**Code Generation Algorithm #2**

**Focus on one Basic Block at a time**

- Ignore all other basic blocks
- Generate best possible code for the basic block

---

**Register Strategy:**

- Store all LIVE variables in memory between basic blocks.
- Within each basic block...
  - Use registers for variables and computation, as necessary
- Each basic block will use registers independently of other basic blocks

---

© Harry H. Porter, 2006
Q: Why store all variables at the end of each Basic Block? Why not leave them in registers?

A: Each Basic Block is processed in isolation. A variable might be put in different registers in different blocks.
Q: Why store all variables at the end of each Basic Block?
Why not leave them in registers?

A: Each Basic Block is processed in isolation.
A variable might be put in different registers in different blocks.

B₅
“x” is in R7

B₆
“x” is in R2

B₈
?

B₉
?

An Alternate Approach:
Assign “x” to one register for the entire routine...
But that ties up a register for too much time!

We Need “Live Variable” Information!
We’ll need to know which variables are LIVE at the end of each basic block.

We’ll only store LIVE variables at the end of each Basic Block.
We Need “Live Variable” Information!

We’ll need to know which variables are LIVE at the end of each basic block

**Option 1:**
Perform live variable analysis beforehand (during optimization phase)

We’ll only store LIVE variables at the end of each Basic Block

**Option 2:**
Assume every variable is live at the end of every basic block
We Need “Live Variable” Information!

We’ll need to know which variables are LIVE at the end of each basic block

Option 1:
Perform live variable analysis beforehand (during optimization phase)

Option 2:
Assume every variable is live at the end of every basic block

Option 3:
Distinguish temporaries from normal variables...
Assume temps are not live between blocks.
Assume normal variables are live between blocks.
(For more precision, we may want to distinguish which variables are in any Use(B_i) sets)

“Definition” and “Use” of Variables

A “Definition” of variable x is an instruction that changes the value
\[ x := \ldots \]
A “Use” of x is an instruction that reads or uses the value
\[ \ldots := \ldots x \ldots \]

This statement defines “y”...

What are the next uses of the variable it defines?

(stmts 106, 109, 112)

...if control flow could allow this value to reach these uses.
The “Next-Use” Information

Consider a bunch of statements.
(Some of the statements define variables.)
For each “definition”, we want to know...
What are its “Next-Uses”?  
What statements “use” the value assigned in the definition?
Control flow must be able to go from the “definition”
to the “use” without any intervening “definitions”.
For each statement, we want to know “What are its Next-Uses”?

<table>
<thead>
<tr>
<th>Line</th>
<th>Definition</th>
<th>Dfs</th>
<th>Next-Uses</th>
</tr>
</thead>
<tbody>
<tr>
<td>104</td>
<td>y := a + 5</td>
<td></td>
<td>{106,109,112}</td>
</tr>
<tr>
<td>105</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>b := y * b</td>
<td>104</td>
<td></td>
</tr>
<tr>
<td>107</td>
<td></td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>108</td>
<td></td>
<td>106</td>
<td>{109,110}</td>
</tr>
<tr>
<td>109</td>
<td>x := b * y</td>
<td>107</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>b := b - x</td>
<td>108</td>
<td></td>
</tr>
<tr>
<td>111</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>c := y + b</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Goal:
Process a single basic block
Compute the Next-Use info
For each IR instruction...
  x := y + z
For each variable in the instruction...
  e.g., x, y, z
Determine...
  Is the variable LIVE or DEAD after the instruction?
  If it is LIVE, then...
    Is it used again in this block?
      If so, where is it used next?
Assumption:
We already have LIVENESS info for all variables
  at the end of the block.
Next-Use Algorithm

- Identify all variables used in this block.
- Use a table
  - One entry for each variable
  - For each variable, store...
    - Its current status
      - LIVE or DEAD
      - If LIVE, its next-use in this block
        - (0=not used again in this block)

- Start with the LIVENess info at the BOTTOM of the block.
- Work through the block in reverse order instruction-by-instruction
- Update the table, as we go upward.

A temporary data structure, used only for this algorithm
(Implementation Idea: Add fields to “VarDecl” to hold this info)
1:  \( t_1 := 4 \times i \)  
2:  \( t_2 := a[t_1] \)  
3:  \( t_3 := 4 \times i \)  
4:  \( t_4 := b[t_3] \)  
5:  \( t_5 := t_2 \times t_4 \)  
6:  \( t_6 := \text{prod} + t_5 \)  
7:  \( \text{prod} := t_6 \)  
8:  \( t_7 := i + 1 \)  
9:  \( i := t_7 \)  
10:  if \( i <= 20 \) goto...

\[
\begin{align*}
t_1 & : D \\
t_2 & : D \\
t_3 & : D \\
t_4 & : D \\
t_5 & : D \\
t_6 & : D \\
t_7 & : D \\
a & : L(0) \\
b & : L(0) \\
\text{prod} & : L(0) \\
i & : L(0) 
\end{align*}
\]
1: t1 := 4 * i
2: t2 := a[t1]
3: t3 := 4 * i
4: t4 := b[t3]
5: t5 := t2 * t4
6: t6 := prod + t5
7: prod := t6
8: t7 := i + 1
9: i := t7
10: if i <= 20 goto ...

i: L(0) t7: D
1: \( t1 := 4 \times i \)
2: \( t2 := a[t1] \)
3: \( t3 := 4 \times i \)
4: \( t4 := b[t3] \)
5: \( t5 := t2 \times t4 \)
6: \( t6 := \text{prod} + t5 \)
7: \( \text{prod} := t6 \)
8: \( t7 := i + 1 \)
9: \( i := t7 \)
10: \( \text{if } i <= 20 \text{ goto ...} \)

\[ i : \text{L(10)} \]
\[ t7 : \text{D} \]

\[\text{t1: D} \]
\[\text{t2: D} \]
\[\text{t3: D} \]
\[\text{t4: D} \]
\[\text{t5: D} \]
\[\text{t6: D} \]
\[\text{t7: L(9)} \]
\[\text{a: L(0)} \]
\[\text{b: L(0)} \]
\[\text{prod: L(0)} \]
\[i: \text{D} \]
1: \( t_1 := 4 \times i \)  
2: \( t_2 := a[t_1] \)  
3: \( t_3 := 4 \times i \)  
4: \( t_4 := b[t_3] \)  
5: \( t_5 := t_2 \times t_4 \)  
6: \( t_6 := \text{prod} + t_5 \)  
7: \( \text{prod} := t_6 \)  
8: \( t_7 := i + 1 \)  
9: \( i := t_7 \)  
10: if \( i \leq 20 \) goto...

\[ \begin{align*}
t_1 & : D \\
t_2 & : D \\
t_3 & : D \\
t_4 & : D \\
t_5 & : D \\
t_6 & : D \\
t_7 & : D \\
a & : L(0) \\
b & : L(0) \\
\text{prod} & : L(0) \\
i & : L(8)
\end{align*} \]
1: t1 := 4 * i
2: t2 := a[t1]
3: t3 := 4 * i
4: t4 := b[t3]
5: t5 := t2 * t4
6: t6 := prod + t5
7: prod := t6
8: t7 := i + 1
9: i := t7
10: if i <= 20 goto...

© Harry H. Porter, 2006
1:   t1 := 4 * i  
2:   t2 := a[t1] 
3:   t3 := 4 * i  
4:   t4 := b[t3] 
5:   t5 := t2 * t4 
6:   t6 := prod + t5 
7:   prod := t6 
8:   t7 := i + 1 
9:   i := t7 
10:  if i <= 20 goto...
1: \( t1 := 4 \times i \)  
2: \( t2 := a[t1] \)  
3: \( t3 := 4 \times i \)  
4: \( t4 := b[t3] \)  
5: \( t5 := t2 \times t4 \)  
6: \( t6 := \text{prod} + t5 \)  
7: \( \text{prod} := t6 \)  
8: \( t7 := i + 1 \)  
9: \( i := t7 \)  
10: if \( i \leq 20 \) goto...

\[ 
\begin{align*} 
\text{t1:} & \quad \text{D} \\
\text{t2:} & \quad \text{L(5)} \\
\text{t3:} & \quad \text{D} \\
\text{t4:} & \quad \text{L(5)} \\
\text{t5:} & \quad \text{D} \\
\text{t6:} & \quad \text{D} \\
\text{t7:} & \quad \text{D} \\
\text{a:} & \quad \text{L(0)} \\
\text{b:} & \quad \text{L(0)} \\
\text{prod:} & \quad \text{L(6)} \\
\text{i:} & \quad \text{L(8)} 
\end{align*} 
\]
1:  $t_1 := 4 \times i$
2:  $t_2 := a[t_1]$
3:  $t_3 := 4 \times i$
4:  $t_4 := b[t_3]$
5:  $t_5 := t_2 \times t_4$
6:  $t_6 := \text{prod} + t_5$
7:  $\text{prod} := t_6$
8:  $t_7 := i + 1$
9:  $i := t_7$
10: if $i \leq 20$ goto ...

© Harry H. Porter, 2006

1:  $t_1 := 4 \times i$
2:  $t_2 := a[t_1]$
3:  $t_3 := 4 \times i$
4:  $t_4 := b[t_3]$
5:  $t_5 := t_2 \times t_4$
6:  $t_6 := \text{prod} + t_5$
7:  $\text{prod} := t_6$
8:  $t_7 := i + 1$
9:  $i := t_7$
10: if $i \leq 20$ goto ...

© Harry H. Porter, 2006
1:   t1 := 4 * i
2:   t2 := a[t1]
3:   t3 := 4 * i
4:   t4 := b[t3]
5:   t5 := t2 * t4
6:   t6 := prod + t5
7:   prod := t6
8:   t7 := i + 1
9:   i := t7
10:  if i <= 20 goto...

31

© Harry H. Porter, 2006

32

© Harry H. Porter, 2006
1: \( t_1 := 4 \times i \)  
2: \( t_2 := a[t_1] \)  
3: \( t_3 := 4 \times i \)  
4: \( t_4 := b[t_3] \)  
5: \( t_5 := t_2 \times t_4 \)  
6: \( t_6 := \text{prod} + t_5 \)  
7: \( \text{prod} := t_6 \)  
8: \( t_7 := i + 1 \)  
9: \( i := t_7 \)  
10: \( \text{if } i \leq 20 \text{ goto...} \)
**Algorithm (in more detail)**

**INITIALIZE the table**

Use results from LIVE-VARIABLE ANALYSIS, if available
Else, set all variables to L(0) -- LIVE after this block

Go through the instructions in REVERSE order...

**FOR each instruction DO**

Let the instruction be:

5. \( t5 := t2 \times t4 \)
6. \( x := y1 \oplus y2 \)

Look up the current status of each variable (\( x, y1, y2, ... \))

Fill in the NEXT-USE info for this instruction.

Set the status of “\( x \)” to “D”
Set the status of “\( y1 \)” to “L(\( n \))”
Set the status of “\( y2 \)” to “L(\( n \))”

**ENDFOR**

---

**The Point of This?**

\( x := y - 68; \)

- If the defined variable is DEAD after this statement...
  Eliminate the statement.
- If the defined variable is LIVE, but has no Next-Use in this block...
  No need to keep it in a register.
  Write back to memory immediately.
- If a used variable is DEAD...
  We can re-use its register.

**Example:**

<Assume \( y \) is in R4>

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUB 68, R4</td>
<td></td>
</tr>
<tr>
<td>MOV R4, x</td>
<td></td>
</tr>
</tbody>
</table>

Otherwise, use another register:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV R4, R5</td>
<td></td>
</tr>
<tr>
<td>SUB 68, R5</td>
<td></td>
</tr>
<tr>
<td>MOV R5, x</td>
<td></td>
</tr>
</tbody>
</table>
Code Generation Algorithm #2

• Generate code for each Basic Block in isolation.
• Assume that Next-Use info. is available (see previous algorithm).
• Go through the statements (in FORWARD order).
• Try to keep variables in registers...
  Leave as long as possible in register.
  Store back to memory only when necessary.
  Some variables may be left in registers for several instructions.
• At the end of the basic block,
  Move all LIVE variables back to memory.

Data Structure:
From statement to statement, we need to remember...
For each variable:
  Is it in a register? Which one?
For each register:
  Which variable(s) does it contain, if any?

Example

<table>
<thead>
<tr>
<th>IR Code</th>
<th>Code Gen. Alg. #1</th>
<th>Code Gen. Alg. #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1 := 43 * a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t1 := t1 + 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a := t1 * 4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Example

<table>
<thead>
<tr>
<th>IR Code</th>
<th>Code Gen. Alg. #1</th>
<th>Code Gen. Alg. #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 := 43 \times a )</td>
<td>LD \ a, R1&lt;br&gt;MUL 43, R1&lt;br&gt;ST R1, t1</td>
<td>LD \ a, R1&lt;br&gt;MUL 43, R1</td>
</tr>
<tr>
<td>( t_1 := t_1 + 7 )</td>
<td>LD \ t1, R1&lt;br&gt;ADD 7, R1&lt;br&gt;ST R1, t1</td>
<td>LD \ t1, R1&lt;br&gt;ADD 7, R1</td>
</tr>
<tr>
<td>( a := t_1 \times 4 )</td>
<td>LD \ t1, R1&lt;br&gt;MUL 4, R1&lt;br&gt;ST R1, a</td>
<td>LD \ t1, R1&lt;br&gt;MUL 4, R1&lt;br&gt;ST R1, a</td>
</tr>
</tbody>
</table>

Assuming \( t_1 \) is **LIVE** at the end of the block, we need to store it. If \( t_1 \) is **DEAD**, this instruction would be omitted!
Data Needed During Code Generation

Register Descriptors

For each register, which variables are currently stored in the register?

Initially, all registers are marked EMPTY.

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>a</td>
</tr>
<tr>
<td>R1</td>
<td>EMPTY</td>
</tr>
<tr>
<td>R2</td>
<td>x</td>
</tr>
<tr>
<td>R3</td>
<td>y, t1</td>
</tr>
<tr>
<td>R31</td>
<td>t2</td>
</tr>
</tbody>
</table>

Variable Descriptors

For each variable, where is its value currently stored?

- Register(s)
- Memory
- Some combination

Initially, all variables will be marked in MEMORY.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>R0</td>
</tr>
<tr>
<td>b</td>
<td>MEM</td>
</tr>
<tr>
<td>x</td>
<td>MEM, R2</td>
</tr>
<tr>
<td>y</td>
<td>R3</td>
</tr>
<tr>
<td>t1</td>
<td>R3</td>
</tr>
<tr>
<td>t2</td>
<td>R4, R31</td>
</tr>
<tr>
<td>t3</td>
<td>&lt;nowhere&gt;</td>
</tr>
</tbody>
</table>

Code Generation Algorithm #2 (Overview)

Initialize REGISTER-DESCRIPTORS to “EMPTY.”
Initialize VARIABLE-DESCRIPTORS to in “MEMORY.”

FOR EACH IR Statement DO

Let x be the defined variable (if any).
Let y and z be the used variables (if any).

x := y - z
x := y

(At this point, the REGISTER-DESCRIPTORS and VARIABLE-DESCRIPTORS tell what is in regs and where the variables are stored.)

Step 1: Determine where we will be storing the result value.
Call it “DEST”

Step 2: Move “y” into “DEST”.

Step 3: Figure out where “z” is.
Generate the instruction.

Step 4: Update REGISTER-DESCRIPTORS and VARIABLE-DESCRIPTORS.

END FOR
Generate stores for all LIVE variables.
Step 1: Determine where to put the result...

In Register?
In Memory?

Example IR instruction: \( a := b - c \)

Might generate this:

\[
\text{SUB ...,R7}
\]

Set \( \text{DEST} = \text{“R7”} \)

Or this:

\[
\text{SUB ...,a}
\]

Set \( \text{DEST} = \text{“a”} \)

IF \( y \) is already in a register (call it \( R_i \)) AND
\( y \) is DEAD after this statement AND
\( R_i \) holds no other variables THEN
\( \text{DEST} := R_i \)
Modify the descriptors to say that \( y \) is not in \( R_i \) anymore.
ELSE IF any register is empty THEN
Let \( \text{DEST} := R_j \) (where \( R_j \) is an empty register)
ELSE IF \( x \) has a Next-Use in this block OR
the operator \( \oplus \) requires a register for its destination THEN
Select an occupied register; call it \( R_k \)
How to choose \( R_k \)?
If the vars in some reg are also in mem, no spills necessary.
If the Next-Uses of vars in some reg are distant, choose it.
Generate SPILL instructions, as necessary.
Assume that \( \text{REG-CONTENTS} [R_k] = \{v,w\} \)
Generate:
\[
\begin{align*}
\text{ST} & \ R_k, \ v \\
\text{ST} & \ R_k, \ w
\end{align*}
\]
\( \text{DEST} := R_k \)
ELSE
No Next-Use in this block...
Put the result straight into memory.
\( \text{DEST} := \text{“x”} \)
END IF
**Step 2:** Determine the location of “y” and get “y” into DEST, if not already there.

**IF** y is in a register **THEN**
\[
\text{LOC}_y := R_y
\]
**ELSE**
\[
\text{LOC}_y := “y”
\]
**ENDIF**

**IF** \(\text{LOC}_y \neq \text{DEST}\) **THEN**
Generate the instruction:
\[
\text{LD/MOV} \quad \text{LOC}_y, \text{DEST}
\]
**ENDIF**

Step 3: Generate the instruction that performs the actual operation

Let \(\text{LOC}_z\) be the location of “z”  
*(If “z” is both in memory and a register, we prefer to use the register.)*

Generate the instruction
\[
\text{SUB} \quad \text{LOC}_z, \text{DEST}
\]  
*(Or whatever operation is involved)*

Update the VARIABLE-DESCRIPTOR for “x”  
...to show that it is in DEST only.

If DEST is a register, update its REGISTER-DESCRIPTOR  
...to show that it contains only “x”.
**Step 4:** Update REGISTER-DESCRIPTORs and VARIABLE-DESCRIPTORs for “y” and “z”.

If y is in a register (call it $R_y$) AND
  y has no Next-Use in this block THEN
  IF y is LIVE THEN
    Generate a “SPILL” instruction
    \[ \text{ST} \quad R_y, y \]
  END
  Modify Descriptors to say that y is no longer in any register.
END

IF z is in a register ... AND
  z has no Next-Use ...
  IF z is LIVE ...
    Generate a “SPILL” instruction
    \[ \text{ST} \quad \ldots \]
  END
  Modify ...
END

Same for “z”

---

**Special Case**

The “assign” IR Instruction:
\[ x := y \]

If y is in a register...
Don’t generate any code.
Just modify the descriptors!

<table>
<thead>
<tr>
<th>Register Descriptors</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ R_5 \quad y ]</td>
</tr>
<tr>
<td>[ R_6 \quad x ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable Descriptors</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ x \quad R_6 ]</td>
</tr>
<tr>
<td>[ y \quad R_5 ]</td>
</tr>
</tbody>
</table>
**Special Case**

The “assign” IR Instruction:

\[ x := y \]

Register Descriptors

If \( y \) is in a register...

Don’t generate any code.

Just modify the descriptors!

Variable Descriptors

---

**At the End of the Basic Block...**

After processing all IR statements in the Basic Block... generate “SPILL” instructions for any LIVE variables.

FOR each variable “\( x \)” that is LIVE at the end of the Basic Block...

Look at \( x \)’s Variable Descriptor

IF \( x \) is only in a register THEN Generate

\[ ST \quad R_i, x \]

END

END

---
### Example

**IR Instructions:**

\[ t1 := b + c \]

**Target Code:**

- \[ \text{DEST} := R0 \]
- \[ \text{LD} \ b, R0 \]
- \[ \text{ADD} \ c, R0 \]

**Register Descriptors**

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>empty</td>
</tr>
<tr>
<td>R1</td>
<td>empty</td>
</tr>
<tr>
<td>R2</td>
<td>empty</td>
</tr>
<tr>
<td>R3</td>
<td>empty</td>
</tr>
<tr>
<td>R4</td>
<td>empty</td>
</tr>
<tr>
<td>R5</td>
<td>empty</td>
</tr>
</tbody>
</table>

**Variable Descriptors**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>MEM</td>
</tr>
<tr>
<td>b</td>
<td>MEM</td>
</tr>
<tr>
<td>c</td>
<td>MEM</td>
</tr>
<tr>
<td>d</td>
<td>MEM</td>
</tr>
<tr>
<td>t1</td>
<td>-</td>
</tr>
<tr>
<td>t2</td>
<td>-</td>
</tr>
<tr>
<td>t3</td>
<td>-</td>
</tr>
</tbody>
</table>

Assume DEAD after block.
Example

IR Instructions: Target Code:

\[ t1 := b + c \]
\[ t2 := b \ast d \]

\[
\begin{array}{|c|c|}
\hline
\text{LD} & \text{b, R0} \\
\hline
\text{ADD} & \text{c, R0} \\
\hline
\end{array}
\]

Register Descriptors:

\[
\begin{array}{|c|c|}
\hline
\text{R0} & \text{t1} \\
\hline
\text{R1} & \text{empty} \\
\hline
\text{R2} & \text{empty} \\
\hline
\text{R3} & \text{empty} \\
\hline
\text{R4} & \text{empty} \\
\hline
\text{R5} & \text{empty} \\
\hline
\end{array}
\]

Variable Descriptors:

\[
\begin{array}{|c|}
\hline
\text{a} & \text{MEM} \\
\hline
\text{b} & \text{MEM} \\
\hline
\text{c} & \text{MEM} \\
\hline
\text{d} & \text{MEM} \\
\hline
\text{t1} & \text{R0} \\
\hline
\text{t2} & - \\
\hline
\text{t3} & - \\
\hline
\end{array}
\]

Assume DEAD after block.

---

Example

IR Instructions: Target Code:

\[ t1 := b + c \]
\[ t2 := b \ast d \]

\[
\begin{array}{|c|c|}
\hline
\text{LD} & \text{b, R0} \\
\hline
\text{ADD} & \text{c, R0} \\
\hline
\end{array}
\]

Register Descriptors:

\[
\begin{array}{|c|c|}
\hline
\text{R0} & \text{t1} \\
\hline
\text{R1} & \text{empty} \\
\hline
\text{R2} & \text{empty} \\
\hline
\text{R3} & \text{empty} \\
\hline
\text{R4} & \text{empty} \\
\hline
\text{R5} & \text{empty} \\
\hline
\end{array}
\]

Variable Descriptors:

\[
\begin{array}{|c|}
\hline
\text{a} & \text{MEM} \\
\hline
\text{b} & \text{MEM} \\
\hline
\text{c} & \text{MEM} \\
\hline
\text{d} & \text{MEM} \\
\hline
\text{t1} & \text{R0} \\
\hline
\text{t2} & - \\
\hline
\text{t3} & - \\
\hline
\end{array}
\]

Assume DEAD after block.

---

© Harry H. Porter, 2006
Example

**IR Instructions:**

<table>
<thead>
<tr>
<th>t1 := b + c</th>
<th>Target Code:</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD b, R0</td>
<td>ADD c, R0</td>
</tr>
</tbody>
</table>

**Register Descriptors**

<table>
<thead>
<tr>
<th>Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
</tr>
<tr>
<td>R1</td>
</tr>
<tr>
<td>R2</td>
</tr>
<tr>
<td>R3</td>
</tr>
<tr>
<td>R4</td>
</tr>
<tr>
<td>R5</td>
</tr>
</tbody>
</table>

**Variable Descriptors**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>MEM</td>
</tr>
<tr>
<td>b</td>
<td>MEM</td>
</tr>
<tr>
<td>c</td>
<td>MEM</td>
</tr>
<tr>
<td>d</td>
<td>MEM</td>
</tr>
<tr>
<td>t1</td>
<td>R0</td>
</tr>
<tr>
<td>t2</td>
<td>-</td>
</tr>
<tr>
<td>t3</td>
<td>-</td>
</tr>
</tbody>
</table>

Assume DEAD after block
Example

IR Instructions: Target Code:

<table>
<thead>
<tr>
<th>IR Instructions</th>
<th>Target Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1 := b + c</td>
<td>LD b, R0</td>
</tr>
<tr>
<td></td>
<td>ADD c, R0</td>
</tr>
<tr>
<td><strong>DEST := R0</strong></td>
<td></td>
</tr>
<tr>
<td>t2 := b * d</td>
<td>LD b, R1</td>
</tr>
<tr>
<td><strong>DEST := R1</strong></td>
<td>MUL d, R1</td>
</tr>
<tr>
<td>t3 := t1 * t2</td>
<td>MUL R1, R0</td>
</tr>
<tr>
<td><strong>DEST := R0</strong></td>
<td></td>
</tr>
</tbody>
</table>

Register Descriptors

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>t1</td>
</tr>
<tr>
<td>R1</td>
<td>t2</td>
</tr>
<tr>
<td>R2</td>
<td>empty</td>
</tr>
<tr>
<td>R3</td>
<td>empty</td>
</tr>
<tr>
<td>R4</td>
<td>empty</td>
</tr>
<tr>
<td>R5</td>
<td>empty</td>
</tr>
</tbody>
</table>

Variable Descriptors

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>MEM</td>
</tr>
<tr>
<td>b</td>
<td>MEM</td>
</tr>
<tr>
<td>c</td>
<td>MEM</td>
</tr>
<tr>
<td>d</td>
<td>MEM</td>
</tr>
<tr>
<td>t1</td>
<td>R0</td>
</tr>
<tr>
<td>t2</td>
<td>R1</td>
</tr>
<tr>
<td>t3</td>
<td>-</td>
</tr>
</tbody>
</table>

Assume DEAD after block

© Harry H. Porter, 2006
Example

**IR Instructions:**

<table>
<thead>
<tr>
<th>t1 := b + c</th>
<th>Target Code:</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD b, R0</td>
<td>ADD c, R0</td>
</tr>
</tbody>
</table>

**Register Descriptors**

<table>
<thead>
<tr>
<th>R0</th>
<th>t3</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>t2</td>
</tr>
<tr>
<td>R2</td>
<td>empty</td>
</tr>
<tr>
<td>R3</td>
<td>empty</td>
</tr>
<tr>
<td>R4</td>
<td>empty</td>
</tr>
<tr>
<td>R5</td>
<td>empty</td>
</tr>
</tbody>
</table>

**Variable Descriptors**

<table>
<thead>
<tr>
<th>a</th>
<th>MEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>MEM</td>
</tr>
<tr>
<td>c</td>
<td>MEM</td>
</tr>
<tr>
<td>d</td>
<td>MEM</td>
</tr>
<tr>
<td>t1</td>
<td>-</td>
</tr>
<tr>
<td>t2</td>
<td>R1</td>
</tr>
<tr>
<td>t3</td>
<td>R0</td>
</tr>
</tbody>
</table>

Assume DEAD after block

© Harry H. Porter, 2006

---

Example

**IR Instructions:**

<table>
<thead>
<tr>
<th>t1 := b + c</th>
<th>Target Code:</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD b, R0</td>
<td>ADD c, R0</td>
</tr>
</tbody>
</table>

**Register Descriptors**

<table>
<thead>
<tr>
<th>R0</th>
<th>t3</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>t2</td>
</tr>
<tr>
<td>R2</td>
<td>empty</td>
</tr>
<tr>
<td>R3</td>
<td>empty</td>
</tr>
<tr>
<td>R4</td>
<td>empty</td>
</tr>
<tr>
<td>R5</td>
<td>empty</td>
</tr>
</tbody>
</table>

**Variable Descriptors**

<table>
<thead>
<tr>
<th>a</th>
<th>MEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>MEM</td>
</tr>
<tr>
<td>c</td>
<td>MEM</td>
</tr>
<tr>
<td>d</td>
<td>MEM</td>
</tr>
<tr>
<td>t1</td>
<td>-</td>
</tr>
<tr>
<td>t2</td>
<td>R1</td>
</tr>
<tr>
<td>t3</td>
<td>R0</td>
</tr>
</tbody>
</table>

Assume DEAD after block

© Harry H. Porter, 2006
**Example**

**IR Instructions:**

- \( t_1 := b + c \)
- \( t_2 := b \times d \)
- \( t_3 := t_1 \times t_2 \)
- \( a := t_3 - t_2 \)

**Target Code:**

- \( \text{LD } b, R0 \)
- \( \text{ADD } c, R0 \)
- \( \text{LD } b, R1 \)
- \( \text{MUL } d, R1 \)
- \( \text{MUL } R1, R0 \)
- \( \text{SUB } R1, R0 \)

**Register Descriptors**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>t3</td>
<td></td>
</tr>
<tr>
<td>R1</td>
<td>t2</td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td>empty</td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td>empty</td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td>empty</td>
<td></td>
</tr>
<tr>
<td>R5</td>
<td>empty</td>
<td></td>
</tr>
</tbody>
</table>

**Variable Descriptors**

- \( a \): \( \text{MEM} \)
- \( b \): \( \text{MEM} \)
- \( c \): \( \text{MEM} \)
- \( d \): \( \text{MEM} \)
- \( t_1 \): \( - \)
- \( t_2 \): \( R1 \)
- \( t_3 \): \( R0 \)

Assume DEAD after block
Example

IR Instructions: Target Code:

- **t1 := b + c**
  - `LD b, R0`
  - `ADD c, R0`
  - **DEST := R0**

- **t2 := b * d**
  - `LD b, R1`
  - `MUL d, R1`
  - **DEST := R1**

- **t3 := t1 * t2**
  - `MUL R1, R0`
  - **DEST := R0**

- **a := t3 - t2**
  - `SUB R1, R0`
  - **DEST := R0**

- **<End of block>**

Register Descriptors:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>a</td>
</tr>
<tr>
<td>R1</td>
<td>t2</td>
</tr>
<tr>
<td>R2</td>
<td>empty</td>
</tr>
<tr>
<td>R3</td>
<td>empty</td>
</tr>
<tr>
<td>R4</td>
<td>empty</td>
</tr>
<tr>
<td>R5</td>
<td>empty</td>
</tr>
</tbody>
</table>

Variable Descriptors:

- **a**
- **b**
- **c**
- **d**
- **t1**
- **t2**
- **t3**

- **Assume LIVE; need to save**
- **Assume DEAD after block**