Improvement Requirements

⬅الهدف هوا اننا نزود عدد العمليات اللي ممكن الـ **ALU** انه يعملها , ودا هيتطلب تحسينات في الـ **ALU** و الـ **ALU Decoder** و الـ **Main decoder** .

ALU :

1- الـ ALU كان بيعمل مجموعة من العمليات زودنا عليهم الـ sll/slli و الـ srl/srli والـ bne :-

ALUControl	Operation	Description	Flags
`3'b000`	Addition (`ADD`)	`ALUResult = SrcA + SrcB` .	None
`3'b001`	Subtraction (`SUB`)	`ALUResult = SrcA - SrcB` . Updates `Zero` .	`Zero` , `NotZero`
`3'b010`	Bitwise AND (`AND`)	`ALUResult = SrcA & SrcB` .	None
`3'b011`	Bitwise OR (`OR`)	`ALUResult = SrcA	SrcB` .	None
`3'b100`	Bitwise XOR (`XOR`)	`ALUResult = SrcA ^ SrcB` .	None
`3'b101`	Set Less Than (`SLT`)	`ALUResult = (SrcA < SrcB)` .	None
`3'b110`	Shift Left Logical (`SLL`)	`ALUResult = SrcA << SrcB[4:0]` .	None
`3'b111`	Shift Right Logical (`SRL`)	`ALUResult = SrcA >> SrcB[4:0]` .	None

2- RTL Code :-

```
`timescale 1ns / 1ps

module ALU (SrcA, SrcB, ALUControl, ALUResult, Zero);
    input signed [31:0] SrcA;          // First operand
    input signed [31:0] SrcB;          // Second operand
    input [2:0] ALUControl;            // (from ALU Decoder)
    output reg signed [31:0] ALUResult;
    output reg Zero;                   // Zero flag: 1 if ALUResult is zero

    always @(*) begin
        // Default values t
        Zero = 0;

        case (ALUControl)
            3'b000:  // ADD
            begin
                ALUResult = SrcA + SrcB;
            end

            3'b001:  // SUB
            begin
                ALUResult = SrcA - SrcB;
                Zero = (ALUResult == 32'b0); // if Zero = 1 >> beq , else >> bne
            end

            3'b010:  // AND
            begin
                ALUResult = SrcA & SrcB;
            end

            3'b011:  // OR
            begin
                ALUResult = SrcA | SrcB;
            end

            3'b100:  // XOR
            begin
                ALUResult = SrcA ^ SrcB;
            end

            3'b101:  // SLT (Set Less Than)
            begin
                ALUResult = (SrcA < SrcB) ? 1 : 0;
            end

            3'b110:  // Shift Left Logical (SLL)
            begin
                ALUResult = SrcA << SrcB[4:0];
            end

            3'b111:  // Shift Right Logical (SRL)
            begin
                ALUResult = SrcA >> SrcB[4:0];
            end

            default:
            begin
                ALUResult = 32'bx;    // Unknown operation
            end
        endcase
    end
endmodule
```

ALU decoder :

1- لازم الـ ALU decoder يكون يقدر يبعت الـ ALUControl signal المناسبة للعمليات الجديدة :-

ALUOp	funct3	funct7b5 & opb5 (RtypeSub)	ALU Operation	ALUControl
00	X	X	Addition (for lw/sw)	000
01	X	X	Subtraction (for beq/bne)	001
10 (R/I-type)	000	0 (Add/Addi)	Addition	000
10 (R/I-type)	000	1 (Sub)	Subtraction	001
10 (R/I-type)	001	X	Shift Left Logical (sll/slli)	110
10 (R/I-type)	010	X	Set Less Than (slt/slti)	101
10 (R/I-type)	100	X	XOR (xor/xori)	100
10 (R/I-type)	101	X	Shift Right Logical (srl/srli)	111
10 (R/I-type)	110	X	OR (or/ori)	011
10 (R/I-type)	111	X	AND (and/andi)	010
Default	X	X	Undefined operation	xxx

2- RTL Code :-

```
`timescale 1ns / 1ps

module ALU_Decoder (opb5, funct3, funct7b5, ALUOp, ALUControl);
    input wire opb5;
    input wire [2:0] funct3;
    input wire funct7b5;
    input wire [1:0] ALUOp;
    output reg [2:0] ALUControl;    // ALU control signals


    wire RtypeSub;
    assign RtypeSub = funct7b5 & opb5; // If TRUE (R-type subtract)

    always @(*) begin
        case (ALUOp)
            2'b00:
                ALUControl = 3'b000; // Addition (for lw/sw)
            2'b01:
                ALUControl = 3'b001; // Subtraction (for branch equal / branch not equal)
            default:
            begin
                case (funct3)         // R-type or I-type operations
                    3'b000:
                        ALUControl = (RtypeSub) ? 3'b001 : 3'b000; // Sub or Add/Addi
                    3'b001:
                        ALUControl = 3'b110;     // sll/slli
                    3'b010:
                        ALUControl = 3'b101;     // slt/slti
                    3'b100:
                        ALUControl = 3'b100;     // xor/xori
                    3'b101:
                        ALUControl = 3'b111;     // srl/srli
                    3'b110:
                        ALUControl = 3'b011;     // or/ori
                    3'b111:
                        ALUControl = 3'b010;     // and/andi

                    default:
                        ALUControl = 3'bxxx;      // Undefined operation
                endcase
            end
        endcase
    end
endmodule
```

Main Decoder :

1- مما سبق هنلاحظ ان ام الـ **sll/slli – srl/srli** طلعوا بسهولة مباشرة من الـ **ALU Decoder** ودخلوا علي الـ **ALU** وكله تمام.

2- ولكن الفكرة كلها في الـ **beq** و الـ **bne** اتعملوا ازاي ومحتاجين ايه لسه من الـ**Main Decoder**...؟

- هنلاقي ان المعادلة **Zero = (ALUResult == 32'b0)** (الـ **ALUResult** اللي هيا بتساوي **A – B**):-

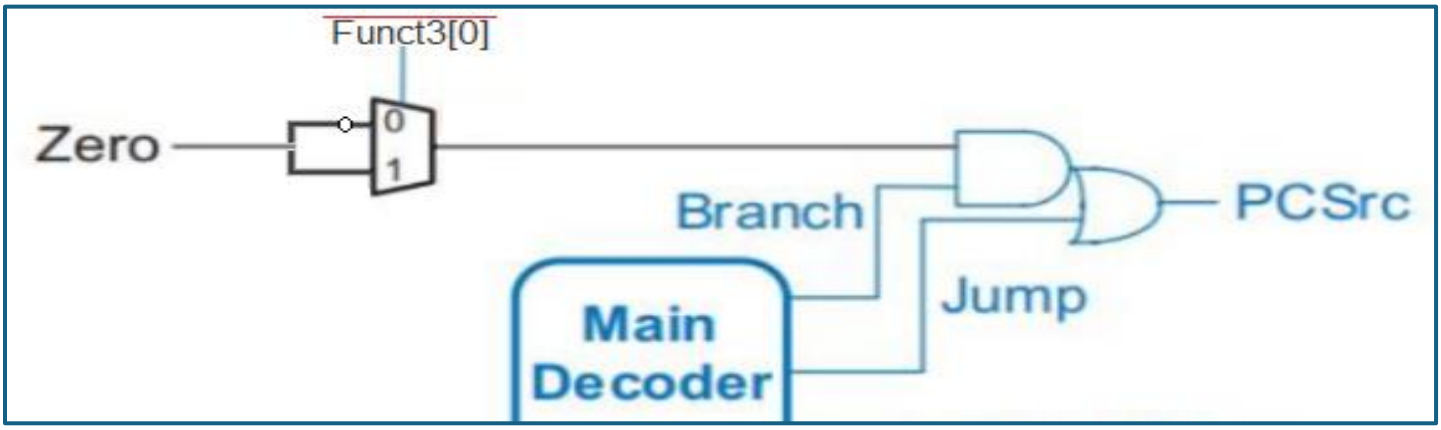
-- لو الـ **ALUResult** ساوت الصفر فالـ **Zero** هتكون بواحد **(Zero = 1)**, يعني لو متساوين **(Zero = 1)**.

-- لو الـ **ALU Result** ساوت حاجه غير الصفر فالـ **Zero** هتكون بصفر**(Zero = 0)**, يعني لو مختلفين **(Zero = 0)**.

- اذا انا همشي في الـ **beq** بنفس طريقة الـ **bne** بمعني اني هعمل **sub** عشان كدا مفرقتش مبينهم في الـ **ALU Decoder**.

3- ولكن ازاي هنفرق ما بينهم ...؟

- دي هتكون مهمة الـ **Main decoder** و الـ **Function 3** :-

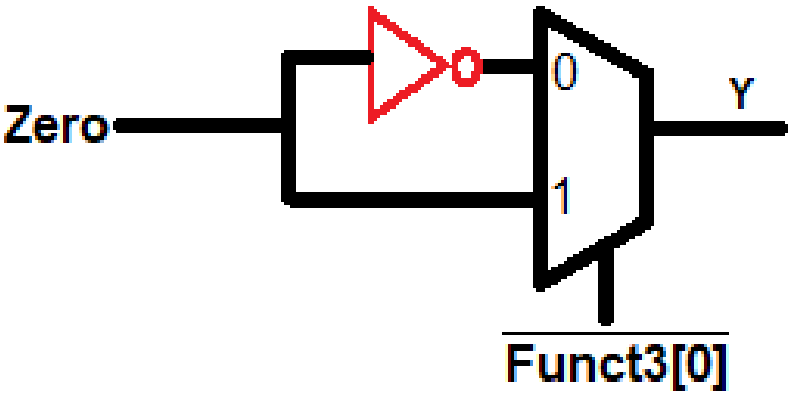


beq	1100111	000
bne	1100111	001

Branch if	Zero	NotZero	Selector	out
Equal (✓)	1	0	1	1
Equal (×)	0	1	1	0
NOT Equal (✓)	0	1	0	1
NOT Equal (×)	1	0	0	0

⬅الـ **Main decoder** عليه ايضا توفير الـ **ALUOp** المناسبة للعمليات المطلوبة.

⬅ ممكن نشيل الـ **Mux** ونحط **XOR Gate** :-



Funct3[0]	$\overline{\text{Funct3[0]}}$	Zero	Y
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0

Made by: Hossam Ahmed Seyam