Question 1 True or false? Justify.

(a) $17 \bmod 5 = 2$ (b) $5 \bmod 17 = 2$ (c) $500 \bmod 17 = 2$

(d) $17 \equiv 5 \pmod 2$ (e) $5 \equiv 17 \pmod 2$ (f) $500 \equiv 2 \pmod{17}$

(a) $\boxed{\textit{True}}$ $17 = 3{\times}5 + \mathbf{2}$.

(b) $\boxed{\textit{False}}$ $5 = 0{\times}17 + \mathbf{5}$, *so* $5 \bmod 17 = 5$

(c) $\boxed{\textit{False}}$ $500 = 29{\times}17 + \mathbf{7}$, *so* $500 \bmod 17 = 7$.

(d) $\boxed{\textit{True}}$ $17 - 5 = 12 = 6{\times}2$ *is a multiple of* 2.

(e) $\boxed{\textit{True}}$ $5 - 17 = -12 = -6{\times}2$ *is a multiple of* 2.

(f) $\boxed{\textit{False}}$ $500 - 2 = 498 = 29{\times}17 + 5$ *is not a multiple of 17.*

Question 2

(a) Without a calculator, complete the following table. Try to be as efficient as possible with your calculations.

x	$7^x \mod 11$
1	7
2	5
3	
4	
5	
6	
7	
8	
9	
10	

(b) Do you notice any patterns in the table? Discuss any patterns you see with your classmates.

(a) The final table is:

x	$7^x \mod 11$
1	*7*
2	*5*
3	*2*
4	*3*
5	*10*
6	*4*
7	*6*
8	*9*
9	*8*
10	*1*

To find the entry in row i, we multiply the entry in row $i-1$ by 7, and then reduce mod 11. For example, to find the entry in row 3 we take $5 \times 7 = 35$, and then reduce mod 11 to get 2.

(b) Answers may vary. The absence of patterns, at least for well-chosen and large replacements for 7 and 11, is important for internet security.

Question 3

Prove the following statement: An integer is a multiple of 3 if and only if the sum of its digits is a multiple of 3.

Let z be an integer. We consider cases.

Consider first the case that $z \geq 0$: Let

$$d_0, d_1, \ldots, d_{n-1} \in \{0, 1, \ldots, 9\}$$

be such that $d_{n-1} \ldots d_1 d_0$ is the decimal representation of z. Then

$$\begin{aligned}
& z \text{ is a multiple of } 3 \\
& z \equiv 0 \pmod 3 \\
\Leftrightarrow & d_{n-1} \times 10^{n-1} + \ldots d_1 \times 10^1 + d_0 \times 10^0 \equiv 0 \pmod 3 \\
\Leftrightarrow & d_{n-1} \times 1^{n-1} + \ldots d_1 \times 1^1 + d_0 \times 1^0 \equiv 0 \pmod 3 \\
& \text{(by the modular arithmetic theorem, because } 10 \equiv 1 \pmod 3) \\
\Leftrightarrow & d_{n-1} + \ldots d_1 + d_0 \equiv 0 \pmod 3.
\end{aligned}$$

Now consider the case that $z < 0$: Let

$$d_0, d_1, \ldots, d_{n-1} \in \{0, 1, \ldots, 9\}$$

be such that $-d_{n-1} \ldots d_1 d_0$ is the decimal representation of z. Then

$$\begin{aligned}
& z \equiv 0 \pmod 3 \\
\Leftrightarrow & -1 \times (d_{n-1} \times 10^{n-1} + \ldots d_1 \times 10^1 + d_0 \times 10^0) \equiv 0 \pmod 3 \\
\Leftrightarrow & (d_{n-1} \times 10^{n-1} + \ldots d_1 \times 10^1 + d_0 \times 10^0) \equiv 0 \pmod 3 \\
& \text{(by the modular arithmetic theorem, multiplying both sides by } -1) \\
\Leftrightarrow & d_{n-1} \times 1^{n-1} + \ldots d_1 \times 1^1 + d_0 \times 1^0 \equiv 0 \pmod 3 \\
& \text{(by the modular arithmetic theorem, because } 10 \equiv 1 \pmod 3) \\
\Leftrightarrow & d_{n-1} + \ldots d_1 + d_0 \equiv 0 \pmod 3. \\
\Leftrightarrow & \text{the sum of the digits of } z \text{ is a multiple of } 3.
\end{aligned}$$

In each case the result holds. $\square$

Question 4 Read the following algorithm:

```
Algorithm for converting a fraction $x$ into binary with $p$ binary places.

Inputs: fraction $x \in \mathbb{Q}$, $0 < x < 1$ and number of places $p \in \mathbb{N}$.
Initialise: $j \leftarrow 1$, $b_1, b_2, \ldots, b_p \leftarrow 0$
Loop: if $j = p + 1$ stop.
    $x \leftarrow 2x$
    If $x \geq 1$ [ $b_j \leftarrow 1$, $x \leftarrow x - 1$ ]
    $j \leftarrow j + 1$
Outputs: bits $b_1, b_2, \ldots, b_p$ such that $(0.b_1 b_2 \ldots b_p)_2$ is the
         best approximation to $x$ using $p$ binary places.
```

(a) Verify that the algorithm converts $\frac{1}{6}$ to 0.00101_2 when $p = 5$. Can you spot how to express $\frac{1}{6}$ exactly in repeating binary?

(b) Express 0.45_{10} as accurately as possible with 8 binary places.

(a)	x	j	b_{j-1}
initially	$\frac{1}{6}$	1	—
after loop 1	$\frac{1}{3}$	2	$0 = b_1$
2	$\frac{2}{3}$	3	$0 = b_2$
3	$\frac{1}{3}$	4	$1 = b_3$
4	$\frac{2}{3}$	5	$0 = b_4$
5	$\frac{1}{3}$	6	$1 = b_5$

stop

We see from the trace table above that the situation after the 4th loop is eactly the same as after the 2nd loop. So the group of the second and third digits repeats forever, and hence

$$\boxed{\frac{1}{6} = 0.0\overline{01}}.$$

(b)	x	j	b_{j-1}
initially	.45	1	—
after loop 1	.9	2	$0 = b_1$
2	.8	3	$1 = b_2$
3	.6	4	$1 = b_3$
4	.2	5	$1 = b_4$
5	.4	6	$0 = b_5$
6	.8	7	$0 = b_6$
7	.6	8	$1 = b_7$
8	.2	9	$1 = b_8$

stop

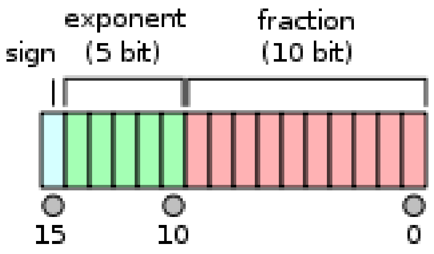
Hence to 8 binary places accuracy

$$\boxed{0.45_{10} \approx 0.01110011_2.}$$

Note that as with (a) it is easy to see the repeating pattern in the trace table and so deduce the exact repeating binary expression

$$0.45_{10} = 0.01\overline{1100}_2.$$

Question 5 The shortest IEEE standard for representing rational numbers is called *half-precision floating point*. It uses a 16-bit word partitioned as in the diagram at right. (This diagram is taken from the Wikipedia article on the subject, where more details can be found.)



As described in lectures, to store a rational number x it is first represented as $(-1)^s \times m \times 2^n$ with $1 \leq m < 2$. The sign bit s is stored as the left-most bit (bit 15), the mantissa m (called "significand" in IEEE parlance) is stored in the right-most 10 bits (bits 9 to 0), and the exponent n is stored in the 5 bits in between (bits 14 to 10). However:

- *Only the fractional part of m is stored.* Because $1 \leq m < 2$, the binary representation of m **always** has the form $1 \cdot \star\star\star \ldots$ where the stars stand for binary digits representing the fractional part of m. Hence there is no need to store the $1\cdot$ part.

- *the exponent n is stored with an "offset".* In order to allow for both positive and negative exponents, but to avoid another sign bit, the value stored is $n + 15$. In principle this means that the five exponent bits can store exponents in the range $-15 \leq n \leq 16$, but 00000 and 11111 are reserved for special purposes so in fact n is restricted to the range $-14 \leq n \leq 15$.

(a) A rational number x is stored in half-precision floating point as the word $3A6B_{16}$. (That's hex shorthand for the 16-bit binary word.) Write x in ordinary decimal notation.

(b) Find the word representing $x = -123.45_{10}$ in half-precision floating point. Use the closest approximation to x that it is possible to store in this format. Give your answer using hex shorthand, like the word you were given for (a).
Note: From a previous questions, $0.45_{10} = 0.01\overline{1100}_2$.

(a) $3\text{A}6\text{B}_{16} = \underbrace{0011}_{3}\underbrace{1010}_{\text{A}}\underbrace{0110}_{6}\underbrace{1011}_{\text{B}} = \begin{array}{c|c|l} 0 & 01110 & 1001101011 \\ s & n{+}15 & \textit{frac.part of } m \\ & = 14 & \end{array}$

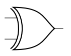
$$\begin{aligned}
\longrightarrow\ & (-1)^0 \times 1.1001101011 \times 2^{14-15} \\
&= 1 \times 1.1001101011 \times 2^{-1} \\
&= (0.11001101011)_2 \qquad \textit{(moving the binary point 1 place to the left)} \\
&= \left(\frac{1}{2} + \frac{1}{4} + \frac{1}{32} + \frac{1}{64} + \frac{1}{256} + \frac{1}{1024} + \frac{1}{2048}\right)_{10} \\
&= \frac{1024 + 512 + 64 + 32 + 8 + 2 + 1}{2048} = \frac{1643}{2048} \approx 0.802246_{10}.
\end{aligned}$$

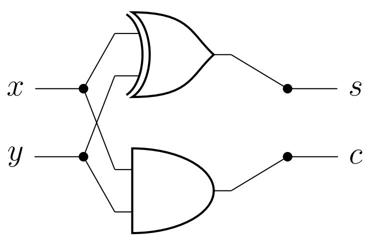
(b)

$$\begin{aligned} 123 &= -(64+32+16+8+2+1 \quad \textit{and} \quad 0.45 = 0.01\overline{1100}_2 \\ -123.45 &= -1111011.011100110011\ldots \\ &\approx -1.1110110111 \times 2^6 \quad \textit{(shifting the binary point 6 places left and truncating)} \end{aligned}$$

$$\longrightarrow \begin{array}{c|c|c} \mathtt{1} & \mathtt{10101} & \mathtt{1110110111} \\ s & 21 & \textit{frac.part} \end{array} \qquad (6+15=21=\mathtt{10101}_2)$$

$$\longrightarrow \underbrace{\mathtt{1101}}_{\text{D}}\underbrace{\mathtt{0111}}_{7}\underbrace{\mathtt{1011}}_{\text{B}}\underbrace{\mathtt{0111}}_{7} = \boxed{\text{D7B7}_{16}}$$

Question 6 In lectures we discussed a circuit for a *half adder* for adding two 1-bit numbers x and y. Using an XOR gate  to simplify that circuit, it can be drawn as shown at right. The outputs are the 'sum bit' s and the 'carry (out) bit' c, so that, in binary, $x + y = cs$.



Recall that to add multi-bit numbers we we use a cascade of *full adders.* A full adder has an additional input of the the carry (out) from the prior addition of digits to the immediate right. To avoid confusion, denote this 'carry-in' by c_{in} and rename c as c_{out} for 'carry-out'. (Of course, any carry bit can be 0, meaning 'no carry'.) So, in binary, $x + y + c_{\text{in}} = c_{\text{out}}s$. (If you need to, see the input-output table shown in lectures.)

Challenge[1] Design a digital circuit for a full adder. Use any kind of gates that you think will help .

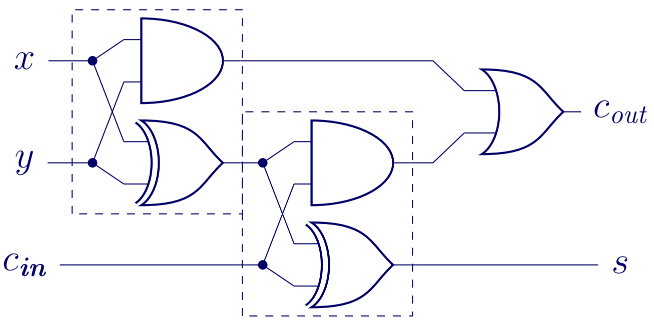
(a)

Inputs			*Outputs*	
x	y	c_{in}	c_{out}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

(b) *One method is to think of $x+y+c_{in}$ as $(x+y)+c_{in}$ and to use a half-adder for $x+y$ and another half-adder for the sum bit of $x+y$ and the c_{in} bit.*

The c_{out} will then depend on the carry out from these two half adders, but it is not immediately obvious how. Some thought reveals that it is in fact the disjuction of the two carry outs.

So a full adder can be made from two half adders and an OR gate:



The above is what you will find from an internet search. Another method of obtaining a circuit is to use disjunctive normal form for each of c_{out} and s and to build circuits for each using the standard method with NOT, AND and OR gates. However, this leads to a far more complicated circuit.

[1]This is less of a challenge if you just look it up on the internet, but try to do it yourself!