

INSTRUCTIONS

Before your workshop Revise the material from the lectures in Week 4. Make a summary and learn the vocabulary. Then make your best attempt at Problems 1, 3 and 5 on the worksheet. Write your attempts neatly, on the worksheet or a separate piece of paper, and bring them to the workshop. At this stage, the correctness of your solution is not important. What is important is that you try your best, and write what you do know so that it is evident that you tried your best. You must take your attempts at Problems 1, 3 and 5 to the workshop. You may take along any summaries or other notes you have made about the material too. You should expect to spend about two hours revising material and making your attempts at Problems 1, 3 and 5.

During your workshop At the start of the workshop, your demonstrator will ask to see your attempts at Problems 1, 3 and 5. While your demonstrator looks these over, you will work collaboratively with your classmates on problems 2, 4 and 6. Your demonstrator will guide you through the workshop, and help you if you are stuck. The demonstrator and class members may comment and correct your work during the workshop. Your goal is to leave the workshop feeling like you understand how to solve as much of the worksheet as possible. The aim is for a collaborative mathematical experience.

You may take notes during a workshop for later reference. Some of you will prefer to take photos, rather than write notes. We need some rules for this. You may take photos of work written on the board, but only after the author of the work has given their explicit permission. You may not take photos of people during the workshops.

For MATH1005 students only, workshop participation is worth 10% of your overall grade. For a maximum participation mark you will need to bring satisfactory attempts at problems 1, 3 and 5 to class (where satisfactory does not have to mean correct) and participate productively throughout the workshop. If illness or other circumstances prevent you attending one or two workshops, don't worry because we will only use the best eight out of ten workshop participation scores.

After your workshop At 8pm after your workshop your assignment document will become available on Wattle. This will contain several questions closely related to the questions on this worksheet. When you have worked out answers write up the solutions neatly and carefully in the spaces provided in the assignment document, using good English spelling and grammar and, most importantly, showing all necessary steps of calculations and reasoning (except on multiple choice questions). You have 6 days to submit your assignment (online). Your demonstrator will then grade it and you will be able to see marks and possible comments online. These assignments will contribute 10% to your overall grade.

Assignments will vary from one day to another, so you must download your own assignment, not rely on someone else who may have a workshop on a different day. Also note that late submissions will not be accepted, and the assignment document itself will disappear from Wattle after the submission deadline.

MY NAME IS:

MY u NUMBER IS:

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) $500 \equiv 2 \pmod{17}$

Question 2 Let $x = 11010_2$, $y = 10111_2$, $s = x + y$, $d = x - y$, $p = xy$.

- (a) Calculate s, d and p directly in binary. Keep the answers in binary.
 (b) Convert x, y to hexadecimal and recalculate s, d and p directly in hexadecimal.
 (c) Finally convert x, y to decimal, recalculate s, d and p (in decimal) and convert the answers to hexadecimal and thence to binary.
 Use the results to check your answers to (a) and (b).

Question 3 This question is about the toggle-plus-one method as it relates to the storage of integers in computer words. It also shows how this method avoids the need for separate subtraction circuits.

As demonstrated in lectures, for a binary word W , toggle-plus-one means:

toggle: replace every 1 by 0, every 0 by 1, then
 add one: treating W as a binary number, add 1.
 Ignore any carry beyond the length of the word.

For $l \in \mathbb{N}$ let $S_l = \{n \in \mathbb{Z} : -2^{l-1} \leq n < 2^{l-1}\}$. Then S_l is the set of all integers that can be stored in words of length l bits, using the standard computer representation. The rules governing the storage of an integer n in a word W may be summarised as:

Rule 1: n is negative if and only if the left-most bit of W is 1
 Rule 2: $-n$ is stored as the word obtained from W by toggle-plus-one.
 This is true even when n is negative.
 Rule 3: If the left-most bit of W is 0 then $n = W_2$.
i.e. n is retrieved by treating W as a binary number.

In computer arithmetic, subtraction uses negation and addition: $x - y = x + (-y)$. In the addition, any carry beyond the length of the word is ignored.

As an example, take $x = 11010_2$, $y = 10111_2$ and use **8-bit** computer arithmetic on:

- (a) $x - y$ (b) $y - x$ (c) $-x - y$

Check your results by expressing x, y and your answers in decimal.

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 $p \in \mathbb{N}$.

Outputs: bits b_1, b_2, \dots, b_p such that $(0.b_1b_2\dots b_p)_2$ is the best approximation to x using p binary places.
--

Initialise: $j \leftarrow 1$ (i.e. set j to 1), $b_1, b_2, \dots, b_p \leftarrow 0$

Loop: if $j = p + 1$ stop.

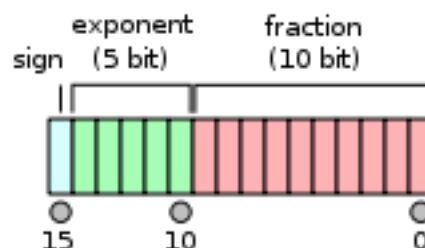
$x \leftarrow 2x$ (i.e. replace x by $2x$)

If $x \geq 1$ [$b_j \leftarrow 1$, $x \leftarrow x - 1$]

$j \leftarrow j + 1$


- (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.

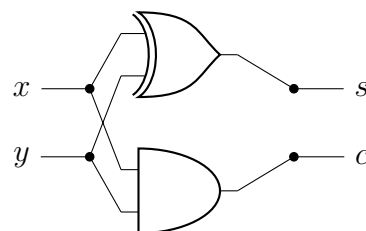
Question 5 The shortest IEEE standard for representing rational numbers is called *half-precision floating point*. It uses a 16-bit word partitioned as in the diagram at right. (This diagram is taken from the Wikipedia article on the subject, where more details can be found.)



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 $1.\star\star\star\dots$ 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 5555_{16} . Express 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 as a hexadecimal number.

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 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.)

Challenge¹ Design a digital circuit for a full adder. Use any kind of gates that you think will help .

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