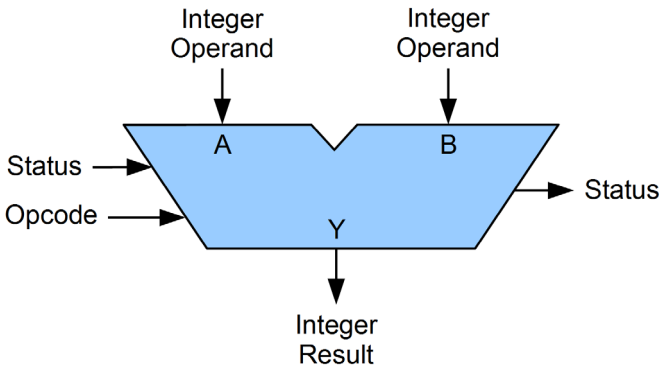
Project 3 report

1. Overview

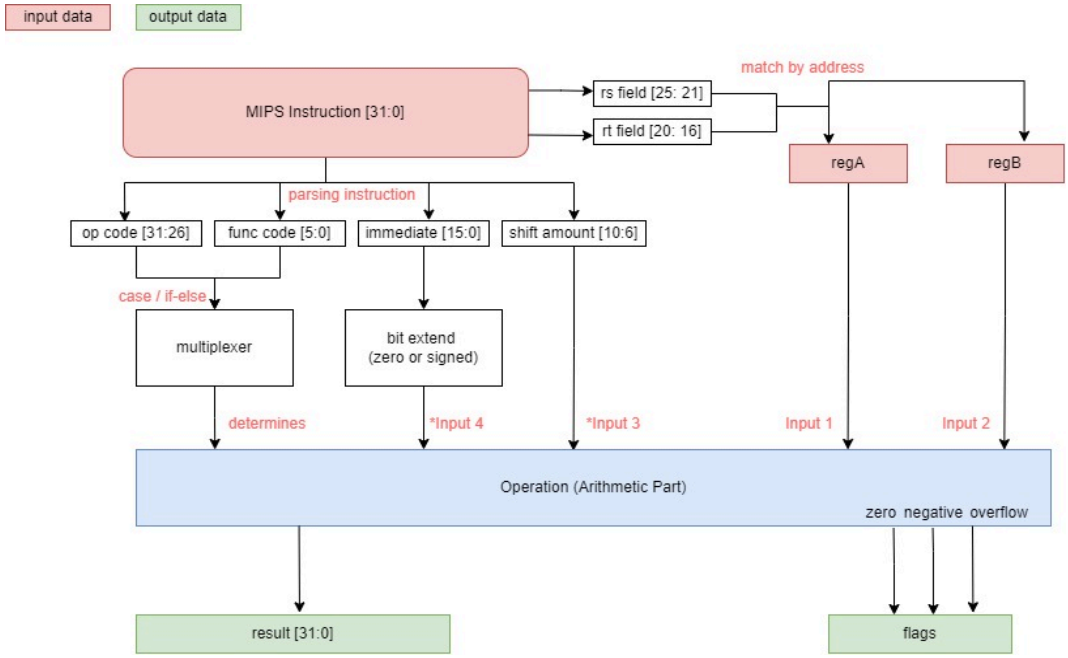
Generally, in computing, an arithmetic logic unit (ALU) is a combinational digital circuit that performs arithmetic and bitwise operations on integer binary numbers. In this project, I designed a simple 32-bit CPU which supports simple instruction parsing, register value fetching, and ALU functions. Since the CPU part is trivial, the main focus of this project is to implement the ALU logic.



Graph 1: standard ALU I/O

2. Data flow chart

The chart below illustrates the basic ideas of this project. Note that this chart is only for demonstrating the logic of the code rather than being a real digital circuit chart.



Graph 2: Data flow chart

3. General Logic

In this part, the interpretation of the data flow chart and the actually implementation in the code are going to be introduced.

As shown in the chart, the three given inputs are `instruction (32-bit machine code)`, `regA (32-bit value)`, `regB (32-bit value)`. The program parses the instruction code into 6 fields, that is, `op code`, `function code`, `immediate`, `shift amount`, `rs_field and rt_field`. Although not every operation needs all of these inputs, the program parses all the instruction and fetch the specific data based on the needs.

After parsing, the program matches `regA (addr: 00000)` and `regB (addr: 00001)` according to the `rs_field` and `rt_field` in the instruction. Again, we do this matching for all instruction even when some of them only uses one register. We are allowed to use rs, rt register directly for any instruction.

Next, we set the flag values to 000 before any instruction begins. In this way, we only need to set the flag value in those instructions including flag changes. For the general case without flag change (the demand is specified in the project3.pdf), we don't need to change them later on.

Then, the program uses `case/if-else` statement to branch on `op_code / funct code` each instruction. In each operation, the program utilizes the arithmetic operator such as `+`, `-`, `^`, `>>`, `>>>`, and etc. to do the calculation, and save the result and flags in the according registers.

After each execution, the program assigns the value of register type output and the flags to the wire type result and flags as the final result.

4. Implement details

4.1 Parse instruction and preset the flag value

```
always @ (instruction, regA, regB)

begin
    opcode = instruction[31:26];
    rs_field = instruction[25:21];
    rt_field = instruction[20:16];
    immediate = instruction[15:0];
    reg_flag = 3'b000;
```

4.2 rs, rt matching detail

```
// match the rs, rt register
if (rs_field == 5'b00000)
    rs = regA;
if (rs_field == 5'b00001)
    rs = regB;
if (rt_field == 5'b00000)
    rt = regA;
if (rt_field == 5'b00001)
    rt = regB;
```

4.3 Branch

```
if (opcode == 6'b000000)
begin
    func = instruction[5:0];
    shift_amount = instruction[10:6];

    case(func)
    6'b100000:  // add
    Begin
… … … …
else begin

case(opcode)

6'b001000:    // addi
begin
```

4.4 Overflow detection

The trick for overflow detection is to initialize a register with 33 bits and assign the result to it. Then we examine whether the MSB is 1, if so, then an overflow occurs.

```
        case(func)
        6'b100000:  // add
        begin
            reg_str = "add";
            reg_result33 = rs + rt;
            reg_result = rs + rt;
            if (reg_result33[32] == 1)  //overflow
                reg_flag = 3'b001;
        end
```

4.5 sltiu

Generally, we assign the number as signed/unsigned and it will extend accordingly. However, for sltiu, it needs to do the signed extension while do the unsigned comparation, so we do the signed extension and save it as an unsigned number.

```
signextend_as_unsigned = {{16{immediate[15]}}, immediate};
```

```
    6'b001011: // sltiu
    begin
        reg_str = "sltiu";
        reg_result = rs - signextend_as_unsigned;
        if ($unsigned(rs) < signextend_as_unsigned)
            reg_flag = 3'b010;
```

4.6 Design Logic for beq/bne, slt/slti/sltiu/sltu

For bne/beq, the result(output) = rs – rt, and the zero flag = 1 when result = 0.

For slt/slti/sltiu/sltu, the result = rs – (rt or immediate), and the negative flag = 1 when result is negative.

4.7 Test result

```
~ cd "ass3"
~/ass3 make
iverilog -o test test_alu.v alu.v
vvp test
     instruction :op :func:   regA   :  regB   : result  :flags
        xxxxxxxx :xx : xx : xxxxxxxx :xxxxxxxx :xxxxxxxx :  xxx
  add   00201820 :00 : 20 : 80000001 :80000001 :00000002 :  001
  add   00201820 :00 : 20 : 00000001 :00000001 :00000002 :  000
 addu   00201821 :00 : 21 : ffffffff :00000001 :00000000 :  000
 addu   00201821 :00 : 21 : fffffffe :00000001 :ffffffff :  000
 addi   20000001 :08 : 21 : ffffffff :00000001 :00000000 :  001
 addi   20200001 :08 : 21 : ffffffff :00000001 :00000002 :  000
addiu   2420ffff :09 : 21 : 00000001 :00000001 :00000000 :  000
addiu   24208001 :09 : 21 : 00000001 :00000001 :ffff8002 :  000
  and   00201824 :00 : 24 : 5a0ddddd :a5f22223 :00000001 :  000
 andi   30008000 :0c : 24 : ffff80dd :a5f22223 :00008000 :  000
  nor   00201827 :00 : 27 : 7ffffffe :80000000 :00000001 :  000
   or   00201825 :00 : 25 : 00000000 :fffffff1 :fffffff1 :  000
  ori   34008001 :0d : 25 : 00000000 :fffffff1 :00008001 :  000
  sll   00001200 :00 : 00 : 00000100 :00000001 :00010000 :  000
 sllv   00011004 :00 : 04 : fdddddc8 :00000001 :00000100 :  000
  sra   000110c3 :00 : 03 : dddddddd :80000000 :f0000000 :  000
 srav   00011007 :00 : 07 : dddddddc :8000007f :ff000000 :  000
  srl   000111c2 :00 : 02 : dddddddd :8000007f :01000000 :  000
 srlv   00011006 :00 : 06 : dddddddc :80000007 :10000000 :  000
  sub   00011822 :00 : 22 : 6ddddddd :80000001 :eddddddc :  001
 subu   00201823 :00 : 23 : ffffffff :00000001 :00000002 :  000
  xor   00201826 :00 : 26 : 6ddddddd :92222222 :ffffffff :  000
 xori   3801800f :0e : 26 : 00008000 :92222222 :0000000f :  000
  slt   0020182a :00 : 2a : 00000001 :ffffffff :fffffffe :  010
 sltu   0020182b :00 : 2b : eddddddd :00000001 :12222224 :  010
 slti   2820182b :0a : 2b : 6ddddddd :80000001 :7fffe7d6 :  010
sltiu   2c20982b :0b : 2b : 6ddddddd :80000001 :800067d6 :  010
  beq   1020182b :04 : 2b : 6ddddddd :6ddddddd :00000000 :  100
  beq   1020182b :04 : 2b : 6ddddddd :6ddddddc :ffffffff :  000
  bne   1420182b :05 : 2b : 6ddddddd :6ddddddd :00000000 :  100
  bne   1420182b :05 : 2b : 6ddddddd :6ddddddc :ffffffff :  000
   lw   8c019998 :23 : 2b : 00000001 :0000000f :ffff9999 :  000
   sw   ac019998 :2b : 2b : 00000001 :0000000f :ffff9999 :  000
~/ass3
```