ALU Decoder Block

1- الهدف هوا اننا ناخد اجزاء من الـ **instruction** ونحدد علي اساسها العملية اللي هتتعمل في الـ **ALU** , يعني هوا عبارة عن اننا هنعمل دائرة اللي هتعمل **Control** للـ **ALU**.

2- المفروض انن الـ **ALU decoder** ياخد اجزاء من الـ **instruction** ويطلع **ALUControl signal** بالشكل دا :

Table 7.3 ALU Decoder truth table

ALUOp	funct3	$\{op_5, funct7_5\}$	ALUControl	Instruction
00	x	x	000 (add)	lw, sw
01	x	x	001 (subtract)	beq
10, 11	000	00, 01, 10	000 (add)	add
	000	11	001 (subtract)	sub
	010	x	101 (set less than)	slt
	110	x	011 (or)	or
	111	x	010 (and)	and
	100	x	100 (xor)	xor

3- الـ **ALU Decoder Block** بيكون ليها كذا دخل وخرج واحد :-

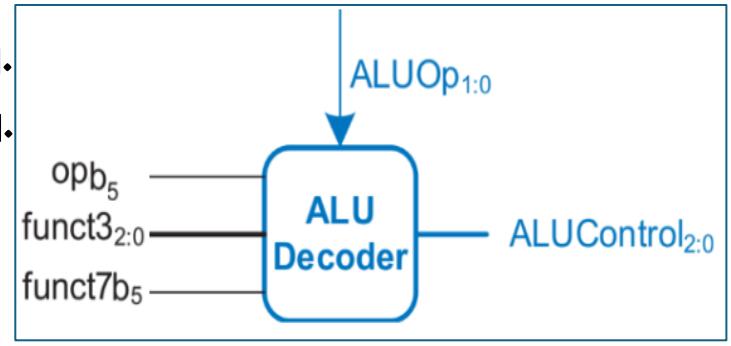
Inputs :

1- **ALUOp** : Determines operation type [2 bits].

2- **funct3** : Specifies detailed operation[3 bits].

3- **opb5** : The fifth bit of the opcode [1 bit].

4- **funct7b5** : The fifth bit of funct7[1 bit].



Output :

➔ **ALUControl** : Control signal for ALU operation selection [3 bits].

📖 الـ **Funct7b5** والـ **opb5** مهمين جدا لانهم مع بعض بيحددوا سواء عمليتي الـ **add/addi** و الـ **sub** في الـ **instructions** ما عدا الـ **lw, sw, beq** كما في الجدول السابق, وهذا لان الـ **lw, sw, beq** بيتم تحديدهم مباشرة من الـ**opcode** بتاعم ومش محتاجين **additional information** من الـ **funct3**أو الـ **funct7** لأن العمليات الخاصة بيهم محددة وواضحة باستخدام الـ **ALUOp** فقط .

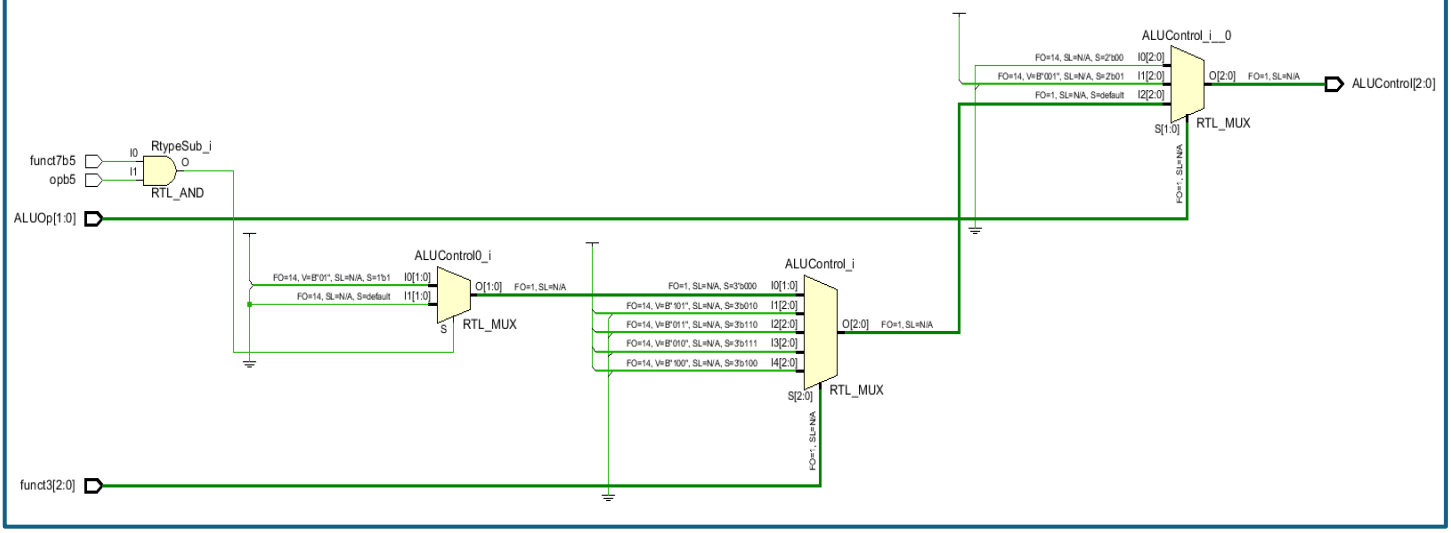
4- The RTL Code :-

```
1  `timescale 1ns / 1ps
2
3  module ALU_Decoder (opb5, funct3, funct7b5, ALUOp, ALUControl);
4      input wire opb5;
5      input wire [2:0] funct3;
6      input wire funct7b5;
7      input wire [1:0] ALUOp;
8      output reg [2:0] ALUControl;   // ALU control signals
9
10
11     wire RtypeSub;
12     assign RtypeSub = funct7b5 & opb5; // If TRUE (R-type subtract)
13
14     always @(*) begin
15         case (ALUOp)
16             2'b00:
17                 ALUControl = 3'b000; // Addition (for lw/sw)
18             2'b01:
19                 ALUControl = 3'b001; // Subtraction (for branch equal)
20             default:
21             begin
22                 case (funct3)        // R-type or I-type operations
23                     3'b000:
24                         ALUControl = (RtypeSub) ? 3'b001 : 3'b000; // Sub or Add/Addi
25                     3'b010:
26                         ALUControl = 3'b101;    //slt/slti
27                     3'b110:
28                         ALUControl = 3'b011;    //(or/ori)
29                     3'b111:
30                         ALUControl = 3'b010;    //(and/andi)
31                     3'b100:
32                         ALUControl = 3'b100;    // (xor)
33                     default:
34                         ALUControl = 3'bxxx;     // Undefined operation
35                 endcase
36             end
37
38         endcase
39     end
40 endmodule
```

لازم الـ**Funct7b5** والـ **opb5** يكونوا الاتنين بـ**1** عشان نعمل عملية الـ **subtract** غير كدا اللي هيحصل **addition**.

التسلسل بتاعنا هوا : ALUOp ⊂ funct3 ⊂ (opb5 & funct7b5) = (RtypeSub).

- Elaborated design:-



5- Testbench :-

```
`timescale 1ns / 1ps

module ALU_Decoder_tb();

    // 1) Declare local reg and wire identifiers with _tb suffix
    reg  opb5_tb;
    reg  [2:0] funct3_tb;
    reg  funct7b5_tb;
    reg  [1:0] ALUOp_tb;
    wire [2:0] ALUControl_tb;

    // 2) Instantiate the ALU module under test
    ALU_Decoder uut (
        .opb5(opb5_tb),
        .funct3(funct3_tb),
        .funct7b5(funct7b5_tb),
        .ALUOp(ALUOp_tb),
        .ALUControl(ALUControl_tb)
    );

    // 3) Generate stimuli using initial and always
    initial
    begin
        //for (lw / sw), output should be ALUControl = 000
        opb5_tb = 1'b1;  funct7b5_tb = 1'b0;  ALUOp_tb = 2'b00;  funct3_tb = 3'b000; #50;
        opb5_tb = 1'b1;  funct7b5_tb = 1'b1;  ALUOp_tb = 2'b00;  funct3_tb = 3'b000; #50;

        //for ( branch if equal ), output should be ALUControl = 001
        opb5_tb = 1'b0;  funct7b5_tb = 1'b0;  ALUOp_tb = 2'b01;  funct3_tb = 3'b000; #50;
        opb5_tb = 1'b0;  funct7b5_tb = 1'b1;  ALUOp_tb = 2'b01;  funct3_tb = 3'b000; #50;

        //for ( add / addi ), output should be ALUControl = 000
        opb5_tb = 1'b1;  funct7b5_tb = 1'b0;  ALUOp_tb = 2'b10;  funct3_tb = 3'b000; #50;

        //for ( sub ), output should be ALUControl = 001
        opb5_tb = 1'b1;  funct7b5_tb = 1'b1;  ALUOp_tb = 2'b11;  funct3_tb = 3'b000; #50;

        //for Logic operations
        opb5_tb = 1'b0;  funct7b5_tb = 1'b0;  ALUOp_tb = 2'b10;  funct3_tb = 3'b010; #50;    // for ( slt / slti ), output should be 101
        opb5_tb = 1'b1;  funct7b5_tb = 1'b0;  ALUOp_tb = 2'b11;  funct3_tb = 3'b110; #50;    // for ( or ), output  should be 011
        opb5_tb = 1'b0;  funct7b5_tb = 1'b1;  ALUOp_tb = 2'b10;  funct3_tb = 3'b111; #50;    // for ( and ), output should be 010
        opb5_tb = 1'b1;  funct7b5_tb = 1'b1;  ALUOp_tb = 2'b11;  funct3_tb = 3'b100; #50;    // for ( xor ), output should be 100

        //for default , output should be ALUControl = xxx
        opb5_tb = 1'b1;  funct7b5_tb = 1'b1;  ALUOp_tb = 2'b10;  funct3_tb = 3'b101; #50;
        opb5_tb = 1'bx;  funct7b5_tb = 1'bx;  ALUOp_tb = 2'b11;  funct3_tb = 3'b011; #50;

    // 4) Specify a stopwatch to stop the simulation
    #600 $stop;
    end
endmodule
```

←نتيجة الـ **Simulation** :

Made by: Hossam Ahmed Seyam