Chapter 6 OPT.

I. Introduction

II. Machine Independent OPT.
   1. Basic OPT.
   2. Loop OPT.
   3. Removing redundant 3ACs

III. Machine dependent OPT.

IV. Dataflow analysis (DFA)

I. Introduction:

OPT is an attempt to simplify and remove as many unnecessary and redundant 3ACs/assembly instructions as possible.

Ex: \( T = a + b; \)  \( x = a + b \)
     \( x = T; \)

Ex: \( T_1 = a * b \)  \( T_1 = a * b \)
     \( C = T_1 + 1 \)  \( C = T_1 + 1 \)
     \( T_2 = a * b \)  \( d = T_1 + 2 \)
     \( D = T_2 + 2 \)

II. Machine Independent OPT.

1) Basic OPT.
   a. constant folding
      \( x = 2 * 3 \Rightarrow x = 6 \)

   b. constant propagation
      \text{myvalue} = 10  \quad \text{Replace myvalue with 10. during compilation}
      \text{use myvalue}

   c. Reduction in strength: replace an expensive operation with a cheaper one.
      \text{Ex:} \quad a = x^2 \quad (* \text{source} *) \quad \downarrow \quad \downarrow 
      \quad a = x * x \quad \quad \quad \quad \quad \quad T_1 = \ln b 
      \quad T_2 = YT_1 
      \quad X = \exp (T_2)

2) Loop Optimization
   a. Loop – invariant expressions
⇒ Take invariant expressions out of the loop.
Ex: for k = 1 to 1000 do
c[k] = 2 * (p – q) * (n – k + 1) / (sqrt(n) + n)

⇒ T1 = 2 * (p – q) / (sqrt(n) + n)
for k = 1 to 1000 do
c[k] = T1 * (n – k + 1)

b. Loop unrolling
unroll nested loops
Ex: for i = 1 to 200 do
for (j = 1 to 2 do)
write (x[i, j]);

⇒ for i = 1 to 200 do
write (x[i, 1],(x[i, 2]);

3) Remove redundant 3ACs
Ex: x = (a + b) * (a + b) (source)
1. T1 = a + b  1. T1 = a + b
2. T2 = a + b  ⇒  2. x = T1 * T1
3. T3 = T1 * T2
4. x = T3

Q: How do we remove?
3-step process:
Step 1: Create a DAG and label the nodes with variables.
Step 2: Do topological sort on DAG.
Step 3: Create a new set of 3Acs by going from the last element of the sort.

Ex: x = (a + b * c) / (d – b * c) / * source */
1. T1 = b * c
2. T2 = a + T1
3. T3 = b * c
4. T4 = d – T3
5. T5 = T2 / T4
6. x = T5

Step 1: Create a DAG:
Step 2: Topological sort as follows:

\[ n = 1; \]
\[
\begin{align*}
\text{repeat} \\
&\quad \text{- select any source node by favoring the left child & assign } n \text{ to that node.} \\
&\quad \text{- Remove that node and all of its incident (adjacent) edges.} \\
&\quad \text{- } n++ \\
\text{until the graph is empty}
\end{align*}
\]

source node: a node without any incoming edges.

1. \( x, T_5 \)
2. \( T_2 \)
3. \( a \)
4. \( T_4 \)
5. \( d \)
6. \( T_1, T_3 \)
7. \( b \)
8. \( c \)

\[ \text{1. } T_1 = b \times c \]
\[ \text{2. } T_4 = d - T_1 \]
\[ \text{3. } T_2 = a + T_1 \]
\[ \text{4. } x = T_2 / T_4 \]

III. Machine dependent OPT.

OPT on the assembly (O.C.)

- peep-hole OPT.
  
  We have a small moving window of instructions & we optimized object instructions within this window.

\[
\begin{array}{|l|}
\hline
1. \_ \_ \\
2. \_ \_ \\
3. \_ \_ \\
4. \_ \_ \\
\hline
\end{array}
\]

Peep-hole
(fixed-size, 4, 5, or...lines)

a. jump over jump

\[
\begin{align*}
&\quad \text{loop-start} \\
&\quad \_ \_ \\
&\quad \_ \_ \\
&\quad \_ \_ \\
&\quad \_ \_ \\
\end{align*}
\]

\[
\begin{align*}
\text{mov} &\quad ax, i \\
\text{cmp} &\quad ax, limit \\
\text{je} &\quad \text{loop-exit} \\
\text{jmp} &\quad \text{loop-start} \\
\Rightarrow &\quad \text{jne loop-start}
\end{align*}
\]

loop-exit

\[
\_ \_ 
\]
b. redundant load & store

1. L 3, B
2. A 3, C
3. ST 3, A
4. L 3, A
5. S 3, E
6. ST 3, D

\[ a = b + c \]
\[ D = a - e \]

IV. DFA: See book: P233 – 245

Suggested Homework Assignments for Chapter 6:
6.3, 6.4.