CMSC430 Spring 2009 Midterm 2

Instructions

- You have until 4:45pm to complete the midterm.
- Feel free to ask questions on the midterm.
- One sentence answers are sufficient for the "essay" questions.
- Use only the following 3-address code and Java stack code instructions for answering code generation questions.

### 3-address Instruction

<table>
<thead>
<tr>
<th>3-addr Instruction</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>load R1 x</td>
<td>R1 ← x</td>
</tr>
<tr>
<td>store x R1</td>
<td>x ← R1</td>
</tr>
<tr>
<td>add R1 R2 R3</td>
<td>R1 ← R2 + R3</td>
</tr>
<tr>
<td>sub R1 R2 R3</td>
<td>R1 ← R2 - R3</td>
</tr>
<tr>
<td>mult R1 R2 R3</td>
<td>R1 ← R2 * R3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Java Stack Code</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>nop</td>
<td>none</td>
</tr>
<tr>
<td>ldc_int c</td>
<td>push constant c onto stack</td>
</tr>
<tr>
<td>iload index(x)</td>
<td>push local variable X onto stack</td>
</tr>
<tr>
<td>istore index(x)</td>
<td>pop stack, store in local variable X</td>
</tr>
<tr>
<td>iadd</td>
<td>pop 2 elems off stack, add, push</td>
</tr>
<tr>
<td>isub</td>
<td>pop 2 elems, subtract top from 2nd, push</td>
</tr>
<tr>
<td>imult</td>
<td>pop 2 elems off stack, multiply, push</td>
</tr>
<tr>
<td>ineq</td>
<td>pop stack, negate, push</td>
</tr>
<tr>
<td>goto L</td>
<td>jump to handle L</td>
</tr>
<tr>
<td>ifeq L</td>
<td>pop stack, jump to handle L if zero</td>
</tr>
<tr>
<td>ifIcmpneq L</td>
<td>pop 2 elems, jump to L if equal</td>
</tr>
<tr>
<td>ifIcmpgt L</td>
<td>pop 2 elems, jump to L if 1st greater</td>
</tr>
<tr>
<td>dup</td>
<td>duplicate top of stack</td>
</tr>
<tr>
<td>pop</td>
<td>pop top of stack</td>
</tr>
<tr>
<td>swap</td>
<td>swap top two positions of stack</td>
</tr>
</tbody>
</table>

1. (15 pts) Intermediate representations.
   Consider the arithmetic expression:
   
   \[
   a - (b - c)
   \]
   
   a. Translate it into an AST
   b. Translate it into 3-address code
   c. Translate it into Java stack code
   d. Name an advantage of using 3-address code instead of stack code.
2. (16 pts) Code generation.
You are generating code for a Java stack machine.
You are given the following grammar attributes and helper functions:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Holds</th>
</tr>
</thead>
<tbody>
<tr>
<td>AstNode.code</td>
<td>list of instructions</td>
</tr>
<tr>
<td>Function</td>
<td>Effect</td>
</tr>
<tr>
<td>genInst(X)</td>
<td>create new instruction X</td>
</tr>
<tr>
<td>append(...)</td>
<td>concatenates list of instructions</td>
</tr>
</tbody>
</table>

a. A Ruby-style UPTO loop iterates \((\text{exp}_2-\text{exp}_1+1)\) number of times. For example, \(5 \text{ upto } (2+6) \text{ do } ... \text{ end}\) would iterate 4 times. Write syntax-directed actions needed to generate code for a UPTO loop for the following grammar production. Note you can only use the Java instructions provided at the beginning of the exam.

\[
\text{stmt} \rightarrow \text{exp}_1 \text{ UPTO exp}_2 \text{ DO } \{ \text{stmtList} \} \text{ END}
\]

\[
\{ : \text{stmt.code} = \text{??} \}
\]

b. The logical operator NAND is false only if BOTH of its operands are true. Write syntax-directed actions needed to generate code for a NAND expression in the following production. Your code should leave a 1 or 0 on the stack, depending on whether the resulting expression is true or false. Make sure you apply short circuiting.

\[
\text{exp} \rightarrow \text{exp}_1 \text{ NAND exp}_2
\]

\[
\{ : \text{exp.code} = \text{??} \}
\]

3. (8 pts) Compiling and optimizing high-level languages
Object-oriented programming languages such as C++ and Java use classes and inheritance to improve programmer productivity. Remember that inheritance allows objects of one class to be used in place of objects in a parent class.

a. How are objects and classes implemented in the run-time environment?
b. What compiler-generated code is need to support inheritance?
c. What is prefixing?

4. (6 pts) Compiler optimizations

a. Give an example of a machine-independent optimization, and explain why it is machine-independent
b. Give an example of a machine-dependent optimization, and explain why it is machine-dependent
c. Why would a compiler writer decide not to implement a particular optimization?
5. (6 pts) Directed acyclic graphs
   Consider the following code.
   
   (1) a := b + c
   (2) b := b + c
   (3) c := b + c

   a. Perform renaming for the code
   b. Build a DAG for the renamed code
   c. How do DAGs support optimizations?

6. (6 pts) Control flow analysis
   For the following problems, consider this code:
   
   <S1>              a := b
   <S2>     L1:    b := c
   <S3>              if (...) goto L2
   <S4>              c := d
   <S5>              if (...) goto L1
   <S6>     L2:    d := a

   a. What are the basic blocks?
   b. What is the control flow graph
   c. What is a reverse Postorder numbering of the basic blocks?
   d. Why do compilers use basic blocks?

7. (6 pts) Local information
   Consider the following basic block for live variables:

   \[
   \begin{array}{c}
   b = a \\
   d = b \\
   a = c \\
   e = d \\
   a = e \\
   \end{array}
   \]

   a. What is GEN for the basic block?
   b. What is KILL for the basic block?
8. (6 pts) Data-flow analysis
Consider the following control flow graph:

Assume you are given IN/OUT for B1, B2, B4, B5, and GEN/KILL for B3. Show the data-flow equations for IN/OUT for B3 (e.g., IN(B3) = OUT(B1)).

   a. For forward data-flow problems
   b. For backwards data-flow problems?

9. (20 pts) Available expressions
Consider the following control flow graph for available expressions:

   a. Calculate GEN/KILL for each basic block
   b. Solve available expressions, showing IN/OUT for each pass

10. (12 pts) Data-flow analysis frameworks
Recall that $\wedge$ is used in data-flow iterative analysis to combine information where paths merge.

   a. Using properties of the $\wedge$ operator, prove $a \leq b$ and $b \leq a$ imply $a = b$.
   b. For reaching definitions, pick values for $a$, $b$, $c$ for which $a > b$ and $b > c$.
   c. For very busy expressions, pick values for $a$, $b$ for which neither $a \leq b$ or $b \leq a$ are true?
   d. How is $\perp$ defined in terms of the $\wedge$ operator?
   e. What is monotonicity and why is it important for iterative data-flow problems?