Chapter 10

1. Figure 10-1 suggests it is possible to improve IPC from 1 to 4 by employing techniques such as instruction reuse or value prediction that collapse true data dependences. However, publications describing these techniques show speedups ranging from a few percent to a few tens of percent. Identify and describe one program characteristic that inhibits such speedups.

   Programs that are not purely dataflow-limited will not experience this type of speedup. Programs can also be control-limited (i.e. they would need better branch prediction). Furthermore, the value prediction or instruction reuse may be breaking data dependences that are not on the program's critical path. Also, programs that have abundant ILP to begin with may not need these techniques to increase the amount of ILP extracted by the hardware; namely, if the dependence distances are relatively large, value prediction and reuse do not help, since by the time a dependent instruction is fetched into the window, it's result has already been computed even without value prediction.

2. As in Problem 1, identify and describe at least one implementation constraint that prevents best-case speedups from occurring.

   Machine bandwidth at any stage in the pipeline (fetch, decode, dispatch, issue, commit) may limit performance. Similarly, other aspects of the machine, such as inaccurate branch prediction, can eliminate much of the benefit of value prediction.

3. Assume you are implementing instruction reuse for integer instructions in the PowerPC 620. Assume you want to perform the reuse test based on value in the dispatch stage. Describe how many additional read and write ports you will need for the integer architected register file (ARF) and rename buffers.

   To perform the reuse test for four instructions, up to eight source operands need to be checked, so eight additional read ports will be needed. Also, up to four instructions can be reused, hence four results may need to be written into the rename buffers, requiring four additional write ports into the rename buffers.

4. As in Problem 3, assume you are implementing instruction reuse in the PowerPC 620, and you wish to perform the reuse test by value in the dispatch stage. Show a design for the reuse buffer that integrates it into the 620 pipeline. How many read/write ports will this structure need?

   Detailed solution not provided. The structure will need four read ports, one for each dispatched instruction.Assuming full update bandwidth, it will also need four write ports. Finally, it will need one snoop port for checking stores against matching load reuse entries.

5. Assume you are building an instruction reuse mechanism that attempts to reuse load instructions by performing the reuse test by name in the PowerPC 620 dispatch stage. Since the addresses of all prior in-flight stores may not be known at this time, you have several design choices: 1) either disallow load reuse if stores with unknown addresses are still in flight, 2) delay dispatch of reused loads until such prior stores have computed their addresses, or 3) go ahead and allow such loads to be reused, relying on some other mechanism to guarantee correctness. Discuss these three alternatives from a performance perspective.

Option 1: there is almost always a store in flight (this would be even more true for modern, deeper pipelines), so almost no load reuse would occur

Option 2: again, there is almost always a store in flight, so this would cause lots of dispatch stalls. Since dispatch is in-order, all subsequent instructions also get stalled. Hence, any benefit from reuse would likely be negated by these additional stalls.

Option 3: this appears like the only feasible approach, from a performance perspective. One could rely on the 620's load queue for correctness, by placing the reused loads into the load queue, just as any other load, and then checking the load queue every time a store commits to make sure there is no alias. This would be fairly straightforward, and should enable some performance gains from reuse.

6. Given the assumptions in Problem 5, describe what existing microarchitectural feature in the PowerPC 620 could be used to guarantee correctness for the third case? If you choose the third option, is your instruction reuse scheme still nonspeculative?

   See previous solution. No, the reuse scheme is now speculative, since reused loads can violate memory RAW dependences, causing squashes whenever a store matches an entry in the load queue.

7. Given the scenario described in Problem 5, comment on the likely effectiveness of load instruction reuse in a five-stage pipeline like the PowerPC 620 versus a 20-stage pipeline like the Intel Pentium 4? Which of the three options outlined is likely to work best in a future deeply-pipelined processor? Why?

   As the pipeline gets deeper (620 -> Pentium 4 -> future processors?), the likelihood of an outstanding inflight store increases dramatically (approaching 100% for the Pentium 4). Hence, options 1 and 2 become infeasible, and option 3 is the only practical solution. Even for option 3, though, a deeper pipeline increases the probability of a violation that will cause the load reuse to fail.

8. Construct a sequence of load value outcomes where a last value predictor will perform better than a FCM predictor or a stride predictor. Compute the prediction rate for each type of predictor for your sequence.

   A, A, A, A, ...

   The last value predictor will outperform FCM and stride due to its lower training cost, and will correctly predict A after a single misprediction. The stride predictor has to observe the stride of 0 at least once, possibly twice, before learning that it is zero, while the FCM predictor will have to fill its context history to find A. If the context history is deep, this will require multiple instances of the load before its value predictions are correct.

9. Construct a sequence of load value outcomes where an FCM predictor will perform better than a last-value predictor or a stride predictor. Compute the prediction rate for each type of predictor for your sequence.

A,B,C,A,B,C,A,B,C,...

This kind of repeating sequence is tailor-made for FCM. After training, the FCM predictor will asymptotically approach 100% accuracy, while last value and stride predictors will get 0%.

10. Construct a sequence of load value outcomes where a stride predictor will perform better than an FCM predictor or a last value predictor. Compute the prediction rate for each type of predictor for your sequence.

    0,2,4,6,8,10,12,14,16,18,20,...,n,n+2,...

    In this case, the stride predictor quickly learns the stride is two and will asymptotically approach 100% accuracy. The last value and FCM predictors will utterly fail.

11. Consider the interaction between value predictors and branch predictors. Given a stride value predictor and a two-level Gag branch predictor with a 10-bit branch history register, write a C-code program snippet for which the stride value predictor can correct a branch that the branch predictor mispredicts.

```
while (1) {
  for(int i=0;i<36;i+=3) {
    .. /* do something */
  }
}
```

    The inner loop iterates 12 times; 12 branches will overwhelm the 10-bit BHR, preventing the branch predictor from correctly predicting that the last iteration's closing branch is not taken. Hence, one branch out of every 12 will be mispredicted. However, a stride value predictor will quickly learn that i strides by three, and will be able to resolve the branch quickly.

12. Consider further the interaction between value predictors and branch predictors. Given a last value predictor and a two-level Gag branch predictor with a 10-bit branch history register, write a C-code program snippet for which the last value predictor incorrectly resolves a branch that the branch predictor predicts correctly.

```
while (1) {
  y = 0;
  x = 0;
  while (x == 0) {
    y += 4;
    x = y >> 4;
  }
}
```

The inner loop will iterate four times; the 10-bit BHR is enough to capture this and correctly predict the last iteration. However, the value predictor will predict that x is zero for that iteration, which will redirect the correctly-predicted loop-closing branch.

13. Given that a value predictor can incorrectly redirect correctly predicted branches, suggest and discuss at least two microarchitectural alternatives for dealing with this problem.

    One solution would be to trust the branch predictor over the value predictor, and prevent value predictions from resolving branches. This could be extended to keep prediction confidence on both the value predictions and the branch predictions and rely on the prediction with the higher confidence whenever the two disagree (similar to a tournament or combining branch predictor).

14. Assume you are implementing value prediction for integer instructions in the PowerPC 620. Describe how many additional read and write ports you will need for the integer architected register file (ARF) and rename buffers.

    No additional read ports are needed. Further, no additional write ports to the ARF are needed. However, four additional write ports to the rename buffers are needed, since up to four instructions can be predicted per cycle.

15. As in Problem 14, assume you are implementing value prediction in the PowerPC 620. You have concluded that you need selective reissue via global broadcast as a recovery mechanism. In such a mechanism, each inflight instruction must know precisely which earlier instructions it depends on, either directly or indirectly through multiple levels in the data flow graph. For the PowerPC 620, design a RAM/CAM hardware structure that tracks this information, and enables direct selective reissue when a misprediction is detected. How many write ports does this structure need?

    Detailed design is not provided. This mechanism can be tacked on to the renamer, since the problem is equivalent: when instructions are renamed, they read a bit vector for each source operand that identifies all instructions with value predictions that could pollute those operands. The source bit vectors are merged (logical or) and then written into the destination register's bit vector. If there are dependent instructions in a parallel rename group, this information must be bypassed, just as matching dest/source rename information must by bypassed.

16. For the hardware structure in Problem 15, determine the size of the hardware structure (number of bit cells it needs to store). Describe how this size would vary in a more aggressive microarchitecture like the Intel P6, which allows up to 40 instructions to be in flight at one time.

    Detailed design is not provided. The bit vectors are proportional in size to the total number of inflight instructions. Hence, the array grows linearly in proportion to the instruction window size.

17. Based on the data in Figure 10-10, provide and justify one possible explanation for why the *gawk* benchmark does not achieve higher speedups with more aggressive value prediction schemes.

    Given the dramatic speedup gawk already experiences, it is likely that the machine parallelism is saturating. Hence, no amount of better value prediction results in any more realizable ILP, since the machine does not have enough bandwidth.

18. Based on the data in Figure 10-10, provide and justify one possible explanation for why the *swm256* benchmark achieves dramatically higher speedup with the Perfect value prediction scheme.

    Going back to Figure 10-3, swm256 has very low prediction accuracy with finite predictors. Hence, it stands to gain alot of benefit from an ideal predictor.

19. Based on the data in Figure 10-3 and Figure 10-10, explain the apparent contradiction for the benchmark *sc*: even though roughly 60% of its register writes are predictable, no speedup is obtained from implementing value prediction. Discuss at least two reasons why this might be the case.

    The sc benchmark may be machine bandwidth limited, i.e. already achieving ILP comparable to what the machine can supply. Alternatively, it may be control-flow limited, and no amount of better value prediction can overcome the branch misprediction delays it experiences.

20. Given your answer to Problem 19, propose a set of experiments that you could conduct to validate your hypotheses.

    Experiment with a wider, more aggressive machine model, then experiment with a perfect branch predictor.

21. Given the deadlock scenario described in Section 10.4.4.5, describe a possible solution that prevents deadlock without requiring all speculatively issued instructions to retain their reservation stations. Compare your proposed solution to the alternative solution that forces instructions to retain their reservation stations until they are deemed nonspeculative.

    The simplest solution is to reserve one (or more) entries in the RS for reinserting incorrectly issued instructions. This will avoid deadlock, but may not perform well, since only a single slot is available and can become a bottleneck if many instructions need to be reinserted. Of course, one could reserve multiple slots. However, conversely, in cases where no reinsertions need to occur, the reserved slots are wasted and would be better utilized for other instructions. A hybrid scheme might reserve a slot only when there are speculatively issued instructions in the ROB (i.e. reserve a single slot for all the speculatively issued instructions). An alternative is to provide a separate issue path for reissued instructions (instead of reinserting them in the RS). Schemes like this are employed in the Pentium 4 recovery logic as well as the WIB proposal by Rotenberg et al.