# Reeksamen 2023

By

Study number: 202204617

AU-ID: 689481

Indholdsfortegnelse

[Reeksamen 2023 1](#_Toc143511214)

[Theoretical questions 2](#_Toc143511215)

[1. What is data overflow and which operations are vulnerable to it? (3 points) 2](#_Toc143511216)

[2. What is the “word size” of a computer? What are the typical values? (3 points) 2](#_Toc143511217)

[3. Briefly describe the memory hierarchy (focus on the most important properties). What is the “memory mountain”? (4 points) 2](#_Toc143511218)

[4. What is instruction encoding? What is the difference between machine code and assembly? (4 points) - Mangler ( måske 206 I bogen. 2](#_Toc143511219)

[5. List and briefly describe 3 major problems that arise with concurrent programming (4 points) - Mangler ( 1007 I bogen ) 2](#_Toc143511220)

[6. What is a Page Table and a Page Fault? How are Page Faults handled? What is Memory Thrashing and why do poorly written programs cause it? (4 points) 2](#_Toc143511221)

[7. How does the compiler compile a switch expression? When and why is it better than a sequence of if-else? What is the main limitation of using switch? (4 points) - Mangler 3](#_Toc143511222)

[8. Briefly describe how a single core CPU runs multiple processes concurrently. Explain the concept of context switching, and the important design decisions about it. (4 points) 3](#_Toc143511223)

[9. List and briefly describe 3 major problems that arise with concurrent programming (4 points) - Mangler 3](#_Toc143511224)

[Practical questions 3](#_Toc143511225)

[1. Amdahl’s law. 3](#_Toc143511226)

[2. Assume a pipeline with 4 stages: 5](#_Toc143511227)

[3. Given the following assembly code, write the C code associated to it: 7](#_Toc143511228)

[4. Consider the following 7-bit IEEE-format floating point representation where each number has 2 exponent bits and 4 fraction bits. 8](#_Toc143511229)

[5. Take the logic gate diagram and convert to HCL code. 9](#_Toc143511230)

[6. Design a 16 byte cache over 64 bytes of main memory, that produces at least 2 hits with the following address trace: 3, 0, 14, 28, 25, 14. Prove that your solution satisfies the goal by showing the cache contents after each read, and marking cache hit ormiss 10](#_Toc143511231)

## Theoretical questions

1. What is data overflow and which operations are vulnerable to it? (3 points)

Data overflow is when the number trying to be represented by the bytes, requires more bytes than what’s it’s been given.

These situations can occur frequently when using multiplication, as these operations can add to needing more than just 1 more bit/Byte, as addition/subtraction can result in.

### What is the “word size” of a computer? What are the typical values? (3 points)

The word size of a computer represents the most frequently used form for size of bytes used within the system. In the early days of computers the typical values of a modern computer were 32 Bytes.   
Today the modern computers usually use 64 Bytes, with smaller gadgets using 32 Bytes and microcontrollers usually using around 8-16 Bytes.

### Briefly describe the memory hierarchy (focus on the most important properties). What is the “memory mountain”? (4 points)

The memory hierarchy dictates in which order the memory is being searched for. When failing to get data at memory level *k*, memory is then looked at, at memory level *k + 1*.

The memory mountain then graphically shows the specifications of each memory level when it comes to speed and storage.

When talking about the levels, the order is set from level 1 to level 3 and then to main memory.

The order is set by its speed, ranked from high speed to low speed.

### What is instruction encoding? What is the difference between machine code and assembly? (4 points) - Mangler ( måske 206 I bogen.

Machine code is the machine running bytes of instructions. Assembly is the language just a bit lower than that. Assembly also goes instruction to instruction but is readable because it has symbols of instructions instead of bytes as instructions.

### List and briefly describe 3 major problems that arise with concurrent programming (4 points) - Mangler ( 1007 I bogen )

### What is a Page Table and a Page Fault? How are Page Faults handled? What is Memory Thrashing and why do poorly written programs cause it? (4 points)

The page tables consists from withing the Memory Management Units and consists of addresses retrieved by the cache or the main memory.

This is used when having with virtual memory, as the MMU is supposed to have fast retrievals of the addresses.   
If an address isn’t in the MMU, then a new page is fetched from the cache if possible and if not, then check for it in main memory.   
This speeds up programs with good locality, taking advantage of the few addresses often or by using the already fetched pages.

Poorly written programs doesn’t take advantage of the already fetched pages or the pages next to the previous pages. This causes the program to keep fetching addresses from lower speed memory systems.

### How does the compiler compile a switch expression? When and why is it better than a sequence of if-else? What is the main limitation of using switch? (4 points) - Mangler

### Briefly describe how a single core CPU runs multiple processes concurrently. Explain the concept of context switching, and the important design decisions about it. (4 points)

The CPU can only run one instruction at a time. Sometimes it has to wait for tasks to respond. It can then put the instruction aside until the task has responded. This occurs frequently when having with network connections, as you have to wait for the data base to retrieve your signal and respond.

This is what context switching is about, it put aside the first task to start the second task, waiting for the second task to terminate or save its context as well to return or go to another task.

### List and briefly describe 3 major problems that arise with concurrent programming (4 points) - Mangler

## Practical questions

### Amdahl’s law.

You have measured the runtime of your program and it looks like it’s not fast enough. After profiling it, you found that there is a bottleneck in the code which can be improved by replacing the sequential code with a concurrent implementation. The bottleneck accounts for 85% of the program’s execution time in the sequential implementation.

Your goal is to parallelize the whole program run in order for it to run in **50% or less than the original runtime**. Assume that it’s possible to parallelize the bottleneck code to any degree and there are no costs/penalties due to the parallelization process.

#### Calculate the runtime when you have 2, 3, and 4 cores and compare to the original runtime.

For this I will be using Amdahl’s law which says,

With being its optimized time and T being the old time.

, and is the part of the program that can be optimized.

, being the speed up factor, set by the amounts of cores.

With 2 cores, the runtime will be 0,575 times the old runtime.

The same follows with 3-, 4 cores.

It’s speed up then comes from

Checking their speed ups:

With 100% meaning its start value, the program will be able to be optimized by , corresponding to having 2, 3 & 4 cores.

========================================

========================================

#### How many cores at least do you need to achieve the stated goal?

With this we use the formular for speedup, shown in previous question.

The wished optimization was 50% of the original runtime.

The required speedup for that is then 1,33

*Ligningen løses for k vha. WordMat.*

The optimization needed to be ATLEAST 50% better than the original. As you can’t add half a core, this then means, that we will have to round up.

The required cores to fullfill the task will then be:

===============

Cores needed >= 2

===============

### Assume a pipeline with 4 stages:

* Stage 1 logic takes 180 ps
* Stage 2 logic takes 125 ps
* Stage 3 logic takes 125 ps
* Stage 4 logic takes 75 ps

Registers in each stage add a delay of 20ps.

Answer the following:

* Compute the throughput for the pipeline in GIPS.

This is **a non-uniform pipeline**, meaning its slowest part dictates the speed.

The speed will then be:

The GIPS will then be

Checking for its prefix:

Therefore meaning

=========

=========

* Compute the delay for an instruction.

With my way of computing the GIPS, I have already done this above:

=========================================

=========================================

* If you transformed it into a 3 stage pipeline, what is the best choice of stages to merge? What is the resulting throughput and delay?

The best stages to merge will be the ones taking the least amount of time to finish, as these are the ones retrieving the most time penalty in a non-uniform pipeline.

The best stages to merge will then stage 4 and & 3 or 2.

* Stage 1 logic takes 180 ps
* Stage 2 logic takes 125 ps
* Stage 3 logic takes 200 ps

=============================================

GIPS =

=============================================

Which comes with a pro and a con, having the GIPS being reduced, but the time per instruction to be sped up.

* If you transformed it into a 2 stage pipeline, what is the best choice of stages to merge? What is the resulting throughput and delay?

The same procedure as last question:

The best ones to merge is now stage 1 & 2.

* Stage 1 logic takes 305 ps
* Stage 2 logic takes 200 ps

=============================================

GIPS =

=============================================

### Given the following assembly code, write the C code associated to it:

# long dummy(long n)

# n: %rdi

dummy:

movq $1, %rax

jmp .TEST

.LOOP:

imulq %rdi, %rax

subq $1, %rdi

.TEST:

cmpq $1, %rdi

jg .LOOP

ret

So this shows a function called dummy which returns a long integer.

Withing the function, a while loop then compares the n with 1. If n’s greater than 1, it then loops.

Withing the loop it then multiplies the result value(being 1 initially), with the *n*, and then subtracts 1 from n.

The code:

### Consider the following 7-bit IEEE-format floating point representation where each number has 2 exponent bits and 4 fraction bits.

Et billede, der indeholder tekst, skærmbillede, linje/række, Font/skrifttype

Automatisk genereret beskrivelse

If a value is represented with the following 1101110 bit sequence, answer the following:

#### Calculate the exponent bias

1 10 1110 bit

The number tells us, that its sign must be negative, it’s exponent in decimal and the fraction in decimal.

Using the formular for the exponent bias:

===========

===========

#### Calculate the value in decimal

The exponent bits aren’t the highest possible value nor is it the smallest possible value.

Then is this case the ***normalized case***, where

Now let’s calculate

=======================

And the value being

=======================

#### Round the number to have 3 fraction bits using the round-to-even mode

With the round to even mode, the least significant bit becomes even:

Then the fraction value will be

==============

==============

### Take the logic gate diagram and convert to HCL code.

Et billede, der indeholder diagram, linje/række, skitse, design

Automatisk genereret beskrivelse

Let’s have a look at the individual parts.

The inputs coming into the And gate is:

This comes into an or gate:

======================================

======================================

### Design a 16 byte cache over 64 bytes of main memory, that produces at least 2 hits with the following address trace: 3, 0, 14, 28, 25, 14. Prove that your solution satisfies the goal by showing the cache contents after each read, and marking cache hit or miss

This will be made with 2-way associative mapping.

|  |  |  |  |
| --- | --- | --- | --- |
| Set index | Valid | Tag | Cache block offset |
| 0 |  |  |  |
| 0 |  |  |  |
| 1 |  |  |  |
| 1 |  |  |  |

The addresses:

000 0 11

000 0 00

001 1 10

011 1 00

011 0 01

001 1 10

With the addresses we see, that the first tag is 000, set 0 which results in a cold miss

It then calls the same tag and same cache, resulting in a hit.

It then calls the 001 tag, set 1 resulting in a cold miss.

It then calls the 011 tag, set 1 resulting in a cold miss.

It then calls the 011 tag, set 0 resulting in

Cold miss ( 000 0 11 )

|  |  |  |  |
| --- | --- | --- | --- |
| Set index | Valid | Tag | Cache block offset |
| **0** | **1** | **000** |  |
| 0 |  |  |  |
| 1 |  |  |  |
| 1 |  |  |  |

HIT! ( 000 0 00 )

|  |  |  |  |
| --- | --- | --- | --- |
| Set index | Valid | Tag | Cache block offset |
| **0** | **1** | **000** |  |
| 0 |  |  |  |
| 1 |  |  |  |
| 1 |  |  |  |

Cold miss ( 001 1 10 )

|  |  |  |  |
| --- | --- | --- | --- |
| Set index | Valid | Tag | Cache block offset |
| 0 | 1 | 000 |  |
| 0 |  |  |  |
| **1** | **1** | **001** |  |
| 1 |  |  |  |

Cold miss ( 011 1 00 )

|  |  |  |  |
| --- | --- | --- | --- |
| Set index | Valid | Tag | Cache block offset |
| 0 | 1 | 000 |  |
| 0 |  |  |  |
| 1 | 1 | 001 |  |
| **1** | **1** | **011** |  |

Cold miss ( 011 0 01 )

|  |  |  |  |
| --- | --- | --- | --- |
| Set index | Valid | Tag | Cache block offset |
| 0 | 1 | 000 |  |
| **0** | **1** | **011** |  |
| 1 | 1 | 001 |  |
| 1 | 1 | 011 |  |

HIT! ( 001 1 10 )

|  |  |  |  |
| --- | --- | --- | --- |
| Set index | Valid | Tag | Cache block offset |
| 0 | 1 | 000 |  |
| 0 | 1 | 011 |  |
| **1** | **1** | **001** |  |
| 1 | 1 | 011 |  |

==========================================

Which produced *cold, hit, cold, cold, cold, hit*.

==========================================