UNIVERSITY OF CALIFORNIA, DAVIS
Department of Electrical and Computer Engineering

EEC171 Parallel Computer Architectures Spring 2009

Homework 1 Due Wednesday 22 April 2009 12:00 pm in homework box

As a professor, I want to give you interesting homework problems and make sure that you cover the breadth of the course material. This is a 4 unit class and I'm only giving 3 homework assignments, so I feel justified in giving homework assignments that are fairly involved. However, I know that your time is valuable, so we'll try allowing you to select from a range of problems. Please let me know what you think about this policy and also about what you think about the amount of homework.

The homework is to choose 6 of 8 book problems and to do at least half the point-value of problems from the midterm exam from two years ago. I will distribute solutions to all problems in class on Monday 20 April.

Problems from H&P. Please do 6 of the following 8 problems.

- 2.1–2.6. Explores instruction scheduling, reordering, multiple-issue.
- 2.10. VLIW & register use.
- 2.11. Branch prediction.

3.1 and 3.2 are terrific problems, and I'm really tempted to assign them, but out of respect for your time, probably not a good idea. However, you should look through them—I'd expect you to be able to do a problem like that.

Additional questions. These are from the ILP midterm from two years ago and are thus great preparation for an exam. There are 60 points of questions. You must do at least 30 points of questions for this homework, although all the questions are good practice. You would have had an entire class (110 minutes) to complete this exam.

1. In this question we will explore the differences between speculation and prediction.

   (a) (4 points) Would the following code segment be better suited for *speculation* or *predication*, and why?

```
if (predictableBranchThatTakesALongTimeToEvaluate) {
  hugeFunction();
} else {
  otherHugeFunction();
}
```

   (b) (4 points) Would the following code segment be better suited for *speculation* or *predication*, and why?

```
if (unpredictableBranchThatTakesALongTimeToEvaluate) {
  shortFunction();
} else {
  otherShortFunction();
}
```

2. The code segment in Figure 1 (on the last page) is an (actual!) 2×2 discrete cosine transform, used in signal and image processing applications (like JPEG compression). For all parts of this problem, assume each instruction takes 1 cycle to execute (results are available on the next cycle). This problem is the basis for the rest of the exam—it would be to your benefit to draw the dependency graph for the code on the last page with the code, since you will use the dependencies in many parts of this problem.

   (a) Identify one instance of each of the following types of hazards, in the form of "for this type of dependency, variablename in instruction n is dependent on instruction m" (for which you can write "variablename $n \to m$")

      i. (2 points) Read-after-write (true dependency)

      ii. (2 points) Write-after-write (output dependency)

      iii. (2 points) Write-after-read (anti-dependency)

   (b) (5 points) Assuming register-renaming hardware and an infinite number of renameable registers, how much instruction-level parallelism is in this code sequence? Express your answer as instructions per cycle (IPC).

(c) (3 points) Consider a multiple-issue machine with the following types of functional unit:

- Multiplier (multiply operations only)
- Arithmetic (add/subtract operations only)
- Logical (shifts and logical operations only)
- Load/store (memory loads and stores only)
- Branch (control flow operations only)

For this code sequence, to be able to achieve the maximum level of instruction-level parallelism you computed in part b, what is the minimum issue rate (instructions per cycle)?

(d) (3 points) For this code sequence, to be able to achieve the maximum level of instruction-level parallelism you computed in part b, what is the minimum number and mix of functional units?

(e) (10 points) Assume a superscalar machine with 8 functional units (2 multipliers, 2 arithmetic units, 2 logical units, and 2 load-store units). Assume this machine is *completely in-order* (in-order issue, execute, and commit)—no instruction can issue before another instruction that comes earlier in the same sequence (but if there are no cross-dependencies, at the same time is OK). What is the minimum number of cycles to complete the code sequence? Show your schedule below, writing the appropriate instruction number in each box (don't write anything for unused hardware on a given cycle).

$\times$	$\times$	+/-	+/-	logical	logical	ld/st	ld/st

Number of cycles:

(f) Assume a superscalar machine with 8 functional units (2 multipliers, 2 arithmetic units, 2 logical units, and 2 load-store units). Assume this machine is *completely out-of-order* (out-of-order issue, execute, and commit), with perfect register renaming, perfect memory disambiguation, and an instruction window of infinite size.

i. (10 points) What is the minimum number of cycles to complete the code sequence? Show your schedule below, writing the appropriate instruction number in each box (don't write anything for unused hardware on a given cycle).

$\times$	$\times$	+/-	+/-	logical	logical	ld/st	ld/st

Number of cycles:

ii. (3 points) For this OOO machine, is register renaming useful in improving the performance of this code segment (over not having register renaming)? Explain why or why not (using particular instructions in the code sequence to help explain if applicable).

iii. (3 points) For this OOO machine, is out-of-order issue/execution/commit useful in improving the performance of this code segment (over an in-order machine)? Explain why or why not (using particular instructions in the code sequence to help explain if applicable).

iv. (3 points) For this OOO machine, what is the minimum size of the instruction window (in number of instructions) that allows maximum performance on this code segment?

v. (3 points) For this OOO machine, is memory disambiguation hardware useful in improving the performance of this code segment (over not having memory disambiguation hardware)? Explain why or why not (using particular instructions in the code sequence to help explain if applicable).

(g) (3 points) For this instruction sequence, note that it is possible to use the compiler to statically reorder instructions so it is just as fast on an in-order machine as dynamic rescheduling done by out-of-order hardware. If that is the case, then why do we use out-of-order hardware at all? Name one advantage of an out-of-order processor compared to a perfectly scheduled in-order processor (not necessarily on this code sequence). (Assume both processors have perfect branch prediction.)

```
tmp4 = coef0 * quant0
tmp5 = coef2 * quant2
tmp0 = tmp4 + tmp5
tmp2 = tmp4 - tmp5
tmp4 = coef1 * quant1
tmp5 = coef3 * quant3
tmp1 = tmp4 + tmp5
tmp3 = tmp4 - tmp5
tmp4 = tmp0 + tmp1
tmp4 = tmp4 + 8
tmp4 = tmp4 >> 4
tmp4 = tmp4 & 255
out0 = load(tmp4)
tmp5 = tmp0 - tmp1
tmp5 = tmp5 + 8
tmp5 = tmp5 >> 4
tmp5 = tmp5 & 255
out1 = load(tmp5)
tmp4 = tmp2 + tmp3
tmp4 = tmp4 + 8
tmp4 = tmp4 >> 4
tmp4 = tmp4 & 255
out2 = load(tmp4)
tmp5 = tmp2 - tmp3
tmp5 = tmp5 + 8
tmp5 = tmp5 >> 4
tmp5 = tmp5 & 255
out3 = load(tmp5)
```

Figure 1: Code for the DCT in Question 2. You should think about drawing a dependency graph on this page for your use on the question.