INSTRUCTION SET ARCHITECTURE

Mahdi Nazm Bojnordi
Assistant Professor
School of Computing
University of Utah
Overview

- Announcement
  - Aug. 28th: Homework 1 release (due on Sept. 4th)
    - Verify your uploaded files before deadline

- This lecture
  - Power and energy
  - Instruction set architecture
Amdahl’s Law

- The law of diminishing returns

\[
\text{Execution time}_{\text{new}} = \text{Execution time}_{\text{old}} \times \left( (1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right)
\]

\[
\text{Speedup}_{\text{overall}} = \frac{\text{Execution time}_{\text{old}}}{\text{Execution time}_{\text{new}}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}
\]
Example Problem

- Our new processor is 10x faster on computation than the original processor. Assuming that the original processor is busy with computation 40% of the time and is waiting for IO 60% of the time, what is the overall speedup?
Example Problem

Our new processor is 10x faster on computation than the original processor. Assuming that the original processor is busy with computation 40% of the time and is waiting for IO 60% of the time, what is the overall speedup?

\[
f = 0.4 \quad s = 10 \\
\text{Speedup} = \frac{1}{0.6 + 0.4/10} = \frac{1}{0.64} = 1.5625
\]
Power and Energy
Power and Energy

- **Power = Voltage x Current** \( (P = VI) \)
  - Instantaneous rate of energy transfer (Watt)
- **Energy = Power x Time** \( (E = PT) \)
  - The cost of performing a task (Joule)
Power and Energy

- **Power** = Voltage x Current \( (P = VI) \)
  - Instantaneous rate of energy transfer (Watt)

- **Energy** = Power x Time \( (E = PT) \)
  - The cost of performing a task (Joule)

- Peak Power = 3W
- Average Power = 1.66W
- Total Energy = 5J
CPU Power and Energy

- All consumed energy is converted to heat
  - CPU power is the rate of heat generation
  - Excessive peak power may result in burning the chip

- Static and dynamic energy components
  - Energy = \((Power_{\text{Static}} + Power_{\text{Dynamic}}) \times \text{Time}\)
  - \(Power_{\text{Static}} = \text{Voltage} \times \text{Current}_{\text{Static}}\)
  - \(Power_{\text{Dynamic}} \propto \text{Capacitance} \times \text{Voltage}^2 \times (\text{Activity} \times \text{Frequency})\)
Power Reduction Techniques

- Reducing capacitance (C)
- Reducing voltage (V)
- Reducing frequency (f)

\[ f \rightarrow \text{C} \]
Power Reduction Techniques

- Reducing capacitance (C)
  - Requires changes to physical layout and technology
- Reducing voltage (V)
- Reducing frequency (f)
Power Reduction Techniques

- Reducing capacitance (C)
  - Requires changes to physical layout and technology

- Reducing voltage (V)
  - Negative effect on frequency
  - Opportunistically power gating (wakeup time)
  - Dynamic voltage and frequency scaling

- Reducing frequency (f)
Power Reduction Techniques

- Reducing capacitance (C)
  - Requires changes to physical layout and technology

- Reducing voltage (V)
  - Negative effect on frequency
  - Opportunistically power gating (wakeup time)
  - Dynamic voltage and frequency scaling

- Reducing frequency (f)
  - Negative effect on CPU time
  - Clock gating in unused resources

. 
Power Reduction Techniques

- Reducing capacitance \((C)\)
  - Requires changes to physical layout and technology

- Reducing voltage \((V)\)
  - Negative effect on frequency
  - Opportunistically power gating (wakeup time)
  - Dynamic voltage and frequency scaling

- Reducing frequency \((f)\)
  - Negative effect on CPU time
  - Clock gating in unused resources

- Points to note
  - Utilization directly effects dynamic power
  - Lowering power does NOT mean lowering energy
Example Problem

For a processor running at 100% utilization and consuming 60W, 30% of the power is attributed to leakage. What is the total power dissipation when the processor is running at 50% utilization?
Example Problem

- For a processor running at 100% utilization and consuming 60W, 30% of the power is attributed to leakage. What is the total power dissipation when the processor is running at 50% utilization?

- @100%
  - Power = 18W + 42W = 60W

- @50%
  - Power = 18W + 21W = 39W
A processor consumes 80W of dynamic power and 20W of static power at 3GHz. It completes a program in 20 seconds. What is the energy consumption if frequency scales down by 20%?
Example Problem

- A processor consumes 80W of dynamic power and 20W of static power at 3GHz. It completes a program in 20 seconds. What is the energy consumption if frequency scales down by 20%?

  - @3GHz
    - Energy = (80W + 20W) x 20s = 2000J

  - @2.4GHz
    - Energy = (0.8x80W + 20W) x 20/0.8 = 2100J
Example Problem

- A processor consumes 80W of dynamic power and 20W of static power at 3GHz. It completes a program in 20 seconds. What is the energy consumption if frequency scales down by 20%?
- What is the energy consumption if voltage and frequency scale down by 20%?
Example Problem

- A processor consumes 80W of dynamic power and 20W of static power at 3GHz. It completes a program in 20 seconds. What is the energy consumption if frequency scales down by 20%?
- What is the energy consumption if voltage and frequency scale down by 20%?

@ 80%V and 80%f

Energy = (80x0.8^2x0.8+20x0.8) x 20/0.8 = 1424J
Instruction Set Architecture
What is ISA?

- Instruction Set Architecture
  - Well-defined interfacing contract between hardware and software
  - Does define
    - The functional operations of units
    - How to use each functional unit
  - Does not define
    - How functional units are implemented
    - Execution time of operations
    - Energy consumption of operations
Example Problem

Which one may be guaranteed by an ISA?
- The number of instructions supported by processor
- The number of multipliers used by processor
- The width of operands
- Sequence of instructions that results in an error
- Sequence of instructions that results in lower energy consumption
- The total number of instructions for an application program
- The total amount of main memory (e.g., DRAM)
Example Problem

Which one may be guaranteed by an ISA?

- The number of instructions supported by processor: YES
- The number of multipliers used by processor: NO
- The width of operands: YES
- Sequence of instructions that results in an error: YES
- Sequence of instructions that results in lower energy consumption: NO
- The total number of instructions for an application program: NO
- The total amount of main memory (e.g., DRAM): NO
ISA to Programmer Interface

- Internal machine states
  - Architectural registers, control registers, program counter
  - Memory and page table
- Operations
  - Integer and floating-point operations
  - Control flow and interrupts
- Addressing modes
  - Immediate, register-based, and memory-based
ISA Types

- Operand locations

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Stack</th>
<th>Accumulator</th>
<th>Register-memory</th>
<th>Register-register/load-store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Push A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Push B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load R1,A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load R2,B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add R3,R1,B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add R3,R1,R2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Store C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Store R3,C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Store R3,C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Diagram of processor and memory configurations:
  - Processor: TOS → ALU
  - Memory: Stack, Accumulator, Register-memory, Register-register/load-store
Which Set of Instructions?

- ISA influences the execution time
  - CPU time = IC x CPI x CT
- Complex Instruction Set Computing (CISC)
- Reduced Instruction Set Computing (RISC)
Which Set of Instructions?

- ISA influences the execution time
  - CPU time = IC x CPI x CT

- Complex Instruction Set Computing (CISC)
  - May reduce IC, increase CPI, and increase CT
  - CPU time may be increased

- Reduced Instruction Set Computing (RISC)
  - May increases IC, reduce CPI, and reduce CT
  - CPU time may be decreased
<table>
<thead>
<tr>
<th>RISC ISA</th>
</tr>
</thead>
<tbody>
<tr>
<td>□ Simple operations</td>
</tr>
<tr>
<td>‣ Simple and fast FU</td>
</tr>
<tr>
<td>□ Fixed length</td>
</tr>
<tr>
<td>‣ Simple decoder</td>
</tr>
<tr>
<td>□ Limited inst. formats</td>
</tr>
<tr>
<td>‣ Easy code generation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CISC ISA</th>
</tr>
</thead>
<tbody>
<tr>
<td>□ Complex operations</td>
</tr>
<tr>
<td>‣ Costly memory access</td>
</tr>
<tr>
<td>□ Variable length</td>
</tr>
<tr>
<td>‣ Complex decoder</td>
</tr>
<tr>
<td>□ Limited registers</td>
</tr>
<tr>
<td>‣ Hard code generation</td>
</tr>
</tbody>
</table>
Memory Addressing

- Register
  - Add r4, r3

- Immediate
  - Add r4, #3

- Displacement
  - Add r4, 100(r1)

- Register indirect
  - Add r4, (r1)
Memory Addressing

- **Register**
  - Add r4, r3 \( \text{Reg}[4] = \text{Reg}[4] + \text{Reg}[3] \)

- **Immediate**
  - Add r4, #3 \( \text{Reg}[4] = \text{Reg}[4] + 3 \)

- **Displacement**
  - Add r4, 100(r1) \( \ldots + \text{Mem}[100 + \text{Reg}[1]] \)

- **Register indirect**
  - Add r4, (r1) \( \ldots + \text{Mem}[\text{Reg}[1]] \)
Memory Addressing

- Indexed
  - Add r3, (r1+r2)

- Direct
  - Add r1, (1001)

- Memory indirect
  - Add r1, @(r3)

- Auto-increment
  - Add r1, (r2)+
Memory Addressing

- **Indexed**
  - `Add r3, (r1+r2) ... + Mem[Reg[1]+Reg[2]]`

- **Direct**
  - `Add r1, (1001) ... + Mem[1001]`

- **Memory indirect**
  - `Add r1,@(r3) ... + Mem[Mem[Reg[3]]]`

- **Auto-increment**
  - `Add r1, (r2) ... + Mem[Reg[2]]`
Memory Addressing

- Auto-decrement
  - Add r1, -(r2)

- Scaled
  - Add r1, 100(r2)[r3]
Memory Addressing

- **Auto-decrement**
  - Add r1, -(r2)
  - $\text{Reg}[2] = \text{Reg}[2] - d$
  - $\ldots + \text{Mem}[\text{Reg}[2]]$

- **Scaled**
  - Add r1, 100(r2)[r3]
  - $\ldots + \text{Mem}[100 + \text{Reg}[2] + \text{Reg}[3] \times d]$
Example Problem

- Find the effective memory address
  - Add r2, 200(r1)
  - Add r2, (r1)
  - Add r2, @(r1)

<table>
<thead>
<tr>
<th>Registers</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>100</td>
</tr>
<tr>
<td>r2</td>
<td>200</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>100</td>
<td>400</td>
</tr>
<tr>
<td>200</td>
<td>500</td>
</tr>
<tr>
<td>300</td>
<td>600</td>
</tr>
<tr>
<td>400</td>
<td>700</td>
</tr>
<tr>
<td>500</td>
<td>800</td>
</tr>
</tbody>
</table>
Example Problem

- Find the effective memory address
  - Add r2, 200(r1)
    - \( r2 = r2 + \text{Mem}[300] \)
  - Add r2, (r1)
    - \( r2 = r2 + \text{Mem}[100] \)
  - Add r2, @(r1)
    - \( r2 = r2 + \text{Mem}[400] \)

<table>
<thead>
<tr>
<th>Registers</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>100</td>
</tr>
<tr>
<td>r2</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>100</td>
<td>400</td>
</tr>
<tr>
<td>200</td>
<td>500</td>
</tr>
<tr>
<td>300</td>
<td>600</td>
</tr>
<tr>
<td>400</td>
<td>700</td>
</tr>
<tr>
<td>500</td>
<td>800</td>
</tr>
</tbody>
</table>
Instruction Format

- A guideline for generating/interpreting instructions
- Example: MIPS
  - Fixed size 32-bit instructions
  - Three opcode types
    - I-type: load, store, conditional branch
    - R-type: ALU operations
    - J-type: jump