You have 50 minutes to complete this exam.

This is a closed-book, closed-notes exam.

Do not share anything with other students. Do not talk to other students. Do not look other students’ exams. Do not expose your exam to easy viewing by other students. Violation of any of these rules will count as cheating.

If you believe there is an error or an ambiguous question, you may seek clarification from the instructor. Please speak quietly or write your question out.

Including this cover sheet, there are 8 pages to the exam, including one blank page for workspace. Please verify that you have all 8 pages.

Please write your name and NetID in the spaces above.

Show your work. Partial credit will be given for incomplete answers.

If you finish with time remaining, check your work!
<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
Problem 1. (5 points)
Name one feature that is present in most intermediate representations but not assembly languages, or vice versa.

Solution: Most IRs have an infinite number of variables/temporary registers available, while assembly languages have a fixed number of machine registers. (There are various other differences as well: multiple-label conditional jumps vs. fall-through, complex expressions vs. one operation per instruction, etc.)

Problem 2. (15 points)
Which of the following is a correct IR translation of this program:
while index < 10 do index := index + 1

a) Seq

```plaintext
Seq
  Seq
  Label start
  Cjump LT index 10
  body
  done
  Label
```

b) Seq

```plaintext
Seq
  Seq
  Label start
  Cjump LT index 10
  body
  done
  Label index 1
  Jump
  body
  index
  Plus
  Label
  Jump
  Label
```
Problem 3. (20 points)
Suppose you were implementing translation to tree IR for an expression of the form try ... catch ...
finally ... such that try e catch e1 finally e2 checks whether the value of e is 0, if so runs e1, and then runs e2 whether e is 0 or not. Write a function Tr_try that translates try-catch-finally expressions into the tree IR.

T_exp Tr_try(T_exp e, T_exp e1, T_exp e2){

Solution:

    Temp_label label1 = Temp_newLabel();
    Temp_label label2 = Temp_newLabel();
    return T_ESeq(T_Cjump(T_eq, e, T_Const(0), label1, label2),
                  T_Seq(T_Label(label1), T_Seq(T_Exp(e1), T_Label(label2))), e2);
}
**Problem 4.** (10 points)
Which of the following are valid basic blocks?

a) $x = 0$
   $y = x + 1$
   jump l1

b) body:
   $x = x + 1$
   cjump ($x < 10$) test

c) true:
   $t = 10$
   jump join
   false:

d) label1:
   $c = x + 1$
   cjump ($c == 0$) label3
   $x = 20$
   jump label5

e) entry:
   $z = 5$
   $a = z - 4$
   jump exit

**Solution:** b and e

**Problem 5.** (15 points)
Suppose you were implementing Maximal Munch for a target language with two instructions that store to memory:

storei k, r, matching the tile $\text{Move}_\text{Mem}_\text{Const}$, and

store r1, r2, matching the tile $\text{Move}_\text{Mem}$

Fill in the part of the $\text{T\_MOVE}$ case of $\text{munchStm}$ that uses these two tiles. (Be sure to only emit either $\text{store}$ or $\text{storei}$ instructions!)

```c
void munchStm(T_stm s){
    switch(s->kind){
        case T_MOVE:
            if(s->MOVE.left->kind == T_TEMP)
                emit("move 's0, 'd0", s->MOVE.left->TEMP, munchExp(s->MOVE.right));
            else if(s->MOVE.left->kind == T_MEM){
```
Solution:

```c
if(s->u.MOVE.right->kind == T_CONST){
    emit("storei %d, 's0", s->MOVE.right->u.CONST,
         munchExp(s->u.MOVE.left->u.MEM));
}
else{
    emit("store 's0, 's1", munchExp(s->u.MOVE.right),
         munchExp(s->u.MOVE.left->u.MEM));
}
}
```

**Problem 6.** (5 points)
Write the live range for each variable in the following program:

1: x = y + 1  
2: z = x + y  
3: w = load(x)  
4: v = f(z, w)  
   // v is used later

**Solution:**
x: 1-3  
y: (before this code)-2  
z: 2-4  
w: 3-4  
v: 4-(after this code)

**Problem 7.** (15 points)
The rules for reaching definition analysis are as follows:
If a node $n$ assigns to a variable $a$, then $\text{gen}[n] = \{n\}$ and $\text{kill}[n]$ is the set of all other nodes that define $a$. Otherwise, $\text{gen}[n]$ and $\text{kill}[n]$ are both empty.

\[
\text{in}[n] := \bigcup_{n' \in \text{pred}[n]} \text{out}[n']
\]

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

Perform reaching definition analysis on the following CFG. Write the final in and out set for each node. You are more likely to receive partial credit if you also show intermediate stages.
1: i = 0

2: cjmp i < 10

3: t = a + i

4: store(0, t)

5: i = i + 1

6: t = i

Solution:

\[
\begin{align*}
in[1] &= \{\} \\
out[1] &= \{1\} \\
in[2] &= \{1, 3, 5\} \\
out[2] &= \{1, 3, 5\} \\
in[3] &= \{1, 3, 5\} \\
out[3] &= \{1, 3, 5\} \\
in[4] &= \{1, 3, 5\} \\
out[4] &= \{1, 3, 5\} \\
in[5] &= \{1, 3, 5\} \\
out[5] &= \{3, 5\} \\
in[6] &= \{1, 3, 5\} \\
out[6] &= \{1, 5, 6\}
\end{align*}
\]
Problem 8. (15 points)
Consider the following interference graph with pre-colored nodes, where the available colors are r1, r2, and r3:

(a) (5 points) Can the graph be 3-colored? Why or why not?

**Solution:** No, because b and c are adjacent to each other and also to nodes colored r1 and r2.

(b) (10 points) Suppose the processor has registers r1, r2, and r3. Compute an optimal (not necessarily optimum) assignment of variables to registers. For each variable, indicate whether it is spilled, and if not, which register it is assigned to.

**Solution:** One possible assignment is:
- a: r1
- b: spilled
- c: r3
- d: r2
- e: r1
- f: r1
- g: r2