CS232 Review

(a) Caches

(a) LRU: least recently used data, often used or approximated to decide what to kick out of the cache

(b) Spacial Locality: Things near what was just used tend to get used again.
Temporal Locality: Things that were just used, tend to get reused.

(c) Associatity: Relates how addresses get mapped inside of the cache to the number of sets n. (Direct Map equal n sets for n addresses, Fully associative 1 set for n addresses)

(d) Write-back: Writing is done only to the cache. A modified cache block is "written" back into higher memory just before it is stored. (requires a dirty bit to denote that it has been written to)

(e) Multilevel caches: A cache that utilized 2 or more separate caches to store information. Typically the lower caches are small/fast/expensive where as the higher caches are large/fast/cheap.
Note: If one can ensure almost certain hits in the higher caches and relatively good rates in the lower caches this greatly increases speed.

(f) Computing cache size, bits, block offset, index, tag, and associatity

i. N-Way associative cache has N blocks per set. Thus address $\rightarrow$ cache address is given by (= denotes number of bits in the bitstring at that point dedicated.

$$|Tag = m - s - o|Index = s|offset = o|, m = addressSize \tag{1}$$

$$|t = m - o - s|s = lg(\frac{size}{blockSize \cdot N})|o = lg(\frac{blockSize}{byte})| \tag{2}$$

(g) Calculating cache performance and Average Memory Acess Time (AMAT) calculations.

i. :

$$AMAT = HitTime + (MissRate * Miss) \tag{3}$$

$$MemoryStallCycles = MemoryAccesses * MissRate * Misspenalty \tag{4}$$

(h) Memory access pattern and linearizing array access:

(i) Memory access pattern analysis: Look for temperal and spacial locaility. Calculate misrates and how to maximize locality. (Blocking/ Tiling)

(b) Cache-Aware programming

(a) Hardware stream/stride pre-fetching

(b) When to software pre-fetch

(c) Blocking/ Tiling

(c) Virtual Memory (VM):

(a) Indirection: Adding an intermediate interface that controls endpoints

(b) Use of indirection in VM: Virtual Memory $\rightarrow$ OS $\rightarrow$ physical memory

(c) Advantages of VM.

i. Larger than memory processes: We can store "memory" on disk via paging. Also because its virtual we can say we have "n" bits of mem but only have to allocate as much as needed.

ii. Identical/Conflicting address mapping: 2 programs can both map to the same address because the OS/PageTable deals with actual mapping. i.e. sure you can have $0x4004$ (hands you $0xff0f00$)

iii. Sand-boxing processes: The page table has permission bits that tells the OS who can touch the cookie jar.

iv. Controlled sharing between processes: If two addresses need to share, it can be done under adult(OS) super vision because of the page table's permission bits.

(d) Page table: Look up table that maps virtual to physical memory

(e) Translation look-aside buffer (TLB): Looking up the page table translations sucks....so we cache it...aka TLB

(f) Hierarchical page table: Page Directories $\rightarrow$ Page Entry $\rightarrow$ Page Table $\rightarrow$ Address . (note saves memory because not all Entries and Directories are filled.
P.S. page tables should be page sizes so they themselves can be looked up easily....

(g) Page faults(and its affect on virtual memory design): Miss in the TLB, (resulting in memory seak for page). Miss in the cache to get page/addresses. ew....

(d) Hard Disks

(a) Platter: Cyclinder that houses magnetic particles organized in tracks:

(b) Head: Reads/Writes magnetics particles (moves by arm)

(c) Track: Circular collection of sectors

(d) Sector: Partition of Track where data is actually stored

(e)
$$readWriteTime = Seek + RotDely + sectorRead \cdot sectors \tag{5}$$

(f) Random vs Sequential read/write performance

(g) Solid state disks (SSD): Flash memory based, everything but random writes it very fast. (writes are slow due to having to clear the whole block before to write)

(a) Error Correcting Codes

i. Error detection vs. Error correction:

A. Error Detection is the ability to detect that somehow a bit (or more) has been flipped.

B. Error Correction is the ability to detect an error has occurred and guess what it might have been before being flipped.

C. Relationship: Error Detection is achievable with fewer bits and can be done to a further level of error (more bits flipped) but does not correct. Correction gives up the ability to detect as far but adds the ability to guess initial states.

ii. Parity: Extra bit that is the result of xor the bitstring (odd or even). Parity is recomputed when reading Ram and checked. If $P_{stored} \neq P_{computed}$ an error has occured.

iii. Hamming distance: Minimum path of states to no longer detect/correct errors.

iv. SECDED: Single-bit Error Correction, Double-bit Error Detection. Standard in ECC protect SDRAM.

(b) Writing code to analyze cache with a hardware pre-fetcher