CSE443
Compilers

Dr. Carl Alphonce
alphonce@buffalo.edu
343 Davis Hall
Phases of a compiler

Figure 1.6, page 5 of text

Target machine code generation
8.6.1 Register and Address Descriptors

A three-address instruction of the form:

\[ v = a \text{ op } b \]

we generate:

LD Rx, a
LD Ry, b
OP Rx, Rx, Ry
ST Rx, v
This results in many redundant loads and stores

This may not make effective use of available registers

Use two data structures
- register descriptor
- address descriptor
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]
Example
(paraphrased from 8.6.2, page 544)

A three-address instruction of the form:

\[ v = a \text{ op } b \]

1. Use getReg\(v = a \text{ op } b\) to select
registers for \(v\), \(a\) and \(b\): \(R_v\), \(R_a\), and
\(R_b\) respectively

2. If \(a\) is not already in \(R_a\), generate LD
\(R_a\), \(a'\) (where \(a'\) is one of the possibly
many current locations of \(a\))

3. Similarly for \(b\).

4. Generate OP \(R_v\), \(R_a\), \(R_b\)
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 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 its 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]

\[ t = a - b \]

\[ u = a - c \]

\[ v = t + u \]

\[ a = d \]

\[ d = v + u \]

What does liveness and next use info looking like here?
Algorithm 8.7 [p. 528]
Determining the liveness and next-use information for each statement in a basic block.

INPUT: A basic block B of three address instructions. Assume the symbol table initially shows all non-temporary variables in B as being live on exit.

OUTPUT: At each statement \( i: x = y + z \) in B, we attach to \( i \) the liveness and next-use information for \( x \), \( y \), and \( z \).

METHOD: We start at the last statement in B and scan backwards to the beginning of B. At each statement \( i: x = y + z \) in B do the following:

1) attach to statement \( i \) the information currently found in the symbol table regarding the next-use and liveness of \( x \), \( y \), and \( z \).
2) In the symbol table, set \( x \) to "not live" and "no next use".
3) In the symbol table, set \( y \) and \( z \) to "live" and the next uses of \( y \) and \( z \) to instruction \( i \).
INPUT: A basic block B of three address instructions. Assume the symbol table initially shows all non-temporary variables in B as being live on exit.

1: \( t = a - b \)
2: \( u = a - c \)
3: \( v = t + u \)
4: \( a = d \)
5: \( d = v + u \)
We start at the last statement in B and scan backwards to the beginning of B. At each statement i:

\[ x = y + z \]

in B do the following:

1) attach to statement i the information currently found in the symbol table regarding the next-use and liveness of x, y, and z.

2) In the symbol table, set x to "not live" and "no next use".

3) In the symbol table, set y and z to "live" and the next uses of y and z to instruction i.
We start at the last statement in B and scan backwards to the beginning of B. At each statement i:

$$x = y + z$$

in B do the following:

1) attach to statement i the information currently found in the symbol table regarding the next-use and liveness of x, y, and z.

2) In the symbol table, set x to "not live" and "no next use".

3) In the symbol table, set y and z to "live" and the next uses of y and z to instruction i.
We start at the last statement in B and scan backwards to the beginning of B. At each statement i:

\[ x = y + z \]

in B do the following:

1) attach to statement i the information currently found in the symbol table regarding the next-use and liveness of x, y, and z.

2) In the symbol table, set x to "not live" and "no next use".

3) In the symbol table, set y and z to "live" and the next uses of y and z to instruction i.
We start at the last statement in B and scan backwards to the beginning of B. At each statement i:

\[ x = y + z \]

in B do the following:
1) attach to statement i the information currently found in the symbol table regarding the next-use and liveness of x, y, and z.
2) In the symbol table, set x to "not live" and "no next use".
3) In the symbol table, set y and z to "live" and the next uses of y and z to instruction i.
We start at the last statement in B and scan backwards to the beginning of B. At each statement i:

\[ x = y + z \]

in B do the following:

1) attach to statement i the information currently found in the symbol table regarding the next-use and liveness of \( x, y, \) and \( z \).

2) In the symbol table, set \( x \) to "not live" and "no next use".

3) In the symbol table, set \( y \) and \( z \) to "live" and the next uses of \( y \) and \( z \) to instruction i.
### Example [p. 546]

We start at the last statement in B and scan backwards to the beginning of B. At each statement $i$:

1. $x = y + z$

in B do the following:

1) attach to statement $i$ the information currently found in the symbol table regarding the next-use and liveness of $x$, $y$, and $z$.

2) In the symbol table, set $x$ to "not live" and "no next use".

3) In the symbol table, set $y$ and $z$ to "live" and the next uses of $y$ and $z$ to instruction $i$.

| 1: $t = a - b$ | \( \begin{array}{cccccc}
      a & b & c & d & t & u & v \\
      L & D &  &  &  &  &  \\
    \end{array} \) |
|----------------|---------------------------------------------------------------|
| 2: $u = a - c$ | \( \begin{array}{cccccc}
      a & b & c & d & t & u & v \\
      L & D &  &  &  &  &  \\
    \end{array} \) |
| 3: $v = t + u$ | \( \begin{array}{cccccc}
      a & b & c & d & t & u & v \\
      L & D &  &  &  &  &  \\
    \end{array} \) |
| 4: $a = d$     | \( \begin{array}{cccccc}
      a & b & c & d & t & u & v \\
      L & D &  &  &  &  &  \\
    \end{array} \) |
| 5: $d = v + u$ | \( \begin{array}{cccccc}
      a & b & c & d & t & u & v \\
      L & D &  &  &  &  &  \\
    \end{array} \) |
We start at the last statement in B and scan backwards to the beginning of B. At each statement i:

\[ x = y + z \]

in B do the following:

1) attach to statement i the information currently found in the symbol table regarding the next-use and liveness of x, y, and z.

2) In the symbol table, set x to "not live" and "no next use".

3) In the symbol table, set y and z to "live" and the next uses of y and z to instruction i.
We start at the last statement in B and scan backwards to the beginning of B. At each statement i:

\[ x = y + z \]

in B do the following:

1) attach to statement i the information currently found in the symbol table regarding the next-use and liveness of x, y, and z.

2) In the symbol table, set x to "not live" and "no next use".

3) In the symbol table, set y and z to "live" and the next uses of y and z to instruction i.
We start at the last statement in B and scan backwards to the beginning of B. At each statement i:

\[ x = y + z \]

in B do the following:

1) attach to statement i the information currently found in the symbol table regarding the next-use and liveness of x, y, and z.
2) In the symbol table, set x to “not live” and “no next use”.
3) In the symbol table, set y and z to “live” and the next uses of y and z to instruction i.
Example [p. 546]

We start at the last statement in B and scan backwards to the beginning of B. At each statement i:

$x = y + z$
in B do the following:

1) attach to statement i the information currently found in the symbol table regarding the next-use and liveness of x, y, and z.

2) In the symbol table, set x to "not live" and "no next use".

3) In the symbol table, set y and z to "live" and the next uses of y and z to instruction i.
\[
t = a - b \\
u = a - c \\
v = t + u \\
a = d \\
d = v + u
\]

At start of block, assume the values of variables \(a\), \(b\), \(c\), and \(d\) are in main memory.

Variables \(t\), \(u\), and \(v\) are compiler-generated temporary variables.
\[ t = a - b \]

```
LD R1, a
LD R2, b
SUB R2, R1, R2
```
\[ t = a - b \]

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

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

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

<table>
<thead>
<tr>
<th>a, R1</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>R2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>R1</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

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.
\[
t = a - b
\]

\[
u = a - c
\]

<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>t</td>
<td></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>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>a</td>
<td>R1</td>
<td>b</td>
</tr>
</tbody>
</table>

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

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

- \( a \) 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.

\[ u = a - c \]

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

\[ u = a - c \]

\[ v = t + u \]

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

LD R3, c
SUB R1, R1, R3

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

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

\[ u = a - c \]

\[ v = t + u \]

<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>

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

ADD R3, R2, R1

R1
R2
R3

a
b
c
d
t
u
v

\text{t and } u \text{ are already in registers - no loads needed.}

\text{Perform addition, putting the result into R3; } c \text{ is no longer needed in this block.}
<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>

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

*Same state as at end of previous slide*
\[
\begin{array}{c|c|c|c}
R1 & R2 & R3 \\
\hline
u & t & v \\
\end{array}
\]

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

\[a = d\]

LD R2, d
Load d into R2, attach a to R2 as well.
\[
a = d
\]
\[
d = v + u
\]
\[
a = d
\]

\[
d = v + u
\]

\[
LD \ R2, d
\]

\[
ADD \ R1, R3, R1
\]

\[
u \text{ and } v \text{ are in registers, so no loads needed.} \\
\text{Cannot destroy } a \text{ (exists only in } R2) \text{ without storing back to memory, so use } R1 \text{ for result.} \\
\text{Move } d \text{ to } R1 \text{ from } R2.
\]
\begin{itemize}
\item \textbf{a = d}
\item \textbf{d = v + u}
\item \textbf{exit}
\end{itemize}

\begin{verbatim}
LD R2, d
ADD R1, R3, R1
ST a, R2
ST d, R1
\end{verbatim}
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?