Linköpings universitet
IDA Institutionen för datavetenskap
Christoph Kessler

TENTAMEN

FDA001 Advanced Compiler Construction

7 april 2004, kl 9-13

Name: _____ **Personnummer:** _____

Please mark solved problems with X:

Question	1	2	3	4	5	6	7
answered							
score							

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

Sign. Examinator: _____

Lokal:

John von Neumann at IDA, and TBA, Växjö

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 examinator (073-0322959); presence (Växjö) Joakim Nivre (0470-708992)

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 (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. **Register allocation (10 p)**

   (a) Describe the principle of register allocation via graph coloring (5p).

   (b) What is register coalescing? How does it fit into a graph-coloring based framework for register allocation? What are the preconditions for coalescing? What are the advantages and (potential) disadvantages? (5p)

3. **Loop transformations (10 p)**

   Given the following loop nest:

   for i from 1 to N **do**

   for j from 1 to N **do**

   $S:$ $\quad A[i+1,j] \leftarrow A[i,j] + A[i,j+1]$

   (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)

   (b) Is it legal to interchange the i and the j loop? If yes, why? If not, why not? (3p)

   (c) Describe (in general) how loop interchange can be used to improve data cache utilization. (3p)

4. **Energy and space optimizations (4p)**

   Alternatively, one of the following questions:

   (a) Give two examples of how code generation can optimize code to reduce energy consumption. (4p)

   or:

   (b) Give an example for an algorithm for instruction scheduling (with a short description) that aims at minimizing the number of registers used. (4p)

5. **Instruction scheduling and software pipelining (16p)**

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 pseudo-targetcode for the architecture described above:

`Loop:`		// invariant: loop counter in symb. reg. $v1$
S_1	$v2 \leftarrow$ `mult` $\#4,\ v1$	// integer multiplication
	(`nop`)	// multiplication delay slot
S_2	$v3 \leftarrow$ `load 0x18f0(`$v2$`)`	// integer load $A[i]$
S_3	$v4 \leftarrow$ `load 0x1a20(`$v2$`)`	// integer load $B[i]$
	(`nop`)	// load delay slot
	(`nop`)	// load delay slot
	(`nop`)	// load delay slot
S_4	$v5 \leftarrow$ `mult` $v3,\ v4$	// integer multiplication
	(`nop`)	// multiplication delay slot
S_5	$v6 \leftarrow$ `add` $v6,\ v5$	// integer addition
S_6	$v1 \leftarrow$ `add` $v1,\ \#1$	// integer addition
S_7	$v7 \leftarrow$ compare $v1$, #100	// integer compare to loop counter in $v1$
S_8	`branch` to `Loop` if $v7! = 0$	// differ $\rightarrow$ do next of 100 iterations
	(`nop`)	// branch delay slot – even if branch is taken
`Exit:`		// now the dot product result is in $v6$

(a) Draw the data dependence graph for the loop. (3p) (*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*.)