A Simple Code Generator

• Generate code for a single basic block
• How to use registers?
  – In most machine architectures, some or all of the operands must be in registers
  – Registers make good temporaries
  – Hold values that are computed in one basic block and used in other blocks
  – Often used with run-time storage management

Register and Address Descriptors

• For each available register, a register descriptor (RD) keeps track of the vars whose current value in that register
  – Initially empty
• For each var, an address descriptor (AD) keeps track of the locations where the current value of the var can be found
  – Location can be a register, a memory address, etc.

Managing Register and Address Descriptors

• For LD R, x
  – Change RD for R so it holds only x
  – Change AD for x by adding R as an additional location
• For ST x, R
  – Change AD for x to include its own memory location
• For ADD R, R, R
  – Change RD for R so it holds only x
  – Change AD for x so its only location is R
  – Remove R from the AD of any var other than x

Example

\[
\begin{align*}
  t &= a - b \\
  u &= a - c \\
  v &= t + u \\
  a &= d \\
  d &= v + u
\end{align*}
\]

• t, u, v are temp vars, while a, b, c, d are global
• Assume registers are enough
  – Reuse registers whenever possible
**Example**

Example

```
R1 R2 R3 a b c d t u v

LD R1, a; LD R2, b; SUB R2, R1, R2

R1 R2 R3 a b c d t u v

a b

LD R3, c; SUB R1, R1, R2

R1 R2 R3 a b c d t u v

c d

ADD R3, R2, R1

v = t + u

R1 R2 R3 a b c d t u v

u t v

LD R2, d

R1 R2 R3 a b c d t u v

d
```

**Function getReg**

- Consider picking $R_y$ for $y$ in $x = y + z$
  - If $y$ in a register, do nothing
  - If $y$ not in a register and there is a empty one, choose it as $R_y$
  - Let $v$ be one of the var in $R$
    - We are OK if $v$ is somewhere besides $R$
    - We are OK if $v$ is $x$
    - We are OK if $v$ is not used later
    - Spill: ST v, R

**Peephole Optimization**

- Exam a sliding window and replace instruction sequence with a shorter or faster sequence
  - Redundant-instruction elimination
  - Flow-of-control optimization
  - Algebraic simplifications
  - Use a machine idioms

**Eliminating Redundant Loads and Stores**

```
LD a, R0
ST R0, a
```
Eliminating Unreachable Code

if debug==1 goto L1
goto L2
L1: print debugging info
L2:

if debug!=1 goto L2
L1: print debugging info
L2:

Flow-of-Control Optimizations

goto L1
...  
L1: goto L2

goto L2
...  
L1: goto L2

L1:
...  
if a < b goto L1  
...  
L1: goto L2

L2:
...  
if a < b goto L2  
...  
L1: goto L2

Register Allocation and Assignment

• Usage Counts
  – An approximate formula for the benefit from allocating a register for x
    \[ \sum_{\text{blocks in } B} \text{use}(x, B) + 2 \times \text{live}(x, B) \]
  – use(x,B) is the number of times x is used in B prior to any definition of x
  – live(x,B) is 1 if x is live on exit from B and is assigned a value in B, 0 otherwise

Instruction Selection by Tree Rewriting

• Instruction selection can be a large combinational task
  – Even the evaluation order is given and register allocation has been done

Example

Instruction Selection by Tree Rewriting

• Instruction selection can be a large combinational task
  – Even the evaluation order is given and register allocation has been done

Tree-Translation Schemes

LD R0, #a
ADD R0, R0, SP
ADD R0, R0, i(SP)
LD R1, b
INC R1
ST *R0, R1
Optimal Code Generation for Expressions

- We can choose registers optimally
  - If a basic block consists of a single expression, or
  - It is sufficient to generate code for a block one expression at a time

Ershov Numbers

- Assign the nodes of an expression tree a number that tells how many registers needed
  - Label leaf 1
  - The label of an interior node with one child is the label of its child
  - The label of an interior node with two children
    - the larger one if the labels are different
    - One plus the label if the labels are the same

Generating Code From Labeled Expression Tree

- Recursive algorithm starting at the root
  - Label k means k registers will be used
  - \( R_{b+1}, R_{b+2}, \ldots, R_{b+k} \), where \( b>1 \) is a base

Ershov Numbers

- For a leaf \( x \), if base is \( b \) generate LD \( R_{b+1}, x \)

Ershov Numbers

- \((a-b)+e*(c+d)\)

Ershov Numbers

- \( t1 = a - b \)
- \( t2 = c + d \)
- \( t3 = e \cdot t2 \)
- \( t4 = t1 + t3 \)
Insufficient Supply of Registers

- Input: a labeled tree and a number $r$ of registers
- For a node $N$ with at least one child labeled $r$ or greater
  - recursively generate code for the big child with $b=1$.
    The result will appear in $R_i$
  - Generate machine instruction `ST t_k, R_i`
  - If the little child has label $r$ or greater, $b=1$. If the label is $j<r$, then $b=r-j$. The result in $R_i$
  - Generate the instruction `LD R_{i-1}, t_k`
  - Generate OP $R_i, R_{i-1}$ or OP $R_i, R_{i-1}, R_i$