4-hour Written Exam in Computer Systems

Department of Computer Science, University of Copenhagen (DIKU)
Date: January 23, 2019

Preamble

This is the exam set for the 4 hour written exam in Computer Systems (CompSys), B1+2-2018/19. This document consists of 23 pages excluding this preamble; make sure you have them all. Read the rest of this preamble carefully. Your submission will be graded as a whole, on the 7-point grading scale, with external censorship.

- You can answer in either Danish or English.
- Remember to write your exam number on all pages.
- You do not have to hand-in this preamble.

Expected usage of time and space

The set is divided into sub-parts that each are given a rough guiding estimate of the size in relation of the entire set. However, your exact usage of time can differ depending on prior knowledge and skill.

Furthermore, all questions includes formatted space (lines, figures, tables, etc.) for in-line answers. Please use these as much as possible. The available spaces are intended to be large enough to include a satisfactory answer of the question; thus, full answers of the question does not necessarily use all available space.
If you find yourself in a position where you need more space or have to redo (partly) an answer to a question, continue on the backside of a paper or write on a separate sheet of paper. Ensure that the question number is included and that you in the in-lined answer space refers to it; e.g. write *"The [rest of this] answer is written on backside of/in appended page XX."*

For the true/false and multiple-choice questions with one right answer give only one clearly marked answer. If more answers are given, it will be interpreted as incorrectly answered. Thus, if you change your answer, make sure that this shows clearly.

Exam Policy

This is an *individual*, open-book exam. You may use the course book, notes and any documents printed or stored on your computer, but you may not search the Internet or communicate with others to answer the exam.

Errors and Ambiguities

In the event of errors or ambiguities in the exam text, you are expected to state your assumptions as to the intended meaning in your answer. Some ambiguities may be intentional.

Exam number:

IMPORTANT
It is important to consider the context and expectations of the exam sets. Each question is designed to cover one of more learning goals from the course. The total exam set will cover all or (more likely) a subset of all the learning goals; this will vary from year to year.
The course has many more learning goals than are realistic to cover during a 4-hour exam. And even though we limit the tested learning goals, it will still be too much.
Therefore, it is not expected that you give full and correct answer to all questions. Instead you can focus on parts in which you can show what you have learned.
It is, however, important to note that your effort will be graded as a whole. Thus, showing your knowledge in a wide range of topics is better that specialising in a specific topic. This is specially true when considering the three overall topics: Arc, OS, and CN.

Specifically, for this exam set it was need to solve:
* about 40 % to pass
* about 55 % to get a mid-range grade
* about 75 % to get a top grade
* no one solved all parts correctly!
NB! we adjust the exam set from year to year, so the above is not written in stone and can change slightly for your exam.

4-hour Written Exam in Computer Systems
Department of Computer Science, University of Copenhagen
Date: January 23, 2019

Exam number:

1 Machine architecture (about 33 %)

1.1 True/False Questions (about 3 %)

For each statement, answer True or False. (Put one "X" in each.)	True	False
a) Within Boolean arithmetic then A ^ B = (A | ~B) & (~A | B).		
b) In the 32-bit two's-complement integer representation there are more even than odd numbers (due to the representation of zero).		
c) In the IEEE 754 floating point format, adding two normalised numbers can result in a denormalised number.		
d) In X86_64, the length (number of used bits) of an instruction can vary between different instructions.		
e) In a function call on the Linux machine, all arguments after the sixth are passed on the call stack.		
f) In machines with separated L1-instruction and L1-data caches (e.g. Intel's Core machines), call-prediction ensures that the next instruction always is available in the L1-instruction cache.		

1.2 Short Answer Questions (about 5 %)

Short Answer Questions, 1.2.1: Explain the advantages of having denormalised numbers in the IEEE 754 floating point format.

(Maximum 8 lines.)

Exam number:

Short Answer Questions, 1.2.2: Briefly explain how *temporal* locality improves performance of programs.

(Maximum 8 lines.)

Short Answer Questions, 1.2.3: Explain how the depth of the pipeline (number of pipeline stages) in a microarchitecture affects branch prediction.

(Maximum 8 lines.)

1.3 Assembler programming (about 10 %)

Consider the following program written in X86prime-assembler.

The following program have been updated from x86-64 assembler to x86prime.

```
.program:
    subq $8, %rsp
    movq %r11, (%rsp)
    movq $1, %eax
    movq $0, %edx
    jmp .L1
.L2:
    addq $1, %rax
.L1:
    cble %rsi, %rax, .L3
    leaq (%rdi, %rax, 8), %r11
    movq (%r11), %rcx
    leaq (%rdi, %rdx, 8), %r11
    movq (%r11), %r10
    cble %rcx, %r10, .L2
    movq %rax, %rdx
    jmp .L2
.L3:
    leaq (%rdi, %rdx, 8), %rax
    movq (%rsp), %r11
    addq $8, %rsp
    ret %r11
```

Assembler programming, 1.3.1: Rewrite the above X86prime-assembler program to a C program. The resulting program should not have a goto-style and minor syntactical mistakes are acceptable.

(Maximum 12 lines.)

Assembler programming, 1.3.2: Explain the functionality of the program and your choice of statements.

Exam number:

(Maximum 10 lines.)

1.4 Pipeline (about 10 %)

Below you find a code fragment written in x86prime; it is equivalent to x86 except for the inclusion of the branch-if-less-than instruction, `cble`. The code fragment shows the inner loop of a function that copies all values from one array to another. Register `%r10` contains the pointer to the source array, `%r11` the pointer to the destination array, and `%r12` contains the total number of elements in the arrays.

```
    Code                         Execution on 5-stage pipeline

.Loop:
  cble $0, %r12, .Done     FDXMW
  movq (%r10), %r9          FDXMW
  movq %r9, (%r11)           FDDXMW
  addq $8, %r10               FFDXMW
  addq $8, %r11                 FDXMW
  subq $1, %r12                  FDXMW
  jmp .Loop                       FDXMW
.Done:
```

The figure also shows the execution of the code fragment on the standard 5-stage pipeline machine as presented on the lecture slides from 03/10-18 (pipelining) and used in assignments.

Recall; The letters above indicates the following stages:

- F: Fetch
- D: Decode
- X: eXecute
- M: Memory
- W: Writeback

All instructions pass through all 5 stages. Unconditional jumps are made in the D-stage, i.e. the instruction to which is jumped can be fetched in the following cycle. A conditional branch will, however, not be executed before the X-stage (which means that the F-stage of the target instruction will occur one cycle later then the X-stage of the branch itself). The architecture has full forwarding of operands from an instruction to following depending instructions. If instructions have to wait for values (e.g. waiting for a prior memory read) they wait in the D-stage until operands are available.

Pipeline, 1.4.1: How many clock cycles does it take to copy an array with n-elements with the above inner loop? Give your answer as a function of n. How much is the contribution from each loop iteration, and how much from the final exit from the loop.

(Maximum 5 lines.)

Some computer scientists found the simple pipeline too slow and have therefore developed a new architecture, called BeerBust. BeerBust has been optimised such that it has a *30% shorter clock period* than the simple pipeline. However, as BeerBust builds on the same memory design *access to the instruction and data caches now costs 2 clock cycles*. Thus, BeerBust has added two extra stages giving the following 7 stages:

- F: Fetch, first part of instruction fetch
- H: Fetch-2, second part of instruction fetch
- D: Decode
- X: eXecute
- M: Memory, first part of access to data cache
- Q: Memory-2, second part of access to data cache
- W: Writeback

As with the simple pipeline, all instructions pass through all 7 stages. Unconditional jumps are made in the D-stage. Conditional branches will, still not be executed before the X-stage. The architecture has full forwarding of operands from an instruction to following depending instructions. If instructions have to wait for values, they also wait in the D-stage until operands are available.

Pipeline, 1.4.2: Redraw the pipeline diagram showing the execution of the inner loop on BeerBust. You can extend the diagram with instructions from the following iteration if you need it to answer.

	Code	Timing
	`.Loop:`	
1	`cble $0, %r12, .Done`	
2	`movq (%r10), %r9`	
3	`movq %r9, (%r11)`	
4	`addq $8, %r10`	
5	`addq $8, %r11`	
6	`subq $1, %r12`	
7	`jmp .Loop`	
	`.Done:` or `.Loop:`	
8	`cbl $0, %r12, .Done`	
9		
10		

Exam number:

Pipeline, 1.4.3: To their despair, the computer scientists experience that their faster BeerBust machine executes the copy program slower than expected.

How can the program be updated to make it run faster on BeerBust? Remember to argue for you suggested changes. How does this change the answers to the previous question?

(Maximum 10 lines.)

1.5 Data Cache (about 5 %)

Given a byte-addressed machine with 8-bit addresses. The machine is equipped with a single L1-cache that is direct mapped and write-allocate, with a block size of 8 bytes. Total size of the data cache is 16 bytes

Data Cache, 1.5.1: For each bit in the table below, indicate which bits of the address would be used for

- block offset (denote it with `O`),
- set index (denote it with `S`), and
- cache tag (denote it with `T`).

7	6	5	4	3	2	1	0

Data Cache, 1.5.2: Consider we are running the following matrix transpose function, transposing a 2×2 array, on the machine above:

```c
void transpose(int dst[2][2], int src[2][2]) {
  for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
      dst[i][j] = src[j][i];
    }
  }
}
```

It is furthermore given that:

- The `src` array starts at address 0x40 and the `dst` array starts at address 0x50.
- Accesses to the `src` and `dst` arrays are the only sources of read and write misses, respectively. `i` and `j` are allocated in registers.
- The cache is initially cold and uses LRU-replacement.

For each element `row` and `col`, indicate whether each access to `src[row][col]` and `dst[row][col]` is a hit (`h`) or a miss (`m`). For example, reading `src[0][0]` is a miss as the cache is cold. Possibly explain your answer.

`dst` array		
	col 0	col 1
row 0		
row 1		

`src` array		
	col 0	col 1
row 0	`m`	
row 1		

(Maximum 5 lines.)

Exam number:

2 Operating Systems (about 80 minutes %)

2.1 True/False Questions (about 8 minutes %)

For each statement, answer True or False. (Put one "X" in each.)	True	False
a) If a process exits without calling `free()` on its allocated memory, that memory is lost until the machine is rebooted.		
b) Memory allocated with `mmap()` should be passed to `free()` when we are done using it.		
c) If a call to `read()` returns fewer bytes than we asked for, then the file is empty and the next call to `read()` will read zero bytes.		
d) If two threads are simultaneously calling `fread()` on the same `FILE*` object, then it is guaranteed that the same bytes are not read twice from the file.		
e) Starvation can happen in a single-processor system.		
f) In C, an `int` is *always* 32 bit.		

2.2 Multiple Choice Questions (about 12 minutes %)

In each of the following questions, you may put one or more answers.

Multiple Choice Questions, 2.2.1: Which of the following can potentially be shared between different processes, such that changes in one process are immediately reflected in the other?

- **a)** A page of memory.
- **b)** A single word in memory.
- **c)** A file descriptor.
- **d)** A single register.
- **e)** All registers.
- **f)** The signal mask.

Multiple Choice Questions, 2.2.2: Consider a demand-paged system with the following time-measured utilisations:

CPU utilisation	97.7%
Paging disk	0.7%
Other I/O devices	85%

Which of the following would likely improve system performance?

- **a)** Install a faster CPU.
- **b)** Install a bigger paging disk.
- **c)** Install a faster paging disk.
- **d)** Install more main memory.
- **e)** Increase the degree of multiprogramming.

Exam number:

2.3 Short Questions (about 24 minutes %)

Short Questions, 2.3.1: Consider the following program. Assuming that `printf()` itself executes atomically, how many characters of output does it produce? Is there more than one answer (i.e. does it contain nondeterministic behaviour or race conditions)? If yes, in what way? If no, why not?

```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>

void* thread(void* arg) {
  printf("a");
}

int main() {
  printf("a");
  pthread_t tid;
  pthread_create(&tid, NULL, thread, NULL);
  fork();
  printf("a");
  pthread_exit(0);
}
```

(Maximum 10 lines.)

Short Questions, 2.3.2: Consider a system with the following properties:

- Memory is byte-addressed.
- Virtual addresses are 15 bits wide.
- Physical addresses are 13 bits wide.
- The page size is 128 bytes.
- The TLB is 3-way set associative with four sets and 12 total entries. Its initial contents are:

Set	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	13	33	1	00	00	0	00	00	1
1	0A	11	0	11	15	1	1F	2E	1
2	0F	10	1	11	15	0	07	12	1
3	14	21	1	00	12	0	10	0A	1

- The page table contains 8 PTEs:

Exam number:

VPN	PPN	Valid	VPN	PPN	Valid	VPN	PPN	Valid	VPN	PPN	Valid
00	00	1	21	10	0	3F	23	0	12	34	1
11	11	1	01	02	1	02	01	1	13	33	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: `0x0712`

1. Bits of virtual address

14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

2. Address translation

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

3. Bits of physical address (if any)

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

Exam number:

Virtual address: `0x0001`

1. Bits of virtual address

14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

2. Address translation

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

3. Bits of physical address (if any)

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

Virtual address: `0x0891`

1. Bits of virtual address

14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

2. Address translation

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

3. Bits of physical address (if any)

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

2.4 Long Questions (about 36 minutes %)

Long Questions, 2.4.1: The following problem concerns dynamic storage allocation.

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

	31	2 1 0
Header	Block size (bytes)	
	⋮	
Footer	Block size (bytes)	

Each memory block, either allocated or free, has a size that is a multiple of eight bytes. 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.

Given the contents of the heap shown on the left, show the new contents of the heap (in the right table) after a call to `free(0x100f010)` is executed. 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	Value
0x100f028	0x00000013
0x100f024	0x100f611c
0x100f020	0x100f512c
0x100f01c	0x00000013
0x100f018	0x00000013
0x100f014	0x100f511c
0x100f010	0x100f601c
0x100f00c	0x00000013
0x100f008	0x00000018
0x100f004	0x100f601c
0x100f000	0x100f511c
0x100affc	0x00000018
0x100afe8	0x00000018

Address	Value
0x100f028	
0x100f024	0x100f611c
0x100f020	0x100f512c
0x100f01c	
0x100f018	
0x100f014	0x100f511c
0x100f010	0x100f601c
0x100f00c	
0x100f008	
0x100f004	0x100f601c
0x100f000	0x100f511c
0x100affc	
0x100afe8	

Exam number:

Long Questions, 2.4.2: In C, we say that a *double free error* occurs when `free()` is called twice on the same pointer, without that pointer having been returned by a new `malloc()` in between, which is not allowed. First, explain what happens if the `free()` call from the previous question is repeated. Second, describe how to modify the allocator such that double freeing can be detected and presumably cause an error message. Feel free to introduce any changes or auxiliary data structures you need. Characterise the properties of your solution: what is its overhead in time or space? Is it able to detect *all* instances of double freeing? Will it ever report a double free despite the true error being something else?

(Maximum 22 lines.)

Exam number:

Long Questions, 2.4.3: Suppose an implementation of `malloc()` is managing a fixed-size heap of 4096 bytes (1024 32-bit words), that each block comes with eight bytes of overhead, and that each block size must be a multiple of eight bytes. Describe the difference between *internal* and *external* fragmentation.

Draw/describe a heap layout that has enough *external* fragmentation to prevent an allocation of 256 bytes from being possible, yet still has as much total free space as possible.

(Maximum 23 lines.)

Exam number:

3 Computer Networks (about 80 minutes %)

3.1 True/False Questions (about 8 minutes %)

For each statement, answer True or False. (Put one "X" in each.)	True	False
a) It is impossible to implement HTTP using UDP as the transport layer protocol.		
b) TCP contains features of both Go-Back-N and Selective Repeat family of protocols.		
c) The broadcast address of the network 10.61.32.77/23 is 10.61.32.255		
d) Ethernet switches learn addresses by looking at the destination addresses of frames passing through it.		
e) traceroute uses the ICMP protocol for its functioning.		

3.2 Error Detection and Network Security (about 18 minutes %)

Error Detection and Network Security, 3.2.1: Suppose the information portion of a packet contains four bytes of the 8-bit binary representation of numbers 3 to 6. (Note: ASCII codes of 3-6 lie contiguously between 0x33-0x36)

a. Compute the 8-bit Internet checksum of this data.

(Maximum 1 lines.)

b. Will the receiver be able to detect an error if the positions of the bytes in the information portion of the packet are swapped (e.g., byte 1 contains number 4 and byte 2 contains number 3) but the checksum remains intact? Justify your answer.

(Maximum 6 lines.)

Exam number:

Error Detection and Network Security, 3.2.2: Within network security, what is a *nonce* and which security problem is its usage mainly trying to mitigate? Argue for your answer.

(Maximum 8 lines.)

3.3 DNS and Reliable Data Transfer (about 18 minutes %)

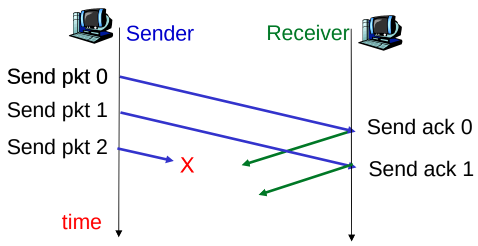
DNS and Reliable Data Transfer, 3.3.1: Does DNS caching benefit recursive DNS queries more than iterative DNS queries? Justify your answer.

(Maximum 14 lines.)

Exam number:

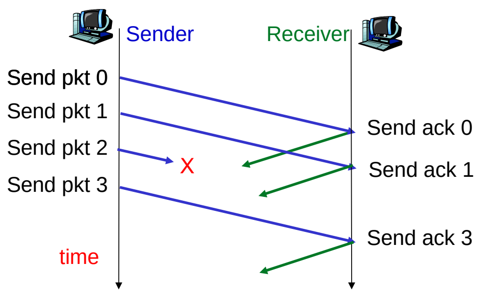
DNS and Reliable Data Transfer, 3.3.2: Consider the sliding window protocols shown in the figures below and identify whether Go-Back-N or Selective Repeat is being used or if there not enough information to tell. Justify your answer.

a. Protocol 1



(Maximum 7 lines.)

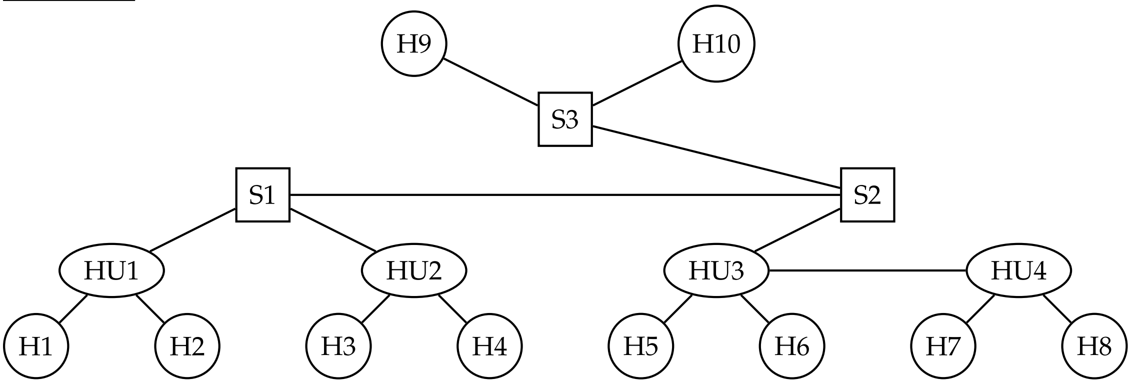
b. Protocol 2



(Maximum 7 lines.)

3.4 LAN (about 12 minutes %)

LAN, 3.4.1: Consider the LAN shown below



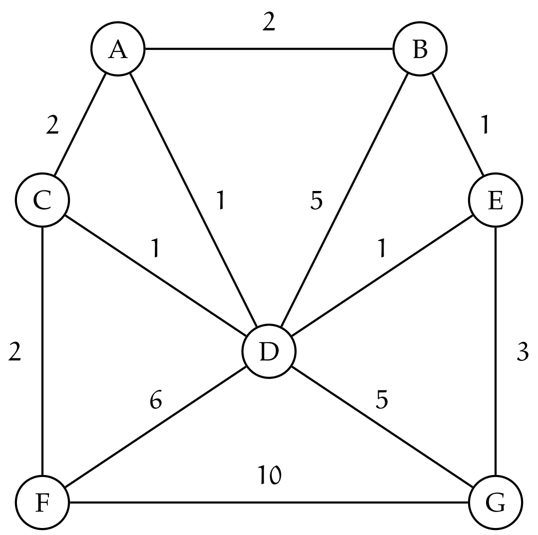
In this network S1-S3 are switches, HU1-HU4 are hubs and H1-H10 are end hosts. Suppose the following frames are sent in the order as indicated. For each frame, identify the end hosts that will receive it. Assume that initially all the switch tables are empty.

Frame	Recipients
H1 sends frame to H2	
H2 sends frame to H1	
H9 sends frame to H1	
H6 sends frame to H9	
H3 sends frame to H8	
H8 sends frame to H6	

Exam number:

3.5 Network Routing (about 24 minutes %)

Consider the network topology outlined in the graph below



Network Routing, 3.5.1: Apply the link state routing algorithm and compute the forwarding table on node A by filling out the following tables

Steps of the algorithm:

Step	N′	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)	D(G),p(G)
0							
1							
2							
3							
4							
5							
6							

(continues on next page.)

Forwarding table on node A:

Destination node	Edge
B	
C	
D	
E	
F	
G	

Network Routing, 3.5.2: Can poisoned reverse solve the general count to infinity problem ? Justify your answer.

(Maximum 12 lines.)

Exam number:

Network Routing, 3.5.3: Does distance vector routing algorithm exhibit better scalability than link state routing algorithm? Justify your answer.

(Maximum 12 lines.)

[]