TENTAMEN
FDA001 Advanced Compiler Construction
4 juni 2004, kl 9-13

Name: ______________________  Personnummer: ______________________

Please mark solved problems with X:

<table>
<thead>
<tr>
<th>Question</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>answered</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>score</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Total score: _____  Grade (U/3/4/5): _____

Sign. Examinator: ______________________

Lokal:
John von Neumann at IDA
Last time to start the exam and earliest time to leave the room is 09:30.

Examinator, presence and jour:
Christoph Kessler (070-3666687), presence (Linköping) 09:00–13:00
Welf Löwe, guest examiner (073-0322959)

Visning:
on request

Admitted material:
dictionary from English to your native language

General instructions
on the back cover (page 2)
General instructions

- Enter your name and personnummer on the cover page and use this as cover for your exam when you hand it in.
- Read all assignments carefully and completely before you begin.
- Use a separate sheet for each assignment.
- Use only one side of each sheet of paper.
- Mark each sheet on top with your name, personnummer, and the course code (FDA001). Number all your pages consecutively.
  This is important because different assignments may be corrected by different examinators and sent via mail or fax.
- Mark each question that you answered in the table on the cover page.
- Do not use a red pencil.
- You may answer in either English or Swedish.
- Write clearly. Unreadable text will be ignored.
- Be precise in your statements. Unprecise formulations may lead to a reduction of points.
- Motivate clearly all statements and reasoning.
- Explain calculations and solution procedures.
- The assignments are not ordered according to difficulty.
- You do not need to answer all the questions. About 30 of the maximum possible 80 points will be sufficient to pass.

Preliminary grading scheme:

passed (3) at 30p, well passed (4) at 40p, excellent (5) at 50p.
1. **Intermediate representations and instruction selection (10 p)**
   (a) Modern optimizing compilers use more than one intermediate representation of the source program. Give a general motivation for this. (1p)
   (b) Muchnick proposes three levels of intermediate representations. Characterize them. (6p)
   (c) For each of these three, give an example of a compiler analysis / optimization where that representation is most appropriate, and explain why. (3p)

2. **Instruction selection (6p)**
   (a) Describe the principle of instruction selection by *tree pattern matching*. (4p)
   (b) For what type of processor architectures and what type of intermediate representation will tree-pattern matching lead to a considerable improvement in code size and/or execution time, compared to naive code generation from intermediate code to target code? Justify your answer. (2p)

3. **Garbage collection (6p)**
   Describe the principles of *Mark-and-Sweep* garbage collection and *Reference counting* garbage collection. What are the advantages/disadvantages of each method?

4. **Loop transformations (10 p)**
   Given the following loop nest:
   
   ```
   for i from 1 to N do
     for j from 1 to N do
   ```
   (a) Draw the data dependence graph and, for each dependence arc, identify dependence type (flow/anti/output), distance, and direction, and, where applicable, the carrying loop(s). (4p)
   
   For (b), (c), (d), and (e) give an argument by the dependence structure, not by example:
   (b) Is it legal to apply loop distribution to the *j* loop? If yes, why? If not, why not? (1p)
   (c) Is it legal to apply loop distribution to the *i* loop? If yes, why? If not, why not? (1p)
   (d) Is it legal to apply loop interchange to the loop nest? If yes, why? If not, why not? (1p)
   (e) Suggest a semantics-preserving transformation of this loop nest that makes the *j* loop *parallelizable*. *Hint: introduce a temporary array and initialize it appropriately.* Show the resulting program (where *forall* denotes a parallel loop). (3p)
5. **Instruction scheduling and software pipelining (18p)**

Assume you have a VLIW processor with three functional units, a memory access unit for loads (latency 4cc) and stores (latency 1cc), a multiplier for multiplications (latency 2cc), and an ALU for all other operations (latency 1cc). All units have occupation time 1 cc. Hence, load has 3 delay slots, multiplication and branch has 1 delay slot, and the other operations have no delay slot. All multiplications must be done on the multiplier. You can issue one long instruction word per clock cycle, each containing (at most) one subinstruction per functional unit.

Given the following target level program representation (i.e., after instruction selection) of the main loop of an integer dot product program, shown here as naively scheduled pseudocode for the architecture described above:

<table>
<thead>
<tr>
<th>Loop:</th>
<th>// invariant: loop counter in symb. reg. v1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>$v2 \leftarrow \text{mult } #4, v1$</td>
</tr>
<tr>
<td></td>
<td>(nop)</td>
</tr>
<tr>
<td>$S_2$</td>
<td>$v3 \leftarrow \text{load } 0\times18f0(v2)$</td>
</tr>
<tr>
<td>$S_3$</td>
<td>$v4 \leftarrow \text{load } 0\times1a20(v2)$</td>
</tr>
<tr>
<td></td>
<td>(nop)</td>
</tr>
<tr>
<td></td>
<td>(nop)</td>
</tr>
<tr>
<td></td>
<td>(nop)</td>
</tr>
<tr>
<td>$S_4$</td>
<td>$v5 \leftarrow \text{mult } v3, v4$</td>
</tr>
<tr>
<td></td>
<td>(nop)</td>
</tr>
<tr>
<td>$S_5$</td>
<td>$v6 \leftarrow \text{add } v6, v5$</td>
</tr>
<tr>
<td>$S_6$</td>
<td>$v1 \leftarrow \text{add } v1, #1$</td>
</tr>
<tr>
<td>$S_7$</td>
<td>$v7 \leftarrow \text{compare } v1, #100$</td>
</tr>
<tr>
<td></td>
<td>(nop)</td>
</tr>
<tr>
<td>Exit:</td>
<td>// now the dot product result is in $v6$</td>
</tr>
</tbody>
</table>

(a) Draw the data dependence graph for the loop. (5p) *(Hint: Do not forget anti-dependences, loop-carried dependences and control dependences.)*

(b) Suggest (any) better (in terms of execution time) schedule for a single loop iteration than the naive schedule given above. (1p)

(c) Give the largest lower bound for the minimum initiation interval based on resource constraints. (2p)

(d) Give the largest lower bound for the minimum initiation interval based on recurrence constraints. (2p)

(e) Construct a modulo reservation table for the initiation interval $II = 5$ and construct a modulo schedule. (8p) *(Hint: Note that $v_1$ can be overwritten by $S_6$ as soon as $S_1$ has executed for 1 cycle (early operand read phase). Begin placing instructions with the longest dependence cycle.)*