← Lab 5 solutions

1. Write the ranges of **unsigned binary numbers** with the following numbers of bits:
    - 4 bits
        - **0 to 15 (2^4 - 1)**
    - 8 bits
        - **0 to 255 (2^8 - 1)**
    - 11 bits
        - **0 to 2047 (2^11 - 1)**
2. Write the ranges of **signed two's complement binary numbers** with the following numbers of bits:
    - 4 bits
        - **-8 to 7** (half of 2^4 - remember, more *negatives* than *positives*)
    - 8 bits
        - **-128 to 127**
    - 11 bits
        - **-1024 to 1023**
3. Convert these **decimal numbers** to binary.
    - 13
        - **0000 1101** (I just wrote leading 0s for consistency, you don't need them)
    - 58
        - **0011 1010**
    - 141
        - **1000 1101**
4. Convert these **unsigned binary numbers** to decimal.
    - `01001001`
        - **73**
    - `00011001`
        - **25**
    - `10000000`
        - **128**
5. Convert these **signed two's complement binary numbers** to decimal.
    - `01001001`
        - **73** (positive numbers look the same in any system)
    - `11111001`

        - **-7** (NOT gives `0000 0110` (6), then add 1)
    - `10000000`
        - **-128** (sign bit is "-128s" place)
6. Write the **binary representation** of these **signed two's complement binary numbers**, but extended to **16 bits.**
    - `01001001`
        - `0000 0000 0100 1001` (sign bit is 0)
    - `11111001`
        - `1111 1111 1111 1001` (sign bit is 1)
    - `10000000`
        - `1111 1111 1000 0000` (sign bit is 1)
7. Compute the following **bitwise operations.**
    - `~00111001`
        - `1100 0110`
    - `11100110 & 01110001`
        - `0110 0000`
    - `11100110 | 01110001`
        - `1111 0111`
8. I have a register which contains the value `0xE315DEAD`. I use `sw` to store it to memory. Write the **sequence of bytes** that would be placed in memory if our computer is using:
    - Little-endian integers
        - **Start at the *little end* of the word (0xAD): `AD DE 15 E3`.**
    - Big-endian integers
        - **Start at the *big end* of the word (0xE3): `E3 15 DE AD`.** (big-endian is kind of "in order.")
9. I have an array where **each item is 16 bytes long.** If I want to access the 7th item (that is, `array[6]`), how many bytes do I have to move forward from the beginning of the array?
    - **Each item is 16 bytes, and we want to access the item at index 6, so it's 6 × 16 = 80 bytes.**
10. Let's say `t3` contains `44` and `a1` contains `1054`. For the instruction `sb t3, (a1)`, explain **what data is copied into what location.**
    - **The value of `t3` is copied into memory at the address given by `a1`.**
    - **So, you could think of it as `Memory[1054] = 44`, since memory is an array of bytes.**
11. In MIPS, when you load a **byte from memory** into a register:
    - What happens to its value? (There are two options.)

        - **It could be sign-extended or zero-extended.**
        - (Sign extension "smears" the sign bit to the left; zero extension just puts 0s.)
    - Why do we do this?
        - **Because the registers are 32 bits and a byte is only 8.**
        - **This preserves the *value* of the byte - a byte-size -44 becomes a word-sized -44, etc.**
12. Encode the following integers as single-precision IEEE 754 floats, and **write your answer as an 8-digit hexadecimal number.** Do not treat them as 2's complement, just use the sign given.
    - `+1000111010`
        - **Sign is `0` for positive**
        - **Signed exponent would be +9, plus the bias constant of 127, is 136 = `1000 1000`**
        - **Fraction is everything after the first 1, so `000111010 000000...`**
        - **Entire 32-bit binary representation is `0100 0100 0000 1110 1000 0000 0000 0000`**
        - **Converted to hex, that's `0x440E8000`**
            - *If you use the floating point representation tool in MARS, you can type in 570 in the bottom right box and hit enter, and you will see all these parts*
    - `-1000111010`
        - **Sign is `1`, signed exponent is +9; fraction is...**
        - *wait a second*
        - *This is just -570*
        - *So no need to continue. Just flip the sign bit of the previous problem's answer, because it's sign-magnitude.*
        - `0xC40E8000`
    - `+1` ` **Sign is 1, exponent is 0 + 127, fraction is all 0s...**
        - `0x3F800000`