CSC506 Homework due Friday, 5/28/99 - Cache questions & VM

Question 1. What advantage does a Harvard cache have over a unified cache?

- A Harvard (split) cache permits the processor to access both an instruction word and a data word in a single cache memory cycle. This can significantly increase the performance of the processor. The only real drawback of a Harvard cache is that the processor needs two memory busses (two complete sets of memory address and data lines), one to each of the caches. However, when the Harvard cache is implemented within the processor chip, the separate memory busses never have to escape the chip, being implemented completely in the metal interconnections on the chip.

Question 2. Why would you want the system I/O to go directly to main memory rather than through the processor cache?

- If the system I/O went through the processor cache, it would cause contention with the processor trying to use the cache. The processor-to-cache bus is almost 100% utilized in most implementations, and any time the system I/O tried to use it the processor would have to wait. On the other hand, the main memory bus in a well-designed uniprocessor system should have a much lower utilization. With a cache hit rate of 97% and a 10 to 1 difference in main memory access to cache access times, the main memory bus is about 30% utilized due to the processor.

Question 3. How long does it take to access any random word from Synchronous DRAM that is rated at 10 ns?

- Since we are not given any information on the number of cycles from RAS to the first data out, we don't know. SDRAM is rated at the maximum clock rate to retrieve sequential bits ***after*** accessing the first random bit. You have to go look at the detail specifications to determine other parameters.

Question 4. If an 8-way set-associative cache is made up of 32 bit words, 4 words per line and 4096 sets, how big is the cache in bytes?

- We convert words/line to bytes/line = 4 bytes/word x 4 words/line = 16 bytes/line. Cache size is LKN = 16 x 8 x 4096 = 512k bytes.

Question 5. What is the shortest time it would take to load a complete line in the above cache using fast page mode DRAM that has a RAS access time of 50 ns, a CAS access time of 13 ns, a cycle time of 95 ns and a fast page mode cycle time of 35 ns?

- We achieve the fastest time by using the fast page mode cycle to retrieve subsequent words after the first, but we need a full RAS access to get the first word. A cache line is 4 words, so the minimum time would be 50 + 35 + 35 + 35 = 155 ns.

Question 6. What is the shortest time it would take to load a complete line in the above cache using EDO DRAM that has a RAS access time of 50 ns, a CAS access time of 13 ns, a cycle time of 84 ns and an EDO cycle time of 20 ns?

- We achieve the fastest time by using the EDO (hyper page mode) cycle to retrieve subsequent words after the first, but we need a full RAS access to get the first word. A cache line is 4 words, so the minimum time would be 50 + 20 + 20 + 20 = 110 ns.

Question 7. What is the shortest time it would take to load a complete line in the above cache using synchronous DRAM that requires 5 clock cycles from RAS to the first data out and is clocked at 100 MHz?

- We achieve the fastest time by using synchronous access to subsequent SDRAM bits to retrieve subsequent words after the first, but we need a full RAS access to get the first word. A cache line is 4 words, so the minimum time would be 5 clocks for the first word plus 1 clock per subsequent word. Clocked at 100 MHz, we get 50 + 10 + 10 + 10 = 80 ns.

Question 8. If the memory bus speed is 50 MHz, which of the above three DRAMs would you use and why?

- Here, we need to consider that real memory systems are designed using a fixed clock rate for the memory bus. Even though a memory part can respond in a certain time, we can't use the data until the next clock cycle after the memory part makes the data available. With a bus speed of 50 MHz, the clock period is 20 ns, so we need to line up our accesses on 20 ns ticks.

- The fast page mode DRAM can provide data from RAS in 50 ns, but we can use it only after 3 clock periods of 20 ns each, or a total of 60 ns. Subsequent to the first word, it can supply data in 35 ns increments, but we can use it only in 20 ns increments, so we have to use 2 clock periods for each subsequent word. So, the total time to fill a cache line is 60 + 40 + 40 + 40 = 180 ns.

- The EDO RAM requires 50 ns from RAS to the first word and can supply subsequent data in 20 ns cycles. The 20 ns cycle just matches our 20 ns bus clock, so the total time to fill the cache is 60 + 20 + 20 + 20 = 120 ns.

- The SDRAM is synchronously clocked already, so we just use the spec of 5 cycles to first data out and 1 cycle per subsequent word, or (5 x 20) + 20 + 20 + 20 = 160 ns.

- For this bus speed, I would use the EDO RAM because it is faster and probably cheaper than the SDRAM. In actual practice, the SDRAM would probably have a different mode that would allow only 3 cycles to first data out when clocked at this low rate, so the EDO and SDRAM would give equal performance. The real advantage of this SDRAM is for a memory bus clock faster than 50 MHz. The SDRAM can deliver data every clock cycle on bus speeds up to 100 MHz. The EDO RAM can't do that on bus speeds greater than 50 MHz.

Question 9. If a memory system consists of a single external cache with an access time of 20 ns and a hit rate of 0.92, and a main memory with an access time of 60 ns, what is the effective memory access time of this system?

- t(eff) = 20 + (0.08)(60) = 24.8 ns

Question 10. We now add virtual memory to the system described in question 9. The TLB is implemented internal to the processor chip and takes 2 ns to do a translation on a TLB hit. The TLB hit ratio is 98%, the segment table hit ratio is 100% and the page table hit ratio is 50%. What is the effective memory access time of the system with virtual memory?

- $t_{eff} = t_{TLB} + (1 - h_{TLB})(t_{SEG} + t_{PAGE}) + t_{CACHE} + (1 - h_{CACHE})t_{MAIN}$
- $t_{eff} = 2 + 0.02(20 + 20 + 0.5(60)) + 20 + (0.08)(60) = 28.2$ ns
- This represents a drop in performance of (28.2 – 24.8)/24.8 = 14%.

Question 11. LRU is almost universally used as the cache replacement policy. Why?

- It is easy to implement and has been proven to be the best predictor of future use behind the OPT algorithm. However, OPT is not achievable, requiring perfect knowledge of future use.

Question 12. In an *n*-way set-associative cache, it is preferable to start a read to all lines in a set in parallel, even though only one line, at most, will be used. Why is it reasonable to do this?

- We access all of the lines in a set concurrently because we can overlap accessing the SRAM with comparing the set tags. It doesn't matter that we accessed data that we will not use because no other process could use these memory arrays while the processor is waiting on the data from the one which it will use. The only possible drawback is that we use more power by cycling all of the memory arrays rather than just the one we will need.

Question 13. In contrast to starting access to all ***n*** lines of an ***n***-way set associative cache even though we know we won't use them all, an access to main memory is not started until we know we have a cache miss. Why don't we start the main memory access in parallel with searching the cache so we can overlap part of the access time?

- We don't start a main memory access unless we know we will need it for two reasons. First, there may be other processes (I/O processors) that may need to use the main memory bus and we cause them to wait while we satisfy the needless access for the processor cache. In fact, the processor cache would be using the memory bus 100% of the time and lock out any other process from using it. Second, we can't just stop a main memory access and start another without waiting some period of time. We would make the processor cache wait this extra time on a legitimate access to satisfy a true cache miss.

Question 14. Stone says that a simple rule of thumb is that doubling the cache size reduces the miss rate by roughly 30%. Given that the cache in question 9 is 256K bytes, what is the expected percentage improvement in the effective access time if we double the cache size to 512K bytes?

- New miss rate = 0.08 - (0.3)(0.08) = 0.056. t_{eff} = 20 + (0.056)(60) = 23.36 ns. Percentage improvement is (24.8 – 23.36)/24.8 = 5.8%.

Question 15. What is the expected percentage improvement in the effective access time over that in question 14 if we double the cache size again to 1024K bytes?

- New miss rate = 0.056 - (0.3)(0.056) = 0.039. t_{eff} = 20 + (0.039)(60) = 22.34 ns. Percentage improvement is (23.36 – 22.34)/23.36 = 4.4%.

Question 16. What is the advantage of using a write-back cache instead of a write-through cache?

- A write-back cache significantly reduces the utilization (and thus contention) on the main memory bus because programs normally write several times to the same cache line (and even the same word). With a write-through cache, we would have to use main memory cycles to write the data to main memory repeatedly. With a write-back cache, we only need to write the line when we displace it because of some other cache miss. This is, on average, the miss rate of the cache. For example, if the miss rate of the cache is 0.03, we would have to write back the line only 3 out of 100 times, significantly reducing contention on the main memory bus. This all assumes that we have a deep enough write buffer FIFO that the processor cache doesn't have to wait for the writes to complete.

Question 17. What is a disadvantage of using a write-allocate policy with a write-through cache?

- The write-allocate policy means that I allocate a line in the cache on a write miss. This is opposed to the write no-allocate policy where I just write around the cache directly to main memory on a write miss. The disadvantage of allocating a line in the cache for a write miss is that I need to fill the rest of the cache line from main memory. This means that I need to read from main memory to fill the line as well as update the one word that was written by the processor in both the cache and main memory. When I am using a write-back cache, I do want to go ahead and allocate the cache slot in anticipation of having several more write hits where I don't go to main memory at all.

Question 18. What is "bus-snooping" used for?

- Bus snooping is where the processor cache monitors the memory bus for others addressing memory at locations that are being held in the cache. When it detects some other process (probably I/O) writing into the main memory location that is being held in the cache, it invalidates the corresponding cache line so that main memory and the cache can be kept in sync. This method is opposed to the I/O processor keeping a separate copy of the cache directory as described by Stone in section 2.2.7. The problem with the method described by Stone is that there may be several I/O processors, or "bus masters," accessing main memory. It is more reasonable for the cache to monitor the memory bus to ensure consistency with main memory than to require every bus master to have a copy of the cache directory.

Question 19. You find that it would be very inexpensive to implement small, direct-mapped cache of 32K bytes with an access time of 30 ns. However, the hit rate would be only about 50%. If the main memory access time is 60 ns, does it make sense to implement the cache?

- t_{eff} with no cache is 60 ns. t_{eff} with the cache is 30 + (.5)(60) = 60 ns. It didn't get any better with the cache, so we shouldn't implement it. Actually, we would need to use memory interleaving to achieve the 60 ns without the cache because the cycle time of DRAM is about twice as long as the access time.

Question 20. Would it make a difference on question 19 if we doubled the cache size to 64K bytes?

- Using Stone's 30% rule of thumb, the hit rate should go up with the 64k byte cache. The miss rate would be 0.5 - (0.3)(0.5) = 0.35, or a hit rate of 0.65. The new effective access time would be t_{eff} = 30 + (.35)(60) = 51 ns so it would make sense to implement it if it were cheap. We get a 15% speedup.