Computer Architecture

*Homework Problems for section Memory Hierarchy*

Deadline: 26 May, 2023  
Upload one solution per group to Brightspace!

TOPICS COVERED

These homework tasks focus on the important concepts about the memory hierarchy, which has been introduced in the past three lectures about the following topics:

* Storage technologies
* Spatial and temporal locality
* Caching
* Virtual memory
* Malloc

Indholdsfortegnelse

[QUESTIONS 3](#_Toc136005568)

[1. What is the difference between volatile and nonvolatile memory? 3](#_Toc136005569)

[2. Why does it take a long time to read or write a value in the main memory? 3](#_Toc136005570)

[3. What’s the easiest way for a HDD manufacturer to increase the capacity of a drive? 3](#_Toc136005571)

[4. Why does the CPU not wait for a disk read? How does it get the data instead? 3](#_Toc136005572)

[5. Name a few advantages and disadvantages of SSD over HDD. What can we do about the disadvantages? 3](#_Toc136005573)

[6. Define temporal and spatial locality. How do they help with slow memory and disk access times? 4](#_Toc136005574)

[7. What is the difference between the 3 cache miss types? 4](#_Toc136005575)

[8. Approximately how many clock cycles does it take to access data from the following memories: registers, CPU caches, main memory, disk, network storage. 5](#_Toc136005576)

[9. How do we read data from a direct mapped cache (step by step)? At which step do we know for sure if we have a cache hit or miss? 5](#_Toc136005577)

[10. What is the “memory mountain”? 5](#_Toc136005578)

[11. Why is virtual memory considered a cache? 6](#_Toc136005579)

[It is considered a cache due to its content being an array with bytes of virtual addresses that are stored in the main memory as cache. 6](#_Toc136005580)

[12. What is a Page Table and a Page Fault? How are Page Faults handled? 6](#_Toc136005581)

[13. What problem is solved by the Translation Lookaside Buffer (TLB)? 6](#_Toc136005582)

[14. In what cases do we get a Segmentation Fault? 6](#_Toc136005583)

[15. How does malloc work in general, and what constraints must be satisfied by its implementations (allocators)? 6](#_Toc136005584)

[16. What rules must be followed by a C programmer when using malloc? What happens when the rules are violated? 7](#_Toc136005585)

[17. What is internal and external fragmentation in memory allocation? 7](#_Toc136005586)

[PRACTICAL PROBLEMS 7](#_Toc136005587)

[**1.** **Data Cache** 7](#_Toc136005588)

[**2.** **Cache miss rate** 8](#_Toc136005589)

[PROGRAMMING EXERCISE 8](#_Toc136005590)

# QUESTIONS

Answer the following questions briefly (1-3 sentences)

## What is the difference between volatile and nonvolatile memory?

* + Volatile memory is memory that is in danger when power is lost.
  + Nonvolatile memory is not in danger of power loss.
  + Ram is volatile, as it is dependend on power to keep the memory.
  + The disk is nonvolatile, as it is written to the disk, and not depended on power to keep the memory.

## Why does it take a long time to read or write a value in the main memory?

* + Main memory isn’t as slow as reading from the disc, but is slower than if it were closer to the CPU. Compared to the memory closer to the CPU, the main memory requires more steps to read or write data, than if it were closer to the CPU.

## What’s the easiest way for a HDD manufacturer to increase the capacity of a drive?

* + The easiest way is to add more discs and implement multiple levels per disc.

## Why does the CPU not wait for a disk read? How does it get the data instead?

* + The CPU only has access to the registers located on its chip and main memory. Therefore, the CPU communicates to the control unit to perform the disk reads and then copy the data to main memory. This happens independently of what the CPU is performing in the meantime.

## Name a few advantages and disadvantages of SSD over HDD. What can we do about the disadvantages?

* + The main advantage of using a SSD over a HDD is the transfer speed. Reading from SSD takes about whereas the HDD takes about 3ms, making the SSD about times faster than the HDD.
  + The HDD uses the conventional disk drive to read and write a data and its speed is limited by the physical maximum rotational speed of the disk drive. A SSD uses charge trap flashes, where it stores electrons in memory cells and depended on how many electrons there are in one cell, the value in bits varies, but is of size 3 bits in one cell.
  + The technology of an SSD is way faster, but is also more expensive than the HDD, which is the disadvantage of the SSD.
  + The picture illustrates it nicely.



## Define temporal and spatial locality. How do they help with slow memory and disk access times?

Et billede, der indeholder håndskrift, skitse, Font/skrifttype, Stregtegning

Automatisk genereret beskrivelse

Locality is well written programs, that reference data items near other recently referenced data items.

Spatial locality is when data is reference to nearby recently referenced data soon. The temporal locality is when a reference is likely to be used again soon, and the programmer then keeps the reference to be reused.

Locality is practical because the smaller jumps between references speeds up programs and is useful to speed up reading/writing of slow memory. One of the reasons why virtual memory is useful even though it is slower, is due to good locality.

## What is the difference between the 3 cache miss types?

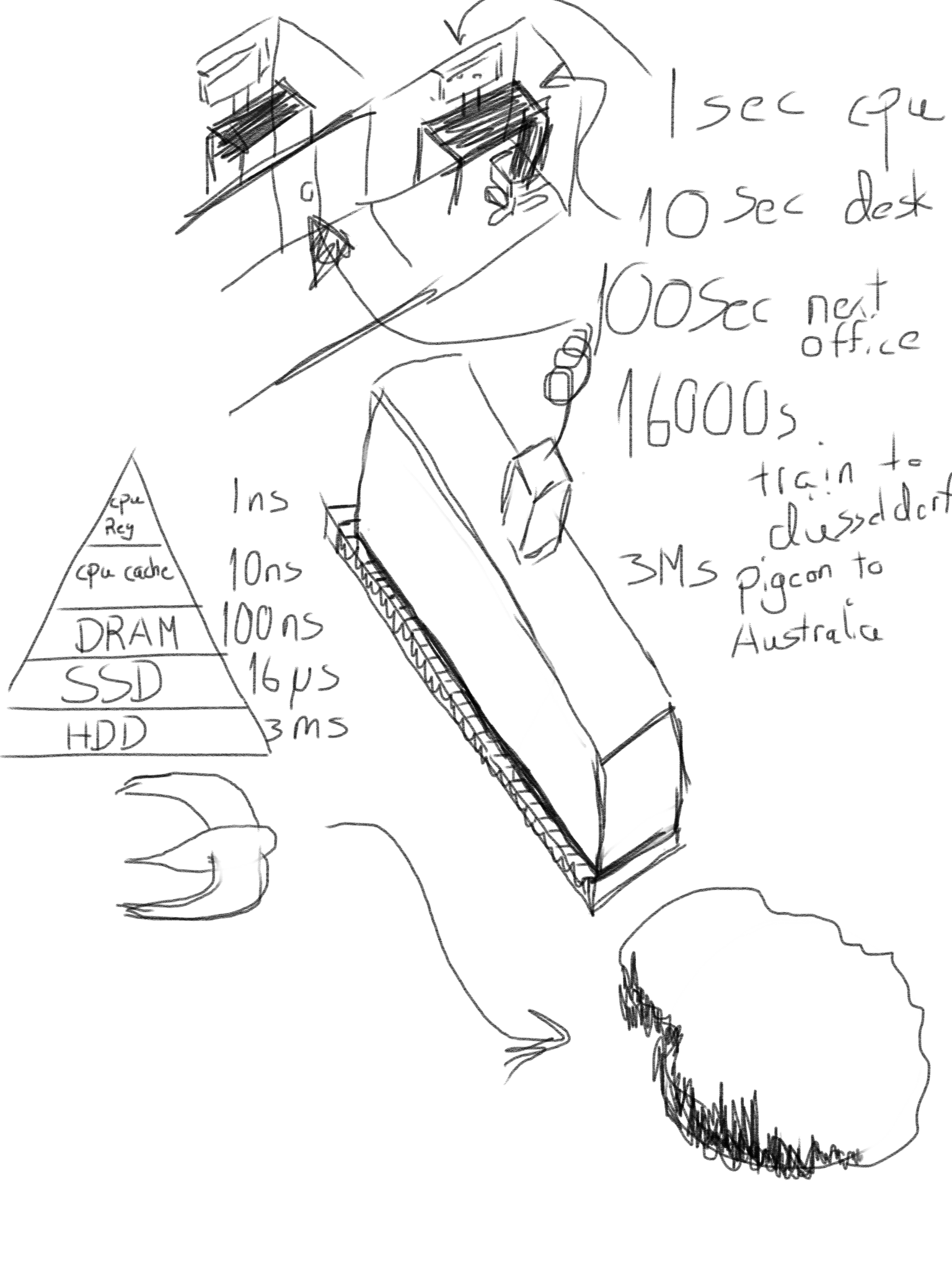
A cold miss is one, and happens when the cache cell is empty.

A conflict miss occurs, when the available cache is large enough to store the data, but because of the mapping of the cache, there isn’t sufficient storage next to one another.



A capacity miss is the last of the 3 misses, and occurs when a set of blocks, a *working set*, exceeds the size of the cache.

## Approximately how many clock cycles does it take to access data from the following memories: registers, CPU caches, main memory, disk, network storage.



The above is an illustration of the hierarchy, that I made with Daniels example of approximate speeds in the memory hierarchy.

Registers: 1ns, 0 cycles.   
CPU cache: 10ns, (4, 10, 50) cycles depended on which level of cache it is.  
Main memory: , (Virtual memory: 200, buffer cache: 200) cycles.   
DRAM: 100ns, \   
SSD: 16s, Local disc. Approximate amount of cycles is 100.000.   
HDD: 3ms, /   
 ,  
Network- /   
Browser cache: , Approximate amount of cycles is 10.000.000.   
Remote server disk: , Approximate amount of cycles is 1.000.000.000.

## How do we read data from a direct mapped cache (step by step)? At which step do we know for sure if we have a cache hit or miss?

With the direct mapped cache, the first 6 bits is used as the tag. The next 4 bits is used for the index. The last bits tells which block to read.   
If the tag is at the index, it’s a hit. If the tag is not at the index, it’s a miss.   
If the block is empty, it’s a cold miss.

## What is the “memory mountain”?

The memory mountain is a function of stride and memory size. The stride refers to the locality, how many bytes, the program must jump to the next data reference.  
With the stride and memory size, a function of two variables is made, and the reading in MB/s is then the z result. Graphically it then looks like a mountain

## Why is virtual memory considered a cache?

## It is considered a cache due to its content being an array with bytes of virtual addresses that are stored in the main memory as cache.

## What is a Page Table and a Page Fault? How are Page Faults handled?

A page table is a data structure that describes the address space of different processes.

A page fault is an exception that leads to the restarting of the faulting instruction. Page faults are handled by the kernel, which makes the required page accessible in the physical memory. Furthermore, the kernel can deny illegal memory access to handle the exception.

## What problem is solved by the Translation Lookaside Buffer (TLB)?

TLB solves the problem of slow address translation that occurs, whenever the CPU generates a virtual address. Without a TLB, the address translation would require an extra fetch from memory, which comes at the cost of many hundreds of cycles.

## In what cases do we get a Segmentation Fault?

Segmentation faults are typically reported due to the general protection fault. The cases where we get this fault is typically, when a program references an undefined area of virtual memory, or when a program attempts to write to a read-only text segment.

## How does *malloc* work in general, and what constraints must be satisfied by its implementations (allocators)?

The malloc function allows programs to allocate blocks from the heap (a block of memory). It works by returning a pointer to the start of the allocated block.

Since the malloc package is an explicit allocator, it must satisfy the following constraints that aim to maximize performance goals such as the throughput and memory utilization:

* Handling of arbitrary request sequences
* Forwarding immediate responses to requests
* Exclusive use of the heap
* Alignment requirement of the block
* No modification of allocated block

## What rules must be followed by a C programmer when using malloc? What happens when the rules are violated?

When using malloc, the program must no request a heap larger than the available virtual memory. If this happens, malloc returns NULL and sets errno.

## What is internal and external fragmentation in memory allocation?

Internal fragmentation occurs when an allocated block is larger than the requested payload.   
External fragmentation occurs when there is sufficient free memory to satisfy an allocate request, but without one single block with sufficient free memory to store it in.

# PRACTICAL PROBLEMS

## **Data Cache**

Design a 16 byte cache over 32 bytes of main memory, that produces at least 2 hits with the following address trace: 0, 3, 14, 25, 28, 14. Prove that your solution satisfies the goal by showing the cache contents after each read, and marking cache hit or miss!

Main memory, implies that it’s physical addresses.

The address consists of 12 bits.



|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 14: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 25: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 28: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 14: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |

So, we have 4 different indexes: 0000, 0011, 0110 & 0111  
If every block is 4 bytes in size, 8 blocks then totals 32 bytes from main memory.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Idx** | **Tag** | **Valid** | **B0** | **B1** | **B2** | **B3** |
| 0 | 0 | 1 | - | 100 | 30 | 99 |
| 3 | 0 | 1 | - | - | 150 | - |
| 6 | 1 | 1 | 30 | 40 | 199 | 20 |
| 7 | 0 | 1 | - | 70 | - | - |

Address 0: idx 0, tag 0, B0 = Cold miss!  
Address 3: idx 0, tag 0, B3 = HIT! Cache: 99  
Address 14: idx 3, tag 0, B2 = HIT! Cache: 150  
Address 25: idx 6, tag 0, B1 = Miss, not the right tag!  
Adresss 28: idx 7, tag 0, B0 = Cold miss!

## **Cache miss rate**

Consider a memory system that consists of two cache layers L1 and L2 above the main memory.

* ●  What is the average access time of a memory address, in CPU cycles? Use the values from the table below to determine the cache hit rates.
* ●  Based on your calculations give the general formula of access time for this 2-level cache configuration.

A = 6th digit of your AU ID (AUID=123456 → A=6)   
B = 5th digit of your AU ID (AUID=123456 → B=5)

page3image2944448

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | L1 hit rate |  | B | L2 hit rate |
| 1, 2 or 3 | 99% |  | 1, 2 or 3 | 70% |
| 4 or 5 | 98% |  | 4 or 5 | 65% |
| 6 or 7 | 96% |  | 6 or 7 | 60% |
| 8 or 9 | 95% |  | 8 or 9 | 55% |

L1 access time: 1 cycle  
L2 access time: 10 cycles  
Memory access time: 100 cycles

Inserting the values from the table gives us:

# PROGRAMMING EXERCISE

Consider the simple program below that tests the memory allocator in your computer. It tries allocating more and more integers in each iteration until we run out of memory and the program crashes.

#include <stdlib.h> // For malloc

#include <stdio.h> // For printf

int main(){

printf("Integers are %d bytes in this computer\n", (int) sizeof(int));

unsigned long long i=0;

while(1){

// Try to allocate 'i' million ints

void \*memory = malloc(i \* 1E6 \* sizeof(int));

if(memory == NULL){

// malloc failed to allocate the memory we asked for

return -1;

}

// 'llu' is for 'long long unsigned'

printf("Successfully allocated %llu million integers\n", i);

++i;

}

return 0;

}

Start by understanding the code and running it on your machine. Save it as **main.c** and compile with the following command: **gcc main.c -o memory\_test** This produces an executable file called *memory\_test* in the same folder. Run it by **./memory\_test** to see the result.

These exercises are done on a Macbook Air 2013:

Running the first code template gave the following result.

Et billede, der indeholder tekst, skærmbillede, sort-hvid, sort

Automatisk genereret beskrivelse

Which took whole 41 minutes to crash!

Exercises:

1. Use ***calloc*** instead of *malloc* (calloc overwrites the newly allocated block with zeros), and check the time penalty (e.g. what’s the total runtime of your program with malloc and calloc).

Declaration of calloc:

Void \*calloc(size\_t nitems, size\_t, size);

Nitems is the amount of items, and size is the size of each items.

In malloc we had one size, which was the amount of items times the size of an int. We then need to rewrite some of the code.

From:

void \*memory = malloc( i \* 1E6 \* sizeof(int) );

to

void \*memory = calloc( i \* 1E6, sizeof(int) )

The result

Et billede, der indeholder tekst, skærmbillede, sort-hvid, sort

Automatisk genereret beskrivelse

Which took 55 minutes! This is taken too long to continue with these exercises.   
We tried with windows computers as well, but they would stop at 100 million integers and then break with no error code. Only my Macbook worked, but it’s taken too long to continue.

1. Use ***realloc*** instead of *malloc* to increase the allocated memory instead of allocating a new block, and see if there’s a difference in the maximum amount of allocated memory!
2. Use ***malloc*** but also ***free*** the memory within the same iteration, before trying to allocate a larger block. See if there’s a difference in runtime or the maximum amount of allocated memory.