計算機結構 HW3 Report 資工三 B07902116 陳富春

Using the technique in tutorial video to divide the whole circuit into sequential part and combinational part by adding o_data_r, o_data_w, o_valid_r, o_valid_w, we can focus on implementing needed circuit and the output result will update at posedge i_clk or reset at negedge i_rst_n.

ALU :

For each case, using corresponding operator to implement each function in ALU.

1. Signed add / subtraction :

If the case is subtraction, then we can negate input b and then operate same operation as add.
Adding input a and b and then check the condition to overflow : sign bit of a and b are same and the sign bit of result is wrong.

2. Signed multiplication :

Calculating negative a and b, and then calculate abs(a * b) by {mul_overflow, o_data_w} = a * b;
If the output should be positive, then there is overflow if output sign bit == 1 or mul_overflow != 0. If the output should be negative, then there is overflow if mul_overflow != 0 or (sign bit == 1 and output != 100…000).
Because the range of 32 bits signed integer is $-2^{31} \sim 2^{31} - 1$, so 100…00 is valid in this case.

3. Signed max / min :

First compare by their sign bit, and then compare according their values.

4. Unsigned add / subtraction / multiplication :

{o_overflow_w, o_data_w} = i_data_a + / - / * i_data_b;
Because Verilog assume variables are unsigned, unsigned operator can be directly used.

5. Unsigned max / min :

Simply compare their values and store the value.

6. And / or / xor / BitFlip :

Using corresponding operator & / | / ^ / ~ to implement the operation.

7. BitReverse :

Use a loop to let o_data_w[i] = i_data_w[DATA_WIDTH - 1 - i] to implement BitReverse.

FPU :

I use {sign_a, exponent_a, fraction_a} to store input a, and precision_a to maintain bits lower than fraction_a during operation.
Variable int is used to check whether the number can be normalized.
First, let a stores the input with larger absolute value and initialize variables int, precision_a, precision_b.
Then, adding bit 1 at leftmost for fraction_a and fraction_b, the bit 1 means its integer part.

1. Add :

Because a is always larger, b is shift right by exponent_a - exponent_b.
Also, sign_result and exponent_result is dominated by a because a is larger.
And then calculate fraction_a + fraction_b and normalize.

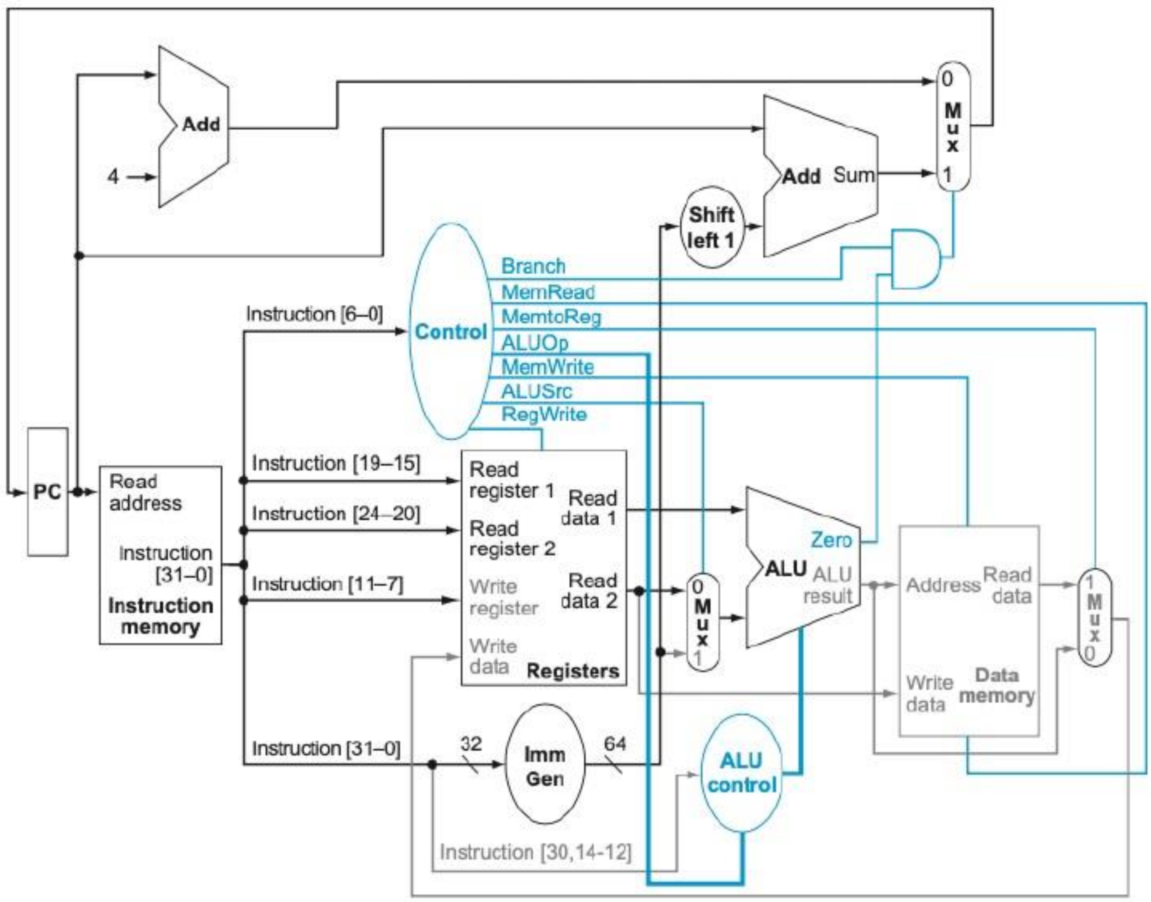
2. Multiplication :

sign_result = sign_a xor sign_b;
exponent_result = exponent_a + (exponent_b - 127);
And then calculate fraction_a * fraction_b and normalize.

After these two operations, we round the result according round-to-nearest-even method.
After rounding, the result is transmitted to o_data_w.

CPU :

I implement the CPU according to the provided circuit.



1. Instruction Fetch :

When i_i_valid_inst is on, fetch the i_i_inst into inst_reg to store the current instruction, and then wait 90ns to let other component finish their work. Then check value of (Branch & zero) to decide the next instruction address and set o_i_valid_addr on to tell instruction memory to give the next instruction.

2. Data Memory :

For operation that will affect data memory (SD or LD), the CPU works depend on MemRead and MemWrite.

(1) SD :

MemRead = 0 and MemWrite = 1. The data to be written into data memory comes from data in register2 and address comes from ALUresult.

So it does o_d_data_w = rs2_data and o_d_addr_w = ALUresult.

(2) LD :

MemRead = 1 and MemWrite = 0. The address of wanted data comes from ALUresult.

So it does o_d_addr_w = ALUresult.

3. Other Component :

(1) Control :

It will set Branch, MemRead, MemWrite, MemtoReg, ALUOp, ALUSrc and RegWrite to correct value according to the instruction's opcode.

(2) Register Files :

In the beginning, it will set all registers' value to zero. Afterward, it will output the stored data according to the given register's index and write to destination register if RegWrite is on.

(3) ALUcontrol :

It will output Optype to control ALU's operation. Optype is determined by ALUOp, instruction's 30, 14 to 12 bit.

(4) ALU :

It will do some operation according to given Optype.

If Optype indicates that the instruction is BEQ or BNE, it will set the output signal zero on.

Otherwise, it will do the corresponding operation and output the result.

(5) Mutiplexer :

It will output either data_a or data_b according to input control signal to select the desired output.

(6) Immideate Generator :

It will output the 64 bits imm according to the input instruction.