“Advanced Compiler Techniques”

Midterm Examination

Fall 2008

You have 1 hour 40 minutes to work on this exam. The examination has 100 points, so you should roughly spend about one minute for each point. Please budget your time accordingly.

You are allowed to bring one single-sided cheat sheet on an A4 paper.

Write your answers in the space provided on the exam.

NAME: ____SOLUTIONS____

Student ID: _____

Problems	Points	Score
1	15	_____
2	20	_____
3	20	_____
4	15	_____
5	30	_____
Total	**100**	**_____**

1. (15 points) True or False. No Explanation is necessary.

   a) "Dominator" relation is a partial order.

   TRUE. A partial order should satisfy reflexive, asymmetric, transitive properties. It's easier to verify these properties all hold on "dominator".

   b) "Immediate Dominator" relation is a partial order.

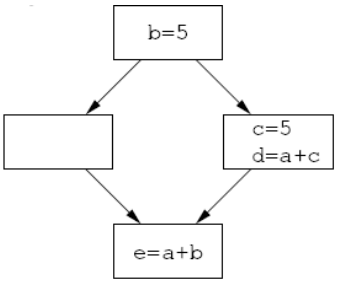
   FALSE. Reflexivity does not hold because a node cannot immediately dominate itself.

   c) If depth-first order is used, the data-flow algorithm for constant propagation converges in at most $d+2$ iterations, where d is the depth of the input flow graph.

   FALSE. Because constant propagation results changes along loops, we cannot determine the maximum number of iterations for it converges.

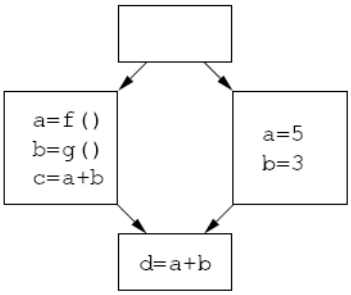
   d) Applying constant propagation before lazy code motion can improve the applicability of lazy code motion.

   TRUE. An example:



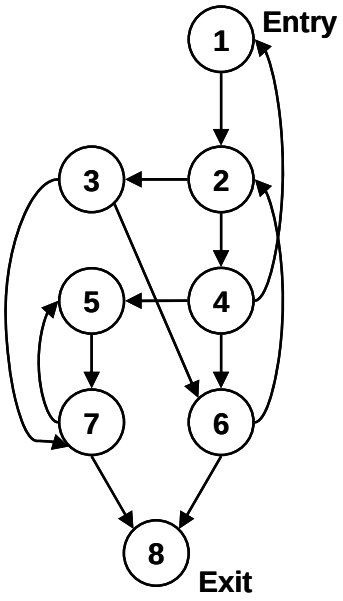
   e) Applying lazy code motion before constant propagation can improve the applicability of constant propagation.

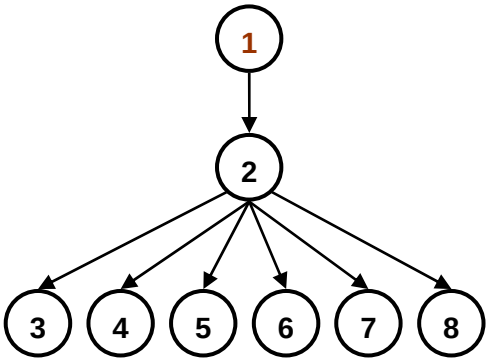
   TRUE. An example:



2. (20 points) Natural Loops

a) (5pt) Construct the dominator tree for the following flow graph.





b) (5pt) What are the back edge(s) in the above flow graph?

4->1, 6->2

c) (5pt) What are the natural loop(s) in the above flow graph?

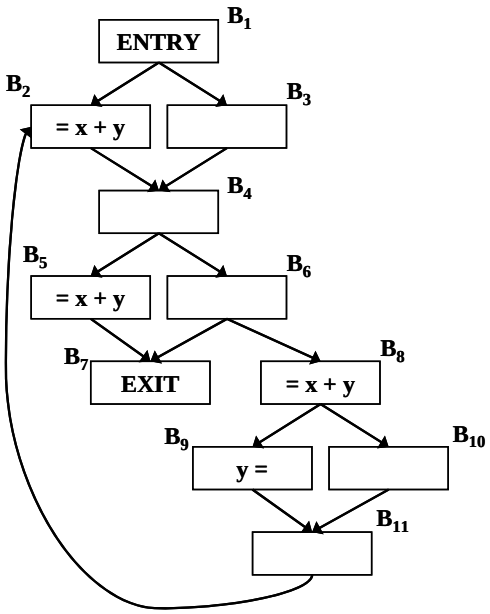
4->1 : {1,2,3,4,6}
6->2 : {2,3,4,6}

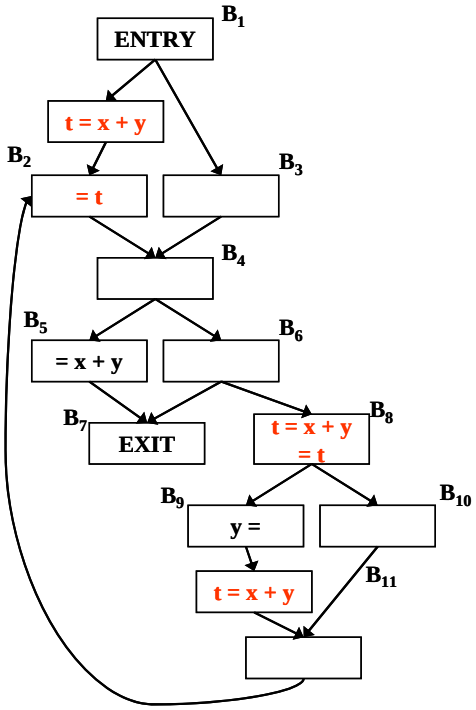
d) (5pt) Is the above flow graph reducible? Explain your answer.

No. The edge 7->5 is a retreating edge that is not a back edge. The graph is not reducible.

3. (20 points) Lazy Code Motion (PRE)

For the following program





a) (5pt) For which blocks B is x+y anticipated at the beginning? At the end? List those sets IN[B] and OUT[B] that contain x+y when we compute anticipated expressions.

IN: B2, B5, B8, B11, B10
OUT: B11, B10, B9

b) (5pt) For which blocks B is x+y "available" (in the sense used in the PRE algorithm) at the beginning? At the end?

IN: B9, B10
OUT: B2, B5, B8, B10,

c) (3pt) For which blocks B is x+y in Earliest of B?

B2, B5, B8, B11

d) (7pt) Show the result of lazy code motion, introduce new blocks if necessary. (you may show the results on the above original graph)

See above.

4. (15 points) Convert the program below to SSA form.
   a) (5pt) Construct a CFG (add ENTRY & EXIT nodes)
   b) (6pt) Insert ϕ-functions at necessary places
   c) (4pt) Rename Variables

```
k = false;
i = 1;
j = 2;   [B1]
while (i<=n) do  [B2]
{
     j = j*2;
     k = true;
     i = i+1;  [B3]
}
if (k) then  [B4]
     print j;  [B5]
else
     i = i+1;   [B6]
```

Draw your CFG with code in SSA form in the space below. (Show work for partial credit)

DF(B1) = {}, DF(B3) = {B2}, DF(B5) = {Exit} DF(B2) = {B2}

ADD Phi-functions for i,j,k in B2, for i in Exit.

5. (30 points) DFA on Value Range

In many cases knowing the range of variables is beneficial. For instance, knowing that variables *a* and *b* are between 0 and 127 may allow us to represent both variables within one byte instead of two words, thereby providing a more compact representation for certain data structures.

Suppose you are analyzing a program consisting of the following types of statements:

- `a = <const>`
- `a = b`
- `a = b + <const>`
- `a = b + c`

where all variables and constants are integers.

Your task is to formulate a dataflow problem called **VarRange** that would allow one to approximate the range of any given variable at any point in the program.

The range is to be represented by an interval **[*x*, *y*]** where both *x* and *y* are constants. Assume that **MAX** is the biggest representable integer and we are dealing with **positive** numbers (including zero) only.

a) (4 pt) What are the top and bottom elements of the lattice for the dataflow framework formulation of VarRange?

TOP: [0, MAX]
BOT: UNDEF

b) (3 pt) What is the JOIN ($\vee$) operator for VarRange?

[low1, high1] $\vee$ [low2, high2] = [Min(low1,low2), Max(high1, high2)]

c) (3 pt) What is the partial order ($\leq$) relation induced by the $\vee$ operator?

[low1, high1] $\leq$ [low2, high2] if and only if low2$\leq$low1 and high1$\leq$high2.

d) (8 pt) Assume for simplicity that each basic block consists of at most one statement. Define the transfer function for VarRange.

- a = <const> $tf(B)_a$ = [const, const]
- a = b $tf(B)_a = [low_b, high_b]$
- a = b + <const> $tf(B)_a = [low_b+<const>, high_b+<const>]$
- a = b + c $tf(B)_a = [low_b+low_c, high_b+high_c]$

e) (3 pt) Is the transfer function you defined above monotonic?

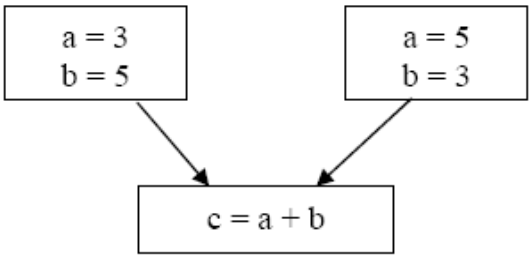
Yes / No (circle one, no explanation needed)

Yes.

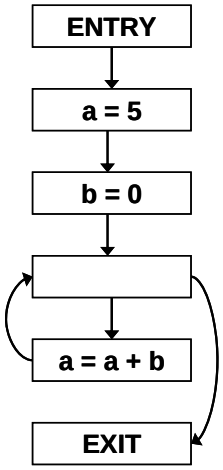
f) (3 pt) Is the transfer function you defined above distributive?

Yes / No (circle one, no explanation needed)

No.



g) (6 pt) What is the range for variable a [on EXIT] as computed by your algorithm for the CFG below?



Variable a belongs to range **[5,5]**