Feedback on Quiz # 0 (Formative)

The **Quiz # 0** was composed of **10** questions, which were randomly selected from a Question Bank of 20 questions. Answers and feedback comments for all of the questions are given below:

Q1.1	Consider the unsigned binary integer $(11110100001)_2$ What would be its equivalent representation in the octal number system?
(a)	$(7502)_8$
(b)	$(1641)_8$
(c)	$(3502)_8$
(d)	$(3641)_8$
Feedback: To obtain the octal equivalent, we take numbers in groups of 3, from right to left as (remember to add a zero at the left most position): 011 110 100 001 3 6 4 1 = $(3641)_8$	
Q1.2	Consider the unsigned binary integer $(11011011001101)_2$ What would be its equivalent representation in the hexadecimal number system?
(a)	$(6D9A)_{16}$
(b)	$(36CD)_{16}$
(c)	$(6D99)_{16}$
(d)	$(DB34)_{16}$
Feedback: To obtain the octal equivalent, we take numbers in groups of 4, from right to left as (remember to add two zero-s at the left most position): 0011 0110 1100 1101 3 6 C D = $(36CD)_{16}$	

Q2.1	On a hypothetical computer, real numbers are stored in a two's complement fixed-point binary format: • The first five bits represent the integer part of the number. • The last four bits represent the fractional part of the number. Compute the binary representation of the answer to the arithmetic equation $(11101.1101)_2$ - $(00110.0111)_2$
(a)	**101110110**
(b)	001000100
(c)	110100010
(d)	110111011

Feedback:

$(11101.1101)_2$ - $(00110.0111)_2$
= $(-16 + 8 + 4 + 1 + 1/2 + 1/4 + 1/16)_{10}$ - $(4 + 2 + 1/4 + 1/8 + 1/16)_{10}$
= (-2.1875)10 - $(6.4375)_{10}$
= $(-8.625)_{10}$
= 10111.0110 => **101110110**

Alternatively:

```
  11101.1101
- 00110.0111
```

Flip and add 1 to second argument (i.e. take 2's complement)

```
   11101.1101
+ 11001.1001
------------------
(1)10111.0110 => 101110110
```

Q2.2	On a hypothetical computer, real numbers are stored in a two's complement fixed-point binary format: • The first five bits represent the integer part of the number. • The last four bits represent the fractional part of the number. Compute the binary representation of the answer to the arithmetic equation $(00111.0111)_2 - (10101.1101)_2$
(a)	110001101
(b)	101101010
(c)	111010100
(d)	**100011010**

Feedback:

$(00111.0111)_2 - (10101.1101)_2$

$= (4+2+1+1/4+1/8+1/16)_{10} - (-16+4+1+1/2+1/4+1/16)_{10}$

$= (7.4375)_{10} + (10.1875)_{10}$

$= (17.625)_{10}$

= 10001.1010 => **100011010**

Alternatively:

```
  00111.0111
- 10101.1101
```

Flip and add 1 to second argument (take 2's complement)

```
   00111.0111
+01010.0011
  ------------------
 10001.1010 => 100011010
```

Q3.1	In Java, what is the result of the following: -7.5/0?
(a)	+infinity
(b)	**-infinity**
(c)	+0
(d)	-0
(e)	NaN
Feedback: The expression -7.5/0 will result in **-infinity**, as the numerator is negative.	
Q3.2	In Java, what is the result of the following: Math.sqrt(-1)?
(a)	+infinity
(b)	-infinity
(c)	+0
(d)	-0
(e)	**NaN**
Feedback: The expression Math.sqrt(-1) will result in **NaN**, as we cannot take square root of negative numbers.	

Q4.1	Consider the octal number $(76417)_8$ What would be its equivalent representation in the hexadecimal number system?
(a)	FA1E
(b)	FA0F
(c)	**7D0F**
(d)	7D1E

Feedback:

To convert an octal number to its hexadecimal equivalent, follow these steps:

$(76417)_8$

= 111 110 100 001 111 (groups of 3 binary numbers from right to left)
= 0111 1101 0000 1111 (groups of 4 binary numbers from right to left)
= 7 D 0 F
Therefore, the answer is $(7D0F)_{16}$.

Q4.2	Consider the hexadecimal number $(4A8C)_{16}$ What would be its equivalent representation in the octal number system?
(a)	**45214**
(b)	19084
(c)	11243
(d)	22506

Feedback:

To convert a hexadecimal number to its octal equivalent, follow these steps:
$(4A8C)_{16}$
= 4 A 8 C
= 0100 1010 1000 1100 (groups of 4 binary numbers from right to left)
= 0 100 101 010 001 100 (groups of 3 binary numbers from right to left)
= 4 5 2 1 4
Therefore, the answer is $(45214)_8$.

Q5.1	Most computer architectures have a set of registers. What are the advantages of instructions using registers compared to fetching operands and storing the result in memory?
(a)	Access to registers is faster than access to main memory.
(b)	The number of bits required to specify a register is fewer than to reference a main memory address. Therefore the instructions can be smaller (in size) and faster to load.
(c)	The number of registers can be changed dynamically, as the program requires.
(d)	Registers are cheaper than main memory.
(e)	Registers can represent 2's complement values more efficiently.
Feedback: Registers are much faster to access than main memory. Because there are only a small number of registers we only need a small address space to access them.	

Q5.2	Most computer architectures have a set of registers. What are the disadvantages when instructions need to access main memory compared to accessing registers?
(a)	Access to main memory is slower than access to registers.
(b)	The number of bits required to reference a memory address is higher than to specify a register. Therefore the instructions will be longer (in size) and slower to load.
(c)	Main memory is volatile whereas registers are persistent.
(d)	Main memory is more expensive than registers.
(e)	Main memory cannot represent 2's complement values.
Feedback: Main memory is much slower to access than registers. There are a small number of registers but main memory will be very large. Therefore the number of bits required to specify a main memory address will be much greater than to specify a register.	

Q6.1	A hypothetical computer stores real numbers in floating point format in 8-bit words: • The first bit is used for the sign of the number (1 is negative). • The second bit used for the sign of the exponent (1 is negative). • The next two bits are used for the magnitude of the exponent. (We **do not** use an offset to the exponent). • The final four bits are used for the magnitude of the mantissa. Convert the value $(11111011)_2$ in this representation into its decimal equivalent.
(a)	-1.0390625
(b)	-1.2109375
(c)	-0.0390625
(d)	**-0.2109375**

Feedback:
Given value: 11111011

Sign number bit = 1, so number is negative
Sign exponent bit = 1, so exponent is negative

Exponent = - $(11)_2$ = -3

Mantissa = 1011
So it can be written in standard form as = 1.1011 * 2^(-3)
Which is equivalent to = 0.0011011

Number = - $(0.0011011)_2$
= - (1/8 + 1/16 + 1/64 + 1/128)
= **- 0.2109375**

Q6.2	A hypothetical computer stores real numbers in floating point format in 8-bit words: • The first bit is used for the sign of the number (1 is negative). • The second bit used for the sign of the exponent (1 is negative). • The next two bits are used for the magnitude of the exponent. We **do not** add an offset to the exponent. • The final four bits are used for the magnitude of the mantissa. Convert the value $(11101001)_2$ in this representation into its decimal equivalent.
(a)	-1.390625
(b)	-1.109375
(c)	**-0.390625**
(d)	-0.109375

Feedback:
Given value: 11101001

Sign number bit = 1, so number is negative
Sign exponent bit = 1, so exponent is negative

Exponent = - $(10)_2$ = -2

Mantissa = 1001
So it can be written in standard form as = 1.1001 * 2^(-2)
Which is equivalent to = 0.011001

Number = - $(0.011001)_2$
= - (1/4 + 1/8 + 1/64)
= **-0.390625**

<table>
<tr><td>Q7.1</td><td>The range of integers that can be stored in a 10-bit register is __________________?</td></tr>
<tr><td>(a)</td><td>-255 to +254</td></tr>
<tr><td>(b)</td><td>-254 to +255</td></tr>
<tr><td>(c)</td><td>-511 to +512</td></tr>
<tr><td>(d)</td><td>-512 to +511</td></tr>
<tr><td>(e)</td><td>0 to 1023</td></tr>
<tr><td>(f)</td><td>0 to 1024</td></tr>
<tr><td colspan="2">Feedback:
There are two possibilities:
<u>For the 2's complement case:</u>
The most negative value that we can represent in 10-bit is 1000000000, where the most significant bit indicates the -ve sign. To find out its equivalent value in decimal, we can take 2's complement.

The inversion of 1000000000 is 0111111111.
Add 1 to 0111111111 = 0111111111+1 = $(1000000000)_2$ = $(512)_{10}$
So we can represent $(-512)_{10}$.

The largest positive value that we can represent in 10-bits = 0111111111 = $(511)_{10}$

In general, for n bits, the range is: -2^{n-1} --> $2^{n-1} - 1$

<u>For the unsigned case:</u>
Total possible number = 2^{10} = 1024. Therefore, it is 0 to 1023.
In general, for n bits, the range is: 0 to $(2^n - 1)$</td></tr>
<tr><td>Q7.2</td><td>The range of integers that can be stored in a 12-bit register is __________________?</td></tr>
<tr><td>(a)</td><td>0 to 4096</td></tr>
<tr><td>(b)</td><td>0 to 4095</td></tr>
<tr><td>(c)</td><td>-1024 to +1023</td></tr>
<tr><td>(d)</td><td>-1023 to +1024</td></tr>
<tr><td>(e)</td><td>-2048 to +2047</td></tr>
<tr><td>(f)</td><td>-2047 to +2048</td></tr>
<tr><td colspan="2">Feedback:
There are two possibilities:
<u>For the 2's complement case:</u>
The most negative value that we can represent in 12-bit is 100000000000, where the most significant bit indicates the -ve sign. To find out its equivalent value in decimal, we can take 2's complement.

The inversion of 100000000000 is 011111111111.
Add 1 to 011111111111 = 011111111111+1 = $(100000000000)_2$ = $(2048)_{10}$
So we can represent $(-2048)_{10}$.</td></tr>
</table>

The largest positive value that we can represent in 12-bits = 011111111111 = $(2047)_{10}$

In general, for n bits, the range is: -2^{n-1} --> $2^{n-1} - 1$

For the unsigned case:
Total possible number = 2^{12} = 4096. Therefore, it is 0 to 4095.
In general, for n bits, the range is: 0 to $(2^n - 1)$

Q8.1	The data-type Float (32-bit total = 23-bit mantissa, 8-bit exponent) gives us about 7-8 significant decimal digits. The data-type Double (64-bit total = 52-bit mantissa, 11-bit exponent) gives us about 15-16 significant decimal digits. Let’s create a new data-type called Triple, which has 96-bits total; a 78-bit mantissa and a 17-bit exponent. Roughly how many significant decimal digits will this type give us?
(a)	21-22 significant decimal digits
(b)	**23-24 significant decimal digits**
(c)	25-26 significant decimal digits
(d)	27-28 significant decimal digits

Feedback:
The mantissa has 78-bits, so (when we include the hidden bit) we get 79 significant bits; or 2^79 values.

2^79 = 6.044629098 x 10^23
(we can work this out through various methods, e.g. using logarithms of base 10)

Therefore we have roughly 23-24 significant decimal digits.

Q8.2	The data-type Float (32-bit total = 23-bit mantissa, 8-bit exponent) gives us about 7-8 significant decimal digits. The data-type Double (64-bit total = 52-bit mantissa, 11-bit exponent) gives us about 15-16 significant decimal digits. Let’s create a new data-type called Triple, which has 96-bits total; a 84-bit mantissa and an 11-bit exponent. Roughly how many significant decimal digits will this type give us?
(a)	21-22 significant decimal digits
(b)	23-24 significant decimal digits
(c)	**25-26 significant decimal digits**
(d)	27-28 significant decimal digits

Feedback:
The mantissa has 84-bits, so (when we include the hidden bit) we get 85 significant bits; or 2^85 values.

2^85 = 3.868562623 x 10^25
(we can work this out through various methods, e.g. using logarithms of base 10)

Therefore we have roughly 25-26 significant decimal digits.

Q9.1	Consider that you are designing a world-wide database for keeping the passport records of every human being alive on earth. Let's assume that the current human population is approx. 9 billion people. For each person, you will be keeping the following information in your database:

Database Field	**Space Required**	**Memory Units (for reference)**
Passport Type	2 bytes	**Units of Computer Memory Measurements** 1 Bit = Binary Digit 8 Bits = 1 Byte 1024 Bytes = 1 KB [Kilo Byte] 1024 KB = 1 MB [Mega Byte] 1024 MB = 1 GB [Giga Byte] 1024 GB = 1 TB [Terra Byte] 1024 TB = 1 PB [Peta Byte] 1024 PB = 1 EB [Exa Byte] 1024 EB = 1 ZB [Zetta Byte] 1024 ZB = 1 YB [Yotta Byte] 1024 YB = 1 Bronto Byte 1024 Brontobyte = 1 Geop Byte **Geop Byte** is the Highest Memory.
Passport Number	14 bytes	
First Name	48 bytes	
Last Name	48 bytes	
Nationality	36 bytes	
Date of Birth	8 bytes	
Expiry Date	8 bytes	

Estimate the total amount of space that you will need to store the above information for all the people alive on earth today?

(a)	**1.34 TB**
(b)	13.4 TB
(c)	1.16 TB
(d)	11.6 TB
(e)	1.48 TB

Feedback:

Considering we have approx 9 billion people i.e. 9 x 10^9 and we need to store the above information for each person. We will need to store:
2 + 14 + 48 + 48 + 36 + 8 + 8 = 164 bytes for every person.

So the total amount of space needed is: 164 x 9 x 10^9 = 1.476×10^{12} bytes
Now, lets see how much this space is, once we convert it to memory units:
1.476×10^{12} / 1024 = 1441406250 KB

1441406250 KB / 1024 = 1407623.29 MB
1407623.29 MB / 1024 = 1374.63 GB
1374.63 GB / 1024 = **1.34 TB (approx)**

Q9.2	Consider that you are designing a world-wide database for keeping the passport records of every human being alive on earth. Let's assume that the current human population is approx. 8 billion people. For each person, you will be keeping the following information in your database:

Database Field	**Space Required**	**Memory Units (for reference)**
Passport Type	4 bytes	
Passport Number	16 bytes	
First Name	40 bytes	
Last Name	56 bytes	
Nationality	24 bytes	
Date of Birth	10 bytes	
Expiry Date	10 bytes	

Units of Computer Memory Measurements

1 Bit	= Binary Digit
8 Bits	= 1 Byte
1024 Bytes	= 1 KB [Kilo Byte]
1024 KB	= 1 MB [Mega Byte]
1024 MB	= 1 GB [Giga Byte]
1024 GB	= 1 TB [Terra Byte]
1024 TB	= 1 PB [Peta Byte]
1024 PB	= 1 EB [Exa Byte]
1024 EB	= 1 ZB [Zetta Byte]
1024 ZB	= 1 YB [Yotta Byte]
1024 YB	= 1 Bronto Byte
1024 Brontobyte	= 1 Geop Byte

Geop Byte is the Highest Memory.

Estimate the total amount of space that you will need to store the above information for all the people alive on earth today?

(a)	**1.16 TB**
(b)	11.6 TB
(c)	1.34 TB
(d)	13.4 TB
(e)	1.28 TB

Feedback:

Considering we have approx 8 billion people i.e. 8 x 10^9 and we need to store the above information for each person. We will need to store:
4 + 16 + 40 + 56 + 24 + 10 + 10 = 160 bytes for every person.

So the total amount of space needed is: 160 x 8 x 10^9 = 1.28×10^{12} bytes
Now, lets see how much this space is, once we convert it to memory units:
1.28×10^{12} / 1024 = 1250000000 KB

1250000000 KB / 1024 = 1220703.12 MB
1220703.12 MB / 1024 = 1192.09 GB
1374.63 GB / 1024 = **1.16 TB (approx)**

<table>
<tr><td>Q10.1</td><td>A 32-bit computer system has 4 GB of memory installed in it; these represent addresses $(00000000)_{16}$ - $(FFFFFFFF)_{16}$. However, the system programmer is told that she can only use memory from $(B1002000)_{16}$ to $(EA001FFF)_{16}$. The memory below the address $(B1002000)_{16}$ is unavailable and the memory above $(EA001FFF)_{16}$ is also unavailable.
How much memory is available to the system programmer?</td></tr>
<tr><td>(a)</td><td>912 MB</td></tr>
<tr><td>(b)</td><td>956.30 MB</td></tr>
<tr><td>(c)</td><td>37.19 MB</td></tr>
<tr><td>(d)</td><td>39 MB</td></tr>
<tr><td>(e)</td><td>933888 KB</td></tr>
<tr><td>(f)</td><td>956301.31 KB</td></tr>
<tr><td colspan="2">Feedback:
EA001FFF
− B1002000

38FFFFFF + 1
= 39000000
$(39000000)_{16} = (956301312)_{10}$
956301312 / 1024 = 933888 KB
933888 KB / 1024 = 912 MB</td></tr>
<tr><td>Q10.2</td><td>A 32-bit computer system has 4 GB of memory installed in it; these represent addresses $(00000000)_{16}$ - $(FFFFFFFF)_{16}$. However, the system programmer is told that she can only use memory from $(CA009000)_{16}$ to $(FF008FFF)_{16}$. The memory below the address $(CA009000)_{16}$ is unavailable and the memory above $(FF008FFF)_{16}$ is also unavailable.
How much memory is available to the system programmer?</td></tr>
<tr><td>(a)</td><td>848 MB</td></tr>
<tr><td>(b)</td><td>889.19 MB</td></tr>
<tr><td>(c)</td><td>33.38 MB</td></tr>
<tr><td>(d)</td><td>35 MB</td></tr>
<tr><td>(e)</td><td>868352 KB</td></tr>
<tr><td>(f)</td><td>889192.45 KB</td></tr>
<tr><td colspan="2">Feedback:
FF009000
− CA009000

34FFFFFF + 1
= 35000000
$(35000000)_{16} = (889192448)_{10}$
889192448 / 1024 = 868352 KB
868352 KB / 1024 = 848 MB</td></tr>
</table>