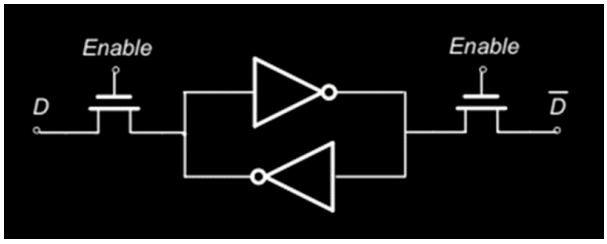
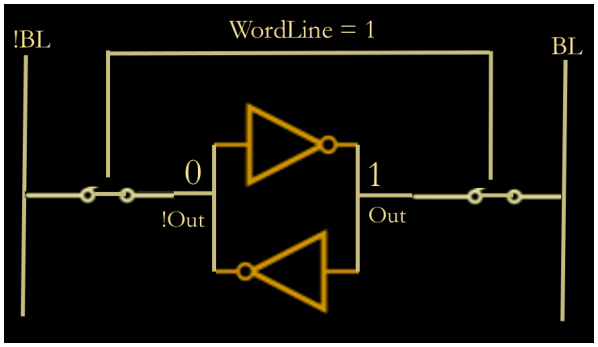
1) Describe the purpose (in the context of locality), benefit, and tradeoff of SRAM. Then, in detail, describe the operation of a 6T-cell SRAM. Include descriptions of how the Bit Line, Word Line, inverters, and nMOS transistors are used to store data to an SRAM cell.

In the memory hierarchy, SRAM is the closest to the CPU and registers. It is designed to be fast (higher performance and endurance) and is best for temporal locality, but its tradeoff is cost due to the implementation of more costly transistors. The price per bit of SRAM is much higher than disk memory and DRAM.

Inverter Loop (two transistors per inverter) and enable signals on both inputs. The value of D is the current value, and D bar contains the inverse. The loop keeps the signal charged. SRAM is larger and more expensive, but much faster, which is why it is used in registers and cache.



When WordLine is 0, the value is held. When WordLine is 1, then we can write to the SRAM. If BL is 0, then !BL is 1. The value 0 goes into the inversion loop via BL, holding the value of 0. If BL is 1 and then !BL is 0. The value of 0 goes into the inversion loop via !BL, holding the value of 1.

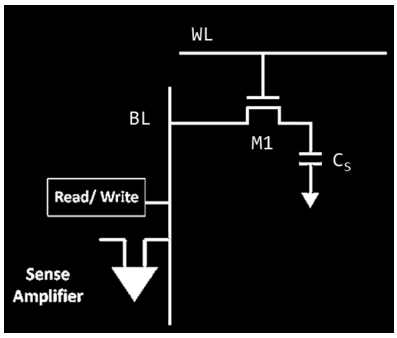


2) Describe the purpose (in the context of locality), benefit, and tradeoff of DRAM. Then, in detail, describe the operation of a 1T-cell DRAM. Include descriptions of how the Bit Line, Word Line, capacitor, and nMOS transistors are used to store data to an DRAM cell.

Dynamic Random Access Memory (DRAM) is used for main memory. In the memory hierarchy, DRAM is the next level deeper, below cache memory (SRAM). It is designed to be cheap and plentiful and is best for spatial locality. The tradeoff, considering its further distance from the CPU and registers, is speed. Also, DRAM is more volatile than SRAM as a result of the data being stored on charges by capacitors.

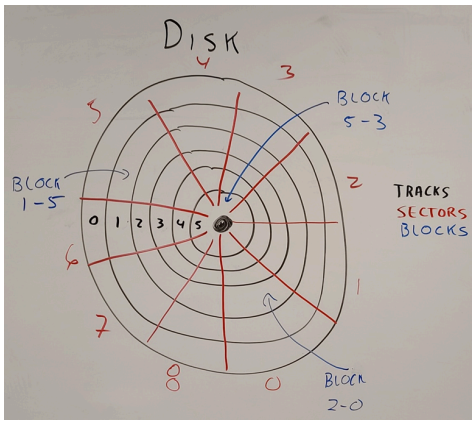
The DRAM consists of 1 transistor and a capacitor. The capacitor holds the charge, and must be continuously refreshed. When Cs = 1 and we are writing a 0, the WL is set to 1 and BL is pre-charged to half of the voltage. If BL is driven to 0 (writing a 0), then the Cs will discharge, becoming 0. When BL is 1, the Cs will eventually go to 1 (much slower since nMOS are bad at 1s, but will eventually charge the Cs.

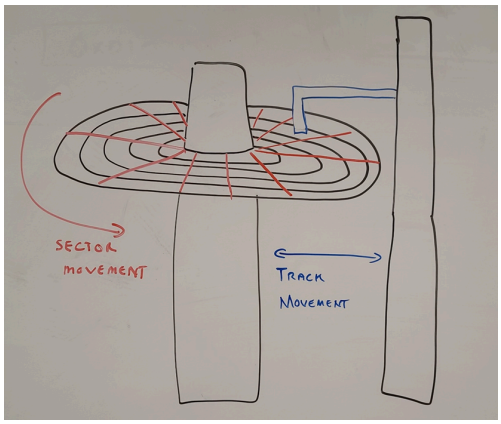
DRAM is slower than SRAM, but smaller, which means it is plentiful and cheaper, which is why it is used in main memory



3) Describe the operation of the magnetic disk structure. Specifically, describe the purpose of tracks, sectors, blocks, and how we perform reads/writes with a disk.

Memory disk consists of tracks and sectors. Tracks are logical circles. Sectors are logical slices.
Block: Intersection of a track and a sector
Therefore, there are n*m blocks of memory per disc. Any address on the disk can be reached if you know the track number and the sector number
The disk is mounted on a spindle. By spinning, the sector will change. The header will move to a specified track, which will take us to the exact block we need.





4) Describe the structure and operation of using three levels of cache memory. A full credit response includes the purpose of cache memory, the relative sizes of the 3 levels of cache, how blocks are moved between the three levels, and why modern architectures typically only have 3 levels of cache.

The purpose of cache memory is to provide a fast, volatile level of memory hierarchy between the registers and the slow, non-volatile disk.

L1 is smallest, L2 is larger and slower than L1, and L3 is larger and slower than L2.

When an element is requested in the cache, blocks are transferred between the levels in order to promote temporal and spatial locality.

After 3 levels, limited benefit. Higher levels must have more memory. Slower design to offset higher cost

Problem 5:

Part 1 - 125 points - Consider a database for the Rare Books & Special Collections at the Hesburg Library, which contains 132,000 separate volumes. Each contains the following information stored in a C struct in an x86_64 architecture.

- Notre Dame Book ID: Unsigned integer
- Dewey Decimal System Entry: double
- ISBN-13: Unsigned integer
- Manuscript Name: char array – Up to 50 characters
- Author's Last Name: char array - Up to 25 characters
- Author's First Name: char array - Up to 18 characters
- Topic Category: character
- Publisher: char array - Up to 50 characters
- Age Range: character
- Publication Year: Unsigned integer

Determine the amount of memory consumed storing these entries on a Disk with block of 1024 bytes using both segmentation and paging. Draw the layout of the first three blocks for each.

4 + 4 (seg) + 8 + 4 + 4 (seg) + 50 + 6 (seg) + 25 + 7 (seg) + 18 + 6 (seg) + 1 + 7 (seg) + 50 + 6 (seg) + 1 + 3 (seg) + 4 = 208 bytes

1024 / 208 = 4.923

Segmentation: Blocks = 132,000 / 4 = 33000

Memory size = 33000 * 1024 =33,792,000

R1	R2	R3	R4	Seg	R5	R6	R7	R8	Seg	R9	R10	R11	R12	Seg
208	208	208	208	192	208	208	208	208	192	208	208	208	208	192

Paging: 132000*208 = 27,456,000 bytes

R1	R2	R3	R4	R5.1	R5.2	R6	R7	R8	R9	R10.1	R10.2	R11	R12	R13	R14	R15.1
208	208	208	208	192	16	208	208	208	208	176	32	208	208	208	208	160

Part 2 - 175 points - Rearrange the C Struct to improve the amount of memory per entry. Determine the amount of memory consumed storing the same 132,000 entries on a Disk with block of 1024 bytes using both segmentation and paging with the improved C struct. Calculate the memory savings for both segmentation and paging.

- Manuscript Name: char array – Up to 50 characters (Any order for strings is fine. Result will be the same)
- Author's Last Name: char array - Up to 25 characters
- Author's First Name: char array - Up to 18 characters
- Publisher: char array - Up to 50 characters
- Dewey Decimal System Entry: double
- Notre Dame Book ID: Unsigned integer
- ISBN-13: Unsigned integer
- Publication Year: Unsigned integer
- Topic Category: character
- Age Range: character

50 + 6 (seg) + 25 + 7 (seg) + 18 + 6 (seg) + 50 +6 (seg) + 8 + 4 + 4 + 4 + 1 + 1 + 2 (seg) = 192 bytes

Segmentation: 1024 / 192 = 5.333

Blocks = 132,000 / 5 = 26400

Memory size = 26000 * 1024 = 26,624,000

Improvement = 33,792,000 / 26,624,000 = 1.269

Paging: 132000*192 = 25,344,000

Improvement = 27,456,000 / 25,344,000 = 1.08

Problem 6 - 200 points - Given a cache hierarchy with 8 words with an average cache access time of 5 ns, and an memory access time of 100ns, and the following set of memory accesses: 22, 31, 14, 22, 6, 22, 6, 6, 22, 14

Part 1 - 100 points - If the cache is a direct mapped cache, draw the direct mapped cache table, show the sequence for accessing the cache, and calculate the total time.

Binary: 22 = 10110, 31 = 11111, 14 = 01110, 6 = 00110

Tag	Valid	Index	Memory
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10->01->10->00->10->00->10->01	M(22)->M(14)->M(22)->M(6)->M(22)->M(6)->M(22)->M(14)
111	Y	11	M(31)

22 (miss), 31 (miss), 14 (miss), 22 (miss), 6 (miss), 22 (miss), 6 (hit), 22 (miss), 14 (miss)

9 miss and 1 hit -> 9*(100+5) + 1*(5) = 950ns

Part 2 - 100 points - If the cache is a 2-way set associative cache, draw the 2-way set associative cache table, show the sequence for accessing the cache and calculate the total time. Calculate the improvement in performance over the Direct Mapped Cache in part a

Binary: 22 = 10110, 31 = 11111, 14 = 01110, 6 = 00110

Tag	V	Index	Memory	V	Index	Memory
00	N			N		
01	N			N		
10	Y	101->011->101->001->101->001->101->001	M(22, 14, 22, 6, 22, 6, 22, 14)	Y	101->011->101->001->101->001->101	M(22, 14, 22, 6, 22, 6, 22)
11	Y	111	M(31)	N		

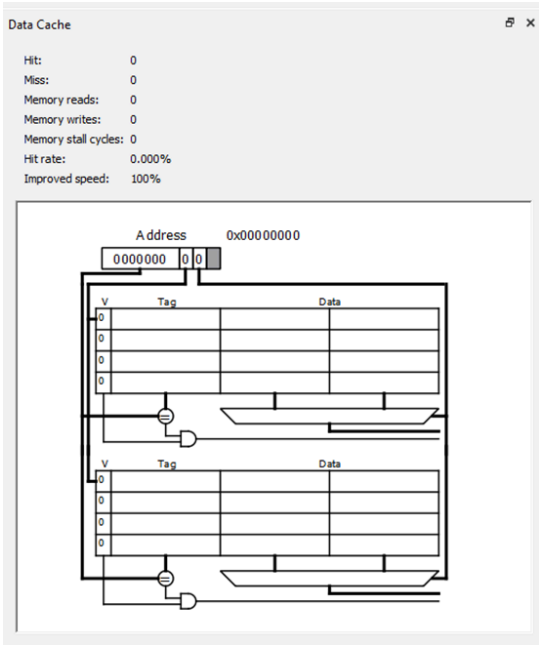
22 (miss), 31 (miss), 14 (miss), 22 (hit), 6 (miss), 22 (hit), 6 (hit), 6 (hit), 22(hit), 14 (miss)

5 misses and 5 hits -> 5*(100+5) + 5*(5) = 550ns

Improvement = 950 / 550 = 1.72

Problem 7 - 100 points - Download `hw06_arr.s`. This assembly program is set up to create an array of 1024 (identical) elements in memory, and then read those back from memory to compute the sum of all elements. This is a contrived example used as a proxy for a program that strides over a large array, without extra setup code to slow the simulator down. Read the assembly code to ensure you understand what it is doing.

Open `hw06_arr.s` in QtRVSim. Create a simulation with the default parameters for a pipelined machine with hazard unit and cache. Click the Windows menu and Data Cache to bring up the data cache view which tracks hits, misses, and hit rate (among other metrics):



Run `hw06_arr.s`. It may take a couple minutes to run through the ~20000 cycles; if you are using a laptop, plugging into external power should prevent your CPU clock rate being throttled by battery saver mode.

Record the total hits, misses, and hit rate for the data cache. Explain why this hit rate was achieved based on the access pattern and the cache design parameters. Be very specific, referencing which array element accesses result in hits and which result in misses, and why.

Every time the program ran, when sw x8, 0(x9) was being executed it resulted in a miss. The program produced a hit every other time through the loop when it entered the sum_loop with lw x10, 0(x9). The other half of the time it produced a miss. That means for half of the program it was only producing misses and then for the other half it produced a hit or a miss 50% of the time which resulted in a 25% hit rate.