Computer Systems Organization

CSE 521/560M
Lecture 4
Prof. Patrick Crowley

Plan for Today

- VHDL preparation
- Questions
- Pipelining discussion 1
VHDL Preparation

- Prof. Zar’s ModelSim (in CEC) tutorial
  - [http://ge.ee.wustl.edu/dzar/tutorials/qv/qv_toc.html](http://ge.ee.wustl.edu/dzar/tutorials/qv/qv_toc.html)
  - (link on course web page)

- If you want to work from home, ask for advice on the course newsgroup. (We will get this sorted out)

---

Instruction Execution Cycle

- Loop forever:
  1. Fetch next instruction, increment PC
  2. Decode instruction
  3. Read operands
  4. Execute or compute memory address or compute branch address
  5. Store result or access memory or modify PC
Multi-cycle Implementation (no pipeline)

• Decompose process into equal subtasks
• For example (using MIPS ISA):
  1. Instruction fetch and increment PC (IF)
     \[
     \begin{align*}
     IR &\leftarrow \text{Mem}[PC]; \\
     NPC &\leftarrow PC + 4;
     \end{align*}
     \]
  2. Instruction decode and register fetch (ID)
     \[
     \begin{align*}
     A &\leftarrow \text{Regs}[rs]; \\
     B &\leftarrow \text{Regs}[rt]; \\
     \text{Imm} &\leftarrow \text{sign-extended immediate field of IR};
     \end{align*}
     \]

Cont’d

3. Execution/effective address cycle (EX)
   \[
   \begin{align*}
   \text{If ALU op:} & \quad \text{ALUOutput} \leftarrow A \text{ op } B; \\
   \text{Else if ALU op with immediate:} & \quad \text{ALUOutput} \leftarrow A \text{ op } \text{Imm}; \\
   \text{Else if memory reference:} & \quad \text{ALUOutput} \leftarrow A + \text{Imm}; \\
   \text{Else (if branch):} & \quad \text{ALUOutput} \leftarrow NPC + (\text{Imm} \ll 2); \\
   & \quad \text{Cond} \leftarrow (A == 0)
   \end{align*}
   \]
Cont’d

4. Memory access/branch completion cycle (MEM)
   PC ← NPC;
   If memory reference:
     LMD ← Mem[ALUOutput] #load
     Mem[ALUOutput] ← B; #store
   Else if branch:
     If (cond) PC ← ALUOutput;

Cont’d

5. Write-back cycle (WB)
   If ALU op:
     Regs[rd] ← ALUOutput;
   Else if ALU op with immediate:
     Regs[rt] ← ALUOutput;
   Else if load instruction:
     Regs[rt] ← LMD;
Unpipelined MIPS Data Path

Pipelining

- Our introduction to instruction-level parallelism (ILP)
- One instruction or result every cycle (ideal)
  - Not in practice due to hazards
- Increase throughput
  - Throughput = number of results/second
- Improve speed-up
  - In the ideal case, an \( n \) stage pipeline has nearly an \( n \)-fold speed-up (why can’t \( n \) be very large?)
- Instruction latency increases slightly
Basic Pipeline Implementation

- Add pipeline registers between the stages
- Our example: 5-stage MIPS pipeline
- Questions:
  - What gets stored in the pipeline registers?
  - How to design control unit?

Pipelined MIPS Data Path
Pipelining Visualization

Pipelining w/ Registers
Pipeline performance

- Increases instruction throughput
- Reduces CPI or clock cycle time (or both)
- Hazards create pipeline stalls

(Ideal) Speedup = \[
\frac{\text{pipeline depth}}{1}
\]

(Hazards) Speedup = \[
\frac{\text{pipeline depth}}{1 + \text{stalls per instruction}}
\]

Hazards

- Structural
  - Resource conflicts
- Data
  - Dependencies between instructions
- Control
  - Branches and other control flow disruptions
Structural Hazards

- Example: single instruction and data memory
  - Stall on each load-store instruction
- Solutions:
  - Instruction buffers
  - Separate I-cache and D-cache
  - Or, both plus a sophisticated fetch unit

Data Hazards

- Data dependencies between instructions that are in the pipeline at the same time
- Example: Read After Write (RAW)

  Add \( R_1, R_2, R_3 \) \#R1 holds result
  Sub \( R_4, R_1, R_5 \) \#use of R1
  And \( R_6, R_1, R_7 \) \#use of R1
  Or \( R_8, R_1, R_9 \) \#use of R1
  Xor \( R_{10}, R_1, R_{11} \) \#use of R1

  (WAW and WAR as well)
Data Hazards

Forwarding
Forwarding

- Idea: Feed pipeline register outputs back to earlier stages (and use as alternate inputs when a hazard exists)
- Cannot solve all conflicts
- Alternative: let the compiler do it (e.g., disallow certain sequences of instructions)
Control Hazards

• When do you know there is a branch?
  – At ID cycle
• When do you know the branch outcome?
  – At EXE cycle
• Easy solution:
  – Stall one cycle after the branch
  – Refetch the instruction
  – Cost: 2 cycles, effect on CPI?
• Better schemes to come (beyond delay slots)

Pipeline Control Unit

• Everything about instruction is known at ID stage
• Instruction is issued if it moves from ID to EXE stage
  – Need to insert bubble if the opcode in ID/EX corresponds to load and result register in ID/EX register = one of the IF/ID register sources. (All done in ID stage)
  – To insert a bubble, zero out all control fields in pipeline register
• For forwarding similar comparisons (more of them) and setup appropriate sources and control for multiplexers
  – Not difficult (in a simple pipeline) because all data and control info is carried along in pipeline registers
Assignment

- VHDL preparation
  - first VHDL assignment next week
- Readings
  - For Monday
    - H&P. Chapter 2, sections 2.11-2.16 (2.13 is optional)
    - H&P. Appendix A, sections A.3- A.11
  - For Wednesday
    - Commentary: A Comparison of Two Pipeline Organizations
    - Re-read Appendix A, section A.8