3460: 641 PROGRAMMING PROJECT
PART II: Code Generation and Optimization

Given an ONEIL program modified as in Part I, do some transformations on its abstract syntax tree and generate an ONEIL program whose expressions satisfy the given requirements.

Control Flow
First, map the control flow within the programs code. Identify all basic blocks and their predecessor and successors. Assign each block a unique identifier with global naming scope. Record this information within the program via comments; see the example for details. (Note the use of the \texttt{rem *} notation so that these may be found quickly if need be. Also note that the example includes revisions to create a 3-op form of the program. You may assume that all benchmark programs for this assignment are already in 3-op form.)

Optimization Phases
Now design and implement the initial optimization part of the compilation. At a minimum implement global redundancy removal, dead code elimination and constant propagation. In addition to these, other optimizations may be discussed in class for implementation. You may choose to implement other techniques as well as long as they are documented in the written report.

As an example redundant assignments are highlighted in the Fibonacci listing below on the right. (More important is the realization that \( t_0 \) and \( t_1 \) are not redundant.) The one constant is \( \text{bound} \), which is italicized and underlined throughout the listing. Your optimizer will substitute the use of \( t_1 \) for those of \( t_7 \) and \( t_8 \), the value 32 for \( \text{bound} \), and then remove declarations and assignments to \( t_7, t_8 \) and \( \text{bound} \) (indicated by arrows to the right of the statements).

Requirements
Code for the two programs (the code generator and the optimizer) is to be turned into the online submission system by midnight of the due date announced in class. You should also hand in a hard copy of a short (1 – 2 pages) written report at the beginning of class on the due date. Include compilation and use instructions for your programs, a short overview of your work, a brief description of tests performed and explanations of any unexpected behavior.

Last Updated 3/21/2013, based on an idea by X. Shen. Previous revision 10/31/2011.
begin
  let bound = 32
  rem initialize array
  let idx = 0
  label L_0
    let array[idx] = -1
    let idx = idx + 1
    if (idx <= bound - 1) then goto L_0
  let array[0] = 1
  let array[1] = 1
  rem end initialize array
  let idx = 0
  prompt "The ... are:\n"
  goto L_1
label L_2
  rem fibonacci calculation
  let t_1 = idx
  let t_2 = array[t_1]
  if (t_2 != -1) then goto target
  let t_3 = idx - 1
  let t_4 = array[t_3]
  let t_5 = idx - 2
  let t_6 = array[t_5]
  let t_7 = idx
  let array[t_7] = t_4 + t_6
  rem end fibonacci calculation
  let t_8 = idx
  let t_9 = array[t_8]
  print t_9
  prompt " 
  let idx = idx + 1
label L_1
  if (idx < bound) then goto L_2
  prompt "\n"
end