Quiz 2
6.035 Spring 2023 May 12, 2023

Before starting the quiz, write your name on this page.

There are 7 problems on this quiz. It is 14 pages long; make sure you have the whole quiz. You will have 50 minutes in which to work on the problems. You will likely find some problems easier than others; read all problems before beginning to work, and use your time wisely. The quiz is worth 67 points total. The point breakdown for the parts of each problem is printed with the problem. Some of the problems have several parts, so make sure you do all of them!

This is an open-book quiz. You may use a laptop to access anything on or directly linked to from the course website. You may also use any handwritten notes. You **may not** use the broader internet, any search engines, large language models, or other resources.

Do all written work on the quiz itself. If you are running low on space, write on the back of the quiz sheets and be sure to write (OVER) on the front side. It is to your advantage to show your work — we will award partial credit for incorrect solutions that are headed in the right direction. If you feel rushed, try to write a brief statement that captures key ideas relevant to the solution of the problem.

Name ______________________________

Kerberos ______________________________

Problem	Points	Score	Grader
1	14		
2	6		
3	5		
4	9		
5	9		
6	10		
7	14		
Total	67		

1. **True/False** [14 pts] (parts a–g)

a. ____ One benefit of function inlining is that it allows you to use caller-save registers more freely.

b. ____ In order to ensure that generated code is correct, all dataflow analyses must be ran to completion (i.e., fixed-point convergence) before applying any corresponding optimization.

c. ____ For a statement to be moved out of a loop (into its pre-header) during loop invariant code motion, it must be true that the statement dominates all exit nodes of the loop before the statement is moved into the loop header.

d. ____ The register live range interference graph discussed in class has each node represent a single use of a variable and each edge indicating that the two uses are of the same variable.

e. ____ The graph coloring based register allocation heuristic discussed in class is guaranteed to generate code that uses the fewest possible number of registers.

f. ____ In a set of nested loops, if the innermost loop has data dependencies exclusively from prior iterations of one of the outer loops, then the innermost loop can always be parallelized.

g. ____ Given a finite lattice, all sound dataflow analyses using the worklist algorithm will eventually terminate.

2. **Anticipated Expressions Analysis** [6 pts] (parts a–c)

In this problem we will run an *anticipated expressions analysis*. This analysis computes sets of expressions $c+d$ at a given program point if all paths leading from that program point eventually compute the value of the expression $c+d$ from the values of c and d available at that point. This is a backwards analysis defined as follows:

$$\text{OUT}[b] = \bigcap_{b' \in \text{SUCC}[b]} \text{IN}[b']$$

$$\text{IN}[b] = (\text{OUT}[b] - \text{KILL}[b]) \cup \text{GEN}[b]$$

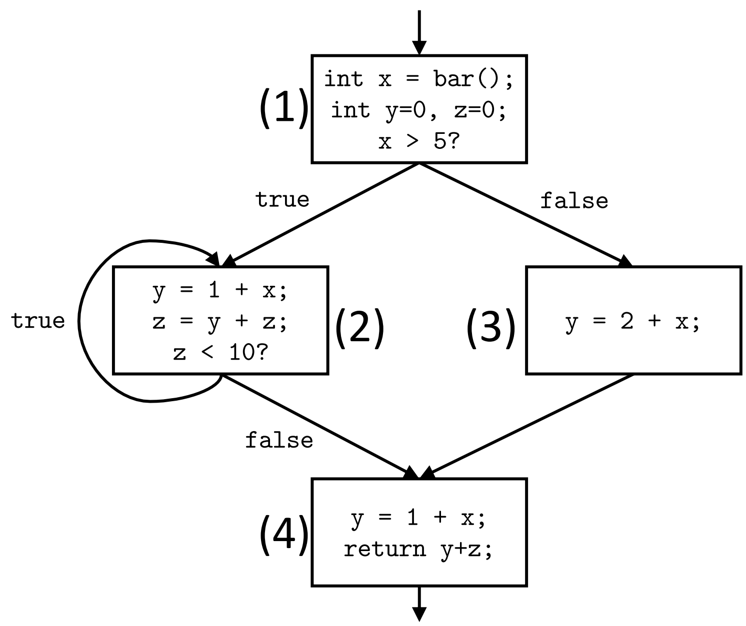
$$\text{OUT}[\text{EXIT}] = \varnothing$$

$\text{GEN}[b] =$ set of expressions $c+d$ computed in b where neither c nor d are redefined earlier in b

$\text{KILL}[b] =$ set of expressions $c+d$ where either c or d is redefined in b

$\text{IN}[b]$ is initialized to be the set of all expressions.

We will use the following CFG:



Throughout this question we will use bitvector notation to represent sets containing the following expressions:

$$1 : 1 + x$$
$$2 : 2 + x$$
$$3 : y + z$$

For example with this notation, the vector 011 represents the set of expressions $\{2 + x, y + z\}$.

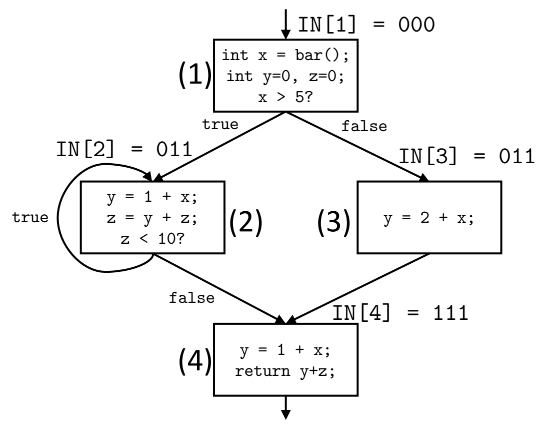
Continued on next page...

(a) [2 pts] Give GEN and KILL for block (2) in the CFG:

GEN[(2)] =

KILL[(2)] =

Now consider an execution of the worklist algorithm for this analysis at the following state (showing the IN of each block). Note that this specific state may not result from an execution of the algorithm as described above.



(b) [2 pts] Give the OUT for block (2) computed in a single iteration of the worklist algorithm if block (2) is chosen to come off the changed set in the state above.

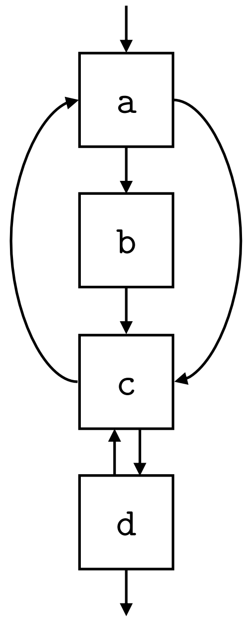
OUT[(2)] =

(c) [2 pts] Give the IN for block (2) computed in a single iteration of the worklist algorithm if block (2) is chosen to come off the changed set in the state above (based on the OUT computed above).

IN[(2)] =

3. **Dominators and Loops** [5 pts] (parts a–b)

   Consider the following CFG:



   (a) [2 pts] Draw the dominator tree for the CFG.

   (b) [3 pts] For each loop in this CFG give 1) the back edge of the loop, and 2) the set of nodes in the loop. Remember that a loop must have a unique entry point (the header) and a back edge, and back edges are those for which the head dominates the tail.

4. **Loop Optimization** [9 pts] (parts a–c)

   Consider the following code:

```
int foo(int x) {
  int i = 1, j = 2, a = 3;

  while (i < x) {
    a = x + 3;
    i = i + 5;
    j = i*3;
  }

  return a + j;
}
```

   (a) [3 pts] Write the induction variable triple $x = \langle y, c, d \rangle$ for variables `i` and `j`.

   (b) [4 pts] Rewrite the above function after performing strength reduction for `j` and induction variable elimination for `i` (and no other optimizations).

(c) [2 pts] Can we move the statement `a = x + 3;` into the loop pre-header? Explain whether this is always possible, never possible, or potentially possible given additional information about the program execution.

5. **Register Allocation** [9 pts] (parts a–c)

Consider the following code:

```c
int foo() {
  int a = 1, b = 2;

  if (a > b) {
    int c = 1;
    c += a * 2;
    a += c;
  } else {
    int d = 2;
    d += a * 4;
    a += d;
  }

  return a + b;
```

Your task will be to assign variables `a`, `b`, `c`, and `d` to registers.

(a) [3 pts] Draw an interference graph for variables `a`, `b`, `c`, and `d`. Do not perform any optimizations on the code.

(b) [3 pts] Assume you're given three registers (`%r8`, `%r9`, `%r10`) to allocate variables into. You are also allowed to *split* a variable's live range; if you choose to do this, then state where the variable will be stored and loaded, and draw a new interference graph with the variable v split into v_1 and v_2. State which variables should get assigned to which registers such that the total number of stores and loads is minimized (note that there may be multiple correct solutions). Again, do not perform any optimizations on the code.

(c) [3 pts] Now assume you're given only two registers (`%r8` and `%r9`).
The problem statement is otherwise identical: you are also allowed to *split* a variable v into two variables v_1 and v_2 by splitting its live range with a load and store; if you choose to do this, then state where the variable will be stored and loaded, and draw a new interference graph with the variable v split into v_1 and v_2. State which variables should get assigned to which registers such that the total number of stores and loads is minimized (note that there may be multiple correct solutions). Again, do not perform any optimizations on the code.

6. **Parallelization** [10 pts] (parts a–d)

Consider the following program:

```c
int foo(int *a, int *b) {
  int i, j;

  for (i = 1; i < 4; i++) {
    for (j = i; j < i + 3; j++) {
      b[i][j] = b[i-1][j] + a[i];
    }
  }

  return 0;
}
```

(a) [2 pts] Say that executing method `foo` takes 50% of the whole program's total run time. How much speedup on the whole program would we get by accelerating `foo` by a factor of 10×? (Do not worry about simplifying any fractions)

(b) [4 pts] Consider the loop iteration space below. Mark each square that is executed. Draw an arrow from square (i, j) to (i', j') if the iteration at (i', j') has a dependency (true, anti, or output) on prior iteration (i, j).

i↓ j→	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0								
1								
2								
3								
4								
5								
6								
7								

(c) [2 pts] What is the distance vector?

(d) [2 pts] According to the distance vector test described in class, which loop(s) can be parallelized?

7. **Dataflow Foundations** [14 pts] (parts a–e)

Your partner is scared of the number 3, and wants to be warned if programs can return it from any function. They have asked you to implement a dataflow analysis that will check whether functions can return 3.

To do this, you decide to implement a program analysis that tracks all possible values a variable can be at a given point in the program. **For simplicity, the lattice tracks only the possible values of a single variable (x).** Elements in your lattice are thus sets of integer values. Note that your lattice does not contain distinguished $\top$ and $\bot$ elements.

You may find it helpful to remember that as defined in class, a program state s at a given point in execution is a mapping from variables to their concrete values. You may also find the following notation to be helpful in this problem:

Notation	**Definition**
$\varnothing$	The empty set
$\mathbb{Z}$	The set of all integers
$\cup$	Set union
$\cap$	Set intersection
$\setminus$	Set difference
$\subseteq$	Subset
$\supseteq$	Superset
$s[v]$	The value of variable v in state mapping s

(a) [5 pts] For each of the following, give the corresponding value in the lattice:

The top element of the lattice ($\top$)	
The bottom element of the lattice ($\bot$)	
The $\leq$ relation between lattice elements	
The $\vee$ operation of the lattice	
The $\wedge$ operation of the lattice	

(b) [2 pts] Give the abstraction function mapping from program states to lattice values. As input, your abstraction function should take a program state. As output, it should produce the lattice value that abstracts the program state as precisely as possible.

AF(s) =

(c) [3 pts] Write the most precise, sound transfer function for each statement:

x = 3	x += 1	x = y

Consider the following code:

```
int foo() {
  int x;
  if (b) {
    x = random(1, 3); // returns any integer between 1 and 3, inclusive
  } else {
    x = 1;
  }
  return x;
}
```

(d) [2 pts] Your partner's implementation uses $\wedge$ to combine values at control-flow join points. What set of values does this implementation say that this function can return?

(e) [2 pts] Argue whether or not this implementation leads to a correct analysis for this program, meaning (as discussed in class) that for all possible program states s at a given program point and for the analysis result in_n at that program point, $\text{AF}(s) \leq \text{in}_\text{n}$.