

1. Consider the uninterpreted bit string associated with the hexadecimal string $c0de$. What decimal value would be encoded by this bit string represent if we interpreted this number as:

Solution: Before we present solutions to the individual problems listed in each part of this question, let's translate the hexadecimal string $c0de$ into a raw, uninterpreted bit string. We focus on the pertinent rows of the hexadecimal nibble chart provided on page 2 of this exam:

Table 3: Hexadecimal Nibble Chart

| Decimal | 4-bit binary nibble | Lowercase Hexadecimal |
|---------|---------------------|-----------------------|
| 0 | 0000 | 0 |
| 12 | 1100 | c |
| 13 | 1101 | d |
| 14 | 1110 | e |

Then, we translate our four-digit hexadecimal string into a 16-bit binary string as follows:

$$c0de = 1100\ 0000\ 1101\ 1110.$$

We can now use this string to respond to each part of this problem.

- (1A) an unsigned binary integer?

Solution: Recall that for an $(m+1)$ -bit sequence of binary digits in the form $y = b_m b_{m-1} \dots b_2 b_1 b_0$ where each digit b_i is either 0 or 1, if we interpreted y as an unsigned binary integer, then we could write the following sum

$$y = b_m \cdot 2^m + b_{m-1} \cdot 2^{m-1} + \dots + b_1 \cdot 2^1 + b_0 \cdot 2^0 = \sum_{i=0}^m b_i \cdot 2^i$$

For the 16-bit number above, we have $(m+1) = 16$ and $m = 15$. Moreover, we can create a value box (as seen below) to help compute the interpreted value of our original bit string. To this end consider

| | | | | | | | | | | | | | | | | |
|------------------|----------|----------|----------|----------|----------|----------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| Weight (power) | 2^{15} | 2^{14} | 2^{13} | 2^{12} | 2^{11} | 2^{10} | 2^9 | 2^8 | 2^7 | 2^6 | 2^5 | 2^4 | 2^3 | 2^2 | 2^1 | 2^0 |
| Weight (decimal) | 32 768 | 16 384 | 8 192 | 4 096 | 2 048 | 1 024 | 512 | 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| Bit value | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |

Then, using this value Table, we see that our value of our original hexadecimal string via the unsigned integer interpretation is

$$\begin{aligned}
 (c0de)_{16} &= (1100\ 0000\ 1101\ 1110)_2 \\
 &= 2^{15} + 2^{14} + 2^7 + 2^6 + 2^4 + 2^3 + 2^2 + 2^1 \\
 &= 32\ 768 + 16\ 384 + 128 + 64 + 16 + 8 + 4 + 2 \\
 &= \boxed{(49\ 374)_{10}}
 \end{aligned}$$

(1B) a signed integer in signed-magnitude representation?

Solution: If, on the other hand, we interpret the sequence of $(m+1)$ -bits given by $b_m b_{m-1} \dots b_2 b_1 b_0$ as a signed integer encoded using the signed-magnitude representation, then we have a different map given by

$$y = (-1)^{b_m} \cdot (b_{m-1} \cdot 2^{m-1} + \dots + b_1 \cdot 2^1 + b_0 \cdot 2^0)$$

$$= \underbrace{(-1)^{b_m}}_{\text{sign}} \cdot \underbrace{\left(\sum_{i=0}^{m-1} b_i \cdot 2^i \right)}_{\text{magnitude}}$$

For the 16-bit number above, we have $(m+1) = 16$ and $m = 15$. Once again, we can create a value box to help compute the interpreted value of our original bit string. To this end consider

| Weight (power) | $(-1)^{b_m}$ | 2^{14} | 2^{13} | 2^{12} | 2^{11} | 2^{10} | 2^9 | 2^8 | 2^7 | 2^6 | 2^5 | 2^4 | 2^3 | 2^2 | 2^1 | 2^0 |
|------------------|--------------|----------|----------|----------|----------|----------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| Weight (decimal) | -1 | 16 384 | 8 192 | 4 096 | 2 048 | 1 024 | 512 | 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| Bit value | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |

Then, using this value Table, we see that our value of our original hexadecimal string via the signed-magnitude interpretation is

$$\begin{aligned} (c\ 0\ d\ e)_{16} &= (1100\ 0000\ 1101\ 1110)_2 \\ &= -1 \times (2^{14} + 2^7 + 2^6 + 2^4 + 2^3 + 2^2 + 2^1) \\ &= -1 \times (16\ 384 + 128 + 64 + 16 + 8 + 4 + 2) \\ &= \boxed{(-16\ 606)_{10}} \end{aligned}$$

(1C) a signed integer in twos complement representation?

Solution: The twos complement interpretation of the $(m+1)$ -bit string given by $b_m b_{m-1} \dots b_2 b_1 b_0$ is defined using the following map

$$\begin{aligned} y &= -b_m \cdot 2^m + (b_{m-1} \cdot 2^{m-1} + \dots + b_1 \cdot 2^1 + b_0 \cdot 2^0) \\ &= -b_m \cdot 2^m + \left(\sum_{i=0}^{m-1} b_i \cdot 2^i \right) \end{aligned}$$

For the 16-bit number above, we have $(m+1) = 16$ and $m = 15$. Here we use our value box a third time to help compute the interpreted value of our original bit string. To this end consider

| | | | | | | | | | | | | | | | | |
|------------------|-----------|----------|----------|----------|----------|----------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| Weight (power) | -2^{15} | 2^{14} | 2^{13} | 2^{12} | 2^{11} | 2^{10} | 2^9 | 2^8 | 2^7 | 2^6 | 2^5 | 2^4 | 2^3 | 2^2 | 2^1 | 2^0 |
| Weight (decimal) | -32 768 | 16 384 | 8 192 | 4 096 | 2 048 | 1 024 | 512 | 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| Bit value | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |

Then, using this value box, we see that our value of our original hexadecimal string via the twos complement interpretation is

$$\begin{aligned} (\text{c 0 d e})_{16} &= (1100\ 0000\ 1101\ 1110)_2 \\ &= -2^{15} + 2^{14} + 2^7 + 2^6 + 2^4 + 2^3 + 2^2 + 2^1 \\ &= -32\,768 + 16\,384 + 128 + 64 + 16 + 8 + 4 + 2 \\ &= \boxed{(-16\,162)_{10}} \end{aligned}$$

(1D) a fixed point number using data class `ufixed162(8,8)`?

Solution: Let's move onto our fixed point data formats. Recall that when we spoke about our $(m + 1)$ -bit fixed-point representations, we defined a fixed-point data class

$$x = \text{ufixed}_{2(\ell, f)}(B_m B_{m-1} \dots B_2 B_1 B_0)$$

where we imagined dividing the raw, uninterpreted binary word $B_m B_{m-1} \dots B_2 B_1 B_0$ stored in memory into two fields. The first of these fields had ℓ -bits and stored the leading part of the number to the left of the binary point. The second field had f -bits and stored the fractional part of the number to the right of the binary point. These fields together formed the $m + 1 = \ell + f$ bit finite binary expansion. The corresponding finite binary expansion is

$$x = b_i b_{i-1} \dots b_2 b_1 b_0 . b_{-1} b_{-2} \dots b_{-f}$$

where $b_k \in \{0, 1\}$ for all $k \in \{i, i - 1, \dots, 2, 1, 0, -1, -2, \dots, -f\}$. This is an $(m + 1)$ -bit binary representation of a rational number, where $m + 1 = \ell + f$ and $\ell = i + 1$. In this case we are told that $\ell = f = 16 = m + 1$. Then, we can use this information to interpret our raw binary string

$$\begin{aligned} x &= \text{ufixed}_{2(8,8)}(1100\ 0000\ 1101\ 1110) \\ &= b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0 . b_{-1} b_{-2} b_{-3} b_{-4} b_{-5} b_{-6} b_{-7} b_{-8} \end{aligned}$$

This interpretation gives rise to a value box that we can use to interpret this binary string as an fixed point number as seen below

| | | | | | | | | | | | | | | | | |
|------------------|-------|-------|-------|-------|-------|-------|-------|-------|---------------|---------------|---------------|----------------|----------------|----------------|-----------------|-----------------|
| Weight (power) | 2^7 | 2^6 | 2^5 | 2^4 | 2^3 | 2^2 | 2^1 | 2^0 | 2^{-1} | 2^{-2} | 2^{-3} | 2^{-4} | 2^{-5} | 2^{-6} | 2^{-7} | 2^{-8} |
| Weight (decimal) | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 | $\frac{1}{2}$ | $\frac{1}{4}$ | $\frac{1}{8}$ | $\frac{1}{16}$ | $\frac{1}{32}$ | $\frac{1}{64}$ | $\frac{1}{128}$ | $\frac{1}{256}$ |
| Bit value | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |

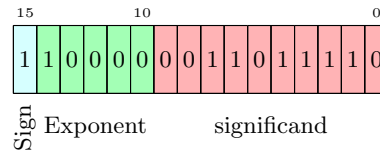
Then, using this value box, we see that our value of our original hexadecimal string via the unsigned fixed-point format is

$$\begin{aligned} (\text{c 0 d e})_{16} &= (1100\ 0000\ 1101\ 1110)_2 \\ &= 2^7 + 2^6 + 2^{-1} + 2^{-2} + 2^{-4} + 2^{-5} + 2^{-6} + 2^{-7} \\ &= 128 + 64 + \frac{1}{2} + \frac{1}{4} + \frac{1}{16} + \frac{1}{32} + \frac{1}{64} + \frac{1}{128} \\ &= \boxed{(192.8671875)_{10}} \end{aligned}$$

- (1E) a half-precision `binary16` floating point number with 1 bit for the sign, 5 bits for the biased exponent (with bias $K = 2^{5-1} - 1$), and 10 bits for the significand?

Solution: Let's now interpret our raw bit string as a half-precision floating-point number x stored using the `binary16` data class. In this case, let's visualize the bit strings that are stored in memory to represent the values of each of these variables.

Bit string stored for variable x



As discussed in the problem statement, we recall that for each variable stored in the `binary16` data class, we assign 1 bit for the sign, 5 bits for the biased exponent (with bias $K = 2^{5-1} - 1$), and 10 bits for the significand. We can convert x into a normalized binary number in scientific notation in the form

$$x = \pm (1.b_{-1} b_{-2} b_{-3} b_{-4} b_{-5} b_{-6} b_{-7} b_{-8} b_{-9} b_{-10}) \times 2^e.$$

To this end, let's first find the exponent value e . We recall the the exponent value is calculated as

$$e = u - k = 16 - 15 = 1, \quad \text{with} \quad u = \text{uint5}(10000) = 16$$

Moreover, we know the value of the significand is given in our bit string. Thus, we see

$$\begin{aligned} x &= -(1.0011\ 0111\ 10) \times 2^1 \\ &= -10.0110\ 1111\ 0 \\ &= -1 \times (2 + 2^{-2} + 2^{-3} + 2^{-5} + 2^{-6} + 2^{-7} + 2^{-8}) \\ &= -1 \times \left(2 + \frac{1}{4} + \frac{1}{8} + \frac{1}{32} + \frac{1}{64} + \frac{1}{128} + \frac{1}{256} \right) \\ &= \boxed{(-2.43359375)_{10}} \end{aligned}$$

2. Consider the following snippet of code:

```
1 x = single(-219.7);
2 y = +219.7;
3 format hex
4 x, y
```

Using pen-and-paper analysis, predict the output that MATLAB will produce in the Command Window after we execute this code. Explain your prediction by specifically highlighting the relevant features of the `single` and `double` data classes. Verify your results by actually running this code in MATLAB.

Solution: Let's begin this problem by find the exact, infinite repeating binary expansion of our number

$$x = 219.7 = 219 + 0.7$$

We start by translating the leading part into a binary integer with

$$\begin{aligned}(219)_{10} &= 128 + 64 + 16 + 8 + 2 + 1 \\ &= 2^7 + 2^6 + 2^4 + 2^3 + 2^1 + 2^0 \\ &= (11011011)_2\end{aligned}$$

To find the binary representation of our fractional part, we use the algorithm we studied in class. To this end, consider the following sequence of steps:

$$\text{Step 1:} \quad 2 \cdot 0.7 = 1.4 = 0.4 + 1$$

$$\text{Step 2:} \quad 2 \cdot 0.4 = 0.8 = 0.8 + 0$$

$$\text{Step 3:} \quad 2 \cdot 0.8 = 1.6 = 0.6 + 1$$

$$\text{Step 4:} \quad 2 \cdot 0.6 = 1.2 = 0.2 + 1$$

$$\text{Step 5:} \quad 2 \cdot 0.2 = 0.4 = 0.4 + 0$$

At the end of step 5, we can repeat steps 2 - 5 at infinitum. Using this conversion, we see

$$(0.7)_{10} = (0.101100110011\overline{0011})_2$$

We confirm this is our value using the geometric series test from Math 1C. Below is that calculation:

$$\begin{aligned}0.1011\overline{0011} &= \frac{1}{2} + \frac{1}{2^3} + \frac{1}{2^4} + \frac{1}{2^7} + \frac{1}{2^8} + \frac{1}{2^{11}} + \frac{1}{2^{12}} + \dots \\ &= \frac{1}{2} + \left(\frac{1}{2^3} + \frac{1}{2^7} + \frac{1}{2^{11}} + \dots \right) + \left(\frac{1}{2^4} + \frac{1}{2^8} + \frac{1}{2^{12}} + \dots \right) \\ &= \frac{1}{2} + \sum_{k=0}^{\infty} \frac{1}{2^{4k+3}} + \sum_{k=0}^{\infty} \frac{1}{2^{4k+4}}\end{aligned}$$

Solution: We recall that the geometric series states the following:

Geometric Series Test

Let $a \neq 0$ and let r be a real number with $|r| < 1$. Then, we conclude that the corresponding geometric series defined by the sequence of terms $a_k = a \times r^k$ for $k \in \mathbb{N}_0$ converges with

$$\sum_{k=0}^{\infty} a r^k = \frac{a}{1-r}.$$

We focus on each infinite sums in our work above separately. We start by recognizing that:

$$\sum_{k=0}^{\infty} \frac{1}{2^{4k+3}} = \sum_{k=0}^{\infty} \frac{1}{2^3} \cdot \frac{1}{2^{4k}} = \sum_{k=0}^{\infty} \frac{1}{8} \cdot \left[\frac{1}{2^4} \right]^k = \frac{1/8}{1 - \frac{1}{16}} = \frac{1}{8} \div \frac{15}{16} = \frac{1}{8} \cdot \frac{16}{15} = \frac{2}{15}$$

$$\sum_{k=0}^{\infty} \frac{1}{2^{4k+4}} = \sum_{k=0}^{\infty} \frac{1}{2^4} \cdot \frac{1}{2^{4k}} = \sum_{k=0}^{\infty} \frac{1}{16} \cdot \left[\frac{1}{2^4} \right]^k = \frac{1/16}{1 - \frac{1}{16}} = \frac{1}{16} \div \frac{15}{16} = \frac{1}{16} \cdot \frac{16}{15} = \frac{1}{15}$$

Then, our original infinite binary expansion of the fractional part of our number is given as:

$$(0.7)_{10} = (0.1011\ 0011\ 0011\ \overline{0011})_2 = \frac{1}{2} + \frac{2}{15} + \frac{1}{15} = \frac{15}{30} + \frac{4}{30} + \frac{2}{30} = \frac{21}{30} = \frac{7}{10}.$$

Using this work, we see that

$$\begin{aligned} 219.7 &= 1101\ 1011 \cdot 1011\ 0011\ 0011\ \overline{0011} \\ &= 1101\ 1011 \cdot 1011\ 0011\ 0011\ \overline{0011} \times 2^{-7} \times 2^7 \\ &= (1.101\ 1011\ 1011\ 0011\ 0011\ \overline{0011}) \times 2^7 \end{aligned}$$

In this case, we see that our desired exponent value $e = 7$. We also know that

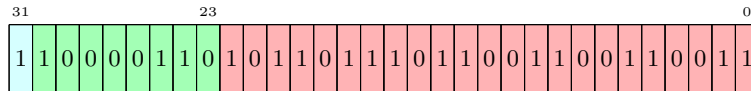
$$e = u - k$$

where the value of the biased $k = 2^{\beta-1} - 1$ depends on the number of bits β dedicated to the exponent field in our floating point encoding scheme.

Solution: Let's begin our analysis by analyzing the `single` data class encoding where $k = 2^7 - 1 = 127$. With this in mind, we see

$$u = e + k = 7 + 127 = 134 = 2^7 + 2^2 + 2^1 = 1000\ 0110$$

Then, we can store our bit string in the single format and specify every bit as seen below:



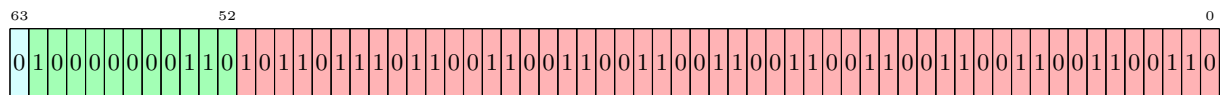
We can partition this string into 4-bit nibbles and translate the raw bit string into hexadecimal yielding

$$1100\ 0011\ 0101\ 1011\ 1011\ 0011\ 0011\ 0011 = \boxed{\text{c35bb333}}$$

Let's continue our analysis by analyzing the `double` data class encoding where $k = 2^{10} - 1 = 1023$. With this in mind, we see

$$u = e + k = 7 + 1023 = 1030 = 2^{10} + 2^2 + 2^1 = 100\ 0000\ 0110$$

Then, we can store our bit string in the single format and specify every bit as seen below:

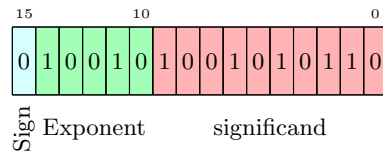


We can partition this string into 4-bit nibbles and translate the raw bit string into hexadecimal yielding

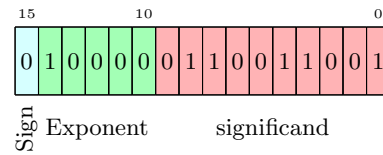
$$0100\ 0000\ 0110\ 1011\ 0111\ 0110\ 0110\ 0110\ 0110\ 0110\ 0110\ 0110\ 0110\ 0110\ 0110\ 0110 = \boxed{406b766666666666}$$

3. Suppose we have two variables, x and y , stored in memory. Suppose also that these variables are stored as `binary16` floating point numbers. Below are the bit strings that are stored in memory to represent the values of each of these variables.

Bit string stored for variable x



Bit string stored for variable y



Recall that for each variable stored in the `binary16` data class, we assign 1 bit for the sign, 5 bits for the biased exponent (with bias $K = 2^{5-1} - 1$), and 10 bits for the significand. Using this information, please solve the problems below.

- (3A) Convert x and y into normalized binary numbers in scientific notation in the form

$$\pm(1.b_{-1}b_{-2}\dots b_{-10}) \times 2^e.$$

Solution: Let's translate our encoded bit strings into their corresponding normalized binary values. Let's start with the variable x .

$$\begin{aligned} x &= +(1.1001010110) \times 2^{e_x} & \text{where} & & e_x &= u_x - k \text{ and } k = 2^4 - 1 = 15 \\ &= +(1.1001010110) \times 2^3 & \text{where} & & e_x &= 18 - 15 \\ &= 12.671875. \end{aligned}$$

Next, we continue with the other variable:

$$\begin{aligned} y &= +(1.0110011001) \times 2^{e_y} & \text{where} & & e_y &= u_y - k \text{ and } k = 2^4 - 1 = 15 \\ &= +(1.0110011001) \times 2^1 & \text{where} & & e_y &= 16 - 15 \\ &= 2.798828125. \end{aligned}$$

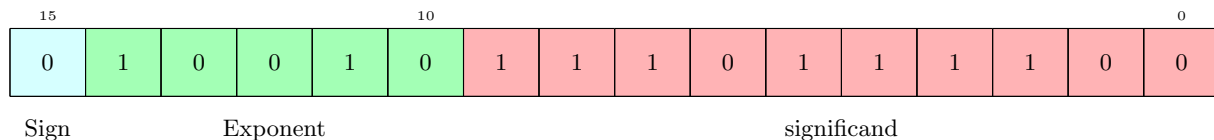
(3B) Add x and y as mathematical objects and compute the exact value of the sum.

Solution: Now, let's add these two numbers together using exact arithmetic:

$$\begin{aligned} x + y &= (1.1001010110) \times 2^3 + (1.0110011001) \times 2^1 \\ &= 1100.1010110000 + 0010.1100110010 \\ &= 1111.0111100010 \\ &= \boxed{+ (1.1110111100010) \times 2^3 = 15.470703125} . \end{aligned}$$

(3C) Typecast the sum you found in Part (3B) above as a `binary16` variable. If necessary, you can assume that we use the “chop” rule for rounding by literally chopping off the extra bits. Write the corresponding bit string for this variable in the boxes below. Be sure to explain how you made your decisions and any other information you feel might communicate the strength of your understanding of these ideas.

Solution: Let's take a look at the the raw, uninterpreted bit string for the sum of our two variable. Notice that in our case, we loose some information after the sum has been executed.



We can actually compute the stored value of the sum versus the exact value of our sum:

$$z = \underbrace{x + y}_{\text{exact sum}} = (1.1110111100010) \times 2^3 = 15.470703125$$

$$\hat{z} = \underbrace{x \oplus y}_{\text{binary16 sum}} = (1.1110111100000) \times 2^3 = 15.46875$$

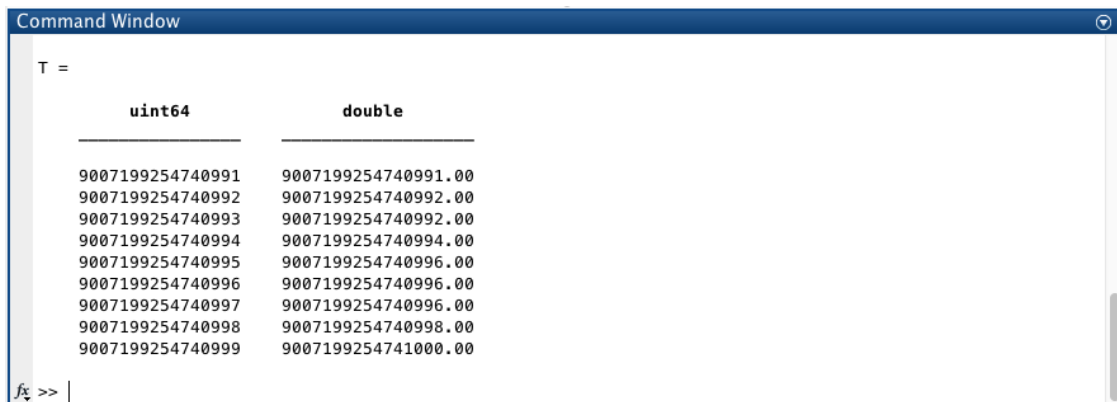
Then, we can compare these two values by analyzing the errors between these values to find:

$$\begin{aligned} z - \hat{z} &= (1.1110111100010) \times 2^3 - (1.1110111100000) \times 2^3 = 15.470703125 - 15.46875 \\ &= (0.0000000000010) \times 2^3 = 0.001953125 \\ &= \boxed{0.0000000010 = \frac{1}{2^9}} \end{aligned}$$

4. Consider the following lines of code:

```
1 format bank
2 min = 2^53-1;
3 u(1,1) = uint64(min);
4 x(1,1) = min;
5 for i = 1:1:8
6     u(i+1,1) = uint64(min) + uint64(i);
7     x(i+1, 1) = min + i;
8 end %for (i)
9 T = table(u, x);
10 T.Properties.VariableNames = {'uint64', 'double'};
11 T
```

If we run this code in MATLAB, we get the following output in the command window:



| uint64 | double |
|------------------|---------------------|
| 9007199254740991 | 9007199254740991.00 |
| 9007199254740992 | 9007199254740992.00 |
| 9007199254740993 | 9007199254740992.00 |
| 9007199254740994 | 9007199254740994.00 |
| 9007199254740995 | 9007199254740996.00 |
| 9007199254740996 | 9007199254740996.00 |
| 9007199254740997 | 9007199254740996.00 |
| 9007199254740998 | 9007199254740998.00 |
| 9007199254740999 | 9007199254741000.00 |

(4A) Notice that u and x are column vectors with size 9×1 . Compare and contrast the entries of u with the entries of x . Identify which of individual coefficient(s) $x(i, 1)$ are the SAME AS the corresponding coefficient(s) $u(i, 1)$. Please expand each of these values in terms of powers of 2. (Hint: there should be 5 entries that do not have any encoding errors).

Solution: We notice that each of the following entries are exactly equal to each other:

$$x(1, 1) = u(1, 1) = 9007199254740991 = 2^{53} - 1$$

$$x(2, 1) = u(2, 1) = 9007199254740992 = 2^{53}$$

$$x(4, 1) = u(4, 1) = 9007199254740994 = 2^{53} + 2^1$$

$$x(6, 1) = u(6, 1) = 9007199254740996 = 2^{53} + 2^2$$

$$x(8, 1) = u(8, 1) = 9007199254740998 = 2^{53} + 2^2 + 2^1$$

(4B) Now identify which individual coefficient(s) $x(i, 1)$ are DIFFERENT FROM the corresponding coefficient(s) $u(i, 1)$. Please expand each of these $x(i, 1)$ AND $u(i, 1)$ values in terms of powers of 2. individual entries of x contain errors as a result of the calculation $x(i+1, 1) = \min + i$. For each of these entries, write the difference between the values of $x(i, 1)$ AND $u(i, 1)$.

Solution: The following entries of the vectors x and u are not equal to each other.

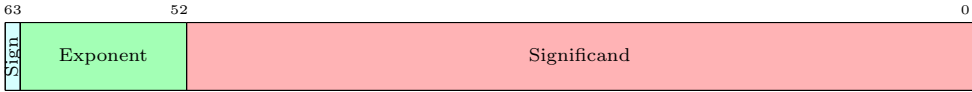
$$\begin{aligned} x(3, 1) &= 9007199254740992 = 2^{53} & u(3, 1) &= 9007199254740993 = 2^{53} + 2^0 \\ x(5, 1) &= 9007199254740996 + 2^2 & u(5, 1) &= 9007199254740995 = 2^{53} + 2^1 + 2^0 \\ x(7, 1) &= 9007199254740996 + 2^2 & u(7, 1) &= 9007199254740997 = 2^{53} + 2^2 + 2^0 \\ x(9, 1) &= 9007199254741000 + 2^3 & u(9, 1) &= 9007199254740999 = 2^{53} + 2^2 + 2^1 + 2^0 \end{aligned}$$

We can analyze the individual differences between each of these entries. Doing so, we find that for each $i = 3, 5, 7, 9$, the absolute error that occurs in our data encoding is identical and given as

$$|x(i, 1) - u(i, 1)| = 2^0$$

Due to the IEEE 754 rounding implementation, every other number is accurate when we attempt to add values of 1 to each number beyond 2^{53} . In Part (4C) below, we discuss more details about where these errors come from.

(4C) Remember that, by default, MATLAB stores variables using the `double` or `binary64` data class. The structure of this data class is specified in the IEEE 754 Standard and assigns 1 bit for the sign, 11 bits for the biased exponent (with bias $K = 2^{11-1} - 1$), and 52 bits for the significand. Visually, we can represent the bits stored in memory using the following diagram:



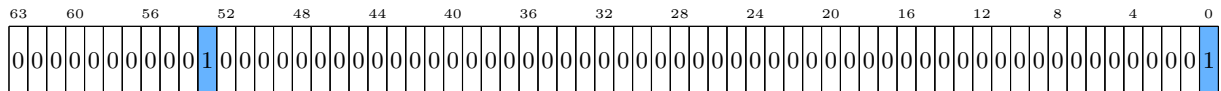
Using this information, please identify the source of the errors in the individual entries of vector `x`. Explain the size of the errors and relate this back to your understanding of the structure(s) of the `double` data class.

Solution: Let's take a look at what is happening inside the computer as we store these integers in memory. Specifically, we will compare and contrast the `uint64` data class with the `double` data class. In doing so, we hope to understand the source of these errors. Let's begin by analyzing the `uint64` entries of the vector `u` by showing the raw, uninterpreted bit strings in the individual entries of this vector, as seen below.

Solution: In the previous two cases, we see exact encodings of the desired values in the `double` data class. But, in the third entry of `x` we see our first encoding error. Let's analyze this in detail. We notice that

$$\begin{aligned} u(3,1) &= (2^{53} + 2^0)_{10} \\ &= (10\,0000\,0000\,0000\,0000\,0000\,0000\,0000\,0000\,0000\,0000\,0001)_2. \end{aligned}$$

Once again, the `uint64` data class stores the individual entries of `u` as a pure unsigned binary integer encoding. Thus, MATLAB stores the following bits in the memory location for `u(2,1)`:



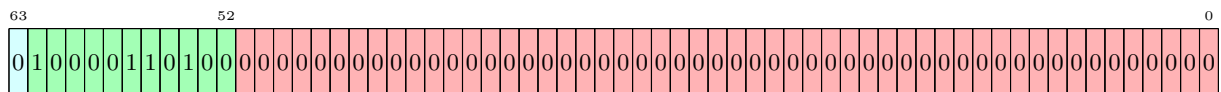
When incrementing the `double` encoding with this arithmetic, we see

$$\begin{aligned} x(3,1) &= 2^{53} + 2^0 \\ &= (10\,0000\,0000\,0000\,0000\,0000\,0000\,0000\,0000\,0000\,0000\,0001)_2. \\ &= (1.0000\,0000\,0000\,0000\,0000\,0000\,0000\,0000\,0000\,0000\,0000\,0000\,0000\,0000\,0000\,0001)_2 \cdot 2^{-53} \times 2^{53} \\ &= (1.0000\,0000\,0000\,0000\,0000\,0000\,0000\,0000\,0000\,0000\,0000\,0000\,0000\,0000\,0001)_2 \times 2^{53} \end{aligned}$$

We know from the last page that our exponent code in this case is given by

$$u = 53 + 1023 = 1075 = 1024 + 32 + 16 + 4 = 100\,0011\,0100$$

We now see the source of our error. The `double` data class stores only 52 fractional bits in the normalized significand. But, our exact value for `x(3,1)` includes 53 bits in our fractional part. So, in order to comply with the specifications of the IEEE 754 standard, MATLAB rounds the value `x(3,1)` downward, resulting in the following bit code:



5. Create a figure with four subplots. Details of the plots are labeled below.

(5A) For the first subplot, define the following variables:

$$x1 = [-5:5]; y1 = \sin(x1);$$

Then, plot $x1$ versus $y1$ using a green dotted line with \times markers. Make sure the legend is present. Also label the horizontal axis using the string “x-axis” and label the vertical axis using the string “y-axis.” Do not label the entire plot with a title for the first subplot.

(5B) For the second subplot, define the following variables:

$$x2 = [1:2:49]; y2 = x2 + x2^{0.5};$$

Then, plot $x2$ versus $y2$ using a red dash-dot line with triangle markers. This time, make sure that NO legend is present. Label the entire plot using the title “Is this a line?” Also, label the horizontal axis a “x” and the vertical axis as “y.”

(5C) For the second subplot, define the following variables:

$$x3 = [1:0.0001:2]; y3 = \exp(x3);$$

Then, plot $x3$ versus $y3$ using a magenta dashed line with pentagram markers. Make sure that NO legend is present and label the entire plot using the title “The study of sharp increase in the exponential function over an interval.” Also, label the horizontal axis with the string “the independent variable” and the vertical axis as “the dependent variable.”

(5D) For the final subplot, define the following variables:

$$x4 = [0:0.01:2*\pi]; y4 = \cos(x4);$$

Then, plot $x4$ versus $y4$ using a blue solid line with no markers. Create a legend and label the entire plot using the title “A study of the shape of the Cosine curve.” Label the horizontal axis with the string “the horizontal axis” and the vertical axis as “the vertical axis.”

-
6. (For fun! Only touch this after you finish all other problems) In this class, we've studied a few different algorithms for converting finite decimal expansions into binary expansions. Let's return to one of these algorithms by studying the number

$$0.390625 = \frac{25}{64} = \frac{16 + 8 + 1}{64}$$

We can find the equivalent binary representation of this number with the following steps

$$\begin{array}{lll} 2 \cdot 0.390625 = 0.78125 & \implies & 0.78125 = 0.78125 + 0 \\ 2 \cdot 0.78125 = 1.5625 & \implies & 1.5625 = 0.5625 + 1 \\ 2 \cdot 0.5625 = 1.125 & \implies & 1.125 = 0.125 + 1 \\ 2 \cdot 0.125 = 0.25 & \implies & 0.25 = 0.25 + 0 \\ 2 \cdot 0.25 = 0.5 & \implies & 0.5 = 0.5 + 0 \\ 2 \cdot 0.5 = 1.0 & \implies & 1.0 = 0.0 + 1 \end{array}$$

Then, we use this to conclude that $(0.390625)_{10} = (0.011001)_2$. With this in mind, let $n \in \mathbb{N}$ and suppose we have an n -digit, finite decimal expansion given by

$$x = 0.d_1d_2\dots d_n = \frac{d_1d_2\dots d_n}{10^n}$$

where $0 < x < 1$. Prove that the algorithm described above will produce the binary representation of x .