Chapter 7

1. The PowerPC 620 does not implement load/store forwarding, while the Pentium Pro does. Explain why both design teams are likely to have made the right design tradeoff.

   Load/store aliases are relatively rare for the PowerPC 620 instruction set, since there are plenty of registers and the compiler can store most actively used variables in registers. Hence, load/store forwarding is not critical for performance. However, the x86 ISA is very register-constrained (only 8 integer registers), so most variables are stored in memory. Hence, there are more memory references and the probability of a store/load alias increases dramatically. Hence, for a register-starved ISA like x86, load/store forwarding is much more important.

2. The P6 recovers from branch mispredictions in a somewhat coarse-grained manner, as illustrated in Figure 7-5. Explain how this simplifies the misprediction recovery logic that manages the reorder buffer (ROB) as well as the register alias table (RAT).

   The P6 pipeline simply drains the out-of-order core before it allows any new instructions following a mispredicted branch to enter. This makes RAT recovery trivial, since once all inflight instructions have drained, the RAT reverts to pointing at the architected register file for all entries. Hence, the RAT can be flash-reverted trivially. Similarly, the reorder buffer does not need to support "partial flushes" where some instructions are deallocated while others remain active. Instead, the standard retirement logic is used to release ROB (and MOB) resources while no new instructions are allowed to be inserted, until the ROB is empty. This is simple and elegant. However, it can increase the branch misprediction delay, since there may be long latency instructions (e.g. cache misses) that take a long time to drain out of the pipeline, and these will prevent new useful instructions from entering the pipeline.

3. AMD's Athlon (K7) processor takes a somewhat different approach to dynamic translation from IA32 macroinstructions to the machine instructions it actually executes. For example, an ALU instruction with one memory operand (e.g., add eax,[eax]) would translate into two μops in the Pentium Pro: a load that writes a temporary register followed by a register-to-register add instruction. In contrast, the Athlon would simply dispatch the original instruction into the issue queue as a macro-op, but it would issue from the queue twice: once as a load and again as an ALU operation. Identify and discuss at least two microarchitectural benefits that accrue from this "macro-op" approach to instruction-set translation.

   This increases the machine bandwidth through the decode and dispatch stages, since a single macro-op now can express the semantics of an entire x86 op. Also, it reduces occupancy of scheduler queue entries and ROB entries, allowing for an effectively larger instruction window. A similar approach is used in the new P6-derived Banias (Centrino) mobile processor cores from Intel.

4. The P6 two-level branch predictor has a speculative and nonspeculative branch history register stored at each entry. Describe when and how each branch history register is updated, and provide some reasoning that justifies this design decision.

   The speculative BHR is updated after a prediction is made, while the nonspeculative BHR is updated whenever a branch is retired (committed). The speculative BHR is used to make predic-

tions, since it always has up-to-date branch history in it. However, in case of a misprediction, this branch history is incorrect. Hence, the nonspeculative history is used to recover to a correct history on a branch misprediction. Having a correct and up-to-date history improves the accuracy of the predictor. Without two copies, one speculative and up-to-date, and one nonspeculative, the predictor could not have both of these properties (correct and up-to-date branch history).

PROJECT IDEA: modify the simplescalar simulator to verify that these two properties are important (having a correct history after a misprediction and having an up-to-date history). Now modify the simplescalar pipeline so it is longer (10-15 pipestages) and commit-time branch history updates occur later. How important are these properties with a deeper pipeline?

5. The P6 dynamic branch predictor is backed up by a static predictor that is able to predict branch instructions that for some reason were not predicted by the dynamic predictor. The static prediction occurs in pipe stage 17 (refer to Figure 7-4). One scenario in which the static prediction is used occurs when the BTB reports a tag mismatch, reflecting the fact that it has no branch history information for this particular branch. Assume the static branch prediction turns out to be correct. One possible optimization would be to avoid installing such a branch (one that is statically predictable) in the BTB, since it might displace another branch that needs dynamic prediction. Discuss at least one reason why this might be a bad idea.

   The branch was statically predictable once, but future instances may not be, so dynamic prediction information can be useful. Furthermore, the static prediction still inserts a pipeline bubble if the branch is taken, which can be avoided with the dynamic branch predictor.

6. Early in the P6 development, the design had two different ROBs, one for integers and another for floating-point. To save die size, these were combined into one during the Pentium Pro development. Explain the advantages and disadvantages of the separate I/F ROBs versus the unified ROB.

   Advantages of unified ROB: a shared resource is always better, since all slots can be filled with integer ops for integer-intensive programs, and vice-versa for FP-intensive programs. Also, tracking in-order retirement of instructions is easier with a single unififed ROB.

   Advantage of separate ROB: the data paths are now separated, leading to potential cycle-time gains (i.e. only the FP ALUs need to be connected to the FP ROB, and int ALUs need to be connected to the int ROB). However, the P6 executes some int instructions (e.g. int divide) in the FP ALU, so this benefit may not be realizable without adding a separate int divide unit.

7. From the timing diagrams you can see that the P6 retirement process takes 3 clock cycles. Suppose you knew a way to implement the ROB so that retirement only took 2 clock cycles. Would you expect a substantial performance boost? Explain.

   One would expect some benefit for programs that are ROB-limited. I.e. programs that fill the ROB with useful work would likely benefit if they could bring in additional instructions more quickly, reducing ROB occupancy, hence increasing its effective size and exposing more ILP. However, reducing ROB occupancy by 1 cycle per instruction is not likely to be too dramatic.

8. Section 7.3.4.5 describes a mismatch stall that occurs when condition flags are only partially written by an in-flight μop. Suggest a solution that would prevent the mismatch stall from occurring in the renaming process.

   Partial flag writes are painful; if they are rare, the solution described is the right tradeoff between complexity and performance. However, if they occur more often, each flag can be renamed separately. Now each flag writer has to update the flag RAT for each flag that it writes, and each flag reader has to check the flag RAT to find each of the flags. If there are many flags, this results in a lot more read/write bandwidth into the flag RAT as well as the flag portion of the ROB and ARF. This could also be done in a slightly less painful way by grouping flags into subsets that are commonly written and read together.

   PROJECT IDEA: investigate the x86 instruction set and propose a detailed design for renaming of partial flag writes. Use the bochs open-source x86 emulator (bochs.sourceforge.net) to evaluate the frequency of partial flag writes and estimate the benefit of your scheme.

9. Section 7.3.5.3 describes the RS allocation policy of the Pentium Pro. Based on this description, would you call the P6 a centralized RS design or a distributed RS design? Justify your answer.

   The "static binding" described in the text appears to indicate a hybrid design that is not truly centralized but not fully distributed either. It is not completely clear from the text.

10. If the P6 microarchitecture had to support an instruction set that included predication, what effect would that have on the register renaming process?

    Predicated instructions complicate renaming, since a false predicate nullifies the register write of an instruction that otherwise writes a register. Hence, until the predicate is known, the renamer does not know whether subsequent instructions should read from the previous or the new definition of a register written by a predicated instruction. Hence, the renamer could stall until the predicate is determined. Or, it could insert a move operation after the predicated op that reads both the old and new definitions of the predicated instruction's destination register, and then copies one or the other definition to its own (nonarchitected) destination. All subsequent readers will then get renamed to the output of the move operation.

11. As described in the text, the P6 microarchitecture splits store operations into a STA and STD pair for handling address generation and data movement. Explain why this makes sense from a microarchitectural implementation perspective.

    Logically, the STA and STD perform two different operations that interact with different control portions of the microarchitecture: the STA uses an AGEN unit to generate the address, and then resides in the MOB to resolve memory dependences against newer loads. The STD simply transfers data from the register file to the store port at commit. Hence, it makes sense to split them. Note that the new Banias (Centrino) designs based on the P6 core no longer split STA/STD, but treat them as a single micro-op. This increases decode bandwidth and reduces ROB and RS occupancy.

12. Following up on Problem 11, would there be a performance benefit (measured in instructions per cycle) if stores were not split? Explain why or why not?

    Front-end decode bandwidth would increase, while ROB and RS occupancy would decrease, permitting an effectively larger window. Also, it is possible that commit bandwidth would increase.

13. What changes would one have to make to the P6 microarchitecture to accommodate stores that are not split into separate STA and STD operations? What would be the likely effect on cycle time?

    The issue logic for merged store operations must track readiness of the address operand and the data operand, requiring twice as much wakeup bandwidth. This may or may not increase complexity and/or cycle time. The MOB would similarly have to accommodate STA/STD as a single entity. Of course, the decoder would also have to be modified to emit the merged operation. Interestingly, the Banias (Centrino) core does exactly this, and does not appear to have suffered a frequency loss, and in fact has better per-clock performance than P6.

14. AMD has recently announced the x86-64 extensions to the Intel IA32 architecture that add support for 64-bit registers and addressing. Investigate these extensions (more information is available from www.amd.com) and outline the changes you would need to make to the P6 architecture to accommodate these additional instructions.

    Solution not provided.