Lecture Topics

I. Introduction to code generation
II. Target language x64
III. Code generation without register allocation

I. Introduction to Code Generation

Symbol Table
Intermediate
Representation
(Tack P Cheney
2-address
virtual
registers)

Code generator

Target assembly code

Target binary code

Libraries

Compiler

Linked
together

gcc -m64 -march=intel main.s

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

Different kinds of assembly languages

<table>
<thead>
<tr>
<th>Description</th>
<th>CISC</th>
<th>RISC</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complex Instruction Set per computer</td>
<td>Reduced Instruction Set per computer</td>
<td>Stack (Virtual) Machine</td>
<td></td>
</tr>
</tbody>
</table>

Example: x64, ARM, Java Bytecode

Instructions

Many, powerful

Few, small

Addresses per ins
2
3
0

Registers

Few, non-uniform

Many, uniform

None

Memory

Many Instructions

Few

Stack: many

Few:

III. Target language x64

- See x64-intro.ppt
- Ready: Figure 1
- Running: Figure 1
- Experimenting with gcc
  - hello.c
  - loop.c
  - record.c
  - call.c

Trade-offs:

Simple, fast compiler

Slow target code

Complex, slow

Fast target code
Code generation without register allocation

For each function:
Preparation:
  for each variable or temporary x:
    \( x.\text{offset} = \text{new offset} \)
  for each string constant z:
    \( \text{string}[z] = \text{new label} \)

Code generation:
  generate prologue
  for each IR instruction
    load operands
    "template" of x64 instruction
    store result
  generate epilogue

Examples:

IR:
\[
x = y + z;
\]
\[
x64
\]
\[
\text{mov} \ \text{rax}, \ [\text{rip} - y.\text{offset}]
\]
\[
\text{add} \ \text{rax}, \ [\text{rip} - z.\text{offset}]
\]
\[
\text{mov} \ [\text{rip} - x.\text{offset}], \ \text{rax}
\]

IR:
\[
x = y[2]
\]
\[
x64
\]
\[
\text{mov} \ \text{rax}, \ [\text{rip} - y.\text{offset}]
\]
\[
\text{mov} \ \text{rdx}, \ [\text{rax} + 8]
\]
\[
\text{mov} \ \text{rax}, \ [\text{rip} - z.\text{offset}]
\]
\[
\text{sal} \ \text{rax}, \ 3
\]
\[
\text{add} \ \text{rax}, \ \text{rdx}
\]
\[
\text{mov} \ \text{rax}, \ [\text{rax}]
\]
\[
\text{mov} \ [\text{rip} - x.\text{offset}], \ \text{rax}
\]

Arrays

\[
\text{Stack} \rightarrow \text{Heap}
\]
\[
\begin{align*}
\text{Size} = n \\
\text{data}
\end{align*}
\]

\[
\text{IR:} \quad x = y[z]
\]
\[
\text{x64}
\]
\[
\text{mov} \ \text{rax}, \ [\text{rip} - y.\text{offset}]
\]
\[
\text{mov} \ \text{rdx}, \ [\text{rax} + 8]
\]
\[
\text{mov} \ \text{rax}, \ [\text{rip} - z.\text{offset}]
\]
\[
\text{sal} \ \text{rax}, \ 3
\]
\[
\text{add} \ \text{rax}, \ \text{rdx}
\]
\[
\text{mov} \ \text{rax}, \ [\text{rax}]
\]
\[
\text{mov} \ [\text{rip} - x.\text{offset}], \ \text{rax}
\]