Peephole Optimization

optimizing assembly

When generating code, can often create sequences of instructions that can be obviously optimized:

- Store followed by a load:

```
SW T1, 4(FP)
LW T2, 4(FP)
```

is the same as:

```
SW T1, 4(FP)
MV T2, T1
```

saving a load

optimizing assembly

When generating code, can often create sequences of instructions that can be obviously optimized:

- Address computation followed by a load:

```
ADDI T1, FP, 8
LW T2, 0(T1)
```

is the same as:

```
LW T2 8(FP)
```

saving an add

optimizing assembly

When generating code, can often create sequences of instructions that can be obviously optimized:

- Multiply by 8:

```
MULI T2, T1, 8
```

is the same as shifting by 3:

```
SLL T2, T1, 3
```

replacing an expensive multiply with a cheap shift

peephole optimization

- Optimizations that match patterns in assembly
  - Intuitively, look through a “peephole” at small sequences of instructions
  - If pattern matches, apply optimization
  - Lots of patterns: LLVM’s InstCombine pass has over 1000 optimizations!
- Can work at the assembly level (based on specific machine instructions) or at the 3AC level (simplifications based on mathematical equivalence)
  - Effectiveness is closely tied to what assembly instructions are available, and how expensive they are
  - Question: why might peephole optimizations be more or less effective for different machines?

more examples

- Constant folding

```
ADD Rx, LIT1, LIT2    ⟶    MOV Rx, LIT1 + LIT2
```

- Instruction selection

```
MOV Rx, LIT1
ADD Rz, Ry, Rx        ⟶    ADDI Rz, Ry, LIT1
```

- Null sequences

```
ADDI Rx, Ry, 0        ⟶    MOV Rx, Ry
```

more examples

- Branch swapping

```
BEQ Rx, Ry, L1
JMP L2
L1: ...
```

→

```
BNE Rx, Ry, L2
```

- Instruction selection

```
SUB Ry, zero, Rx
```

→

```
NEG Ry, Rx
```

Superoptimization

- Peephole optimization/instruction selection writ large
- Given a sequence of instructions, find a different sequence of instructions that performs the same computation in less time
- Huge body of research, pulling in ideas from all across computer science
  - Theorem proving
  - Machine learning
  - Program Synthesis

next: local optimization