MIPS ISA & Basic Pipelining

Administrivia

- Read Appendix A – Basic Pipelining
- Skim Appendix B – MIPS ISA reference
  - worth reading, but we’ll only touch on parts of it in lecture as needed

COMPUTER ARCHITECTURE VS. INSTRUCTION SET ARCHITECTURE

Example: MIPS

Programmable storage

2^32 x bytes
31 x 32-bit GPRs (R0=0)
32 x 32-bit FP regs (paired DP)
PC

Arithmetic logical
Add, AddU, Sub, SubU, And, Or, Xor, SLT, SLTU,
Addi, AddiU, SLTI, SLTIU, Addi, AddiU, ori, xor, sll, srl, sra

Memory Access
LB, LBU, LH, LHU, LW,
SB, SH, SW

Control
J, JAL, JR, JALR
BEq, BNE, BLEZ, BGTZ, BLTZ, BGEZ

Data types?
Format?
Addressing Modes?

Instruction Set Architecture: Critical Interface

Properties of a good abstraction
- Lasts through many generations (portability)
- Used in many different ways (generality)
- Provides convenient functionality to higher levels
- Permits an efficient implementation at lower levels

Instruction Set Architecture

"... the attributes of a [computing] system as seen by the programmer, i.e. the conceptual structure and functional behavior, as distinct from the organization of the data flows and controls the logic design, and the physical implementation."
- Amdahl, Blaauw, and Brooks, 1964

Organization of Programmable Storage
- Data Types & Data Structures: Encodings & Representations
- Instruction Formats
- Instruction (or Operation Code) Set
- Modes of Addressing and Accessing Data Items and Instructions
- Exceptional Conditions
ISA vs. Computer Architecture

- Old definition of computer architecture
  - instruction set design
- H&P’s view is computer architecture >> ISA
- Architect’s job much more than instruction set design; technical hurdles today more challenging than those in instruction set design
- Since instruction set design not where action is, some conclude computer architecture (using old definition) is not where action is
  - H&P disagree on conclusion
  - Agree that ISA not where action is (ISA in CA:AQA 4/e appendix)

Comp. Arch. is an Integrated Approach

- What really matters is the functioning of the complete system
  - hardware, runtime system, compiler, operating system, and application
  - In networking, this is called the “End to End argument”
- Computer architecture is not just about transistors, individual instructions, or particular implementations
  - E.g., Original RISC projects replaced complex instructions with a compiler + simple instructions

Computer Architecture is Design and Analysis

- Architecture is an iterative process:
  - Searching the space of possible designs
  - At all levels of computer systems
  - Creativity
  - Good Ideas
  - Mediocre Ideas
  - Bad Ideas
  - Cost / Performance

MIPS INSTRUCTION SET ARCHITECTURE

A "Typical" RISC ISA

- 32-bit fixed format instruction (4 formats)
- 32 32-bit GPR (R0 contains zero, DP take pair)
- 3-address, reg-reg arithmetic instruction
- Single address mode for load/store: base + displacement
  - no indirection
- Simple branch conditions
- Delayed branch
  - see: SPARC, MIPS, HP PA-Risc, DEC Alpha, IBM PowerPC, CDC 6600, CDC 7600, Cray-1, Cray-2, Cray-3

Example: MIPS

<table>
<thead>
<tr>
<th>Register-Register</th>
<th>31</th>
<th>26</th>
<th>25</th>
<th>21</th>
<th>20</th>
<th>16</th>
<th>15</th>
<th>11</th>
<th>10</th>
<th>6</th>
<th>5</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Op</td>
<td>R3</td>
<td>R2</td>
<td>R1</td>
<td>Rd</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Register-Immediate</th>
<th>31</th>
<th>26</th>
<th>25</th>
<th>21</th>
<th>20</th>
<th>16</th>
<th>15</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Op</td>
<td>R3</td>
<td>R2</td>
<td>Rd</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Branch</th>
<th>31</th>
<th>26</th>
<th>25</th>
<th>21</th>
<th>20</th>
<th>16</th>
<th>15</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Op</td>
<td>R3</td>
<td>R2</td>
<td>Rz/Ox</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Jump / Call</th>
<th>31</th>
<th>26</th>
<th>25</th>
<th>target</th>
</tr>
</thead>
<tbody>
<tr>
<td>Op</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Pipelining is not quite that easy!

- 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)
  - **Data hazards**: Instruction depends on result of prior instruction still in the pipeline (missing sock)
  - **Control hazards**: Caused by delay between the fetching of instructions and decisions about changes in control flow (branches and jumps).
### One Memory Port/Structural Hazards

(sa Similar to Figure A.5, Page A-15)

<table>
<thead>
<tr>
<th></th>
<th>Cycle 1</th>
<th>Cycle 2</th>
<th>Cycle 3</th>
<th>Cycle 4</th>
<th>Cycle 5</th>
<th>Cycle 6</th>
<th>Cycle 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr 1</td>
<td><img src="image1" alt="Diagram" /></td>
<td><img src="image2" alt="Diagram" /></td>
<td><img src="image3" alt="Diagram" /></td>
<td><img src="image4" alt="Diagram" /></td>
<td><img src="image5" alt="Diagram" /></td>
<td><img src="image6" alt="Diagram" /></td>
<td><img src="image7" alt="Diagram" /></td>
</tr>
<tr>
<td>Instr 2</td>
<td><img src="image8" alt="Diagram" /></td>
<td><img src="image9" alt="Diagram" /></td>
<td><img src="image10" alt="Diagram" /></td>
<td><img src="image11" alt="Diagram" /></td>
<td><img src="image12" alt="Diagram" /></td>
<td><img src="image13" alt="Diagram" /></td>
<td><img src="image14" alt="Diagram" /></td>
</tr>
<tr>
<td>Stall</td>
<td><img src="image15" alt="Diagram" /></td>
<td><img src="image16" alt="Diagram" /></td>
<td><img src="image17" alt="Diagram" /></td>
<td><img src="image18" alt="Diagram" /></td>
<td><img src="image19" alt="Diagram" /></td>
<td><img src="image20" alt="Diagram" /></td>
<td><img src="image21" alt="Diagram" /></td>
</tr>
<tr>
<td>Instr 3</td>
<td><img src="image22" alt="Diagram" /></td>
<td><img src="image23" alt="Diagram" /></td>
<td><img src="image24" alt="Diagram" /></td>
<td><img src="image25" alt="Diagram" /></td>
<td><img src="image26" alt="Diagram" /></td>
<td><img src="image27" alt="Diagram" /></td>
<td><img src="image28" alt="Diagram" /></td>
</tr>
</tbody>
</table>

How do you “bubble” the pipe?

### Speed Up Equation for Pipelining

\[
\text{CPI}_{\text{pipeline}} = \frac{\text{Ideal CPI}}{1 + \frac{\text{Average Stall cycles per Inst}}{\text{Ideal CPI}}}
\]

\[
\text{Speedup} = \frac{\text{Ideal CPI} \times \text{Pipeline depth}}{1 + \frac{\text{Pipeline stall CPI}}{\text{Ideal CPI} + \text{Pipeline depth}}}
\]

For simple RISC pipeline, ideal CPI = 1:

\[
\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \frac{\text{Pipeline stall CPI}}{\text{Ideal CPI} + \text{Pipeline depth}}}
\]

### Example: Dual-port vs. Single-port

- Machine A: Dual ported memory ("Harvard Architecture")
- Machine B: Single ported memory, but its pipelined implementation has a 1.05 times faster clock rate
- Ideal CPI = 1 for both
- Loads are 40% of instructions executed
  - \( \text{Speedup}_A = \frac{\text{Pipeline depth}}{1 + 0} \times \frac{\text{clock}_{\text{unpipe}}}{\text{clock}_{\text{pipe}}} \)
  - \( \text{Speedup}_A = \text{Pipeline Depth} \)
  - \( \text{Speedup}_B = \frac{\text{Pipeline depth}}{1 + 0.4 \times 1} \times \frac{\text{clock}_{\text{unpipe}}}{\text{clock}_{\text{unpipe}} / 1.05} \)
  - \( \text{Speedup}_B = \frac{\text{Pipeline Depth}}{1.4} \times 1.05 \)
  - \( \text{Speedup}_B = 0.75 \times \text{Pipeline Depth} \)
- Machine A is 1.33 times faster

\[
\begin{align*}
\text{Speedup}_A / \text{Speedup}_B &= \frac{\text{Pipeline depth}}{0.75 \times \text{Pipeline depth}} = 1.33
\end{align*}
\]

### Data Hazard on R1

*Figure A.6, Page A-16*  

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Time (clock cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>add r1, r2, r3</td>
<td><img src="image29" alt="Diagram" /></td>
</tr>
<tr>
<td>sub r4, r1, r3</td>
<td><img src="image30" alt="Diagram" /></td>
</tr>
<tr>
<td>and r6, r1, r7</td>
<td><img src="image31" alt="Diagram" /></td>
</tr>
<tr>
<td>or r8, r1, r9</td>
<td><img src="image32" alt="Diagram" /></td>
</tr>
<tr>
<td>xor r10, r1, r11</td>
<td><img src="image33" alt="Diagram" /></td>
</tr>
</tbody>
</table>

**Read After Write (RAW)**  
Instr\(_J\) tries to read operand before Instr\(_I\) writes it

\[
\begin{align*}
\text{I:} & \quad \text{add r1, r2, r3} \\
\text{J:} & \quad \text{sub r4, r1, r3}
\end{align*}
\]

- Caused by a “true / flow dependence” (in compiler nomenclature). This hazard results from an actual need for communication.

### Three Generic Data Hazards

- **Read After Write (RAW)**  
  Instr\(_J\) tries to read operand before Instr\(_I\) writes it

\[
\begin{align*}
\text{I:} & \quad \text{add r1, r2, r3} \\
\text{J:} & \quad \text{sub r4, r1, r3}
\end{align*}
\]

- Called an “anti-dependence” by compiler writers. This results from reuse of the name “r1”.

- Can’t happen in MIPS 5 stage pipeline because:
  - All instructions take 5 stages, and
  - Reads are always in stage 2, and
  - Writes are always in stage 5
Three Generic Data Hazards

- **Write After Write (WAW)**
  - Instr\(_i\) writes operand before Instr\(_j\) writes it.
  - Called an "output dependence" by compiler writers. This also results from the reuse of name "r1".
  - Can't happen in MIPS 5 stage pipeline because:
    - All instructions take 5 stages, and
    - Writes are always in stage 5
  - Will see WAR and WAW in more complicated pipes

---

**HW Change for Forwarding**

Figure A.23, Page A-37

---

**Forwarding to Avoid Data Hazard**

Figure A.7, Page A-18

---