CIS501– Computer Architecture	Final Exam Solutions
Prof. Martin	Wednesday, Dec. 10, 2008

1. **[10 Points] True or False**. If a statement is "false," briefly explain how so by describing how the statement may most simply be made true. Unjustified (or poorly justified) "false" answers will be marked wrong. Simply stating the negation of the false statement is *not* sufficient justification. *Please be specific!*

   (a) Power-related issues are relevant only when designing battery-powered devices.

   Answer: False, cooling issues limit the power dissipation (and thus the performance) of server and desktop processors.

   (b) Smaller transistors generally use less energy when switching.

   Answer: True.

   (c) Reliability is an increasing concern as transistors continue to shrink in size.

   Answer: True.

   (d) For a cache of a fixed capacity, doubling the associativity of the cache will halve the size of the tag array.

   Answer: False. Associativity doesn't change the overall number of frames in the cache, and thus doesn't change the number or size of the tags significantly.

   (e) A victim buffer improves the average miss penalty of a cache.

   Answer: True.

   (f) A write buffer (also called a store buffer) is primarily intended to hide store latency.

   Answer: True.

   (g) Assuming similar pipeline depths, improving branch prediction accuracy is generally more important in a superscalar (multiple-issue) processor than in a scalar (single-issue) processor.

   Answer: True.

   (h) Because branch prediction is so critical for performance, a processor today typically uses a branch predictor that is 1 MB or larger.

   Answer: False, branch predictors are in the low kilobytes of state, not a thousand kilobytes.

   (i) A branch target buffer (BTB) is tagged because it must be correct; in contrast, a branch direction predictor is untagged because it is only a predictor and thus does not always need to be correct.

   Answer: False, both the BTB or direction predictor are just predictors, and thus neither needs to be correct.

   (j) When a cache block in the "E" cache coherence state is evicted from the cache, it must be written back to the memory.

   Answer: False, the "E" is a clean state, so it isn't written back. When a block in the "M" state is evicted, it is dirty and must update the memory.

2. **[5 Points]**

(a) **Processor Performance**. Assume a typical program has the following instruction type breakdown:

- 35% loads
- 10% stores
- 50% adds
- 3% multiplies
- 2% divides

Assume the current-generation processor has the following instruction latencies:

- loads: 4 cycles
- stores: 4 cycles
- adds: 2 cycles
- multiplies: 16 cycles
- divides: 50 cycles

If for the next-generation design you could pick one type of instruction to make twice as fast (half the latency), which instruction type would you pick? Why?

Answer:[2 points] First, we calculate the CPI adder of each type of instruction:
loads: 1.4
stores: 0.4
adds: 1.0
multiplies: 0.48
divides: 1.0
As loads contribute the most to the CPI, we should half the latency of load operations.

(b) **Caches**. You've been asked to determine the cache parameters that minimize average memory access latency for a next-generation flagship processor. The processor has a simple non-pipelined datapath in which all non-memory instructions take one cycle and all memory operations that hit in the cache take either two or three cycles (depending on the below cache configuration). After several experiments, you have narrowed down the caches to two choices:

i. A 64 KB cache which has a 2-cycle access time and a 90% hit rate

ii. A 128 KB cache which has a 3-cycle access time and a 95% hit rate

Under what condition would you select cache (*i*) over cache (*ii*)? Use a calculation to justify your answer.

Answer: [3 points] To determine the cross-over point, we find the point where t_{avg} is the same for both caches and solve for t_{miss}:
$t_{avg1} = t_{hit1} + miss\%_1 * t_{miss}$
$t_{avg2} = t_{hit2} + miss\%_2 * t_{miss}$
$t_{hit1} + miss\%_1 * t_{miss} = t_{hit2} + miss\%_2 * t_{miss}$
$2 + 0.10 * t_{miss} = 3 + 0.05 * t_{miss}$
$0.05 * t_{miss} = 1$
$t_{miss} = 20$ cycles
Thus, if the L1 miss latency is larger than 20 cycles, the 128KB cache is faster.

3. **[6 Points] Miscellaneous Short Answer**. Answer the following questions in just a few sentences.

(a) What is the primary trade-off between coarse-grained locking and fine-grained locking? That is, what is the primary advantage/disadvantage of one over the other?

Answer: [1 point] Fine-grained locking allows for more parallelism at the cost of higher programming difficulty.

(b) What are micro-operations (uops)? What problem do they attack?

Answer: [1 point] Micro-ops are used to break up complex instructions into multiple simpler instructions. CISC instructions are difficult to pipeline, but when broken up into micro-ops, the micro-ops can be executed on a RISC-like pipeline.

(c) If a single-cycle datapath with delay of n nanoseconds is converted into a pipelined datapath with p stages, the clock period will be longer (slower) than n/p nanoseconds. Give two reasons for this:

Answer: [2 points] (1) The overhead of the pipeline latches. (2) The logic is likely not evenly divided among all the pipeline stages (that is, not all pipeline stages will have the same delay).

(d) In a multiprocessor, increasing the cache block size can have an additional effect on performance. What causes this effect? Is it a positive or negative impact?

Answer: [2 points] False sharing becomes more of a problem as the block size increases. This has a negative impact on performance.

4. **[9 Points] Branch Prediction and Pipeline Depth**.

Branch prediction accuracy affects the choice of pipeline depth (i.e., number of pipeline stages). Consider the two simple single-issue pipelines below:

Pipeline A: A traditional five-stage pipeline with a single-cycle load-use penalty a two-cycle branch misprediction penalty at a clock frequency of 250Mhz (4ns clock cycle).

Pipeline B: A twelve-stage pipeline with a two-cycle load-use penalty and a six-cycle branch misprediction penalty at a clock frequency of 333Mhz (3ns clock cycle).

Consider a program that is 30% loads, 20% branches, and 50% other instructions. One third of loads have no dependent instructions within two instructions after it, one third of the loads have a dependent instruction immediately following the load, and the remaining one third of loads are followed by an independent instruction and then a dependent instruction.

(a) Give the cycles per instruction (CPI) and nanoseconds per instruction (NPI) for each of the two pipelines assuming a simplistic "predict not taken" that gives a prediction accuracy of 50% on this program.

Answer: [4 points]
Pipeline A: CPI $= 1 + (10\% * 2) + (10\% * 1) = 1.3$ CPI, CPI $*$ 4ns clock is 5.2 NPI
Pipeline B: CPI $= 1 + (10\% * 6) + (10\% * 2) + (10\% * 1) = 1.9$ CPI, CPI $*$ 3ns clock is 5.7 NPI

(b) At what branch prediction accuracy does the twelve-stage pipeline outperform the five-stage pipeline?

Answer: [4 points] Solve the equation for x:
$(1 + x * 20\% * 2 + 10\% * 1) * 4 = (1 + x * 20\% * 6 + 10\% * 2 + 10\% * 1) * 3$
$4 + x * 20\% * 8 + 10\% * 4 = 3 + x * 20\% * 18 + 10\% * 6 + 10\% * 3$
$4 + x * 20\% * 8 + 10\% * 4 = 3 + x * 20\% * 18 + 10\% * 6 + 10\% * 3$
$4.4 + 1.6x = 3.9 + 3.6x$
$0.5 = 2x$
$x = 25\%$
So, the prediction accuracy must be 75%.

(c) Based on typical branch prediction accuracies, which pipeline would you choose for best performance?

Answer: [1 points] As branch prediction rates are often over 90%, the 12-stage pipeline is likely performs best on average.

5. **[6 Points] Storage I**.

Consider a disk with rotational speed of 6000 RPM (which is 100 rotations per second or one full rotation every 10ms), an average seek time of 10ms (with zero seek time to an adjacent track), and negligible controller overhead. Each track on the disk has 1024 sectors and each sector is 1024 bytes.

(a) Calculate the maximum read throughput of the disk for sequential reads.

Answer: [2 points] The disk spins 100 times per second, and each track has 1MB of data (1024*1024), thus the disk has a maximum read throughput of 100MB/second.

(b) Now consider random reads for the same disk. How large must the chunks of data read from the disk be to ensure a non-sequential read throughput of 14.25 MB/second?

Answer:[4 points] Let S be the size of the non-sequential block. If the average latency in milliseconds to fetch the block is t_{avg}, then $14.6 = S/t_{avg}$ or $t_{avg} = S/14.6$.
Furthermore,
$t_{avg} = t_{seek} + t_{rotate} + t_{transfer}$
$t_{avg} = 10ms + 5ms + 10ms * (S/1024)$
$t_{avg} = 15ms + 10ms * (S/1024)$
Then, we solve for S $S/14.6 = 15ms + 10ms * (S/1024)$
$S = 14.6 * 15ms + 14.6 * 10ms * (S/1024)$
$S = 14.6 * 15ms + 14.6 * 10ms * (S/1024)$
$S = 219 + 0.1425 * S$
$0.857 * S = 219$
$S = 256KB$
We can check the answer doing a straightforward calculation:
$t_{avg} = t_{seek} + t_{rotate} + t_{transfer}$
$t_{avg} = 10ms + 5ms + 10ms * (S/1024)$
$t_{avg} = 10ms + 5ms + 10ms * (256/1024)$
$t_{avg} = 10ms + 5ms + 2.5$
$t_{avg} = 17.5ms$
The total throughput in KBs/millisecond is $S/t_{avg} = 256/17.5 = 14.6KB/ms$. Which is 14.25 MB/second.

6. **[8 Points] Storage II**.

You've been asked to design a storage subsystem that has a total capacity of 5 TB (1 TB is 1024 GBs) and that can support 30,000 random reads per second.

(a) If each disk costs $100, can support 100 non-sequential reads per second, and has a capacity of 1TB. How much would such a storage subsystem cost?

Answer: [2 points] To meet capacity, only five disks are needed. But to meet throughput, 300 disks are needed. Thus, 300 disks at $100 are needed, thus the disk subsystem costs $30,000.

(b) Flash memory is non-volatile storage without any moving parts, thus it avoids all seek and rotation latencies to provide a dramatic increase in throughput for non-sequential reads. On October 15th, Intel released the X25-E flash-based solid-state storage device that uses a standard disk interface, has a capacity of 64GBs, can perform 35,000 non-sequential reads per second, and costs around $1000. Given the same storage system requirements as above, how much would such a storage system cost if it was comprised entirely of these X25-E storage devices?

Answer: [2 points] Even though a single drive would provide enough throughput, to meet the capacity requirements, 80 of these drives are needed. Thus, 80 disks at $1000 is $80,000.

(c) How might you build a storage subsystem using a combination of the disk drives and solid-state drives described above?

Answer: [1 point] Use disk drives to fulfil the capacity needs, while using solid-state drives as a disk cache for providing sufficient bandwidth.

(d) Would your implementation of this hybrid storage system use extra hardware? Change the software? Both? Neither? Justify your answer.

Answer: [1 point] Answers will vary, but I would just change the software. In fact, Sun Microsystem's ZFS file system (part of Solaris) already had such hybrid disk support built in, as pointed out in Intel's press release for the X25-E solid state drive.

(e) Assuming the best case, how much might such a hybrid storage subsystem cost?

Answer: [1 point] Five disk drives would be enough for capacity, and one solid-state drive would be enough for throughput. So, assuming the workload has sufficient temporal locality, the total system cost would be five $100 disks ($500) plus a single $1000 solid state disk, for a total of $1500. This system consumes substantially less money, less space, and less power than either of the above systems.
For this to work, only 500 of the 30,000 reads per second could "miss" in the solid-state drive, thus requiring a hit rate of 98%.

(f) What property would need to be present in the workload for such a system to work?

Answer: [1 point] Temporal locality

7. **[8 Points] Static and Dynamic Scheduling**.

(a) What are the three primary impediments that limit the compiler's ability to perform effective static scheduling?

Answer: [3 Points] limited scheduling scope, limited number of architectural registers, and imprecise memory aliasing information.

(b) What techniques do dynamically scheduled processors use to overcome each of these three limitations?

Answer: [3 Points] (1) branch prediction and speculative execution, (2) register renaming with a larger physical register file, and (3) memory dependence speculation and hardware to detect memory dependency violations.

(c) Although today's mainstream laptop and desktop processors employ dynamic scheduling and wide-issue superscalar (four or more instructions per cycle) execution. In contrast, some newer chips in other domains do not. Give two reasons we're seeing a resurgence in-order processor designs and more modest superscalar execution (for example, only two instructions per cycle)?

Answer: [2 Points] (1) a wide-issue superscalar with dynamic scheduling is more power hungry and (2) the extra transistors needed to implement the larger core are instead being used to place more cores on a chip (a larger number of simpler cores versus a smaller number of more sophisticated cores).

8. **[11 Points] Superscalar versus Vectors**. Superscalar execution and vector instructions are two ways of increasing the amount of computation a processor can perform each cycle. Consider adding these features to a 64-bit pipelined processor.

(a) What impact–*if any*–would adding **superscalar execution** (for example, the ability to perform four instructions per cycle) have on the following six aspects of the design:

Answer: [6 Points]

- Instruction fetch: Need to fetch more instructions (and likely improve the branch prediction)
- Stall logic: N^2 checking. For each instruction need to check the previous instructions in the cycle (and all the loads in the prior cycle) during the dependence check.
- Bypassing: N^2 bypassing. Need to forward any of N output values to any of 2N input values.
- Register file: Need 2N read ports and N write ports.
- Execution units: Need N copies of the various execution units.
- Data cache: To support more than one memory operation per cycle, the data cache would need to have multiple ports.

(b) What impact–*if any*–would adding **vectors** (for example, using four-element vectors to perform four 64-bit operations in parallel for each vector instruction) have on the following six aspects of the design:

Answer: [3 Points]

- Instruction fetch: none
- Stall logic: none
- Bypassing: Wider datapaths (more muxes, but not more inputs to the bypass muxes)
- Register file: Wider register read/write ports (much cheaper than more ports)
- Execution units: A wider execution unit
- Data cache: Wider data cache read and write. As with the register file, wider is cheaper than more independent ports. (For example, a multiported cache requires multiple tag matches, a wider aligned access does not.)

(c) Apart from the above hardware implementation differences, what are the two most important advantages of superscalar execution over vector operations:

Answer: [2 Points]

i. Superscalar can extract parallelism from unmodified binaries and even more with some compiler static scheduling; in contrast, vectors require a sophisticated vectorizing compilers or—more likely—programmer-inserted annotations.

ii. Not all programs have nicely data parallel constructs that map nicely to vector instructions. Whereas superscalar (especially with dynamic scheduling) can transparently extract at least some instruction-level parallelism from almost all programs.

9. **[8 Points] Multithreading**. Consider a video encoding computation running on an in-order dual-issue superscalar processor. You have two video files to encode, each taking 10 minutes to execute. Thus, encoding both files back-to-back on this simple processor will take 20 minutes.

This computation spends 30% of its cycles executing two instructions, 50% executing one instruction, and 20% of its cycles executing zero instructions because of a cache miss. The cache miss latency is 20 cycles.

Consider the change to the runtime when running these two computations at the same time on various multithreaded version of the above two-way superscalar processor.

(a) Calculate a reasonable best-case estimate of the combined runtime on a coarse-grained multithreaded processor (which switches threads on a cache miss) with a thread-switch time of 10 cycles.

Answer: 18 minutes. Half of the miss latency can be hidden by stitching to a thread (10 is half of 20), so the minutes spend in cache missing decreases from 4 minutes to two minutes.

(b) Calculate a reasonable best-case estimate of the combined runtime on a fine-grained multithreaded processor (which can switch threads every cycle without penalty).

Answer: 16 minutes. Without a miss penalty, all of the miss latency can be hidden at best.

(c) Calculate a reasonable best-case estimate of the combined runtime on a simultaneously multithreaded processor.

Answer: 11 minutes. All the memory latency is hidden, but both tasks spend 3 minutes executing at full utilization (6 minute total). Assuming SMT allows us to combine the remaining 5 minutes each to be just 5 minutes total, that give us $3 + 3 + 5 = 11$ minutes.

Now consider a different implementation of the computation. This computation spends 10% of its cycles executing two instructions, 30% executing one instruction, and 60% of its cycles executing zero instructions because of a cache miss.

(a) Calculate a reasonable best-case estimate of the combined runtime on a coarse-grained multithreaded processor (which switches threads on a cache miss) with a thread-switch time of 10 cycles.

Answer: 14 minutes. Same reasoning as above.

(b) Calculate a reasonable best-case estimate of the combined runtime on a fine-grained multithreaded processor (which can switch threads every cycle without penalty).

Answer: 10 minutes. Using the reasoning above, we would get a value less than 10 minutes. However, that isn't possible, as any single task would take 10 minutes. The problem is that there isn't enough work to totally hide the memory latency.

(c) Calculate a reasonable best-case estimate of the combined runtime on a simultaneously multithreaded processor.

Answer: 10 minutes. Same reasoning as above.

(d) Give two general reasons these simple calculations are optimistic estimates of performance? That is, what two assumptions did you make in the above calculations?

Answer: First, multithreaded processors share resources internally (such as caches, branch predictors, TLBs), so there is likely some performance penalty to running multiple threads (more cache misses, branch mis-predictions, TLB misses, etc.) not captured in the above estimates. Second, the above estimates assumed an even distribution of events such as cache misses. In reality, cache misses tend to happen in bursts (a region of frequent cache misses followed by a phase of fewer cache misses). These bursts will increase the likelihood that both threads are stalled on a long-latency cache miss, reducing overall processor utilization.

10. **[8 Points] Cross-Cutting ISA Issues**.

(a) Most ISAs have distinct "integer" and "floating point" register files. Give an advantage and a disadvantage of having separate register files versus a single register file for both types of data.

Answer: [3 points] Advantages: One advantage is that separating the types of operations makes it easier to implement superscalar execution (for example, the Alpha 21164's two-way integer and two-way floating point pipeline). Separate register files also allows more data to be held in registers without increasing register file access latency or the number of bits of register specifies (the type of operation determines which register file to access).
Disadvantages: One disadvantage is that any resources and power consumed by the floating point circuity provides no advantage to programs without floating point operations. Another disadvantage is that you need additional instructions to store/load data from the floating point registers.

(b) Throughout the semester, we discussed several ISA extensions. What new instruction(s) have been added for each of the following:

- To reduce branch mispredictions?
  Answer: Prediction (either full support for predication or just conditional move instructions)
- To reduce cache misses?
  Answer: Software prefetching instructions
- To express data-level parallelism?
  Answer: Vector/SIMD instructions
- To facilitate shared-memory multiprocessing?
  Answer: Atomic swap or "test and set" instructions

(c) What aspect of an ISA has the most impact when implementing multithreading? Why?

Answer: The number of register in the ISA, because these architectural registers are replicated in multithreaded cores (which leads to a very large and slow register file).

11. **[11 Points] Parallelism At All Levels**.

Throughout the semester we've explored parallelism at many different levels: beginning with parallelism at the level of instructions (or phases of instructions) to multiple processors cores on a chip.

Based on what we've discussed this semester, give an example use of parallelism at at least five different levels of granularity. For each of them also describe both: (1) the specific reasons for (or benefits of) employing parallelism, and (2) any disadvantages or challenges of exploiting parallelism at that level of granularity.

Answer:

(a) Pipelining (multiple phases of instructions) - greatly increases the throughput of the datapath. The disadvantages are that (1) it increases the complexity of the design (for example, to handle stalls) and (2) it has diminishing benefits because of branch mispredictions and other pipeline stalls.

(b) Superscalar (multiple independent instructions in the same cycle) - up to a point, can multiply the performance of a processor. The advantage is that can be done transparently to the programmer, but it does require sophisticated hardware implementation (and it can reduce clock frequency).

(c) Multithreading (instructions of different threads executing concurrently) - multithreading increases the utilization of a pipeline that would otherwise be idle. It adds some complexity, and it doesn't give much advantage past just a few threads per core.

(d) Multiprocessors (multiple pipelines all executing concurrently) - adding more processor cores adds many more transistors, but has the potential to linearly increase performance. Its main disadvantages are (1) that it requires programmers to write parallel software and (2) few program have enough inherent parallelism to achieve linear speedup on large number of cores.

(e) Vectors/SIMD (one instruction that performs the same arithmetic operation on multiple data elements in parallel) - Vectors can increase performance, but they require programmer-inserted annotations.

As an aside, all of these forms of parallelism meet with diminishing returns beyond some point. That is why all of these forms of parallelism are employed, as exploiting any *one* of them in the extreme isn't as good as using them all to a moderate degree.

Give an example of a system that exploits all of these forms of parallelism:

Answer: The XBox 360