CS243 Final Examination

Winter 2003-2004

You have 3 hours to work on this exam. The examination has 180 points. Please budget your time accordingly.

Write your answers in the space provided on the exam. If you use additional scratch paper, please turn that in as well.

Your Name: ___________________________

In accordance with both the letter and spirit of the Honor Code, I have neither given nor received assistance on this examination.

Signature: ___________________________

Problem	Points	Score
1	15	__________________
2	10	__________________
3	15	__________________
4	20	__________________
5	30	__________________
6	30	__________________
7	40	__________________
8	20	__________________
Total	180	__________________

1. (15 points) Loop level parallelism.

   a. (5 points) Is the following a DoAll loop? Explain your answer briefly.

```
        for (i = 0; i < n; i++) {
            a[i-2] = a[i-1] + 1;
        }
```

   b. (5 points) Is the following a DoAll loop? Explain your answer briefly.

```
        for (i = 0; i < n; i++) {
            a[2i-1] = a[2i] + 1;
        }
```

   c. (5 points) What is the complexity of data-dependence analysis?

2. (10 points) Suppose G'=<N, E'> is a reverse graph of G = <N, E>. That is, (n_1, n_2) $\in E$ iff (n_2, n_1) $\in E'$. Is it TRUE or FALSE that a pre-order traversal of G' is a reverse post-order of G. (A pre-order is a depth-first traversal where a parent is visited before its children). Give a brief explanation.

3. (15 points) Is there a scenario where a mark-and-compact garbage collector outperforms a generational garbage collector? If so, describe such a scenario; otherwise, explain briefly why not.

4. (20 points) Show the points-to result obtained with a flow-insensitive, context-insensitive inclusion-based points-to analysis. The allocation sites are named `a1, a2, ..., a6`.

```
main ()
{
    List list1 = new List();     (a1)
    List list2 = new List();     (a2)
    List list3 = new List();     (a3)
    C elem1 = new C();           (a4)
    D elem2 = new D();           (a5)
    list1.add (elem1);
    list3.add (elem2);
    list2.add (list3);
}
public class Node {
    public Node next;
    public Object elem;
}
public class List {
    private Node head;
    void add (Object item) {
        Node t = new Node();     (a6)
        t.next = head;
        t.elem = item;
        this.head = t;
}
```

Name the objects pointed to by:

a. `a1.head`

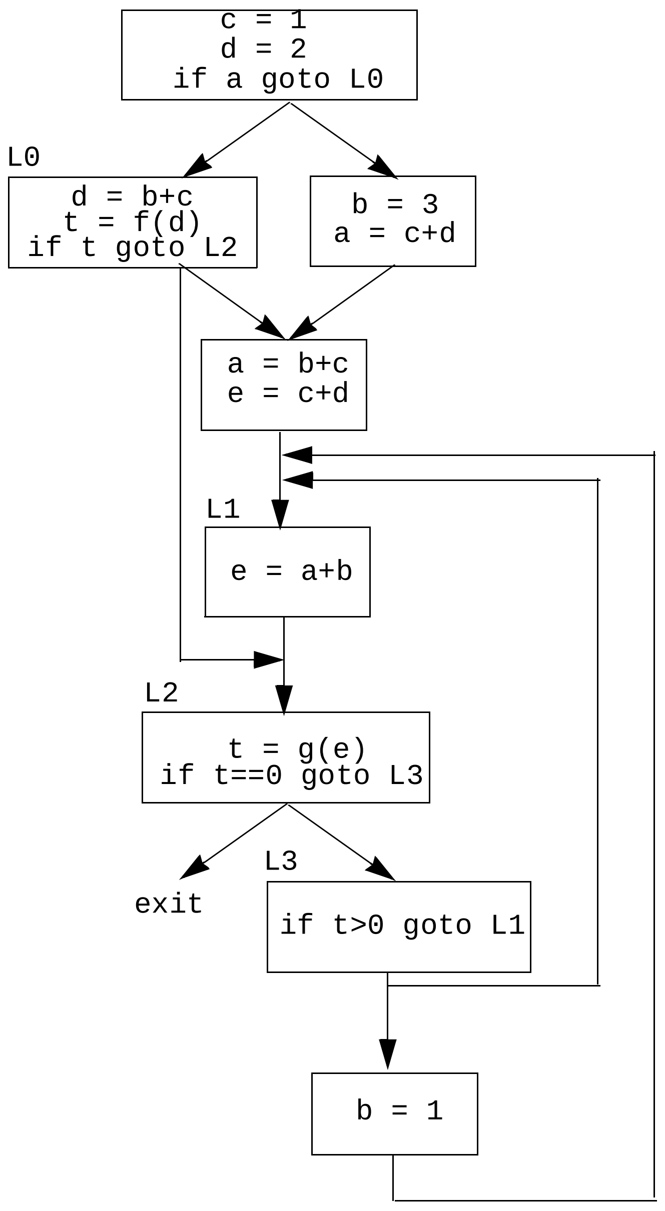
b. `a2.head`

c. `a3.head`

d. `a6.next`

e. `a6.elem`

5. (30 points) Optimize the following program using all the techniques discussed in class: You may assume that all variables are dead at the end of the program; however, you may not remove any function calls from the program.



6. (30 points) Consider an architecture that can issue three memory operations, three ALU operations, and a loop back operation in a single clock. There are two kinds of memory operations: LD (load) and ST (store). Memory operations have a post-increment mode, where "`0(r+=f)`" means that register `r` is first used as an address to access memory, then incremented by constant offset `f`. All operations have a single-cycle latency. Registers are read at the beginning of the clock cycle; registers are written at the end of the clock cycle. The loop back instruction "`LB r, L`" decrements register `r` by 1 and branch to `L` only if the result is greater than 0. Each operation only uses one unit of a resource of the appropriate type in the clock the operation is issued.

   Consider the following source program:

```
for (i = 1; i < 1000; i++) {
    A[i] = C[i-1] + t1;
    B[i] = A[i-1] + t2;
    C[i] = B[i-1] + t3;
}
```

   You can assume that `A, B, C` are disjoint arrays. (You are not allowed to perform any optimization on this code). The loop has been translated to the following list of instructions:

```
        // pr0 = 999
        // pr1 = &A
        // pr2 = &B
        // pr3 = &C
        // pr4 = &A+4
        // pr5 = &B+4
        // pr6 = &C+4
        // pr7 = t1
        // pr8 = t2
        // pr9 = t3
    L:
        (1) LD  pr10 = 0(pr3+=4)
        (2) ADD pr11 = pr10 + pr7
        (3) ST  0(pr4+=4) = pr11
        (4) LD  pr12 = 0(pr1+=4)
        (5) ADD pr13 = pr12 + pr8
        (6) ST  0(pr5+=4) = pr13
        (7) LD  pr14 = 0(pr2+=4)
        (8) ADD pr15 = pr14 + pr9
        (9) ST  0(pr6+=4) = pr15
        (10)LB  pr0, L
```

a. Draw the data-dependence graph and label the edges with their iteration differences and delays.

b. What is the minium initiation interval due to resource constraints?

c. What is the minimum initiation interval due to precedence constraints?

d. Apply software pipelining as described in class. What is the initiation interval of your software pipelined loop?

e. Show the modulo resource-reservation table for your software pipelined schedule.

f. Show the schedule of an iteration in the original program after software pipelining. You need not worry about the registers. Simply specify for each clock cycle, the ID of the operations being issued. Use the following format:

Clock	Instructions
0	(1), (2)
1	...

7. (40 points) Critical sections are a fundamental concept in concurrent programming. A communicating process must issue a LOCK operation before entering a critical section and issue an UNLOCK operation on exit in order to guarantee mutual exclusion.

   For simplicity, assume that there is only one lock in the system, and therefore the LOCK and UNLOCK operations do not have any arguments. You can ignore all the other operations in the program. In a correct execution sequence, every LOCK operation must be immediately followed by an UNLOCK operation, and every UNLOCK operation must be preceded by a LOCK operation.

   Your task is to develop a static program analysis that will issue the following four types of warnings:

   - Warning I: LOCK $\longrightarrow$ LOCK. Issue a warning on a LOCK operation if it can potentially follow another LOCK operation.
   - Warning II: UNLOCK $\longrightarrow$ UNLOCK. Issue a warning on an UNLOCK operation if it can potentially follow another UNLOCK operation.
   - Warning III: LOCK $\longrightarrow$ EXIT Issue a warning on a LOCK operation if the program may terminate without performing an UNLOCK.
   - Warning IV: ENTRY $\longrightarrow$ UNLOCK. Issue a warning on an UNLOCK operation if it may be executed before any LOCK operations.

   You may assume that there are no procedure calls in the input programs. Describe your analysis. You may want to define one or more data flow algorithms to solve this problem. If you solution uses data flow analysis, specify the algorithm fully by answering the following questions:

   i. What is the direction of your data flow analysis?
   ii. What is the set of values in the semi-lattice?
   iii. Draw a diagram of the lattice, identifying the top and bottom elements clearly.
   iv. How would you initialize the iterative algorithm?
   v. Define the transfer function of a basic block.
   vi. How would you initialize the information at the entry/exit nodes?
   vii. Is your data flow framework monotone? (No explanation is necessary).
   viii. Is your data flow framework distributive? (No explanation is necessary).
   ix. Will your algorithm necessarily converge? If so, why?

   Make sure you specify how you identify the operations that have raised the warning.

*** Extra Space ***

8. (20 points) As discussed in class, conventional reference counting cannot identify garbage that has circular references. Can you suggest an extension to reference counting that can detect circular garbage without tracing all the objects from the root set? (Hint: Suppose a pointer p is pointing to o and that p is about to be overwritten. Is it possible to tell if p points transitively only to objects reachable only through p?)