CSM 61C Spring 2020

Number Representation

Exam Question Compilation Solutions

This document is a PDF version of old exam questions by topic, ordered from least difficulty to greatest difficulty.

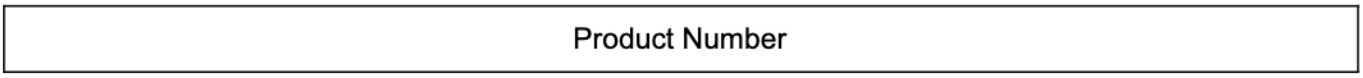
Questions:

- Summer 2019 Midterm 1 Q1
- Fall 2019 Quest Q1, Q2, Q3
- Spring 2015 Final M1-1AB
- Fall 2018 Quest Q1
- Spring 2018 Midterm 1 Q1
- Spring 2018 Final Q1A, Q1B
- Summer 2018 Midterm 1 Q1
- Fall 2018 Midterm Q1 A-D
- Fall 2015 Final MT2-5
- Fall 2019 Final Q2

Question 1: We're bored of Euclid. (Number Representation)

Morgan, Nick, and Branden are disappointed in the selection of food available near Soda, so they open a store selling many different kinds of products. They need YOUR help to come up with a barcode scheme for everything they sell.

1. Branden wants to assign each product in the store its own unique number. This will be encoded as the barcode. If they sell 29 unique items, what is the smallest number of bits they can use to encode the barcode?



Barcode = _____5 (ceiling(log2(29)))_____ bits

2. When the store runs out of a particular item, it would be helpful to see what other kinds of that item there are in stock. Morgan proposes adding a "product group" field to the barcode in addition to the existing product number. Note that now each product number does not need to be globally unique and instead just needs to be unique within its product group. If there are 5 unique product groups, what is the smallest number of bits they can use for the product group field?

Product Group = ____3 (ceiling(log2(5)))____ bits

3. We expand to have 12 product groups. The largest has 15 items in it while the smallest has one item. Nick argues the entire barcode can now be condensed to only 6 bits without losing product grouping or unique identifiers. Is he correct? If yes, explain why, if no, what is the actual minimum size?

[ ] Yes, he is correct [X] No, he is incorrect

____At least 4 bits for items ceiling(log2(15))_____

____At least 4 bits for groups ceiling(log2(12))_____

____So it can't be less than 8 bits_____

4. The team decides on the following barcode field sizes (which may or may not reflect your answers above).

Product Group (4 bits)	Product Number (5 bits)

Morgan loads all the barcodes into the database but runs into a problem; she'd like to reserve the barcode of all zeros (so, product group = 00...0, product number = 00...0) for products that are out of stock. Assuming there are 8 product groups holding between 1 and 31 products each, can she implement the all-zero barcode without adding bits to the scheme? Explain.

[X] Yes, she can [ ] No, she can't

____There are 4 bits for product group (possible 2^4 = 16 groups)_____

_____But only 12 groups are used, leaving 4 possible groups unclaimed_____

__So, we just make product group = 0b0000 the "empty group", and product number 0b00000 is in it__

5. Business is booming and the team has the opportunity to expand! They purchase a new store and modify the barcode to keep track of products sold at store 0 and store 1 separately.

Store Code (1 bit)	Product Group (4 bits)	Product Number (5 bits)

Assuming the same item sold across stores differs ONLY in the top bit (that is, they have the same product group and product number regardless of the store they're sold at) what is the maximum number of items Morgan, Nick, and Branden can uniquely identify with this barcode scheme? You may leave your answer as an unsimplified equation.

2^9 = 512 (Unique items are product group and product number, not store) Unique Items

Q1) [10 Points] **Negate** the following **nibble *binary/hex*** numbers, or write N/A if not possible. Remember to write your answer in the appropriate base. (A nibble is 4 bits)

(Unsigned) **0b0101**	*(Bias = -7)* **0b0100**	*(Bias = -7)* **0xF**	*(Two's Comp)* **0b1100**	*(Two's Comp)* **0xA**
0b N/A	**0b1010**	**0x N/A**	**0b0100**	**0x6**

...scratch space below...

(unsigned) 0b0101) We want to negate this value but we are using an unsigned representation so this is impossible, so the answer is N/A.
(Bias = -7) 0b0100) The value given's unsigned representation is 4. We we will plug it into the equation given in question 3: unsigned + bias value = final value, thus 4 + -7 = -3. We want to invert this so we want to represent the value 3. To find our unsigned representation, we can just plug the values back into the above equation: unsigned + -7 = 3 ⇒ unsigned = 10 thus 0b1010.
(Bias = -7) 0xF) 0xF == 0b1111. We will do the same as above: unsigned + bias value = final value ⇒ 15 + -7 => 8. (we also might have remembered that the standard biased notation gives the extra number to the positive side, so the range is [-7,8] and 0b1111 is the biggest bias number so it's 8) So the value we need to invert is 8, so we want to represent -8. The issue is since our bias is only -7, we cannot represent -8 even if we have our unsigned value as zero, so the answer is N/A.
(Two's Comp) 0b1100) For twos complement, we do not need to worry about converting this to decimal. We can use the trick of flipping the bits and adding one thus: ~0b1100 + 1 = 0b0011 + 1 = 0b0100
(Two's Comp) 0xA) First we have to convert this value into binary. We know that A = 10 in decimal thus the binary representation is 0b1010. Now we can apply the trick described for the last question: ~0b1010 + 1 = 0b0101 + 1 = 0b0110. We can now convert this back to hex, thus 0x6.

Q2) [6 Points] Which of the following sums will yield an **arithmetically incorrect result** when computed with **two's complement nibbles**?

Correct ● Incorrect ○	Correct ● Incorrect ○	Correct ○ Incorrect ●
0xD + 0xE + 0xF	**0x7 + 0x8**	**0x3 + 0x5**

...scratch space below...

For all of these problems, we are given nibbles which are just 4 bits. We are interpreting them as Two's Complement. This means we can only represent the values in the range [-8, 7]. Thus if we get a result which goes beyond this, we cannot represent this.
0xD + 0xE + 0xF) -3 + -2 + -1 = -6, which is in our range, thus the result is correctly representable.
0x7 + 0x8) 7 + -8 = -1 which is in our range, thus the result is correctly representable
0x3 + 0x5) 3 + 5 = 8 which is NOT in our range, thus we cannot represent it correctly, so it is incorrect.

Q3) [12 Points] For each of the following representations, what is the *fewest number of bits* needed to cover the given range, which is inclusive of the endpoints (e.g., [1, 4] is the numbers 1, 2, 3 and 4). Write "N/A" if it is impossible. For the **Bias *Value*** (final value = unsigned + bias value), we'll let YOU specify whatever offset you wish to minimize the total number of bits needed for the Bias encoding.

Range	Unsigned	One's Comp	Two's Comp	Sign&Mag	Bias	Bias *Value*
`[ 0, 10 ]`	4	5	5	5	4	0
`[ -4, -1 ]`	N/A	4	3	4	2	-4
`[ 1, 4 ]`	3	4	4	4	2	1

...scratch space below...

Here are the ranges of each of the representations given n bits:
Unsigned: [0, 2^n–1]. So [0,10] needs 4 bits [0,15] with 5 unused bit patterns 11-15, [-4,1] is impossible, and [1,4] needs 3 bits [0,7] with {0,5,6,7} bit patterns unused.
One's Complement Range: [-(2^{n-1}–1), 2^{n-1}–1]. [1,4] needs 4 bits [-7,7]
Two's Complement Range: [-2^{n-1}, 2^{n-1}–1]. [-4,1] needs 3 bits [-4,3], but [1,4] needs 4 bits [-8,7].
Sign&Mag: [-(2^{n-1}–1), 2^{n-1}–1]. [0,10] needs 5 bits [-15,15], and [-4,1] needs 4 bits [-7,7]
Bias: [Bias Value, 2^n–1 + Bias Value]. A Bias Value of 0 is just unsigned, so that's 4 bits [0,15].To do -4,1, many answers work, but the simplest is Bias Value = -4, and only 2 bits are needed to represent [-4,-1] -- nice and efficient! This is the same idea with [1,4], if you only have 2 bits, you'd better bias to the smallest number to fit the representation perfectly, so Bias Value = 1.

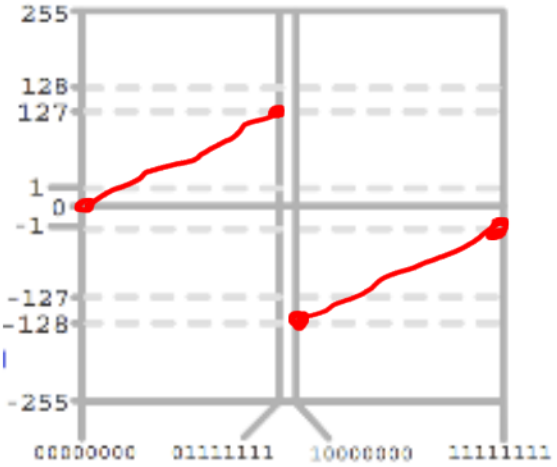
M1-1: *I smell a potpourri section covering midterm one…* (9 points)

a) Which of the following number representations give **`0xFFFFFFFE`** the **most positive** value when converted to decimal?

Explanation on NEXT Page

A) Bias (with standard bias) B) Unsigned C) Two's complement D) Sign and Magnitude

b) Consider a plot that shows the mapping between 8-bit two's complement binary numbers and their decimal equivalents (i.e. binary is on the x-axis and decimal is on the y-axis). Fill in the plot to the left and answer the following questions.



i) Fill in the plot to the left. Translate the numbers on the x-axis to decimal

ii) Describe (in binary) where discontinuities occur in the plot, if any:

The discontinuity occurs between 01111111 to 10000000, as shown by the graph.

iii) What are the most positive and most negative decimal values that this representation can store?

Most Positive: 127

Most Negative: -128

1A) In both signed and two's complement, the number 0xFFFFFFFE is negative. In unsigned format, $0xFFFFFFFE = 2^{32} - 2$

In Biased format, the bias will be $2^{31} - 1$, so $0xFFFFFFFE = (2^{32} - 2) - (2^{31} - 1)$.

This is less than our unsigned representation, yielding our solution.

UC Berkeley Fall 2018 CS61C Quest Answers

Q1a) With **3 bits**, how do we represent **-2**? If it can't be done, select "**N/A**". (Select ONE per row)

	000	001	010	011	100	101	110	111	N/A
Unsigned	○ 0	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6	○ 7	●
Sign/Magnitude	○ +0	○ 1	○ 2	○ 3	○ -0	○ -1	● -2	○ -3	○
One's Complement	○ +0	○ 1	○ 2	○ 3	○ -3	● -2	○ -1	○ -0	○
Two's Complement	○ 0	○ 1	○ 2	○ 3	○ -4	○ -3	● -2	○ -1	○
Bias; use bias of $-(2^{N-1}-1)$ from lecture	○ -3	● -2	○ -1	○ 0	○ 1	○ 2	○ 3	○ 4	○

Q1b) Convert 26_{10} to the following bases (and remove any leading zeros)

Binary	*Hex*
0b11010 (**1***16+**1***8+**0***4+**1***2+**0***1=26)	**0x1A** (**1***16+10(=**A**)*1=26)

Q1c) Add these *Two's Complement* nibbles:

```
  1001
+ 1011
------
 10100
```

Does it overflow a nibble? (Select ONE)

● **Yes [1001 (-7) + 1011 (-5) = -12. .Since 2s complement nibbles can only represent [-8,7] it certainly can't hold -12. You don't have to even do the addition to answer this correctly]**

○ No

Problem 1 ***Number Representation***

(a) Translate the following decimal numbers into 8-bit two's complement and unsigned binary representation in the table below. If a translation is not possible, please write "N/A". **Write your final answer in hexadecimal format.**

Solution:

Decimal Number	Two's Complement	Unsigned Number
10	0x0A	0x0A
129	None	0x81
-12	0xF4	None

(b) Suppose that we define the negative of x to be just $\overline{x}$. We will call this new number representation scheme **one's complement**. Note that the top bit of a one's complement number still denotes the number's sign (0 for positive, 1 for negative).

Translate the following decimal numbers into 8-bit one's complement binary representation. If the translation is not possible, please write "N/A". **Write your final answer in hexadecimal format.**

Solution:

Decimal Number	One's Complement
13	0x0D
-6	0xF9

(c) What is the range of integers (in decimal format) that we can represent with an n-bit one's complement binary number?

Solution: $[-2^{n-1}+1, 2^{n-1}-1]$

Problem 1 ***[MT1-1] Number Rep***

Answer the following questions about number representation:

(a) Unsigned Base 4

(i) What is the range that a 4 digit unsigned base 4 number can represent? Write the bounds in decimal.

Solution: $0000_4 \sim 3333_4 = 0_{10} \sim 255_{10}$

(ii) Convert 107_{10} to unsigned base 4.

Solution: $107_{10} = 64 + 16 * 2 + 4 * 2 + 3 = 1223_4$

(b) Signed Base 4

(i) Suppose we wanted to use a bias in order to represent negative numbers in base4. If we are working with a 4 digit base 4 number, what should we choose as our bias? (Our bias should create equal amounts of negative and positive numbers for our range. If this is not possible, select a bias that will result in 1 more negative number than positive numbers). Express your answer in decimal.

Solution: 255/2 = 127. So the bias is -128 to favor negative numbers.

(ii) Suppose rather than using a bias notation, we decide to do the following.

For each base 4 number, we will reserve the most significant digit to strictly be used as a sign bit. A digit value of 1 will indicate a negative number, and a digit value of 0 will indicate a positive number. Any other values will result in an invalid number. For instance:

$$0003_4 = +3 \qquad 1003_4 = -3 \qquad 2003_4 = Invalid$$

How many valid representation can we represent with a 4 digit base 4 number using this scheme?

Solution: 2*4*4*4 = 128

Question 1: Number Representation and Floating Point (12 pts)

Given the following bit string 0b1111 1100, answer the following questions:

1) What is this bitstring's value if it was interpreted as an **unsigned number**?
(2^8 - 1) - 3 = 252

2) What is this bitstring's value if it was interpreted in **two's complement**?
(Flip all the bits and add 1) * -1 = (0b0000 0011 + 1) * -1 = -4

3) Suppose the bit string was represented as **fixed point** where the bits following the dot (.) represent the base (2) to the power of a negative exponent. What number does the bitstring 0b1111.1100 represent?
0b1111 = 15; 0b.1100 = 0.5 + 0.25 = 0.75 -> 15.75

4) Now let's devise a scheme for interpreting fixed point numbers as positive or negative values. Complete the following sentence:
Given an 8-bit fixed point bitstring 0bXXXX.XXXX with a value of Y, in order to compute -Y, we must flip all the bits of Y and add:
The idea in two's complement is that you add 0b00...001 to the flipped bitstring.
0b0000.0001 in fixed point has a value of 1/16. 1/16

5) What is the value of 0b1111.1100 given the **two's complement fixed point** representation described above?
. (Flip all the bits and add 1/16) * -1 = (0b0000.0011 + 1/16) * -1 = -0.25

6) You are given the following field breakdown and specifications of an 8-bit floating point, which **follows the same rules** as standard 32-bit IEEE floats, except with different field lengths:

Sign: 1 bit
Exponent: 3 bits
Significand: 4 bits

Exponent Value	Significand Value	Floating Point Value
Smallest	Zero, Non-Zero	±0, Denormalized
Largest	Zero, Non-zero	±Infinity, NaN

What is the floating point value of 0b1111 1100:

Exponent field is the largest exponent (0b111) and the significant is non-zero-> NaN

7) We now modify the floating point description in part 6 so that the exponent field is now in **two's complement** instead of in bias notation. Compute the floating point value of 0b1111 1100.

Exponent: 0b111 = -1, Signif: 0.75 = (-1) x 2^-1 x 1.75 = -0.875

Q1) Float, float on… (6 points)

Consider an 8-bit "minifloat" SEEEEMMM (1 sign bit, 4 exponent bits, 3 mantissa bits). All other properties of IEEE754 apply (bias, denormalized numbers, ∞, NaNs, etc). The bias is -7.

a) How many NaNs do we have? ____6____

6, E is all 1s, MM is anything other than 0: {01, 10, 11} then double for sign bit

b) What is the bit representation (in hex) of the next minifloat bigger than the minifloat represented by the hexadecimal value is **`0x3F`**? ___0x40___

0x40, you don't even have to do any math here, it's the next bit pattern up

c) What is the bit representation (in hex) of the encoding of -2.5? ___C1___

Standard float calculation: -2.5 = -10.1 = -1.01 x 10^1 ⇒ 1 EEEEE 01, where EEEEE + -15 = 1 so EEEEE = 16, so it's 1 10000 01 = **C1**

d) What does **`should_be_a_billion()`** return? *(assume that we always round down to 0)* ___8.0____

This is basically the value when you start counting by 2s, since once you start counting by 2s and always round down, your sum doesn't increase. There are 2 mantissa bits, so there are always 4 numbers in the range [2^i, 2^{i+1}). So you ask yourself, what gap has exactly 4 numbers between consecutive values of [2^i, 2^{i+1}), meaning when are you counting by 1? Easy, [4-8) ⇒ {4, 5, 6, 7}. When you get to 8 you're counting by 2s since you have to cover 8 to 16 with only 4 numbers: {8, 10, 12, 14}. **So it's 8.0** and you didn't have to do any work trying to encode numbers in and out of minifloats, since that was what question c was supposed to be about.

```
minifloat should_be_a_billion() {
        minifloat sum = 0.0;
        for (unsigned int i = 0; i < 1000000000; i++) { sum = sum + 1.0; }
        return(sum);
}
```

MT2-5: What is the *floating point* of *complex* numbers? (5 points)

We realize that you want to represent complex numbers, which are in the form $a + bi$, where a is the real component, b is the imaginary component, and the magnitude is $\sqrt{a^2 + b^2}$.

We create a 16-bit representation for storing both the real and imaginary components as floating point numbers with the following form: The first 8 bits will represent the real component, and the latter 8 bits will represent the complex component. Our new representation will look like:

Sign	Exponent	Significand	Sign	Exponent	Significand
15	14-12	11-8	7	6-4	3-0

Bits per field:

Sign: 1
Exponent: 3
Significand: 4
Everything else follows the IEEE standard 754 for floating point, except in 16 bits

Bias: 3

11/17

a. Convert 0xB248 into the complex number form a + bi

0xB248 -> 0b1011 0010 0100 1000

For the real part, significand is 0010, exponent is 011, sign is 1. If we convert the exponent to bias, we actually get an exponent of 0. Thus, converting the significand to normalized form gives -1.001 in binary, which is -1.125 in decimal.

For the imaginary part, significand is 1000, exponent is 100, sign is 0. If we convert the exponent to bias, we get an exponent of 1. Thus, converting the significand to normalized form gives 1.1 * 2^1 in binary (which equals 11.0… in binary), which is just 3.

Thus -1.125 + 3i is the answer.

b. What is the smallest positive number you can represent with a nonzero real component and zero complex component?

For the positive component, the smallest exponent possible is 0 and the smallest significand is 0001. Remember, this is denormalized because the exponent is zero, thus the bias is always the negative of one smaller than the positive bias (-2). Converting this to normalized form gives 0.0001*2^-2 in binary, which is 0.000001 = 2^-6

Recall the following floating point representation from the midterm:

Sign	Exponent	Significand
15	14-9	8-0

Bits per field:

Sign: 1
Exponent: 6
Significand: 9
Everything else follows the IEEE standard 754 for floating point, except in 16 bits

c. Ignoring infinities, which of the two representations presented above can represent a number with the larger magnitude.

The midterm floating point representation can represent 2^6 in magnitude. For the complex number representation, if you had the largest exponent in the real and the imaginary part possible, and took the magnitude per the equation in the directions, your exponent would be 2^5 (due to the square root). Thus, the midterm floating point representation is larger in magnitude.

Q2) Open to Interpretation **(11 pts = 2 + 3 + 4 + 2)**

Let's consider the hexadecimal value **`0xFF000003`**. How is this data interpreted, if we treat this number as…

a) an array `A` of unsigned, 8-bit numbers? Please write each number in **decimal,** assume the machine is **big endian**, and write **`A[0]`** on the left, **`A[3]`** on the right.

255, 0, 0, 3

SHOW YOUR WORK HERE

Big endian means the big part of the number is at A[0] (unlike how we would normally store the number) so that means the number reads left to right. The first byte is 0xFF, which is 2^8-1 = 255, the second and third are 0x00, which are zero, and the last is 0x03, which is 3.

b) an IEEE-754 single-precision floating point number?

$-(2^{127} + 2^{105} + 2^{104})$

SHOW YOUR WORK

1|111 1111 0|00000...011

exp=254, so number is -1.00...11 x $2^{254-127}$

$-2^{127}\ (1 + 2^{-22} + 2^{-23})$

$-(2^{127} + 2^{105} + 2^{104})$

c) a RISC-V instruction? If there's an immediate, write it in decimal.

`lb x0 -16(x0)`

SHOW YOUR WORK

The opcode is 0b0000011 and the func3 is 0b000, which corresponds to the "lb" instruction. "lb" is an I-type instruction, so we extract rd = 0b00000, rs1 = 0b00000, Imm = 0b111111110000. The register 0b00000 is the x0 register. Finally, we calculate the immediate, taking care to note that immediates are stored in two's complement signed form. Negating the immediate yields 0b000000001111+1 = 0b000000010000 = 16, so the immediate is -16. We then write the instruction in lb format.

d) a **`(uint32_t *)`** variable **`c`** in **little-endian** format, and we call **`printf((char *) &c)`**? If an error or undefined behavior occurs, write "Error". If nothing is printed, write "Blank". Please refer to the ASCII table provided on your reference sheet. For non-printable characters, please write the value in the Char column from the table. For example, for a single backspace character, you would write "**BS**".

ETX

SHOW YOUR WORK

Since the data is in little-endian format, the first byte printed is 0x03, which corresponds to ETX. The second character is 0x00, which is NULL, the null terminator. printf doesn't read past the first null terminator, so we finish printing after we write ETX. Note that the VALUE of c is our number in little-endian format which is why when we do &c, we are saying that value is a string when plugged into printf.