Plan for Today

- Announcement:
- Questions
- Pipelining discussion 1
  - Basic processor operation
  - Basic pipelining & implementation
  - Hazards
  - Forwarding
- Assignment
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)
     IR ← Mem[PC];
     NPC ← PC + 4;
  2. Instruction decode and register fetch (ID)
     A ← Regs[rs];
     B ← Regs[rt];
     Imm ← sign-extended immediate field of IR;
3. Execution/effective address cycle (EX)

*If \( ALU \text{ op} \):*

\[
\text{ALUOutput} \leftarrow A \text{ op } B;
\]

*Else if \( ALU \text{ op with immediate} \):*

\[
\text{ALUOutput} \leftarrow A \text{ op Imm};
\]

*Else if memory reference:*

\[
\text{ALUOutput} \leftarrow A + \text{Imm};
\]

*Else (if branch):*

\[
\text{ALUOutput} \leftarrow \text{NPC} + (\text{Imm} \ll 2);
\]

\[
\text{Cond} \leftarrow (A ==0)
\]
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;
5. Write-back cycle (WB)

If $ALU\ op$:

Regs[$rd$] $\leftarrow$ ALUOutput;

Else if $ALU\ op$ with immediate:

Regs[$rt$] $\leftarrow$ ALUOutput;

Else if load instruction:

Regs[$rt$] $\leftarrow$ LMD;
Unpipelined MIPS Data Path
Pipelining

- Our introduction to instruction-level parallelism (ILP)
- One instruction or result every cycle (ideal)
  - Not achieved 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 to move data and operate logic blocks correctly through time?
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 (fetch several at once)
  – 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)

\[
\begin{align*}
\text{Add} & \quad R1, R2, R3 \quad \text{#R1 holds result} \\
\text{Sub} & \quad R4, R1, R5 \quad \text{#use of R1} \\
\text{And} & \quad R6, R1, R7 \quad \text{#use of R1} \\
\text{Or} & \quad R8, R1, R9 \quad \text{#use of R1} \\
\text{Xor} & \quad R10, R1, R11 \quad \text{#use of R1}
\end{align*}
\]
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)
Forwarding isn’t Magic
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
  – Re-fetch the instruction
  – Cost: 2 cycles; effect on CPI?

• Better schemes to come (beyond delay slots)
Reducing Control Stalls
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 are made (more of them)

• 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

• Readings
  – For Monday
    • HW1 assigned, available on course calendar
    • Turner & Zar VHDL concepts tutorial (again if needed)

– For Wednesday
  • H&P: Appendix A, sections A3- A.5
  • Commentary: Retrospective on HLL Computers
    – submit commentary to newsgroup before class