

**ECE540S: Optimizing Compilers**  
**Miterm Exam, March 1, 2005**

**Closed Book, Closed Notes, No Electronic Devices**  
**110 Minutes**

NAME Solution  
STUDENT NUMBER \_\_\_\_\_  
EMAIL ADDRESS \_\_\_\_\_

Q1:	/ 15
Q2:	/ 10
Q3:	/ 15
Q4:	/ 10
Q5:	/ 15
Q6:	/ 15
Q7:	/ 10
Q8:	/ 10
TOTAL:	/ 100

## CONTROL FLOW ANALYSIS (25 marks)

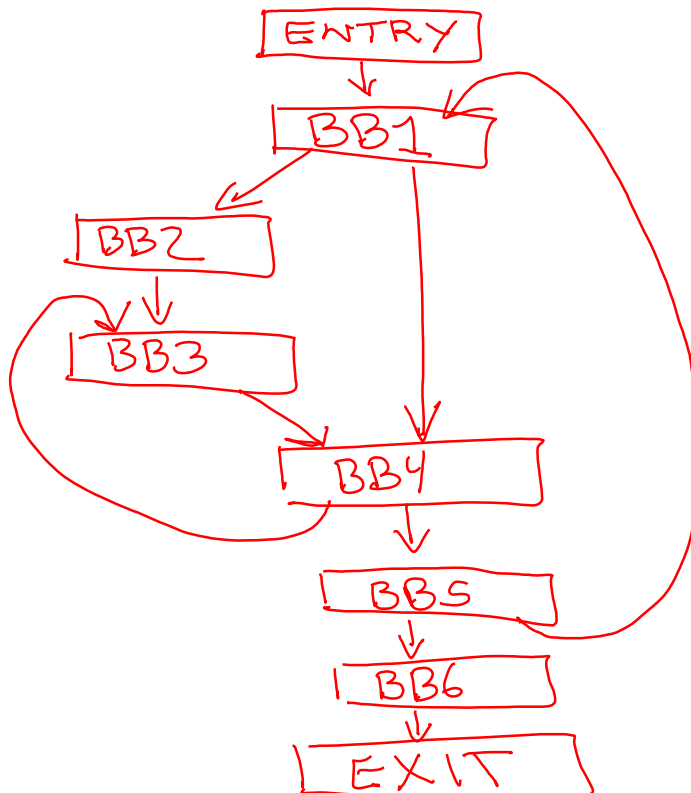
1) Use the following code to answer parts (a) and (b).

```

è   S1:  X = X + 1
    S2:  IF (Y < 10) GOTO S5
è   S3:  K = K + 1
è   S4:  X = X + 1
è   S5:  IF (X < 100) GOTO S4
è   S6:  Y = Y + 1
    S7:  X = 0
    S8:  IF (Y < 100) GOTO S1
è   S9:  PRINT X
    S10: PRINT Y
    S11: RETURN
  
```

} BB 1  
 } BB 2  
 } BB 3  
 } BB 4  
 } BB 5  
 } BB 6

a) Identify the leader instructions and their corresponding basic blocks. You may do this directly on the code snippet. Draw the control flow graph below (10 marks):



b) Identify the back-edge(s) in the control flow graph drawn in part (a). Provide them below using the form  $T \rightarrow H$ , where  $T$  is the basic block at the tail of the edge and  $H$  is the basic block at the head of the edge (5 marks).

**BB5  $\rightarrow$  BB1 is the only back-edge. For BB4  $\rightarrow$  BB3, BB3 does not dominate BB4.**

2) For each statement below, state whether the statement is true or false. (10 marks)

a) If  $w$  dominates  $x$  and  $y$  dominates  $z$ , then  $x$  dominates  $z$  if and only if  $w$  dominates  $y$ . (true/false)

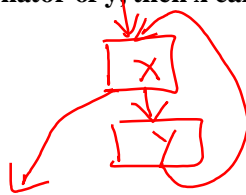
False.

*y could dominate w*



b) If  $x$  is the direct/immediate dominator of  $y$ , then  $x$  cannot be the direct/immediate post-dominator of  $y$ . (true/false)

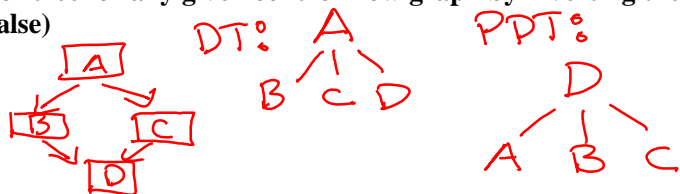
False.



c) You can create the post-dominator tree for any given control-flow graph by inverting the edges of its dominator tree. (true/false)

False.

*consider*



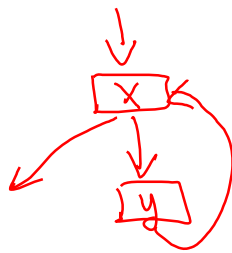
d) A node can directly/immediately dominate itself. (true/false)

False.

*see definition of idom  
a ≠ b*

e) For any back-edge  $(t, h)$ , the tail  $t$  post-dominates all nodes contained in the natural loop defined by  $(t, h)$ . (true/false)

False.



*Back-edge  $y \rightarrow x$   
x in loop,  
but y does  
not pdom x*

## DATA FLOW ANALYSIS (25 marks)

3) An important technique for some object-oriented languages is *escape analysis*. A variable *escapes* a function if it becomes accessible from outside of the function. This could happen for example, if a variable's address is assigned to a global pointer or if the variable is returned from the function by reference. Variables that do NOT escape from the function might be allocated on the stack and optimized more aggressively.

Consider a data flow analysis framework that determines at each program point  $p$ , the variables that may escape between point  $p$  and the exit of the routine (i.e. there is at least one path from  $p$  to exit on which the variable  $v$  escapes). To be conservative, a variable  $v$  escapes at a statement  $s$ , if (1) its address is taken, i.e.  $s$  is of the form " $x = \text{addr } v$ " or (2)  $s$  returns  $v$  by reference, i.e.  $\text{return addr } v$ . There is no way to un-escape, that is, once a variable has escaped the function, it has escaped and cannot be recaptured.

Answer parts (a) – (e) based on the above description.

a) What would the bit vectors represent in escape analysis (2 marks)?

**The bit vectors would represent all of the variables in the function. Each bit would signify whether the variable possibly escapes (1) or does not escape (0).**

b) Would you implement this solver as a forward or backward problem (2 marks)?

**Backward. We care about what occurs after  $p$ , between  $p$  and Exit.**

c) What would the meet operator be (2 marks)?

**We want to know if a variable escapes along ANY path between  $p$  and Exit. Therefore we would UNION the paths together. The meet operator would therefore be UNION.**

d) Define (in words) the  $\text{Gen}(b)$  set and  $\text{Kill}(b)$  set for a basic block  $b$  (4 marks).

**Gen(b):** The variables that may escape during the block  $b$ . A variable  $v$  is said to escape  $b$  if the block contains "... = addr  $v$ " or "return addr  $v$ ".

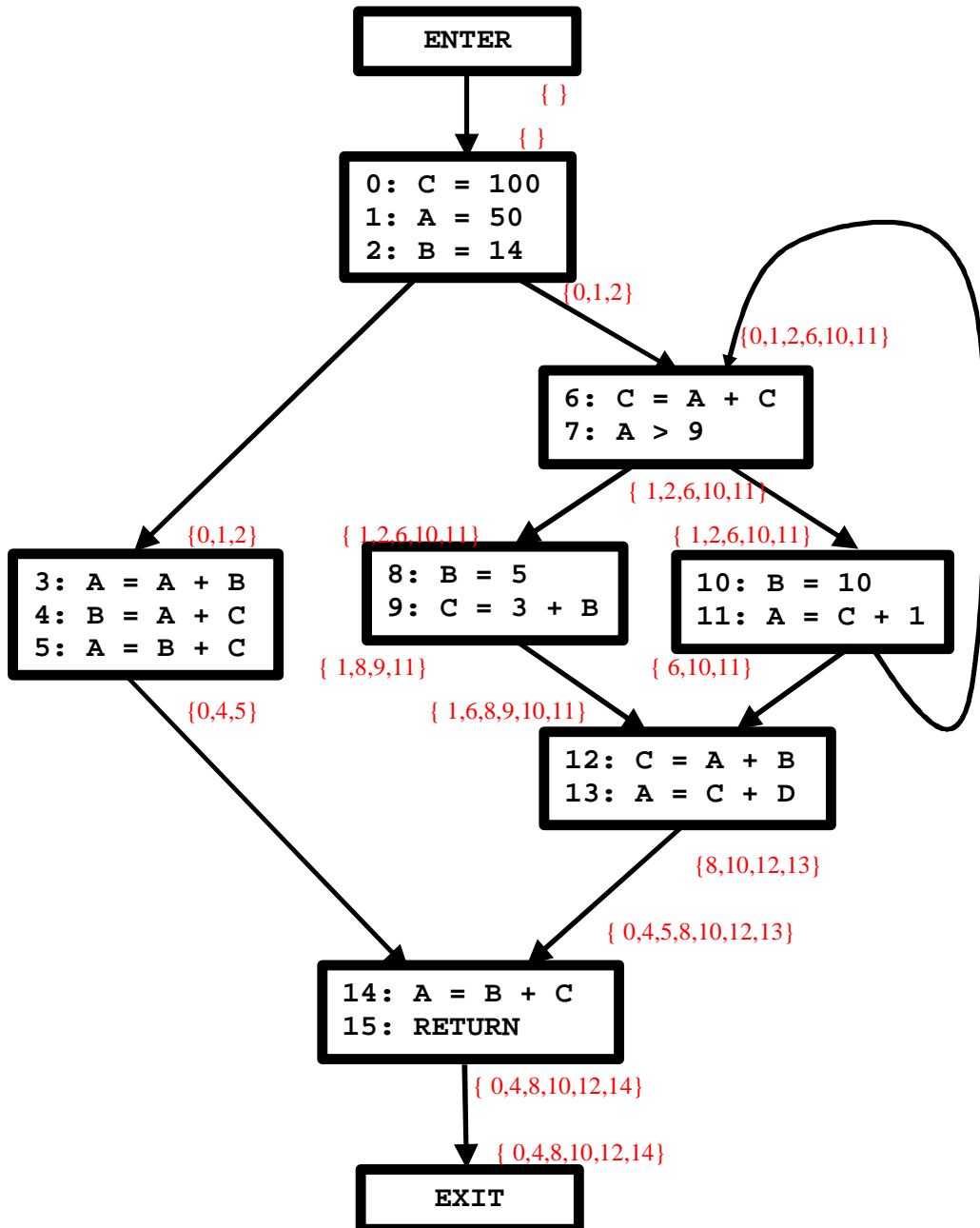
**Kill(b):** The variables that unescape during block  $b$ . There is no way to be recaptured by the function and therefore  $\text{Kill}(b) = \{ \}$  for all blocks  $b$ .

e) Provide the data flow equations for calculating the In and Out sets for a basic block  $b$  (5 marks):

$$\text{EscIn}(b) = \text{EscOut}(b) \cup \text{Gen}(b)$$

$$\text{EscOut}(b) = \bigcup_{\text{all } i \text{ in Succ}(b)} \text{EscIn}(i)$$

4) Perform Reaching Definitions analysis on the following CFG. Label each basic block with the final contents of its In and Out sets (10 marks).



**OPTIMIZATIONS (40 marks)**

5) Use the following code to answer parts (a) and (b). (15 marks)

```
L1:  W = addr K
      Y = 2*X   Ç  DIV
      Z = W + Y Ç  DIV
      [Z] = Z
      X = X + A Ç  BIV
      if (R == 0) GOTO L2
      V = 4*X   Ç  DIV
      Q = W + V Ç  DIV
L2:  R = [Q]
      [Q] = Z
      X = X + B Ç  BIV
      IF (X < M) GOTO L1
```

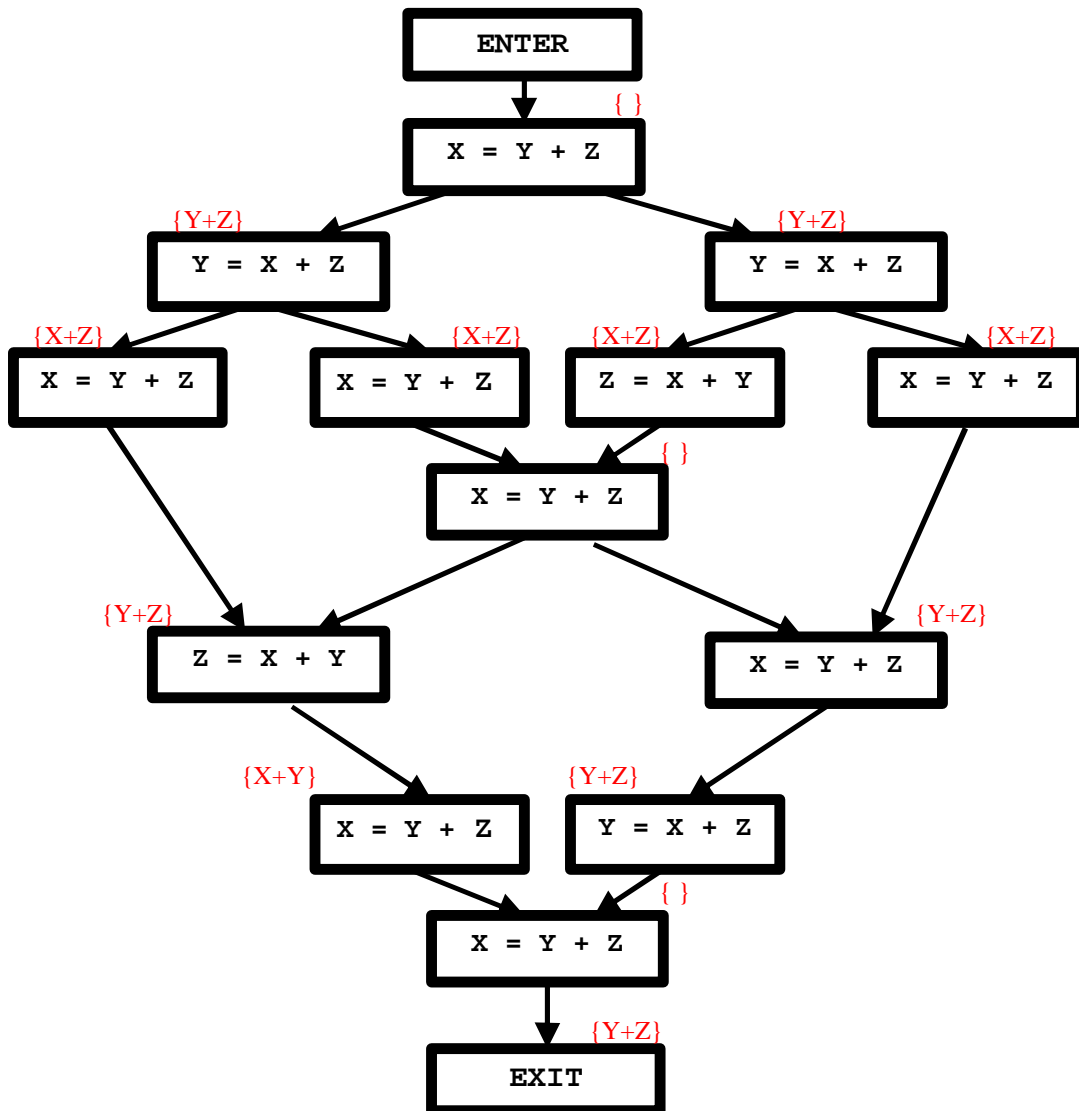
a) Identify the Basic Induction Variables (BIV) and Derived Induction Variables (DIV) in the code above. Place an arrow next to each statement that defines a BIV or DIV, and state whether it defines a BIV or DIV. (5 marks)

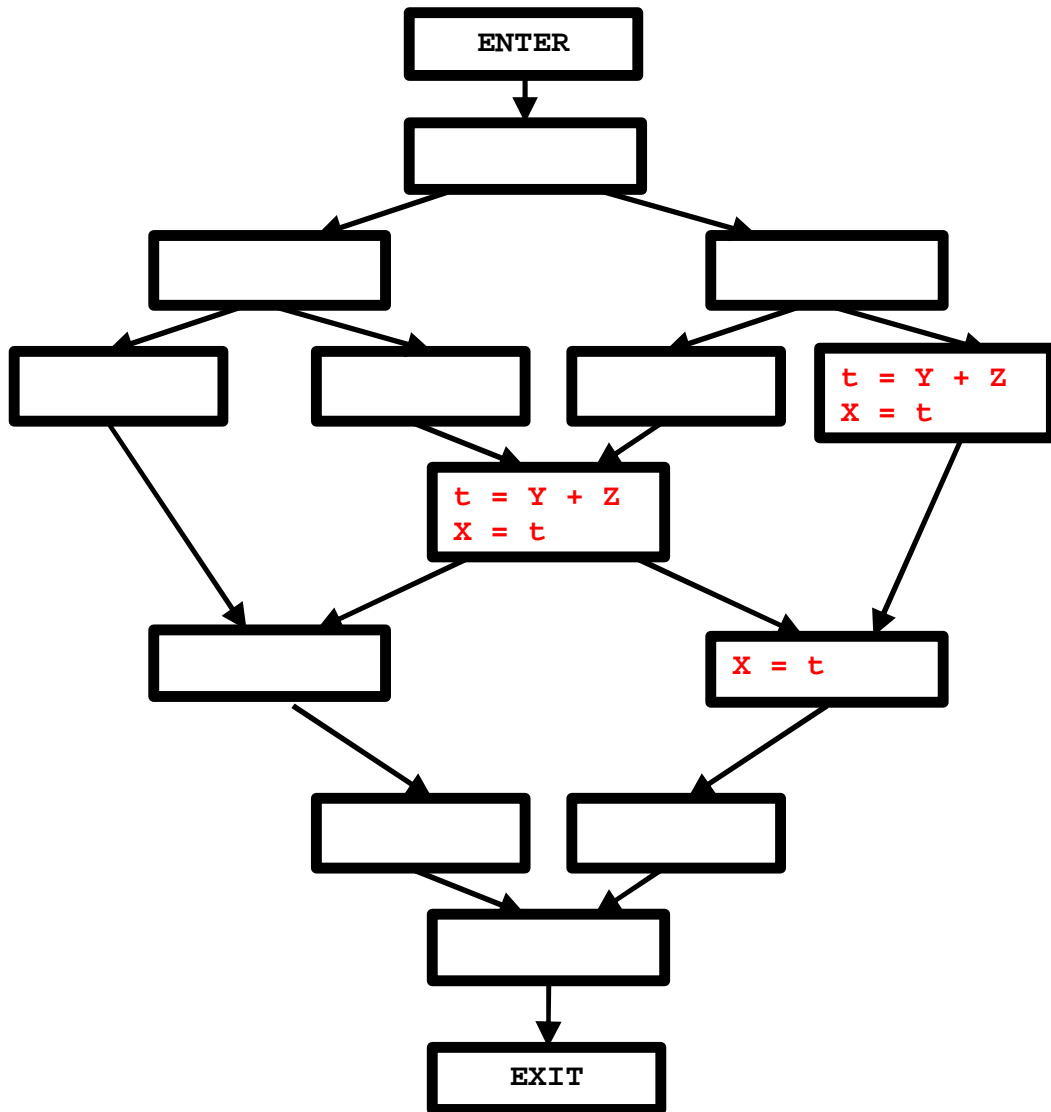
b) Perform strength reduction on the loop above (i.e. simplify the calculation of the induction variables). You should remove induction variables when possible, perform loop invariant code motion, propagate copies etc... You may assume that all induction variables are NOT live after the loop. Place the final code below: (10 marks)

```
W = addr K
Ty = 2 * X
Tz = Ty + W
C = A + B
C1 = 2 * C
C2 = 4 * C
T = X + A
Tv = 4 * T
Tq = Tv + W - C2
Tm = 4*M + W

// remove all need for V and Y, 2 marks
L1: [Tz] = Tz // handling Z correctly, 3 marks
    Tq = Tq + C2
    if (R == 0) goto L2
    Q = Tq
L2: R = [Q]
    [Q] = Tz // handling Q correctly, 3 marks
    Tz = Tz + C1
    if (Tq < Tm) goto L1 // Removing X, 2 marks
```

6) Perform common sub-expression elimination on the graph below. Show the Available Expressions In set for each block in the graph below. Show the simplified graph after global CSE on the next page. Perform only CSE. (15 marks)





7) Use the following code to answer parts (a) and (b). (10 marks).

```
I = J + K
I = J + K
J = I + K
I = J + K
K = I + J
J = I + K
I = J + K
```

a) Show the above code after removing redundant code by using local value numbering. Perform no other optimizations (5 marks):

```
I = J + K
I = I
J = I + K
I = J + K
K = I + J
J = I + K
I = J + K
```

b) Show the above code after removing redundant code by using local common sub-expression elimination (begin with the original code, not the result from part a). Perform no other optimizations (5 marks):

```
T = J + K
I = T
I = T
J = I + K
I = J + K
K = I + J
J = I + K
I = J + K
```

MISCELLANEOUS (10 marks)

8) Provide short answers to the following questions (5 marks each)

a) Your manager asks you which optimization will reduce execution time more, common sub-expression elimination or loop invariant code motion. What do you say?

It depends completely on the programs that are to be optimized. If there are many loops in a program, and these loops contain invariant code, then you will get a large benefit from loop invariant code motion. CSE removes redundant computations. If there are many computations that contain common sub-expressions, CSE might be better. CSE may remove redundant computations that are computed within loops, so even if there are many loops in a program, it does not imply that LICM will perform better than CSE. So to know which is more important, the programs that are important to the company should be studied to determine what opportunities for optimization exist in them.

b) Assume that you have a compiler that has a front-end that operates on a high-level intermediate representation (IR), an optimizer that operates on a medium-level IR and a back-end that operates on a low-level IR. For each level, described the characteristics of optimizations that would make sense to implement at that level.

Placing optimizations in the front-end moves them closer to the source language. So optimizations at this level would probably be there so that they can exploit structures, features or characteristics that are language dependent.

Placing optimizations in the back-end moves them closer to the target architecture. Optimizations at this level are there so that they can address target-specific features, characteristics and limitations.

Finally, as much as practically possible should be put into the MIR optimizer. This part of the compiler is not closely tied to either the source language or the target architecture. Therefore optimizations that do not require source or target information should be placed here so that they can be reused across languages and targets.