Lab 1: Setup and Instruction Set

- **Link**
  - Download and install MPLAB X IDE : https://youtu.be/tplN7KKPd1k
  - Introduction to Instruction Set : https://youtu.be/APFJanXuhPs
  - HackMD : https://hackmd.io/@hazelnut9/HykeNJcugx

- **Basic** (50%)
  - **Description**

    Store four unsigned numbers in memory such that x1 is placed at address [0x000], x2 at [0x001], y1 at [0x002], and y2 at [0x003]. First, add x1 and x2 together and store the result, denoted as A1, in [0x010]. Next, subtract y2 from y1 and store the result, denoted as A2, in [0x011]. Finally, compare the two results, A1 and A2. If A1 is greater than A2, write the value 0xFF to memory location [0x020]; otherwise, write the value 0x01 to [0x020]. You may assume that the addition will not overflow (i.e., x1 + x2 ≤ 0xFF) and that the subtraction will not produce a negative result (i.e., y1 ≥ y2).

  - **Example**

Address	Testcase 1	Testcase 2
[0x000]	0x03	0x08
[0x001]	0x02	0x08
[0x002]	0xB5	0x0F
[0x003]	0x04	0x02
[0x010]	0x05	0x10
[0x011]	0xB1	0x0D
[0x020]	0x01	0xFF

  - **Standard of grading**
    1. You are required to use the **CPFSGT** instruction. For addition and subtraction operations in your design, you may also use **ADDWF**, **SUBWF**, or other similar instructions as needed.
    2. All computed results must be stored in the correct and specified memory locations.

- **Advance** (30%)
  - **Description**

    Store an 8-bit number at memory location [0x000], then design a loop to

calculate its count leading zeros (CLZ) value. The calculation should begin by examining the most significant bit (bit 7, MSB) and proceed sequentially toward bit 0. Count the consecutive zeros that appear before the first 1 is encountered, and stop once the first 1 is found. Finally, store the resulting **clz** value in memory location [0x010].

- **Example**

Address	Testcase 1	Testcase 2
[0x000]	b'00001000	b'01101000
[0x010]	0x04	0x01

- **Standard of grading**
  1. Please use a loop to complete this task without using any brute force methods.
  2. The result must be stored in [0x010].

- **Hard** (20%)
  - **Description**

    In the previous section, we implemented a loop-based version of the **count leading zeros (CLZ)** function. In this section, we turn to a **branchless implementation** of **clz**. From the perspective of computer organization, we know that branch instructions may cause **pipeline stalls** or **idleness**. By eliminating branches, the branchless version avoids these stalls and achieves more consistent performance.

    The following code demonstrates a branchless implementation of **clz** for a **32-bit unsigned integer**. However, since our target microprocessor operates on **8-bit data**, we must adapt this approach to an **8-bit branchless clz**.

    Your task is as follows:
    1. Store an **8-bit input number** in memory at address [0x000].
    2. Compute the number of leading zeros using the branchless **clz** method.
    3. Store the result at address [0x010].

    In the hard section, we expect you to:
    1. **Derive** the 8-bit version of the branchless clz algorithm.
    2. **Translate** it into **assembly code** under the given microprocessor's instruction set.

```c
int clz32(uint32_t x) {
  int r = 0, c;

  c = (x < 0x00010000) << 4;
  r += c;
  x <<= c;  // off 16

  c = (x < 0x01000000) << 3;
  r += c;
  x <<= c;  // off 8

  c = (x < 0x10000000) << 2;
  r += c;
  x <<= c;  // off 4

  c = (x < 0x40000000) << 1;
  r += c;
  x <<= c;  // off 2

  c = x < 0x80000000;
  r += c;
  x <<= c;  // off 1

  r += x == 0; // If all bits are zero, then the code adds one to the result.
  return r;
}
```

- **Example**

Address	Testcase 1	Testcase 2
[0x000]	b'00001000	b'01101000
[0x010]	0x04	0x01

- **Standard of grading**
  1. The result must be stored at address [0x010].
  2. The CLZ (count leading zeros) function must be implemented in a provided branchless manner (no conditional branches and in an 8-bit version).
  3. A C reference implementation may be provided to support troubleshooting and lab review.

- **Note:**

  If you have any questions, please send an email to the TA mailbox (mprocessor2025@gmail.com). Emails sent to individual TAs will not be answered.