4-hour Written Re-Exam in Computer Systems

Department of Computer Science, University of Copenhagen (DIKU)
Date: April 8, 2019

Preamble

Solution

Disclaimer: The reference solutions in the following range from exact results to sketch solutions; this is entirely on purpose. Some of the assignments are very open, which reflects to our solutions being just one of many. Of course we try to give a good solution and even solutions that are much more detailed than what we expect you to give. On the other hand, you would expect to give more detailed solutions that our sketches. Given the broad spectrum of solutions, it is not possible to directly infer any grade level from this document. All solutions have been written in the in-lined space with this colour.

This is the exam set for the 4 hour written exam in Computer Systems (CompSys), B1+2-2018/19. This document consists of 24 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 time needed. 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.

Exam number: ____________

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.

4-hour Written Re-Exam in Computer Systems
Department of Computer Science, University of Copenhagen
Date: April 8, 2019

Exam number: ____________

1 Machine architecture (about 80 minutes %)

1.1 True/False Questions (about 8 minutes %)

For each statement, answer True or False. (Put one "X" in each.)	True	False
a) Within Boolean arithmetic then A ^ B ^ C = C ^ B ^ A.	X	
b) In the 16-bit two's-complement integer representation the value -1032 is represented with the bit string `1111 1101 1111 1011`.		X
c) On most standard x86_64 machines today (e.g. Intel's Core machines), adding two `single` precision values takes longer time than adding two `long` values.	X	
d) In X86_64 the `set`-instructions *sets* the value of a given destination register depending on the values of the conditional registers.	X	
e) In X86_64 the `%r10` register is a caller saves register.	X	
f) In machines with separated L1-instruction and L1-data caches (e.g. Intel's Core machines), a cache miss in the L1-instruction cache will also result in a miss in the L1-data cache.		X

Exam number: ____________

1.2 Short Answer Questions (about 12 minutes %)

Short Answer Questions, 1.2.1: Explain the advantages of using two's complement number representation over binary numbers with a sign-bit. How does it affect value negation?

There is only one representation of zero.

Addition functions like binary (positive) numbers.

Value negation now have to be done by bit-wise negation plus 1 (using addition). Not a problem as addition is fast.

Short Answer Questions, 1.2.2: Explain the functionality of the **stack frame** of the x86_64 architecture. Explain the most important part of the frame structure.

Important for function calls. Keeping order on the stack.

Could be previous frame pointer, return address, extra arguments.

Exam number: ___________

1.3 Assembler programming (about 24 minutes %)

Consider the following program written in X86-assembler.

```
.prog:
  movq  $0, (%rsi)
  leaq  8(%rsi), %rax
  movq  $1, 8(%rsi)
.L1:
  cmpl  $2, %edx
  jle .L2
  movq  (%rax), %rcx
  addq  -8(%rax), %rcx
  cmpq  %rdi, %rcx
  jg   .L3
  addq  $8, %rax
  movq  %rcx, (%rax)
  subl  $1, %edx
  jmp .L1
.L2:
  movl  $0, %eax
  ret
.L3:
  movl  $1, %eax
  ret
```

Assembler programming, 1.3.1: Rewrite the above X86-assembler program to a C program. The resulting program should not have a goto-style and minor syntactical mistakes are acceptable. Describe also the functionality of the program and argue for you choices in the decompilation.

The program calculates the Fibonacci sequence and places from address `res` and forward. The program terminates either when the Fibonacci number exceeds `max` with return value `1` or when the array is full (number of elements given by `size`) with return value `0`.

(continues on next page.)

```
int fibseqmax(long max, long* res, int size) {
  *res = 0;
  res++;
  *res = 1;
  while (size > 2) {
    long c = *(res - 1) + *res;
    if (c > max) {
      return 1;
    }
    res++;
    *res = c;
    size--;
  }
  return 0;
}
```

Exam number: ____________

1.4 Pipeline (about 16 minutes %)

Below you find a code fragment written in x86prime; it is equivalent to x86 except for the inclusion of the branch-if-equal instruction, `cbe`. The code fragment shows the inner loop of a function that adds 1 to all elements of an array. Register `%r10` contains the pointer to the source array, `%r11` the length of the array.

```
    Code                          Execution on 5-stage pipeline

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

3 cycles for first instruction of last iteration (`cbl`) as the branch is taken in the X stage, 9 cycles for each iteration (note the `jmp` is taken in the D stage). Thus: $9n + 3$.

Some engineers wants to make the pipeline faster by simplifying it. They find that the memory (in M-stage) is the slowest part of the pipeline, so to make it faster they remove as much from this stage as possible. Specifically, they

- remove the forwarding from M-stage to all other stages.

All other parts of the pipeline are unchanged, including forwarding from other stages.

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

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

First note that it is only forwarding from M-stage to all other stages that are removed. Specially, forwarding from W-stage and X-stage is still in place.

In line 2, read values from `movq` is forwarded to `addq` in line 3. This is done in M-stage due to a memory-stall. This there are two stalls now.

In line 5 we now have a new stall as the value in `%r9` from 3 needs to be forwarded. It is to be used by the X-stage, but given that there is an instruction in-between the value will recite in the M-stage where we removed forwarding. Thus, we have to stall and wait until the W-stage to forward the value to the X-stage.

Line 5 only reads `%r10` and line 6 is, thus, not affected.

Exam number: ___________

1.5 Data Cache and Program Locality (about 20 minutes %)

Consider we are running the following procedure that iterates element-wise through an array and updates all following elements of the array. Do not focus on the actual update in line 4, other than it augments `arr[j]` with a value depending on `arr[i]`.

```
1 void arr_update(long arr[], int n) {
2   for (int i = 0; i < n; i++) {
3     for (int j = i+1; j < n; j++) {
4       arr[j] += 3 * arr[i];
5     }
6   }
7 }
```

In the execution of the program the local variables (`i` and `j`) and the integral input parameter (`n`) are allocated in registers. The array (`arr`) is located in memory.

Note: Following Questions 1.5.1 and 1.5.2 are completely independent.

Data Cache and Program Locality, 1.5.1: Given a byte-addressed machine with 32-bit addresses that is equipped with a fully associative L1 data-cache. It has **4 cache lines** each with a **block size of 16 bytes**. It functions as write-allocate and updates with LRU-replacement. We also know that `sizeof(int) == 4` and `sizeof(long) == 8`.

Assume that we are executing `arr_update` under the following conditions:

- the `arr` array starts at address `0x00008000`,
- the size of array is 12 (`n == 12`), and
- the cache is assumed to be cold on entry.

For each of the first 8 iterations (`i == 0` to `i == 7`) indicate whether each access results in a cache hit (`h`), a miss (`m`), or if it is unused (keep it empty). For example, in first iteration (`i == 0`) reading `arr[0]` is a miss as the cache is cold. Explain the considerations behind your answer below the table.

	arr indices											
	0	1	2	3	4	5	6	7	8	9	10	11
Iteration 1 (`i = 0`)	m	h	m	h	m	h	m	h	m	h	m	h
Iteration 2 (`i = 1`)		h	m	h	m	h	m	h	m	h	m	h
Iteration 3 (`i = 2`)			m	h	m	h	m	h	m	h	m	h
Iteration 4 (`i = 3`)				h	m	h	m	h	m	h	m	h
Iteration 5 (`i = 4`)					m	h	m	h	m	h	m	h
Iteration 6 (`i = 5`)						h	h	h	h	h	h	h
Iteration 7 (`i = 6`)							h	h	h	h	h	h
Iteration 8 (`i = 7`)								h	h	h	h	h

Exam number: ___________

The block size gives that two elements of the array is held in each cache line. The array is aligned such that each block starts on even array indexes.

References to `arr[i]` are misses only for even `i` as the block is kept in memory. Consecutive updates of the array (`arr[j]` for `j > i`) will create the miss-hit oscillation.

This will continue until the 5th iteration, where the rest of the array can now be completely contained in the 4-way cache.

Data Cache and Program Locality, 1.5.2: The given program exhibits spatial locality but *not* temporal locality. Argue why this is the case (consider both temporal and spatial locality) and describe how the program can be updated to also have temporal locality. For the last part you are welcome to exemplify with an updated program, but giving it as the only solution will not give credits.

The indexes of both the inner and outer loops are forth-running from 0 to the size. Given that the input data is a (one-dimensional) array this gives spatial locality.

However, in the inner loop we have a continues reference to `arr[i]`. This is an address that is directly indexed by the outer loop variable and is assured not to be updated by the inner loop. Thus we do not have temporal locality. To exhibit temporal locality the reference to in the inner loop must be stored in a local variable in the outer loop:

```
void arr_update(long arr[], int n) {
  for (int i = 0; i < n; i++) {
    long tmp = 3 * arr[i];
    for (int j = i+1; j < n; j++) {
      arr[j] += tmp;
    }
  }
}
```

Exam number:

Data Cache and Program Locality, 1.5.3: Will the cache behaviour in the execution on the machine of Question 1.5.1 be affected by updating the program to exhibit temporal locality? How will it affect execution speed? Argue for your answer.

Yes. Every second iteration (even iterations) have a reference to `arr[i]` (first reading updating value) will now be a miss instead of a hit (block is pushed out after writing to local variable). However the continues hits will start an iteration earlier as we no can use all 4 sets for from `arr[j]`.

Note; for a longer executions this can actually result in a higher miss-rate, but given that all reference (except to the first) to `arr[i]` now will be in a register this give a lower execution speed.

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) Before calling `thread_cond_wait(cond, mutex)`, it is important to unlock `mutex` first.		X
b) Two processes can share read/write memory via `mmap()`.	X	
c) To free an array allocated with `calloc()`, we can pass the address of any element of the array to `free()`.		X
d) `EOF` (as returned by e.g. `getchar()`) is of type `unsigned char`.		X
e) A semaphore can be implemented with mutexes and condition variables.	X	

2.2 Multiple Choice Questions (about 16 minutes %)

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

Multiple Choice Questions, 2.2.1: Under normal circumstances `stdout` will be buffered. Under which of the following cases of process termination can we depend on the `stdout` buffer being flushed for us?

- [✓] **a)** `return` from `main()`.
- [✓] **b)** A call to `exit()`.
- [ ] **c)** The process being terminated due to `SIGKILL`.
- [ ] **d)** The process being terminated due to a segmentation fault.

Multiple Choice Questions, 2.2.2: A *page table entry* (PTE) contains

- [ ] **a)** Virtual address of the page.
- [✓] **b)** Physical address of the page.
- [ ] **c)** Page size.
- [✓] **d)** Read/write permission bits.
- [ ] **e)** The contents of the page.

Multiple Choice Questions, 2.2.3: 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) {
        if (fork() == 0) {
            printf("3");
        }
        else {
            pid_t pid; int status;
            if ((pid = wait(&status)) > 0) {
                printf("4");
            }
        }
    }
    else {
        printf("2");
        exit(0);
    }
    printf("0");
    return 0;
}
```

Which of the following strings is a possible output of the program?

[✓] **a)** 32040

☐ **b)** 32002

[✓] **c)** 30402

[✓] **d)** 23040

☐ **e)** 40302

2.3 Short Questions (about 20 minutes %)

Short Questions, 2.3.1: Consider a system with the following properties:

- Memory is byte-addressed.
- Virtual addresses are 15 bits wide.
- Physical addresses are 16 bits wide.
- The page size is 256 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	00	00	0	11	33	0	00	00	1
1	07	21	1	09	01	1	11	51	0
2	10	11	0	12	51	1	1F	A2	1
3	11	12	1	00	21	0	10	A0	1

- The page table contains 8 PTEs:

Exam number: ____________

VPN	PPN	Valid	VPN	PPN	Valid	VPN	PPN	Valid	VPN	PPN	Valid
12	12	1	01	02	1	02	01	1	31	33	1
00	00	1	10	21	0	13	32	0	21	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: `0x0855`

1. Bits of virtual address

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

2. Address translation

Parameter	Value
VPN	8
TLB index	0
TLB tag	2
TLB hit? (Y/N)	N
Page fault? (Y/N)	Y
PPN	

3. Bits of physical address (if any)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Exam number:

Virtual address: `0x0042`

1. Bits of virtual address

14	13	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	0	1	0

2. Address translation

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

3. Bits of physical address (if any)

15	14	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	0	1	0

Virtual address: `0x0123`

1. Bits of virtual address

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

2. Address translation

Parameter	Value
VPN	1
TLB index	1
TLB tag	0
TLB hit? (Y/N)	N
Page fault? (Y/N)	N
PPN	2

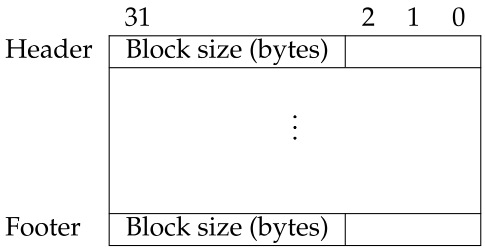
3. Bits of physical address (if any)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	1

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:



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(0x010f010)` 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
0x010f028	0x00000012
0x010f024	0x010f611c
0x010f020	0x010f512c
0x010f01c	0x00000012
0x010f018	0x00000011
0x010f014	0x010f511c
0x010f010	0x010f601c
0x010f00c	0x00000011
0x010f008	0x0000001a
0x010f004	0x010f601c
0x010f000	0x010f511c
0x010affc	0x010f510c
0x010afe8	0x0000001a
0x010afe4	0x0000001a

Address	Value
0x010f028	0x0000003a
0x010f024	0x010f611c
0x010f020	0x010f512c
0x010f01c	0x00000012
0x010f018	0x00000011
0x010f014	0x010f511c
0x010f010	0x010f601c
0x010f00c	0x00000011
0x010f008	0x0000001a
0x010f004	0x010f601c
0x010f000	0x010f511c
0x010affc	0x010f510c
0x010afe8	0x0000001a
0x010afe4	0x0000003a

Long Questions, 2.4.2: Consider the following function, which copies from one file to another (via file descriptors):

```c
void copy(int from, int to) {
  unsigned char c;
  while(read(from, &c, 1) != 0) {
    write(to, &c, 1);
  }
}
```

Is this program efficient? Why or why not? Explain a better way of copying, and why your solution is faster. Is your improved solution an example of a time/space-tradeoff?

This program is inefficient, because it performs two system calls per byte being copied. A better way of copying is to use buffering, such that we read a large block of bytes per system call (say, 4KiB, but the optimal amount is situation-dependent), and also write that entire block at once. This is most easily done by using the standard IO facilities and `fwrite()`/`fread()`. This is an example of a time/space-tradeoff, because the buffer will take up memory—hence, it should be large enough to amortise the system calls, but no larger.

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) HTTP response messages can never have an empty message body.		X
b) The broadcast address of the network 7.8.9.12/22 is 7.8.11.255.	X	
c) IP provides a best-effort unreliable message delivery guarantee.	X	
d) ARP is used to resolve IP addresses to MAC addresses.	X	
e) TCP provides a connection-oriented but not ordered message delivery service.		X

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

Error Detection and Network Security, 3.2.1: Consider a two dimensional even parity scheme for error detection where each byte of the word forms a row for the two dimensional parity scheme. Suppose the information portion of a packet contains four bytes of the 8-bit ASCII representation of the word "EVEN". [1]

a. Compute the parity bits (row bits followed by column bits) for the packet data.

0xA 0x18

b. Will the receiver be able to detect an error if the positions of byte 1 and byte 2 in the information portion of the packet are swapped but the parity bits remain intact? Justify your answer.

Yes, because the first two column parity bits are not the same so the error would be detected when the column parity bits are recomputed and matched with the parity bits in the packet data.

[1] Hex(E)=0x45, Hex(V)=0x56, Hex(N)=0x4E

Error Detection and Network Security, 3.2.2: Within network security, name the main advantage and disadvantage of asymmetric encryption over symmetric encryption. Name two significant applications of asymmetric encryption and argue for the usage of asymmetric encryption to these.

Main advantage: no shared secret. Also does not require a stream of data.

Main disadvantage: slow in comparison to symmetric encryption.

Applications:

* Key exchange: Small amount of data. Have not shared secret as this is what want to establish.

* Document signing: Based on verifiable public knowledge, no shared secret. Signing of hash-value makes is applicable to small amount of data.

* PGP (single message privacy): No stream of data (communication is one-directional) makes symmetric encryption unusable, messages tend to be smaller so we can accept the computational overhead.

3.3 HTTP and TCP (about 16 minutes %)

HTTP and TCP, 3.3.1: Consider the following HTTP request and response message between the DIKU web-server (hjemmesider.diku.dk) and a telnet client and answer the following questions.

```
HEAD /~bonii/ HTTP/2.0
Host: hjemmesider.diku.dk
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Mon, 18 Mar 2019 10:14:34 GMT
Server: Apache/2.4.6 (Red Hat Enterprise Linux)
OpenSSL/1.0.2k-fips mod_fcgid/2.3.9
Last-Modified: Mon, 17 Sep 2018 11:23:39 GMT
ETag: "4104-5760f67a95aa4"
Accept-Ranges: bytes
Content-Length: 16644
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html
```

a. What version of HTTP is being used?

The client requests version 2.0 but the server responds in version 1.1

b. Was the TCP connection closed by the web server after sending the response message? Why?

No, because the client requested a persistent connection that was acknowledged by the server in the response message using the Connection header.

c. Why would a conditional GET request benefit from using both Last-Modified and ETag header fields instead of using only one of them?

The ETag header uses hashing to determine the version of the document. Since hashes are prone to collisions two different document versions could end up having the same ETag value. While the Last-Modified header uses the file modification time of the document to check for newer versions, file meta-data modification leads to changes in this value. Using a combination of both these headers provides a higher degree of resilience and faster performance (check last-modified time before hashing) than using one of them.

HTTP and TCP, 3.3.2: Consider five Internet hosts, each with an ongoing TCP session. All these TCP sessions share a common bottleneck link and any packet loss on the end-to-end paths for these sessions occurs at just this one link. The bottleneck link has a transmission rate of R. The round trip times (RTT) for all the hosts to their destinations are approximately the same. Assume that there are no other sessions using this link and the five sessions have been running for a long time.

a. What is the approximate throughput of each of these TCP sessions? Justify your answer.

R/5 since flow and congestion control algorithms ensure TCP shares bandwidth fairly for connections with same RTT.

b. If one of the sessions terminates what is the new throughput achieved by each of the remaining sessions? Justify your answer.

R/4 since TCP shares bandwidth fairly.

c. Now if one of the remaining four hosts starts a second session that also crosses this bottleneck link and has similar RTT. What is the throughput achieved (in aggregate) by this host with two sessions, and by each of the two hosts with one session each?

Each session will again get R/5, so the one host with two sessions will get 2R/5 in aggregate and the other three hosts will each get R/5.

d. Now if one of the hosts starts using a UDP session across the bottleneck link. Is it possible for this host to monopolize the entire bandwidth R of the bottleneck link?

Yes, because UDP has no congestion or flow control so it will blast off data without backing off.

3.4 IP Forwarding (about 14 minutes %)

IP Forwarding, 3.4.1: Consider the following forwarding table on a router that utilizes longest prefix match to route network datagrams and answer the following questions

Prefix	Outgoing Link
10.201.129.0/18	2
10.201.2.0/19	1
10.201.39.0/19	1
10.201.135.0/17	1
10.201.128.0/18	2
default	0

a. Which outgoing link would a network datagram with destination address 10.201.7.202 be sent on?

Port 1

b. Which outgoing link would a network datagram with destination address 10.202.2.16 be sent on?

Port 0

c. Compact the forwarding table further to reduce the number of entries in it without affecting its forwarding rules.

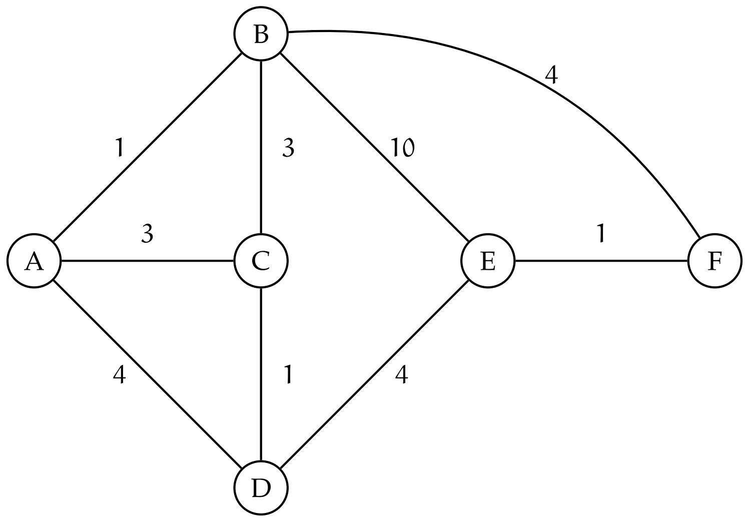
Compacted forwarding table:

Prefix	Outgoing Link
10.201.0.0/16	1
10.201.64.0/17	2
10.201.128.0/17	2
default	0

Exam number: ____________

3.5 Network Routing and Ethernet (about 24 minutes %)

Consider the network topology outlined in the graph below



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

Steps of the algorithm:

Step	N′	D(A),p(A)	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(F),p(F)
0	E	∞	10,E	∞	4,E	1,E
1	EF	∞	5,F	∞	4,E	1,E
2	EFD	8,D	5,F	5,D	4,E	1,E
3	EFDB	6,B	5,F	5,D	4,E	1,E
4	EFDBC	6,B	5,F	5,D	4,E	1,E
5	EFDBCA	6,B	5,F	5,D	4,E	1,E

(continues on next page.)

Exam number:

Forwarding table on node E:

Destination node	Edge
A	(E,F)
B	(E,F)
C	(E,D)
D	(E,D)
F	(E,F)

For same costs, link state computation can pick nodes (B,C) in different order.

Network Routing and Ethernet, 3.5.2: What is the reason behind Ethernet's exponential backoff mechanism being better than randomizing retransmission attempts over a fixed-length time interval?

The exponential backoff mechanism of Ethernet maintains an interval of time T over which it will randomize when it attempts a frame re-transmission post a collision. After every collision of the same frame, the mechanism doubles the length of T up to some fixed maximum. This mechanism is better than using just a single, fixed value of T because it adapts dynamically to the number of collisions. When there are only a few colliding nodes, the re-transmission will be randomized quickly over a small T, allowing the nodes to transmit quicker. Conversely, when there are a lot of colliding nodes, the randomization interval will be large, allowing the nodes to make a successful re-transmission.