University rules dictate strict penalties for any form of academic dishonesty. Looking sideways will be penalized. Look at only your own exam at all times.

There are 7 questions, some with subparts. Read them carefully to avoid throwing away points!! Write your answer in the space provided. Closed book, closed notes. Calculators are allowed.

**Partial credit rule:** Must show your intermediate steps clearly for partial credit!

1. Fill in the blanks with one, two, or at most three words:
   (1 point * 15 = 15 points)

   (a) Compiler usually represents the program internally using architecture-independent instructions whose specification is called its _______ intermediate form _________.

   (b) The _______ linker _________ is a software that runs after the compiler that allows multiple object files to be combined into a single executable.

   (c) Before register allocation is run, the operands of register-only instructions are _______ virtual registers ________ which are infinite in number.

   (d) A control-flow graph represents control flow within a _______ procedure _________.

   (e) In a _______ reducible _________ flow graph, all the cycles in the graph are in loops.

   (f) The _______ dominator tree ________ is a data structure that succinctly represents the information about dominance between flow-graph nodes.

   (g) A tree is a graph in which each node has only one _______ parent _________.

   (h) A/an _______ induction _______ variable is one whose only update in a loop is an increment by a constant value.

   (i) Elementary statements in the language for traditional dataflow analysis are called _______ quadruples _______.

   (j) In compiler terminology, a/an _______ affine _______ function is a linear combination of enclosing loop induction variables.

   (k) Dead-code elimination is most naturally performed using the results of the _______ liveness analysis _______, which is a type of traditional dataflow analysis.

   (l) _______ Constant folding _______ is the name of an optimization that executes program instructions at compile-time if their source operands are known.

   (m) In SSA form, the number of arguments of a φ function inserted in a node is equal to the number of _______ predecessors ________ of that node.

   (n) For _______ backward _______ dataflow analysis, an end node must be inserted into the flow graph for proper iterative computation of the dataflow results.
(o) The ____________ optimization can eliminate some variable-to-variable copy instructions from the program.

2. For each subpart (i) to (iii) below, circle all correct answers from among the four given - note that more than one answer may be correct!

   (2 points * 3 = 6 points)

   (i) Traditional dataflow analysis

   (a) needs the concept of control-flow graphs to compute its results.

   (b) is linear in runtime in the number of instructions in the compiled program.

   (c) can enable strength reduction of induction variable computations.

   (d) is a collection of analyses mechanisms rather than a single such mechanism.

   (ii) Alias analysis

   (a) is a form of dataflow analysis.

   (b) can be done in a limited way by using the address-taken information for each variable.

   (c) can affect the results of dataflow analysis.

   (d) can represent its information as a bi-partite graph.

   (iii) Induction variable analysis

   (a) helps identify affine function accesses.

   (b) produces results that are used by dead code elimination.

   (c) is essential for correctly compiling programs into executable form.

   (d) enables hoisting in some cases.
3. Consider the following control flow graph:

(a) Using intuitive reasoning, draw the dominator tree of the above CFG.

(b) State a formal definition of a back edge. What are the back edges in the CFG?

An edge (n,h) is a back edge if h dom n,

Back edges in above CFG: (3,1) and (4,1).

(c) How many natural loops are present in the above CFG? For each, list the member nodes and the header.

There are two natural loops, since each back edge induces a natural loop.

Loop for (3,1) = {1,2,3}; header = 1.
Loop for (4,1) = {1,2,4}; header = 1.
4. It is sometimes useful to know if the value of a variable is input dependent. For this reason, in this question we will define a new traditional data flow analysis, called **input-dependent variables**. In this analysis, the $\text{in}[n]$ and $\text{out}[n]$ sets for each instruction $n$ in the program contain the set of variables whose values are input-dependent. A variable is said to be input-dependent if its value could be affected by the values of the external inputs to the program. Assume that the `get_input()` library routine, which may be called from the program, is the only way to obtain an external input value (for example, from a file).

(a) For this forward analysis, write down the formulation for $\text{in}[n]$ and $\text{out}[n]$ for each statement $n$ in the program.

$$\text{in}[n] = \bigcup_{p \in \text{pred}[n]} \text{out}[p]$$

$$\text{out}[n] = \text{gen}[n] \cup (\text{in}[n] - \text{kill}[n])$$

(b) Write down the gen and kill sets for the **input-dependent variables** analysis in the table below. Assume that a function $\text{ID}(x)$ is available for each read operand $x$ of each statement $n$, which returns TRUE if $x \in \text{in}[n]$ and FALSE otherwise (i.e., $\text{ID}(x) = x$ is input-dependent). You can use $\text{ID}(x)$ in writing $\text{gen}[s]$ and $\text{kill}[s]$. Assume that $\text{ID}(f)$ is also available for each function $f$ where $\text{ID}(f)$ is TRUE if function $f$ returns an input-dependent value for any one of its return statements. Although $\text{ID}(f)$ is itself computed using dataflow equations, you do not need to show its computation.

(c) What should be the initial values of $\text{in}[n]$ and $\text{out}[n]$ to correctly solve the equations in the previous parts using a fixed point iteration?

$$\text{in}[n] = \text{NULL set (for all n)}.$$  

$$\text{out}[n] = \text{NULL set (for all n)}.$$
5. Consider the following program fragment:

   a = 0
   While (sum < x) {
      sum = sum + a
      a = 5
   }

(a) The assignment `a = 5` cannot be hoisted out of its enclosing while loop. State an intuitive reason why. Also state the formal rule whose being false prevents hoisting in this case.

   Intuitive reason: The value of `a` in `sum = sum + a` in the first iteration of the loop is defined outside the loop, and thus hoisting makes the code incorrect.

   Formal rule whose being false prevents hoisting in this case: `a` should not be live-out of the loop pre-header.

(b) Describe a compiler optimization in words that will allowing the hoisting of the `a = 5` statement out of its enclosing loop. Show the resulting code after applying this optimization and hoisting to the code above. Do not apply constant propagation or folding.

   Compiler optimization: Do the first iteration in an `if` statement, and subsequent iterations thereafter in a `while` loop inside of the `if` statement.

   Code after optimization and hoisting:

   ```
   a = 0
   if (sum < x) {
      sum = sum + a
      a = 5
   While (sum < x) {
      sum = sum + a
   }
   ```

6. Prove by contradiction that if `a dom b` and `b dom c`, then `a dom c`.

   Assume for the sake of contradiction that `a dom c`.

   `⇒` ∃ Path P from S0 to c not containing a. (1)

   We know that P must contain b since `b dom c`. // from definition of dominance.
   We also know that portion of P from S0 to b must contain a. // from definition of dominance.

   `⇒` P contains a. (2)
   (1) and (2) are a contradiction.

   `⇒` a dom c.
7. Pointer analysis is run on the following C program without optimizing the program in any way:

```c
main() {
    int a=5, b=3, c = 7; *p, **q;
    q = &p;
    if (foo(a)) {
        p = &a;
    } else {
        p = &b;
    }
    a = *p + 2;  // Statement 1
    *q = &c;
    b = *p + 4;  // Statement 2
}
foo(int x) { return x > 0; }
```

(a) For a flow-sensitive pointer analysis, what is the set of variables that p can point to in statement 1?

\{a, b\}

(b) For a flow-insensitive pointer analysis, what is the set of variables that p can point to in statement 1?

\{a, b, c\}

(c) For a flow-sensitive analysis, at the end of statement 2, state the value (TRUE or FALSE) of (i) *p may-alias a; and (ii) p may alias &q.

* p may-alias a = TRUE
  p may alias &q = FALSE