Exam number: _____

7-hour Take-Home Exam in Computer Systems, B1-2021/22

Department of Computer Science, University of Copenhagen (DIKU)
Date: January 26, 2022

1 Data Representation and Cache (about 14 %)

1.1 Data representation (about 4 %)

Question 1.1.1: What are the advantages of using two's complement number representation over other integer representations?

We get both positive and negative numbers with a unique 0. At the same time we can use our standard methods for arithmetic that is also used to standard binary representation.

Question 1.1.2: In the IEEE floating point format, why does the inaccuracy of calculations become larger by larger numbers? What part of the format does this come from? What should you as a programmer be aware of because of this?

Floating point numbers have a fixed-sized bit representation. When the exponent becomes larger, the fixed-point fraction does not increase in size. That means that distance between numbers becomes lager, thus, increasing the inaccuracy.
As a programmer you should be aware of this when your arithmetic make the number larger and then smaller again. This can means that any following arithmetic and comparison with small numbers can be faulty as you should expect the accuracy from the large numbers.

Exam number: _____

1.2 Data Cache and Locality (about 10 %)

Consider the following C-like program that adds a row-vector (of length `M`) to each row of a matrix (of size `M`×`N`). We assume some relevant high definition of `M` and `N`.

```
long row_vector_add(long row_vector[M], long matrix[M][N]) {
  int i, j;

  // For each element for the row_vector
  for (i = 0; i < L; i++) {
    // Add the element to each element of a column
    for (j = 0; j < N; j++) {
      matrix[i][j] += row_vector[i]
    }
  }
}
```

Question 1.2.1: What is the stride of the arrays of the program, that is how are they accessed? How does this relate to the temporal and spacial locality.

An 2D array in C is written row-wise in memory and thus cache; row 1 follows by row 2... Given that the indexes in both arrays are written in order, the stride of the program for both is 1.
A low stride will often result in a good spacial locality, while it does not affect temporal locality.

Exam number: _____

Question 1.2.2: Is it possible to improve the *temporal locality* of the program? If not, explain why this is the case. If it is possible, explain how and what the improvement gives.

Given that we don't change the index of `row_vector` we have fine temporal locality.
We can improve the program by reading out `row_vector[i]` to a temporal value before the inner loop.
This does not improve the temporal locality (in the cache understanding), but still improves the running time due to fewer cache accesses.

Question 1.2.3: Is it possible to improve the *spacial locality* of the program? If not, explain why this is the case. If it is possible, explain how and what the improvement gives.

The program has very good spacial locality. We are reading the matrix by iterating over the columns in the inner loop.
The vector is also read in element order.

Exam number: _____

2 Operating Systems (about 32 %)

2.1 Multiple Choice Questions (about 6 %)

In each of the following questions, you may put one or more answers. Use the lines to argue for your choices.

Question 2.1.1: When a process calls `fork()`, the following things are copied (if relevant, specify how you interpret "copied"):

☒ **a)** Register contents

☒ **b)** Process memory

☐ **c)** Threads

☒ **d)** File descriptors

☐ **e)** File system

The copy of process memory is done on-demand in a "copy-on-write" manner, so it depends on how the memory is subsequently used how much is actually physically copied.

Question 2.1.2: Threads running as part of the same process have different:

☒ **a)** Instruction counter

☐ **b)** Virtual memory spaces

☐ **c)** Open files

☒ **d)** Register contents

☐ **e)** Heaps

☒ **f)** Stacks

Exam number: _____

Question 2.1.3:

Consider the C program below. For space reasons, we are not checking error return codes, so assume that all functions return normally.

```
int main () {
  if (fork() == 0) {
    printf("1");
    if (fork() == 0) {
      printf("2");
    } else {
      pid_t pid; int status;
      if ((pid = waitpid(pid, NULL, 0)) > 0) {
          printf("3");
      }
    }
  } else {
    printf("4");
    exit(0);
  }
  printf("5");
}
```

Which of the following strings is a possible output of the program? Place any number of marks.

☒ **a)** 125354

☒ **b)** 412535

☒ **c)** 124535

☐ **d)** 512345

☐ **e)** 412355

Exam number: _____

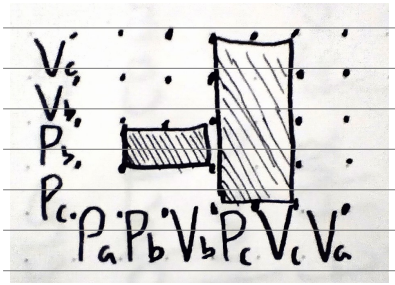
2.2 Short Questions (about 12 %)

Question 2.2.1: Consider two threads performing the following semaphore operations.

```
Initially: a = 1; b = 1; c = 1;

Thread 1:        Thread 2:
P(a);            P(c);
P(b);            P(b);
V(b);            V(b);
P(c);            V(c);
V(c);
V(a);
```

Draw a process graph with thread 1 along the horizontal axis and thread 2 along the vertical axis, show the forbidden regions, and argue whether this means deadlocks are possible or not.



At any point in the graph, there is always a path either up or to the right that avoids entering a forbidden region. Therefore no deadlocks are possible.

Question 2.2.2: Consider a system with the following properties:

- Memory is byte-addressed.
- Virtual addresses are 13 bits wide.
- Physical addresses are 14 bits wide.
- The page size is 64 bytes.
- The TLB is 3-way set associative with 4 sets and 12 total entries. Its initial contents are:

Set	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	11	00	0	11	15	1	04	00	1
1	0A	10	0	0F	10	1	1F	2F	1
2	12	34	1	00	00	0	00	00	1
3	13	15	1	00	12	0	10	0A	1

- The page table contains 12 PTEs:

VPN	PPN	Valid	VPN	PPN	Valid	VPN	PPN	Valid	VPN	PPN	Valid
31	33	1	13	11	1	01	02	1	02	01	1
41	01	1	02	33	0	0B	0D	0	09	17	1
00	00	1	10	21	0	13	32	0	03	43	1

Note that all addresses are given in hexadecimal. In the following questions, you are asked, for various virtual addresses, to show the translation from virtual to physical addresses in the memory system just described. *Hint: there is one TLB hit, one page table hit, and one page fault (not necessarily in that order). This should help you double-check your work.*

Exam number: _____

Virtual address: `0x1192`

1. Bits of virtual address

12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	0	0	1	0	0	1	0

2. Address translation

Parameter	Value
VPN	46
TLB index	2
TLB tag	11
TLB hit? (Y/N)	N
Page fault? (Y/N)	Y
PPN	

3. Bits of phys. (if any)

13	12	11	10	9	8	7	6	5	4	3	2	1	0

Virtual address: `0x1104`

1. Bits of virtual address

12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0	0	0	1	0	0

2. Address translation

Parameter	Value
VPN	44
TLB index	0
TLB tag	11
TLB hit? (Y/N)	Y
Page fault? (Y/N)	N
PPN	15

3. Bits of phys. (if any)

13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	1	0	0	0	1	0	0

Exam number: _____

Virtual address: `0x11`

1. Bits of virtual address

12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	0	0	0	1

2. Address translation

Parameter	Value
VPN	0
TLB index	0
TLB tag	0
TLB hit? (Y/N)	N
Page fault? (Y/N)	N
PPN	0

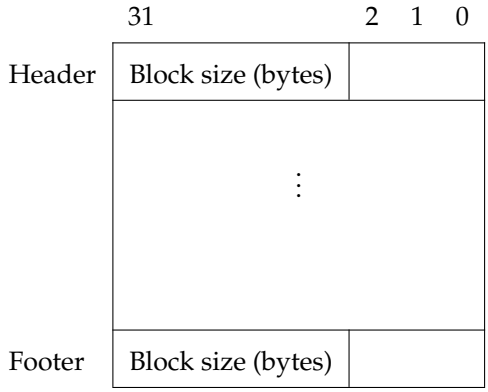
3. Bits of phys. (if any)

13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	1	0	0	0	1

Also answer the following: Describe shortly how you calculated your answers.

2.3 Long Questions (about 14 %)

Question 2.3.1: Consider an allocator that uses an implicit free list and immediate coalescing of neighbouring free blocks. The layout of each allocated and free memory block is as follows, with one 32-bit word per row:



Each memory block, either allocated or free, has a size that is a multiple of eight bytes, rounding up allocations if necessary. Thus, only the 29 higher order bits in the header and footer are needed to record block size, which includes the header and footer. The usage of the remaining 3 lower order bits is as follows:

- `bit 0` indicates the use of the current block: 1 for allocated, 0 for free.
- `bit 1` indicates the use of the previous adjacent block: 1 for allocated, 0 for free.
- `bit 2` is unused and is always set to be 0.

Important: We must *never* create blocks with zero payload (i.e. we must *never* create blocks with size 8).

Given the heap shown on the left, show the new heap contents after *consecutive* calls to

1. `realloc(0x12c000, 8)`. Assume the the return value is `0x12c000`, and that the existing allocation is resized to be as small as possible.
2. `free(0x12c000)`.

Your answers should be given as hex values. Note that the address grows from bottom up. Assume that the allocator uses immediate coalescing, that is, adjacent free blocks are merged immediately each time a block is freed.

Exam number: _____

Address	Original value	After `realloc`	After `free`
0x12c028	0x00000012	0x22	0x32
0x12c024	0x012c611c	0x012c611c	0x012c611c
0x12c020	0x012c512c	0x012c512c	0x012c512c
0x12c01c	0x00000012	0x12	0x12
0x12c018	0x00000023	0x23	0x23
0x12c014	0x012c511c	0x012c511c	0x012c511c
0x12c010	0x012c601c	0x012c601c	0x012c601c
0x12c00c	0x00000000	0x22	0x22
0x12c008	0x00000000	0x13	0x13
0x12c004	0x012c601c	0x012c601c	0x012c601c
0x12c000	0x012c511c	0x012c511c	0x012c511c
0x12bffc	0x00000023	0x13	0x32

Also answer the following: Would it be possible for a free block to start at address `0x12c02C`?

No, because we use immediate coalescing, meaning free blocks are never adjacent, and the initial heap contents show a free block just before the one at `0x12c02C`.

Question 2.3.2: Consider processing N pieces of data with T threads on a system with P processors. Suppose each piece can be processed independently of the others, such that we are free to choose T, and to divide the work as we see fit. For each of the following choices of T, describe a kind of workload (e.g. how long it takes to process a single piece of data) that could plausible result in the lowest (wall clock) runtime with that choice of T:

1. $T = P$
2. $T > P$
3. $T = N$

$T = P$ is likely efficient when each chunk of $N \div P$ data pieces takes the same amount of time, i.e. when the computation is load balanced. $T > P$ could be effective when the difference pieces of data vary widely in how long they take to process, as we can schedule the excess threads on processors that would otherwise be idle. It might also be effective if the processing is heavily dependent on waiting for slow IO operations, as a form of latency hiding. $T = N$ is probably only effective for fairly low N, or when the time taken to process one piece of data is high enough to make thread creation cost negligible.

3 Computer Networks (about 32 %)

3.1 Application Layer (about 8 %)

Question 3.1.1: DNS is a distributed, hierarchical Database. Explain what this means, and how a client request is dealt with by the DNS system. For this answer assume there is no DNS caching.

DNS information is not stored in one location, but a collection of servers worldwide. These exist at different abstraction layers with root servers at the top serving all below. Top-level domains server specific domain names (eg. .com, .dk). Individual organisations have their own DNS servers at the bottom. This makes the system more robust by removing a central failure point, and quicker to respond by having servers closer to worldwide requests.
When a client requests a domain lookup they send a request to their local DNS server. Assuming no cache, the local DNS will not be able to help directly, but will then make a request to the root DNS server. This will parse the domain name to identify which top level domain can parse the request. The root will reply to the local DNS with a TLD DNS server address, and the local DNS sends a request to this. The TLD server parses the authoritative DNS within the domain and sends this address to the local DNS. The Local DNS sends a request to the authoritative DNS and retrieves the IP of the domain, which is sent to the Local DNS. It can finally reply to the client with the IP.

Question 3.1.2: What role does DNS caching play in DNS lookup? Does this benefit iterative or recursive queries more? Justify your answer.

DNS caching allows the different levels of DNS servers to save previous DNS search results. This means that if if gets a request again it can immediately respond without having to make further queries. This speeds up DNS replies. It also means far fewer DNS requests being sent over the network. The process in the previous answer is iterative, but DNS requests may also be recursive such as if the local DNS requested from the root, which requested directly from the TLD which requested directly from the authoritative. In this case if any of these stages had a cache and it hit, messages can be cut. In practice, caching allows for the vast majority of DNS requests to be resolved before the root server needs to be invoked.

Question 3.1.3: Assume you were to connect to a website via HTTP, and that several weeks later you were to connect to the same website. Between the two connections the website has shifted to using a CDN that locates data physically closer to your location. What changes (if any) would you expect in the HTTP requests and responses? Justify your answer and state any assumptions you have made.

The initial client request would be identical as HTTP does not include specific IP addressing, therefore the requests are the same regardless of if a CDN is in use or not.

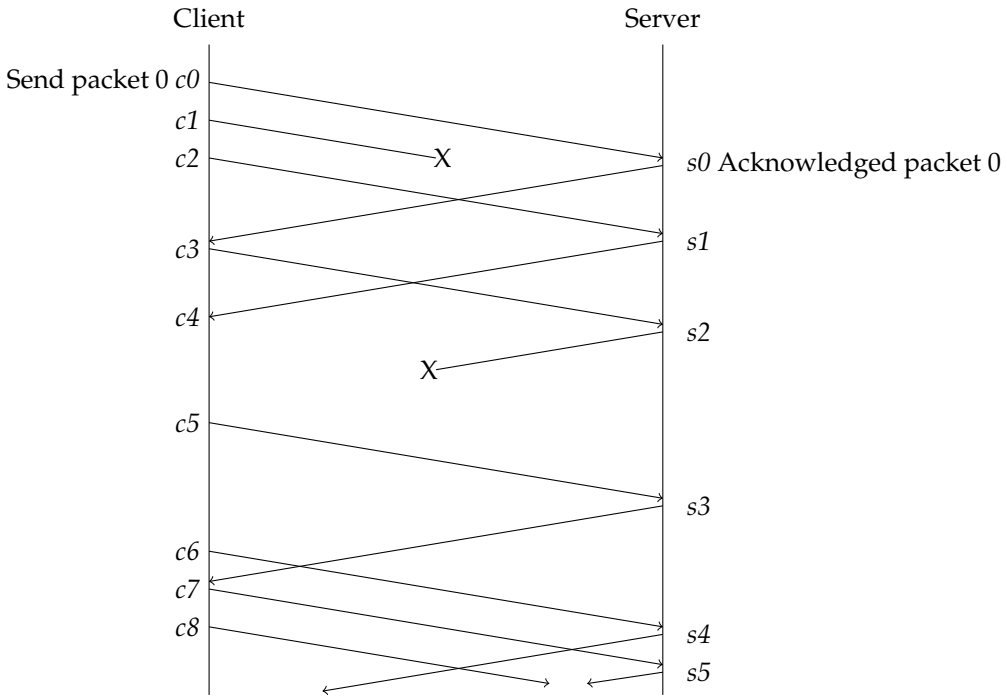
The DNS lookup will be different however depending on if an old address is cached or not. If an old address is cached then the HTTP request will be sent to the old website location in which case the HTTP response will include a 301 Moved Permanently status. The new location should be included in which case a further HTTP request will be generated to that location. This is taken care of automatically by TCP and so again, as far as a user knows their request was identical and still works.

Assuming either no DNS cache hit, or a hit to the newer location the HTTP response would be mostly identical to the response several weeks ago, though it may include different meta-data identifying the replying resource.

Exam number: _____

3.2 Transport Layer (about 8 %)

Consider the following TCP communication diagram, showing a Selective Repeat Protocol, with package losses denoted by an X.



Exam number: _____

Question 3.2.1: Label what is happening at *c1* to *c8*, and *s1* to *s5* in the diagram above. Use the table below to note your answers, and the lines below that to state any assumptions you have had to make, if any.

c0	Sent packet 0
c1	
c2	
c3	
c4	
c5	
c6	
c7	
c8	

s0	Acknowledged packet 0
s1	
s2	
s3	
s4	
s5	

c1=send p1, c2=send p2, c3=rec ak0+send p3, c4=rec ak2, c5=p1 timeout+resend p1, c6=p3 timeout+resend p3, c7=rec ak1+send p4, c8=send p5, s1=ak p2, s2=ak p3, s3=ak p1, s4=ak p3, s5=ak p4

assume sender window is of size 3, and is unchanging in size

Exam number: _____

Question 3.2.2: At *c4* the client receives a message, but there is no immediate follow-up message as happens at *c3* and *c7*. What is happening here, and what can be inferred about the senders window size from this?

Sender is waiting as at c4 its sending window contains packets 1, 2 and 3. All have been sent but ak1 has not been received yet so not further packets can be sent. We need to wait till c5 for p1 to timeout and packets to be resent.

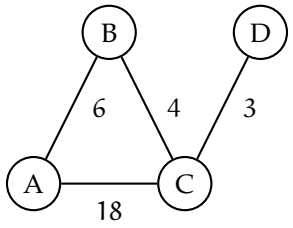
Question 3.2.3: In an ideal world if two TCP connections connect through the same bottleneck router of transmission rate R, each TCP connection will have a transmission rate of R/2. In practice, this is not the case and the rate will osculate around R/2. Why is this? For this question you can assume there are no other connections and that the router will always be the bottleneck for both connections.

congestion control mechanisms in TCP will use less than R/2 initially for both connections. These will each be automatically increased over time as the full throughput is unused an no packet loss occurs. As they each increase the probability of packet loss increases, until the total throughput expected by the combined connections is over R. One, or potentially both of the connections will then cut its connection and start increasing again. As each connection linearly increases then halves its throughput, it is probable that these will occur against each other (eg whilst one is high the other is low and vice versa), and so each will oscillate around R/2.

Exam number: _____

3.3 Network Layer (about 8 %)

Consider the network topology outlined in the graph below



Question 3.3.1: Describe how to apply the Distance Vector Algorithm for Node A in the above diagram, so that it gets a complete routing table to all other Nodes. You do not need to describe each individual step on each Node in detail, but should be able to provide a generalised description that could apply to any Node. For this answer you can assume no connection costs change.

DV is a decentralised algorithm and each node only starting knowing the distance to each of its direct neighbours, with all other values initialised to infinity. The nodes then send this information to each of their neighbours, and will in turn receive the distances from their neighbour. Using these they can calculate the distances to their neighbours neighbours. Sometimes shortcuts are identified such as from A to C, which is quicker when done via B. In this case A's distance to C is updated to the lower value 10. After each change in distance value, all neighbours are informed. This ensures that changes propogate through the system until the lowest cost of each connection is known.

Question 3.3.2: Complete the expected final state of the routing table. For this answer you can assume no connection costs change.

From	Cost To			
	A	B	C	D
A	0	6	10	13
B	6	0	4	7
C	10	4	0	3

Exam number: _____

Question 3.3.3: What is the Count to Infinity problem, and how might it occur in the above network? Demonstrate your answer by identifying a possible location for this to occur in the above diagram.

Count to Infinity is a problem where the cost of a connection dramatically increases. This is usually a sign of a link failure. For instance, if the connection from A to B suddenly increased to 30, then B would detect this change. It would notify its neighbours, including A. To get to A now B only knows that the direct cost to A is 30, but that C still expects to be able to get there at cost 10 (4 + the old value 6). B therefore sends its update to A via C, which will then route it back to B. B and C will then be stuck in a loop routing the packet between each other. In this case it is finite, but if the connection cost had increased dramatically then the needless loop will take significantly longer.

3.4 Network Security (about 8 %)

Question 3.4.1: What is a non-cryptographic hash function, and how does it help ensure the integrity of network communication?

Hash functions are a way of changing an arbitrarily sized piece of data to a fixed length, in a such a way that the the product is (practically speaking) distinct for diverse inputs, and that the same input produces the same output every time. By recomputing a hash function repeatedly on a piece of data you can detect changes, as this will be reflected in a differing output. Therefore, you can compute the hash of some data before it is sent, and include the product in the message. The receiver can then compute the hash of the received data and compare the result with the hash contained in the message. If they are the same, then we can say that all data have been received correctly. As opposed to cryptographic hashes, these hashing algorithms are relatively computationally simple and so much quicker to compute. They are also easier to reverse engineer so are not suitable for encryption.

Question 3.4.2: Consider a network application that uses asymmetric, public-key encryption on every communication. What would you expect the shortcomings of this system to be? Suggest some potential improvements, and what effect(s) you would then expect to see.

This system would probably be slow at communication, as public key encryption takes a long time to compute. If this is being done on every message there will be a very high overhead. Within an individual session it is more normal to use the initial asymmetric communication to transmit a shared secret, which can then be used for symmetric communication for the duration of the session. Symmetric algorithms are much quicker to compute, so security can be maintained with less overhead.

Exam number: _____

Question 3.4.3: A nonce is a common solution to a playback attack. What is a playback attack, and how does the nonce solve this problem?

A playback attack is when someone listens to network communications and takes a copy of a clients message to a server. They will then resend this message to the server in the hope of immitating the client and so gaining whatever response the server sends. A nonce is way around this as the server will generate this (practically) unique number which will be used throughout this session. If an attacker copies a message and resends it, the unique number will be included. The server can then identify that this is part of an old session as so should not respond to the message.

Exam number: _____

4 Machine architecture (about 24 %)

4.1 Assembler programming (about 12 %)

Consider the following program written in X86prime-assembler.

```
.START:
    subq $8, %rsp
    movq %r11, (%rsp)
    movq $1, %r8
    jmp .L2
.L1:
    leaq 8(%rdi, %rax, 8), %r11
    movq %rcx, (%r11)
    addq $1, %r8
.L2:
    cbl %rsi, %r8, .L4
    leaq (%rdi, %r8, 8), %r11
    movq (%r11), %rcx
    leaq -1(%r8), %rax
.L3:
    cbg $0, %rax, .L1
    leaq (%rdi, %rax, 8), %r11
    movq (%r11), %rdx
    cbge %rcx, %rdx, .L1
    leaq 8(%rdi, %rax, 8), %r11
    movq %rdx, (%r11)
    subq $1, %rax
    jmp .L3
.L4:
    movq (%rsp), %r11
    addq $8, %rsp
    ret %r11
```

Exam number: _____

Question 4.1.1: Identify the control structures of the program (e.g. loops, conditionals, and function calls).

Vi har en ydre løkke fra L26 -> L29 og inden i en indre løkke fra L25 -> L29.
Endvidere er der et betinget exit (break) fra den indre løkke. Alternativt kan man opfatte det som om den indre løkke har en kompliceret betingelse. Det bliver tydeligere nedenfor prolog = kode for at opsætte stakramme til funktionen
epilog = kode for at nedtage stakramme for funktionen

Se side 28 for detaljer.

Question 4.1.2: Specify which registers are used for pointers. How can you see this? What are the operations?

%rsp, %rdi, %r11

Question 4.1.3: Which registers contains functions arguments and what are their type? Describe how you identified this.

%rdi, %rsi

Question 4.1.4: Rewrite the above X86prime-assembler program to a C program. The resulting program must not have a goto-style and minor syntactical mistakes are acceptable.

```
void sort(long num_elem, long array[]) {
  for (long i = 1; i < num_elem; ++i) {
    long x = array[i];
    long j = i - 1;
    while (j >= 0 && array[j] > x) {
      array[j + 1] = array[j];
      --j;
    }
    array[j + 1] = x;
  }
}
```

Se side 29 for detaljer.

Question 4.1.5: Descripe shortly the functionality of the program.

Funktionen er insertionsort.
Den leverede tabel sorteres gradvist. Den ydre løkke gennemløber alle elementerne et for et og udvider den sorterede del af tabellen. Den indre løkke placerer hvert element det rigtige sted blandt de sorterede elementer.

4.2 Pipeline and instruction level parallelism (about 10 %)

Consider the following fragment of a program

```
Loop:   movq (%r12),%r15
        movq %r15,(%r13)
        addq $8,%r12
        addq $8,%r13
        cbne $0,%r15,Loop
```

The following execution graph (afviklingsplot) illustrates the execution sequence of two iterations on a simple 5-step pipeline machine, where the instructions have to wait for all operants to be ready in the D-step:

```
Loop:   movq (%r12),%r15    FDXMW
        movq %r15,(%r13)     FDDXMW
        addq $8,%r12          FFDXMW
        addq $8,%r13             FDXMW
        cbne $0,%r15,Loop         FDXMW
Loop:   movq (%r12),%r15           FDXMW
        movq %r15,(%r13)            FDDXMW
        addq $8,%r12                   FDXMW
        addq $8,%r13                    FDXMW
        cbne $0,%r15,Loop                FDXMW
```

Unless explicitly stated, the following questions assumes that all jumps are correctly predicted and all access to the memory results in a cache hit.

Question 4.2.1:

It is possible to make a change to the pipeline such that values that are written to the memory should now be ready before the X-step instead of the D-step as shown above. This can affect instruction 2 and 7 in the above execution graph.

How much would this change reduce the execution of the above loop?

Ja, der er ikke brug for en stall i `movq %r15,(%r13)`.

Udførelsestiden er således reduceret med (6-5)/6 = 1/6.

Exam number: _____

Question 4.2.2: Make an execution diagram, showing the execution of above code on a two-way superscalar machine, as presented in the course note. (See on the title "Eksempel: Superskalar".) Explain shortly any assumptions you may have to make.

	Code	Timing																	
	`Loop:`																		
1	`movq (%r12),%r15`																		
2	`movq %r15,(%r13)`																		
3	`addq $8,%r12`																		
4	`addq $8,%r13`																		
5	`cbne $0,%r15,Loop`																		
	`Loop:`																		
6	`movq (%r12),%r15`																		
7	`movq %r15,(%r13)`																		
8	`addq $8,%r12`																		
9	`addq $8,%r13`																		
10	`cbne $0,%r15,Loop`																		

Der accepteres to forskellige svar her. Det skyldes beskrivelsen af superskalare processorer som findes i noten om afviklingsplot, og så den beskrivelse der er givet i forelæsninger og på slides og i vejledning over discord i løbet af kurset kan give anledning til forskellige fortolkninger.

Noten om afviklingsplot angiver at tilgang til lageret kræver to clock-perioder i 'M'-trinnet. Dette er egentlig en fejl, sådan at forstå, at selvom en sådan maskine kunne bygges, så vil det ikke være det typiske valg. I virkelighedens maskiner er der kun en clock-periode i 'M' trinnet - hvilket var hvad undervisningen i efteråret i øvrigt antog.

Se 31 for plots.

Begge løsninger og forståelse af ændringer til en superskalar maskine giver kredit.

Below you see the execution graph for the code on a more realistic modern 3-way out-of-order machine, where access to the primary cache is extended over 3 clock-periods and where values that are written to memory should only be ready in the X-step:

```
Loop:   movq (%r12),%r15     F----QAM--C
        movq %r15,(%r13)     F----Q-AM-VC
        addq $8,%r12         F----QX----C
        addq $8,%r13          F----QX---C
        cbne $0,%r15,Loop     F----Q---B-C
Loop:   movq (%r12),%r15       F----QAM--C
        movq %r15,(%r13)       F----Q-AM-VC
        addq $8,%r12           F----QX----C
        addq $8,%r13            F----QX---C
        cbne $0,%r15,Loop       F----Q---B-C
```

(The machine presented in the note has a 2-cycle fetch redirection delay when predicting a taken branch. As you can see above, the machine used here has only a single cycle delay. This is not an error)

Question 4.2.3: How does this machine compare in clock-periods per iteration to the simple 5-step pipeline machine presented in the beginning of the section?

Her bruges 2 clock perioder per iteration mod 6 i den allerførste del af opgaven.

Question 4.2.4: Assume that 10% of jumps is predicted wrongly, while the rest (90%) is predicted correctly.

How many clock-periods does every iteration of the loop takes on average? Explain how you found your answer.

Her er det nødvendigt at fastlægge omkostningen for et gennemløb med en fejlforudsigelse. Efter en fejlforudsigelse kan den efterfølgende F fase først ske efter B-fasen for hop instruktionen, da det er på det tidspunkt fejlforudsigelsen detekteres.

```
    Loop: movq (%r12),%r15 F----QAM--C
          movq %r15,(%r13) F----Q-AM-VC
          addq $8,%r12 F----QX----C
          addq $8,%r13 F----QX---C
          cbne $0,%r15,Loop F----Q---B-C
    Loop: movq (%r12),%r15 F----QAM--C
                       ....
```

Man kan se at hver 10nde gennemløb (som ses ovenfor) tager 11 clock perioder. De resterende 9 gennemløb tager stadig 2 cykler.

Den gennemsnitlige tid for et gennemløb bliver da (11+9*2)/10 = 2.9 clock perioder.

Exam number: _____

Vedr. 4.1.1

```
.LFB10:
    subq $8, %rsp                                  prolog
    movq %r11, (%rsp)                              prolog
    movq $1, %r8
    jmp .L24                                       ----\
.L26:                                              <-- | ---\---\
    leaq 8(%rdi, %rax, 8), %r11                        |    |   |
    movq %rcx, (%r11)                                  |    |   |
    addq $1, %r8                                       |    |   |
.L24:                                              <---/    |   |
    cbl %rsi, %r8, .L29                            -------  | - | --\
    leaq (%rdi, %r8, 8), %r11                             |   |   |
    movq (%r11), %rcx                                     |   |   |
    leaq -1(%r8), %rax                                    |   |   |
.L25:                                              <---\    |   |   |
    cbg $0, %rax, .L26                             --  |  --/   |   |
    leaq (%rdi, %rax, 8), %r11                            |        |   |
    movq (%r11), %rdx                                     |        |   |
    cbge %rcx, %rdx, .L26                          --  |  ------/   |
    leaq 8(%rdi, %rax, 8), %r11                           |            |
    movq %rdx, (%r11)                                     |            |
    subq $1, %rax                                         |            |
    jmp .L25                                       ----/              |
.L29:                                              <------------------/
    movq (%rsp), %r11                              epilog
    addq $8, %rsp                                  epilog
    ret %r11                                       epilog
```

Vedr. 4.1.4
Vi starter med at omskrive til pseudo-C med goto og registernavne:

```
    prolog.
    r8 = 1;
    goto L24
L26:
    rdi[rax] = rcx
    ++rax
L24:
    if rsi < r8 goto L29
    rcx = rdi[r8]
    rax = r8 - 1
L25:
    if 0 > rax goto L26
    rdx = rdi[rax]
    if rcx > rdx goto L26
    rdi[rax] = rdx
    --rax
    goto L25
L29:
    epilog.
```

Den ydre løkke starter ved L24 (kan vi se fordi der øverst hoppes til L24) Vi flytter nu blokken fra L26 til nederst i den ydre løkke, således at L24 ligger øverst - det gør strukturen mere indlysende og får løkkebetingelsen op i toppen:

```
    prolog.
    r8 = 1;
L24:
    if rsi < r8 goto L29
    rcx = rdi[r8]
    rax = r8 - 1
L25:
    if 0 > rax goto L26
    rdx = rdi[rax]
    if rcx > rdx goto L26
    rdi[rax] = rdx
    --rax
    goto L25
L26:
    rdi[rax] = rcx
    ++rax
    goto L24
L29:
    epilog.
```

Fortsætter på næste side...

Vi kan nu omskrive den ydre løkke:

```
    prolog.
    r8 = 1;
    while (rsi >= r8) {
        rcx = rdi[r8]
        rax = r8 - 1
L25:
        if 0 > rax goto L26
        rdx = rdi[rax]
        if rcx > rdx goto L26
        rdi[rax] = rdx
        --rax
        goto L25
L26:
        rdi[rax] = rcx
        ++rax
    }
    epilog.
```

Så den indre:

```
    prolog.
    r8 = 1;
    while (rsi >= r8) {
        rcx = rdi[r8]
        rax = r8 - 1
        while (0 <= rax) {
            rdx = rdi[rax]
            if (rcx > rdx) break;
            rdi[rax] = rdx
            --rax
        }
        rdi[rax] = rcx
        ++rax
    }
    epilog.
```

Og endeligt i rigtig C:

```
void sort(long num_elem, long array[]) {

  for (long i = 1; i < num_elem; ++i) {
    long x = array[i];
    long j = i - 1;
    while (j >= 0 && array[j] > x) {
      array[j + 1] = array[j];
      --j;
    }
    array[j + 1] = x;
  }
}
```

Det er også fint at give en løsning der stadig bruger "break" i stedet for at omskrive til en mere kompliceret betingelse for den indre løkke.
Bemærk at læsningen fra array[j] i den indre løkke sker to gange i C kildeteksten, men af compileren er blevet optimeret til en enkelt tilgang i maskinkoden.

Vedr. 4.2.2
Hvis man følger noten om afviklingsplot får man:

```
Loop:   movq (%r12),%r15     FDXMMW
        movq %r15,(%r13)     FDXXXMM
        addq $8,%r12          FDXXW
        addq $8,%r13          FDDDXW
        cbne $0,%r15,Loop      FDDX
Loop:   movq (%r12),%r15        FFDXMMW
        movq %r15,(%r13)        FFDXXXMM
        addq $8,%r12               FDXXW
        addq $8,%r13               FDDDXW
        cbne $0,%r15,Loop           FDDX
```

Hvis man derimod følger den øvrige undervisning får man:

```
Loop:   movq (%r12),%r15     FDXMW
        movq %r15,(%r13)     FDXXM
        addq $8,%r12          FDXW
        addq $8,%r13          FDDXW
        cbne $0,%r15,Loop      FDX
Loop:   movq (%r12),%r15        FDXMW
        movq %r15,(%r13)        FDXXM
        addq $8,%r12             FDXW
        addq $8,%r13             FDDXW
        cbne $0,%r15,Loop         FDX
```

Vi antager i begge tilfælde at et korrekt forudsagt taget hop blot forsinker F med 1 clock.