**CIS501– Computer Architecture**

**Prof. Martin**

**Midterm Exam Solutions**

**Thursday, Oct. 29, 2009**

1. **[ 11 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) *Computer architecture* involves using computers to design buildings.

   **Answer:** False, computer architecture involves designing computers.

   (b) A typical high-end chip today has approximately 100 million transistors.

   **Answer:** False. High-end chips today have around a billion transistors.

   (c) Decreasing the gate "length" and increasing the "width" of transistor both reduce resistance.

   **Answer:** True

   (d) Each process generation transition (say, 45nm to 32nm) doubles transitor density.

   **Answer:** True

   (e) A 1Ghz processor takes longer to execute a program than a 2Ghz processor.

   **Answer:** False, without the instruction per cycle (IPC) of the processor, we can't reason about their relative performance.

   (f) The x86 ISA is an example of a CISC ISA.

   **Answer:** True

   (g) Both the CPI and the clock frequency of a processor vary from program to program.

   **Answer:** False. The clock frequency is generally independent of the specific program being executed, but the CPI does vary.

   (h) Performance metrics based on instructions per second (for example, MIPS), is a reasonable metric to compare the performance of two processors with the **same ISA** and the **same compiler**.

   **Answer:** True

   (i) With optimal buffer insertion, the delay of a wire is quadratic in its length.

   **Answer:** False, buffer insertion allows wire delay to be linear in the length of the wire.

   (j) A victim cache is accessed in parallel with the main data cache, and thus increases the hit latency.

   **Answer:** False, a victim cache is accessed on the miss path, explicitly so that it doesn't impact the hit time of the first level cache.

   (k) The primary advantage of a multi-level page table is faster address translation.

   **Answer:** False, multi-level page tables reduce the size of the page table.

2. **[ 11 Points ]** **Processor Performance**

   (a) 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:[3 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) Assume that a program executes one branch ever 4 instructions for the first 1M instructions and a branch every 8 instructions for the next 2M instructions. What is the average number instructions per branch for the entire program?

**Answer: [3 points]** Average Instructions per Branch = (Total Instructions) / (Total Branches) = (1M + 2M) / (1M/4 + 2M/8) = 6 instructions per branch.)

(c) You are the architect of a new line of processors, and you have the choice to add new instructions to the ISA if you wish. You consider adding a fused multiply/add instruction to the ISA (which is helpful in many signal processing and image processing applications). The advantage of this new instruction is that it can reduce the overall number of instructions in the program. The disadvantage is that its implementation requires extending the clock period by 10% and increases the CPI by 15%. Calculate under what conditions would adding this instruction lead to an overall performance increase.

**Answer: [3 points]** The runtime of a program is: (instruction/program) * (cycles/instruction) * (seconds/cycle). An unmodified program will take 1.10 * 1.15 = 1.265 as long. Thus, the new instruction count must be (1/1.265) = 0.79 times the original instruction count to make up for the increase in clock period and CPI. Or, a 21% decrease in the number of instructions. So, if 42% or more of instructions in the original program were multiply/add pairs, a speedup would result.

(d) What qualitative impact would this modification have on the overall MIPS (millions of instructions per second) of the processor? What does this say about using MIPS as a performance metric?

**Answer: [2 point]** MIPS will always decline (in this case by 21%), because both the CPI and clock frequency are getting worse. As this change could either be a performance improvement or performance loss (depending on the application) it highlights that MIPS is a misleading metric when the number or mix of instructions is changing.

3. **[ 8 Points ] Short Answer**

(a) What are two important hardware features present in today's microprocessors that are *unrelated* to performance, cost, or energy consumption.

**Answer:** (1) Virtual memory, (2) reliability features such as error correction codes, (3) compatibility, and (4) physical size (form factor).

(b) Consider two different computing applications: a mobile wireless internet device and a server for processing transactions for a large bank. What are three design considerations that differentiates the design of a computer for these two application domains.

**Answer: [3 points]** The biggest difference is the cost constraints; a mobile device must be cheap (a few hundered dollars) whereas a server for a bank is less cost sensitive. Another difference is the high reliability demands of the bank; if the bank's computer breaks (or silently calculates the wrong answer), millions of dollars are at stake. The remaining issue could be physical size (the mobile device must be highly integrated) or power efficiency for battery life of the mobile device. Power efficiency is also important in the large data centers, but probably not as critical for a single server handling transactions for a bank.

(c) Give two different reasons why increasing the die (chip) size of a microprocessor increases the cost of the microprocessor. Recall that die (chips) are tiled on a wafer for fabrication.

**Answer: [2 points]** 1. Larger die means fewer die per wafer (and much of the cost is per wafer). 2. Larger die have less chance of being defect-free (given random defects across the wafer).

(d) What is the largest speedup that can be achieved by optimizing something that represents 80% of a program's execution time.

**Answer: [1 point]** In the limit, only the 20% will remain. Going from 100% to just 20% is a 5x speedup.

4. **[ 7 Points ] Energy**

(a) What are three hardware approaches for reducing the **dynamic energy** to perform a given computation:

**Answer:** Reducing the voltage, reducing the number of transistors, reduce the activity via clock gating, reduce the capacitance of each transistor, etc.

(b) What are three hardware approaches for reducing the **static energy** to perform a given computation:

**Answer:** Reduce the number of transistors, turn off transistors via power gating, reduce leakage by adjusting the threshold voltage, dual Vt, low-leakage transistors (metal gates and high-k), etc. The answer "reduce voltage" isn't right for this one, as reducing the voltage would also cause the frequency to decrease, thus keeping energy unchanged.

(c) What can be done in software to reduce energy consumption?

**Answer:** The same sort of optimizations that improve performance (better algorithms, compiler optimisations, etc.) will also reduce energy consumption for the computation. The operating system can also control the frequency and voltage of the system so that it runs the chip at an energy-efficient mode.

5. **[ 9 Points ] Stream Buffers**

(a) What specific performance problem do stream buffers target?

**Answer:** Stream buffers prefetch data to reduce capacity misses and to some extent compulsory misses.

(b) Why does each stream buffer have multiple entries?

**Answer:** To keep ahead of the processor, the stream buffer must fetch multiple blocks into the stream.

(c) Would stream buffers in today's systems likely have more entries or fewer entries than when they were first proposed? Why?

**Answer:** As memory latency is longer today than when the paper was written, it is likely the stream buffers are even deeper (more entries) today. Other reasonable answers, if well justified, are also acceptable. Some students said the buffers would be small, as other more advanced prefetchers would also be employed, for which we also gave full credit.

(d) What is the benefit of multi-way stream buffers?

**Answer:** For instructions and data; for accessing multiple arrays. Consider memory copy or a vector-vector addition, both of which would benefit from multiple stream buffers.

(e) As proposed, stream buffers do not place data directly into the cache. What is one advantage and two disadvantages of such a choice?

**Answer: [3 points]** Advantage: doesn't pollute the cache with prefetched data, it also doesn't take up cache bandwidth or otherwise slow down the performance-critical first-level caches. Disadvantages: takes up extra area to have dedicated buffers (rather than just sharing the cache), requires extra latecy to access (if the data was in the cache, it would have lower latency, as it would just be the hit latency). Having dedicate buffers is perhaps a bit more complicated, so that is another potential answer.

(f) What is the most significant potential *negative* performance impact caused by stream buffers?

**Answer:** It increases traffic to the second-level caches and memory. These additional requests can slow down demand misses (non-prefetch misses), thus slowing down the program.

(g) How might the presence of stream buffers impact the choice of cache block size?

**Answer:** Stream buffers likely capture much of the spatial locality of large block sizes, but stream buffers don't have the negative pollution and write-back traffic impacts of larger block sizes. Thus, it would likely push designers to smaller block sizes than otherwise.

## 6. [ 8 Points ] Cache Traffic

(a) How does increasing a cache's block size typically impact (increase, decrease, or unchanged) its fill traffic (measured in bytes)?

**Answer:** Increase. (Large blocks bring in more data each miss, so even though it may improve the miss rate, its improvements to the miss rate are generally not enough to offset the additional traffic.)

(b) How does increasing a **write-back** cache's block size typically impact (increase, decrease, or unchanged) its write traffic (measured in bytes)?

**Answer:** Increase. Larger blocks means that the entire block must be written back, even if only one byte of it was written.

(c) How does increasing a **write-through** cache's block size typically impact (increase, decrease, or unchanged) its write traffic (measured in bytes)?

**Answer:** Unchanged. In a write-through cache, the amount of write traffic is unaffected by the cache (all stores generate write traffic).

In the cache simulation assignment, a block in a write-back cache had a single dirty bit. Consider a new design for a write-back cache that has a dirty bit for each byte in the cache. The corresponding dirty bit is set whenever that byte is written. Upon replacement, only the dirty bytes are written back to the second-level cache.

(a) What is the main advantage of such a cache? Specifically, how does it compare to traditional write-back and write-through caches?

**Answer:** The dirty bits avoid the inefficiency of writing back bytes to the second-level cache that weren't actually changed. Instead, only the bytes actually written will create write traffic (making it less than or equal to a write-through cache). When compared to a write-back cache, it will never be worst, because the worst case is that even dirty bit is set, in which case it just writes back the whole block (which is what a write-back cache would do).
Thus the traffic will always be equal to or lower than the smaller of the write-though cache's traffic or the write-back cache's traffic. It will often be smaller than both.

(b) How might such a cache effect the choice of block size?

**Answer:** The extra write traffic cost of using larger block size goes away, so larger block sizes might be more attractive.

(c) Give two disadvantages of such a cache design:

**Answer:** (1) The extra chip area overhead for storing the per-byte valid bits, and (2) the extra design complexity to select just the dirty bytes and write them back to the next level.

7. **[ 11 Points ] Virtual Memory Mechanics**. Consider a system with a 28-bit virtual address space, a 20-bit physical address space, and a 4KB (4096 byte) page size.

(a) Number of bits in the page offset:

**Answer:**4096 is $2^{12}$, so 12 bits

(b) Number of bits in the virtual page number:

**Answer:**28 minus 12 is 16 bits

(c) Number of bits in the physical page number:

**Answer:**20 minus 12 is 8 bits

(d) Assuming a simple single-level page table, how large is the page table (in bytes)? (Ignore any valid bits, permission bits, etc.)

**Answer: [2 points]** One entry per VPN ($2^{16}$) and each entry is approximately 1 byte. Thus, $2^{16}$ bytes which is 64 KB.

(e) If the processor incorporated a 256-entry **fully-associative** TLB, how large (in bytes of storage) would it be?

**Answer:** Each entry maps a VPN (2 bytes) to a PPN (1 byte), so each entry is 3 bytes. 256*3 is 768 bytes.

(f) If the processor incorporated a 256-entry **direct-mapped** TLB, how large (in bytes of storage) would it be?

**Answer:** The TLB is indexed by the VPN. As there are 256 entries, 8 bits are used for index, leaving 8 bits of tag. The PPN is the same as before (8 bits) so the total is 512 bytes.

(g) Consider a two-level page table. Each level is indexed using an equal number of bits. How large is each second-level page table allocation?

**Answer: [2 points]** Each level is indexed with $2^8$ bits, so each second-level page table has 256 entries and each entry is 1 byte. So each second-level allocation is 256 bytes.

(h) Consider a program that accesses the virtual addresses listed below (in binary).

```
0000 0000 0000 0000 0000 0000 0001
0000 0000 0000 0000 0000 0001 0000
0000 0000 0000 0000 0001 0000 0000
0000 0000 0000 0001 0000 0000 0000
0000 0000 0001 0000 0000 0000 0000
0000 0001 0000 0000 0000 0000 0000
0001 0000 0000 0000 0000 0000 0000
```

How many physical pages would the OS allocate to the program?

**Answer:** Looking only at the VPN bits, we see five different VPNs, thus five pages were allocated (for a total of 20 KBs). To determine the number of second-level page table entries, we want to look for the number of unique 8-bit VPN prefixes. There are three unique 8-bit prefixes, so there are three second-level page table entries (for a total of 768 bytes).

How much memory would the OS allocate for the second-level of the program's page table?

**Answer:**See above.

8. **[ 10 Points ] Cache Conflicts**. This problem explores different ways to eliminate cache conflicts. Consider a processor with 16-bit addresses executing a program that repeatedly accesses the addresses given below (say, in a loop). The addresses are in binary with the most significant bit first, and the list is the same for all parts of the question.

(a) **Larger Capacity.** For a direct-mapped cache with 256B cache blocks, what is the smallest cache size sufficient to avoid repeated conflict misses for the above accesses?

```
0100 1000 0100 0000
0001 0110 0011 0000
0010 0110 0010 0000
0001 0110 0001 0000
```

**Answer:** The index bits must be unique (ignore offset and tag), so increasing the cache size increases the number of index bits. Once there are 6 index bits, so the total cache size must be $2^5$ multiplied by the block size (256B) for a total of 8KB.

```
tag index offset
010 01000 01000000
000 10110 00110000
001 00110 00100000
```

(b) **Smaller Block Size.** For direct-mapped 1KB cache, what is the largest block size that is sufficient to avoid conflicts?

```
0100 1000 0100 0000
0001 0110 0011 0000
0010 0110 0010 0000
0001 0110 0001 0000
```

**Answer:** The index bits must be unique (ignoring offset and tag), and decreasing the block size increases the number of index bits. 16-byte blocks ($2^4$) makes the index unique.

```
tag    index  offset
010010 000100 0000
000101 100011 0000
001001 100010 0000
000101 100001 0000
```

(c) **Higher Associativity.** For a 1KB cache with 128B block size, what is the lowest associativity sufficient to avoid conflicts?

```
0100 1000 0100 0000
0001 0110 0011 0000
0010 0110 0010 0000
0001 0110 0001 0000
```

**Answer:** First we notice that two of the accesses are to the same block (and thus can be combined into a single entry). Of the remaining three addresses, a direct mapped cache is insufficient because more then one of the three unique addresses have the same index.
If we then consider two-way set associative, we must reduce the number of index bits to two. In this case, the three different blocks still all have the same index. Thus, a four-way set associative cache is required.
Direct mapped:

```
tag    ind offset
010010 000 1000000
000101 100 0110000
001001 100 0100000
```

Two-way set associative:

```
tag    ind offset
0100100 00 1000000
0001011 00 0110000
0010011 00 0100000
```

(d) **Higher Associativity with Way Prediction.** For a 4-way set-associative 1KB cache with a 64B block size, what is the smallest way predictor (in bytes) that is guaranteed to avoid repeated way mis-predictions?

```
0100 1000 0100 0000
```

```
0001 0110 0011 0000
0010 0110 0010 0000
0001 0110 0001 0000
```

**Answer:** The way predictor is indexed with the index bits plus the low-order tag bits. Seven bits are needed to make this unique (the way predictor is direct mapped), so 128 2-bit entries are needed which is 32 bytes of state. Note: the number of entries in the way predictor is the same number of entries required by the direct-mapped cache (for the same reasons).

```
    /-pred-\
/--tag--\ index  offset
010 01000 01     000000
000 10110 00     110000
001 00110 00     100000
```

(e) **Virtual to Physical Page Mapping.** Consider a system with 16-bit physical addresses, 16-bit virtual addresses, 256-byte pages, and a 0.5KB (512-byte) direct-mapped cache with 64-byte blocks. The given addresses are virtual addresses that are translated to physical addresses before each cache access. The mapping chosen by the OS is given below (right). Complete the missing mapping to avoid cache conflicts.

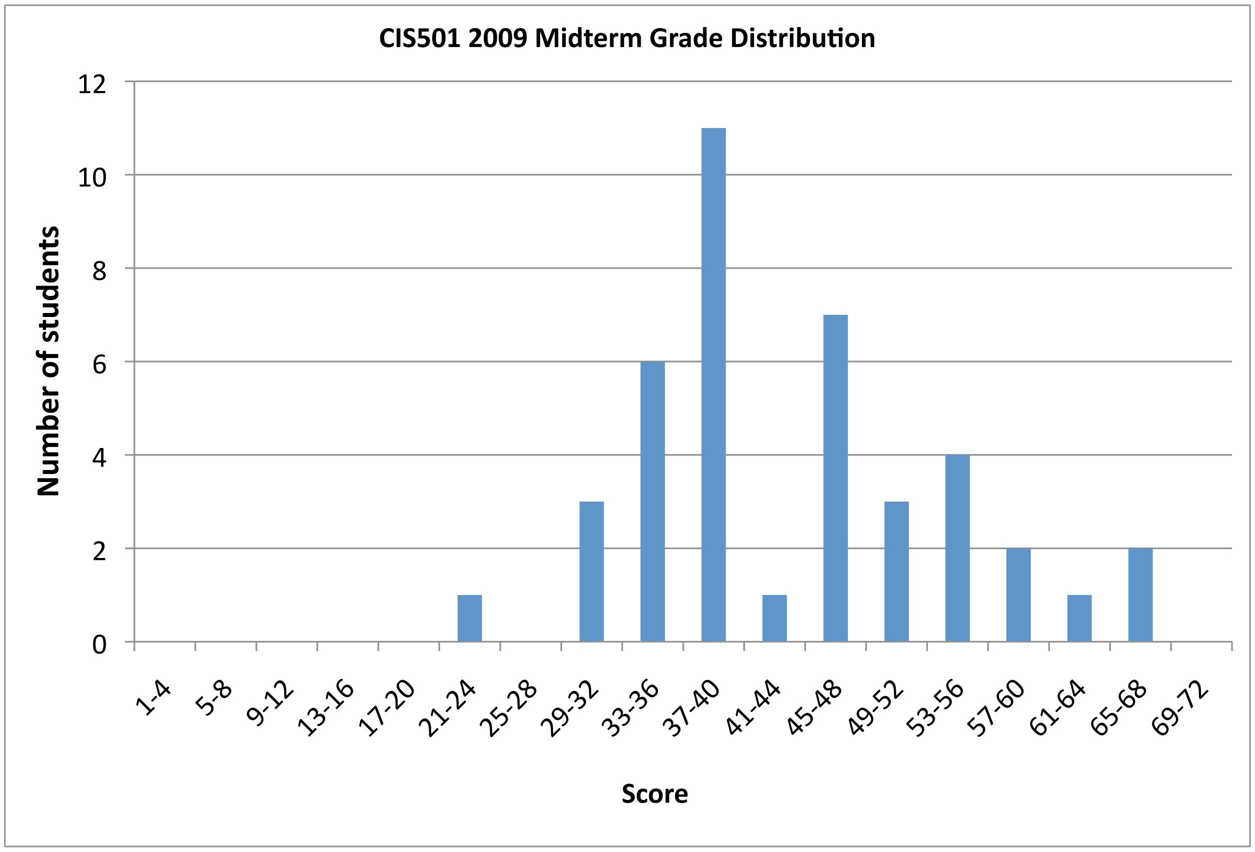
```
0100 1000 0100 0000             /- VPN -\    /- PPN -\
0001 0110 0011 0000             01001000  -> 00001001
0010 0110 0010 0000             00010111  -> 00000101
0001 0110 0001 0000             00100111  ->
```

**Answer:** The only bit that matters is the last bit of the PPN (as that is the only bit the impacts the index). To ensure all three addresses have different indexes after mapping, the missing mapping can be anything as long as its last bit is zero. If it ended in a 1, two of the addresses would have the same index (100).

```
/- VPN -\ /- page -\     /- PPN -\ /- page -\
tag     index offset     tag     index offset
0100100 0 01  000000  -> 0100100 1 01  000000
0001011 0 00  110000  -> 0001011 1 00  110000
0010011 0 00  100000  -> 0010011 0 00  100000
```

End of exam

## Distribution



Mean: 44 points (59%) — Median: 40 points (53%) — High: 65 (87%)