## **Combinational Arithmetic Logic Unit (ALU) Design Overview**

The objective of this project is to design a combinational Arithmetic Logic Unit (ALU) comprising the following key components:

1. **Adder/Subtractor**
2. **Comparator**
3. **Boolean Operations**
4. **Shifter**

Each of these blocks is designed as a purely combinational module, following principles of logic synthesis.

**1. Adder / Subtractor Unit**

The adder is implemented using a Ripple Carry Adder (RCA) built from full adders. It supports both addition and subtraction operations based on alufn[0].

**Status Outputs:**

* **Z :** It is computed using a 32-bit NOR gate that compares all bits of the output. If all bits are zero, Z = 1.
* **V :** This flag detects arithmetic overflow conditions.
  + **Positive Overflow:** Occurs when adding two positive numbers produces a negative result.
  + **Negative Overflow:** Occurs when adding two negative numbers produces a positive result.
  + The flag V is set (V = 1) when either overflow condition occurs.
* **N :** Indicates the sign of the result by outputting the most significant bit (MSB). If N = 1, the result is negative, if N = 0, it is positive.

**2. Comparator Unit**

The comparator is designed using a 4-to-1 multiplexer (mux\_4), where the selection is based on the alufn[2:1] control input. The comparator supports:

* **Equality Check:** Performed using Z.
* **Less Than Check (A < B):** Determined based on the Negative (N) and Overflow (V) :
  + A < B if (N = 1 and V = 0) or (N = 0 and V = 1).
* **Less Than or Equal To Check (A ≤ B):** Computed as the logical OR of the equality and less-than conditions:
  + A ≤ B if Z = 1 or (N XOR V) = 1.

**3. Boolean Operations Unit**

The boolean logic unit also leverages a mux\_4 multiplexer design, allowing flexible selection of operations based on a predefined truth table controlled by alufn[3:0]. The selection logic is as follows:

* If (a=0, b=0): Select alufn[0]
* If (a=1, b=0): Select alufn[1]
* If (a=0, b=1): Select alufn[2]
* If (a=1, b=1): Select alufn[3]

Where a truth table pattern is created:

| **Operation** | **alufn[3:0]** | **Description** |
| --- | --- | --- |
| AND | 1000 | Output is 1 only when both a and b are 1 |
| OR | 1110 | Output is 1 when either a or b is 1 |
| XOR | 0110 | Output is 1 when a XOR b is 1 |
| Buffer A | 1010 | Outputs the value of a |

Additional operations such as Buffer B, NOR, NAND, and XNOR can be easily implemented by setting the appropriate alufn values.

**4. Shifter Unit**

The shifter is designed using a combination of a 2-to-1 multiplexer (mux\_2) and a modular x\_bit\_left\_shifter.

**Left Shifting:**

Each x\_bit\_left\_shifter module shifts the input left by a specific parameterized amount (1, 2, 4, 8, or 16 bits) when enabled. The shifted result is generated by:

* Taking the lower (32 - SHIFT) bits from the input a
* Concatenating it with SHIFT copies of a padding value (pad), typically zero or the sign bit.

**Right Shifting:**

Right shifts are performed using the left shifter logic with the following modifications:

* **Logical Right Shift:**
  + Reverse the bit order of the input
  + Apply the left shifters
  + Reverse the bits again
  + Pad with zeros
* **Arithmetic Right Shift:**
  + Similar to logical right shift
  + Pads with the sign bit instead of zeros to preserve the sign during shifting

This modular design allows shift operations of any amount from 0 to 31 bits.