Concepts Introduced in Chapter 9
- introduction to compiler optimizations
- basic blocks and control flow graphs
- local optimizations
- global optimizations

Compiler Optimizations
- Compiler optimization is a misnomer.
- A code-improving transformation consists of a sequence of changes that preserves the semantic behavior (i.e. are safe).
- A code-improving transformation attempts to make the program
  - run faster
  - take up less space
  - consume less power
- An optimization phase consists of a sequence of code-improving transformations of the same type.

Places for Potential Improvement
- source code
  - frontend
    - intermediate code
    - code generator
    - target code
  - user can: profile program change algorithm transform loops
  - compiler can: improve loops procedure calls address calculations
    - compiler can: use registers select instructions do peephole transformations

Basic Blocks
- Basic block - a sequence of consecutive statements with exactly 1 entry and 1 exit
- leaders are instructions that start a new basic block
  - the first three-address instruction in the intermediate code is a leader
  - any instruction that is the target of a conditional or unconditional jump is a leader
  - any instruction that immediately follows a conditional or unconditional jump is a leader
Control Flow

- Control flow graph - a directed graph where the nodes are basic blocks and block B→block C iff C can be executed immediately after B
  - there is a jump from the end of B to beginning of C
  - C follows B in program order
- B is a predecessor of C, and C is a successor of B
- Local optimizations - performed only within a basic block
- Global optimizations - performed across basic blocks

Example Control Flow Graph

Types of Compiler Optimizations

- Function call
- Loop
- Memory access
- Control flow
- Data flow
- Machine specific
**Function Call Optimizations**

- Procedure integration or inlining
- Procedure specialization or cloning
- Tail recursion elimination
- Function memoization

**Loop Optimizations**

- Invariant code motion
- Strength reduction
- Induction variable elimination
- Unrolling
- Collapsing
- Fusion
- Software pipelining

**Memory Access Optimizations**

- Register allocation
- Memory hierarchy improvement
  - Array padding
  - Scalar replacement
  - Loop interchange
  - Prefetching

**Control Flow Optimizations**

- Jump elimination
  - Branch chaining
  - Reversing branches
  - Code positioning
  - Loop inversion
  - Useless jump elimination
- Unreachable code elimination
Data Flow Optimizations

- Common subexpression elimination
- Partial redundancy elimination
- Dead assignment elimination
- Evaluation order determination
- Recurrence elimination

Machine-Specific Optimizations

- Instruction scheduling
- Filling delay slots
- Exploiting instruction-level parallelism
- Peephole optimization (includes instruction selection)

Optimizations after Code Generation

Instruction Selection

- Accomplished by combining RTLs.
- Data dependences (links) are detected between RTLs.
- Pairs or triples of RTLs are symbolically merged.
- Legality is checked via a machine description.
Combining a Pair of RTLs

26 \[ r[1] = r[30]+i; \]
27 \{26\} \[ r[2] = M[r[1]]; \] \[ r[1]: \]
\[ \Rightarrow \]
or
\[ r[2] = M[r[30]+i]; \] \[ r[1]: \]

Combining Three RTLs

31 \[ r[2] = M[r[3]]; \]
32 \{31\} \[ r[2] = r[2]+1; \]
33 \{32\} \[ M[r[3]] = r[2]; \] \[ r[2]: \]
\[ \Rightarrow \]
\[ M[r[3]] = M[r[3]]+1; \] \[ r[2] = M[r[3]]+1; \] \[ r[2]: \]
or
\[ M[r[3]] = M[r[3]]+1; \] \[ r[2]: \]

Cascading Instruction Selection

Actual example on PDP-11 (2 address machine)

38 \[ r[36]=r[5]; \]
39 \{38\} \[ r[36]=r[36]+i; \]
40 \[ r[37]=r[5]; \]
41 \{40\} \[ r[37]=r[37]+i; \]
42 \{41\} \[ r[40]=M[r[37]]; \] \[ r[37]: \]
43 \[ r[41]=1; \]
44 \{42\} \[ r[42]=r[40]; \] \[ r[40]: \]
45 \{43,44\} \[ r[42]=r[42]+r[41]; \] \[ r[41]: \]
46 \{45,39\} \[ M[r[36]]=r[42]; \] \[ r[42]:r[36]: \]

Cascading Instruction Selection (cont.)

38 \[ r[36]=r[5]; \]
39 \{38\} \[ r[36]=r[36]+i; \]
40 \[ r[37]=r[5]; \]
42 \{40\} \[ r[40]=M[r[37]+i]; \] \[ r[37]: \]
43 \[ r[41]=1; \]
44 \{42\} \[ r[42]=r[40]; \] \[ r[40]: \]
45 \{43,44\} \[ r[42]=r[42]+r[41]; \] \[ r[41]: \]
46 \{45,39\} \[ M[r[36]]=r[42]; \] \[ r[42]:r[36]: \]
Cascading Instruction Selection (cont.)

38 \( r[36]=r[5]; \)
39 \{38\} \( r[36]=r[36]+i; \)

42 \( r[40]=M[r[5]+i]; \)
43 \( r[41]=1; \)
44 \{42\} \( r[42]=r[40]; \quad r[40]: \)
45 \{43,44\} \( r[42]=r[42]+r[41]; \quad r[41]: \)
46 \{45,39\} \( M[r[36]]=r[42]; \quad r[42]:r[36]: \)

Cascading Instruction Selection (cont.)

38 \( r[36]=r[5]; \)
39 \{38\} \( r[36]=r[36]+i; \)

44 \( r[42]=M[r[5]+i]; \)
45 \{44\} \( r[42]=r[42]+1; \quad r[42]: \)
46 \{45,39\} \( M[r[36]]=r[42]; \quad r[42]:r[36]: \)
Cascading Instruction Selection (cont.)

44 \[ r[42]=M[r[5]+i]; \]
45 \{ 44 \} \[ r[42]=r[42]+1; \]
46 \{ 45 \} \[ M[r[5]+i]=r[42]; \]
\[ r[42]: \]

Example Sequence of Optimizations

for (sum=0, j = 0; j < n; j++)
sum = sum + a[j];

⇒ after instruction selection

\[ M[r[13]+\text{sum}] = 0; \]
\[ M[r[13]+j] = 0; \]
\[ PC = L18; \]
L19
\[ r[0] = M[r[13]+j] << 2; \]
\[ M[r[13] + \text{sum}] = M[r[13] + \text{sum}] + M[r[0] + \_a]; \]
L18
\[ IC = M[r[13] + j] \? M[\_n]; \]
\[ PC = IC < 0 \rightarrow L19; \]

Example Sequence of Optimizations (cont.)

⇒ after register allocation

\[ r[2] = 0; \]
\[ r[1] = 0; \]
\[ PC = L18; \]
L19
\[ r[0] = r[1] << 2; \]
\[ r[1] = r[1] + 1; \]
L18
\[ IC = r[1] \? M[\_n]; \]
\[ PC = IC < 0 \rightarrow L19; \]
Example Sequence of Optimizations (cont.)

⇒ after code motion

\[
\begin{align*}
r[2] &= 0; \\
r[1] &= 0; \\
r[4] &= M[\_n]; \\
PC &= L18 \\
L19 \\
r[0] &= r[1] \ll 2; \\
r[1] &= r[1] + 1; \\
L18 \\
IC &= r[1] \? r[4]; \\
PC &= IC < 0 \rightarrow L19;
\end{align*}
\]

Example Sequence of Optimizations (cont.)

⇒ after loop strength reduction

\[
\begin{align*}
r[2] &= 0; \\
r[1] &= 0; \\
r[4] &= M[\_n]; \\
r[3] &= \_a; \\
PC &= L18 \\
L19 \\
r[1] &= r[1] + 1; \\
L18 \\
IC &= r[1] \? r[4]; \\
PC &= IC < 0 \rightarrow L19;
\end{align*}
\]

Example Sequence of Optimizations (cont.)

⇒ after induction variable elimination

\[
\begin{align*}
r[2] &= 0; \\
r[4] &= M[\_n] \ll 2; \\
r[3] &= \_a; \\
PC &= L18 \\
L19 \\
L18 \\
IC &= r[3] \? r[4]; \\
PC &= IC < 0 \rightarrow L19;
\end{align*}
\]

Example of Common Subexpression Elimination

\[
\begin{align*}
r[1] &= M[r[13] + i] \ll 2; \\
r[1] &= M[r[1] + \_b]; \\
r[2] &= M[r[13] + i] \ll 2; \\
⇒ \\
\begin{align*}
r[1] &= M[r[13] + i] \ll 2; \\
r[1] &= M[r[1] + \_b]; \\
r[2] &= r[1];
\end{align*}
\]
Example of Unreachable Code Elimination

\[ \ldots \]
\[ PC = L12; \]
\[ r[1] = r[5] + r[1]; \]
\[ M[r[13] + j] = r[1]; \]
\[ L13 \]
\[ \Rightarrow \]
\[ : \]
\[ PC = L12; \]
\[ L13 \]
\[ : \]

Example of Branch Chaining

IF a THEN b
compiles into

While c DO d
compiles into

\[ a \]
\[ PC = IC ! = 0 \rightarrow L1; \]
\[ b \]
\[ PC = IC ! = 0 \rightarrow L2; \]
\[ L1: \]
\[ PC = L2; \]
\[ L3: \]

Example of Branch Chaining (cont.)

WHILE c DO
IF a THEN b

\[ \Rightarrow \] after branch chaining

\[ L2: \]
\[ c \]
\[ PC = IC ! = 0 \rightarrow L3; \]
\[ a \]
\[ PC = IC ! = 0 \rightarrow L2; \]
\[ b \]
\[ L1: \]
\[ PC = L2; \]
\[ L3: \]
Example of Jump Elimination by Reversing Branches

PC = IC == 0 → L1;
PC = L2;
L1:
⇒
PC = IC != 0 → L2;
L1:

Example of Instruction Scheduling

M[r[30] + k] = r[2];
r[1] = r[1] + 1;
⇒
r[1] = r[1] + 1;
M[r[30] + k] = r[2];