Problem 1.

1. What is the 6-bit two’s complement representation of the decimal number -21?

\[ 21 = 16+4+1 = 0b010101, \text{ (using 6 bit binary)} \]
\[ -21 = 0b101010 + 1 = 0b101011 \]

2. What is the hexadecimal representation for decimal -51 encoded as an 8-bit two’s complement number?

\[ 51 = 32+16+2+1 = 0b0011_0011, \text{ (using 8-bit binary)} \]
\[ -51 = 0b1100_1101 = 0xCD \]

3. The hexadecimal representation for an 8-bit two’s complement number is 0xD6. What is its decimal representation?

\[ 0xD6 = 0b1101_0110 = -128+64+16+4+2 = -42 \]

Alternative (may be easier):
\[ 0xD6 = 0b1101_0110 \text{ which is negative, so use } -0xD6 = 0b0010_1010 = 42, \text{ which gives } +0xD6 = -42 \]

4. Since the start of official pitching statistics in 1988, the highest number of pitches in a single game has been 172. Assuming that remains the upper bound on pitch count, how many bits would we need to record the pitch count for each game as a two’s complement binary number?

9 bits

8-bit two’s complement only allows us to encode numbers from -128 to 127, whereas 9 bits can encode from -256 to 255.

5. Can the value of the sum of two 2’s complement numbers 0xB3 + 0x47 be represented using an 8-bit 2’s complement representation? If so, what is the sum in hex? If not, write NO.

Yes: negative + positive is always within range.
\[ 0xB3+0x47 = \]
\[ 0b1011_0011 + \]
\[ 0b0100_0111 = \]
\[ 0b1111_1010 = 0xFA \text{ (in decimal: -6, can you see why it’s a very small negative?)} \]
6. Can the value of the sum of two 2’s complement numbers 0xB3 + 0xB1 be represented using an 8-bit 2’s complement representation? If so, what is the sum in hex? If not, write NO.

No: negative + negative gave us positive, not okay.
0xB3 + 0xB1 =
0b1011_0011 +
0b1011_0001 =
0b0110_0100 :(( (“9th bit” dropped: we’re in 8-bit notation)

7. Please compute the value of the expression 0xBB – 8 using 8-bit two’s complement arithmetic and give the result in decimal (base 10).

0xBB - 8 = 0b1011_1011 + -0b0000_1000 =
0b1011_1011 +
0b1111_1000 =
0b1011_0011 =
-77 (negative, so positive is 0b0100_1101 = 77) =

This is okay: negative - positive is always okay. (Same as positive-negative)

8. What is the smallest (most negative) integer that can be represented as an 8-bit two’s complement integer? Give your answer as a decimal integer.

0b1000_0000 = -128

9. The following operations are performed on an 8-bit adder. Give the 8-bit sum produced for each, in hexadecimal.

0xF0 + 0x34 = 0x24

0xF0 + 0x80 = 0x70

10. Using a 5-bit two’s complement representation, what is the range of integers that can be represented with a single 5-bit quantity?

-2^4 to (2^4)-1
-16 to 15

11. Consider the following subtraction problem where the operands are 5-bit two’s complement numbers. Compute the result and give the answer as a decimal (base 10) number.
$$\begin{array}{c}
10101 \\
-00011
\end{array}$$

\begin{align*}
\text{0b1_0101} & \quad \text{0b1_0101} & \quad \text{0b1_0101} \\
- \text{0b0_0011} & \Rightarrow + \text{0b1_1100} + 1 & \Rightarrow + \text{0b1_1101} \\
= \text{0b1_0010} & = - (\text{0b0_1101} + 1) = -14
\end{align*}
Problem 2.

1. Given an unsigned $N$-bit binary integer $= b_{n-1} \ldots b_1 b_0$, prove that $\nu$ is a multiple of 4 if and only if $b_0 = 0$ and $b_1 = 0$.

- Powers of 2 greater than or equal to 4 are multiples of 4 (for all $n \geq 2$, $2^n = 4 \cdot 2^n$ and $2^n$ is an integer)
- The sum of numbers divisible by 4 is divisible by 4.
  Therefore any number of the form $b_{n-1} \ldots 00$ is a multiple of 4

  If a number ends in one of 01, 10, or 11 we are adding 1, 2, or 3 respectively to a multiple of 4. Therefore it a number is a multiple 4 only if it ends in 00.

2. Does the same relation hold for two’s complement encoding?

   Yes. The above proof is unmodified: the highest order bit is now $-2^n$ instead of $+2^n$. 
**Problem 3.**

Contrast zero-extension and sign-extension, and discuss the `zeroExtend()`, `signExtend()`, and `truncate()` functions.

Zero-extension always pads with 0s, whereas sign-extension pads with the current highest-order bit.
- Zero extension preserves the arithmetic meaning of positive numbers (5 stays 5), but not of negative numbers.
- Sign-extension preserves the arithmetic meaning of all numbers (5 stays 5, -2 stays -2).

Truncate also changes the width of a number by removing the higher-order bits. This does not preserve the arithmetic meaning of the number.
Problem 4.

You are given a Bluespec function `add16` that adds two 16-bit numbers with carry-in and carry-out, shown below (the carry-out is the most significant bit of the output).

```
function Bit#(17) add16(Bit#(16) a, Bit#(16) b, Bit#(1) c_in);
```

1. Using `add16`, write code for the 32-bit carry-select adder shown right (this is the same carry-select adder we discussed in lecture).

```
function Bit#(32) csa (Bit#(32) a, Bit#(32) b);
    Bit#(32) sum = 0;
    let lower_add = add16(a[15:0], b[15:0], 1'b0);
    sum[15:0] = lower_add[15:0]; // Lower 16-bits

    // Do the computation for both carry=0 and carry=1
    Bit#(16) upper_add_0 = truncate(add16(a[31:16], b[31:16], 1'b0));
    Bit#(16) upper_add_1 = truncate(add16(a[31:16], b[31:16], 1'b1));

    // Choose the right computation
    sum[31:16] = (lower_add[16] == 0) ? upper_add_0 : upper_add_1;
    return sum;
endfunction
```

2. Carry-select adders can be built using more than two blocks. The circuit shown below uses four 16-bit carry-select blocks connected in series to perform 64-bit addition. Using `add16`, write code for this 64-bit adder.