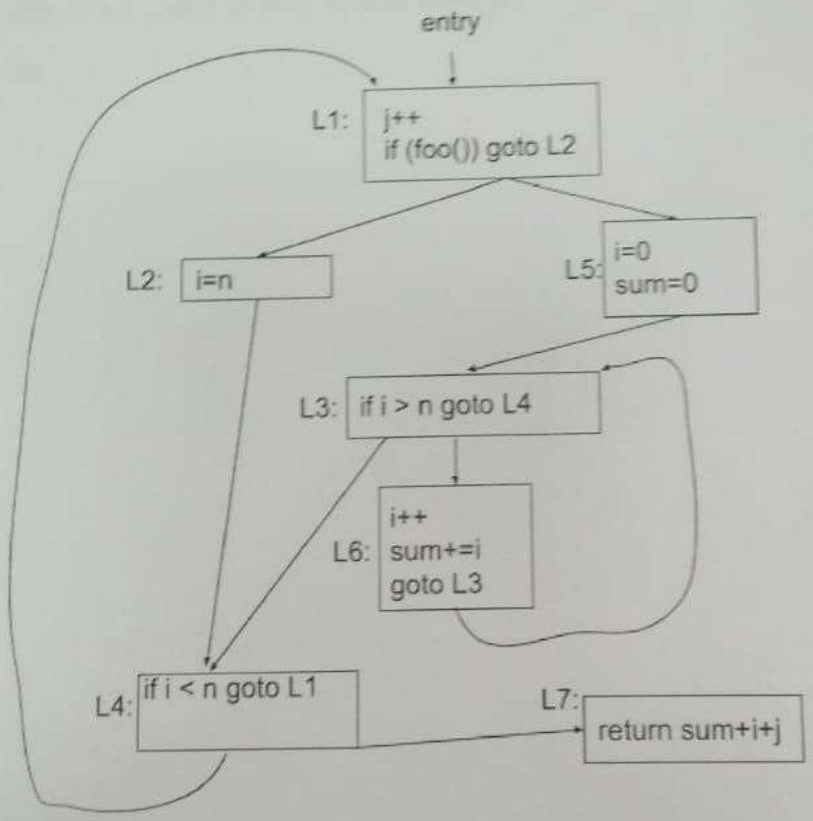
COL729 Major Exam
Compiler Optimization
Sem II, 2022-23

Max. Marks: 40

Answer all five questions

1. For the following program, answer the questions that follow.



1a, Is this program's control flow graph reducible? If not, how will you convert this to a reducible control-flow graph? After/If this control flow graph is reducible, identify all the natural loops in this program. Recall that a natural loop is identified by a back edge and all the constituent nodes of the loop. [4]

1b. What will be the results of running a (flow-sensitive) available expressions analysis on this program? Show the dataflow analysis (DFA) values at the beginning and the end of every basic block.

Use the standard GEN/KILL-based transfer function in your available expressions analysis.

For example, for a statement s of the form z: = x + y, GEN(s) = { (z, x+y) } and KILL(s) = { all expressions that involve z either on the LHS or on the RHS }.

Similarly, for a statement s of the form z: = c, GEN(s) = { (z, c) } and KILL(s) = { all expressions that involve z either on the LHS or on the RHS }.

[4]

1c. Now, we will improve the results of the available expressions dataflow analysis by using a region-based analysis. Show the region hierarchy for this program, using the common region hierarchy scheme used by compilers (as discussed in the class). [4]

1d. For loop regions in the constructed region hierarchy, use a induction variable analysis to improve the precision of the available expressions analysis.

For the induction variable analysis, write the relationships between program variables using LLVM-style recurrences for scalar evolution. Also, use a loop trip count analysis to determine the relationship between the total number of iterations of a loop (on every loop entry) and the value of the program variables (at loop entry).

Show the revised (flow-sensitive) datataflow analysis (DFA) values at the beginning and the end of every basic block (based on your region hierarchy and the results of the induction variable analysis). [4]

1e. Using the results of the (region-based) available expressions analysis, in conjunction with other standard DFAs that you are aware of, identify standard optimization opportunities in this program. Draw the control-flow graph of the optimized program below. [4]

11. What will be the results of running a **flow-insensitive** available expressions analysis on this program? Do not convert the program into SSA before running the flow-insensitive analysis. Show the dataflow analysis (DFA) values at the beginning and the end of every basic block.

Use the standard GEN/KILL-based transfer function in your available expressions analysis. Do not use a region-based analysis for this question. [4]

1g. Identify the dominance frontiers of all nodes in this program using the optimized algorithm discussed in class. Show the DF_up and DF_local sets for each node, and then show the final computed dominance frontiers of each node [4]

1h. Convert this program to miniimal-SSA by using the results of the dominance frontiers analysis Show the final program which is in minimal-SSA form [3]

2. Consider the following C++ program that involves memory allocations and memory accesses.

```
class List {
private:
  int m_val;
  List* m_next;

public:
   List(int val) : m_val(val), m_next(nullptr)
   { }

   List(int val, List* next) : m_val(val), m_next(next)
   { }

   List* append(int val) const
   {
      return new  List(val, this);
   }
};


   int main()
   {
      List* hd1 = new List(1);
      int i = 0;
      List* hd2 = hd1;
      do {
         hd2 = hd2->append(i);
      } while (foo());
      ....
   }
```

a. Show the points-to graph obtained by running a context-sensitive Andersen's points-to analysis on the program above, using the allocation-site abstraction and a **call-context of depth one**. [3]

b. Also show the points-to graph obtained by running an object-sensitive Andersen's points-to analysis with an **object context of depth 2.** [3]

c. Show the points-to graph obtained by running an object-sensitive **Steensgard's** points-to analysis with an **object context of depth 2.** [3]