Lecture topics

I Introduction to code generation
II Target language x64
III Code gen w/o register allocation

I Introduction to code gen

Stack vs. registers

<table>
<thead>
<tr>
<th>Stack</th>
<th>Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack (fixed set, e.g., 16)</td>
<td>(stat, static, etc.)</td>
</tr>
<tr>
<td>load</td>
<td>rax</td>
</tr>
<tr>
<td>store</td>
<td>rcrx</td>
</tr>
<tr>
<td>spill</td>
<td>rdx</td>
</tr>
<tr>
<td></td>
<td>dfsp</td>
</tr>
<tr>
<td></td>
<td>fssp</td>
</tr>
</tbody>
</table>

Addressing modes:
- Register
- Memory
- Immediate (constant)
- Add r,i | add r,r | add r,m | add m,i | add m,r
- but not:
- add m,m

Code generator has two tasks:
- Instruction selection
- Register allocation

Trade-offs:
- Simple, fast compiler
- Slow target code
- Complex, slow compiler
- Fast target code
- today's Dragon book
- Section 8.6
- JIT compiles
- AOT compiles

II Target language x64

Different kinds of assembly:

<table>
<thead>
<tr>
<th>IR</th>
<th>Assembly</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction Name</td>
<td>Name</td>
<td>Number</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Register or stack offset</td>
<td></td>
</tr>
<tr>
<td>Label Name</td>
<td>Name</td>
<td>Number</td>
</tr>
</tbody>
</table>

CISC

<table>
<thead>
<tr>
<th>Description</th>
<th>x64</th>
<th>ARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complex Instruction Set Computer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reduced Instruction Set Computer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stack (Virtual) machine</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RISC

<table>
<thead>
<tr>
<th>Example</th>
<th>Instruction Set Computer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java bytecode</td>
<td></td>
</tr>
</tbody>
</table>

Memory instructions

<table>
<thead>
<tr>
<th>Addresses per Instruction</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>Few</td>
</tr>
<tr>
<td>Memory instruction</td>
<td>Most</td>
</tr>
</tbody>
</table>

Trade-offs:

- Reading Figure 1
- Running Figure 1
- On energy[1...4], cims.nyu.edu
- Experimenting with gcc
- hello.c
- loop.c
- record.c
- call.c
III Code generation (p. 2)

- Preparation:
  - For each function:
    - For each variable or temporary X:
      - \( X\cdot\text{offset} = \text{new offset} \)
    - For each string constant S:
      - \( \text{Strings}[S] = \text{new label} \)

- Code generation:
  - Generate prologue
  - For each IR instruction:
    - Load operands
    - "Template" of x64 instructions
    - Store result
    - Generate epilogue

Examples:

\[
\text{IR} \quad x = y + z; \quad \xrightarrow{\text{x64}} \quad \text{mov} rax, [rbp - y.offset] \\
\quad \text{add} rax, [rbp - 2.offset] \\
\quad \text{mov} [rbp - x.offset], rax
\]

\[
\text{IR} \quad \text{if } x \text{ goto } l; \quad \xrightarrow{\text{x64}} \quad \text{cmp} [rbp - x.offset], 0 \\
\quad \text{je } l
\]

Arrays

- Stack
- Heap

\[
\text{IR} \quad x = y \cdot [i]; \quad \xrightarrow{\text{x64}} \quad \text{mov} rax, [rbp - y.offset] \\
\quad \text{mov} rdx, [rax + 8] \\
\quad \text{mov} rax, [rbp - z.offset] \\
\quad \text{sal} rax, 3 \\
\quad \text{add} rax, rdx \\
\quad \text{mov} rax, [rax] \\
\quad \text{mov} [rbp - x.offset], rax
\]