Phases of a compiler

Figure 1.6, page 6 of text
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]
"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 Ra, 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?