NAME: **ROLL NO:**

Compiler Optimizations
End of Course Examination, 2012

Max Time: 1.5 Hours Max Marks: 80

NOTE:

- There is no separate answer script. **Write down the answers in the space provided on the question paper.** There is enough space for rough work too, so do not ask for extra sheets.
- Presenting your answers properly is your responsibility. You lose credit if you can not present your ideas clearly, and in proper form. Please DO NOT come back for re-evaluation saying, "What I actually meant was . . . ".
- Be precise and write clearly. Remember that somebody has to read it to evaluate!
- **ONE** A4 size *cheat-sheet* is allowed. Sharing of cheat-sheet or any other exam material is **not allowed**.

Question No.	Max Marks	Marks Obtained
1	20	
2	20	
3	40	
Total	**80**	

I pledge my honour as a gentleman/lady that during the examination I have neither given assistance nor received assistance.

Signature

1. Perform available expressions analysis, i.e. compute gen, kill, in and out for each block for the given flow graph. [20]

Block	**Gen**	**Kill**
1		
2		
3		
4		
5		
6		
7		
8		
9		

PASS:

Block	**IN**	**OUT**
Entry		
1		
2		
3		
4		
5		
6		
7		
8		
EXIT		

PASS:

Block	**IN**	**OUT**
Entry		
1		
2		
3		
4		
5		
6		
7		
8		
EXIT		

PASS:

Block	**IN**	**OUT**
Entry		
1		
2		
3		
4		
5		
6		
7		
8		
EXIT		

PASS:

Block	**IN**	**OUT**
Entry		
1		
2		
3		
4		
5		
6		
7		
8		
EXIT		

2. Perform reaching definition analysis, i.e. compute gen, kill, in and out for each block for the given flow graph. [20]

[illegible]

Block	**Gen**	**Kill**
1		
2		
3		
4		
5		
6		
7		
8		
9		

PASS:

Block	**IN**	**OUT**
Entry		
1		
2		
3		
4		
5		
6		
7		
8		
EXIT		

PASS:

Block	**IN**	**OUT**
Entry		
1		
2		
3		
4		
5		
6		
7		
8		
EXIT		

PASS:

Block	**IN**	**OUT**
Entry		
1		
2		
3		
4		
5		
6		
7		
8		
EXIT		

PASS:

Block	**IN**	**OUT**
Entry		
1		
2		
3		
4		
5		
6		
7		
8		
EXIT		

3. **Sign Analysis** :

Consider a simple programming language with only integer variables and two operations: **multiplication** $(\times)$ and **addition** $(+)$, where the operations have their natural meaning.

The language has *Positive Constants* (including Zero) and *Negative Constants*, i.e.:

Constant	**Sign**
$0, 1, 2, 3, \ldots$	Positive
$-1, -2, -3, \ldots$	Negative

The language obviously contains basic constructs like *assignment* statements, simple branches and while loops (all the cases as discussed in class).

We want to compute at each program point the **POSITIVE** set of variables, i.e. variables that have values **definitely** ≥ 0. Note that we do not want to compute the values of variables.

The basic rules, in English, are:

(a) All Positive constants are always POSITIVE.

(b) For $x = y$ (y is a constants or a variable), the variable "x" is POSITIVE at OUT if "y" is POSITIVE at IN of the statement.

(c) For $x = y + z$ (each of y and z is a constant or a variable), "x" is POSITIVE at OUT if **both** "y" and "z" are POSITIVE at IN.

(d) For $x = y \times z$ (each of y and z is a constant or a variable), "x" is POSITIVE at OUT if **both** "y" and "z" are POSITIVE at IN.

(e) Any variable other than "x" is unaffected by the assignment $x = \ldots$.

(f) Initially, all variables are considered not POSITIVE.

For example

```
// STATEMENT    // PROPERTIES
x = 5;      // x is POSITIVE at OUT
y = -1 * x; // x is POSITIVE, y is not POSITIVE at OUT
z = y * y;  // x POSITIVE, y not, z POSITIVE

if (something) {
    w = y; // w not POSITIVE
} else {
    w = x; // w POSITIVE
}
// at this point w is NOT POSITIVE because it is
// positive along one path, but not other.
```

We want to design an **intraprocedural flow-insensitive** data flow analysis to infer signs of variables for programs written in the above language. Answer the following questions.

(a) Define the gen and kill constraints for each type of basic statement (x = y, x = y + z, x = y × z). [12]

(b) Describe the gen and kill constraints for each type of structure in the program (sequence of statements, if-branch, while loop). [12]

(c) Is your Analysis *Forward* or *Backward*? Justify your answer. Also describe the *BoundaryInfo*, the initialization information at the boundary (ENTRY or EXIT) of the flow graph. [2+3]

(d) What is the optimistic value (top) and the pessimistic value (bot) for this analysis?

[5]

(e) Consider the rule for $x = y \times z$. Suppose we change the rule as follows:
For $x = y \times z$ (each of y and z is a constant or a variable), "x" is POSITIVE at OUT if "y" and "z" are either **both** *POSITIVE or* **both** *NOT POSITIVE.*
Is this rule correct? If yes, prove it. If no, give an example program where we get incorrect results. [6]