INSTRUCTION SET ARCHITECTURE

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

- Announcement
  - Sept. 5\textsuperscript{th}: Homework 1 release (due on Sept. 12\textsuperscript{th})

- This lecture
  - Instruction set architecture (ISA)
  - RISC vs. CISC
  - Memory addressing
  - Instruction format
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?
  - YES □ The number of instructions supported by processor
  - NO □ The number of multipliers used by processor
  - YES □ The width of operands
  - YES □ Sequence of instructions that results in an error
  - NO □ 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)
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**

![Diagram showing processor and memory with different operand locations: Stack, Accumulator, Register-memory, Register-register/load-store.]

<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>Store C</td>
<td></td>
<td></td>
<td>Add R3,R1,R2</td>
</tr>
<tr>
<td>Add</td>
<td></td>
<td>Store R3,C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pop C</td>
<td></td>
<td></td>
<td>Store R3,C</td>
<td></td>
</tr>
<tr>
<td>Load A</td>
<td></td>
<td></td>
<td>Load R1,A</td>
<td></td>
</tr>
<tr>
<td>Load R1,A</td>
<td></td>
<td></td>
<td></td>
<td>Load R2,B</td>
</tr>
<tr>
<td>Load R2,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>
</tbody>
</table>
Which Set of Instructions?

- ISA influences the execution time
  - \[ \text{CPU time} = \text{IC} \times \text{CPI} \times \text{CT} \]
- 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
RISC vs. SISC

- Simple operations
  - Simple and fast FU
- Fixed length
  - Simple decoder
- Limited inst. formats
  - Easy code generation

- Complex operations
  - Costly memory access
- Variable length
  - Complex decoder
- Limited registers
  - Hard code generation
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)  \[ ... + \text{Mem}[100 + \text{Reg}[1]] \]

- Register indirect
  - Add r4, (r1)  \[ ... + \text{Mem}[\text{Reg}[1]] \]
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) \( \ldots + \text{Mem}[\text{Reg}[1]+\text{Reg}[2]] \)

- Direct
  - Add r1, (1001) \( \ldots + \text{Mem}[1001] \)

- Memory indirect
  - Add r1, @(r3) \( \ldots + \text{Mem}[	ext{Mem}[\text{Reg}[3]]] \)

- Auto-increment
  - Add r1, (r2)+ \( \ldots + \text{Mem}[\text{Reg}[2]] \)
Memory Addressing

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

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

- **Auto-decrement**
  - Add r1, -(r2)
    - ... + Mem[Reg[2]]

- **Scaled**
  - Add r1, 100(r2)[r3]
Example Problem

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

<table>
<thead>
<tr>
<th>Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
</tr>
<tr>
<td>r2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
<tr>
<td>100</td>
</tr>
<tr>
<td>200</td>
</tr>
<tr>
<td>300</td>
</tr>
<tr>
<td>400</td>
</tr>
<tr>
<td>500</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>...</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