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

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

Exam number: 284

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?

Man behøver ikke at implementere nogen speciel logik, for at håndtere negative tal i two's complement, som man skal i sign-and-magnitude. Two's complement behøver man heller ikke have bruge en end-around carry værdi, når man laver addition, som man skal i one's complement. Derudover har one's complement en positiv og en negativ værdi for 0, som kan være problematisk. Det er der ikke i two's complement.

(Maximum 8 lines.)

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?

I IEEE floating point format, er der kun et bestemt antal bits til at representere alle tal. De tal der kan representeres er derfor blevet spredt ud, og andre til vil blive "afrundet" til det nærmeste tal, som kan representeres. De fleste af de representerbare tal er sat omkring 0, og der bliver større og større afstand mellem disse tal, jo længere væk fra 0 man kommer. Store tal vil derfor have en større fejl, da de kan risikere at bliver "afrundet" mere.

Som programør skal man derfor passe på med at lave mange udregning på store tal floating point tal, da der kommer større og større afvigelse fra det rigtige tal, jo flere beregninger man laver. Derudover skal man også passe på, hvis man konvertere floating point tal til integers, da fx 4, kan være representeret som 3.999..., og derfor vil blive rundet ned til 3 i nogle programmerings sprog. Derudover skal man også passe på med sammenligninger med disse tal, da der også kan være afvigelser i disse. Dette kan løses med et machine epsilon.

(Maximum 12 lines.)

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.

Vi går ud fra L skulle have været M i opgaven.
Matricen i koden er tilgået med en unit stride. Dette kan ses, da det indre loop er tilgået row-major. Det er også sådan arrays i c normalt er gemt. Dette giver god spatial locality, da hver iteration af loopet tilgår den del af memory, som er lige ved siden af den forrige iteration af loopet.

Vektoren er også tilgået med en unit stride, da den ydre løkke inkrementeres med hver gang. Dette giver også god spaciallocality.

Da den indre løkke ikke ændre på i, har funktionen også god temporal locality, da vi ikke springer frem og tilbage i vektoren, men kun kigger på den i'te værdi j gange, hvorefter vi går videre til den næste, og ikke vender tilbage.

(Maximum 12 lines.)

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.

Temporal locality kan ikke forbedres i denne funktion, da vektoren bliver tilgået element for element, og det er det samme element, som bliver tilgået i den indre løkke. Elementerne i matricen bliver kun tilgået en gang, så der kan man heller ikke forbedre temporal locality.

(Maximum 12 lines.)

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.

Spacial locality kan ikke forbedres i dette program, da vi altid tilgår elementer lige ved siden af det forrige i løkkerne.

(Maximum 12 lines.)

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
- [X] **b)** Process memory
- [ ] **c)** Threads
- [X] **d)** File descriptors
- [X] **e)** File system

(Maximum 5 lines.)

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

- [X] **a)** Instruction counter
- [X] **b)** Virtual memory spaces
- [ ] **c)** Open files
- [X] **d)** Register contents
- [ ] **e)** Heaps
- [X] **f)** Stacks

(Maximum 5 lines.)

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

(Maximum 5 lines.)

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.

(Maximum 10 lines.)

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.*

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	0x46
TLB index	0x2
TLB tag	0x11
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	0x44
TLB index	0x0
TLB tag	0x11
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

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	1	0	0	1

2. Address translation

Parameter	Value
VPN	0x0
TLB index	0x0
TLB tag	0x0
TLB hit? (Y/N)	N
Page fault? (Y/N)	N
PPN	0x00

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

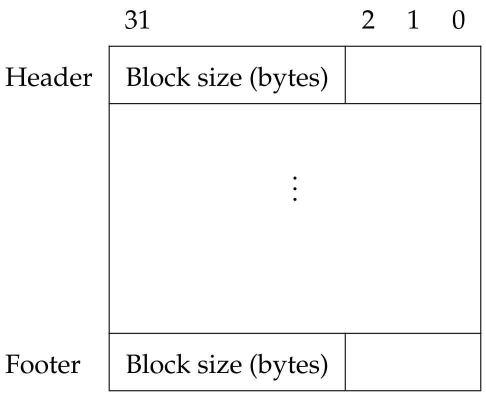
Also answer the following: Describe shortly how you calculated your answers.

Størrelsen af VPO = log[2](P) = 6, hvor P=64 er page size
VPO er derfor bits 0 til og med 5
Størrelsen af VPO = log[2](P) = 6, hvor P=64 er page size
VPN er derfor bits 6 til og med 12
Størrelsen af TLB index = log[2](antal sets) = 2 bits
TLB indexet er de to første bits af VPN
TLB taget er resten af VPN, som er 5 bits

(Maximum 15 lines.)

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.

Address	**Original value**	**After `realloc`**	**After `free`**
`0x12c028`	`0x00000012`	0x10	0x10
`0x12c024`	`0x012c611c`	`0x012c611c`	`0x012c611c`
`0x12c020`	`0x012c512c`	`0x012c512c`	`0x012c512c`
`0x12c01c`	`0x00000012`	0x10	0x10
`0x12c018`	`0x00000023`	0x23	0x23
`0x12c014`	`0x012c511c`	`0x012c511c`	`0x012c511c`
`0x12c010`	`0x012c601c`	`0x012c601c`	`0x012c601c`
`0x12c00c`	`0x00000000`	0x0	0x0
`0x12c008`	`0x00000000`	0x13	0x12
`0x12c004`	`0x012c601c`	`0x012c601c`	`0x012c601c`
`0x12c000`	`0x012c511c`	`0x012c511c`	`0x012c511c`
`0x12bffc`	`0x00000023`	0x13	0x12

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

Vi ved ikke om blocken over er allokeret, og vi ved derfor ikke om vi kan free. Hvis den er allokeret kan vi godt, ellers kan vi ikke.

(Maximum 10 lines.)

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. $T = N$

(Maximum 20 lines.)

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.

Der er mange DNS serverer rundt omkring i verden, i steder for en store server et sted.
Hierakiet er sat op således:

Root DNS -> Top Level DNS-> Authoritative DNS

Hvis vi fx kigger på en client, som godt kunne tænke sig at tilfå absalon.ku.dk, starter clienten med at sende en request til en root DNS server, som kigger på domænet, som i dette tilfælde er .dk. Den sender så IP-adressen på en DNS server, som håndtere requests med domænet .dk tilbage til clienten, som sender en ny request til en TLD server. Denne TLD server kigger så på næste del af adressen som er .ku, og finder en DNS. Denne besked for clienten så tilbage, sender en ny request til den nye DNS server, som er en Authoritative DNS server. Denne DNS server sender så IP-adressen på hjemmesiden tilbage til clienten, som så kan forbinde til denne IP-adresse, og tilfå hjemmesiden.

(Maximum 16 lines.)

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

Lokale DNS server kan cache de request de får, og kan derfor sende hurtigt svar, hvis der kommer flere af de samme request.
DNS caching er gode i recursive queries, da alle levels af DNS hierakiet kan cache disse requests. Det har ikke rigtigt nogen fordel i iterative queries, da hver server kun kigger på hvad den selv har, og så sender requesten videre til en anden DNS server, hvis den ikke har svaret. Den for derefter ikke information om, hvad den anden DNS server har svaret.

(Maximum 12 lines.)

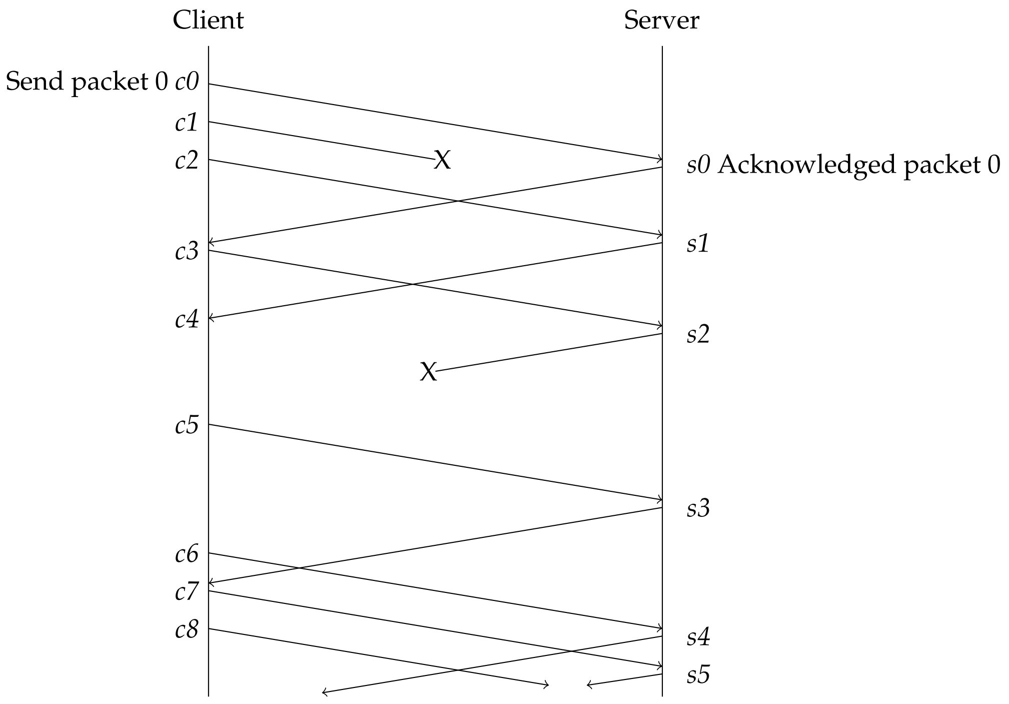
Question 3.1.3: Assume you were to connect to a website via HTTP, and that several weeks later you were to connect to the same website. Between the two connections the website has shifted to using a CDN that locates data physically closer to your location. What changes (if any) would you expect in the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

Da serveren er tættere på, vil der være mindre forsinkelse, og hjemmesider vil derfor loade hurtigere, da downloading vil blive håndteret hurtigere, og da HTTP request vil blive håndteret hurtigere.

(Maximum 12 lines.)

3.2 Transport Layer (about 8 %)

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



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	Sent packet 1
c2	Sent packet 2
c3	Recieve acknowledgment 0, then Sent packet 3
c4	Recieve acknowledgment 2
c5	Sent packet 5
c6	Sent packet 6
c7	Recieve acknowledgment 5, then Sent packet 7
c8	Sent packet 8

s0	Acknowledged packet 0
s1	Acknowledged packet 2
s2	Acknowledged packet 3
s3	Acknowledged packet 5
s4	Acknowledged packet 6
s5	Acknowledged packet 7

(Maximum 5 lines.)

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?

(Maximum 8 lines.)

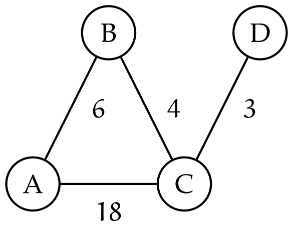
Question 3.2.3: In an ideal world if two TCP connections connect through the same bottleneck router of transmission rate R, each TCP connection will have a transmission rate of R/2. In practice, this is not the case and the rate will osculate around R/2. Why is this? For this question you can assume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

De to forbindelser, vil følges ad og stille og roligt øge deres transmission rate indtil de overskrider den mulige transmission rate af routeren, da der kommer et packet loss. De to forbindelser vil derefter halvere deres transmission rate, og så igen stille og roligt øge den til de når grænsen igen.

(Maximum 16 lines.)

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.

Distance Vector Algorithm starter med at lave en vektor for hver node, med deres respektive distancer til hver anden node, som de direkte kan tilgå. Derefter bliver denne vektor sendt til de andre noder. Noderne beregner nu igen en ny vektor, med de nye korteste distancer, hvorefter de sender den ud igen. Dette forsætter til der ikke er nogen noder, som ændre på deres vektor.

(Maximum 12 lines.)

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

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.

Der kan ske et Count to Infinity problem, hvis kosten mellem to node øges. Det kan fx ske hvis kosten mellem (B,C) øges til fx 60. Hvis A vil sende en besked til D, tror den, den kortest vej er at gå til B. Da B har fået opdateret sin værdi til C og til D, kan den se, den hurtigste vej til D burde være igennem A, da A ikke har fået informationen om distancen (B,C). B opdatere sin værdi til D og sender denne besked tilbage til A. A og B kan så sende disse besked frem og tilbage indtil A kan se, den hurtigste vej vil være igennem C, hvorefter alle vektorene vil blive opdateret med de nye værdier. I nogle tilfælde kan dette blive en meget "dyr" proces.

(Maximum 10 lines.)

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?

Det smarte ved non-cryptographic hash funktioner er, at man kan beregne en værdi for fil, som vil være specifik til den fil. Denne værdi kan nu sendes med til modtageren af filen, som til sidst kan udregne hashen af den modtagede fil, og sammenligne med den hash der er blevet sendt. Dette er meget mere effektivt end at inkryptere og dekryptere hele filen.

(Maximum 15 lines.)

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.

Alle kan gå ind og lade som om de er en anden person og være en del af samtalen. En forbedring kunne være at bruge en cryptographic hash funktion i stedet for, hvor man har en public og en secret key i stedet for kun en public key. Dette vil øge sikkerheden markant.

(Maximum 10 lines.)

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?

Et playback attack er, når en person C fanger en besked mellem en person A og en person B. Personen C kan nu lade som om de er person A.
En nonce er en arbitrært tal som kun kan bruges en gang i en protocol. Hvis person A sender, kan B sende en nonce tilbage til A. A kan nu kryptere noncenog sende den afsted hvorefter B kan validere integriteten af besked ved at se på noncen. Hvis ikke det er den rigtige krypterede nonce, ved B der er noget galt.

(Maximum 10 lines.)

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
```

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

Vi kan se der er:
- en ydre løkke
- en indre løkke
- et if-statement, som ligger inden i den indre løkke

(Maximum 8 lines.)

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

%rsp er stackpointeren. Det kan man se i den første linje i programmet
%rdi er en pointer til starten på et array. Det kan ses, da vi har en counter, som vi bruger til at inkrementere i arrayet med 8 bits hver gang fx i linje 1 i L2
%r11 er returadressen, og så bruges den derudover, som en midlertidig pointer til at pointe til et bestemt index i arrayet, hvor indexet er bestemt af vores counter fx i linje 2 i L2

(Maximum 8 lines.)

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

%rdi er en pointer til starten på et array, som er et argument til vores funktion. Det kan vi se, da vi arbejder med %rdi registret inden vi tildeler den en værdi
%rsi er et argument, som indeholder længen på vores array, som er en long. Vi kan se det er en long, da vi sammeligner %rsi værdien med vores counter. Vi kan se det er et argument, da vi arbejder på %rsi uden at tildele den en værdi.

(Maximum 8 lines.)

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 insertionSort (long* array, long length) {
    long i = 1;
    while (i < length) {
        long indexI = array[i];
        for (long j = ii - 1; j > 0; j--) {
            indexJ = array[j];
            if (indexJ <= indexI) {
                break;
            }
            array[j+ 1] = indexI;
        }
        array[j + 1] = indexI;
        i++;
    }
}
```

(Maximum 20 lines.)

Question 4.1.5: Descripe shortly the functionality of the program.

Programmet lavet insertion sort

(Maximum 10 lines.)

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?

(Maximum 5 lines.)

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`	

(Maximum 10 lines.)

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?

CPI'en for den første vil være 16/10, mens CPI for den nye vil være 15/10, og denne maskine er derfor lidt hurtigere.

(Maximum 8 lines.)

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.

Vi antager, at maskinen først finder ud af den har lavet et forkert hop efter Q, og at den næste F, vil starte efter dette. Derudover antager vi, at vi køre funktionen igennem rigtigt mange gange, så CPI'en vil tilnærme sig CPI'en for et gennemløb af løkken i midten.
CPI ved korrekt gæt = 2/5
Korrekt % = 0.9
CPI ved forkert gæt = 7/5
forkert % = 0.1
Average CPI = CPI_k * %_k + CPI_f * %_f = 2/5 * 0.9 + 7/5 * 0.1 = 0.5

(Maximum 8 lines.)