Loop Unrolling

Source:
for i := 1 to 100 by 1
endfor

Transformed Code:
for i := 1 to 100 by 4
endfor

Benefits:
• The overhead of testing and branching is reduced.
• This optimization may “enable” other optimizations.
Loop Unrolling

Source:

```c
for i := 1 to 100 by 1
endfor
```

Transformed Code:

```c
for i := 1 to 100 by 4
endfor
```

Benefits:
- The overhead of testing and branching is reduced.
- This optimization may “enable” other optimizations.

Larger Basic Blocks are Good! More opportunities for optimizations such as scheduling

Do 0 to 3 more iterations, as necessary, to finish
Number of iterations is not known at compile-time.
Loop-Invariant Computations

An assignment
\[ x := y \oplus z \]
is “loop-invariant” if:
- It is in a loop, and
- All definitions of \( y \) and \( z \) that reach the statement are outside the loop.

We may be able to move the computation into the “preheader”.

Step 1: Detect the Loop-Invariant Computations.
Step 2: See if it is okay to move the statement into the pre-header.

Example

\[
\begin{array}{c}
y := \ldots \\
z := \\
\end{array}
\quad
\begin{array}{c}
y := \\
z := \ldots \\
\end{array}
\quad
\begin{array}{c}
x := y \oplus z \\
\ldots \\
\ldots \\
\end{array}
\]

© Harry H. Porter, 2006
Detecting Loop-Invariant Computations

**Input:**
Loop L (= a set of basic blocks)
U-D Chain information

**Output:**
The set of loop-invariant statements.

**Idea:**
- Mark some of the statements as “loop-invariant”.
- This may allow us to mark even more statements as loop-invariant.
- Remember the order in which these statements are marked.
Detecting Loop-Invariant Computations

repeat until no new statements are marked...
Look at each statement in the loop.
If all its operands are unchanging then
mark the statement as “loop-invariant”.
An operand is “unchanging” if...
• It is a constant
• It has all reaching definitions outside of the loop
• It has exactly one reaching definition and that definition has already been marked “loop-invariant”.

end

Remember the order in which statements are marked “loop-invariant.”

Moving Loop-Invariant Computations
Consider moving statement
S: x := y ⊕ z
into the loop’s preheader.

The statement must satisfy three conditions.
If it satisfies all conditions, then it can be moved.
**Condition 1**
The block containing S must dominate all exits from the loop.

```plaintext
x := 1

This is loop-invariant

x := 2

A Exit Block

...x...
```

© Harry H. Porter, 2006
Condition 2
There must be no other assignments to “x” in the loop.

This is loop-invariant

x := 1
:= ...x...

x := a+b

This is loop-invariant

x := 1
:= ...x...

x := a+b

© Harry H. Porter, 2006
Condition 3
All uses of “x” in the loop must be reached by ONLY the loop-invariant assignment.

This is loop-invariant
If all three conditions are satisfied, move the statements into the preheader in the order they were marked Loop-Invariant.

\[
\begin{align*}
  w & := a + b \\
  x & := w + 1 \\
  y & := x * 5
\end{align*}
\]
If all three conditions are satisfied, move the statements into the preheader in the order they were marked Loop-Invariant.

\[
\begin{align*}
  w &:= a + b \\
  x &:= w + 1 \\
  y &:= x \times 5
\end{align*}
\]

Now this becomes loop-invariant.
If all three conditions are satisfied, move the statements into the preheader in the order they were marked Loop-Invariant.

1. \( w := a + b \)
2. \( x := w + 1 \)
3. \( y := x * 5 \)

Move this into preheader first.

Move this into preheader second.