CSCI 565 - Compiler Design

Spring 2011

Second Test

April 27, 2011 at 3.30 PM in Room RTH 115

Duration: 2h 30 min.

Please label all pages you turn in with your name and student number.

Name: _____ **Number:** _____

Grade:

Problem 1 [25 points]:

Problem 2 [35 points]:

Problem 3 [40 points]:

Total:

Instructions:

1. This is a closed book Exam.
2. The test booklet contains four (4) pages including this cover page.
3. Clearly label all pages you turn in with your name and student ID number.
4. Append, by stapling or attaching your answer pages.
5. Use a black or blue pen (not a pencil).

Problem 1. Code Generation and Run-Time Environment [25 points]

In this problem we will explore a code generation scheme for concurrency generation and termination spawn and wait constructs respectively. The spawn construct conceptually creates a new thread of execution that begins after the invocation of the spawn completes. The arguments include the address of a function (returning void) and a selected set of arguments. The spawn construct returns a handle to the executing threads, which can be passed around to a wait construct. The wait construct will block until the corresponding thread completes execution.

```
/* this is the spawning parent context */l
…
tid = spawn(myFunc, v1, v2)
…
wait (tid)
…

void myFunc(int p1, int p2){
   …
  return;
}
```

In terms of code generation this means that the enclosed thread, i.e., the one within which the code executes the spawn function, needs to create an execution environment. Similarly, the thread executing the wait primitive (the parent thread) needs to dispose of the created execution context (the child thread). Termination of the spawn is done via the return instruction. Notice also that while the spawned thread executes the spawning threads (or parent thread) will continue its execution possibly spawning other threads and/or execution sequentially and invoking other functions sequentially before attempting to synchronize with the spawned thread(s).

Discuss in broad terms how to create and maintain the execution environment for the spawned threads and establish the synchronization between the spawning thread and the spawned threads. Be specific about when and where to allocate the AR for the spawned threads, how to pass the corresponding arguments and how to synchronize on termination of the spawned thread.

Problem 2. Live Variables Analysis and Register Allocation [35 points]

In this problem we will explore the use of a global graph-coloring based register allocation and assignment and the simpler top-down register allocation algorithm based on the frequency of occurrence of each variable used in the code. For this problem consider the following 3-address instruction sequence below where you need not be concerned about the code that precedes the line 8 of the code nor the code afterwards. Assume the first basic block where line 8 is located is basic block labeled BB0 and the last basic block where line 23 is located is basic block BB3.

```
8:          t1 = 4
9:          t6 = 0
10:  L1:    t1 = t1 – 1
11:         t2 = t1 * 4
12:         t3 = GP + t2
13:         t4 = 0
14:         *t3 = t4
15:         if t1 = 0 goto L2
16:         t4 = 0
17:         t5 = ARP + 8
18:         *t5 = t4
19:         goto L1
20:  L2:    t6 = t6 + 1
21:         t7 = ARP + t1
22:         *t7 = t6
23:         goto L3
```

a. [10 points] **Recognize** the basic blocks of this code. Under the assumption that t6 is live at the end of basic block containing line 23 **compute the live ranges** of the variables in this code segment. You need not to show the iterative data-flow analysis algorithm steps.

b. [10 points] Derive the interference graphs (or table) for these variables using both conflict definitions described in class.

c. [10 points] Can you color the resulting interference graphs with 3 colors? Why or why not? If not suggest a way to split one of the webs so that the resulting interference graph is 3-colorable.

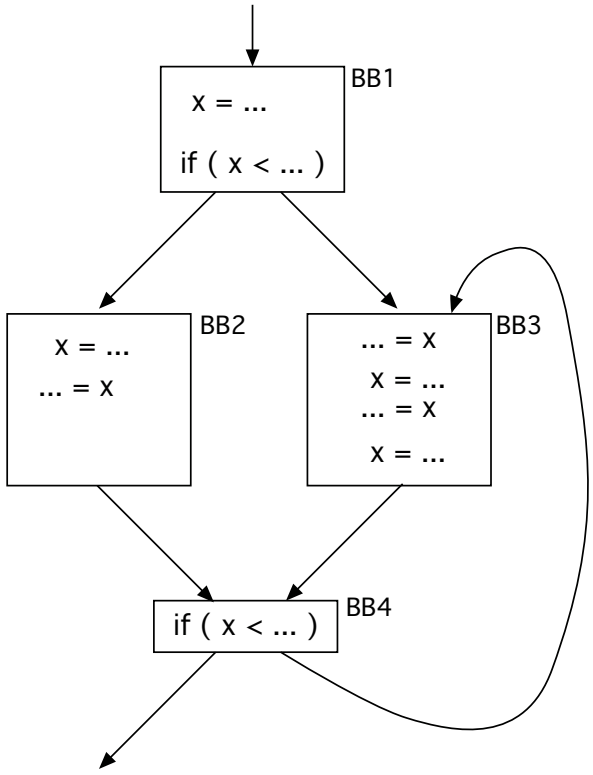
d. [5 points] Would the top-down algorithm described in class work well for this example? Justify.

Problem 3: Analysis and Code Representation [40 points]

There are several compiler passes that rely on the information about which variables are defined and used. Register allocation is such a case whose information can be derived directly from live variable analysis or by *def-use* (defined-used) chains. In this problem you are asked to describe the *def-use* (DU) data-flow analysis in detail and discuss its use to derive information for program enhancing transformations. Specifically address the following questions.

a. [20 points] Formulate the *def-use* iterative data-flow analysis problem indicating the structure of the lattice, the meet operator; the transfer functions and the initialization values for the nodes in the program assumed to be the nodes in the CFG corresponding to the program's basic blocks. Justify the choice of initial value in terms of precision and safety of the initial and the resulting final solution it leads to.

b. [10 points] Using the formulation you have developed in a. above apply it to the CFG structure depicted below where you can assume that the initial values for the data-flow abstractions are empty and that they do not change over the various iterations of your algorithm. Moreover, assume there are no other definitions to the `x` variable other than the ones depicted here. Show the intermediate values of the IN and OUT abstractions for each basic block.



c. [10 points] Use the solution you have found in b. above to convert the instructions in the basic blocks for the snippet of code in b. into SSA form. While a very efficient algorithm for the placement of the phi-function is exceedingly sophisticated, suggest a simple algorithm that uses the information in the DU-chains to place (and in many program instance correctly) the phi-function and thus transform a 3-address representation to SSA form.