CMPT 295
Unit - Instruction Set Architecture
Lecture 23 – Design and evaluation of ISA
Last Lecture

- Instruction Set Architecture (ISA)
  A formal specification of ...
  - Registers and Memory model
  - Set of instructions
  - Set of data types
  - Conventions
  - etc...

- ISA design
  - Each instruction must have an unambiguous encoding
  - Functionally complete (Turing complete)
  - Instruction sets may be large (CISC) or small (RISC)
  - Design principles when creating set of instructions
Today’s Menu

- Instruction Set Architecture (ISA)
  - Definition of ISA
  - ISA design
  - ISA evaluation
    - Strategies to improve our ISA

- Execution of machine instructions
  - Intro to logic design + Sequential logic circuit
  - Sequential execution of machine instructions
  - Pipelined execution of machine instructions
    - Hazards
Let’s design an instruction format for an ISA (cont’d)

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Bit pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>padding</td>
<td>0000</td>
</tr>
<tr>
<td>ADD</td>
<td>0001</td>
</tr>
<tr>
<td>SUB</td>
<td>0010</td>
</tr>
<tr>
<td>MUL</td>
<td>0011</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Evaluation of our ISA “\texttt{x295}”

- **sample C code:** \[ z = (x + y) \times (x - y) \]

### Memory Address:

<table>
<thead>
<tr>
<th>Memory Address of ( x )</th>
<th>Memory Address of ( y )</th>
<th>Memory Address of Result ( z )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Word 1  | Word 2  | Word 3  |
---------|---------|---------|
Memory address of \( x \) | Memory address of \( y \) | Memory address of Result \( z \) |
How good is our ISA?

- Performance is usually measured using time
  - If an ISA design results in faster program execution then it is better

- Since memory access is slow, in order to evaluate our ISA, we could count memory accesses (in bytes)
  - The fewer memory accesses our program makes, the faster it executes, the better it is

- Other criteria:
  - Number of instructions needed to represent C program
Why is memory access slower than register access?

- External memory access is the slowest than registers
  - Most constraining aspect of instruction execution
- Because of transfer rate limitation of bus between memory and CPU
- Known as The von Neumann Bottleneck
The von Neumann Bottleneck

- During fetch
  - Each instruction must be retrieved from memory
  - Each explicit operands must be retrieved from memory
    - Operand can be explicit or implicit

- During execute phase
  - Some instructions will read/write memory
Evaluation of our ISA – cont’d

- Sample C code: \[ z = (x + y) \times (x - y) \]
- Let’s count the memory accesses:

<table>
<thead>
<tr>
<th>fetch</th>
<th>execute</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD x, y, tmp1</td>
<td></td>
</tr>
<tr>
<td>SUB x, y, tmp2</td>
<td></td>
</tr>
<tr>
<td>MUL tmp1, tmp2, z</td>
<td></td>
</tr>
</tbody>
</table>

Total:
How to improve our ISA?

How to reduce the number of memory accesses

- Strategy 1
  - Introduce registers
    - This reduces instruction size
Strategy 1 - ISA with operand model “3 Operand - Registers”

- Memory model of the computer: \(2^m \times n = 2^{12} \times 16\)
- \(8 \times 16\)-bit registers -> \((r0 \leftrightarrow r7)\)
- \(8 \rightarrow 2^3\) so 3 bits to uniquely specify each register

<table>
<thead>
<tr>
<th>Instruction Set:</th>
<th>Semantic:</th>
<th>Opcode:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD rA, rB, rC</td>
<td>rC &lt;- rA + rB</td>
<td></td>
</tr>
<tr>
<td>SUB rA, rB, rC</td>
<td>rC &lt;- rA - rB</td>
<td></td>
</tr>
<tr>
<td>MUL rA, rB, rC</td>
<td>rC &lt;- rA * rB</td>
<td></td>
</tr>
<tr>
<td>COPY rA, rC</td>
<td>rC &lt;- rA</td>
<td></td>
</tr>
<tr>
<td>LOAD a, rC</td>
<td>rC &lt;- M[a]</td>
<td></td>
</tr>
<tr>
<td>STORE rA, c</td>
<td>M[c] &lt;- rA</td>
<td></td>
</tr>
</tbody>
</table>
Strategy 1 - ISA with operand model
“3 Operands - Registers” - cont’d

Format:

```
opcode | Dest | Src1 | Src2

<table>
<thead>
<tr>
<th>opcode</th>
<th>Dest</th>
<th>Src</th>
</tr>
</thead>
<tbody>
<tr>
<td>opcode</td>
<td>Src</td>
<td>Dest</td>
</tr>
</tbody>
</table>
```

Examples:
Evaluation of Strategy 1 - ISA with operand model “3 Operand - Registers” - cont’d

- Sample C code: \( z = (x + y) \times (x - y) \)
- Let's count the memory accesses:

<table>
<thead>
<tr>
<th>fetch</th>
<th>execute</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOAD x, r0</td>
<td></td>
</tr>
<tr>
<td>LOAD y, r1</td>
<td></td>
</tr>
<tr>
<td>ADD r0, r1, r2</td>
<td></td>
</tr>
<tr>
<td>SUB r0, r1, r3</td>
<td></td>
</tr>
<tr>
<td>MUL r2, r3, r5</td>
<td></td>
</tr>
<tr>
<td>STORE r5, z</td>
<td></td>
</tr>
</tbody>
</table>

Total: 13
How to improve our ISA?

How to reduce the number of memory accesses

- **Strategy 1**
  - Introduce registers
    - This reduces instruction size

- **Strategy 2**
  - Reduce the number of operands
    - This further reduces instruction size
Strategy 2 - ISA with operand model “2 Operand - Registers”

- Memory model of the computer: \(2^m \times n = 2^{12} \times 16\)
- 8 \times 16-bit registers -> \((r0 \leftrightarrow r7)\)
- 8 -> \(2^3\) so 3 bits to uniquely specify each register

<table>
<thead>
<tr>
<th>Instruction Set</th>
<th>Semantic:</th>
<th>Opcode:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD rA, rC</td>
<td>rC &lt;- rA + rC</td>
<td></td>
</tr>
<tr>
<td>SUB rA, rC</td>
<td>rC &lt;- rA - rC</td>
<td></td>
</tr>
<tr>
<td>MUL rA, rC</td>
<td>rC &lt;- rA * rC</td>
<td></td>
</tr>
<tr>
<td>COPY rA, rC</td>
<td>rC &lt;- rA</td>
<td></td>
</tr>
<tr>
<td>LOAD a, rC</td>
<td>rC &lt;- M[a]</td>
<td></td>
</tr>
<tr>
<td>STORE rA, c</td>
<td>M[c] &lt;- rA</td>
<td></td>
</tr>
</tbody>
</table>
Strategy 2 - ISA with operand model
“2 Operand - Registers” - cont’d

Format:

<table>
<thead>
<tr>
<th>opcode</th>
<th>Dest</th>
<th>Src</th>
</tr>
</thead>
</table>

bits:

<table>
<thead>
<tr>
<th>opcode</th>
<th>Dest</th>
<th>Src</th>
<th>Dest / Src</th>
</tr>
</thead>
</table>

Examples:

| | | | |
| | | | |
| | | | |
Evaluation of Strategy 2 - ISA with operand model “2 Operand - Registers” - cont’d

- Sample C code: \( z = (x + y) \times (x - y) \)
- Let’s count the memory accesses:

```
fetch                      execute 
To be completed in Assignment 7
```

Total:
Summary

- ISA design
  - Started with: ISA with operand model “3 Operand – Memory”

- ISA Evaluation
  - Examining the effect of the von Neumann bottleneck on our program by counting # of memory accesses

- Improvements:
  - Decreasing effect of von Neumann bottleneck by reducing the number of memory accesses
  - Strategy 1 -> Introduce registers (reduces instruction size)
    - ISA with operand model “3 Operand - Registers”
  - Strategy 2 -> Reduce the number of operands (ditto)
    - ISA with operand model “2 Operand - Registers”
Next Lecture

- Instruction Set Architecture (ISA)
  - Definition of ISA
  - ISA design
  - ISA evaluation
    - Strategies to improve our ISA
- Execution of machine instructions
  - Intro to logic design + Sequential logic circuit
  - Sequential execution of machine instructions
  - Pipelined execution of machine instructions
    - Hazards