EX.0.3.7, Sauer

Write each of the given numbers in Matlab's *format hex*. Show your work. Then check your answers with Matlab. (a) 8 (b) 21 (c) 1/8 (d) fl(1/3) (e) fl(2/3) (f) fl(0.1) (g) fl(−0.1) (h) fl(−0.2)

EX.0.3.7, Sauer, solution, Langou

- Only turning the Python code is not a good answer.
- The copy-paste from this PDF to python code does not work great. It is better to copy-paste from colab.
- The Colab Jupyter Notebook is available at: `https://colab.research.google.com/drive/1Pv58EsXEaVojnQqBEMLjxlWPP-tFwLzL`.
- Do not forget to execute the piece of code below first.

```
import struct
def double_to_hex(f):
  return hex(struct.unpack('<Q', struct.pack('<d', f))[0])
```

a.

$$8 = (1000)_2 = +2^3(1.0000000000000000000000000000000000000000000000000000)_2$$

The sign is positive, so the bit sign is 0. The exponent is 3. Adding 1023 to the exponent gives $1026 = 1024 + 2$, or $(10000000010)_2$. We get

0	10000000010	0000000000000000000000000000000000000000000000000000

Grouping by four:

0100	0000	0010	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
4	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0

The 64 bits (given as 16 hexadecimal numbers) representing 8 on the computer are

$$\boxed{\texttt{4020000000000000}}$$

```
print( double_to_hex( 8.) )
```

```
0x4020000000000000
```

b.

$$21 = 16 + 4 + 1 = (10101)_2 = +2^4(1.0101000000000000000000000000000000000000000000000000)_2$$

The sign is positive, so the bit sign is 0. The exponent is 4. Adding 1023 to the exponent gives $1027 = 2^{10} + 2 + 1$, or $(10000000011)_2$. We get

0	10000000011	0101000000000000000000000000000000000000000000000000

Grouping by four:

0100	0000	0011	0101	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
4	0	3	5	0	0	0	0	0	0	0	0	0	0	0	0

The 64 bits (given as 16 hexadecimal numbers) representing 21 on the computer are

$$\boxed{\texttt{4035000000000000}}$$

```
print( double_to_hex( 21.) )
```

```
0x4035000000000000
```

c.

$$1/8 = 2^{-3} = +2^{-3}(1.0000000000000000000000000000000000000000000000000000)_2$$

The sign is positive, so the bit sign is 0. The exponent is -3. Adding 1023 to the exponent gives $1020 = 512 + 256 + 128 + 64 + 32 + 16 + 8 + 4 = (01111111100)_2$. We get

0	01111111100	0000000000000000000000000000000000000000000000000000

Grouping by four:

0011	1111	1100	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
3	f	c	0	0	0	0	0	0	0	0	0	0	0	0	0

The 64 bits (given as 16 hexadecimal numbers) representing 1/8 on the computer are

$$\boxed{\texttt{3fc0000000000000}}$$

```
print( double_to_hex( 1./8. ) )
```

```
0x3fc0000000000000
```

d. We first need to convert 1/3 in base 2:

$$\begin{array}{ccccc}
\frac{1}{3} & * & 2 & = & 0\frac{2}{3} \rightarrow 0 \\
\frac{2}{3} & * & 2 & = & 1\frac{1}{3} \rightarrow 1 \\
\hline
\frac{1}{3} & * & 2 & = & 0\frac{2}{3} \rightarrow 0
\end{array}$$

We find:

$$x = (\frac{1}{3})_{10} = (0.\overline{01})_2$$

In scientific notation:

$$x = +2^{-2}(1.\overline{01})_2$$

x is not a machine number. It cannot be stored exactly on the computer. We need to find the two machine numbers, x_- and x_+, around it.

We have

$$
\begin{aligned}
x_- &= +2^{-2}(1.0101010101010101010101010101010101010101010101010101 \qquad)_2 \\
x = 1/3 &= +2^{-2}(1.0101010101010101010101010101010101010101010101010101|0101 \quad)_2 \\
x_+ &= +2^{-2}(1.0101010101010101010101010101010101010101010101010110 \qquad)_2
\end{aligned}
$$

We see that x_- is the closest. So we have $\mathrm{fl}(1/3) = x_-$. So now we need to find the 64 bits representing x_- on the computer.

The sign is positive, so the bit sign is 0. The exponent is -2. Adding 1023 to the exponent gives $1021 = 512 + 256 + 128 + 64 + 32 + 16 + 8 + 4 + 1 = (01111111101)_2$. We get

0	01111111101	0101010101010101010101010101010101010101010101010101

Grouping by four:

0011	1111	1101	0101	0101	0101	0101	0101	0101	0101	0101	0101	0101	0101	0101	0101
3	f	d	5	5	5	5	5	5	5	5	5	5	5	5	5

The 64 bits (given as 16 hexadecimal numbers) representing 1/3 on the computer are

$$\boxed{\texttt{3fd5555555555555}}$$

```
print( double_to_hex( 1./3. ) )
```

```
0x3fd5555555555555
```

e. We observe that $2/3 = 2 \times 1/3$. (What an observation!!!) So 2/3 is simply 1/3 with an exponent of +1. So by looking at question (d), we infer:

$$\boxed{\texttt{3fe5555555555555}}$$

Let us go slower and redo everything. But please note that we essentially copy-paste question (d).

We first need to convert 2/3 in base 2:

$$
\begin{array}{ccccc}
\frac{2}{3} & * & 2 & = & 1\frac{1}{3} & \to 1 \\[1ex]
\frac{1}{3} & * & 2 & = & 0\frac{2}{3} & \to 0 \\[1ex]
\hline
\frac{2}{3} & * & 2 & = & 1\frac{1}{3} & \to 1
\end{array}
$$

We find:

$$x = (\frac{2}{3})_{10} = (0.\overline{10})_2$$

In scientific notation:

$$x = +2^{-1}(1.\overline{01})_2$$

x is not a machine number. It cannot be stored exactly on the computer. We need to find the two machine numbers, x_- and x_+, around it.

We have

$$
\begin{aligned}
x_- &= +2^{-1}(1.0101010101010101010101010101010101010101010101010101 \quad)_2 \\
x = 2/3 &= +2^{-1}(1.0101010101010101010101010101010101010101010101010101|0101 \quad)_2 \\
x_+ &= +2^{-1}(1.0101010101010101010101010101010101010101010101010110 \quad)_2
\end{aligned}
$$

We see that x_- is the closest. So we have $\mathrm{fl}(2/3) = x_-$. So now we need to find the 64 bits representing x_- on the computer.

The sign is positive, so the bit sign is 0. The exponent is -1. Adding 1023 to the exponent gives $1022 = 512 + 256 + 128 + 64 + 32 + 16 + 8 + 4 + 1 = (01111111110)_2$. We get

0	01111111110	0101010101010101010101010101010101010101010101010101

Grouping by four:

0011	1111	1110	0101	0101	0101	0101	0101	0101	0101	0101	0101	0101	0101	0101	0101
3	f	e	5	5	5	5	5	5	5	5	5	5	5	5	5

The 64 bits (given as 16 hexadecimal numbers) representing 2/3 on the computer are

$$\boxed{\texttt{3fe5555555555555}}$$

```
print( double_to_hex( 2./3. ) )
```

```
0x3fe5555555555555
```

f. We first need to convert 0.1 in base 2:

$$
\begin{array}{ccccc}
0.1 & * & 2 & = & 0.2 \quad \rightarrow 0 \\
0.2 & * & 2 & = & 0.4 \quad \rightarrow 0 \\
0.4 & * & 2 & = & 0.8 \quad \rightarrow 0 \\
0.8 & * & 2 & = & 1.6 \quad \rightarrow 1 \\
0.6 & * & 2 & = & 1.2 \quad \rightarrow 1 \\
\hline
0.2 & * & 2 & = & 0.4 \quad \rightarrow 0
\end{array}
$$

We find:

$$x = (0.1)_{10} = (0.0\overline{0011})_2$$

In scientific notation:

$$x = +2^{-4}(1.\overline{1001})_2$$

x is not a machine number. It cannot be stored exactly on the computer. We need to find the two machine numbers, x_- and x_+, around it.

We have

$$
\begin{aligned}
x_- &= +2^{-4}(1.1001100110011001100110011001100110011001100110011001 \qquad)_2 \\
x = 0.1 &= +2^{-4}(1.1001100110011001100110011001100110011001100110011001|1001 \quad)_2 \\
x_+ &= +2^{-4}(1.1001100110011001100110011001100110011001100110011010 \qquad)_2
\end{aligned}
$$

We see that x_+ is the closest. So we have $\mathrm{fl}(0.1) = x_+$. So now we need to find the 64 bits representing x_+ on the computer.

The sign is positive, so the bit sign is 0. The exponent is -4. Adding 1023 to the exponent gives $1019 = 512 + 256 + 128 + 64 + 32 + 16 + 8 + 2 + 1 = (01111111011)_2$. We get

0	01111111011	1001100110011001100110011001100110011001100110011010

Grouping by four:

0011	1111	1011	1001	1001	1001	1001	1001	1001	1001	1001	1001	1001	1001	1001	1010
3	f	b	9	9	9	9	9	9	9	9	9	9	9	9	a

The 64 bits (given as 16 hexadecimal numbers) representing 0.1 on the computer are

`3fb999999999999a`

```
print( double_to_hex( 0.1 ) )
```

```
0x3fb999999999999a
```

g. So -0.1 is simply 0.1 with a minus sign. So looking at question (f), we change the first bit from a 0 to 1. So the first four bits are 1011 (instead of 0011), and so the first hexadecimal letter is b.

`bfb999999999999a`

```
print( double_to_hex( -0.1 ) )
```

```
0xbfb999999999999a
```

h. We observe that $-0.2 = 2 \times (-0.1)$. (What an observation!!!) So -0.2 is simply (-0.1) with an exponent of $+1$. So by looking at question (g), we infer:

`bfc999999999999a`

```
print( double_to_hex( -0.2 ) )
```

```
0xbfc999999999999a
```