Final Exam (CS6013)

Maximum marks = 60, Time: 3hrs

24-Apr-2017

Read all the instructions and questions carefully. You can make any reasonable assumptions that you think are necessary; but state them clearly. It is your responsibility to write legibly. There are total five questions totaling 60 (+ 5 bonus) marks. Each fifteen marks question will approximately require 40-45 minutes to answer. For questions with sub-parts, the division for the sub-parts is mentioned 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 get compensation space.

1. [10+3+2] **Semantic Analysis**. (a) Write the grammar and present an SDD (or SDT) to identify uninitialised variables in Java [10]. Handle only the following statements: variable declaration (of type int), assignment (to constants or variables+constant, for example, `x=3`, or `x=y+4`), statement-block (consisting of one or more statements inside curly braces, separated by a semicolon; for example {S1; S2}), if-then, and a while-statement. Assume: the expressions in the loop/if-then can only be simple variables (and the predicate is true, if the value of the variable is `!=0`).

(b) If I disallow pointer dereference (such as `x = *y` and `*x = y`) instructions in the three address codes, what alternatives do I have? [1]. How does a C compiler translate "sizeof" operator and & (addressof) operator? [2]

(c) State true or false [1+1].

1. Number of operands in IR instructions depend on the number of HW registers.
2. There is no need to maintain the line number information of the input program in the IR.

2. [10+3+2] **Mixed bag**. (a) For the following optimisations show the typical transformation, along with the profitability and correctness conditions: loop reversal, loop tiling, loop unswitching, loop fission and vectorisation [5×2].

(b) We compute the IDFs to identify locations of ϕ nodes. Instead, is it not sufficient to insert ϕ-nodes using either i) dominance frontiers of the definitions, or ii) intersection of the forward paths from each definition to the "exit" node [1.5 + 1.5]. Prove it or use a counter example to disprove it.

(c) State true or false [1+1].

1. JIT compilers can perform optimisations that a static compiler cannot.
2. Static compilers can afford to do optimisations that a JIT compiler cannot.

3. [10+3+2] **Data flow analysis**. (a) Copy propagation is the process of replacing the occurrences of direct targets with their values. For example, in the following code

```
a = b
c = a + 3
```

copy propagation will lead to elimination of the first assignment statement (provided there is no other uses of 'a', except the ones identified) and replacement of the second statement with `c=b+3`. Further, this is performed on scalars only.

Give an algorithm to do copy-propagation within a basic block [5]. Be careful on when you can eliminate the copy instruction. The input to your algorithm will be a basic-block (you can assume a way to iterate over the set of instructions). Return or print the optimized basic-block. Assume that the code is in 3-address code.

Give an iterative data flow algorithm to do intra-procedural copy-propagation [5]. The input to your algorithm would be a function and you have to return (or print) the optimised (copy-propagated) function.

(b) Is there any difference between register coalescing and copy propagation [1.5]? Is it enough to perform coalescing during register allocation and completely skip copy propagation? If Yes, prove it; if no, give a counter example [1.5].

(c) State true or false [1+1].

1. Given an interference graph, optimal register allocation is undecidable.
2. A phase ordering relation exists between variable packing and register allocation.

4. [10+3+2] **Register allocation**.

(a) Draw the flow chart of iterated register coalescing (register allocation) approach [3]. State the conservation coalescing criteria [1], and the simplify step [1]. For the code shown, do register allocation using iterated register coalescing [4]. Show/state each step clearly. Also mention the spilling criteria used (if there is any spilling). Show the final code after register allocation [1]. Assume: (i) the first two arguments are passed in registers R1 and R2. And the return value is returned in R1. R1, R2 are caller save registers and R3 and R4 are callee save registers. (ii) the memory location for any variable v (if it needs one) is given by M_v. (iii) Set of available registers = {R1, R2, R3, R4}.

```
entry:
c = r3 // saving callee save regs
d = r4 // saving callee save regs
p = r1 // read arg1
q = r2 // read arg2
a = p
b = q;
L0: if (a == b) goto L2;
  if (a < b) goto L1
   a = a - b
  goto L
  L1: b =  b - a
  L: goto L0
L2:
e = p*q / b;
r3 = c  // restoring callee save regs
r4 = d  // restoring callee save regs
r1 = e; // storing the return value
return; // r1, r3, r4 are live
```

(b) The greedy optimal variable packing (OVP) algorithm of Nandivada and Barik (studied in the class) requires that the variables' actual width must be a power of two. In practice it is not always the case. Instead of padding additional bits, what if we use the variables actual widths as is. Will the packing obtained be any worse than what is obtained by OVP? [3] If yes, give an example. If No, argue why so.

(c) State true or false [1+1].

1. Coalescing always helps generate more efficient code.
2. Under some conditions, two interfering variables can share a register.

5. [5] **Bonus**. *Suggestion: Attempt at the end.* SSA form creates many copies of the same variable. Does it mean that it may lead to more number of variables being spilled? Take a simple register allocation algorithm (say Linear Scan or Kempe's heuristic) and argue. Either prove that it does not or give an example stating otherwise.