Ideas in Computer Architecture (Machine Structures)
Instruction Level Parallelism

Instructors:
Randy H. Katz
David A. Patterson
http://inst.eecs.Berkeley.edu/~cs61c/fa10

Agenda

• Review
• Pipelined Execution
• Pipelined Datapath
• Administrivia
• Pipeline Hazards
• Peer Instruction
• Summary

Review: Single-cycle Processor

• Five steps to design a processor:
  1. Analyze instruction set → datapath requirements
  2. Select set of datapath components & establish clock methodology
  3. Assemble datapath meeting the requirements
  4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.
  5. Assemble the control logic
      • Formulate Logic Equations
      • Design Circuits

Single Cycle Performance

• Assume time for actions are
  – 100ps for register read or write; 200ps for other events
• Clock rate is?

<table>
<thead>
<tr>
<th>Instr</th>
<th>Instr fetch</th>
<th>Register read</th>
<th>ALU op</th>
<th>Memory access</th>
<th>Register write</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td>200ps</td>
<td>100 ps</td>
<td>800ps</td>
</tr>
<tr>
<td>sw</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td>200ps</td>
<td>100 ps</td>
<td>700ps</td>
</tr>
<tr>
<td>R-format</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td>200ps</td>
<td>100 ps</td>
<td>600ps</td>
</tr>
<tr>
<td>beq</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td>200ps</td>
<td></td>
<td>500ps</td>
</tr>
</tbody>
</table>

• What can we do to improve clock rate?
• Will this improve performance as well?
  Want increased clock rate to mean faster programs
Gotta Do Laundry

- 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

- Sequential laundry takes 8 hours for 4 loads

Pipelined Laundry

- Pipelined laundry takes 3.5 hours for 4 loads!

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
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 reduces speedup

Steps in Executing MIPS

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

Redrawn Single Cycle Datapath

- Data Memory \( [R_{rs}] + \text{SignExt}[\text{imm16}] = R_{rt} \)

Single Cycle Datapath
Pipeline registers

1. Instruction Fetch
2. Decode/Register Read
3. Execute
4. Memory
5. Write Back

- Need registers between stages
  - To hold information produced in previous cycle

More Detailed Pipeline

IF for Load, Store, ...

ID for Load, Store, ...
EX for Load

MEM for Load

WB for Load

Corrected Datapath for Load

Wrong register number
Agenda

• Review
• Pipelined Execution
• Pipelined Datapath
• Administrivia
• Pipeline Hazards
• Peer Instruction
• Summary

Why both rt and rd as MIPS write reg?

- Need to have 2 part immediate if 2 sources and 1 destination always in same place

Administrivia

• Project 3: Thread Level Parallelism + Data Level Parallelism + Cache Optimization
  – Due Part 2 due Saturday 11/13
• Project 4: Single Cycle Processor in Logicsim
  – Due Part 2 due Saturday 11/27
  – Face-to-Face grading: Signup for timeslot last week
• Extra Credit: Fastest Version of Project 3
  – Due Monday 11/29 Midnight
• Final Review: TBD (Vote via Survey!)
• Final: Mon Dec 13 8AM-11AM (TBD)

Survey

• Hours/wk OK? avg 13, median 12-14 (4 units = 12 hours)
• Since picked earliest time for review, redoing to see if still Thu best (Mon vs Thu)
Computers in the News

- Giants win World Series! (4-1 over Dallas Texas Rangers)
- “S.F. Giants using tech to their advantage”
  – Therese Poletti, MarketWatch, 10/29/10
- “Giants were an early user of tech, and it looks like these investments are paying off.”
  – Bill Neukom (chief executive) @ Microsoft 25 years
- Scouts given cameras to upload video of prospects
- XO Sports Sportsmotion, which outfits players with sensors that measure everything they do: player development, evaluate talent, rehab after injury (swing changed?)
- Internal SW development team to mine data for scouting (other teams use standard SW packages)
- 266 Cisco Wi-Fi access points throughout park; 1st in 2004
- Voice over IP to save $ internally for SF Giants

Pipelined Execution Representation

<table>
<thead>
<tr>
<th>Time</th>
<th>IFetch</th>
<th>Dcd</th>
<th>Exec</th>
<th>Mem</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFetch</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IFetch</td>
<td>Dcd</td>
<td>Exec</td>
<td>Mem</td>
<td>WB</td>
<td></td>
</tr>
<tr>
<td>IFetch</td>
<td>Dcd</td>
<td>Exec</td>
<td>Mem</td>
<td>WB</td>
<td></td>
</tr>
<tr>
<td>IFetch</td>
<td>Dcd</td>
<td>Exec</td>
<td>Mem</td>
<td>WB</td>
<td></td>
</tr>
<tr>
<td>IFetch</td>
<td>Dcd</td>
<td>Exec</td>
<td>Mem</td>
<td>WB</td>
<td></td>
</tr>
<tr>
<td>IFetch</td>
<td>Dcd</td>
<td>Exec</td>
<td>Mem</td>
<td>WB</td>
<td></td>
</tr>
</tbody>
</table>

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

Graphical Pipeline Diagrams

- Use datapath figure below to represent pipeline

**Graphical Pipeline Representation**

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

| Time (clock cycles) |
|---------------------|----------------|
| Load                |                |
| Add                 |                |
| Store               |                |
| Sub                 |                |
| Or                  |                |

11/3/10
**Pipeline Performance**

- Assume time for stages is
  - 100ps for register read or write
  - 200ps for other stages
- What is pipelined clock rate?
  - Compare pipelined datapath with single-cycle datapath

<table>
<thead>
<tr>
<th>Instr</th>
<th>Inst fetch</th>
<th>Register read</th>
<th>ALU op</th>
<th>Memory access</th>
<th>Register write</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td>200ps</td>
<td>100 ps</td>
<td>800ps</td>
</tr>
<tr>
<td>sw</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td>200ps</td>
<td></td>
<td>700ps</td>
</tr>
<tr>
<td>R-format</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td></td>
<td>100 ps</td>
<td>600ps</td>
</tr>
<tr>
<td>beq</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td></td>
<td></td>
<td>500ps</td>
</tr>
</tbody>
</table>

**Pipeline Speedup**

- If all stages are balanced
  - i.e., all take the same time
  - Time between instructions_{pipelined} = Time between instructions_{nonpipelined} / Number of stages
- If not balanced, speedup is less
- Speedup due to increased throughput
  - Latency (time for each instruction) does not decrease

**Instruction Level Parallelism (ILP)**

- Another parallelism form to go with Request Level Parallelism and Data Level Parallelism
- RLP – e.g., Warehouse Scale Computing
- DLP – e.g., SIMD, Map Reduce
- ILP – e.g., Pipelined instruction Execution
- 5 stage pipeline => 5 instructions executing simultaneously, one at each pipeline stage
Hazards

- Situations that prevent starting the next instruction in the next cycle
- Structural hazards
  - A required resource is busy (roommate studying)
- Data hazard
  - Need to wait for previous instruction to complete its data read/write (pair of socks in different loads)
- Control hazard
  - Deciding on control action depends on previous instruction (how much detergent based on how clean prior load turns out)

Structural Hazards

- Conflict for use of a resource
- In MIPS pipeline with a single memory
  - Load/store requires data access
  - Instruction fetch would have to stall for that cycle
  - Would cause a pipeline “bubble”
- Hence, pipelined datapaths require separate instruction/data memories
  - Really separate L1 instruction cache and L1 data cache

Structural Hazard #1: Single Memory

Time (clock cycles)

<table>
<thead>
<tr>
<th>Instruction Order</th>
<th>Load</th>
<th>Instr 1</th>
<th>Instr 2</th>
<th>Instr 3</th>
<th>Instr 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IS</td>
<td>IS</td>
<td>IS</td>
<td>IS</td>
<td>IS</td>
</tr>
<tr>
<td></td>
<td>Reg</td>
<td>Reg</td>
<td>Reg</td>
<td>Reg</td>
<td>Reg</td>
</tr>
<tr>
<td></td>
<td>DS</td>
<td>DS</td>
<td>DS</td>
<td>DS</td>
<td>DS</td>
</tr>
<tr>
<td></td>
<td>Reg</td>
<td>Reg</td>
<td>Reg</td>
<td>Reg</td>
<td>Reg</td>
</tr>
</tbody>
</table>

Read same memory twice in same clock cycle

Structural Hazard #2: Registers (1/2)

Time (clock cycles)

<table>
<thead>
<tr>
<th>Instruction Order</th>
<th>sw</th>
<th>Instr 1</th>
<th>Instr 2</th>
<th>Instr 3</th>
<th>Instr 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IS</td>
<td>IS</td>
<td>IS</td>
<td>IS</td>
<td>IS</td>
</tr>
<tr>
<td></td>
<td>Reg</td>
<td>Reg</td>
<td>Reg</td>
<td>Reg</td>
<td>Reg</td>
</tr>
<tr>
<td></td>
<td>DS</td>
<td>DS</td>
<td>DS</td>
<td>DS</td>
<td>DS</td>
</tr>
<tr>
<td></td>
<td>Reg</td>
<td>Reg</td>
<td>Reg</td>
<td>Reg</td>
<td>Reg</td>
</tr>
</tbody>
</table>

Can we read and write to registers simultaneously?
Structural Hazard #2: Registers (2/2)

- Two different solutions have been used:
  1) RegFile access is *VERY* fast: takes less than half the time of ALU stage
     - Write to Registers during first half of each clock cycle
     - Read from Registers during second half of each clock cycle
  2) Build RegFile with independent read and write ports
- Result: can perform Read and Write during same clock cycle

Data Hazards

- An instruction depends on completion of data access by a previous instruction
  - `add $s0, $t0, $t1`
  - `sub $t2, $s0, $t3`

Forwarding (aka Bypassing)

- Use result when it is computed
  - Don’t wait for it to be stored in a register
  - Requires extra connections in the datapath

Load-Use Data Hazard

- Can’t always avoid stalls by forwarding
  - If value not computed when needed
  - Can’t forward backward in time!
Code Scheduling to Avoid Stalls

- Reorder code to avoid use of load result in the next instruction
- C code for \( A = B + E; \ C = B + F; \)

\[
\begin{align*}
\text{lw } &\$t1, 0($t0) \\
\text{lw } &\$t2, 4($t0) \\
&\text{add } \$t3, \$t1, \$t2 \\
&\text{sw } \$t3, 12($t0) \\
\text{lw } &\$t4, 8($t0) \\
&\text{add } \$t5, \$t1, \$t4 \\
&\text{sw } \$t5, 16($t0)
\end{align*}
\]

11 cycles

\[
\begin{align*}
\text{lw } &\$t1, 0($t0) \\
\text{lw } &\$t2, 4($t0) \\
&\text{add } \$t3, \$t1, \$t2 \\
&\text{sw } \$t3, 12($t0) \\
&\text{lw } \$t4, 8($t0) \\
&\text{add } \$t5, \$t1, \$t4 \\
&\text{sw } \$t5, 16($t0)
\end{align*}
\]

13 cycles

Peer Instruction

I. Thanks to pipelining, I have reduced the time it took me to wash my one shirt.

II. Longer pipelines are always a win (since less work per stage & a faster clock).

A)(red) I is True and II is True
B)(orange) I is False and II is True
C)(green) I is True and II is False
D)(yellow) I is False and II is False

Pipeline Summary

**The BIG Picture**

- Pipelining improves performance by increasing instruction throughput: exploits ILP
  - Executes multiple instructions in parallel
  - Each instruction has the same latency
- Subject to hazards
  - Structure, data, control