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 → front end → intermediate code → code generator → target code

**user can:**
- profile program
- change algorithm
- transform loops

**compiler can:**
- improve loops
- procedure calls
- address calculations
- 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

followed by Fig. 8.7, 8.9
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

sum <- 0
j <- 0
goto B3

B1

t1 <- j << 2
t2 <- addr(a)
t3 <- t2[t1]
sum <- sum + t3

B2

if j < n goto B2

B3
Organization of the Code Optimizer

- Front end
- Code optimizer
- Code generator
- Control-flow analysis
- Data-flow analysis
- Code transformations
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

1. Front End
   - Simple intermediate code
2. Code Expander
   - Simple rts
3. Optimizer
   - Assembly code
4. Machine Description

Diagram:
- Front End
  - Simple intermediate code
  - Code Expander
  - Simple rts
  - Optimizer
  - Assembly code
  - Machine Description
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

33 \{32\} \[M[r[3]] = r[2]; \ r[2]:\]
    ⇒
    \[M[r[3]] = M[r[3]]+1; \ r[2] = M[r[3]]+1; \ r[2]:\]
    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\} \[ r[36]=r[36]+i; \]

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

38 \quad r[36]=r[5];

44 \quad r[42]=M[r[5]+i];

45 \{44\} \quad r[42]=r[42]+1;

46 \{45,38\} \quad M[r[36]+i]=r[42]; \quad r[42]:r[36]:
Cascading Instruction Selection (cont.)

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

Example Sequence of Optimizations

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

⇒ after instruction selection

```
M[r[13] + sum] = 0;
M[r[13] + j] = 0;
PC = L18;
L19
    r[0] = M[r[13] + j] <<2;
L18
    PC = IC < 0 →L19;
```
Example Sequence of Optimizations (cont.)

⇒ after register allocation

\[
\begin{align*}
  r[2] &= 0; \\
  r[1] &= 0; \\
  PC &= L18; \\
\end{align*}
\]

L19
\[
\begin{align*}
  r[0] &= r[1] \ll 2; \\
  r[1] &= r[1] + 1; \\
\end{align*}
\]

L18
\[
\begin{align*}
  IC &= r[1] \ ? \ M[_n]; \\
  PC &= IC < 0 \rightarrow L19; \\
\end{align*}
\]
Example Sequence of Optimizations (cont.)

⇒ after code motion

\[
\begin{align*}
& \text{r}[2] = 0; \\
& \text{r}[1] = 0; \\
& \text{r}[4] = \text{M}[\_n]; \\
& \text{PC} = \text{L18}
\end{align*}
\]

L19
\[
\begin{align*}
& \text{r}[0] = \text{r}[1] \ll 2; \\
& \text{r}[2] = \text{r}[2] + \text{M}[\text{r}[0] + \_a]; \\
& \text{r}[1] = \text{r}[1] + 1;
\end{align*}
\]

L18
\[
\begin{align*}
& \text{IC} = \text{r}[1] ? \text{r}[4]; \\
& \text{PC} = \text{IC} < 0 \rightarrow \text{L19};
\end{align*}
\]
Example Sequence of Optimizations (cont.)

⇒ after loop strength reduction

\[
\begin{align*}
\text{r}[2] &= 0; \\
\text{r}[1] &= 0; \\
\text{r}[4] &= \text{M}[\_n]; \\
\text{r}[3] &= \_a; \\
\text{PC} &= \text{L}18
\end{align*}
\]

\textbf{L}19

\[
\begin{align*}
\text{r}[2] &= \text{r}[2] + \text{M}[\text{r}[3]]; \\
\text{r}[3] &= \text{r}[3] + 4; \\
\text{r}[1] &= \text{r}[1] + 1;
\end{align*}
\]

\textbf{L}18

\[
\begin{align*}
\text{IC} &= \text{r}[1] \ ? \ \text{r}[4]; \\
\text{PC} &= \text{IC} < 0 \rightarrow \text{L}19;
\end{align*}
\]
⇒ after induction variable elimination
  r[2] = 0;
  r[3] = _a;
  PC = L18;
L19
L18
  IC = r[3] ? r[4];
  PC = IC < 0 → L19;
Example of Common Subexpression Elimination

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

```
PC = L12;
r[1] = r[5] + r[1];
M[r[13] + j] = r[1];
L13
  ...
⇒
  ...
  PC = L12;
L13
  ...
```
Example of Branch Chaining

IF a THEN b

compiles into

\[
\begin{align*}
a \\
\text{PC} = \text{IC} \neq 0 \rightarrow \text{L1}; \\
b \\
\text{L1}: 
\end{align*}
\]

While c DO d

compiles into

\[
\begin{align*}
\text{L2: } c \\
\text{PC} = \text{IC} \neq 0 \rightarrow \text{L3}; \\
\text{d} \\
\text{PC} = \text{L2}; \\
\text{L3: }
\end{align*}
\]
Example of Branch Chaining (cont.)

WHILE c DO
    IF a THEN b

compiles into

L2 : c
    PC = IC != 0 → L3;
    a
    PC = IC != 0 → L1;
    b
L1:   PC = L2;
L3:
Example of Branch Chaining (cont.)

⇒ after branch chaining

L2 : c
    PC = IC != 0 → L3;
    a
    PC = IC != 0 → L2;
    b
L1:  PC = L2;
L3:  

Example of Jump Elimination by Reversing Branches

\[
\begin{align*}
PC = IC &= 0 \rightarrow L1; \\
PC &= L2; \\
L1: & \\
& \Rightarrow \\
PC &= IC \neq 0 \rightarrow L2; \\
L1: &
\end{align*}
\]
Example of Instruction Scheduling

\[
\begin{align*}
  M[r[30] + k] &= r[2]; \\
  r[1] &= r[1] + 1;
\end{align*}
\]

⇒

\[
\begin{align*}
  r[1] &= r[1] + 1; \\
  M[r[30] + k] &= r[2];
\end{align*}
\]