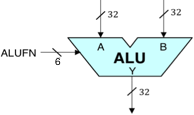
**Introduction:**

Microprocessors are some of the most widely used components in circuits. They are present in laptops, desktops, mobiles, cars and many more devices. Every microprocessor has an essential component called the ALU. The ALU is performs the basic arithmetic and logical operations in a circuit.

The aim of the project is designing and simulating 32 bit ALU ,using hardware description language. The ALU consists of 9 different operations, based on operation code (OP code) the ALU decides that which operation is to be done among various 9 operations. Basically, it is a combinational circuit which take 32 bit data as input , and gives 32 bit output by performing specified arithmetic and logical operations.

**Designing ALU:**

It is a combinational circuit taking two 32-bit data words A and B as inputs, and producing a 32-bit output Y by performing a specified arithmetic or logical function on the A and B inputs. The particular function to be performed is specified by a 6-bit control input. The ALU also has a Z flag and N flag as outputs, signifying whether or not the main (32bit) output is zero or negative respectively. The logical symbol of the ALU is shown below.



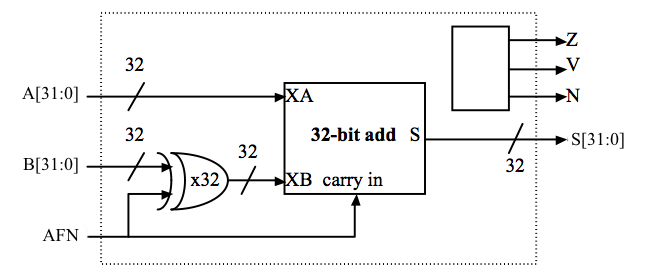
The ALU consists of 9 different operations .The operations and their Boolean expressions are listed in following table.

|  |  |  |
| --- | --- | --- |
| No. | Operation | Boolean expression |
| 1. | Full adder | t[32:0] = A + B + Cin  sum = t[31:0]  Cout = t[32] |
| 2. | Full subtractor | t[32:0] = A – B – Bin  diff = t[31:0]  Bout = t[32] |
| 3. | Logical left shift | Y =A << B |
| 4. | Logical right shift | Y = A >> B |
| 5. | Arithmetic right shift | Y = A>>>B |
| 6. | Equality | Y = A == B |
| 7. | Inequality | Y = A != B |
| 8. | Less than or equal to | Y = A <= B |
| 9. | Greater than | Y = A > B |

**Simulation:**

**1.Arithmetic unit:**  
The ALU has an adder/subtractor that operates on 32-bit two's complement inputs and generates a 32-bit output. It will be useful to generate three other output signals to be used by compare unit Z which is true when the S outputs are all zero, V which is true when the addition operation overflows (*i.e.*, the result is too large to be represented in 32 bits), and N which is true when the sum is negative (*i.e.*, S[31] = 1). Overflow can never occur when the two operands to the addition have different signs; if the two operands have the same sign, then overflow can be detected if the sign of the result differs from the sign of the operands:

V=XA31⋅XB31⋅¯¯¯¯¯¯¯S31+¯¯¯¯¯¯¯¯¯¯¯¯¯XA31⋅¯¯¯¯¯¯¯¯¯¯¯¯¯XB31⋅S31



AFN will be set to 0 for an ADD (S=A+BS=A+B) and 1 for a SUBTRACT (S=A−BS=A−B); A[31:0] and B[31:0] are the 32-bit two's complement input operands; S[31:0] is the 32-bit result; Z/V/N are the three condition code bits described above. The AFN input signal selects whether the operation is an ADD or SUBTRACT. To do a SUBTRACT, the circuit first computes the two's complement negation of the B operand by inverting B and then adding one.

**2.Compare unit:**

The ALU provides four comparison operations for the A and B operands. The adder unit explained above helps to compute A−BA−B and then look at the result (actually just the Z, V and N condition codes) to determine if A=B, A >B ,A!=B or A <= B. The compare operations generate a 32-bit result using the number 0 to represent false and the number 1 to represent true.

ALU has a 32-bit compare unit that generates one of two constants (0 or 1) depending on the control signals (used to select the comparison to be performed) and the Z, V, and N outputs of the adder/subtractor unit. Clearly the high order 31 bits of the output are always zero. The least significant bit (LSB) of the output is determined by the comparison being performed and the results of the subtraction carried out by the adder/subtractor.

**3.Shift unit:**

The ALU consist of a 32-bit shifter that implements logical left shift (SHL), logical right shift (SHR) and arithmetic right shift (SRA) operations. The A operand supplies the data to be shifted and the low-order 5 bits of the B operand are used as the shift count (*i.e.*, from 0 to 31 bits of shift).

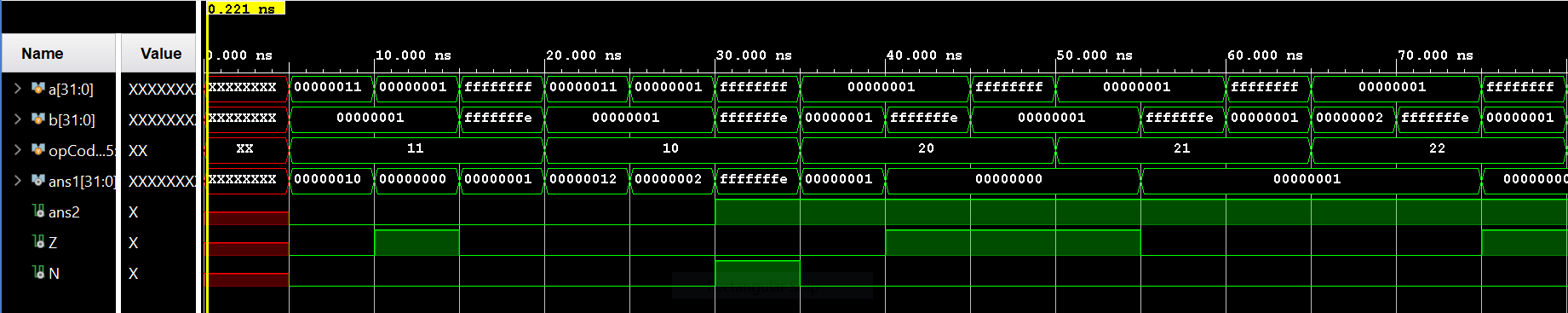
|  |  |
| --- | --- |
| **Operation** | **SFN[1:0]** |
| SHL (shift left) | 00 |
| SHR (shift right) | 01 |
| SRA (shift right with sign extension ) | 11 |

With this encoding, SFN[0] is 0 for a left shift and 1 for a right shift and SFN[1] controls the sign extension logic on right shift. For SHL and SHR, 0's are shifted into the vacated bit positions. For SRA ("shift right arithmetic"), the vacated bit positions are all filled with A[31], the sign bit of the original data so that the result will be the same as dividing the original data by the appropriate power of 2.

The simplest implementation is to build two shifters — one for shifting left and one for shifting right. It's easy to build a shifter after noticing that a multi-bit shift can be accomplished by cascading shifts by various powers of 2. The figure below shows a possible implementation of the left shift logic; the right shift logic is similar with the slight added complication of having to shift in either 0 or A[31], depending on the value of SFN[1].

**Results:**

The ALU design created dealt with 8 operations – Addition ,Subtraction ,Equality check between 2 numbers ,not equal to check between 2 numbers ,greater than and less than or equal to check between 2 numbers ,logical left shift ,logical right shift and lastly arithmetic right shift .The design was simulated using the test bench and after a successful run produced the following outputs



This is the snippet of the waveform generated while simulating the designed ALU a ,b are the inputs ans1 is the required output N tells weather the answer is negative or positive and Z tells weather the answer is zero or not .The wave is produced using the simulation tool VIVADO.

**Conclusion:**

The 32 bit ALU is successful and it ran very well in the simulation part where it dealt with the various conditions in the testbenches under different modules and simulated the correct outputs .

Therefore, we succeeded in programming a 32 bit ALU which performs the desired tasks .

The future scope of this project can be planning and development of a CU which can be integrated with this ALU which in turn helps us understand the overall working.

**References:**

[C:\Users\Dell\Videos\alu\playlist.htm](file:///C:\Users\Dell\Videos\alu\playlist.htm)

[C:\Users\Dell\Videos\alu\download.htm](file:///C:\Users\Dell\Videos\alu\download.htm)

[C:\Users\Dell\Videos\alu\playlist (1).htm](file:///C:\Users\Dell\Videos\alu\playlist%20(1).htm)