Problem 1: Control-Flow, Loops and Loop Optimizations [40 points]

Consider the code depicted below for a function with integer local variables i, a, b, c and d, and parameters of the enclosing procedure p.

```
00:   i = 0
01:   a = 0
02:   b = 1
03:   d = 1
04:   if (p = 0) goto L2
05:   L1 i = 0
06:   a = 1
07:   b = 1
08:   d = 1
09:   L2: c = 0
10:   L3: a = 0
11:   c = 0
12:   i = i + 1
13:   if (p = 0) goto L4
14:   A[c] = i
15:   c = c + 1
16:   goto L5
17:   L4: b = b + 1
18:   A[i] = a
19:   a = a + 1
20:   L5: i = i + 1
21:   d = A[i]
22:   B[i] = d
23:   if (i < 32) goto L3
24:   d = d + 1
25:   if (i < 32) goto L6
26:   b = 1
27:   c = c + 1
28:   i = 0
29:   goto L1
30:   L6: A[i] = 0
31:   d = 1
32:   if (i < 16) goto L7
33:   goto L3
34:   L7: return c
```

For this code fragment determine:

a. [05 points] The Control Flow Graph (CFG).
b. [10 points] The dominator tree and the loop(s) (identify the back edges).
c. [05 points] Use algebraic properties, copy propagation and dead code elimination to improve the execution of the body of the loop(s) identified in (b).
d. [10 points] Based on the code obtained in (c) recognize loop induction variables and transform the code to eliminate them as much as possible. Indicate which variables are basic induction variables and which are derived induction variables.
e. [10 points] Discuss if there are opportunities for loop invariant code motion in the original code versus the transformed code obtained in (d).
Problem 2: Code Duplication Transformation [20 points]

A technique used to increase the size of the basic blocks so as to improve the effectiveness of instruction scheduling is basic block replication. In the illustrative example below, the basic block BB6 could be replicated thus creating two larger basic blocks \{BB4; BB6a\} and \{BB5; BB6b\} respectively.

In this problem you are asked to outline a code transformation algorithm that looks for opportunities for this code replication by inspecting the control-flow-graph and detecting situations in which basic blocks like C in this example can be replicated. Notice that you need to exploit the notion of dominance and redo the control-flow-graph to reflect the changes of the code. Also develop an algorithm that is recursive, i.e, it can apply the same analysis and transformation repeatedly over the transformed code. For the purpose of termination, you can assume that you have a function over the control-flow-graph which evaluates the profitability of replication of the code. Once this function returns a negative result, your algorithm should stop applying the replication transformation. Argue about the correctness of your transformation.
Problem 3: Iterative Data Flow Analysis [40 points]

In this problem you will develop and show the application of the Live Variable analysis problem to the code of a given procedure. The live variable analysis problem seeks to determine for each scalar variable (or temporary register) if its contents is live. This has the fundamental application in the context of register allocation since if a given variable is no longer live at a given point then there is no need to keep it in register any longer.

Formally, a variable \( v \) is live at an execution point \( p \) if either its value is used at \( p \) or there exists an execution path from \( p \) to \( q \) along which the value of \( v \) is not written and is used in \( q \).

a. [10 points] Formalize the live variable problem as an iterative data flow analysis problem showing the equations for the gen and kill abstractions as well as the initialization of the values for each basic block and statement. In this section you need to determine if this is a forward or backward data flow problem.

b. [15 points] Apply your data flow problem formalization to the procedure code shown below (do not apply any transformations) showing the final result of the IN and OUT set of live variables for each basic block in the code.

c. [10 points] Review your work if you first apply constant propagation and dead-code elimination.