CMPT 295
Unit - Instruction Set Architecture
Lecture 25 – Improving our ISA – Reducing instruction size
Last Lecture

- 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”
Recap of our instruction sets: x295, x295+, x295++

x295:
- Memory model of the computer
  - Size of external memory (RAM): $2^{12} \times 16$
  - Memory address: 12 bits
  - Word size: 16 bits
  - Number of registers: 0
- Instruction set
  - Maximum number of instructions: 16
  - Opcode size: 4 bits ($2^4 = 16$)
  - Operand Model: Memory, 3 operands, order: Dest, Src1, Src2
  - Memory addressing mode: Direct

- Instructions (so far):
  - ADD a, b, c
  - SUB a, b, c
  - MUL a, b, c
- Data size: 16 bits
Recap of our instruction sets: x295, x295+, x295++

x295+:
- Memory model of the computer
  - Size of external memory (RAM): $2^{12} \times 16$
  - Memory address: 12 bits
  - Word size: 16 bits
  - Number of registers: 8 x 16-bit registers
- Instruction set
  - Maximum number of instructions: 16
  - Opcode size: 4 bits ($2^4 = 16$)
  - Operand Model: Registers, 3 operands, order: Dest, Src1, Src2
  - Memory addressing mode: Direct

Instructions (so far):
- ADD rA, rB, rC
- SUB rA, rB, rC
- MUL rA, rB, rC
- COPY rA, rC
- LOAD a, rC
- STORE rA, c

Data size: 16 bits
Recap of our instruction sets: x295, x295+, x295++

x295++:
- Memory model of the computer
  - Size of external memory (RAM): \(2^{12} \times 16\)
  - Memory address: 12 bits
  - Word size: 16 bits
  - Number of registers: 8 x 16-bit registers
- Instruction set
  - Maximum number of instructions: 16
  - Opcode size: 4 bits (\(2^4 = 16\))
  - Operand Model: Registers, 2 operands, order: Src, Dest
  - Memory addressing mode: Direct

Instructions (so far):
- ADD rA, rC
- SUB rA, rC
- MUL rA, rC
- COPY rA, rC
- LOAD a, rC
- STORE rA, c

Data size: 16 bits
Today’s Menu

- Instruction Set Architecture (ISA)
  - Definition of ISA
  - ISA design
  - ISA evaluation
    - Improving our ISA -> Our goal is to decrease the effect of von Neumann bottleneck
      - Strategy 3: Introduce other types of operands
- Execution of machine instructions
  - Intro to logic design
  - Sequential execution of machine instructions
  - Pipelined execution of machine instructions
How to improve our ISA?

How to reduce the number of memory accesses

➤ Strategy 1
  ➤ Introduce registers
    ➤ This reduces instruction size
    ➤ This reduces memory accesses during Fetch and Execute steps

➤ Strategy 2
  ➤ Reduce the number of operands
    ➤ This further reduces instruction size

➤ Strategy 3
  ➤ Introduce other types of operands
    ➤ This can also reduces instruction size
Back in Lecture 10 ... we talked about 3 types (modes) of operands to x86-64 assembly instructions

Operand (data) of an instruction can be
1. A constant integer data – **Immediate**
2. Data stored ... not in variables but in registers - **Register**
3. Data stored in **memory**

**Memory addressing modes**

- **Question:** How do we access memory in x86-64 assembly?
- **Answer:** Various "memory addressing modes"

1. Absolute (direct) ✓
2. Indirect
3. "Base + displacement"
4. 2 indexed
5. 4 scaled indexed

General Syntax: `Imm(r_b, r_i, s)`

Effect: `M[Imm + R[r_b] + R[r_i] * s]`

See Table of x86-64 Addressing Modes on Resources web page of our course web site
Our instruction sets x295, x295+, x295++ ...

... so far, contain

- **Register mode**
  - Operand is the register (indicated by its register id) containing the value to be used in the execution of the instruction

- **Memory addressing mode -> Direct mode (or Absolute)**
  - Operand is the memory address of the value to be used in the execution of the instruction
Adding **immediate mode** to x295++

- **Defn:** Operand is the actual value to be used in the execution of instruction
  - Example: In x86-64, $0x400, $-533

- **2-operand instructions:**
  - Data manipulation instructions: `ADD $value, rC` **Meaning:** `rC ← value + rC`
  - Data transfer instructions: `COPY $value, rC` **Meaning:** `rC ← value`

- **Format/Template:**
  - | Opcode | Dest | Src (value) |
  - | 15 | 1211 | 98 |
  - 4 bits 3 bits

- **Size of operands?**
- **Range of value of immediate operand?**
  - Limited to 9-bit size -> $2^9 = [-256 to 255]$
  - Copy: `$-256, rC`
  - What to do with the other 7 bits in register (registers are 16 bit wide)
  - Sign extension: copy bit 8 to bits 9 to 15
    - `rC` 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
    - **sign bit**
Adding *indirect mode* to x295++

- **Defn**: Operand is a register containing the memory address of the value to be used in the execution of the instruction.

- **2-operand instructions**: 
  - **LOAD** `a, rC`  
  - **STORE** `ra, (rc)`

- **Data transfer instructions**: 
  - **LOAD** `(ra), rC`  
  - **STORE** `ra, (rc)`

- **Format/Template**:
  - ```
    | Opcode | Dest | Src | XXXXXXXXXXXXXX |
    |--------|------|-----|----------------|
    | 15     | 211  | 98  | 65             |
  ```

- **Size of operands?**

**ADV**: This mode helps reduce size of instructions.
Adding base + displacement mode to x295++

- Defn: Operand is a register containing the base memory address and the value is the displacement
  - The sum of this value and the base memory address is the effective memory address which is the memory address of the value to be used in the execution of the instruction

- 2-operand instructions:
  - Data transfer instructions: LOAD value(rA), rC Meaning: rC ← M[rA + value]
    STORE rA, value(rC) Meaning: M[rC + value] ← rA

- Format/Template:
<table>
<thead>
<tr>
<th>Opcode</th>
<th>Dest</th>
<th>Src</th>
<th>Src (value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>12 11</td>
<td>98</td>
<td>65</td>
</tr>
</tbody>
</table>

- Size of operands?

- Range of displacement (value) operand?
  - Limited to 6-bit size → 2^6 = [-32 to 31]
Adding relative mode to x295++

- Defn: Operand is a displacement value to be added to the program counter, hence generating the memory address of the next instruction, should the branching occur

- 2-operand instructions: **BRANCH**
  - Program control instructions: BRle \( rA, \) label  
    - Meaning: if \( rA \leq 0 \), jump to label

- Format/Template:
  - 
    | Opcode | L-2 | Src | Label-1 (L-1) |
    |--------|-----|-----|---------------|
    | 15     | 1211| 98  | 65            |

- Size of operands?

- Range of label operand?
  - Limited to 9-bit size \( \Rightarrow 2^9 = [-256 \text{ to } 255] \)
  - Can only jump forwards/backwards by +/- 256
  - Handles most loops, ifs, etc, ...
Adding relative mode to x295++

- Defn: Operand is a displacement value to be added to the program counter, hence generating the memory address of the next instruction, should the branching occur.

- 2-operand instructions:
  - Program control instructions: `BRle rA, label`  
    `BRle rO, $250`  
    **Meaning:** if rA <= 0, jump to label

- Format/Template:
  - **Branch Instruction Diagram**

- Size of operands?

- Range of label operand?
  - Limited to 9-bit size -> $2^9 = [-256 \text{ to } 255]$
  - Can only jump forwards/backwards by +/- 256
  - Handles most loops, ifs, etc, ...
When BRle is executed
PC → 3
BRle (assuming rA <= 0)
→ branches to
memory address 253_{10}

M

253_{10}

... instruction

BRle 01110001 111010

Second

First instruction

label → 250_{10} = 0111110102

16 bits

L-2

L-1
x295++ ISA so far

- **Assembly instruction**
- **Format/Template**
- **Machine instruction**

```
COPY r0, r2
```

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Dest</th>
<th>Src</th>
<th>XXXXXXXXXXX</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>010</td>
<td>000</td>
<td>0000000000</td>
</tr>
</tbody>
</table>
Evaluation of our ISA

- x295++ with base + displacement mode
- sample C code: $z = (x + y) \cdot (x - y)$
- Let's count the memory accesses:

<table>
<thead>
<tr>
<th>Operation</th>
<th>fetch</th>
<th>execute</th>
</tr>
</thead>
<tbody>
<tr>
<td>COPY $x, r7</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>LOAD 0(r7), r1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>LOAD (y-x)(r7), r2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>COPY r2, r3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ADD r1, r2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SUB r1, r3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MUL r2, r3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>STORE r3, (z-x)(r7)</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Total: 8 + 3 = 11

Assumptions: • x, y and z are memory addresses at which we find values for x, y and z  • x is 9 bit wide
Evaluation of our ISA

- x295++ with base + displacement mode + immediate
- sample C code: $z = (x + y) \times \frac{7}{15} - \frac{8}{15}$
- Let's count the memory accesses:

Hand tracing our code!

<table>
<thead>
<tr>
<th>Fetch</th>
<th>Execute</th>
</tr>
</thead>
<tbody>
<tr>
<td>COPY $x$, r7 $\rightarrow$ memory address 10o</td>
<td>...</td>
</tr>
<tr>
<td>LOAD 0(r7), r1 $\rightarrow$ 7</td>
<td>13o</td>
</tr>
<tr>
<td>LOAD (y-x)(r7), r2 $\rightarrow$ 8</td>
<td>12o</td>
</tr>
<tr>
<td>COPY r2, r3 $\rightarrow$ 8</td>
<td>z = -15</td>
</tr>
<tr>
<td>ADD r1, r2 $\rightarrow$ 15</td>
<td>11o</td>
</tr>
<tr>
<td>SUB r1, r3 $\rightarrow$ -1</td>
<td>y = 8</td>
</tr>
<tr>
<td>MUL r2, r3 $\rightarrow$ -15</td>
<td>10o</td>
</tr>
<tr>
<td>STORE r3, (z-x)(r7)</td>
<td>x = 7</td>
</tr>
</tbody>
</table>

Total:

Assumptions:
- $x$, $y$, $z$ are memory addresses at which we find values for $x$, $y$, $z$
- $x$ is 9 bit wide
Summary

- How to improve our ISA
  - By decreasing the effect of von Neumann bottleneck
  - How do we do this?  
    -> By reducing the number of memory accesses
- Strategy 3
  - Introduce other types of operands
    - Immediate
    - Register
  - Memory addressing mode:
    - Absolute (direct)
    - Indirect
    - Base + displacement
    - Relative
- To keep instructions short, immediate and displacement values may have limited range
Next Lecture

- Instruction Set Architecture (ISA)
  - Definition of ISA
  - ISA design
  - ISA evaluation
    - Improving our ISA --> Decreasing effect of von Neumann bottleneck
      - 3 Strategies
- Execution of machine instructions
  - Intro to logic design
  - Sequential execution of machine instructions
  - Pipelined execution of machine instructions