INTRO TO PROCESSOR ARCHITECTURE
ASSIGNMENT-1

Name: Anushka Agrawal & Meghana Tedla

Roll No: 2021102023 & 2021102002

ALU DESIGN

An arithmetic-logic unit is the part of CPU that carries out arithmetic and logical operations on the operands in computer instruction words.

We designed an ALU that performs four different arithmetic and logical operations on 64-bit operands: ADD, SUB, AND, XOR. This ALU takes two 64-bit operands and the operations to be performed as its inputs and returns a 64-bit output and an overflow bit (in case of ADD and SUB).

We have created three different modules for the specified operations (ADD and SUB are performed using the same module). A separate test bench has been written to test the functioning of each module separately.

ADDITION and SUBTRACT OPERATION:

The addition and subtraction operations were performed with the help of Full Adders. A single Full Adder performs the addition of two one-bit numbers and the carry input. For performing the addition of binary numbers with more than one bit, more than one full adder is required in parallel, and the number of Full Adders depends on the number of bits.

ADDER:

By connecting 64 full adders in parallel, a 64-bit Parallel Adder can be constructed.

SUBTRACTOR:

A 64-bit parallel subtractor can be implemented using 64 full adders. The subtraction operation is performed by considering the principle that the addition of minuend and the complement of the subtrahend is equivalent to the subtraction process. We know that the subtraction of A by B is obtained by taking 2's complement of B and adding it to A. The 2's complement of B is obtained by taking 1's complement and adding 1 to the least significant pair of bits. Hence, we can obtain the 1's complement of B with the inverters and a 1 can be added to the sum through the input carry.

IMPLEMENTATION OF ADDITION AND SUBTRACTION IN ONE MODULE:

The operations of both addition and subtraction can be performed by one common binary adder. Such a binary circuit can be designed by adding an XOR gate with each full adder. The mode input control line M relates to the carry input of the least significant bit of the full adder. This control line decides the type of operation, whether addition or subtraction.

When M = 1, the circuit is a subtractor, and when M=0, the circuit becomes an adder. The XOR gate consists of two inputs to which one is connected to the B and the other to input M.

When M = 0, B XOR 0 produces B. Hence, Addition operation is performed.

When M = 1, B XOR of 1 produces complement of B and the carry is 1. Hence the complemented B inputs are added to A and 1 is added through the input carry (2's complement operation). Therefore, the subtraction operation is performed.

We created a module named "addsub_" to perform the addition and subtraction operations for 64-bit inputs. This module takes two 64-bits numbers as input and returns a 64-bit number as output.

This was implemented by creating a full adder module and using the module within a 'for' loop which runs 64 times for all the 64 bits while specifying the control line input 'M' that decides the type of operation.

- Overflow:

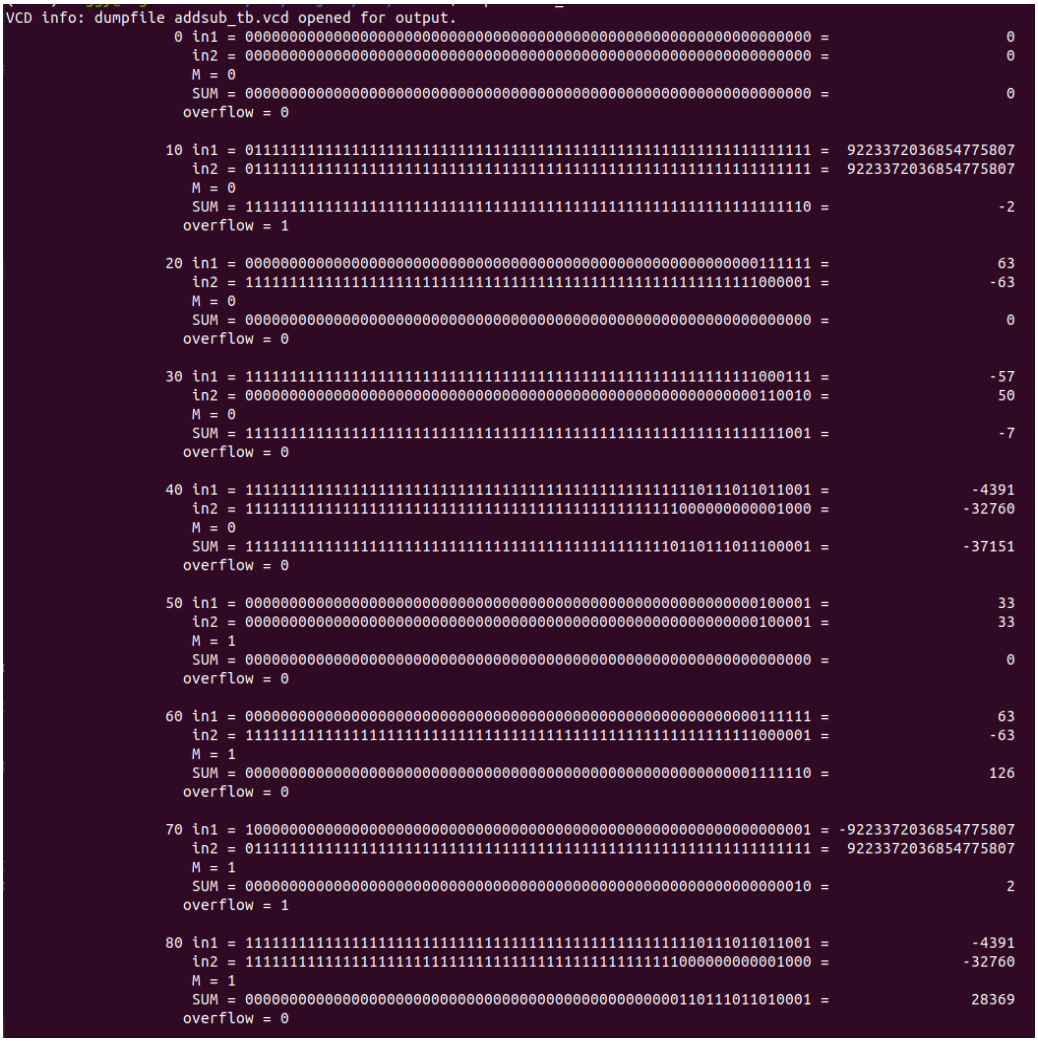
  When two signed numbers are added, the sign bit is treated as part of the number and the end carry does not indicate an overflow. An overflow may occur if the two numbers added are both positive or both negative. An overflow condition can be detected by observing the carry into the sign bit position and the carry out of the sign bit position. If these two carries are not equal, an overflow occurs. If the two carries are applied to an XOR gate, an overflow is detected when the output of the gate is equal to 1.

The verilog file for ADD and SUBTRACT module:

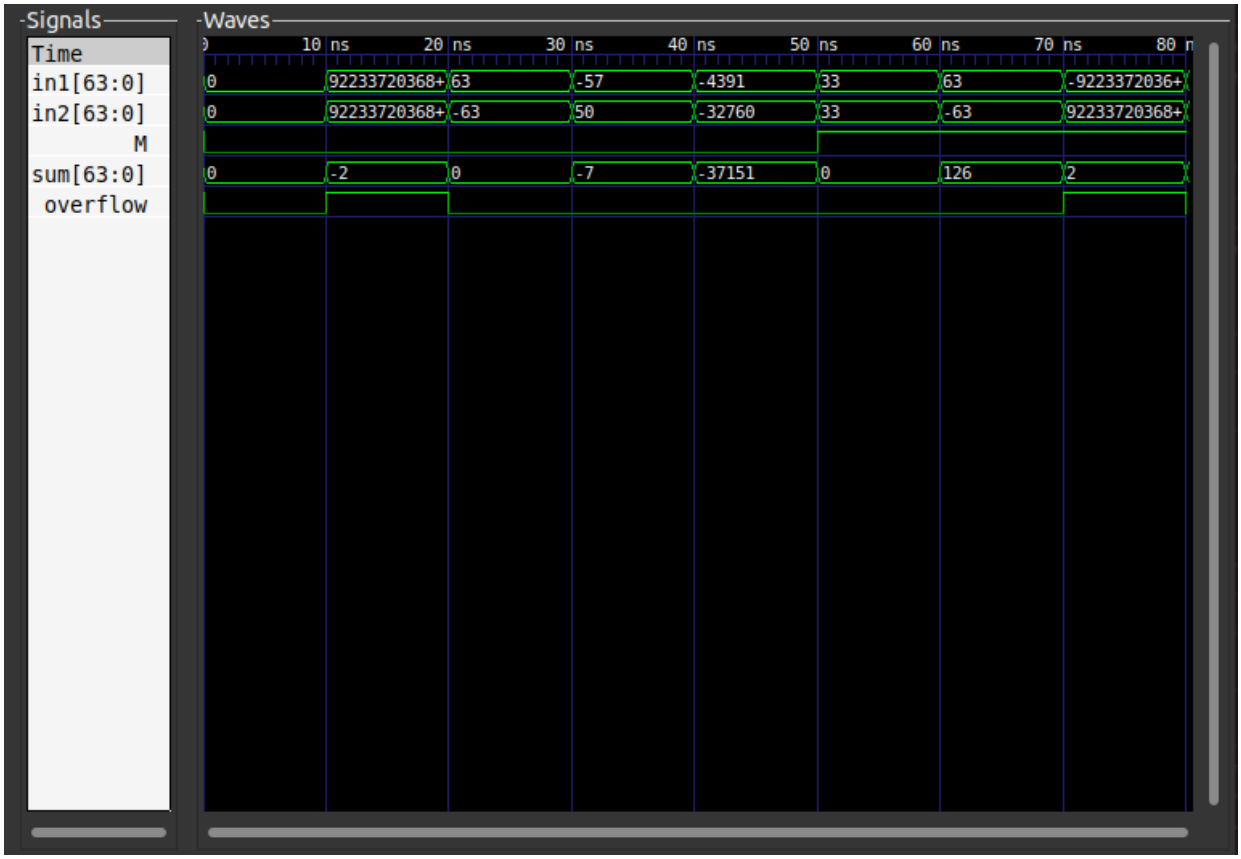
Project > ALU > ADDSUB > addsub.v

```v
// Implementation of a 64 bit Adder and Subtractor using 64 1-bit Adders

module FA(a, b, c, M, sum, carry);

input a, b, c;
input M;
output sum, carry;
wire B, x, y, z;

xor x1(B, b, M);

xor x2(x, a, B);
xor x3(sum, x, c);
and a1(y, a, B);
and a2(z, x, c);
or o1(carry, y, z);

endmodule


module addsub (in1, in2, M, sum, overflow);

input [63:0]in1;
input [63:0]in2;
input M;

output [63:0]sum;
output overflow;

wire [64:0]C;

assign C[0] = M;

genvar i;

generate
    for(i = 0; i < 64; i=i+1)
    begin
        FA full_adder(in1[i], in2[i], C[i], M, sum[i], C[i+1]);
    end
endgenerate


xor x1(overflow, C[64], C[63]);

endmodule

```

The output produced for some input combinations:



The gtk waveforms for these input combinations:



AND OPERATION:

We created a module named “and_” to perform AND operation for 64-bits input. This module takes two 64-bits numbers as input and returns a 64-bit number as output who's each bits the AND of corresponding bits of the two inputs. We wrote a for loop that runs 64 times and AND’s all the bits and stores the result in the corresponding output bit.

The verilog file for AND module:

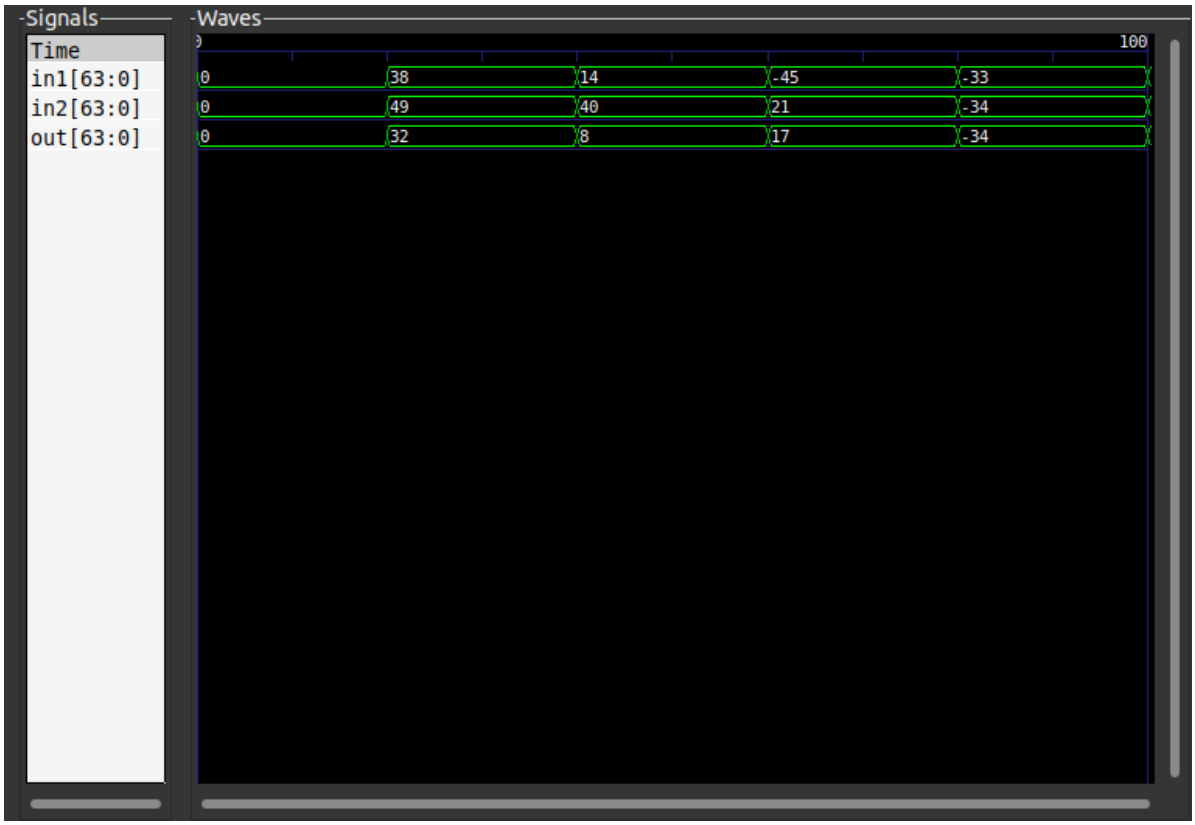
ALU > AND > and.v

```
1   module and_ (in1, in2, out);
2
3       input  [63:0] in1, in2;
4       output [63:0] out;
5
6       genvar i;
7       generate
8           for (i = 0; i < 64; i = i + 1) begin
9               and(out[i], in1[i], in2[i]);
10          end
11      endgenerate
12
13  endmodule
```

The output produced for some input combinations:

```
VCD info: dumpfile and_tb.vcd opened for output.
                   0 in1 = 0000000000000000000000000000000000000000000000000000000000000000
                     in2 = 0000000000000000000000000000000000000000000000000000000000000000
                     out = 0000000000000000000000000000000000000000000000000000000000000000

                  20 in1 = 0000000000000000000000000000000000000000000000000000000000100110
                     in2 = 0000000000000000000000000000000000000000000000000000000000110001
                     out = 0000000000000000000000000000000000000000000000000000000000100000

                  40 in1 = 0000000000000000000000000000000000000000000000000000000000001110
                     in2 = 0000000000000000000000000000000000000000000000000000000000101000
                     out = 0000000000000000000000000000000000000000000000000000000000001000

                  60 in1 = 1111111111111111111111111111111111111111111111111111111111010011
                     in2 = 0000000000000000000000000000000000000000000000000000000000010101
                     out = 0000000000000000000000000000000000000000000000000000000000010001

                  80 in1 = 1111111111111111111111111111111111111111111111111111111111011111
                     in2 = 1111111111111111111111111111111111111111111111111111111111011110
                     out = 1111111111111111111111111111111111111111111111111111111111011110

                 100 in1 = 0000000000000000000000000000000000000000000000000000000000101111
                     in2 = 0000000000000000000000000000000000000000000000000000000000111001
                     out = 0000000000000000000000000000000000000000000000000000000000101001
```

The gtk waveforms for these input combinations:



XOR OPERATION:

We created a module named “xor_” to perform XOR operation for 64-bits input. This module takes two 64-bits numbers as input and returns a 64-bit number as output who's each bits the XOR of corresponding bits of the two inputs. We wrote a for loop that runs 64 times and XOR's all the bits and stores the result in the corresponding output bit.

The verilog file for XOR module:

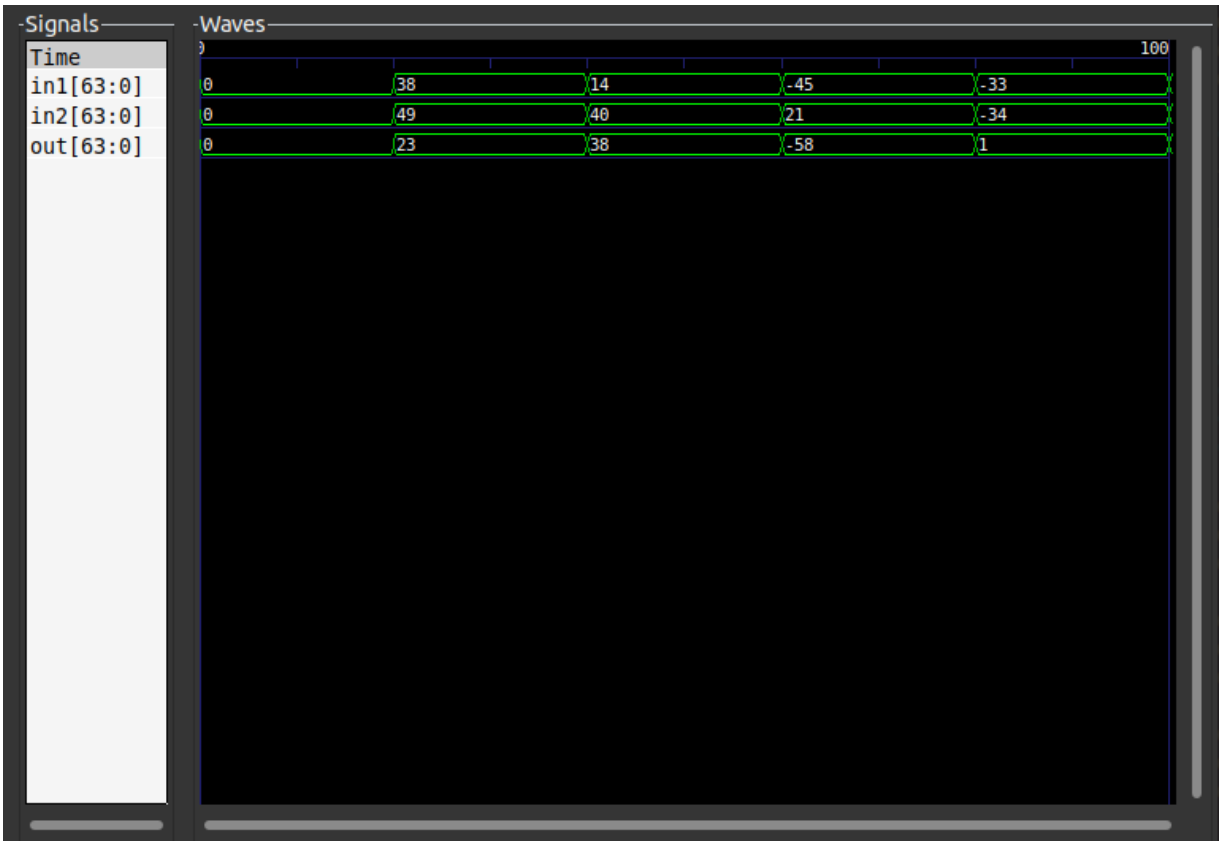
ALU > XOR > xor.v

```v
module xor_ (in1, in2, out);

    input  [63:0] in1, in2;
    output [63:0] out;

    genvar i;
    generate
        for (i = 0; i < 64; i = i + 1) begin
            xor(out[i], in1[i], in2[i]);
        end
    endgenerate

endmodule
```

The output produced for some input combinations:

```
VCD info: dumpfile xor_tb.vcd opened for output.
                   0 in1 = 0000000000000000000000000000000000000000000000000000000000000000
                     in2 = 0000000000000000000000000000000000000000000000000000000000000000
                     out = 0000000000000000000000000000000000000000000000000000000000000000

                  20 in1 = 0000000000000000000000000000000000000000000000000000000000100110
                     in2 = 0000000000000000000000000000000000000000000000000000000000110001
                     out = 0000000000000000000000000000000000000000000000000000000000010111

                  40 in1 = 0000000000000000000000000000000000000000000000000000000000001110
                     in2 = 0000000000000000000000000000000000000000000000000000000000101000
                     out = 0000000000000000000000000000000000000000000000000000000000100110

                  60 in1 = 1111111111111111111111111111111111111111111111111111111111010011
                     in2 = 0000000000000000000000000000000000000000000000000000000000010101
                     out = 1111111111111111111111111111111111111111111111111111111111000110

                  80 in1 = 1111111111111111111111111111111111111111111111111111111111011111
                     in2 = 1111111111111111111111111111111111111111111111111111111111011110
                     out = 0000000000000000000000000000000000000000000000000000000000000001

                 100 in1 = 0000000000000000000000000000000000000000000000000000000000101111
                     in2 = 0000000000000000000000000000000000000000000000000000000000111001
                     out = 0000000000000000000000000000000000000000000000000000000000010110
```

The gtk waveforms for these input combinations:



ALU:

A final wrapper ALU unit from where the ADD, SUBTRACT, AND, and XOR modules are called based on the control input. The ALU unit takes as input the control signal, and two 64-bit inputs, and returns the 64-bit output corresponding to the control signal chosen.

Control 0 - ADD x and y

Control 1 – Subtract y from x

Control 2 – AND x and y

Control 3 – XOR x and y

Every time an input is given, all the four operations add, subtract, AND, XOR are computed and based on the control signal the required value is given as output.

The verilog file for XOR module:

Project > ALU > alu.v

```v
`include "ADDSUB/addsub.v"
`include "AND/and.v"
`include "XOR/xor.v"

module alu_(inp1, inp2, op, out, overflow);
    input signed [63:0] inp1, inp2;
    input  [1:0] op;
    output signed [63:0] out;
    output overflow;

    wire signed [63:0] out_add, out_sub, out_and, out_xor;
    reg signed [63:0] ans;
    reg overflow_;

    addsub_ add (inp1, inp2, 1'b0, out_add, overflow1);
    addsub_ sub (inp1, inp2, 1'b1, out_sub, overflow2);
    and_ a1 (inp1, inp2, out_and);
    xor_ x1 (inp1, inp2, out_xor);

    always@(*)
    begin
        if(op==2'b00)
        begin
            ans = out_add;
            overflow_ = overflow1;
        end
        else if(op == 2'b01)
        begin
            ans = out_sub;
            overflow_ = overflow2;
        end
        else if(op == 2'b10)
            ans = out_and;
        else if(op == 2'b11)
            ans = out_xor;
    end

    assign out = ans;
    assign overflow = overflow_;

endmodule
```

The output produced for some input combinations:

```
VCD info: dumpfile alu_tb.vcd opened for output.
                   0 Operation = 00
                     inp1 = 00000000000000000000000000000000000000000000000000000000000000000000 =                    0
                     inp2 = 00000000000000000000000000000000000000000000000000000000000000000000 =                    0
                     out  = 00000000000000000000000000000000000000000000000000000000000000000000 =                    0
                     OF = 0

                  20 Operation = 00
                     inp1 = 00000000000000000000000000000000000000000000000000000000000000110101 =                   53
                     inp2 = 00000000000000000000000000000000000000000000000000000000000000010110 =                   22
                     out  = 00000000000000000000000000000000000000000000000000000000000001001011 =                   75
                     OF = 0

                  40 Operation = 01
                     inp1 = 00000000000000000000000000000000000000000000000000000000000000110101 =                   53
                     inp2 = 00000000000000000000000000000000000000000000000000000000000000010110 =                   22
                     out  = 00000000000000000000000000000000000000000000000000000000000000011111 =                   31
                     OF = 0

                  60 Operation = 10
                     inp1 = 00000000000000000000000000000000000000000000000000000000000000110101 =                   53
                     inp2 = 00000000000000000000000000000000000000000000000000000000000000010110 =                   22
                     out  = 00000000000000000000000000000000000000000000000000000000000000010100 =                   20
                     OF = 0

                  80 Operation = 11
                     inp1 = 00000000000000000000000000000000000000000000000000000000000000110101 =                   53
                     inp2 = 00000000000000000000000000000000000000000000000000000000000000010110 =                   22
                     out  = 00000000000000000000000000000000000000000000000000000000000000100011 =                   35
                     OF = 0

                 100 Operation = 00
                     inp1 = 11111111111111111111111111111111111111111111111111111111111111001011 =                  -53
                     inp2 = 00000000000000000000000000000000000000000000000000000000000000010110 =                   22
                     out  = 11111111111111111111111111111111111111111111111111111111111111100001 =                  -31
                     OF = 0

                 120 Operation = 01
                     inp1 = 11111111111111111111111111111111111111111111111111111111111111001011 =                  -53
                     inp2 = 00000000000000000000000000000000000000000000000000000000000000010110 =                   22
                     out  = 11111111111111111111111111111111111111111111111111111111111110110101 =                  -75
                     OF = 0

                 140 Operation = 10
                     inp1 = 11111111111111111111111111111111111111111111111111111111111111001011 =                  -53
                     inp2 = 00000000000000000000000000000000000000000000000000000000000000010110 =                   22
                     out  = 00000000000000000000000000000000000000000000000000000000000000000010 =                    2
                     OF = 0

                 160 Operation = 11
                     inp1 = 11111111111111111111111111111111111111111111111111111111111111001011 =                  -53
                     inp2 = 00000000000000000000000000000000000000000000000000000000000000010110 =                   22
                     out  = 11111111111111111111111111111111111111111111111111111111111111011101 =                  -35
                     OF = 0

                 180 Operation = 00
                     inp1 = 00000000000000000000000000000000000000000000000000000000000000110101 =                   53
                     inp2 = 11111111111111111111111111111111111111111111111111111111111111101010 =                  -22
                     out  = 00000000000000000000000000000000000000000000000000000000000000011111 =                   31
                     OF = 0

                 200 Operation = 01
                     inp1 = 00000000000000000000000000000000000000000000000000000000000000110101 =                   53
                     inp2 = 11111111111111111111111111111111111111111111111111111111111111101010 =                  -22
                     out  = 00000000000000000000000000000000000000000000000000000000000001001011 =                   75
                     OF = 0

                 220 Operation = 10
                     inp1 = 00000000000000000000000000000000000000000000000000000000000000110101 =                   53
                     inp2 = 11111111111111111111111111111111111111111111111111111111111111101010 =                  -22
                     out  = 00000000000000000000000000000000000000000000000000000000000000100000 =                   32
                     OF = 0

                 240 Operation = 11
                     inp1 = 00000000000000000000000000000000000000000000000000000000000000110101 =                   53
                     inp2 = 11111111111111111111111111111111111111111111111111111111111111101010 =                  -22
                     out  = 11111111111111111111111111111111111111111111111111111111111111011111 =                  -33
                     OF = 0

                 260 Operation = 00
                     inp1 = 11111111111111111111111111111111111111111111111111111111111111001011 =                  -53
                     inp2 = 11111111111111111111111111111111111111111111111111111111111111101010 =                  -22
                     out  = 11111111111111111111111111111111111111111111111111111111111110110101 =                  -75
                     OF = 0

                 280 Operation = 01
                     inp1 = 11111111111111111111111111111111111111111111111111111111111111001011 =                  -53
                     inp2 = 11111111111111111111111111111111111111111111111111111111111111101010 =                  -22
                     out  = 11111111111111111111111111111111111111111111111111111111111111100001 =                  -31
                     OF = 0

                 300 Operation = 10
                     inp1 = 11111111111111111111111111111111111111111111111111111111111111001011 =                  -53
                     inp2 = 11111111111111111111111111111111111111111111111111111111111111101010 =                  -22
                     out  = 11111111111111111111111111111111111111111111111111111111111111001010 =                  -54
                     OF = 0

                 320 Operation = 11
                     inp1 = 11111111111111111111111111111111111111111111111111111111111111001011 =                  -53
                     inp2 = 11111111111111111111111111111111111111111111111111111111111111101010 =                  -22
                     out  = 00000000000000000000000000000000000000000000000000000000000000100001 =                   33
                     OF = 0

                 340 Operation = 00
                     inp1 = 01111111111111111111111111111111111111111111111111111111111111111111 =  9223372036854775807
                     inp2 = 00000000000000000000000000000000000000000000000000000000000000000001 =                    1
                     out  = 10000000000000000000000000000000000000000000000000000000000000000000 = -9223372036854775808
                     OF = 1
```

```
360 Operation = 01
    inp1 = 0111111111111111111111111111111111111111111111111111111111111111 =  9223372036854775807
    inp2 = 1111111111111111111111111111111111111111111111111111111111111111 =                    -1
    out  = 1000000000000000000000000000000000000000000000000000000000000000 = -9223372036854775808
    OF = 1
```

The gtk waveforms for these input combinations: