**1 Negative numbers**

* 1. **From decimal to binary**

Convert the following numbers to binary for a CPU that works with 8 bits and stores negative numbers in the 2’s complement format:

1. **-128**  
   Note: 8-bit binary can express values from -128 to 127!
2. Binary Conversion Algorithm is utilized on the non-negative value 128:

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 128 | 64 | 0 |
| 64 | 32 | 0 |
| 32 | 16 | 0 |
| 16 | 8 | 0 |
| 8 | 4 | 0 |
| 4 | 2 | 0 |
| 2 | 1 | 0 |
| 1 | 0 | 1 |
| Binary value is: 10000000 -> 0b10000000 | | |

1. Since initial value was negative, we invert all the bits:  
   0b10000000 -> 0b01111111
2. Then we add 1 to the binary:

0b01111111+ 0b00000001 = 0b10000000

1. -128 is represented by the signed binary 0b10000000
2. **127**
3. Binary Conversion Algorithm is utilized on the non-negative value 127:

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 127 | 63 | 1 |
| 63 | 31 | 1 |
| 31 | 15 | 1 |
| 15 | 7 | 1 |
| 7 | 3 | 1 |
| 3 | 1 | 1 |
| 1 | 0 | 1 |
| Binary value is: 1111111 -> 0b01111111 | | |

No need for further actions, since the number is positive.

1. 127 is represented by the signed binary 0b01111111
2. -1  
   Using the programming calculator inside windows, we get: 0b11111111
3. 3  
   Using the programming calculator inside windows, we get: 0b00000011
4. -64  
   Using the programming calculator inside windows, we get: 0b11000000
   1. **From binary to decimal**Convert the following numbers to decimal. They are stored in the 2’s complement format, in 8-bit registers.
5. **00001111**

We assume signed integer.

1. Since value begins with 0, is must be positive! Thus we DO NOT invert the bits.

Convert to decimal:

1. Decimal value is:

Solution = 15

1. **10001111**
2. Find 2’s Complement by inverting all the bits, since binary represents negative value (begins with 1)  
   0b10001111 -> 0b01110000 (*it is now a positive number*)
3. Add 1 to the inverted bits, and throw away any carry:

0b01110000 + 00000001 -> 0b01110001

Convert to decimal:

1. Invert the decimal value, because the initial binary pattern was negative.

Solution = -113

1. **11010000**
2. Find 2’s Complement by inverting all the bits, since binary represents negative value (begins with 1)  
   0b11010000 -> 0b00101111 (*it is now a positive number*)
3. Add 1 to the inverted bits, and throw away any carry:

0b00101111 + 00000001 -> 0b00110000

Convert to decimal:

1. Invert the decimal value, because the initial binary pattern was negative.

Solution = -48

* 1. **Addition/Subtraction**

What’s the result of the following calculation on an 8-bit processor:

1. 10001111 + 00001111 = ?
2. First convert both values to dec:

10001111 is negative, since it begins with 1.

We could using the methods presented earlier, for now we just use the calculator:

10001111 = -113

00001111 = 15

1. Calculate addition:

-113 + 15 = -98

1. Convert to 8-bit signed binary again:

Ansvar is: 10011110

1. 00001111 + 11111111 = ?
2. First convert both values to dec:

We could using the methods presented earlier, for now we just use the calculator:

00001111 = 15

11111111 = -1

1. Calculate addition:

15 + -1 = 14

1. Convert to 8-bit signed binary again:

Ansvar is: 00001110

* 1. **Multiplication**

What’s the result of the following calculations on an 8-bit processor:

1. 00001111 \* 11111111 = ?
2. First convert both values to dec:

We could using the methods presented earlier, for now we just use the calculator:

00001111 = 15

11111111 = -1

1. Calculate addition:

15 \* -1 = -15

1. Convert to 8-bit signed binary again:

Ansvar is: 11110001

1. 00010000 + 11111110 = ?
2. First convert both values to dec:

We could using the methods presented earlier, for now we just use the calculator:

00010000 = 16

11111110 = -2

1. Calculate addition:

16 \* -2 = -32

1. Convert to 8-bit signed binary again:

Ansvar is: 11100000

1. **Negative numbers**

I.e. show the individual separate steps. Do it by hand first and then check your solution against https://www.boolean-algebra.com/ .

**A math equations with arrows and letters

Description automatically generated with medium confidence**

1. A screenshot of a grid of numbers

   Description automatically generated**Boolean arithmetic**

**A white background with black text

Description automatically generated**

**From the truth table, we can extract the following function by evaluating each input, that leads to a 1 in output:**

1. **??**
2. **Data Flip Flop**

**4.1. Simulation**

The Data Flip Flop can be implemented in multiple different ways. 2 of the ways can be seen below.

**A diagram of a computer

Description automatically generated**

Build these 2 circuits in Logisim (using “abstract gates”), and test that they behave the same.

**Consider the following 2 questions:**

**• When is a bit saved in a memory cell?** Answer: Memory is saved whenever the WriteEnable pins are enabled (powered to high)!

**• Does it matter whether a 0 or a 1 is saved?** Answer: No, for the purposes of these gates, both zeroes and ones are saved equally,  
whenever the writeEnable is powered to high.

**4.2. Timing diagram. Fill out the following timing diagrams:**

The timing diagrams shows the inputs W\_E (Write Enable) and Data to some DFF (Data Flip Flop). Draw the missing line for the DFF’s output:

A diagram of a diagram

Description automatically generated



**4.3 Flip Flop implementation with real chips**

Now we will recreate the Data Flip Flop with NAND gates as shown in Figure 1 with real chips.

1. But first create a Logisim simulation of the Flip Flop using the TTL building block(s) (i.e. not the “abstract gates” but the ones inside a chip with real pins, voltage and ground). Apart from finding out how to use the 4 nand gates in the chip, you will also get a good idea about how to wire up your breadboard.