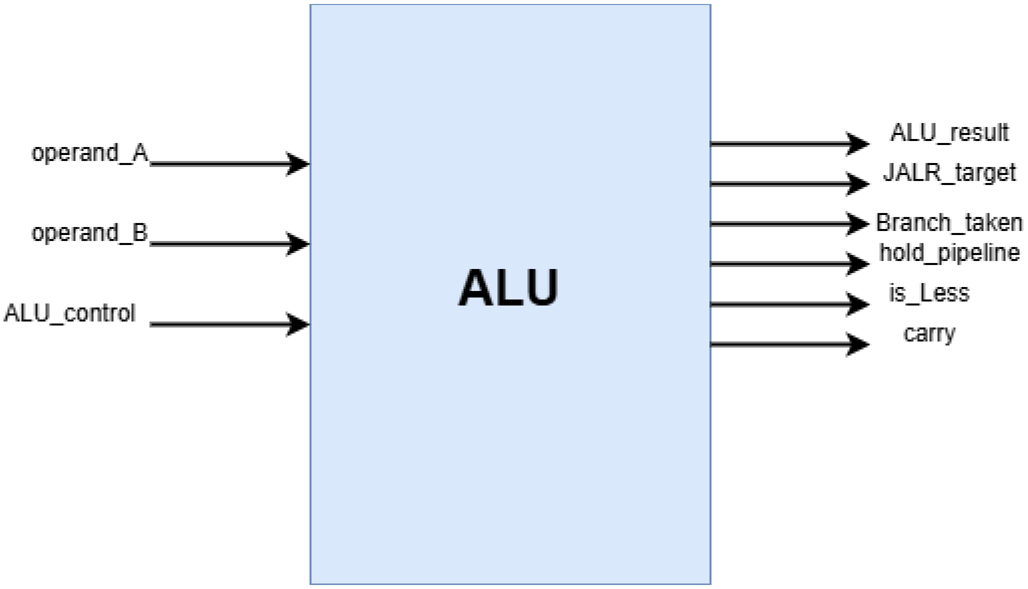
ALU (Arithmetic Logic Unit)

The ALU (Arithmetic Logic Unit) module in a RISC-V processor that supports I (Integer) and M (Multiply) instructions, it’s a total combinational block.

Design and Implementation



Inputs:

- **Alu_control:** Control from controller module, it’s a signal specifies the operation to be performed by the ALU (e.g., addition, subtraction, logical AND, etc.) 6 bits.

e.g. 6'b000000 >>> Add, 6'b000001 >>> SUB... etc.

- **operand_A:** The first operand for the ALU operation (from a register or immediate value) **(**32 bits).
- **operand_B:** The second operand for the ALU operation (from a register or immediate value) (32 bits).

Outputs:

- **Alu_result:** The result of the ALU operation **(**32bit).
- **JALR_target:** signal of the address that pc will fetch, sent to decoder (32 bits).

Flags

- **carry:** The C (Carry out) flag is asserted when the adder produces a carry out and the ALU is performing addition or subtraction (indicated by ALUControl1 = 0) (1 bit).
- **branch_taken:** a (one-bit) signal that raised when conditions for branch are achieved e.g., BEQ, BNQ, BLT...etc.
- **is_less:** when the result of comparator is less, Signal equals 1(one-bit).
- **hold_pipeline:** this signal is high in two conditions**,** first when Branch_taken signal is high and bit [11] of operand_B is zero, the second condition is when JALR instruction is performed which has ALU_control (6'b100111) and operand_A does not equal zero(32’b0) then hold_pipeline is high, else it’s low.

Parameters:

data Width: The bit width of the ALU's operands and result (32 bits).

Supported Operations

ALU module supports various operations based on the RISC-V ISA specifications for I, R, M and J instructions. These operations include:

- Integer arithmetic operations:

- Add (LUI, LW, SW, ADDI, ADD).
- Sub (SUB).

- Logical operations:

- OR (OR, ORI).
- Xor (XORI, XOR).
- And (ANDI, AND).

- Shift operations:

- Logical Shift Left (SLLI, SLL).
- Logical Shift Right (SRLI, SRL).
- Arithmetic Shift Right (SRAI, SRA).

- Comparison operations: equality, inequality, less than, less than or equal to, greater than, greater than or equal to. SLTI, SLT

-Branch operations: BEQ, BNE, BLT, BGE, BLTU.

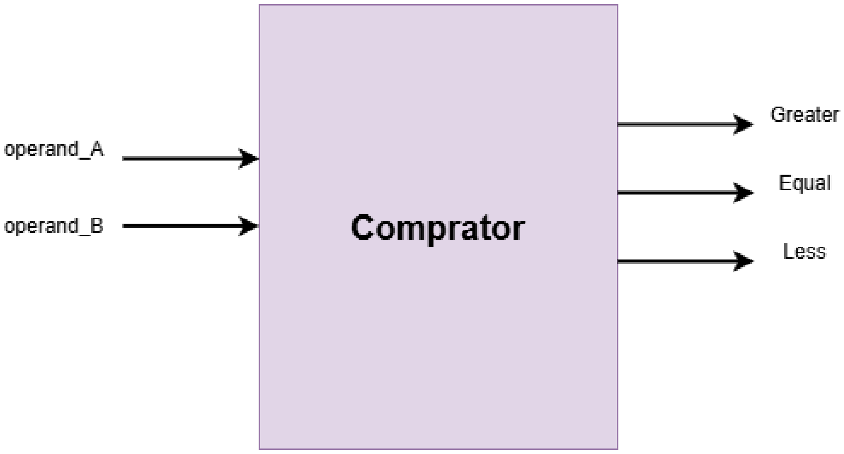
-Jump instruction: jump and link register (JALR).

Submodules

Comparator: It’s A novel comparator module that executes comparison instructions using subtractor, instantiated into ALU module.

Problem statement:

Comparison operations like SLT or BEQ... and so on, requires a huge area then consumes lots of power in the design, as we are aiming to implement a low power RISC-V core, so we implemented a comparator module that does not consume area and therefore does not consume power.



Operation: comparator module subtracts operand_A from operand_B and checks the difference, if the difference result is zero, then two operands are equal, but if the difference result’s most significant bit is 1, then the result is negative, that means operand_A is greater than operand_B, finally if difference is positive which means the most significant bit is zero also it’s not equal, then operand_A is less than operand_B.

Inputs:

- **operand_A:** The first operand for the ALU operation (from a register or immediate value) **(**32 bits).
- **operand_B:** The second operand for the ALU operation (from a register or immediate value) (32 bits).

Outputs:

- **Greater:** This signal is raised when operand_A is greater than operand_B (1 bit).
- **Equal:** raised when two operands are equal (1 bit).
- **Less:** raised when operand_A is less than operand_B (1 bit).

Table of ALU signals

Signal	**Width**	**Direction**	**Interface**
Operand_A	32bit	input	GPR
Operand_B	32bit	input	GPR
ALU_control	6bit	input	Controller
ALU_result	32bit	output	GPR/D-Memory
JALR_target	32bit	output	Decoder
Branch_taken	1bit	output	Decoder
hold_pipeline	1bit	output	Controller
Is_Less	1bit	output	GPR
Carry	1bit	output	GPR

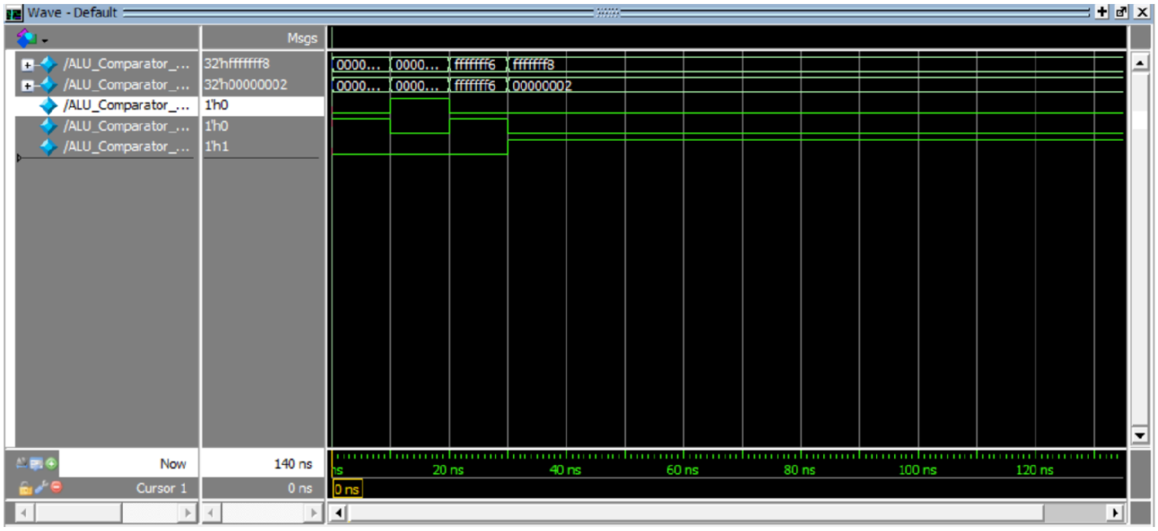
Table of Comparator Signals

Signal	width	Direction
Operand_A	32bit	input
Operand_B	32bit	input
Greater	1bit	output
Equal	1bit	output
Less	1bit	output

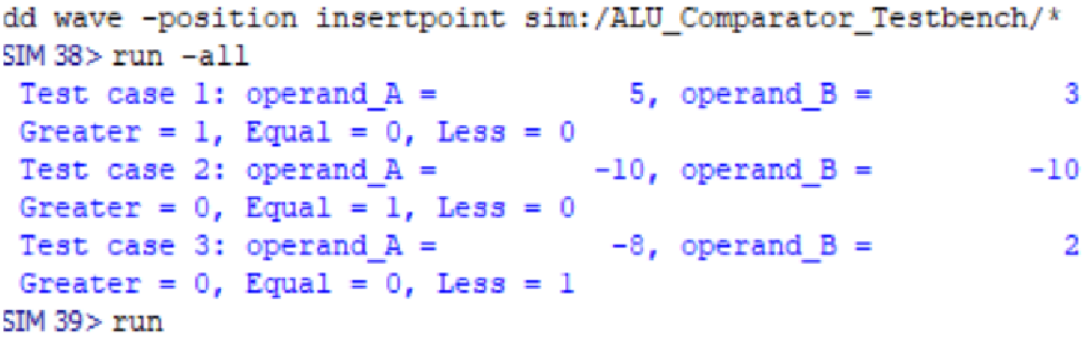
Table of ALU_control signals

ALU_control	Operation
6'b000000	Add (LW, SW, ADDI, ADD)
6'b001000	Sub (SUB)
6'b000110	Or (OR, ORI)
6'b000100	Xor (XORI, XOR)
6'b000111	And (ANDI, AND)
6'b000001	Logical Shift Left (SLLI, SLL)
6'b000101	Logical Shift Right (SRLI, SRL)
6'b001101	Arithmetic Shift Right (SRAI, SRA)
6'b000010	Signed Less Than (SLTI, SLT)
6'b010000	BEQ
6'b010001	BNE
6'b000010	BLT
6'b010101	BGE
6'b010110	BLTU
6'b010111	BGEU
6'b100111	JALR

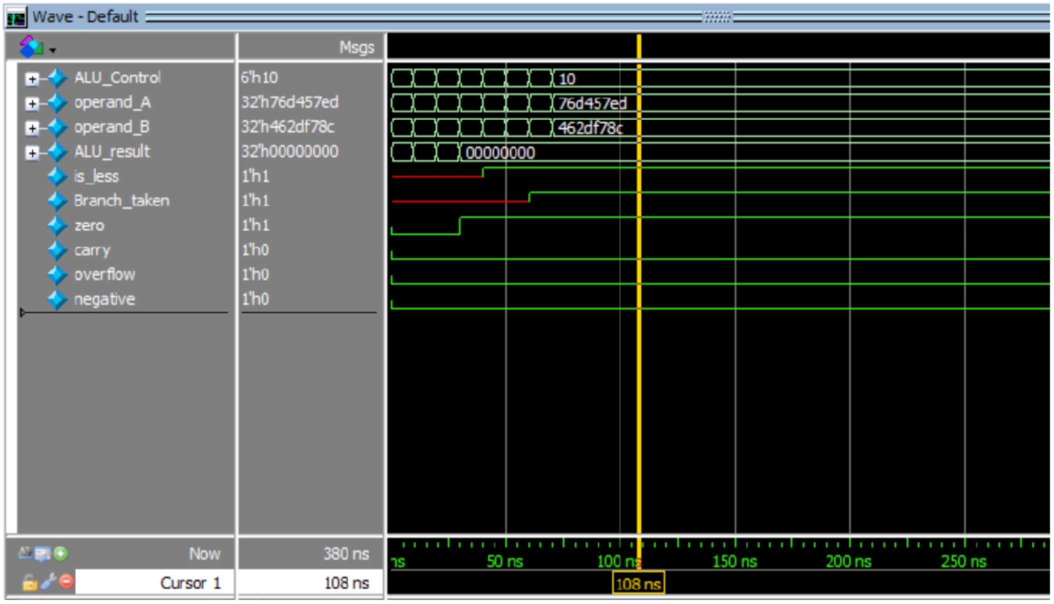
Testbench wave form and monitor results



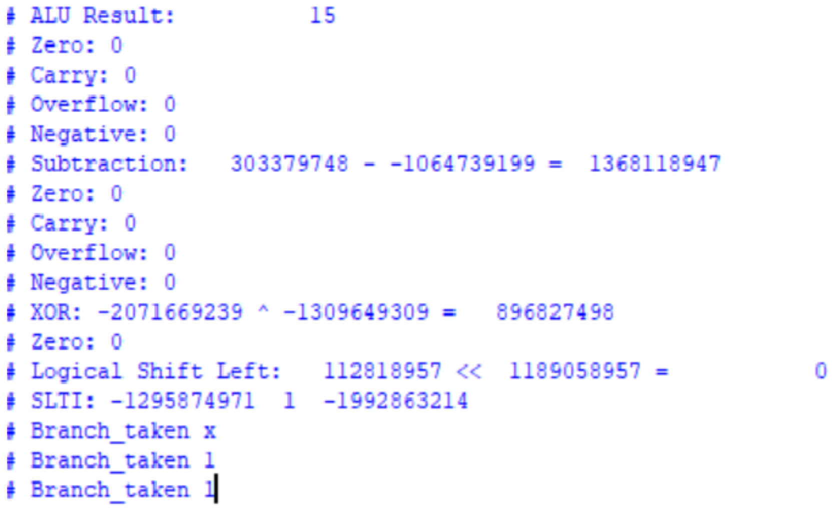
```
/ALU_Comparator_...   32'hfffffff8
/ALU_Comparator_...   32'h00000002
/ALU_Comparator_...   1'h0
/ALU_Comparator_...   1'h0
/ALU_Comparator_...   1'h1
Now                   140 ns
Cursor 1              0 ns
```



```
dd wave -position insertpoint sim:/ALU_Comparator_Testbench/*
SIM 38> run -all
 Test case 1: operand_A =           5, operand_B =           3
 Greater = 1, Equal = 0, Less = 0
 Test case 2: operand_A =         -10, operand_B =         -10
 Greater = 0, Equal = 1, Less = 0
 Test case 3: operand_A =          -8, operand_B =           2
 Greater = 0, Equal = 0, Less = 1
SIM 39> run
```



```
ALU_Control    6'h10
operand_A      32'h76d457ed
operand_B      32'h462df78c
ALU_result     32'h00000000
is_less        1'h1
Branch_taken   1'h1
zero           1'h1
carry          1'h0
overflow       1'h0
negative       1'h0
Now            380 ns
Cursor 1       108 ns
```



```
# ALU Result:          15
# Zero: 0
# Carry: 0
# Overflow: 0
# Negative: 0
# Subtraction:   303379748 - -1064739199 =  1368118947
# Zero: 0
# Carry: 0
# Overflow: 0
# Negative: 0
# XOR: -2071669239 ^ -1309649309 =   896827498
# Zero: 0
# Logical Shift Left:   112818957 <<  1189058957 =           0
# SLTI: -1295874971  1  -1992863214
# Branch_taken x
# Branch_taken 1
# Branch_taken 1
```