CS 553 Compiler Construction	Midterm Examination
Fall 2005	October 18, 2005

Name:

Please put your name on every sheet. The total number of possible points is 100. **Please think before you write,** so that your answers can be brief and fit in the space provided.

Total points: 100, 20% of course grade

Good luck!

1. [10 points] Induction Variables. Identify the basic and derived induction variables in the provided example and specify the triple that describes them (with the triple ordering used in the Tiger book).

```
y = 0
z = read()
loop:
if y >= 10 goto loopexit
a = y*4
b = a + z
z = M[a]
c = y-1
y = y+1
goto loop
```

2. Register Allocation. The goal is to perform register allocation on the procedure body provided below using two registers: $s0 is callee-saved and $t0 is caller-saved.

	Liveness
`t1 = 3`	
`t2 = 4 * t1`	
`t3 = 77 + t2`	
`t2 = t3 * 5`	
`t4 = t2 + 20`	

(a) [5 points] Modify the above program to model the live ranges of the callee-saved register $s0 and the caller-saved register $t0.

(b) [5 points] In the column entitled "Liveness" indicate the set of variables and/or registers that are live between each statement in the above procedure body.

(c) [10 points] Construct the interference graph based on the calculated liveness information and include the callee-save and caller-saved registers as precolored nodes.

(d) [10 points] Use optimistic simplification ala Briggs with no coalescing to color the interference graph. Label each node in the interference graph with a color to indicate the coloring. When it is necessary to select among a number of variables, select the variable with the smallest number associated with it.

(e) [5 points] Based on the coloring, indicate which variables/temporaries map to which registers below:

Variable	Register
t1	
t2	
t3	
t4	

3. Data-flow analysis and Program Optimization. The goal is to perform one pass of copy propagation followed by one pass of dead-code elimination on the code provided below. Follow the detailed instructions on the next page.

Reaching Definitions **Liveness**

```
S1:     j = 0

S2:     y = read()

S3:     x = y


S4:     z = x


S5:     L1:


S6:     if  j>10 goto L2


S7:     z = x


S8:     j = j + 1


S9:    goto L1


S10:     L2:


S11:    print x, y, z
```

(a) [5 points] Draw a control-flow graph node around each statement on the previous page and draw the appropriate control-flow edges.

(b) [10 points] Indicate the reaching definitions for each statement under the column entitled "Reaching Definitions". The reaching definition should be in terms of the statement that is the reaching definition and variable that is defined in the reaching statement. It might help to assign some of the data flow sets to temporary set variables to reduce writing. For example, T1 = {(S1, j)}.

(c) [10 points] Modify the code in-place to indicate the effects of one pass of copy propagation. NOTE: Assume a version of copy propagation that only considers propagating copies if there is only one reaching definition for the use in question.

(d) [5 points] Indicate the set of live variables for each statement in the modified program under the column entitled "Liveness".

(e) [5 points] In-place, cross out the statement(s) that are dead-code.

4. SSA and Program Optimization. The goal is to perform copy propagation when the same program is in the SSA representation. Follow the detailed instructions on the next pages.

	SSA	**SSA after Copy Prop**
S1:	`j = 0`	
S2:	`y = read()`	
S3:	`x = y`	
S4:	`z = x`	
S5:	`L1:`	
S6:	`if  j>10 goto L2`	
S7:	`z = x`	
S8:	`j = j + 1`	
S9:	`goto L1`	
S10:	`L2:`	
S11:	`print x, y, z`	

(a) [10 points] Convert the program to SSA. DO NOT use Briggs or Pruned for placing phi functions, just use the basic rule. Write the SSA representation next to the original program in the column marked "SSA".

(b) [10 points] Perform copy propagation for SSA, which simultaneously performs dead-code elimination. Do the transformation on a copy of your SSA representation and write the final version of the SSA code in the column entitled "SSA after Copy Prop". NOTE: $\phi(v_2, v_2)$ should be considered equivalent to $\phi(v_2)$.