Quiz 4 Solutions

1. Consider a workload that finds about 90% of its data in Level 1 cache, and 95% of its data in Level 2 cache. Compute the average time to load an item from memory as seen by the processor, given that the L1 cache requires 1 cycle to load, the L2 requires 10 cycles to load, and the main memory requires 65 cycles to load.

   An access has a probability of 0.9 of finding the data item in the L1. If the item is not in the level one cache, then it is among the 10% of accesses that miss in L1. For these, about half of them can be found in the L2, while the others will miss in L2. So, on average, if we have n accesses:

   A hit in the cache = 0.9 n * 1 cycles.
   A miss in L1 but a hit in L2 = 0.1 * 0.5 n * (10) cycles
   A miss in L2, go to memory = 0.1 * 0.5 n * (65) cycles

   Average access = (0.9 n + 0.5 n + 3.25 n) / n
   Average access = 4.65 cycles

2. Reconsider Problem 1. A brilliant engineer working with you comes up with the idea of adding an L3 cache. Using simulation, you discover that the workload in problem 1 now finds its data in the L3 98% of the time. What is the average time to load an item from memory as seen by the processor. If the addition of the L3 will increase the cost of the processor by 50%, do you believe that this is worth doing?

   As seen from the question above, the memory component dominates the access time.

   The memory contribution is now = 0.1 * 0.5 * 0.4 * 65 = 1.3 cycles
   Depending on the latency of accessing the L3, the average access time will be 0.9n + 0.5n + 0.1 * 0.5 * 0.6 * x n + 1.3n
   Or (2.7n + 0.03 x n) where x is the access time of the L3.
   Given that the L3 is never going to be faster than the L2, then the access time above gives us a lower bound on the access time
   Access time ≥ 2.75n + 0.03 * 10 n [at best, the L3 is as fast as the L2]
   Access time ≥ 3.05n
   Given that the L3 will likely be much more than the L2 in terms of access time, it appears that the improvement by adding an L3 cache *of these parameters* is not exciting

3. Assume a memory array chip of "256M x 1" bits as a basic component. How many chips do you need to configure a 4GB memory? Assume one parity bit is added to every 8 bits of storage.

   The number of chips to represents 256MB = 8 + 1 (8 for the 8 bits of data and one chip to store the parity bits)

   The number of chips therefore is = 9 * (4 * 1024 / 256)
   = 16 * 9 = 144 chips.

4. Repeat the above for an array of memory chips of "256M x 4" bits as the basic component. What is the overhead that is added for reliability?

   The naïve solution here is to allocate one chip to represent the parity bit, but this would be quite a waste. Instead, we organize the data such that we have an array of 32 bits + 4 bits of parity (1 for the first 8 bits of data, 1 for the second, etc.). Thus, 8 + 1 chips (8 for the 32 bits of data and 1 for parity) will give us 256 * 4 MB of data. The number of chips therefore is 9 * (4 * 1024/(256 * 4)) or 36 chips.

5. Consider the following code:

```
double a[2 << 20];
double b[2 << 20];
double c[2 << 20];
for(i = 0; i < n; i++) {
        a[i] = b[i] * c[i];
}
```

   Assume that the cache line is 64 bytes. The processor runs at 2GHz. Calculate the necessary memory bandwidth that the processor needs to enable computation at 2GF/sec.

   Each floating-point multiplication requires bringing 24 bytes from memory and writing 8 bytes to memory. To run at 2GF with a 2GHz processor, then I must execute a multiplication every cycle. So, every cycle we will need to put 32 bytes on the memory bus, so the bandwidth is 32B * 2 GHz = 64 GB/sec

6. In a 32-bit machine we subdivide the virtual address into 4 segments as follows:

10-bit	8-bit	6-bit	8 bit

   We use a 3-level page table, such that the first 10-bit are for the first level and so on.

- What is the page size in such a system?
- What is the size of a page table for a process that has 256K of memory starting at address 0?
- What is the size of a page table for a process that has a code segment of 48K starting at address 0x1000000, a data segment of 600K starting at address 0x80000000 and a stack segment of 64K starting at address 0xf0000000 and growing upward (like in the PA-RISC of HP)