Phases of a compiler

Target machine code generation

Figure 1.6, page 5 of text
Flow Graph

Figure 8.9 [p. 530]

Entry and exit nodes added.

Jump targets replaced by block names.
DAG for basic block

Example 8.10 [p. 534]

1) \( a = b + c \)
2) \( b = a - d \)
3) \( c = b + c \)
4) \( d = a - d \)
Example 8.10 [p. 534]

Apply the "value-number" method from section 6.1.1

1) \( a = b + c \)
2) \( b = a - d \)
3) \( c = b + c \)
4) \( d = a - d \)
Example 8.10 [p. 534]

1) $a = b + c$
2) $b = a - d$
3) $c = b + c$
4) $d = b$

Diagram:

- $a$ and $d$
- $b$ and $c$
- $b, d$
- $c$
8.6 A Simple Code Generator [p. 542]

- algorithm focuses on generation of code for a single basic block
- generates code for each three address code instruction
- manages register allocations/assignment to avoid redundant loads/stores
register descriptor

"For each available register, a register descriptor keeps track of the variables names whose current value is in that register." [p. 543]
address descriptor

"For each program variable, an address descriptor keeps track of the location or locations where the current value of that variable can be found." [p. 543]
getReg function

"...getReg(I)...selects registers for each memory location associated with the three-address instruction I." [p. 544]
copy instructions

\[ x = y \]

"We assume getReg will always choose the same register for both \( x \) and \( y \). If \( y \) is not already in that register \( Ry \), then generate the machine instruction \( \text{LD} \ Ry, y \). If \( y \) was already in \( Ry \), we do nothing. It is only necessary that we adjust the register descriptor for \( Ry \) so that it includes \( x \) as one of the values found there." [p. 544]
Writing back to memory at end of block

At the end of a basic block we must ensure that live variables are stored back into memory.

"...for each variable \( x \) whose address descriptor does not say that is value is located in the memory location for \( x \), we must generate the instruction \( ST \ x, R \), where \( R \) is a register in which \( x \)'s value exists at the end of the block." [p. 545]
Updating register descriptors (RD) and address descriptors (AD)

1. LD R, x
   (a) Set RD of R to only x
   (b) Add R to AD of x
2. ST x, R
   (a) Add &x to AD of x
3. OP Rx, Ry, Rz for x = y op z
   (a) Set RD of Rx to only x
   (b) Set AD of x to only Rx (&x not in AD of x!)
   (c) Remove Rx from the AD of any variable other than x
4. "When we process a copy statement x = y, after generating the load for y into register Ry, if needed, and after managing descriptors as for all load statement (per rule 1):" [p. 545]
   (a) Add x to the RD of Ry
   (b) Set AD of x to only Ry
Example [p. 546]

\[ t = a - b \]
\[ u = a - c \]
\[ v = t + u \]
\[ a = d \]
\[ d = v + u \]
<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>t</td>
<td>u</td>
<td>v</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
</tbody>
</table>
\begin{align*}
\text{LD R1, a} \\
\text{LD R2, b} \\
\text{SUB R2, R1, R2}
\end{align*}
\[ t = a - b \]

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

No registers are in use - pick the first two available for a and b. Choose to put t in R2 because b is not used again in this block.
\[ t = a - b \]

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

\[ u = a - c \]

LD R3, c  
SUB R1, R1, R3
\[ t = a - b \]

- **LD R1, a**
- **LD R2, b**
- **SUB R2, R1, R2**
- **LD R3, c**
- **SUB R1, R1, R3**

- **u = a - c**
- **LD R3, c**
- **SUB R1, R1, R3**

**Notes:**
- \( a \) is already in \( R1 \), so no load needed.
- \( t \) is used later, so don't overwrite \( R2 \).
- Load \( c \) into \( R3 \).
- Put result into \( R1 \) since \( a \) is not needed again in this block.
<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>t</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>u</td>
<td>a, R1</td>
<td>b, c</td>
</tr>
<tr>
<td>v</td>
<td>a, b</td>
<td>c, R3</td>
</tr>
<tr>
<td></td>
<td>a b</td>
<td>c d</td>
</tr>
</tbody>
</table>

1. \( t = a - b \)
2. \( u = a - c \)
3. \( v = t + u \)

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

LD R3, c
SUB R1, R1, R3

ADD R3, R2, R1
\[
t = a - b
\]

```
LD R1, a
LD R2, b
SUB R2, R1, R2
```

\[
u = a - c
\]

```
ADD R3, R2, R1
```

\[
v = t + u
\]

```
ADD R3, R2, R1
```

- t and \( u \) are already in registers - no loads needed.
- Perform addition, putting the result into \( R3 \); \( c \) is no longer needed in this block.
Same state as at end of previous slide
a = d

LD R2, d
Load d into R2, attach a to R2 as well.
<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>v</td>
<td>t</td>
</tr>
</tbody>
</table>

\[ a = d \]

| a  | b  | c  | d  | t  | u  | v  |

| a  | b  | c  | d  | R2 | R1 | R3 |

\[ d = v + u \]

<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>a,d</td>
<td>v</td>
</tr>
</tbody>
</table>

LD R2, d

<table>
<thead>
<tr>
<th>R2</th>
<th>b</th>
<th>c</th>
<th>d,R2</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>R2</th>
<th>b</th>
<th>c</th>
<th>d,R2</th>
</tr>
</thead>
</table>

ADD R1, R3, R1
\[ a = d \]

\[
\begin{array}{|c|c|c|}
\hline
R1 & R2 & R3 \\
\hline
u & t & v \\
\hline
\end{array}
\]

LD R2, d

\[
\begin{array}{|c|c|c|}
\hline
R1 & R2 & R3 \\
\hline
u & a,d & v \\
\hline
\end{array}
\]

\[ d = v + u \]

ADD R1, R3, R1

\[
\begin{array}{|c|c|c|}
\hline
R1 & R2 & R3 \\
\hline
d & a & v \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
a & b & c \\
\hline
a & b & c & d \\
\hline
a & b & c & d & t \\
\hline
a & b & c & d & t & u & v \\
\hline
R2 & b & c & d,R2 & R1 & R3 \\
\hline
R2 & b & c & R1 & R3 \\
\hline
\end{array}
\]

- u and v are in registers, so no loads needed.
- Cannot destroy a (exists only in R2) without storing back to memory, so use R1 for result.
- Move d to R1 from R2.
<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>t</td>
<td>v</td>
</tr>
</tbody>
</table>

\( a = d \)

<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>a,d</td>
<td>v</td>
</tr>
</tbody>
</table>

\( d = v + u \)

<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>a</td>
<td>v</td>
</tr>
</tbody>
</table>

LD R2, d

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>t</th>
<th>u</th>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ADD R1, R3, R1

<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>a</td>
<td>v</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>t</th>
<th>u</th>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>R2</td>
<td>b</td>
<td>c</td>
<td>d,R2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R2</th>
<th>R1</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ST a, R2
ST d, R1
We're at the end of the block. Make sure that values of R1 and R2 are stored back to memory (d and a respectively). Value of R3 can be lost - it is a temporary of only this block.
getReg function

\[ x = y \; \text{op} \; z \]

How do we do this?
getReg function

\[ x = y \text{ op } z \]

1. If \( y \) is currently in a register, pick a register already containing \( y \) as \( R_y \). Do not issue a machine instruction to load this register, as none is needed.

2. If \( y \) is not in a register, but there is a register currently empty, pick one such register as \( R_y \). \([LD \ R_y, y]\)
getReg function

\[ x = y \text{ op } z \]

3. The difficult case occurs when \( y \) is not in a register, and there is no register that is currently empty. We need to pick one of the allowable registers anyway, and we need to make it safe to reuse. Let \( R \) be a candidate register, and suppose \( v \) is one of the variables that the register descriptor for \( R \) says is in \( R \). We need to make sure that \( v \)'s value either is not really needed, or that there is somewhere else we can go to get the value of \( v \). The possibilities are:
getReg function

\[ x = y \text{ op } z \]

(a) If the address descriptor for \( v \) says that \( v \) is somewhere else besides \( R \), then we are OK.
(b) If $v$ is $x$, the variable being computed by instruction $I$, and $x$ is not also one of the other operands of instruction $I$ ($z$ in this example), then we are OK. The reason is that in this case we know this value of $x$ is never again going to be used, so we are free to ignore it.
getReg function

\[ x = y \text{ op } z \]

(c) Otherwise, if \( v \) is not used later (that is, after the instruction I, there are no further uses of \( v \), and if \( v \) is live on exit from the block, then \( v \) is recomputed within the block), then we are OK.
getReg function

\[ x = y \text{ op } z \]

(d) If we are not OK by one of the first three cases, then we need to generate the store instruction \( ST \ v, R \) to place a copy of \( v \) in its own memory location. This operation is called a spill.” [p. 547-548]
getReg function

\[ x = y \text{ op } z \]

Repeat the above (a) - (d) steps for each variable \( v \) currently in \( R \).

Let the score of \( R \) be the number of ST instructions generated. Choose the \( R \) with lowest score to actually use.
getReg function
\[ x = y \text{ op } z \]

We also need a register for the result, Rx. “The issues and options are almost as for y, so we shall only mention the differences.

1. Since a new value of x is being computed, a register that holds only x is always an acceptable choice for Rx. This statement holds even if x is one of y and z, since our machine instructions allow two registers to be the same in one instruction.
getReg function

\[ x = y \text{ op } z \]

2. If \( y \) is not used after instruction I, in the sense described for variable \( v \) in item (3c), and \( R_y \) holds only \( y \) after being loaded, if necessary then \( R_y \) can also be used as \( R_x \). A similar option holds regarding \( z \) and \( R_z \).” [p. 548]
Interesting question

Can $x := \text{"Hi!"};$ and $y := \text{"Hi!"};$ be aliases? Yes. Does the language mandate this? No. What can happen in a loop with a string literal? Either. Hmm.

Double-check language spec on what has to happen with string literals.