Zhehua XIang (zhehuax)

IEEE 754 FPU

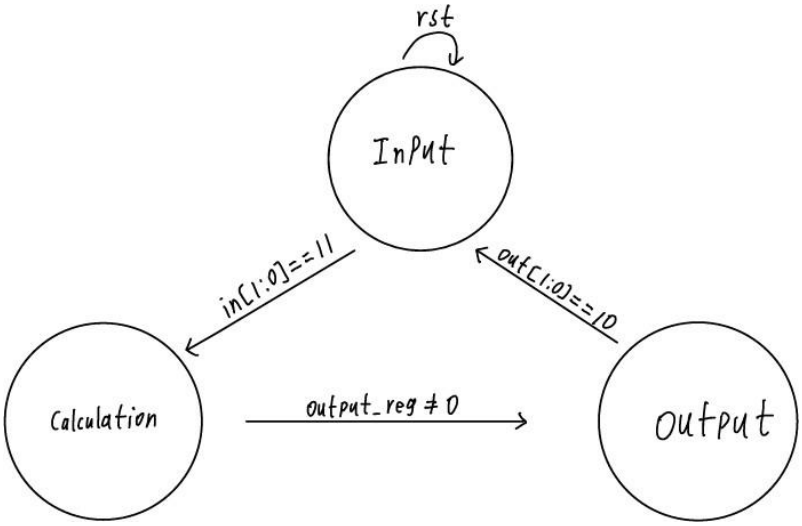
- Summary:
  - The proposed project is a floating-point unit that follows the IEEE 745 standard. It receives two 32-bit floating point numbers and the required operation (addition, subtraction, multiplication, and division if area allows). It takes several clock cycles to collect input data, several clock cycles to compute, and several clock cycles to output results. This FPU does not care about denormalized numbers and treats them as 0.

- I/O:
  - Given the constraint that only 12 IO ports each for input and output (clk and rst have their separate ports), and IEEE 745 floating point numbers have 32 bits, it takes several clock cycles to fully transport data. Protocols for data transfer are discussed below:
  - We use 10 of 12 ports for both input and output
  - Input ports:
    - First 8 bits: input number or operation.
    - Last 2 bits: what is currently being sent (00 means nothing, 01 means number 1, 10 means number 2, 11 means operation)
    - Once the task starts, the first 10 bits that go into the input ports should consist of the first 8 bits being [7:0] of number 1 and the last 2 bits being 01. Then, at the next clock cycle, the first 4 bits should change to [15:8] of number 1, while the last 2 bits remain at 01 (since we're still sending number 1). Thus, data transfer for number 1 should take 4 consecutive cycles.
    - Once the data transfer for number 1 has finished, the lowest 2 bits should no longer be 01, otherwise initial data 1 would be overwritten.
    - Same thing for number 2.
    - Sending operations should only take 1 cycle. 0001 means addition, 0010 means subtraction, 0100 means multiplication, and 1000 means division.
    - Once the FPU receives operation input, it will start to compute with whatever it received for number 1 and number 2, and will not accept any input until finished.
    - Always send input numbers in 4 consecutive cycles.

  - Output ports:
    - First 8 bits: output number.
    - Last 2 bits: 00 means no output available, 01 means currently sending outputs, and 10 means finished.
    - It takes 4 consecutive cycles to output the result, with first cycle being [7:0] of the result number and last cycle being [31:24] of the result.
    - That means 01 will always last for 3 cycles, and 10 will only last for 1 cycle.

- Internal data storage:
    - When receiving data, two 32-bit registers will be used to hold number 1 and number 2.
    - One 32-bit register will be used to hold the output.
    - Other registers will be used when doing calculations to pass data from one stage to another

- High-level FSM:
    - There should be a top-level FSM to separate 3 states: input state, calculation state, and output state.

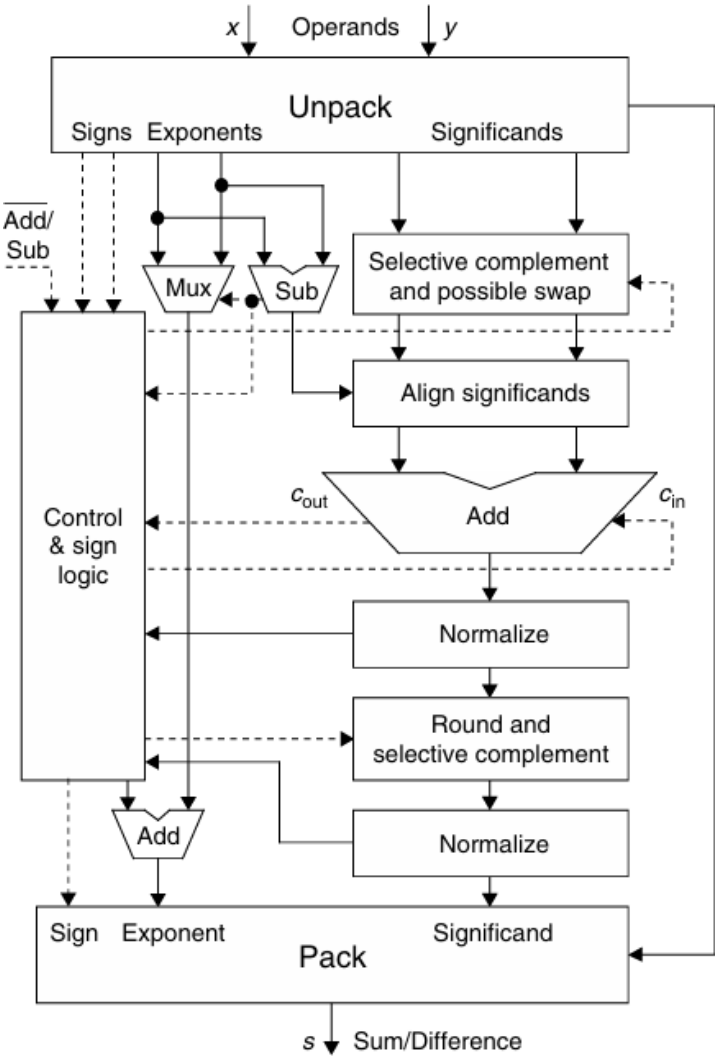


- Datapath for calculation:
    - Before doing any calculation, it is necessary to check for special cases. The table below shows possible special cases:

Operation	Result
NAN + <Anything>	NAN
NAN × <Anything>	NAN
$\infty + \infty$	∞
$\infty \times \infty$	∞
$-\infty + \infty$	NAN
∞ + <Normal>	∞
∞ × <Normal>	∞

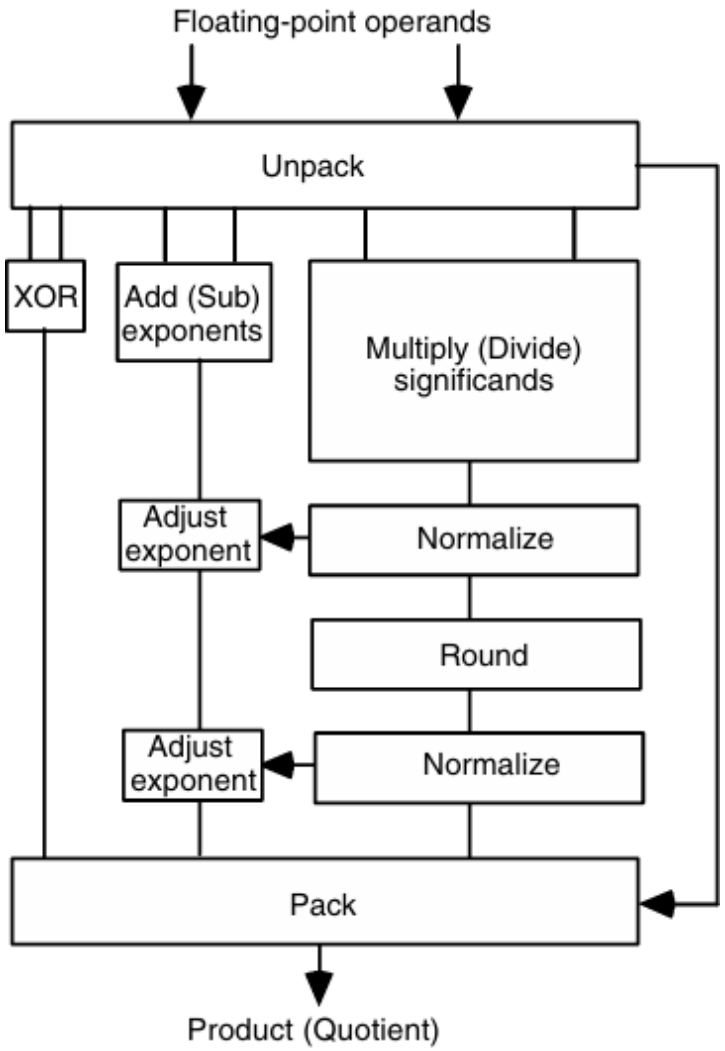
- In this FPU, rounding logic is applied with 2 guard bits and 1 sticky bit.

- Addition/subtraction:
  - There are 5 major steps for addition/subtraction:
    - 1. Reconstruct inputs to match exponents.
    - 2. Add/subtract significants to get a temp result.
    - 3. Do normalization and adjust exponents if necessary.
    - 4. Check if the exponent is valid, then do rounding.
    - 5. After rounding, check for normalization and valid exponent again.
  - **To increase clk frequency, the above 5 steps should be broken into 5 stages, with each stage taking 1 clock cycle. There should be Registers holding temp values for each stage.**
  - A datapath for addition/subtraction is shown below.



Source: Behrooz Parhami

- Multiplication/division
  - There are 5 major steps for multiplication/division:
    - 1. Get real exponents by getting rid of bias.
    - 2. Add/subtract real exponents and multiply/divide significants to get a temp result.
    - 3. Do normalization and adjust exponents if necessary.
    - 4. Check if the exponent is valid, then do rounding.
    - 5. After rounding, check for normalization and valid exponent again, then add the bias back.
  - **To increase clk frequency, the above 5 steps should be broken into 5 stages, with each stage taking 1 clock cycle. There should be Registers holding temp values for each stage.**
  - A datapath for addition/subtraction is shown below.



Source: Behrooz Parhami

- Test plan:
    - I plan to simply use cocoTB with some test vectors and a programming code for FP operation to check if the result match the golden model. There shouldn't be any tricky part since cocoTB could control clock cycles.

- Clock/performance consideration:
    - Not enough information to guess for clock frequency at this point.
    - After dividing the computation into 5 stages, it is reasonable to guess that state 2 for multiplication/division would be the crit path due to the use of large multipliers/divisors. We could in theory further divide this stage, but the area constraint would suffer as more registers are introduced.
    - No clocking gating, no CDC stuff.

- Area consideration:
    - Addition and subtraction share the same piece of hardware; multiplication and division share the same piece of hardware. Also, steps 3,4,5 for all 4 operations are mostly the same, so these pieces could also be shared.
    - If the area constraint is still not met, consider bit truncation for multipliers/divisors to reduce arithmetic logic units and registers.
    - Further area-reducing techniques involve:
        - Simplify rounding logic
        - More truncation (sacrifices precision)
        - Less operations (maybe remove the entire division)

- Reference:
    - Behrooz, Parhami. "Computer arithmetic: Algorithms and hardware designs. 2nd edition"