Problem 1. Control-Flow Analysis and Register Allocation [40 points]

Consider the three-address code below for a procedure with input/output arguments p0 and p1 and using several temporary variables.

01:  t5 = 0  
02:  t0 = p1  
03:  t2 = 0  
04:  t1 = p2  
05:  if (t1 > 0) goto L1  
06:  t5 = t0  
07:  t0 = 1  
08:  t2 = 0  
09:  L1:  t1 = t0  
10:   t4 = t0  
11:   if (t4 > t5) goto L2  
12:   t4 = t1 + 1  
13:   t2 = t2 + 1  
14:   goto L1  
15:  L2:  t0 = p1  
16:   t3 = t0  
17:  t1 = t2 + t3

Questions:

For this code determine the following:

a. [05 points] Basic blocks and the corresponding control-flow graph (CFG) indicating for each basic block the corresponding line numbers of the code above.

b. [05 points] Dominator tree and the natural loops in this code (if any) along with the corresponding back edge(s). Explain which a given edge is a back edge.

c. [20 points] Determine the live ranges using both notions of interference for the variables t0, t1, t2, t3, t4 and t5. Assume that you do not need registers for the parameters p0 and p1 and assume that on exit of the last basic block (the one ending in line 17) t1, t2 and t5 are live but the remainder temporaries are dead.

e. [10 points] Can you color the resulting interference graphs with 4 colors? Why or why not? Present a coloring assignment for 4 colors.

Solution:

a. [05 points] Basic blocks BB1 through BB5 shown below.

b. [05 points] The dominator tree is shown below (right). There is a single edge whose basic block at its head dominates the BB at its tail. This is the edge (3,4) and so the natural loop in this code consists of the basic blocks BB3 and BB4.
c. [20 points] The figure below shows the live ranges for each variable using line numbers. In the case of the second definition the nomenclature 10- means that the variable is alive up to the instruction on line 10 but only up to the RHS of the operation’s execution (the load phase). The nomenclature 10+ means that the variable is line beginning on this lines. As such, a variable whose live range ends at 10- and another whose live ranges begins at 10+ will not interfere.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Live Range</th>
<th>Variable</th>
<th>Live Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>t0:</td>
<td>{2 ..., 5, 6, ..., 16}</td>
<td>t0:</td>
<td>{2+ ..., 5-, 6+, 7-, 16+}</td>
</tr>
<tr>
<td>t1:</td>
<td>{4 ..., 5, 9, ..., 12, 17 ...}</td>
<td>t1:</td>
<td>{4+ ..., 5, 9+, ..., 12-, 17+ ...}</td>
</tr>
<tr>
<td>t2:</td>
<td>{3 ..., 5, 8, 9, ..., 17 ...}</td>
<td>t2:</td>
<td>{3+ ..., 8- ..., 9, ..., 17 ...}</td>
</tr>
<tr>
<td>t3:</td>
<td>{16 ..., 17}</td>
<td>t3:</td>
<td>{16+, ..., 17}</td>
</tr>
<tr>
<td>t4:</td>
<td>{12, 10, ..., 11}</td>
<td>t4:</td>
<td>{12+, 10+, ..., 11-}</td>
</tr>
<tr>
<td>t5:</td>
<td>{1 ..., 17 ...}</td>
<td>t5:</td>
<td>{1+, ..., 17 ...}</td>
</tr>
</tbody>
</table>
e. [10 points] Using the simpler definition of interference, the resulting interference graph (figure (a)) below includes a clique of size 5, namely \{t0, t1, t2, t3, t5\}. As such, the graph cannot be colored with 4 colors. Using the more refined notion of interference we have an interference graph (figure (c)) where there is a clique of size 4, namely \{t0, t1, t2, t5\} and it is possible to color the graph with only 4 colors. Figures (b) and (d) depict the corresponding colorings.
Problem 2. Code Optimization [20 points]

Consider the three-address code below already captured as a CFG for a procedure with input/output arguments passed on the Activation Record (AR) on the stack. Explicit register variables are indicated with the $ sign. All other variables are either local variables to the procedure or compiler-inserted temporary variables. The translation of the accesses to local variable and array variables \( A \) and \( B \) are not yet translated into accesses to the corresponding fields of the AR.

Questions:

a. [10 points] Identify opportunities for the application of constant propagation, common-sub-expression elimination (CSE), algebraic simplification and strength reduction.

b. [10 points] Identify and exploit opportunities for loop invariant code motion (LICM) identifying basic induction variables of the loop(s) in this code. Furthermore, perform dead code elimination taking into account the results of live-variable analysis.

In both cases, reasons about the legality of the code transformation with respect to the control-flow and the notion of dominance.
Solution:

a. [10 points] Regarding constant propagation, one can propagate the value of variable `b` defined in line 05 as the value 1 and rewrite lines 12, 22 and 24. The codes in lines 23 and 26 can also be rewritten as the sub-expression `(b * c)` is common across the two statements. There are opportunities for simple strength reduction computation in lines 12 and 16. The figure below depicts the revised code after applying these transformations.
b. [10 points] The computation of \((z^*c)\) in line 22 (of the revised code) is clearly loop invariant and can be moved to a loop pre-heard basic block. Both operands of this expression are themselves loop-invariant. While \(c\) is defined in BB1 the computations of \(z\) consists of \(B[1]\) which is never modified in the loop. Regarding induction variables, there are two. One, trivially identified as the statement \(i = i + 1\) in line 28. There is a non-trivial induction variable established between variables \(t1\) and \(t2\).

Noted as well, that \(i\) is a dead variable and there are also some opportunities for copy propagation. The figure below depicts the revised code after applying these transformations.
Problem 3: Iterative Data-Flow Analysis [40 points]

Your task is to formalize and apply the Copy-Propagation data flow analysis for a problem where we want to determine which pairs of variables “bound” by an assignment of the form \( a = b \) at program point \( p \) reach a given program point \( q \) if along any program path neither or \( a \) nor \( b \) are redefined. For instance, the assignment \( b = \text{arg2} \) reaches BB2 and this the predicate \( (a = b) \) can be rewritten as \( (a = \text{arg2}) \).

Questions:

Describe your approach to anticipation analysis by answering the following questions:

a. [05 points] What is the set of values in the lattice and the initial values?

b. [05 points] What is the direction of the problem, backwards or forward and why?

c. [05 points] What is the meet function for this data-flow problem, i.e., the GEN and KILL and the equations the iterative approach needs to solve?

d. [05 points] How do you construct the transfer function of a basic block based on the GEN and KILL at the instruction level or another algorithmic method?

e. [10 points] Describe transformations and optimizations that can leverage the information uncovered by this analysis in general and in particular to the example code provided. Provide a couple of examples to illustrate your points.
Solution:

a. [05 points] At each program point we will keep track of which pairs of variables are bound to the same value at run time, i.e., we have an unordered set of tuples of the form \( \{u,v\} \). The initial values will be the empty set.

b. [05 points] The direction of the data flow problem is forward. At each program point we are asking if the binding of two variables is still value. As such as progress forward until there is one assignment to either of the variables that affect each tuple. As such the flow of information is naturally forward along the control flow.

c. [05 points] At the confluence of control-flow paths we intersect the sets as we need to ensure that along all paths the binding of two variables will hold. Regarding the equations they are as shown below reflecting the forward-looking nature of this data-flow problem.

\[
\begin{align*} 
\text{OUT} &= \text{Gen} \cup (\text{IN} - \text{Kill}) \\
\text{Gen} &= \{ <v,u> | v = u \text{ is the statement} \} \\
\text{Kill} &= \{ <v,u> | \text{LHS var. of an assignment stmt. is either } v \text{ or } u \} 
\end{align*}
\]

d. [05 points] The Gen can be accomplished by a single forward pass on the instructions of a basic block where we just keep track at each instructions what variables are assigned on the LHS of the assignment and are not subsequently killed by other assignments. The Kill set is simply the merge of all the variables that show up on the LHS of the assignments.

e. [10 points] This fact that we know that two variables will hold the same value will allow use to replace one of them, in the hope that the corresponding variable (and consequently the instructions that make the assignments, become dead. In addition, and when combined with constant propagation this can lead to even more dead-code elimination. In this particular example the assignment to the c variable is basic block BB1 and BB3 are dead. The only association that reaches the use of c in BB4 for example is the assignment c = arg1. As such the predicate in this BB4 can be rewritten as if \((b = \text{arg1})\) goto L1. Notice also that the two associations that reach BB6 for c are the same \((c, \text{arg1})\) and as a result the return statement can be rewritten as return \((b+\text{arg1})\). After this the references to c can simply be removed from the code.