Principles of Programming Languages [PLP]

Exercises on Code Generation and Optimization

1. Consider the three address code fragment to the right
   a. Partition it in basic blocks showing the resulting Control Flow Graph
   b. Show the dominator tree

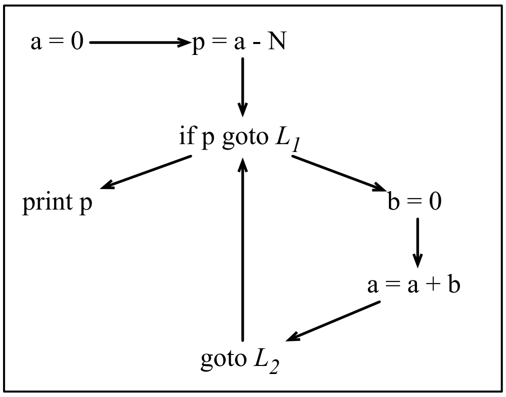
```
     k := 4
     n := 1
     i := k + 7
     if k > 0 goto L2
L1:  i := i - 1
     n := 2 * n
     if i != 0 goto L1
     goto L3
L2:  n := 2*k
L3:  halt
```

2. Consider the pseudo code program to the right
   a. Draw the Control Flow Graph representation of the program
   b. Apply (global) liveness analysis to the CFG
   c. Draw the *conflict graph* of the variables based on the live ranges, and determine the minimum number of registers needed execute the program without spilling during runtime
   d. Assign registers to the variables **a** to **f**

```
begin
  a := readint();
  b := readint();
  c := a + b;
  if (a > b)
    d := c;
    e := 2;
    f := d + e;
  else
    d := 0;
    if (a == b)
      d := 1;
    endif
    e := 1;
    f := d + e;
  endif
  writeint(e);
  writeint(f);
end
```

4. A simple data flow analysis allows one to detect the arithmetic sign of the numeric variables in a program. This analysis associates each variable with an element in the set {+, −, 0}. For example, if a variable can only assume the values 0, 1, 2 and 3 during the execution of a program, then its *abstract state* is {0, +}.
   a. Design a set of transfer functions to compute this analysis. Assume that your underlying programming language has the instructions listed below to the left.
   b. Show the result of the sign analysis you defined to the CFG below to the right.

(a) $a = n, n \in N$
(b) $a = b, \{a, b\} \subset Var$
(c) $a = b - c, \{a, b, c\} \subset Var$
(d) $a = b + c, \{a, b, c\} \subset Var$
(e) $a = b \times c, \{a, b, c\} \subset Var$
(f) if a goto L_i, $a \in Var, L_i \in Label$
(g) goto L_i, $L_i \in Label$
(h) print a, $a \in Var$



5. On the control flow graph to the right,
   a. execute *reaching definition analysis,* showing the resulting IN[B] and OUT[B] sets for each block B
   b. execute *available expression analysis,* showing the resulting IN[B] and OUT[B] sets for each block B.