Coursework commentary 2017–2018

CO1110 Introduction to computing and the internet

Coursework assignment 1

General remarks

On the whole both coursework assignments were attempted well. Often students lost credit, not because their answers were wrong, but because their answers lacked clarity or detail.

Question 1

a. Students were asked to convert the positive decimal number 17.45 to IEEE 754 single precision representation. Students were asked to show all of their working, and could not receive full credit for this question without demonstrating how they arrived at the answer. The model answer below identifies each part of the IEE 754 number (sign bit, exponent, biased exponent, mantissa and normalised mantissa), and would have received full credit.

Solution

Sign bit is 0 because the number is positive

Converting 17 (decimal) to binary gives 10001

Converting 0.45 (decimal) to binary gives 0.011100

The underlined numbers above will repeat indefinitely.

This gives the 17.45_{10} in binary as: 10001.011100 (underlined part repeats)

Move the binary point to after the first 1: 1.0001011100 x 2^4

Hence the **mantissa** is 1.0001011100

Giving the **normalised mantissa** as: (unrounded)	0001 0111 0011 0011 0011 001
or (rounded)	0001 0111 0011 0011 0011 010

Exponent of the normalised mantissa is 4

The **biased exponent** is 127 + 4 = 131 (decimal) = 1000 0011 (binary)

IEEE 754 32-bit expression	0 1000 0011 0001 0111 0011 0011 0011 010
	or
	0 1000 0011 0001 0111 0011 0011 0011 001

b. Students were asked to explain why there are many more 1's in the mantissa of the IEEE 754 representation of 1.26 than 1.25. A good answer would have explained that 1.25 (decimal) can be represented exactly in binary as 1.01, hence the mantissa is 01, with the rest of the spaces of the normalised mantissa filled by zeros. The fractional part of 1.26 (decimal), 0.26, cannot be represented exactly in binary, even though it can be represented exactly in decimal. Each number base will have some numbers that can be represented exactly, and some that cannot, and these will vary between bases. In binary, 1.26 (decimal) is 1.010000101000111101011 – the 20 digits in the underlined part repeat infinitely.

c. Students were asked to explain why the exponent is biased in IEEE 754 representation. The simple answer is so that negative exponents, and hence very small numbers, can be represented. Without biasing the exponent, only positive exponents would be possible, unless a bit was sacrificed to indicate the sign, leading to a smaller range of numbers being represented.

Question 2

a. Students were asked to consider the problem of finding the total time to access and read data stored as tracks on a computer disk. Students could lose marks for this question by not showing all their working, allowing the examiners to give marks for answers that were partially correct.

Solution

The file occupies 6.5 tracks sequentially. This means that once the right place to start reading from has been found, the entire track can be read with no further seek time, or time allowed for rotational latency.

The time to read the first track is:

Average seek time 6ms

Plus rotational latency

Plus time to read one track

Rotational latency = half the time for one disk rotation

12,000/60 = 200, therefore there are 200 rotations each second

1/200 = 0.005 – there is one rotation every 0.005 seconds, i.e. one rotation per 5 ms

This gives a **rotational latency** of 5ms / 2 = 2.5ms

The time to read one track is 5ms, the time for one rotation.

Time to read first track is (6 + 2.5 + 5)ms = 13.5ms

The next 5.5 tracks can be read with no seek time, just rotational latency, followed by reading time for each track. For 5 tracks we have

5 x (2.5 + 5) = 5 x 7.5 = 37.5

For the final half a track we have a rotational latency of 2.5ms, and the time to read half a track is the time for half a rotation = 2.5ms, so the total time is **5ms**.

Total time to access and read = (13.5 + 37.5 + 5)ms = 56ms

Some students lost marks here for assuming that once the start of the file had been found, the entire file could be read with no further time allowed for rotational latency.

b. Students were asked to calculate the total time to access and read the same amount of data as in part (a), but with each sector of the file now distributed at random across the disk.

Solution

Since there are 500 sectors per track, and since there are 6.5 tracks when the file was stored sequentially, there are 6.5 x 500 sectors in the file, that is, the file comprises 3,250 sectors.

Since the sectors are distributed randomly, each sector must be accessed and read independently of the others. Therefore, the time to access and read the entire file is 3,250 x (time to access and read one sector).

The time to access and read one sector is:

Seek time = 6ms

+ Rotational latency = 2.5 ms

+ 1/500 x 5ms = 0.01 ms ([1 sector divided by 500 per track] times the time taken for one rotation)

Total time to read one sector = 8.51 ms

Total time to access and read the file

= 3,250 × 8.51 ms

= 27,657.5 ms (or 27.6575 seconds, or nearly half a minute).

c. Students were asked to express the time taken to read the file with the data stored sequentially, as a percentage of the time taken to read the file if the data is stored randomly. The answer should be given to one significant figure as follows:

56 / 27,657.5 = 0.0020247672421585 = 0.2% (1sf)

This demonstrated the large time differential in reading a file, depending on how it was stored.

Question 3

a. Students were asked to explain what is meant by *locality of reference* and how this is exploited in cache memory to improve performance. Many students lost marks for lack of clarity, or for not giving enough detail in their answers.

A good answer might have said that *locality of reference* is the principle that 'memory references by the processor, for both instructions and data, tend to cluster' (Stallings 2016, section 4.1, p149). Clustering is considered to be both temporal and spatial. Temporal clustering means that recently used memory locations are likely to be used again soon, for example when a program iterates through a loop. Spatial clustering means that data locations that are close together are likely to be accessed sequentially. Over time the clusters of memory references change, but over a short period of time the processor is likely to be working with a defined set of references.

Cache memory exploits locality of reference to reduce cache misses. Temporal locality is exploited by keeping recently used data and instructions in the cache, and spatial locality can be exploited by fetching not just the current needed data location, but those surrounding it too.

b. Students were given the following information about a processor:

- a direct mapped cache
- data words are 8 bits long
- data addresses are to the word
- a physical address is 33 bits long
- the tag is 11 bits
- each block holds 16 kB of data.

Students were then asked to calculate the number of lines (blocks) in this cache, and to show all working. Students could lose marks by not giving enough justification for their answers in two ways. Firstly, even a correct answer could lose credit for lack of detail, and secondly examiners could not give credit for partially correct answers without being able to see where the student's reasoning was correct. The following answer would have received full credit.

Solution

The physical address is 33 bits long and, in a direct-mapped cache, the cache views these bits as comprising:

- *s* bits to reference one of the 2^s blocks of main memory
- an offset *w* (referencing a unique word in a block).

The cache views the *s* bits referencing the blocks of main memory as comprising a tag (a unique identifier for a group of data in main memory) plus a field of *r* bits to identify one of the 2^r lines of the cache.

Since the tag is 11 bits, this means that the cache line index *r* and the offset *w*, together add up to the remaining 22 bits of the address.

A kilobyte is 2^{10} bytes, so each line (block) holds 16×1024 (2^{14}) bytes of data. Because the data is byte address;able, this means that 14 bits are required for the offset *w*.

This means that the cache line index, $r = 22 - 14$ bits, i.e. 8 bits.

Since 8 bits allow for 2^8 blocks to be addressed, there are 256 blocks of data in the cache.

Question 4

a. & b. Many good attempts were seen at this question, but also some mistakes were made. Common mistakes were not understanding that a stall delays all subsequent instructions, and not recognising the data hazard between instruction 4 and instruction 5.

Stalls and subsequent instructions

Assume that we have a series of instructions, and the second instruction must stall in order to receive the result of the first. The table below shows what would happen if the stall in instruction 2 did not delay subsequent instructions. In this scenario by instruction 4 the same stages are happening twice in the same cycle, which is not possible.

Table A: Subsequent cycles not delayed by earlier stalls

Clock cycle →	1	2	3	4	5	6	7	8
Ins ↓								
1	IF	ID	EX	MEM	WB			
2		IF	stall	stall	**ID**	**EX**	**MEM**	**WB**
3			IF	ID	EX	MEM	WB	
4				IF	**ID**	**EX**	**MEM**	**WB**

Hence a delay must delay all subsequent instructions, in order to maintain the integrity of the pipeline. The instructions after instruction 2 must be delayed, as shown in the following tables B and C:

Table B: Subsequent cycles delayed by previous stalls

Clock cycle →	1	2	3	4	5	6	7	8	9	10
Ins ↓										
1	IF	ID	EX	MEM	WB					
2		IF	**stall**	**stall**	ID	EX	MEM	WB		
3					IF	ID	EX	MEM	WB	
4						IF	ID	EX	MEM	WB

Another way to view this is below in table C:

Table C: Another view of how stalling delays subsequent cycles

Clock cycle →	1	2	3	4	5	6	7	8	9	10
Ins ↓										
1	IF	ID	EX	MEM	WB					
stall		○	○	○	○	○				
stall			○	○	○	○	○			
2				IF	ID	EX	MEM	WB		
3					IF	ID	EX	MEM	WB	
4						IF	ID	EX	MEM	WB

Note that a stall is also called a *bubble*, as it is considered to be like a bubble in a water pipe, in that it must move along to the end before it can escape.

Data hazards

1. ADD R1, R2, R1
2. INC R5, R5
3. ADD R2, R5, R5
4. SUB R1, R3, R3
5. ADD R3, R4, R4

Instruction sequence given to students. Note that the result is saved in the final register given by the instruction.

A data hazard is when a subsequent instruction needs the value calculated by a previous instruction, but that value has not yet been written into the register. In the instructions given to students (listed above), there are the following potential data hazards:

- Instruction 1 saves a new value in register R1; instruction 4 reads the value saved in R1.
- Instruction 2 saves a new value in register R5; instruction 3 reads the value saved in R5.
- Instruction 4 saves a new value in R3; instruction 5 reads R3.

In practice instructions 1 and 4 are far enough apart that there is no hazard, but since 2 and 3 are sequential, as are instructions 4 and 5; stalls must be introduced for both hazards. For some reason, many students added stalls to deal with the data hazard between instructions 2 and 3, but ignored, or failed to recognise, the hazard between 4 and 5.

Different answers are possible depending on the assumptions made

Students were expected to give an answer to part (a) similar to the table below, to explain that the instructions take 13 cycles to complete without operand forwarding. However, different answers are possible, depending on what assumptions are made. For example, it is possible to assume that stages are missed out completely when they are not needed. The classic RISC pipeline has five stages, which were given to students as part of the question: IF, ID, EX, MEM, WB). Patterson and Hennessy (2017, page 285), note that both the MEM stage (accessing main memory) and the WB stage (writing to the registers) may not be necessary for all instructions. The

MEM stage is not needed in any of the instructions above, hence students could assume in their answers that this stage would be omitted, but needed to write down that assumption for full credit.

The ID stage is where instructions are decoded and register values read, and the WB stage is where the result of calculations are written back to the registers. Most students made the assumption that when there is a data hazard, the WB stage of the first instruction, and the ID stage of the subsequent instruction needing the value written to the register by the first instruction, can take place in the same cycle. This is a valid assumption as Patterson and Hennessy (2017, page 316) point out when a register is both written to and read in the same clock cycle:

We assume that the write is in the first half of the clock cycle and the read is in the second half, so the read delivers what is written. As is the case for many implementations of register files, we have no data hazard in this case.

Other students made the assumption that where there was a data hazard between two instructions, the ID stage of the subsequent instruction would need to be delayed to take place in the clock cycle immediately after the cycle in which the WB stage of the first instruction happened.

In the model answer below, the assumptions are:

- All five stages of the pipeline are included in every instruction, even if not carried out.
- Where there is a data hazard between two instructions, the WB stage of the first instruction and the ID stage of the second can take place in the same clock cycle.

Table 1: The answer to (a) and (b)

Clock cycle →	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Ins ↓													
1	**IF**	**ID**	**EX**	**MEM**	**WB**								
2		**IF**	**ID**	**EX**	**MEM**	**WB**							
3			**IF**	**stall**	**stall**	**ID**	**EX**	**MEM**	**WB**				
4				**stall**	**stall**	**IF**	**ID**	**EX**	**MEM**	**WB**			
5							**IF**	**stall**	**stall**	**ID**	**EX**	**MEM**	**WB**

In the table 1 above, register values written by WB, and read by a subsequent instruction at the ID stage, have been colour coded to match each other.

Table 2: The answer to (a) and (b)

Clock cycle →	1	2	3	4	5	6	7	8	9	10	11	12	13
Ins ↓													
1	IF	ID	EX	MEM	WB								
2		IF	ID	EX	MEM	WB							
stall			○	○	○	○	○						
stall				○	○	○	○	○					
3					IF	ID	EX	MEM	WB				
4						IF	ID	EX	MEM	WB			
stall							○	○	○	○	○		
stall								○	○	○	○	○	
5									IF	ID	EX	MEM	WB

The answer is given again in table 2 above, but with a different view of the stalls introduced for the data hazard between instructions 2 and 3, and the data hazard between instructions 4 and 5.

c. To reorder the instructions to reduce (or eliminate) stalls, we first note instructions that must take place in a particular order, because the second instruction uses the data value calculated by the first:

- Instruction 1 must be followed by instruction 4
- Instruction 2 must be followed by instruction 3
- Instruction 4 must be followed by instruction 5

Combining the first and last bullet point, we have the order 1, 4, 5. Instructions 2 and 3 can be interleaved with 1, 4, and 5 in order to reduce stalls, giving: 1, 2, 4, 3, 5. Some students arrived at exactly this ordering, by swapping the order of instructions 3 and 4, in order to provide more separation between instructions 2 and 3. These students arrived at the correct answer, even though they were following through on their mistake in part (a), that of ignoring, or not recognising, the data hazard between instructions 4 and 5.

Other students reduced stalls by moving instruction 2 to the first place, again to deal with the data hazard between 2 and 3 only, and ignoring that between 4 and 5. This gives the following result:

Table 3: Executing instruction 2 first

Clock cycle →	1	2	3	4	5	6	7	8	9	10	11	12
Ins ↓												
2	IF	ID	EX	MEM	WB							
1		IF	ID	EX	MEM	**WB**						
3			IF	**STALL**	ID	EX	MEM	WB				
4				**STALL**	IF	ID	EX	MEM	WB			
5						IF	**STALL**	**STALL**	ID	EX	MEM	WB

Hence moving instruction 2 to the first place has reduced stalls by one.

Table 4: Reordering instructions as 1, 2, 4, 3, 5

Clock cycle →	1	2	3	4	5	6	7	8	9	10	11
Ins ↓											
1	**IF**	**ID**	**EX**	**MEM**	**WB**						
2		**IF**	**ID**	**EX**	**MEM**	**WB**					
4			**IF**	**stall**	**ID**	**EX**	**MEM**	**WB**			
3				**stall**	**IF**	**ID**	**EX**	**MEM**	**WB**		
5						**IF**	**stall**	**ID**	**EX**	**MEM**	**WB**

In table 4 above, stalls have been reduced by 2, and the instructions now take 11 clock cycles to complete. Hence it is not possible to eliminate stalls by reordering the instructions, but it is possible to reduce them.

Table 5: Another view of reordering instructions as 1, 2, 4, 3, 5

Clock cycle →	1	2	3	4	5	6	7	8	9	10	11
Ins ↓											
1	**IF**	**ID**	**EX**	**MEM**	**WB**						
2		**IF**	**ID**	**EX**	**MEM**	**WB**					
stall			○	○	○	○	○				
4				**IF**	**ID**	**EX**	**MEM**	**WB**			
3					**IF**	**ID**	**EX**	**MEM**	**WB**		
stall						○	○	○	○	○	
5							**IF**	**ID**	**EX**	**MEM**	**WB**

Coursework commentary 2017–2018

CO1110 Introduction to computing and the internet

Coursework assignment 2

General remarks

On the whole this coursework assignment was attempted well.

Comments of specific questions

Question 1

a. Students were asked to describe the cumulative ACK process used in the Transmission Control Protocol (TCP). A good answer would have included the following points:
   - Data packets are sequentially ordered for transmission.
   - Not all segments must be explicitly ACKed.
   - Only in-order segments will be ACKed, hence a data packet can only be acknowledged if all previous data packets have been received.
   - An ACK implicitly acknowledges all packets received up to that point with a smaller sequence number.

b. Students were told that when the TCP receiver gets a packet that is out of order, it acknowledges again the last correctly received packet, and asked how this helps to detect missing packets. A good answer would have noted that repeated acknowledgement of a data packet can be used as a marker of packet loss in the TCP protocol, if it implements the fast retransmission algorithm. The algorithm is called 'fast' because the idea is to take 3 duplicate acknowledgements as notification of packet loss, instead of waiting for the timer to run down.

   As an example, assume that the receiver gets a packet with the sequence number x, and sends the sequence number x+1 to the sender in acknowledgement. The receiver next receives packet x+2, and so acknowledges, for the second time, packet x. The receiver then receives packet x+3, and so for the third time acknowledges packet x (by sending x+1 as the sequence number of the next packet it is expecting). After 3 duplicate ACKs, under the fast retransmission algorithm, the sender will assume that packet x+1 was lost, and send it again.

c. Students were asked to describe the other method the TCP protocol uses to detect missing packets. A good answer would have noted the use of timeout based retransmission in the original TCP protocol, and would have explained that the sender sets a timer for every packet it sends, with an expected time for an ACK to be received. If no ACK is received in this time, the packet is retransmitted. Only these basic facts were needed for full credit.

d. Students were asked to give a reason why the cumulative ACK process from the original TCP protocol could be inefficient. In answer to this question many students wrote a great deal about SACK, the Selective Acknowledgement scheme, which allows a receiver to specify exactly which data has been received, and which is missing. Giving details about SACK was not necessary for full credit (note that the scheme is optional and does not replace cumulative acknowledgement), but explaining fully the reason why the original cumulative ACK process could be inefficient was.

A good answer would have noted that the receiving TCP engine saves out-of-order data (provided that it has the space to do so). Each time it receives a packet that is out of order it responds by ACKing again the last in-order data received. The receiver cannot tell the sender that it has successfully received and saved much out-of-order data.

Once the sender reaches the timeout for acknowledgement of a missing segment (or receives 3 duplicate ACKs if it has implemented the fast retransmission algorithm), it sends the data packet again. However, the sender does not know if all packets that it sent from that point are lost, or if only some, or just one. So the sender could resend the missing packet, and wait for the ACK, to see if it needs to resend all packets. Alternatively, the sender could immediately send again all packets starting with the missing one, which is inefficient. But as Comer (2014, page 216) makes clear, so is waiting:

> 'If the sender follows the accepted standard and retransmits only the first unacknowledged segment, it must wait for the acknowledgement before it can decide what and how much to send. Thus, retransmission reverts to a send-and-wait paradigm, which loses the advantages of having a large window.'

Question 2

Students were told that a network administrator with an IP address of 192.168.67.0 wishes to apply a subnet mask such that each subnet will have ***at least*** 25 hosts, and asked for the category of this network address, and how many bits of the address were reserved for the network, and how many for the host. The answer to this part of the question was that the address was category C, with 24 bits for the network, and 8 for the host.

Following this, students were asked what subnet mask the network administrator should use in order to have the most possible subnets. Students should have noted the restriction that each subnet should have at least 25 hosts, in which case the subnet mask 255.255.255.224 would give the most subnets (8) meeting the condition of at least 25 hosts (in fact there would be 30 hosts in each subnet). Other subnet masks would either give more hosts but fewer subnets, or not enough hosts.

In part (c) students were asked to give the number of total usable hosts with the chosen subnet mask, and to justify the answer. For this all that was necessary was to write that there were 8 subnets, and each subnet has 30 usable hosts hence there were 8 x 30 = 240. Note that while each subnet would have 32 addresses, the all zeros address is the same as the subnet address and so is not used, and the all 1's address is the broadcast address. A router that receives a data packet that has the all ones host address understands that the data packet is to be delivered to every host on the network.

In part (d) the correct CIDR notation for the network was 192.168.67.0/27, and in part (e) the address of each subnet, the corresponding range of usable host addresses, and the broadcast address were as follows:

Subnet 1	:	192.168.67.0; hosts 1-30, broadcast 31.
Subnet 2	:	192.168.67.32; hosts 33-62, broadcast 63.
Subnet 3	:	192.168.67.64; hosts 65-94, broadcast 95.
Subnet 4	:	192.168.67.96; hosts 97-126, broadcast 127
Subnet 5	:	192.168.67.128; hosts 129-158, broadcast 159
Subnet 6	:	192.168.67.160; hosts 161-190, broadcast 191
Subnet 7	:	192.168.67.192; hosts 193-222, broadcast 223
Subnet 8	:	192.168.67.224; hosts 225-254, broadcast 255

Question 3

In this question students were asked to describe advantages and disadvantages of the Internet of Things (IoT), and to discuss security issues. There was no right or wrong answer to this question; students were expected to do some independent research and come to their own conclusions. Marks were given for sensible, relevant points, clarity of thought, depth of knowledge, presenting a coherent argument and referencing evidence for any position taken.

While question 3 had no right or wrong answer, a good answer would have included some historical background on the IoT, perhaps noting how many devices were in service, and expectations for future growth, moving on to discuss the advantages that IoT devices bring to their users. After discussing advantages, a good answer would discuss disadvantages, and this should include that most IoT devices were effectively general-purpose computers, and as such could be programmed to do anything a computer could do. Since most IoT devices are lacking in good security this had led to IoT devices being used in DDoS attacks, including one of the largest such attacks to date. Poor security could also compromise the user's personal data and home security, or even their health and life if malicious attacks were made on safety critical devices.

When discussing advantages, most answers focussed on IoT devices in the home environment, but did not discuss more safety critical applications, such as in transport or medical devices. Very few answers discussed the long-term implications of the IoT, including that devices may be in service for much longer than a computer would be, hence in future security, issues can be expected to proliferate as aging devices with outdated operating systems remain in service. A good answer might have noted, following on from this point, the potential role of planning and government regulation in enforcing standards and protecting the public.

References

- Stallings, W. *Computer organization and architecture: designing for performance*. (Pearson Education Ltd, 2016) 10th global edition [ISBN 9781292096858].
- Patterson, D.A. and J.L. Hennessy *Computer organization and design: the hardware/software interface.* (Amsterdam: Elsevier/Morgan Kaufmann, 2017) ARM edition [ISBN 9780128017333].
- Comer, D.E. *Internetworking with TCP/IP Volume One.* (Pearson Education, 2014) Pearson new international edition [ISBN 9781292040813].