Back-end missing pieces

Simone Campanoni
simone.campanoni@northwestern.edu
Instruction selection is part of the backend

Are these all the steps that modern compilers have in their back-ends? No
Register allocation after instruction selection

Total cost: 5

v1 *= 4
v2 ← v1
v2 += 5
v3 ← mem v1 0

A register allocation

Optimum!
Register allocation after instruction selection

lea (5+%v1*4), %v2
subq %v2, %v1
movq 0(%v1), %v3

A register allocation
v1 -> rax
v2 -> rbx
v3 -> stack O

lea (5+%rax*4), %rbx
subq %rbx, %rax
movq 0(%rax), %r10
movq %r10, O(%rsp)
Register allocation after instruction selection

lea (5+%v1*4), %v2
subq %v2, %v1
movq 0(%v1), %v3
movq %v3, %v4

A register allocation
v1 -> rax
v2 -> rbx
v3 -> stack O
v4 -> r8

lea (5+%rax*4), %rbx
subq %rbx, %rax
movq 0(%rax), %r10
movq %r10, O(%rsp)
movq O(%rsp), %r8

Peephole matching

Wait, I thought we found the optimum ...
Instruction selection is part of the backend

Instruction selection → Register allocation → Peephole matching → Assembly generation
Peephole matching

• Basic idea: compiler can discover local improvements locally
  • Look at a small set of adjacent operations
  • Move a “peephole” over code & search for improvement

• Example: store followed by load

\[
\begin{align*}
\text{movq} & \%r10, O(\%rsp) \\
\text{movq} & O(\%rsp), \%r8 \\
\end{align*}
\]

\[
\begin{align*}
\text{movq} & \%r10, O(\%rsp) \\
\text{movq} & \%r10, \%r8 \\
\end{align*}
\]
Are we happy now with the generated assembly?

Of course NOT!
The problem left

Instruction scheduling

Better schedule of instructions
Tracing and data layout → Instruction selection → Register allocation → Peephole matching → Instruction scheduling → Assembly generation → Assembly
Are these all the steps that modern compilers have in their back-ends? Almost
Legalizer

• Some instructions and/or data types might not be valid at lower-level representations (e.g., L3)
  • E.g., 8-bits variables in IR exist, but all variables in lower-level representation are only at 32 bits

• Solution 0: instruction selection
  • Advantage: no extra compilation step
  • Disadvantage: it makes a complex compilation step even more complex

• Solution 1: legalizer
  • Changes the code to replace data types and instructions that are “illegal” for the target architecture with “legal” data types and instructions
**Back-end**

1. **IR**
2. **Instruction and type legalizers**
3. **Tracing and data layout**
4. **Instruction selection**
5. **Instruction scheduling**
6. **Register allocation**
7. **Instruction scheduling**
8. **Peephole matching**
9. **Assembly generation**
10. **Assembly**
Tracing and data layout → Instruction selection → Instruction scheduling → Register allocation

Back-end:
- Instruction and type legalizers
- Peephole matching
- Assembly generation

Assembly
Always have faith in your ability

Success will come your way eventually

Best of luck!