CMSC 411
Computer Systems Architecture
Lecture 4
MIPS ISA & Basic Pipelining

COMPUTER ARCHITECTURE
VS. INSTRUCTION SET
ARCHITECTURE
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

Example: MIPS

<table>
<thead>
<tr>
<th>r0</th>
<th>r1</th>
<th>r31</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Programmable storage:
- $2^{32} \times$ bytes
- 31 x 32-bit GPRs (R0=0)
- 32 x 32-bit FP regs (paired DP)

Arithmetic logical:
- Add, AddU, Sub, SubU, And, Or, Xor, SLT, SLTU, AddI, AddIU, SLTI, SLTIU, AndI, OrI, XorI,
- SLL, SRL, SRA

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

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

32-bit instructions on word boundary
Instruction Set Architecture (ISA)

“... 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

SOFTWARE
-- 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
  – Other aspects of computer design called implementation
  – Insinuates implementation is uninteresting or less challenging

• 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)
Computer Architecture 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 & Analysis

Architecture is an iterative process:
- Searching the space of possible designs
- At all levels of computer systems

Creativity

Cost / Performance Analysis

Good Ideas

Mediocre Ideas

Bad Ideas
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**

**Register-Register**

<table>
<thead>
<tr>
<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>rs</td>
<td>rt</td>
<td>rd</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>opx</td>
</tr>
</tbody>
</table>

rd ← rs OP rt

**Register-Immediate**

<table>
<thead>
<tr>
<th>31</th>
<th>26</th>
<th>25</th>
<th>21</th>
<th>20</th>
<th>16</th>
<th>15</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>rs</td>
<td>rt</td>
<td></td>
<td></td>
<td></td>
<td>immediate</td>
<td></td>
</tr>
</tbody>
</table>

rt ← rs OP immed

**Jump / Call**

<table>
<thead>
<tr>
<th>31</th>
<th>26</th>
<th>25</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td></td>
<td>target</td>
<td></td>
</tr>
</tbody>
</table>

---

**5 Steps of MIPS Datapath**

1. **Instruction Fetch**
   - IR ← mem[PC];
   - PC ← PC + 4

2. **Instr. Decode**
   - Reg[IR_{rd}] ← Reg[IR_{rs}] OP_{op} Reg[IR_{rt}]

3. **Execute Addr. Calc**
   - Next SEQ PC

4. **Memory Access**
   - Next PC

5. **Write Back**
   - WB Data

---

IMSIC 411 - 3 (from Patterson)
5 Steps of MIPS Datapath

Instruction Fetch

Intr. Decode

Reg. Fetch

Execute Addr. Calc

Memory Access

Write Back

IR ← mem[PC];
PC ← PC + 4
A ← Reg[IR\_rs];
B ← Reg[IR\_rt]
result ← A op_{IR\_op} B
WB ← result
Reg[IR\_rd] ← WB

Instruction Set Processor Controller

IR ← mem[PC];
PC ← PC + 4
A ← Reg[IR\_rs];
B ← Reg[IR\_rt]
result ← A op_{IR\_op} B
WB ← result
Reg[IR\_rd] ← WB
Visualizing Pipelining

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

How do you “bubble” the pipe?
**Speed Up Equation for Pipelining**

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

\[
\text{Speedup} = \frac{\text{Ideal CPI} \times \text{Pipeline depth}}{\text{Ideal CPI} + \text{Pipeline stall CPI}} \times \frac{\text{Cycle Time}_{\text{unpipelined}}}{\text{Cycle Time}_{\text{pipelined}}}
\]

For simple RISC pipeline, ideal CPI = 1:

\[
\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall CPI}} \times \frac{\text{Cycle Time}_{\text{unpipelined}}}{\text{Cycle Time}_{\text{pipelined}}}
\]

**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

SpeedUp_A = Pipeline Depth/(1 + 0) x (clock_{unpipe}/clock_{pipe})
= Pipeline Depth

SpeedUp_B = Pipeline Depth/(1 + 0.4 x 1) x (clock_{unpipe}/(clock_{unpipe} / 1.05))
= (Pipeline Depth/1.4) x 1.05
= 0.75 x Pipeline Depth

SpeedUp_A / SpeedUp_B = Pipeline Depth/(0.75 x Pipeline Depth) = 1.33

- Machine A is 1.33 times faster
Data Hazard on R1

Time (clock cycles)

IF ID/RF EX MEM WB

Instr Order

1. add \texttt{r1, r2, r3}
2. sub \texttt{r4, r1, r3}
3. and \texttt{r6, r1, r7}
4. or \texttt{r8, r1, r9}
5. xor \texttt{r10, r1, r11}

Three Generic Data Hazards

- **Read After Write (RAW)**
  
  Instr\textsubscript{J} tries to read operand before Instr\textsubscript{I} writes it

  \begin{align*}
  I: & \quad \text{add } \texttt{r1, r2, r3} \\
  J: & \quad \text{sub } \texttt{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

• **Write After Read (WAR)**
  Instr\(_J\) writes operand *before* Instr\(_I\) reads it.

  - I: \texttt{sub r4, r1, r3}
  - J: \texttt{add r1, r2, r3}
  - K: \texttt{mul r6, r1, r7}

  • 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\(_J\) writes operand *before* Instr\(_I\) writes it.

  - I: \texttt{sub r1, r4, r3}
  - J: \texttt{add r1, r2, r3}
  - K: \texttt{mul r6, r1, r7}

  • 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
Forwarding to Avoid Data Hazard

\begin{itemize}
  \item add \texttt{r1}, \texttt{r2}, \texttt{r3}
  \item sub \texttt{r4}, \texttt{r1}, \texttt{r3}
  \item and \texttt{r6}, \texttt{r1}, \texttt{r7}
  \item or \texttt{r8}, \texttt{r1}, \texttt{r9}
  \item xor \texttt{r10}, \texttt{r1}, \texttt{r11}
\end{itemize}

HW Change for Forwarding
Forwarding to Avoid LW-SW Data Hazard

Data Hazard Even with Forwarding
Data Hazard Even with Forwarding

Try producing fast code for

\[ a = b + c; \]
\[ d = e - f; \]

assuming \( a, b, c, d, e, \) and \( f \) in memory.

**Slow code:**

- `LW Rb,b`
- `LW Rc,c`
- `ADD Ra,Rb,Rc`
- `SW a,Ra`
- `LW Re,e`
- `SW RF,f`

**Fast code:**

- `LW Rb,b`
- `LW Rc,c`
- `ADD Ra,Rb,Rc`
- `SW a,Ra`
- `LW RF,f`

Compiler optimizes for performance. Hardware checks for safety.