ALU Component for Single-Cycle Processor

Kyle Clements
April 29, 2025

ALU Function in SCP

Performs core arithmetic and comparison operations

Inputs: 32-bit operands input1 (rs) and input2 (rt or immediate)

Controlled via aluOP:

- 1 = ADD *(ADD, MOVL, MOVS)*
- 0 = CMP *(comparison only)*

Outputs:

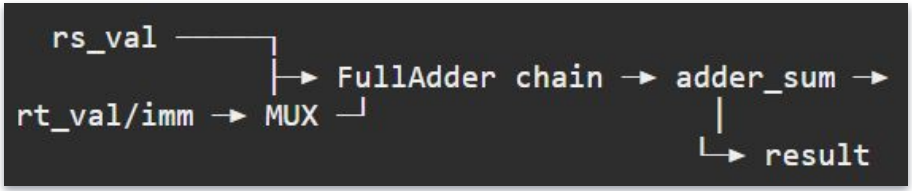
- result *(sum or passthrough)*
- zero flag *(set when input1 == input2)*

Why It’s Important

- ➔ Enables all arithmetic logic
- ➔ Feeds zero flag into the branch logic *(for JE)*
- ➔ Essential for ADD, MOVL, MOVS, CMP instructions

High-Level Architecture

- RippleCarryAdder: 32-bit chain of FullAdders
- ALU MUX chooses between adder output or passthrough
- Zero flag calculated directly from input comparison



Implementation Details

FullAdder.v: XOR and majority logic

RippleCarryAdder.v: 32-bit looped instantiation

ALU.v:

- Uses aluOP to select mode
- Assigns zero flag via comparison

Challenges & Solutions

- ➔ Ensuring zero was captured reliably for JE
- ➔ Ripple delay could be slow—acceptable for this scale
- ➔ Integration with control signals and status register

Pros & Cons

Pros:

- Simple, testable, and reusable
- Works for both data and address logic

Cons:

- Ripple-carry doesn't scale well
- Only supports ADD and CMP *(no logic ops yet)*

Interesting Aspects & Extensions

- ➔ Test instrumentation: '$display' in ALU.v and StatusRegister.v for live trace
- ➔ Easy to extend: add logic ops by augmenting MUX and control signals
- ➔ Serves as template for multi-cycle or pipelined ALU enhancements

Performance

- ALU is invoked in every loop iteration *(ADD + CMP)*
- CMP result controls JE; ALU result updates register
- Observed correct output in both loop-example and halt cases

Educational Impact

- ➔ Built foundational understanding of datapath design
- ➔ Reinforced Verilog syntax and simulation concepts
- ➔ Developed debugging and modular testing skills

Closing Thoughts

- ➔ Simplicity of the ALU made debugging easier
- ➔ Challenges like the zero flag encouraged clean design
- ➔ Ready to extend into more complex operations or architectures