

## ENEE759C Spring 2003

**Quiz**  
Name: \_\_\_\_\_ *Answer key*  
Student ID: \_\_\_\_\_

**Pages: 6 printed sides**  
Time allotted: **1 hour 15 min**  
Maximum score: **55 points**

University rules dictate strict penalties for any form of academic dishonesty. Looking sideways will be penalized. Look at only your own exam at all times.

There are 6 questions, some with subparts. **Read them *carefully* to avoid throwing away points!!** Write your answer in the space provided. *Closed book, closed notes. Calculators are allowed.*

***Partial credit rule: Must show your intermediate steps clearly for partial credit!***

---

1. Fill in the blanks with one, two, or at most three words:

**(1 point \* 12 = 12 points)**

- (a) Parsing is a process that converts a source-level program into an internal tree-based compiler representation.
- (b) In a VLIW processor, instruction-level parallelism must be discovered and scheduled by the compiler.
- (c) A/An abstract syntax tree tree is a type of compiler intermediate representation in which the programs are stored as machine independent lists of high-level and low-level nodes.
- (d) Fortran was the world's first programming language.
- (e) In a control-flow graph, the nodes represent basic blocks.
- (f) A call graph is a data structure that relates how the procedures in a program call each other.
- (g) There is always only one immediate dominator for a node in the dominator tree of a program.
- (h) A node X dominates node Y if all paths from the start node to Y in the CFG contain X.
- (i) If node A dominates node B in the CFG, then an edge from B to A is called a back edge.
- (j) An example optimization for which induction variable analysis is useful is strength reduction.
- (k) Copy propagation is less useful if the register allocation phase of the compiler uses an efficient algorithm.
- (l) In conversion to SSA form, for each definition in the program,  $\phi$  functions are inserted at each node of its dominance frontier.

↖  
'variable definition' also okay

2. For each subpart (i) to (vii) below, circle **all** correct answers from among the four given - note that **more than one** answer may be correct!

(2 points \* 7 = 14 points)

(i) Some features often present in embedded processors that complicate compilation are:

- (a) Heterogeneous register files.
- (b) Presence of multi-level caches.
- (c) VLIW ISAs.
- (d) More speculative hardware than desktops.

(ii) A basic block is defined as a sequence of instructions

- (a) having no branches in it.
- (b) having only a single entry at the beginning and a single exit from the end.
- (c) having no labels within in.
- (d) that defines a scope for the definition of variables.

(iii) Dominator relationships

- (a) are defined on the call graph of the program.
- (b) are useful for identifying natural loops.
- (c) are useful in performing traditional dataflow analysis for reaching definitions.
- (d) are useful for converting a program to SSA form.

(iv) Conditions that *may* prevent a loop invariant definition  $d: t \leftarrow a \oplus b$  from being hoisted out of the loop include:

- (a)  $t$  is multiply defined in the loop.
- (b) the loop is a **repeat-until** loop.
- (c)  $t$  is live-out of the instruction immediately before the loop.
- (d) the register allocator is unable to find a register to store  $t$  in the loop.

(v) Traditional data flow analysis

- (a) formulates a set equations and solves them using fixed point iteration.
- (b) can be used to perform the loop-invariant code motion optimization.
- (c) for reaching definitions is useful for doing dead-code elimination.
- (d) when used for copy propagation may aid common-subexpression elimination.

(vi) Compared to traditional data flow analyses, SSA form

- (a) requires less space to store analysis results.
- (b) does not require different types of analysis to do different optimizations.
- (c) is capable of enabling optimizations not possible with traditional analyses.
- (d) is less affected by the presence of ambiguous memory definitions.

(vii) Pointer analysis

- (a) aims to derive the data objects pointed to by each memory access in the program.
- (b) is essential before data flow optimization can be performed.
- (c) improves the effectiveness of data flow optimization.
- (d) makes sense for global and stack data references only.

3. Consider a node N in a control flow graph of a program.

**(2 + 2 = 4 points)**

- (a) Is it possible for the dominance frontier of N to contain nodes that are the children of N in the dominator tree? If not possible, give a reason. (1-2 sentences).

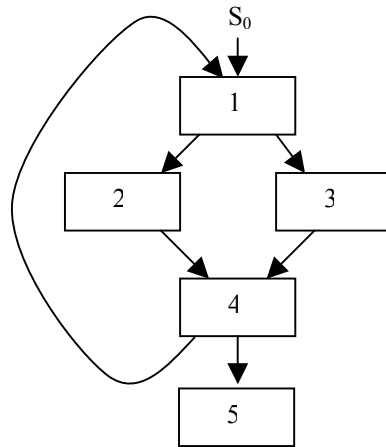
No, it is not possible. The children of N are all dominated by N, but none of the nodes in the dominance frontier of N are dominated by N (by the definition of DF).

- (b) Is it possible for the dominance frontier of N to contain nodes that are the ancestors of N in the dominator tree? If not possible, give a reason. (1-2 sentences).

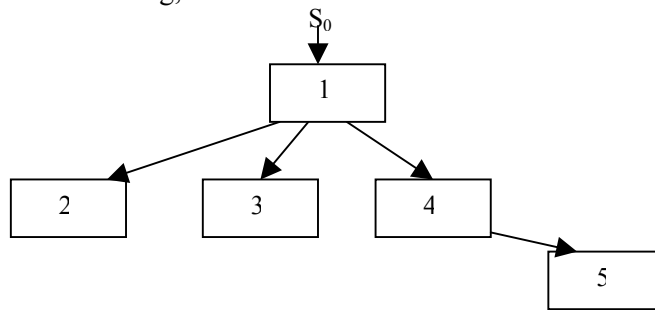
Yes, it is possible.

4. Consider the following control flow graph:

(4 + 2 + 4 = 10 points)



(a) Using intuitive reasoning, draw the dominator tree of the above CFG.



(b) Write the set equations used to solve for  $D[n]$ , the set of nodes that dominate node  $n$ . Describe the initial values for the fixed point iteration

$$D[S_0] = \{ S_0 \}$$

$$D[n] = \{n\} \cup \left( \bigcap_{p \in \text{pred}[n]} D[p] \right) \quad /* \text{ for all } n \neq S_0 */$$

Initially,  $D[S_0] = \{ S_0 \}$

$$D[n] = \text{set of all basic blocks in the program} \quad /* \text{ for all } n \neq S_0 */$$

(c) Solve the values of  $D[n]$  for all  $n$  for the CFG above. Derive  $D[n]$  in multiple passes of the nodes, until the values of  $D[n]$  do not change between passes. Within each pass, the nodes are visited in order 1,2,3,4,5. Represent your solution as a table, whose rows are the values of  $D[n]$  for different  $n$ , and whose columns are the different passes in the fixed point iteration.

Let  $\text{all} = \{ S_0, 1, 2, 3, 4, 5 \}$

	Initial	Pass 1	Pass 2
$D[1]$	all	1, $S_0$	1, $S_0$
$D[2]$	all	1, 2, $S_0$	1, 2, $S_0$
$D[3]$	all	1, 3, $S_0$	1, 3, $S_0$
$D[4]$	all	1, 4, $S_0$	1, 4, $S_0$
$D[5]$	all	1, 4, 5, $S_0$	1, 4, 5, $S_0$

Final values of  $D[n]$  are those in the last column.

5. In this question we will define a new traditional data flow analysis, called **initialized variables**, for generating at each point in the program the set of variables that have surely been initialized regardless of the control flow path followed to that point. We will use it to generate compiler warnings to the programmer on variables that are possibly used without being initialized, to aid in debugging.

(2 + 2 + 2 + 2 = 8 points)

(a) Write down the gen and kill sets for the **initialized variables** analysis in the table below for the four types of statements shown.

Statement s	gen[s]	kill[s]
d: $t \leftarrow b \oplus c$	{t}	{}
d: $t \leftarrow M[b]$	{t}	{}
M[a] $\leftarrow b$	{}	{}
Goto L	{}	{}

(b) Write down the formulation for in[n] and out[n] for each statement n in the program. (These sets contain the variables that are surely initialized regardless of the control flow path taken, just before and just after the statement n, respectively.)

$$\text{in}[n] = \bigcap_{p \in \text{pred}[n]} \text{out}[p]$$

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

$$= \text{gen}[n] \cup \text{in}[n] \quad /* \text{Since } \text{kill}[n] \text{ is always NULL } */$$

(c) What should be the initial values of in[n] and out[n] to correctly solve the equations in the previous part using a fixed point iteration?

$$\text{in}[n] = \text{NULL set (for all } n).$$

$$\text{out}[n] = \text{Set of all variables in the program (for all } n).$$

(d) We wish to use the above analysis to print messages in the compiler warning of possible uses of program variables without them being initialized. What is the condition that variable b used as a source operand of an instruction n might be used without initialization?

$$b \notin \text{in}[n]$$

6. Consider the following program fragment:

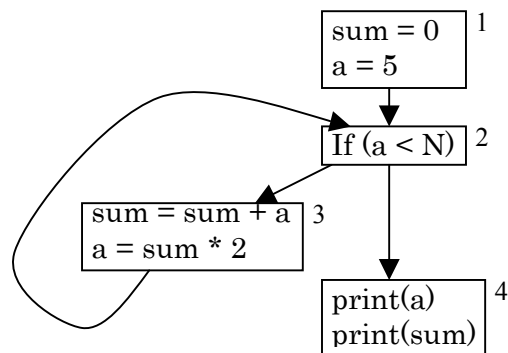
(3 + 4 + 2 = 7 points)

```

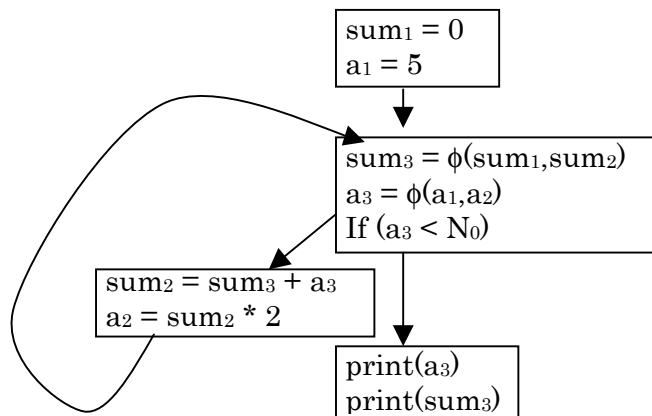
sum = 0
a = 5
While (a < N) {
    sum = sum + a
    a = sum * 2
}
print(a)
print(sum)

```

(a) Draw the control flow graph of program with instructions inside. Number the basic blocks 1,2,... in the order they appear in the program.



(b) Using intuitive reasoning to convert to SSA form, re-draw the control flow graph of the program showing the program in SSA form.



(c) When converting the program to SSA form using a formal method,  $\phi$  functions are inserted at the dominance frontier of each definition in the program. Using your intuition, which basic block numbers are the dominance frontier of the assignment to sum before the loop? Which basic block numbers are the dominance frontier of the assignment to sum inside the loop?

DF(sum assignment before loop) = Basic block 2  
 DF(sum assignment inside loop) = Basic block 2