HW 3: Intermediate Code

CSC 4351, Spring 2014

Due: 9 April 2014

1. Suppose we change the Tiger language to use Ada's copy-in-copy-out parameter passing mechanism. If a parameter is declared with the keywords `in out`, it uses copy-in-copy-out. Otherwise, we use call-by-value as before. With copy-in-copy-out, the argument is copied into the parameter variable on entering the function. On function exit, the parameter is copied back out into the argument variable. Describe in detail which changes (if any) are necessary to semantic analysis, escape analysis, activation records, and the intermediate representation.

2. Draw the intermediate code trees for questions 7.1(f), 7.1(h), 7.2(g), 7.2(j) on p. 160.

   7.1 Suppose a certain compiler translates all expressions and subexpressions into `Tree.Exp` trees, and does not use the `Nx` and `Cx` constructors to represent expressions in different ways. Draw a picture of the IR tree that results from each of the following expressions. Assume all variables are nonescaping unless specified otherwise.

   e. `a<b`, which should be implemented by making an `ESEQ` whose left-hand side moves a 1 or 0 into some newly defined temporary, and whose right-hand side is the temporary.

   f. `if a then b else c`, where `a` is an integer variable (true if $\neq 0$); this should also be translated using an `ESEQ`.

   h. `if a<b then c:=a else c:=b`, translated using the `a<b` tree from part (e) above; the whole statement will therefore be rather clumsy and inefficient.

   7.2 Translate each of these expressions into IR trees, but using the `Ex`, `Nx`, and `Cx` constructors as appropriate. In each case, just draw pictures of the trees; an `Ex` tree will be a Tree `Exp`, an `Nx` tree will be a Tree `Stm`, and a `Cx` tree will be a `Stm` with holes labeled *true* and *false* into which labels can later be placed.

   g. `if a then b else c`, where `a` is an integer variable (true if $\neq 0$).

   j. `if a<b then c:=a else c:=b`