Peephole Optimization
Special case individual instructions

- Use algebraic identities
  - Knowledge of mema
- Can eliminate second instruction without needing any global pairs
- See if an obvious replacement is possible: store/load

Examine a few consecutive instructions: 2 to 4

Final pass over generated code:

Peephole Optimization
More delicate with floating-point numbers:

\[
\begin{align*}
A / 1 &= A \\
0 * 0 &= 0 \\
A * 1 &= A \\
A * 2 &= A + A
\end{align*}
\]

operand:

Worth recognizing single instructions with a constant operand:

Algebraic Identities
A. Pnueli

Is This Ever Helpful?

Also, seemingly redundant code can be produced by other optimizations. This is an important effect.

Why would anyone write

\[
\text{a = b * MAX\_TASKS};
\]

\[
\ldots
\]

\[\text{#define MAX\_TASKS 1}\]

In fact one might write

Why bother to correct such obvious junk code?

Why would anyone write

\[
\text{X*1}?
\]

Is This Ever Helpful?
Replace Multiply by Shift

A := A \times 4;

A := A / 4;

If unsigned, can replace with "shift right". But must worry about overflow if language does.

But shift right arithmetic is a well-known problem in language may allow it anyway (traditional C).

Replacing Multiply by Shift
Note similarity with efficient exponentiation method.

If multiplication is very slow (or on a machine with no multiply instruction like the original SPARC), decomposing a constant operand into a sum of powers of two can be effective:

\[ X \cdot 125 = X \cdot 128 - X \cdot 4 + X \]

Two shifts, one subtract and one add, which may be faster than one multiply.

Addition Chains for Multiplication
was negative
truncated towards or away from zero if either operand
to
Prior to C99, implementations were allowed to
in most languages \((-5)/2 = -2\)
which is -3 not -2
Shift right and use sign bit to fill most significant bits
-Arithmetic Right Shift:

The Right Shift Problem
As a result, label \( \text{lab1} \) may become dead (unreachable).

\[
\text{JNE lab2}
\]
\[
\text{JMP lab2}
\]
\[
\text{...}
\]
\[
\text{JNE lab1}
\]

A jump to an unconditional jump can copy the target address.

Folding Jumps to Jumps
As a result, lab1 may become dead.

Ret

Can be replaced by

lab1 Ret

...  

jmp lab1

A jump to a "return" can be replaced by a "return"
A subprogram is tail-recursive if the last computation is a call to itself. A program can be made more efficient by replacing tail-recursive calls by a sequence of goto start statements.

---

Tail Recursion Elimination

A program is tail-recursive if the last computation is a call to itself.
Advantages of Tail Recursion Elimination

- Saves stack space: converts linear stack usage to constant usage.
- Saves time: an assignment and jump are faster than a call with one parameter.
- In languages with no loops, this may be a required optimization.

Optimization: specified in Scheme standard.
Lecture 12: Peephole Optimization

Tail Recursion Elimination at the Instruction Level

A. Pnueli

TailRecursionElimination at the Instruction Level

Now RET in func returns to original caller, because

single return address on stack.

CALL func

RET

JMP func

Can generate instead:

CALL func

RET

CALL func

Consider the sequence on the X86:

CALL push return address on stack, RET in body

removes it, RET in caller returns.

CALL func

RET

Honors Compilers, NYU, Spring 2007
Peephole Optimization

A. Pnueli

Lecture 12: Peephole Optimization

Peephole Optimization in the REALIA

COBOL Compiler

Full compiler for standard COBOL targeted to the IBM PC.

Now distributed by Computer Associates.

Run in 150K bytes, but must be able to handle very large programs that run on mainframes.

No global optimization possible: multiple linear passes.

No global data: no flow graph.

Multiple peephole optimizations, compiler iterates until code is stable.

Each pass scan code backwards to minimize address recomputations.

Lecture, NVU, Spring 2007.
... Recor-Credit.
... Send-Bill.
end-it.
perform Recor-Credit
else
perform Send-Bill
if Balance is negative then
Process-Balance.

Typical COBOL Code: Control Structures

A. Pnueli
Simple Assembly: Perform Equivalent to Call

```
Pb: cmp balance, 0
    jnl L1
    Call Sb
    jmp L2
    Call Rc
L1: Call...
L2: Call...
Sb: ret...
Rc: ret...
```
Fold Jump to Return Statement

- Jump to "Return"

```
Pb: cmp balance, 0
     jnl L1
     Call Sb
     jmp L2
     Call Rc

L1: Call
L2: ret
Sb: ret
Rc: ret
```
Lecture 12: Peephole Optimization

\[ \text{P}: \text{cmp, balance, 0} \]

- Jumps to unconditional jump

- Will become useless

- Folded

Corresponding Assembly

\[
\begin{align*}
\text{Rc:} & \quad \text{ret} \\
\text{Sb:} & \quad \text{ret} \\
\text{L2:} & \quad \text{ret} \\
\text{L1:} & \quad \text{jmp \ Rc} \\
\text{Csl:} & \quad \text{ret} \\
\text{Js:} & \quad \text{init \ L1} \\
\text{Pb:} & \quad \text{cmp \ balance, 0}
\end{align*}
\]
Code Following a Jump is Unreachable

\begin{itemize}
\item \texttt{L1:}
\item \texttt{Sb:}
\item \texttt{Rc:}
\end{itemize}

\begin{itemize}
\item \texttt{cmp balance, 0}
\item \texttt{jl}\ 
\item \texttt{jmp Sb}
\item \texttt{ret}
\end{itemize}

\begin{itemize}
\item \texttt{folded}
\item \texttt{unreachable}
\end{itemize}
Jump to Following Instruction is a Noop

Pb: cmp balance, 0
   jnl Rc
   jmp Sb
   – jump to next instruction

Sb:
   :::

Rc:
   :::

Sb:
   :::

Rc:
   :::

Honors Compilers, NYU, Spring, 2007
Final Code

A. Pnueli

Final Code is efficient as inlining.

Pb: cmp balance, 0

Rc:

Pb: cmp balance, 0

JnlRc

Sb:

Sb:

Rc:

Rc:

Iterate till no further change

Yield further optimization opportunities.

All transformations are local. Each optimization may

Final Code Leaners' NYU, Spring, 2007
For simplicity assume all unsigned and all in registers

\[ \text{if } A > B \text{ then } C := A; \]
\[ \text{else } C := B; \]

For simplicity assume all unsigned and all in registers

\[ \text{if } A \leq B \text{ then } C := A; \]

Consider typical maximum computation

Arcane Tricks
Eliminating Max Jumps on X86

A Pnueli

Lecture 12: Peephole Optimization

Honors Compilers, NYU, Spring, 2007

Simple minded ASM code

One jump in either case

L2:
L1:
MOV A,C
JNE L1
CMP A,B

JMP L2
MOV B,C
JNAEL1

A Pnueli

Computing Max Without Jumps on x86

Super-compiler: exhaustively search of instruction patterns to uncover similar tricks.

More instructions but NO JUMPS

OR %eax, %eax
AND %eax, %eax
AND %eax, %eax
NOT %eax, %eax
MOV %eax, %eax
SBP %eax, %eax
CMP A, B

Instruction and carry flag

Architecture-specific trick: use subtract with borrow