Register transfer ASM ALU

Digital design principles.

Teacher:
Jorge Martínez Carballido

Work made by:
Christian Aaron Ortega Blanco

Hermosillo, Son. 18 de mayo de 2022.

ASM ALU means to an arithmetic-logic unit who as the capacity to add, subtract and multiplier the value of a register (B) to another register (A). on this case the initial value of the registers was given automatically by their position on the memory. For example, after the reset settings, if the register on the position 5 is multiplied by the register 1, that means that the value in binary 00000101 on the 5th register is multiplied by the value 00000001 on the 1st register, the temporary result is saved on an external register. The characteristic of the calculator is expressed in the fallowing block diagram Figure 1.

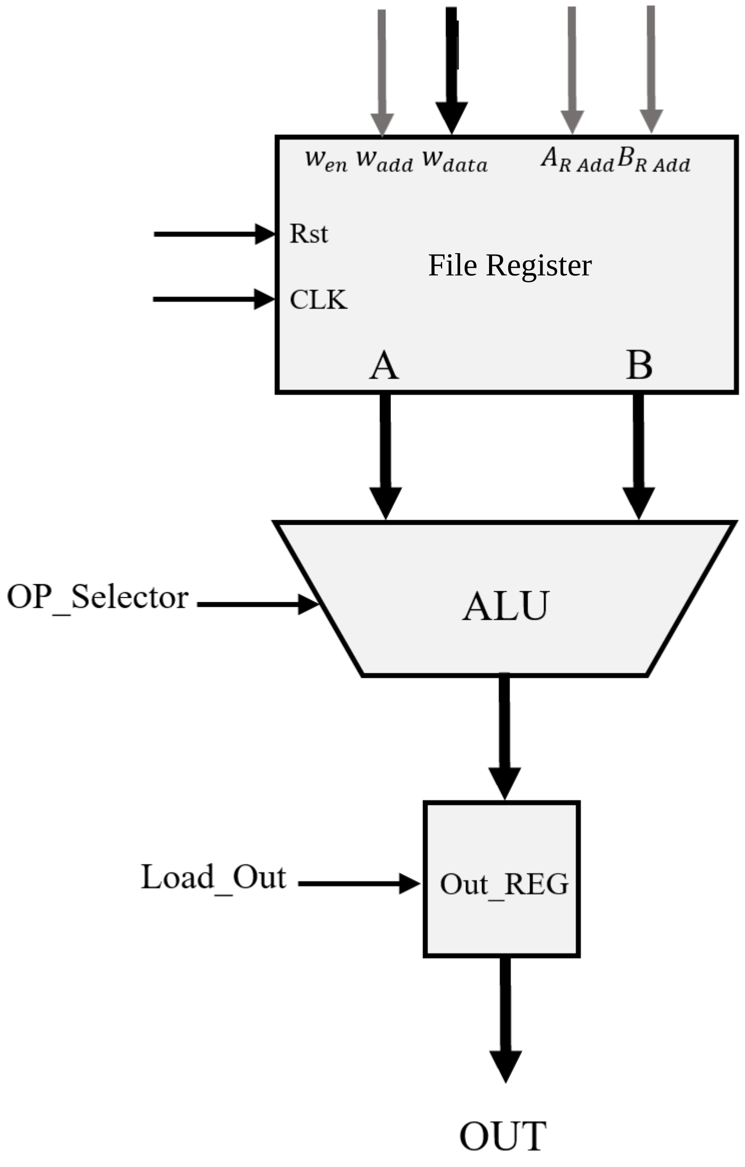


Figure 1 Calculator Block diagram

The design of the ALU is designed before the Verilog design, that's makes easier the programing flux. Because the ALU implements only three operations and two of them could be implemented in one single module (RCA), the ALU design reduces to the fallowing block diagram figure 2.

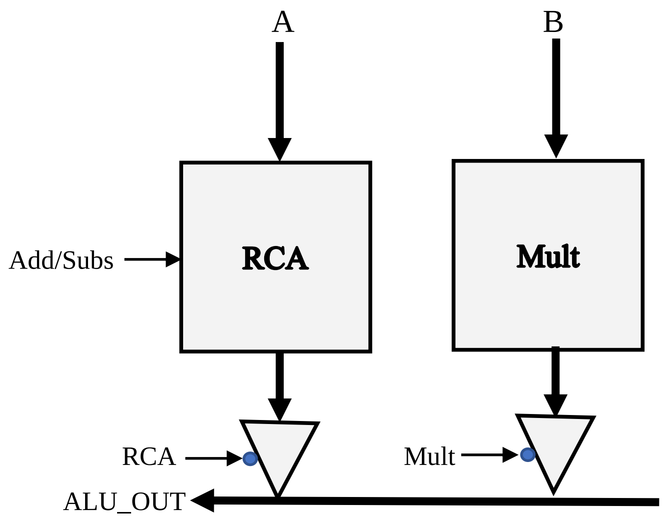


Figure 2 ASM_ALU structure

Now that the architecture is already stabilized and all the modules has been designed, the Verilog programing initialized assigning the main module inputs and outputs, as we are making an 8 bits length calculator, and the memory has only 32 memory spaces, that means that in order to read all the memory, we need 5 bits of addressing inputs, an 8bit output and an 8bit write input. then the module variables could be assigned like these.

$$Calculator(CLK, Rst, W_{en}, W_{add}, W_{data}, A_{add}, B_{add}, OP_{selector}, LD_{out}, RESULT)$$

Where, $\boldsymbol{CLK}$ = clock signal, $\boldsymbol{Rst}$ = reset signal, $\boldsymbol{W_{en}}$ = write enable signal, $\boldsymbol{W_{add}}$ = write address [5bit], $\boldsymbol{W_{data}}$ = write data [8bits], $\boldsymbol{A_{add}}$ = A reading address [5], $\boldsymbol{B_{add}}$ = B reading address [5], $\boldsymbol{OP_{selector}}$ = Arithmetic selection [2][Mul,RCA[subs]], $\boldsymbol{LD_{out}}$ = Load output register signal, $\boldsymbol{RESULT}$ = arithmetic result [8].

As we are adding a subtraction module, it is optimal to include a flag bit that can provide the sign of the operands, and the result. And including a carry bit could help in the future to expand the ALU capacity. So, the bit distribution of the register could be something like.

Sign	Carry	D5	D4	D3	D2	D1	D0

This means that the calculation capacity is reduced to 6bit, but now it’s easier to work with negative numbers.

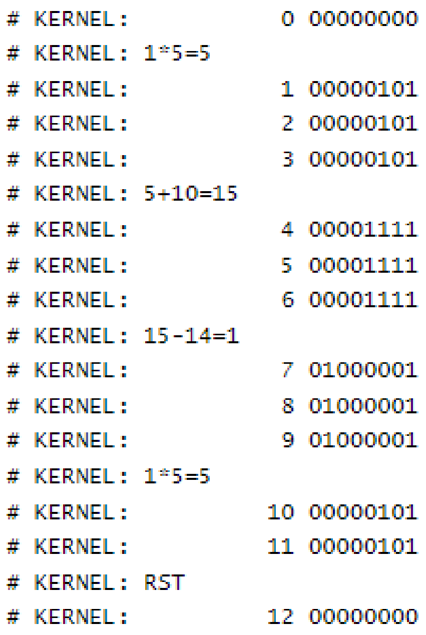
The module could be tested using a cycle control structure in the benchmark, this is useful to send parameters on each clock cycle as bellow.

```
///////////////Secuence///////////////
while (i<14) begin
  if (clk)$display("%d %b", i-1,RSLT);
  if (~clk) begin
    // w=1  FR_Wdata=8 FR_Waddr=5 FR_RAddr_1=5 FR_RAddr_2=5  SRM=2 LD=1            SMR [mult, RCA]   rca= 0suma, 1resta
    case (i)
      0 : begin rst=1'b0; w=1'b0; FR_RAddr_1=1; FR_RAddr_2=5;   end /// reset and set target numbers 1,5 """"
      1 : begin  w=1'b0; rst=1'b1; LD=1; SRM=2'b10; $display("1*5=5"); end  /// set operation and load Result 1*5=5
      2 : begin  w=1'b1; LD=0; FR_Wdata=RSLT;  FR_Waddr=1;  end // save answer in add1 5=>add1



      3 : begin w=1'b0;  FR_RAddr_1=1; FR_RAddr_2=10; end /// set next target to add1 5,10""""""""""""""""
      4 : begin w=1'b0; rst=1'b1; LD=1; SRM=2'b00; $display("5+10=15"); end /// set operation and load Result 5+10=15
      5 : begin w=1'b1; LD=0; FR_Wdata=RSLT;  FR_Waddr=1;  end // save answer in add1 15=>add1




      6 : begin w=1'b0;  FR_RAddr_1=1; FR_RAddr_2=14; end /// set next target to add1 15,15""""""""""""""""
      7 : begin w=1'b0; rst=1'b1; LD=1; SRM=2'b01; $display("15-14=1"); end /// set operation and load Result 15-14=1
      8 : begin w=1'b1; LD=0; FR_Wdata=RSLT;  FR_Waddr=1;  end // save answer in add1 0=>add1



      9 : begin w=1'b0;  FR_RAddr_1=1; FR_RAddr_2=5; end /// set next target to add1 1,5""""""""""""""""
      10 : begin w=1'b0; rst=1'b1; LD=1; SRM=2'b10; $display("1*5=5");  end /// set operation and load Result 1*5=5
      11 : begin w=1'b1; LD=0; FR_Wdata=RSLT;  FR_Waddr=1; end // save answer in add1 5=>add1



      12 : begin rst=1'b0; w=1'b0; FR_RAddr_1=1; FR_RAddr_2=5; $display("RST");  end /// reset and set target numbers 1,5 """"
      default : begin  end
    endcase
    i++;
  end
  #1;
end
```

The behavior of the previous structure allow to specificity on which cycle the change is implemented, for example, if we want to start with a restart settings, we put the restart variable like rst=1'b0; on the first case option, and if we want to write something on a register we put w=1'b1; like on the 2nd case option on the code, this is an instruction to write the Result on the 1st register.

The results of the testbench are reviewed bellow:

Operation cycle Result



```
# KERNEL:              0 00000000
# KERNEL: 1*5=5
# KERNEL:              1 00000101
# KERNEL:              2 00000101
# KERNEL:              3 00000101
# KERNEL: 5+10=15
# KERNEL:              4 00001111
# KERNEL:              5 00001111
# KERNEL:              6 00001111
# KERNEL: 15-14=1
# KERNEL:              7 01000001
# KERNEL:              8 01000001
# KERNEL:              9 01000001
# KERNEL: 1*5=5
# KERNEL:             10 00000101
# KERNEL:             11 00000101
# KERNEL: RST
# KERNEL:             12 00000000
```

Figure 3 testbench results

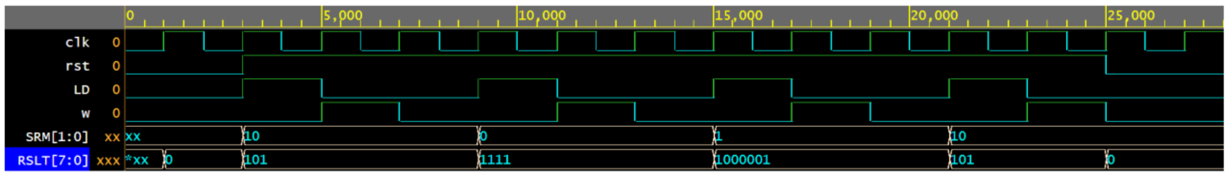


Figure 4 Clock diagram

We can see that every calculation sequence take 3 cycles before the next calculation, but this is only on the case that we have to save the immediate result on the file_register memory.