# Eksamen 2020

## Teoretiske spørgsmål

### Et billede, der indeholder tekst, skærmbillede, kvittering, nummer/tal Automatisk genereret beskrivelseSpørgsmål 1. How do you calculate the number of bits needed to represent a positive integer? How are negative integers represented in binary? (4 points)

Show it with 4 bits in the table below.

1. Representing integers is the same whether they are positive or negative. The difference occurs when one of them is an unsigned integer.

Integers use the MSB integer to determine, whether to interpret the value as negative or positive.

The table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Bit pattern | 1100 | 0011 | 1011 | 0010 |
| Unsigned value | 12 | 3 | 11 | 2 |
| Signed value | -4 | +3 | -5 | +2 |

### Spørgsmål 2. Briefly describe how fractional numbers are represented in binary. What is the decimal value of ​*F*?​ (4 points)

The fractional part of numbers are read as

Where *n* is the bits position next to the dot, and thereby it’s position. Bytes are interpreted in the same way as the non-fractional. MSB from left to right.

Ex.

|  |  |  |
| --- | --- | --- |
| Last digit of your AU ID |  |  |
| 1, 2, 3 | 1010.111 | 10.875 |
| 4, 5, 6 | 1100.001 | 12.125 |
| 7, 8, 9, 0 | 1001.011 | 9.375 |

### Spørgsmål 3. What is the purpose of the Jump instruction? (3 points)

The purpose of jump instructions is to move to a block of code. The block of code is then only called if the jump instruction occurs. This comes in handy when combining it with conditions, and thereby only jumping to the block of code, if a condition is met.

### Spørgsmål 4. What is the benefit of pipelining? What is the main limitation of pipelining ? What are the two types of hazards when talking pipelining and why do they happen? (4 points)

With pipelining processes can be split up into parts. This is useful when the next part doesn’t require data from the previous part.

When dealing with network communication, there can be some time waiting for a response from the internet. Instead of the process to await the response, it can do other tasks in the meantime and then return when the response is there.

The main limitation is, that only if the time saved from separating the parts is greater than the extra time it takes from saving to register, then pipelining is useful.

Another limitation is, that the separating of parts can only be as fast as its slowest part.

A hazard can occur, if the next part requires data from the current part, then the result can be misleading.

### Spørgsmål 5. What is a cache miss and what happens when it occurs? (3 points)

A cache miss occurs when an address is not accessed properly or if the cache is empty.

When a cache miss occurs, the process depended on the data fails to receive any.

### Spørgsmål 6. What is the main difference between physical and virtual memory addressing? (4 points)

The main difference between physical memory addressing and virtual memory addressing is that the physical stores addresses in DRAM while the virtual stores addresses on the DISC. The difference in speed is about 1000 times, with DRAM being the fastest.

Only when the PTE encounters a page miss, the virtual memory is used, otherwise the physical memory is used.

### Spørgsmål 7. What is the role of number 8 in the following memory addressing mode and what is it generally used for? (3 points)

The movq moves 8 bytes to the destination, from the sources. What comes before the destination is the offset, so this moves 8 bytes from the register %rdx, to %rbp + 8 bytes offset.

### Spørgsmål 8. What is the stack, and where does the stack pointer ​%rsp​ point to? What do the ​push and ​pop​ instructions do? (4 points)

The registers have a “bottom” and a “top”, as seen illustrated.



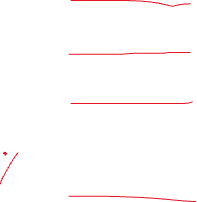
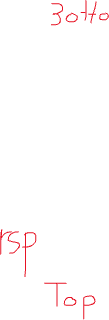
They are a bit confusing. %rsp is then the register at the top.

Pushing inserts a new top element at %rsp and decrements the

pointer 8 bytes, moving down.

Popping deletes the top element at %rsp and increments the

pointer 8 bytes, moving up closer to the bottom.



### Spørgsmål 9. Why does the C fork() call return twice, and how do we know which branch we are in? What are the similarities and differences between the parent and child process after using ​fork()​? (4 points)

From the fork call and forward, everything will be called as a parent process and a child process. The fork then returns the id of the child and parent. The fork return value is then how we distinquish one from the other.

Child processes id returns 0, everything else than that is the parent.

The child is a duplicate of the parent. The child process is supposed to be run before the parent one otherwise, a zombie process might still take up resources, even after the child process has been terminated. This is why the parent process should wait for the child process to terminate before running itself.

## Practical questions ( 11 points each )

### Spørgsmål 1. Consider a memory system that consists of two cache layers L1 and L2 cache above the main memory.

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

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

|  |  |  |  |
| --- | --- | --- | --- |
| A | L1 hit rate | B | L2 hit rate |
| 1, 2 or 3 | 97 % | 1, 2 or 3 | 65 % |
| 4 or 5 | 95 % | 4 or 5 | 75 % |
| 6 or 7 | 99 % | 6 or 7 | 55 % |
| 8, 9 or 0 | 98 % | 8, 9 or 9 | 50 % |

L1 access time: 1 cycle

L2 access time: 20 cycles

Memory access time: 100 cycles

1. What is the average access time of a memory address, in CPU cycles? Use the values from the table above to determine the cache hit rates.

L1 access time = 1 cycle.

Average memory access time = time for the hits + time for the misses.

When a miss occours, a time penalty is received.



The notation for hitRatios will be labeled as , where *n* is the level of memory in the hierarchy.

The notation for missRatios will be labeled as .

There can be 3 scenaries in this case. A hit, a miss -> hit, a miss -> miss -> hit.

The average memory access time will then be:

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

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

1. Based on your calculations give the general formula of access time for this 2-level cache configuration.

is a garantied hit.

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

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

Let’s check if it’s valid

Which should be the same as the above expression but doesn’t equal the same answer.

### Spørgsmål 2. With the pipeline segmented into stages below:

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

Automatisk genereret beskrivelse

* Compute the throughput for the pipeline in GIPS.

As this is a non-uniform pipeline, the slowest of parts determines the pace.

The throughput is:

The amount of instructions is 5.

So the GIPS is

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

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

* Compute the throughput for the pipeline in GIPS if Comb logic A and Comb logic B are combined into a single block (with 200ps response).

Combining this into a 4 stage process, we calculate the GIPS once more.

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

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

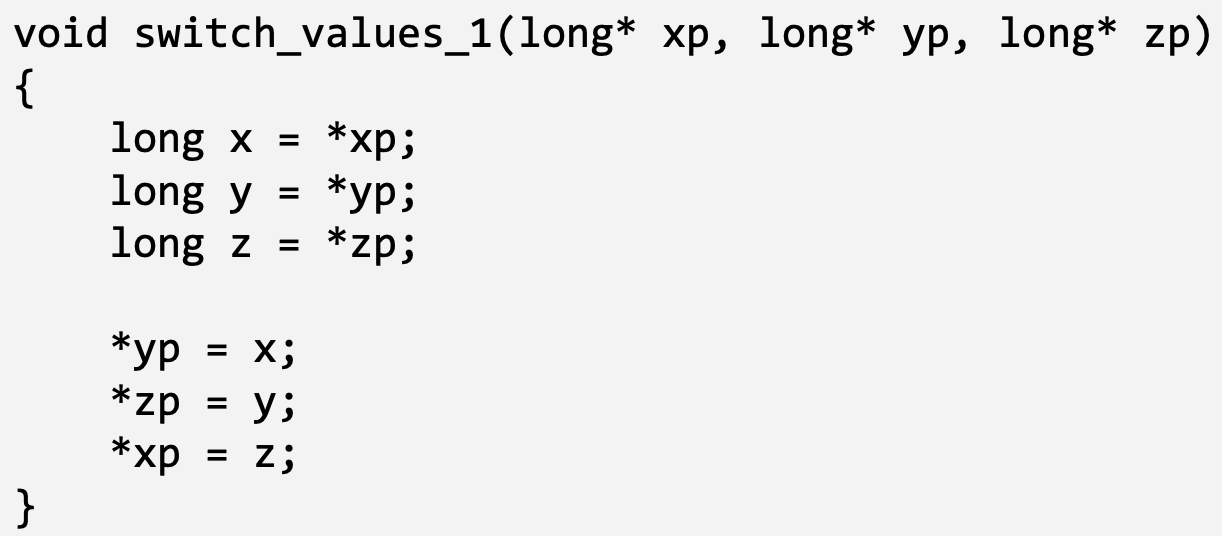
As seen, changing these doesn’t matter, as the pipeline is nonuniform, and this combined stage is still far faster than the comb logic D. Therefore the throughput hasn’t changed.

* What is the delay of an instruction going through the pipeline in both cases?

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

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

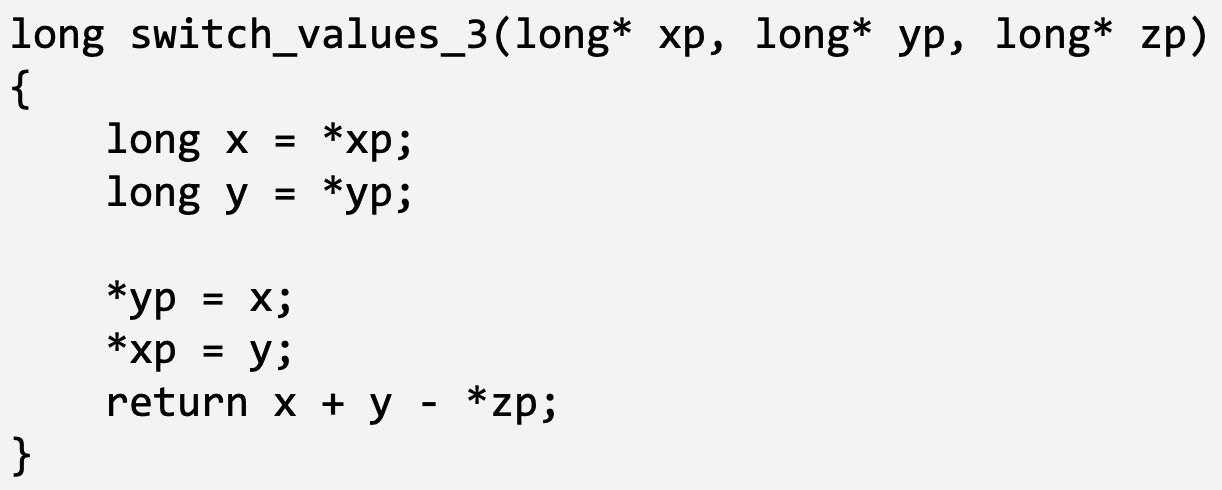
### Spørgsmål 3. Given the following C code and resulting assembly code:

 Et billede, der indeholder tekst, Font/skrifttype, skærmbillede, kvittering

Automatisk genereret beskrivelse

How would the assembly code change if the code changes to a slightly different function?

**If your AU ID’s last digit is even, choose ‘switch\_values\_2’. Otherwise, choose ‘switch\_values\_3’**



Let’s compare the two.

1. The function now returns a long.
2. The value of pointer zp now doesn’t change.
3. The pointer xp now becomes the value of the y pointer instead of the z pointer.

Let’s look at how the assembly now should look like:

- Remains the same

- Remains the same

- Remains the same

- Changed from

-

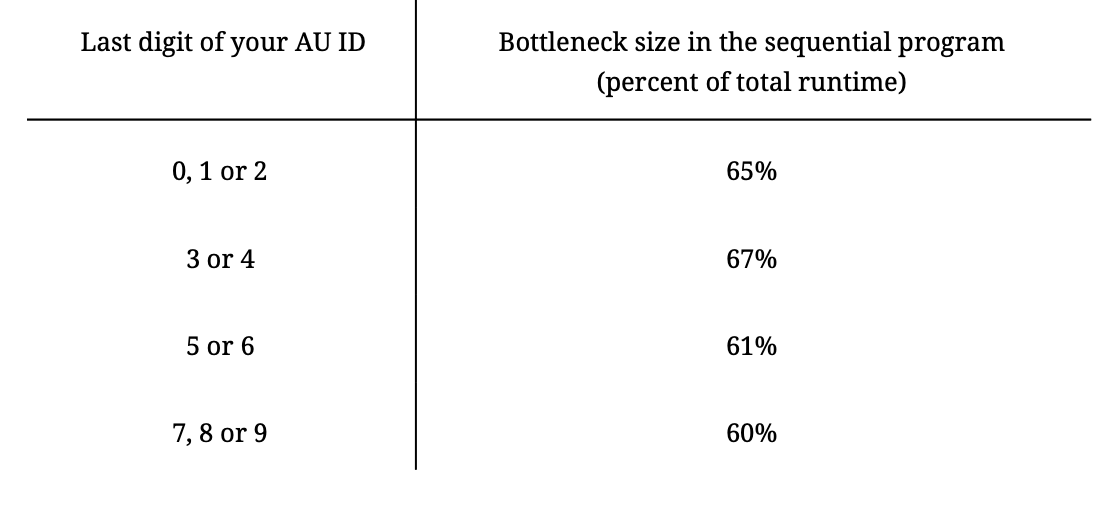
-

is the register to put the return values of a function. In reality one instruction happens at a time, so firstly the x value happens to be in %rax already so nothing to do there. Next y is added to the register, and lastly the value of pointer z is subtracted from the %rax. The function then returns.

### Spørgsmål 4. Optimize your program by changing from sequential to concurrent.

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. Your goal is to speed up the whole program ​**by at least 50%**​. Assume that it’s possible to parallelize the bottleneck code to any degree and you get linear speedup with each new CPU core.

How much speedup can be achieved with an 8 core CPU? At least how many cores does your customer’s computer need in order to to realize the advertised 50% speedup? Use the value from the table below.



For this we use Amdahls law.

Where

*T* Is the total sequential time.

*p* Fraction of the total time that can be sped up.

*k* Sped up factor.

The speed up is then

From the task description we see notice, that the wished optimization is said to be 50%, requiring a speed up of 150%. From the table we notice, that

*Ligningen løses for k vha. WordMat.*

For the program to be optimized by at least 50%, the required cores is above 2.

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

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

Checking for what optimization is possible with 8 cores, we then substitute *k* with 8.

Even with 8 cores, only just more than double the speed can be achieved.

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

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

### Spørgsmål 5. Take the HCL code and convert it to a logic gate diagram.

Et billede, der indeholder tekst, skærmbillede, Font/skrifttype, kvittering

Automatisk genereret beskrivelse

My AU-ID ends with 1:

So this is two parts or’ed.

1. a AND NOT b
2. a AND b



### Spørgsmål 6. Consider a S-way associative cache with the following properties:

Main memory size: 64 bytes  
Cache block size: 4 bytes  
**If your AU ID last digit is odd, S =1 set, E= 8 blocks/set. Otherwise, it is S=2 sets, and E = 4 blocks/set.**

The CPU is reading memory addresses in this order: 0, 3, 4, 2, 8, 14, 2. Which reads cause a cache miss and which are hits?

My last digit of my AU-ID is odd:

Addresses:

000 00

000 11

000 10

001 00

011 10

000 10

M, H, H, M, M, H

Not knowing what’s in the blocks, we are just interested in the tag bits.

Placement policies:

1. Any block at memory level *k* *+ 1* can be placed in any block at memory level *k*. Random.
2. A block *i* at memory level *k + 1* will be placed in block (*i mod Amounts of blocks)*. If *i = 3* and *amounts of blocks = 4*, then *i* % *4 = 3*, and so *i* is placed in block 3.
3. A block at memory level *k + 1* will be placed in the least recently used block.

Empty cache => Cold miss

Storage is large enough to store more blocks, but the reference is to an occupied block => Conflict miss

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Valid** | **Tag** | **LRU** | **Block offset** | | | |
| 1 | 000 |  |  |  |  |  |
|  | 001 |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  | 011 |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Not knowing whether or not the cache started from being empty, I assumed that it was.  
With them all being empty at the start, they all experienced cold misses. One of the addresses was referenced twice which resulted in a hit.

Making the miss ratio be

======

======

With this pattern. Which is probably wrong. When thinking about it, it should rather be, M, H, H, M, M, H not caring about the block offset, making it more like . I will do this in the next task like this one.