Name _____

Computer Architecture & Implementation

EE 7700-4

Midterm Examination

3 November 1998, 18:00-18:50 CST

Problem 1	_____	(30 pts)
Problem 2	_____	(20 pts)
Problem 3	_____	(50 pts)
Exam Total	_____	(100 pts)

Alias _____

Problem 1: The code fragment below is to be run on a 3-PE trace processor with 16-instruction traces and two functional units per PE. The loop is held by exactly one trace, `Trace 1231: L0, L1, L2, L3, L4, L5, L6`; where 1231 is the trace number. Operation latencies are the same as the chapter 3 and 4 Hennessy and Patterson DLX implementations. Within a PE results are fully bypassed, between PEs there is an extra cycle delay before results can be used. The loop runs for many iterations, next-trace prediction is perfect. (30 pts)

```
LOOP:
L0:  add  r3, r1, r2
L1:  lw   r3, 0(r3)
L2:  add  r3, r3, r6
L3:  andi r6, r3, #0x39
L4:  addi r1, r1, #0x4
L5:  slt  r3, r1, r4
L6:  bneq r3, LOOP   ! For answers below branch always taken.
```

(*a*) Identify the live-in, live-out, and local registers in the trace. How much storage is needed for local register values?

(*b*) In the code above, which data values will the trace processor predict? Which values would it likely predict well? Justify your answer.

Problem 1, continued. The code from the previous page is repeated for convenience.

```
LOOP:
L0:  add  r3, r1, r2
L1:  lw   r3, 0(r3)
L2:  add  r3, r3, r6
L3:  andi r6, r3, #0x2AA
L4:  addi r1, r1, #0x4
L5:  slt  r3, r1, r4
L6:  bneq r3, LOOP   ! For answers below branch always taken.
```

(*c*) Show the execution of the code above on the trace processor using the diagram below, assuming the trace processor does *not* use data prediction.

On the diagram indicate when each instruction is dispatched (starts to execute) using the line numbers (*e.g.*, `L0, L1`). Using vertical lines or by boxing the line numbers belonging to a trace, show when each trace is issued to a PE and when it completes[1]. Initially all PEs are idle and all register values are available. Loads always hit the cache.

Cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
PE 0																					
PE 1																					
PE 2																					

(*d*) What is the issue rate (execution speed in instructions per cycle) of the trace processor running the loop above for a large number of iterations (on the trace processor without data prediction)?

[1] There is only one trace, but it is executed for each iteration of the loop. Assume the processor re-issues a trace to a PE even if it is already present.

Problem 1, continued. The code from the previous page is repeated again for convenience.

```
LOOP:
L0:  add  r3, r1, r2
L1:  lw   r3, 0(r3)
L2:  add  r3, r3, r6
L3:  andi r6, r3, #0x39
L4:  addi r1, r1, #0x4
L5:  slt  r3, r1, r4
L6:  bneq r3, LOOP   ! For answers below branch always taken.
```

(*e*) Show the execution of the code on the diagram below on a trace processor that *does* use data prediction. Assume that data prediction is perfect.

Cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
PE 0																					
PE 1																					
PE 2																					

(*f*) What is the issue rate of the trace processor running the loop above for a large number of iterations (on the trace processor with perfect data prediction)?

Problem 2: Write a single code fragment that will (*i*) run faster on a machine with a store barrier cache than one that assumes all unresolved store/load pairs are dependent; (*ii*) run faster on a machine using the store-set dependency scheme than on one using a store barrier cache; and (*iii*) run faster on a machine using memory renaming than on one using store-sets. Give register values and any other state needed so that the program has the necessary behavior.

Briefly explain how each of the load/store dependency schemes mentioned speeds execution of the program. (For reduced credit, two or more programs can be written.) (20 pts)

Problem 3: Answer each question below.

(*a*) Two simulation techniques, dynamic compilation and augmentation, are being considered for evaluating two proposed architectural features, a new ALU instruction and additional integer registers. (The new ALU instruction computes the number of `1`'s in the operand's binary representation. The number of integer registers is increased from 32 to 64.) Compilers are available that can emit code for each feature. Accurate timing data is *not* needed.

If feasible, how would a dynamic compilation system that can simulate the unmodified ISA (for example Shade) be changed to simulate each of the two proposed features? If feasible, how would augmentation be used to simulate each feature? If it is infeasible for either simulation technique to simulate either feature, explain why. (10 pts)

(*b*) As described in class, a gshare branch predictor in which the size of the outcome history shift register is smaller than the address size of the BHT performs better when the outcome history is exclusive-ored with the higher-order bits of the branch address than if the lower-order bits were used. (10 pts)

Explain why, describing specific problems that would occur if the lower order bits were used.

Suppose the following alternative scheme were used. The branch predictor has a table of distinct random numbers (uniformly distributed over $[0, 2^n)$ where 2^n is the number of BHT entries, ignore the cost). The branch address is used to retrieve a number from this table, which is exclusive ored with the outcome history. Would it matter how the outcome history was aligned with the number? How would prediction accuracy compare to versions of gshare in which outcome history was exclusive ored with the higher-order and lower-order address bits? Explain.

(*c*) When a processor discovers a load/store dependence misspeculation twelve instructions past the load have been issued and ten of them (none of which are branches) are data-dependent on the load. Explain why re-execution would be faster than squashing in this situation. Explain why re-execution would be faster even if all instructions following the load were data-dependent. (10 pts)

(*d*) Do the ILP limits obtained by Wall (described in Section 4.7 of Hennessy and Patterson) apply to processors (*i*) using multiple-path eager execution? (*ii*) trace processors without data prediction? (*iii*) trace processors using data prediction? Explain. (10 pts)

(*e*) What are the four outcome patterns in a fixed-pattern confidence estimator, how are they used, and why were they chosen? (10 pts)

Would a similar technique for a gshare or gselect branch predictor (possibly with different fixed patterns) be effective? Explain.