# Project Report

## 1. Objective

The objective of this project is to write a Verilog module which do is to do math co-processing like ALU. This ALU supports the following instructions, and their corresponding opcodes are also shown below:

```
parameter  sla = 5'b00000,  // sla  -- 0
srai = 5'b00001,            // srai -- 1
add = 5'b00010,             // add  -- 2
sub = 5'b00011,             // sub  -- 3
mult = 5'b00100,            // mult -- 4
div = 5'b00101,             // div  -- 5
addi = 5'b00110,            // addi -- 6
addu = 5'b00111,            // addu -- 7
subu = 5'b01000,            // subu -- 8
multu = 5'b01001,           // multu-- 9
divu = 5'b01010,            // divu -- 10
addiu = 5'b01011,           // addiu-- 11
sqrt = 5'b01100,            // sqrt -- 12

// To prevent conflict with key word, the logic instructions begin with '_'
_and = 5'b01101,            // _and -- 13
_or = 5'b01110,             // _or  -- 14
_nor = 5'b01111,            // _nor -- 15
_xor = 5'b10000,            // _xor -- 16
_xnor = 5'b10001,           // _xnor-- 17

andi = 5'b10010,            // andi -- 18
ori = 5'b10011,             // ori  -- 19
slt = 5'b10100,             // slt  -- 20
slti = 5'b10101;            // slti -- 21
```

## 2. I/O

This ALU takes **4** inputs:

Two 32-bit inputs ***a, b***;
One 5-bit input ***opcode***;
One 16-bit input ***immediate***.

And it generates **6** outputs:

Three 32-bit outputs ***c, HI, LO***;
Three 1-bit flags: ***zero, overflow, neg***.

```
module alu(a,b,immediate,opcode,c,HI,LO,zero,overflow,neg);

output signed[31:0] c, HI, LO;
output zero;
output overflow;
output neg;

input signed[31:0] a,b;
input[4:0] opcode;
input [15:0] immediate;
```

For input ***immediate***: It is a 16-bit binary number. Since input ***a*** is 32-bit binary number, immediate will be zero-extended (*addiu, andi, ori*) or sign-extended (*addi, slti*) to 32-bit.

In some instructions, some inputs may not be used, for example, input ***b*** will not be used in *addi* instruction. In this case, the unused input will keep its earlier value instead of clearing it to 0.

For outputs ***c, HI, LO***: ***HI*** and ***LO*** are outputs for *mult, multu, div, divu*, and *c* is output for the rest.

For the flags: ***zero*** and ***neg*** detect the output of all instructions; ***overflow*** only detects *add, addi, sub* (no overflow in *mult* and *multu* since ***HI*** and ***LO*** can store the output).

## 3. Unfinished work

```
// For 'sqrt', I fail to implement it since verilog doesn't allow variable index.
// However, I still leave this block of C-style code here to show my effort :)
// ------------------------------------------------------------------------------
// sqrt:
// begin
//     for (i = 15; i >= 0; i = i - 1) begin
//         reg_C = reg_C * 2 + 1;
//         if ((reg_C ** 2) > reg_A[31:i*2]) reg_C[0] = 0;
//         else reg_C[0] = 1;
//     end
//     zf = !reg_C[31:0];
// end
```

## 4. How to run

To run the program, you need to write this in command line under the directory including module files:

1. Compile:

*iverilog -o test 32bALU.v test_32bALU.v*

2. Run:

*./test*

*3.Result:*

```
yihongpengdeMacBook-Pro:project3 yihongpeng$ iverilog -o test 32bALU.v test_32bALU.v
yihongpengdeMacBook-Pro:project3 yihongpeng$ ./test
op: a      : b      : imm: c      : HI     : LO     : zf: of: nf: reg_A  : reg_B  : reg_C
 xx:xxxxxxxx:xxxxxxxx:xxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx: x : x : x :xxxxxxxx:xxxxxxxx:xxxxxxxxx
 00:dddddddd:xxxxxxxx:xxxx:bbbbbbba:00000000:00000000: 0 : 0 : 0 :dddddddd:xxxxxxxx:0bbbbbbba
 00:40404040:xxxxxxxx:xxxx:80808080:00000000:00000000: 0 : 0 : 0 :40404040:xxxxxxxx:080808080
 01:fdfdfdfd:xxxxxxxx:xxxx:7efefefe:00000000:00000000: 0 : 0 : 0 :7efefefe:xxxxxxxx:07efefefe
 01:39393939:xxxxxxxx:xxxx:1c9c9c9c:00000000:00000000: 0 : 0 : 0 :1c9c9c9c:xxxxxxxx:01c9c9c9c
 02:80000000:ffffffff:xxxx:7fffffff:00000000:00000000: 0 : 1 : 0 :80000000:ffffffff:17fffffff
 03:00000001:ffffffff:xxxx:00000002:00000000:00000000: 0 : 0 : 0 :00000001:ffffffff:100000002
 04:80000002:00000002:xxxx:00000000:ffffffff:00000004: 0 : 0 : 1 :80000002:00000002:000000000
 05:00000009:00000002:xxxx:00000000:00000001:00000004: 0 : 0 : 0 :00000009:00000002:000000000
 06:00000001:00000002:ffff:00000000:00000000:00000000: 1 : 0 : 0 :00000000:00000002:100000000
 07:80000001:80000001:ffff:00000002:00000000:00000000: 0 : 0 : 0 :80000001:80000001:100000002
 08:00000001:80000000:ffff:80000001:00000000:00000000: 0 : 0 : 1 :00000001:80000000:180000001
 09:80000002:00000002:ffff:00000000:00000001:00000004: 0 : 0 : 0 :80000002:00000002:000000000
 0a:80000001:80000001:ffff:00000000:00000000:00000001: 0 : 0 : 0 :80000001:80000001:000000000
 0b:00000001:80000001:ffff:00010000:00000000:00000000: 0 : 0 : 0 :00010000:80000001:000010000
 0d:ffffffff:ff0f0f0f:ffff:ff0f0f0f:00000000:00000000: 0 : 0 : 0 :ffffffff:ff0f0f0f:0ff0f0f0f
 0e:00000006:00000007:ffff:00000007:00000000:00000000: 0 : 0 : 0 :00000006:00000007:000000007
 0f:00000006:00000007:ffff:fffffff8:00000000:00000000: 0 : 0 : 0 :00000006:00000007:1fffffff8
 10:00000006:00000007:ffff:00000001:00000000:00000000: 0 : 0 : 0 :00000006:00000007:000000001
 11:00000006:00000007:ffff:fffffffe:00000000:00000000: 0 : 0 : 0 :00000006:00000007:1fffffffe
 12:00000006:00000007:ffff:00000006:00000000:00000000: 0 : 0 : 0 :00000006:00000007:000000006
 13:00000006:00000007:ffff:0000ffff:00000000:00000000: 0 : 0 : 0 :0000ffff:00000007:00000ffff
 14:00000006:00000007:ffff:00000001:00000000:00000000: 0 : 0 : 0 :00000006:00000007:000000001
 15:00000006:00000007:ffff:00000000:00000000:00000000: 1 : 0 : 0 :00000006:00000007:000000000
yihongpengdeMacBook-Pro:project3 yihongpeng$
```