Final Exam
CS3300

Maximum marks = 50, Time: 2.5hrs

26-Nov-2013

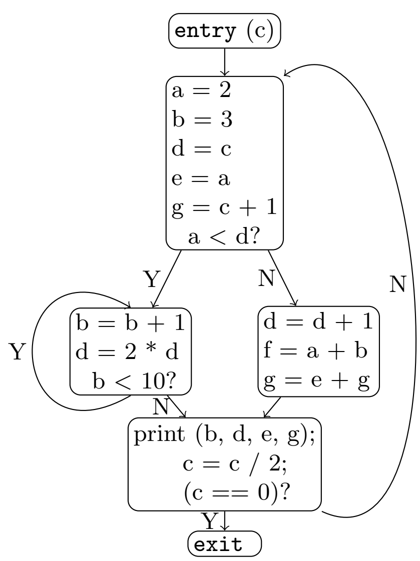
Read all the instructions and questions carefully. You can make any reasonably assumptions that you think are necessary; but state them clearly. There are total five questions totaling 50 marks. Each 10 marks will approximately take 30 minutes. For questions with sub-parts, the division for the sub-parts are given in square brackets.

Leave the first page empty. Start each question on a new page. Think about the question before you start writing and write briefly. **The answer for any question (including all the sub-parts) should NOT cross more than two pages.** If the answer is spanning more than two pages, we will ignore the spill-over text. If you scratch/cross some part of the answer, you can use space from the next page.

1. [10] **Control flow**: For the following code, draw the control flow graph and mark the basic blocks. For the first statement in each of the basic blocks, compute the dominator and post-dominator information.

```
x = 1; y = 2;
do  {
  x = x + 1;
  if (cond) y = y + 1;
  while (cond1) {
   x = x + 1;
   cond1 = foo(x);
  }
  x = x - y;
} while (x < y);
print (x, y);
```

2. [10] **Register Allocation**: Compute the liveness information, draw the interference graph, and do register allocation using Kempe's heuristic, assuming four registers.



3. [10] **Code Generation**: Write the tree patterns for the following instructions with their usual meanings [2]:

instruction	form
add	$r_i = r_i + r_j$
mul	$r_i = r_i * r_j$
addi	$r_i = r_i + c$
load	$r_i = M[r_j + c]$
store	$M[r_j + c] = r_i$
MemMove	$M[r_i] = M[r_j]$

Draw the intermediate-code tree for the assignment statement `a[i+1] = x * y` [4]. Assume that, register allocation has been done and all of the above variables are located on stack. Generate machine code using the maximal munch method [2]. Argue if your generated code is optimal or optimum [2].

4. [10] **Optimizations**: Optimize the following code in a step by step manner, using machine independent optimizations. At each step, indicate the optimization applied and the resulting code.

```
void foo(int z){
  q = 2;
  c = q;
  goto L1;
  c = c + z;
  b = z + 1;
  L1: b = c + 3;
  for (i=2 * m; i > 0; i = i / 2) {
    for (j=4 * m; j > 0; j = j / 4) {
      y = T[i] * b;
      S[i, j] = S[i, j] + V[i, j] * y + c;
      if (c > z) goto L2;
      V[i, j] = q + T[i] * c;
      L2: U[i, j] = T[i] - V[i, j] * y;
      V[i, j] = y - T[i] / c;
    }
  }
}
```

5. [10] **Garbage collection**

Give a scheme to do garbage collection via reference counting [8]. Consider the subset of MiniJava language. Our goal is to translate MiniJava to C, such that we can use `malloc` and `free` at runtime to allocate and free memory, respectively. Focus on the translation of the following statements: `new`, assignment (of the form, `x = e, x.f = e, y = x.f`) and discuss how the reference counts will be updated and when we can free some allocated memory. If you are using any data structures, explain the same clearly. Briefly give a scheme to handle cycles in your object graph. [2] A sample Java code and sketch for the corresponding C code can be seen below.

```
{
   A a = new A();
   a.f = 2;
   ... // uses a.f

   a = x;
}
```

```
{
   A *a = // code to do malloc
   a->f = 2;
   ... // uses a.f
   // code to invoke free (a);
   a = x;
}
```