Chapter 6

1. Assume the IBM instruction mix from Chapter 2, and consider whether or not the PowerPC 620 completion buffer versus integer rename buffer design is reasonably balanced. Assume that load and ALU instructions need an integer rename buffer, while other instructions do not. If the 620 design is not balanced, how many rename buffers should there be?

   The IBM instruction mix states that 40% of instructions are register-register ALU ops, and 25% are loads. Hence, 65% of all instructions need a rename register. The PowerPC 620 provides 8 rename buffers and 16 completion buffer entries. Hence, only 50% (8/16) of the inflight instructions in the completion buffer can have a rename register. This appears unbalanced for a program with the IBM instruction mix. A more appropriate number would be 65% x 16 = 11 or 12 rename registers.

2. Assuming instruction mixes in Table 5-2, which benchmarks are likely to be rename buffer constrained? That is, which ones run out of rename buffers before they run out of completion buffers? And vice-versa?

   Any benchmark where the sum of ALU and loads is greater than 50% will be rename buffer constrained. For integer rename buffers, this includes all four (compress, eqntott, espresso, li). For floating point, alvinn has enough (12.27%+.08%+26.85% < 50%), hydro2d has nearly enough (26.99%+1.87%+22.53% is slightly more than 50%), while tomcatv is FP rename register constrained (37.82%+0.7%+27.84% >> 50%).

3. Given the dispatch and retirement bandwidth specified, how many integer ARF (architected register file) read and write ports are needed to sustain peak throughput? Given instruction mixes in Table 5-2, also compute average ports needed for each benchmark. Explain why you would not just build for the average case. Given the actual number of read and write ports specified, how likely is it that dispatch will be port-limited? How likely is it that retirement will be port-limited?

   For peak throughput, 8 ARF read ports are needed at dispatch, and 4 ARF write ports are needed at retirement. Given an average mix of 36.14% ALU which require 2 read ports and 1 write port, 1.44% multicycle ALU which require 2 read ports and 1 write port, 14.64% int loads which require one read port and one write port, 11.03% FP loads which require one read port but no write ports (only write FP), 7.18% int stores which require two read ports and no write ports, and 4.12% FP stores which require one read port. The weighted average for 4-wide dispatch read ports is 4 x (.3614x2 + .0144x2 + .1464x1 + .1103x1 + .0718x2 + .0412x1) = 4.77 read ports. The weighted average for 4-wide retirement write ports is 4 x (.3614 + .0144 + .1464) = 2.1 write ports.

   Building for the average case causes stalls whenever there are bursts of instructions that require multiple read ports. It is unlikely that dispatch and commit will be port-limited.

4. Given the dispatch and retirement bandwidth specified, how many integer rename buffer read and write ports are needed? Given instruction mixes in Table 5-2, also compute average ports needed for each benchmark. Explain why you would not just build for the average case.

The dispatch bandwidth is four instructions, with two source operands each. Hence, eight read ports are required. However, one of these instructions can be a stx instructions, which has three source operands, and one can be a floating-point multiply-add, which also has three source operands. Hence, the total read ports is actually 10, or 7 integer read ports and 3 floating-point read ports. Each instruction can write one register, so at dispatch time up to 4 write ports are needed. However, the load-update instructions can write two registers, so an extra write port is needed for those. Hence, the dispatch stage needs 5 integer write ports, and two floating-point write ports (one for an FP op, one for a FP load). The completion stage is limited by 4 integer and 2 floating-point register file write ports. Hence, up to 9 integer write ports may be needed, while up to 4 (2 at dispatch, 2 at commit) floating-point write ports are needed. Building for the average case causes stalls whenever there are bursts of instructions that require multiple ports.

5. Compare the PowerPC 620 BTAC design to the next-line/next-set predictor in the Alpha 21264 as described in the *Alpha 21264 microprocessor hardware reference manual* (available from www.compaq.com).

   Essay solution not provided.

6. How would you expect the results in Table 5-5 to change for a more recent design with a deeper pipeline (e.g. 20 stages, like the Pentium 4)?

   Assuming benchmarks with high branch prediction rates, one would expect many more branches to be in flight. Hence, the distribution would shift towards the right-hand columns of the table. This effect is due to the fact that the number of cycles separating the fetch of a branch from its resolution would increase (in the Pentium 4, the minimum time from fetch to resolution for a branch is 20 cycles, vs. 4 cycles in the 620).

7. Judging from Table 5-7, the PowerPC 620 appears reservation station-starved. If you were to double the number of reservation stations, how much performance improvement would you expect for each of the benchmarks. Justify your answer.

   Essay solution not provided. Answer should note that other bottlenecks (rename buffer saturation and completion buffer saturation) would likely preclude dramatic gains. In cases where reservation station saturation is dominant, there could be considerable gains.

8. One of the most obvious bottlenecks of the 620 design is the single load/store unit. The IBM POWER3, a subsequent design based heavily on the 620 microarchitecture, added a second load/store unit along with a second floating-point multiply/add unit. Compare the SPECINT2000 and SPECFP2000 score of the IBM POWER3 (as reported on www.spec.org) with another modern processor, the Alpha AXP 21264. Normalized to frequency, which processor scores higher? Why is it not fair to normalize to frequency?

   (Comparison from www.spec.org not provided). Normalizing to frequency is not fair, since one design may sacrifice ILP or IPC in order to gain higher frequency. When comparing performance, all three terms of the iron law must be included.

9. The data in Table 5-10 seems to support the 620 designers' decision to not implement load/store forwarding in the 620 processor. Discuss how this trade-off changes as pipeline depth increases and relative memory latency (as measured in processor cycles) increases.

   The deeper the pipeline, the unretired stores there will be in flight in the store buffer. This increases the probability that a load will alias to one of those stores, resulting in an increase in the number of cases where loads have to stall to wait for a store to retire. Furthermore, with a deeper pipeline, they will have to wait longer. With a deeper pipeline, this tradeoff should be revisited. In fact, the POWER4 design implements load forwarding, so that loads that alias to a store in the store buffer can obtain their data from the store buffer with a slight delay. As relative memory l atency increases, the same trends hold true: the fact that a load or store instruction misses the cache and blocks subsequent stores from retiring while the miss is being resolved will increase both the probability and the latency of satisfying aliased loads. Here again, a load forwarding scheme will alleviate the performance-degrading effects.

10. Given the data in Table 5-9, present a hypothesis for why XSU1 appears to have consistently longer execution latency than XSU0. Describe an experiment you might conduct to verify your hypothesis.

    From Table 5-6, we see that utilization for the fixed point RS is fairly low. Furthermore, with the exception of compress, XSU1 is consistently lower, despite the fact that average execution latency (from Table 5-9) is higher for XSU1. Hypothesis: the dispatch logic tends to place instructions with intra-dispatch group dependences into XSU1, whereas instructions without intra-dispatch group dependences are placed in XSU0. This happens because the first integer instruction will always go into XSU0 (unless it is full, which it almost never is), and the first integer instruction is less likely (unless preceded by a load) to have intra-group dependences. The second integer instruction, if any, will end up in XSU1, and is more likely to have intra-group dependences. One could construct several microbenchmarks to test this hypothesis. In one microbenchmark, a dispatch group would contain two integer instructions, but ones that were independent, while in another microbenchmark, the dispatch group would contain two dependent integer instructions. The first microbenchmark should report similar execution latency for both XSU0 and XSU1, while the second should reflect behavior similar to that in Table 5-9.

11. The IBM POWER3 can detect up to four regular access streams and issue prefetches for future references. Construct an address reference trace that will utilize all four streams.

    Example code: for(i=0;i<10000;++i) sum += A[i] + B[i] + C[i] + D[i];

    Reference stream: A, B, C, D, A+4, B+4, C+4, D+4, A+8, B+8, C+8, D+8, etc.

12. The IBM POWER4 can detect up to eight regular access streams and issue prefetches for future references. Construct an address reference trace that will utilize all eight streams.

    Example code: for(i=0;i<10000;++i) sum += A[i] + B[i*2] + C[i] + D[i*2] +E[i] + F[i*2] + G[i] + H[i*2];

Reference stream: A, B, C, D, E, F, G, H, A+4, B+8, C+4, D+8, E+4, F+8, G+4, H+8, A+8, B+16, C+8, D+16, E+8, F+16, G+8, H+16, A+12, B+24, etc.

13. The stream prefetching of the POWER3 and POWER4 processors is done outside the processor core, using the physical addresses of cache lines that miss the L1 cache. Explain why large virtual memory page sizes can improve the efficiency of such a prefetch scheme.

    Once the prefetch stream reaches a page boundary (4KB for PowerPC), it has to stop, since there is no guarantee that consecutive virtual pages map to consecutive physical pages. However, if larger page sizes are supported (as they are in the POWER4), the prefetch streams can be much longer, since they only need to stop at the larger page boundaries.

14. Assume that a program is streaming sequentially through a 1 GB array by reading each aligned 8-byte floating-point double in the 1 GB array. Further assume that the prefetch engine will start prefetching after it has seen three consecutive cache line references that miss the L1 cache (i.e. the fourth cache line in a sequential stream will be prefetched). Assuming 4K page sizes, and given the cache line sizes for the POWER3 and POWER4, compute the overall miss rate for this program assuming no prefetching, assuming physical-address prefetching, and assuming virtual-address prefetching. Report the miss rate per L1 D cache reference, assuming that there is a single reference to every 8-byte word.

    The overall miss rate can be constructed as follows:

$$missrate = \frac{misses}{reference} = \frac{misses}{stream} \times \frac{stream}{blocks} \times \frac{block}{references}$$

    Note that all the units cancel out.

    The first term is 3 training misses / stream.

    For physical addressing, each stream is limited to the page size of 4K, so the second term is 4K/128 = 32 blocks per stream, or (1 stream) / (32 blocks)

    For virtual address prefetching, the stream encompasses the entire 1GB array, so the second term is 1GB/128 = 2^{23}, or (1 stream) / (2^{23} blocks).

    Both POWER3 and POWER4 have 128B cache lines, hence there are 128/8 = 16 FP doubles per cacheline, so the last term is (1 block) / (16 refs).

    Substituting into the original equation, we have:

    Physical addressing: 3 / 1 x 1 / 32 x 1 / 16 = 3 / 512 = 0.58% misses/reference

    Virtual addressing: 3 / 1 x 1 / 2^{23} x 1 / 16 = 3 / 2^{27} = 0.0000022 % misses/reference

15. Download and install the sim-outorder simulator from the Simplescalar simulator suite (available from www.simplescalar.com). Configure the simulator to match (as closely as possible) the microarchitecture of the PowerPC 620. Now collect branch prediction and cache hit data using the instructional benchmarks available from the Simplescalar web site. Compare your results to Table 6-3 and Table 6-10 and provide some reasons for differences you might observe.

    (solution not provided)