**Score: \_\_\_\_\_**

**DA3 – Signed Binary Numbers**

**Activities**

COMP256 – Computing Abstractions

Dickinson College

Spring 2023

Prof. Grant Braught

**Name:**

**Introduction:**

Today’s class introduced two ways that we can represent signed (positive and negative) whole numbers in binary. The sign magnitude representation uses the most significant bit (MSb) to represent the sign of the number. The two’s complement representation uses a process of flipping the bits and adding 1 to change a number to its complement (i.e. 7 to -7 or -7 to 7). Today’s activities will give you practice with those representations and ask you to explore some of their uses and limits. What you learn, will give you enough information to explain the “strange” infinite loop example that we saw on the first day of class.

**Sign Magnitude Representation:**

🔑 1. Give the base 10 value of the following 8-bit sign magnitude binary values. Write your answer so as to be convincing that you understand the process. Using steps similar to those shown in class is a good way to do this. Correct answers without sufficient work will receive ✔︎ or ✔︎- scores.

 a. 0100 11012SM

 b. 1011 01112SM

🔑 2. Give the 8-bit sign magnitude representation of the following base 10 values. Write your answer so as to be convincing that you understand the process. Using steps similar to those shown in class is a good way to do this. Correct answers without sufficient work will receive ✔︎ or ✔︎- scores.

 a. 8610

 b. -11810

**Two’s Complement Representation:**

It is not required viewing, but if you would like to have some additional explanation and examples of two’s complement numbers you can check out the following videos:

* The One's Complement, Two's Complement, and Signed Magnitude video from BitMerge illustrates negative binary numbers and how two’s complement works. Note: they also introduce one’s complement (which we did not see) as a stepping stone to get to two’s complement.
	+ <https://www.youtube.com/watch?v=Z3mswCN2FJs> (4:15)
* Craig and Dave discuss Two’s complement in their video *OCR A’Level Two’s Complement*. They use a slightly different way of doing the conversions but achieve the same results. Perhaps their approach will resonate with you.
	+ <https://www.youtube.com/watch?v=YtMv4u-9poQ> (4:02)

🔑 3. Give the base 10 value of the following 8-bit two’s complement binary numbers. Write your answer so as to be convincing that you understand the process. Using steps similar to those shown in class is a good way to do this. Correct answers without sufficient work will receive ✔︎ or ✔︎- scores.

 a. 0100 11012TC

 b. 1101 01102TC

🔑 4. Give the 8-bit two’s complement representation of the following base 10 values. Write your answer so as to be convincing that you understand the process. Using steps similar to those shown in class is a good way to do this. Correct answers without sufficient work will receive ✔︎ or ✔︎- scores.

 a. 10310

 b. -7810

🔑 5. Use your answers to the previous question to find the two’s complement representations of the following base 10 values *by taking their complement*. It is not necessary to show the conversion to Two’s Complement again. You did that in the earlier exercise. Write your answer so as to be convincing that you understand the process. Using steps similar to those shown in class is a good way to do this. Correct answers without sufficient work will receive ✔︎ or ✔︎- scores.

 a. -10310

 b. 7810

**Two’s Complement Arithmetic:**

🔑 6. Unlike trying to do arithmetic with sign magnitude representation, binary additions of both positive and negative numbers in two’s complement representation will give the correct result (unless there is arithmetic overflow – more later).

Perform the following binary additions of 4-bit two’s complement numbers. Your answers must also be in 4-bit two’s complement. So, as with unsigned additions, you must simply discard any carry out from the most significant bit.

Check your answers by converting the result of the binary addition to from two’s complement to base 10 and writing it as the answer to the base 10 arithmetic problem given (feel free to use an on-line converter).

 a.

 10112TC -5

 + 00112TC + 3

b.

 01012TC 5

 + 10012TC + -7

 c.

 00112TC 3

 + 00102TC + 2

d.

 11012TC -3

 + 11002TC + -4

One other added bonus of two’s complement arithmetic is that it makes subtraction quite easy. Instead of building circuits for both addition and subtraction, the subtraction operation can be performed by negating the subtracted value. For example, the operation A – B can be performed as A + (-B). The operations of flipping the bits of B and adding 1 can be performed very fast with very little hardware. Thus, the complexity of the circuits for arithmetic using two’s complement are both faster and cheaper than they would be using other representations, which might be more intuitive to us.

**Why Two’s Complement?**

Sign magnitude is fairly straight forward and simple to understand. Two’s complement on the other hand is a little more obtuse, seems unnecessarily complicated and is harder to understand. But Two’s Complement is the representation that is used by all modern computer for signed whole numbers. This section digs into Two’s Complement a little bit to shed some light on why it is used instead of the simpler sign magnitude.

🏆 7. In class we saw that sign magnitude representation had the strange quirk of having two different ways to represent 0 (0000 00002SM = positive 0 and 1000 00002SM = negative 0). Give a detailed example and explanation that shows that Two’s Complement has only one representation of 0 (i.e. it does not have both a positive and negative 0).

🏆 8. In class we also saw that in sign magnitude, if we add a number to its negative, we did not get zero as expected. Produce an example showing that Two’s Complement does not suffer from this same issue. Give a few sentences explaining your example and how it demonstrates that Two’s Complement does not have the same problem as sign magnitude.

**Two’s Complement Data Types:**

9. Modern computer processors are built to perform arithmetic of signed integers using two’s complement representation. Thus, languages such as Java/C/C++ (and pretty much every other language out there) also use two’s complement representation for signed integers. Java like many other languages has a number of different signed integer types, each of which uses a different number of bytes. Use your favorite search engine to complete the table below for each of Java’s signed integer data types.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  | **Data****Type** | **Number ofbits** | **MinimumBase 10 Value** | **MaximumBase 10 Value** |  |
|  | byte |   |   |   |  |
|  | short |   |   |   |  |
|  | int |   |   |   |  |
|  | long |   |   |   |  |
|  |  |  |  |  |  |

10. Using your favorite search engine, find an expression for each of the following values:

a. The minimum (i.e. largest negative) base 10 value that can be represented in Two’s Complement using n bits. Check your answer against the table in question 9.

b. The maximum (i.e. largest positive) base 10 value that can be represented in Two’s Complement using n bits. Check your answer against the table in question 9.

🏆 11. You may have noticed in the table in question 9 and in your answers to question 10 the range of positive and negative numbers for each data type is not the same. There is always one more negative number than there are positive numbers. That seems a bit quirky. Why is that? Let’s see…

a. Does 1000 00002TC represent a positive or negative 8-bit Two’s complement number? How do you know?

b. Take the complement of the number from part a. What happens?

c. Is the result in part b a positive or negative two’s complement value? How do you know?

d. One way to interpret what you have seen in parts a-c is that the number 1000 00002TC does not have a complement. That is, there is no 8-bit Two’s Complement representation of the complement (i.e. positive) of this negative two’s complement number. So, this value is that one extra negative number.

Given that and what you know about the range of 8-bit Two’s Complement numbers, what is the base 10 value represented by the 8-bit Two’s Complement value 1000 00002TC be?

While that’s all a little weird, given what we’ve seen about abstractions so far, it shouldn’t be too surprising. We know that sometimes the details that we have hidden behind our abstractions cause seemingly strange behavior… at least until we peal back the abstraction and look inside.

**Arithmetic Overflow in Two’s Complement:**

From the previous section you know that the range of integer values that can be represented by each datatype that uses Two’s Complement is limited. Just like with the unsigned representations we saw in homework 5, arithmetic overflow can occur when an operation (addition, multiplication, subtraction, etc.) produces a value that is out of range for the datatype. The questions in this section explore arithmetic overflow in Two’s Complement.

12. For this question we will assume we are working with a Java byte (8-bits) data type, which uses Two’s Complement representation.

a. What is the largest positive base 10 value that can be stored in a Java byte? Yep, you found this earlier. Copy it here so that it is fresh in your mind for this question.

b. What is the Two’s Complement binary representation of the largest positive value that can be stored in a Java byte?

c. Write out and solve an addition problem in Two’s Complement binary representation that shows 1 being added to the value you found in part b.

d. Is the result you obtained in part c a positive or negative Two’s complement value? How can you tell?

13. You now know enough that we can look inside Java’s int abstraction and fully understand and explain the behavior of the (non)infinite loop example we saw on the first day of class. You can find that program here:

* [https://repl.it/@braughtg/NonInfiniteLoop](https://repl.it/%40braughtg/NonInfiniteLoop#Main.java)

Using what you now know, explain in a few sentences why the while loop in the program linked above is not an infinite loop. You should include sufficient detail in your explanation to convince a reader that you fully understand. Hint: Fully understanding question 12 will help!

**Overflow in the Real World:**

The (non)infinite loop may seem like a bit of a silly example. But it turns out that this same issue also shows up in the real world, sometimes with significant associated costs.

A fairly harmless example happened on YouTube when the 2012 smash hit Gangnam Style “*broke the internet*” by appearing to be liked a hugely negative number of times.



🏆 14. There are also many examples of less harmless instances of arithmetic overflow occurring in real systems. Use your favorite search engine to read and learn about one of the two following issues:

* The 2038 Problem (also known as the Y2K38 Problem)
* The Boeing 787 Dreamliner Reboot Problem (you’ll probably need to look at a few articles until you find one that talks about the underlying cause).

Give a few sentences describing the issue that occurs and how it is related to today’s topic.

🏆🏆 15. Study and then run the short program at: [https://replit.com/@braughtg/ByteAdd](https://replit.com/%40braughtg/ByteAdd). Describe what you might reasonably expect the output to be and why. Then give a detailed and precise explanation for why the output is what it actually is.

Optional: To help me improve and scope these activities for future semesters please consider providing the following feedback.

a. Approximately how much time did you spend on this activity outside of class time?

b. Please comment on any particular challenges you faced in completing this activity.