EE 4720 **Homework 5** **Due: 18 March 2001**

Problem 1: Solve Problems 3 and 4 from Fall 2000 Homework 5, available via `http://www.ee.lsu.edu/ee4720/2000f/hw05.pdf`. Using the solutions at `http://www.ee.lsu.edu/ee4720/2000f/hw05_sol.pdf` assign yourself a grade in the range $[0, 1]$. Either: indicate the grade you assigned yourself or write "Did not solve." A solution can be provided along with a grade. It will be corrected but your grade will be used. If you opt not to solve it you will receive full credit but will be hurting your ability to solve future problems.

Most of the problems below ask about the Alpha 21264 implementation of the Alpha Architecture. The answers to these questions can be found in Kessler 99, R. E. Kessler, "The Alpha 21264 microprocessor," IEEE Micro Magazine, March 1999, vol. 19, no. 2, pp. 24–36, available via `http://www.ee.lsu.edu/ee4720/kessler99.pdf`

Problem 2: Which of the dynamic scheduling methods described in class most closely matches the 21264? What terminology does Kessler 99 use for the following three terms (as used in class): Reorder Buffer, Commit, and Physical Register Number?

Problem 3: The 21264 is described as a four-way superscalar processor. What are the maximum number of instructions that can issue per cycle? What are the maximum number of instructions that can commit per cycle? How do these numbers differ from the corresponding values for the default dynamically scheduled machine as described in class? Call the higher of these two numbers x. Why would DEC (okay, Compaq) dare not call the 21264 an x-way superscalar processor?

Problem 4: What is the size of the reorder buffer in the 21264? How is its use slightly different than the one described in class?

Problem 5: The stages making up the 21264 (Figure 2 in Kessler 99) differ from the stages in the dynamically scheduled DLX implementation (using the appropriate method) described in class. For each 21264 stage indicate which DLX stage does the equivalent (or close) work. Consider Slot 1 to be an extension of the fetch stage.

Problem 6: A gshare or gselect two-level predictor can perfectly predict the loop branches for loops with a small, constant number of iterations, such as:

```
 addi r1, r0, #3
LOOP:
 lw r2,0(r3)
 beqz r2, SKIP
 sw 4(r3), r2
SKIP:
 addi r3, r3, #8
 subi r1, r1, #1
 bnez r1, LOOP
```

Consider a processor using a gshare predictor with a 12-bit global history register. Would the processor predict the last branch perfectly (after warmup and assuming no collisions in the BHT)? *Hint: Yes.* Modify the loop above by adding code between `LOOP` and `SKIP` so that the gshare predictor no longer predicts the last branch correctly. The number of loop iterations must not change.

Problem 7: Suppose the code from the previous problem, translated to Alpha, ran on the Alpha 21264. Assuming no collisions, why would the modifications made in the previous problem not remove the perfect prediction of the last branch?

(Information to solve the problem can be found on page 27 (PDF page 4) of `http://www.ee.lsu.edu/ee4720/kessler99.pdf`. The McFarling paper, not needed to solve this problem but referenced by Kessler, can be found via `http://www.ee.lsu.edu/tca/mcf.pdf`.)