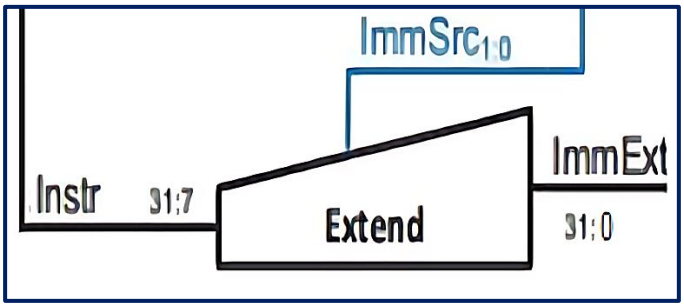
Extended Unit Block

1- الفكرة من استخدام الـ **Extended Unit** انه في الهاردوير لازم لازم الرقمين اللي هنجمعهم او نطرحهم يكونوا بنفس الـ **Length** يعني عندنا لازم الرقمين يكونوا **32 bits** .

2- وعشان كد بنستخدم الـ **Extended Unit** لما يكون عندنا رقم اقل من رقم تاني عشان نساويهم مع بعض عن طريق الـ **Signed extension** وهوا تكرير الـ **MSB** حتي الـ **Length** اللي محتاجينه

3- الـ **Extended Unit Block** بيكون ليها دخلين وخرج واحد :-



Inputs :

1- **instr** : Instruction [31 : 7] (25 bits).

2- **immsrc** : Immediate source<Control> (2 bits).

Output :

➔ **immext** : immediate extended (32 bits).

📖 الـ **instruction** بيكون **32 bits** في كل انواعة وكل الانواع بتبدا دائما بـ **7 bits** بتوع الـ **opcode** ودا احنا مش محتاجينه وعشان كدا بندخل **25 bits** بس , من ضمن الـ **25** دول في حجات تانيه مش بناخدها زي الـ **rd** والـ **rs1** والـ **rs2** والـ **functions** بانواعها لان كل اللي هاممنا هوا الـ **immediate** فقط ولكن الـ **immediate** مكانه بيتغير في انواع الـ **instruction** المختلفه عشان كدا وعشان نخلي التصميم ابسط في الـ **RISC-V** بندخل كل الـ **25 bits** وبنختار اللي احنا عاوزينه جوا الـ **Extended Unit Block** .

3- The RTL Code :-

```
1   `timescale 1ns / 1ps
2
3   module Extend_Unit(instr, immsrc, immext);
4
5       input wire [31:7] instr;          // instruction part
6       input wire [1:0] immsrc;          // immediate source selector
7       output reg [31:0] immext;         // extended immediate output
8
9       always @(*)
10      begin
11          case(immsrc)
12              // I-type
13              2'b00: immext = {{20{instr[31]}}, instr[31:20]};
14
15              // S-type (stores)
16              2'b01: immext = {{20{instr[31]}}, instr[31:25], instr[11:7]};
17
18              // B-type (branches)
19              2'b10: immext = {{20{instr[31]}}, instr[7], instr[30:25], instr[11:8], 1'b0};
20
21              // J-type (jal)
22              2'b11: immext = {{12{instr[31]}}, instr[19:12], instr[20], instr[30:21], 1'b0};
23
24              default: immext = 32'bx;  // undefined
25          endcase
26      end
27  endmodule
```

⬅في نقطتين مهمين :-

الاولي : احنا بنشوف اشارة الـ **control** بكام وعلي هذا الاساس بنختارايه الطريقة اللي همجمع بيها الـ **Bits** اللي بعد كدا هنعملها **sign extension** عشان تطلع في الاخر بالـ **length** اللي احنا عاوزينه **(32 bits)**.

الثانية : كل **instruction** ليه طريقة في التجميع مختلفة عن غيرة , الغرض االنهائي ليا اني اجمع الـ **immediate** بشكل مظبوط عشان اعملة **extension** بعدها .

⇦ وعشان نعمل كدا محتاجين نستخدم **2 operators** من الـ **Verilog** وهما :-

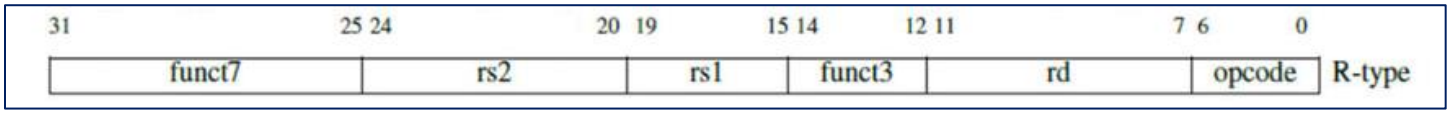
Category	Examples	Bit Length
Concatenate	{A,...,B}	L(A) + ··· + L(B)
Replication	{B{A}}	B * L(A)

1- الـ **Concatenation** : وهوا بيضم **A bits** علي **B bits**.

2-الـ **Replication** : وهوا تكرار الـ **A** عدد **B** من المرات .

⇨R-Type instruction :-

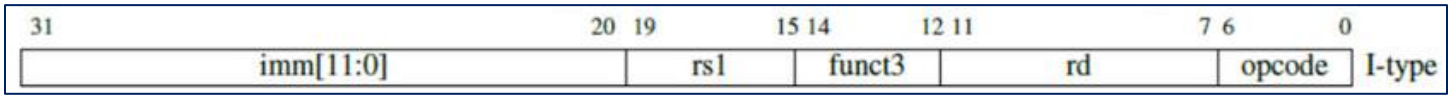
← دا توزيع الـ **32 bits** اللي موجدين في الـ**Register instruction** :



← لا يحتاج لـ **extended unit**, لانه لا يوجد فيه **immediate** ولان الـ **Sources** اللي فيه بتكون متخزنه بشكل اساسي **32 bits**.

⇨I-Type instruction :-

← دا توزيع الـ **32 bits** اللي موجدين في الـ**Immediate instruction** :



← يوجد هنا **immediate** بحجم **12 bits** , وهذا الـ **immediate** سيتم جمعه مع الـ **base address** اللذي يتكون من **32 bits** حتي نحصل علي الـ**Actual memory address** , ولذلك يجب عمل **signed extension**.

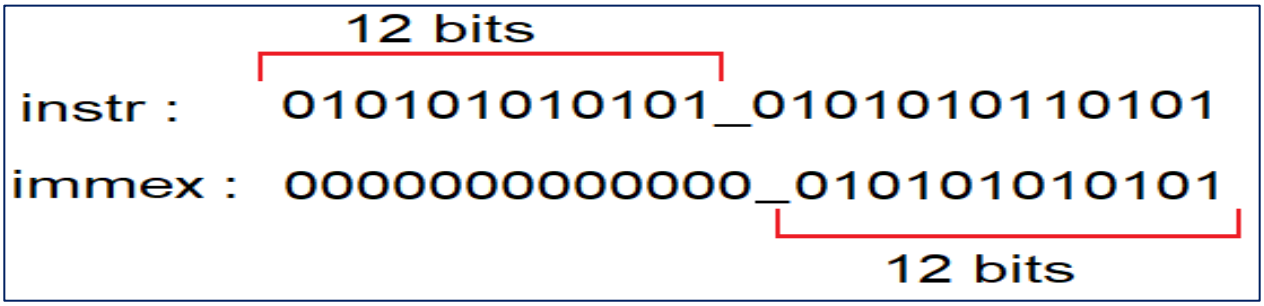
← وهنا الـ **immediate** في الـ **I-instruction** في الـ**RISC-V** مرتب علي ان يكون الـ **12 bits** مرتبين ورا بعض من **[31 : 20]**.

--ولذلك كل اللي عملنا اننا اخدنا الـ **12 bits** ثم اضفنا لهم (extension) **20 bits** من الـ **MSB** (**signed extension**) فاصبح الـ **immediate** الان هوا **12 + 20 = 32 bits**.

--وعشان نعمل كدا هناخد الـ **11 bits [0 : 10]** و نضمهم (**Concatenation**) علي الـ **bit** الـ **12** بعد منكرره (**Replication**) **20** مرة.

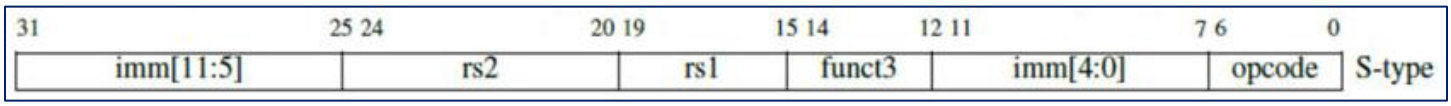
```
// I-type
immext = {{20{instr[31]}}, instr[31:20]};
```

EX:



⇨S-Type instruction :-

← دا توزيع الـ **32 bits** اللي موجدين في الـ**Store instruction** :



← يوجد هنا **immediate** بـحجم **12 bits** , ولكن هذا الـ **immediate** متفرق في الـ **instruction** وعشان كدا هنجتاج نعمل حاجتين مهمين

اولا : نجمعة علي بعضه بحيث يكون **12 bits** كاملين , ودا هيتم عن طريق الـ**concatenation** .

ثانيا : نعمل **signed extension** عن طريق الـ**Replication** ثم نعمل **concatenation** لهم .

```
// S-type
immext = {{20{instr[31]}}, instr[31:25], instr[11:7]};
```

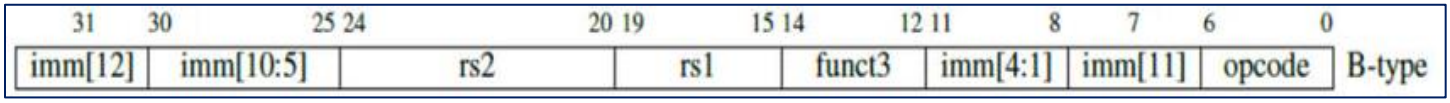
EX:

instr : 10110011100011110000011111

immex 11111111111111111111101100111111

⇨B-Type instruction :-

← دا توزيع الـ **32 bits** اللي موجدين في **Branch instruction** :



-- هنلاحظ : انهم مش مرتبين بشكل منظم كما سبق ولكن مفيش مشاكل هنظبطهم بالـ **Concatenation** .

-- هنلاحظ : ان انا عندي **imm[12 : 1]** ولكن لا يوجد **imm[0]** ودا لاني انا اللي هضيفها ولازم تكون بـ **0** وبكدا يكون الـ **imm** هنا **13 bit** عشان تبقي **word alignment**.

```
// B-type (branches)
immext = {{20{instr[31]}}, instr[7], instr[30:25], instr[11:8], 1'b0};
```

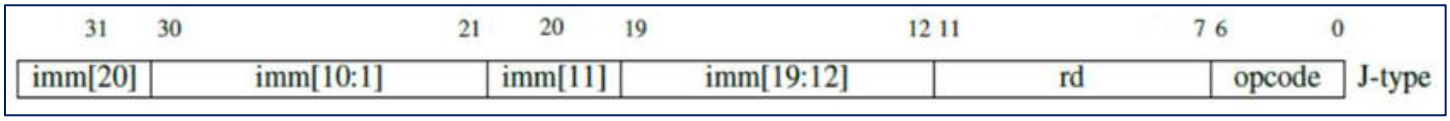
EX:

instr : 10110011100011110000011111

immex: 11111111111111111111101100111110

⇨J-Type instruction :-

← دا توزيع الـ **32 bits** اللي موجدين في **Jump instruction** :



-- هنلاحظ : انهم مش مرتبين بشكل منظم كما سبق ولكن مفيش مشاكل هنظبطهم بالـ **Concatenation** .

-- هنلاحظ : ان انا عندي **imm[12 : 19]** و **imm[11]** و **imm[1 : 10]** و **imm[10]** ولكن لا يوجد **imm[0]** ودا لاني انا اللي هضيفها ولازم تكون بـ **0** وبكدا يكون الـ **imm** هنا **21 bit** عشان تبقي **word alignment**.

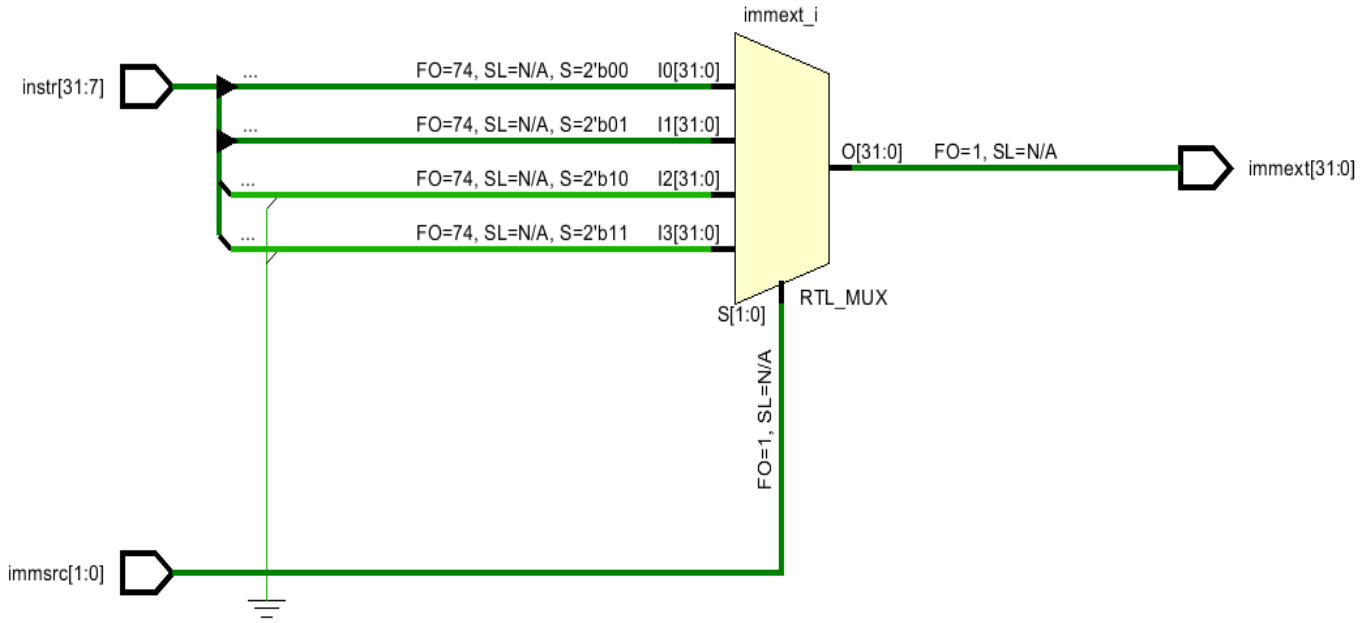
```
// J-type (jal)
immext = {{12{instr[31]}}, instr[19:12], instr[20], instr[30:21], 1'b0};
```

EX:

instr : 01100111000111100001111111

immex: 00000000000111000011110011100000

4- Elaborated design:-

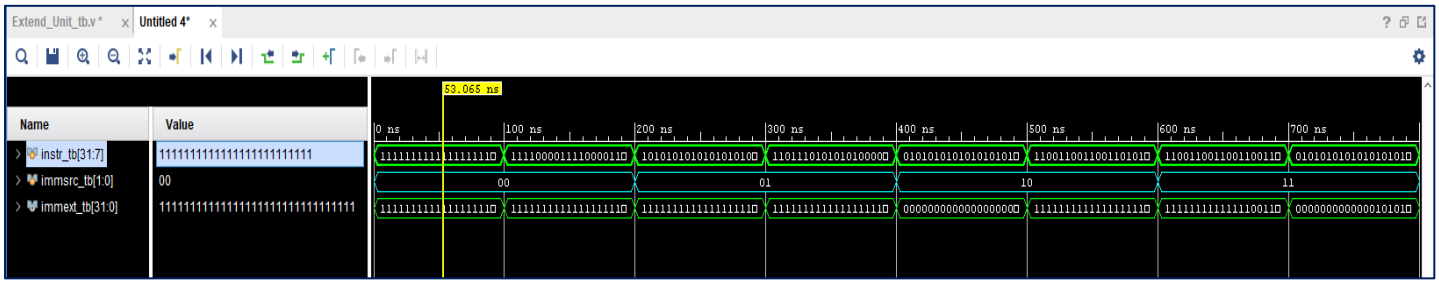


5- Testbench :-

```
`timescale 1ns / 1ps

module Extend_Unit_tb();

    // 1) Declare local reg and wire identifiers
    reg  [31:7] instr_tb;
    reg  [1:0] immsrc_tb;
    wire [31:0] immext_tb;

    // 2) Instantiate the module under test
    Extend_Unit uut (
            .instr(instr_tb),
            .immsrc(immsrc_tb),
            .immext(immext_tb)
     );

    // 3) Generate stimuli using initial and always
    initial
     begin
        // Stimulus 1: I-type instruction
        instr_tb = 25'b111111111111_1111111111111;   immsrc_tb = 2'b00;  #100;
        instr_tb = 25'b111100001111_0000111000110;   immsrc_tb = 2'b00;  #100;

        // Stimulus 2: S-type instruction
        instr_tb = 25'b1010101_01010101010101_0101;   immsrc_tb = 2'b01;  #100;
        instr_tb = 25'b1101110_10101010000101_0100;   immsrc_tb = 2'b01;  #100;

        // Stimulus 3: B-type instruction
        instr_tb = 25'b0_101010_101010101010_1010_1_0;   immsrc_tb = 2'b10;  #100;
        instr_tb = 25'b1_100110_011001101010_0010_1_0;   immsrc_tb = 2'b10;  #100;

        // Stimulus 4: J-type instruction
        instr_tb = 25'b1_1001100110_0_11001100_11001;   immsrc_tb = 2'b11;  #100;
        instr_tb = 25'b0_1010101010_1_01010101_01010;   immsrc_tb = 2'b11;  #100;

        $stop;
    end
endmodule
```

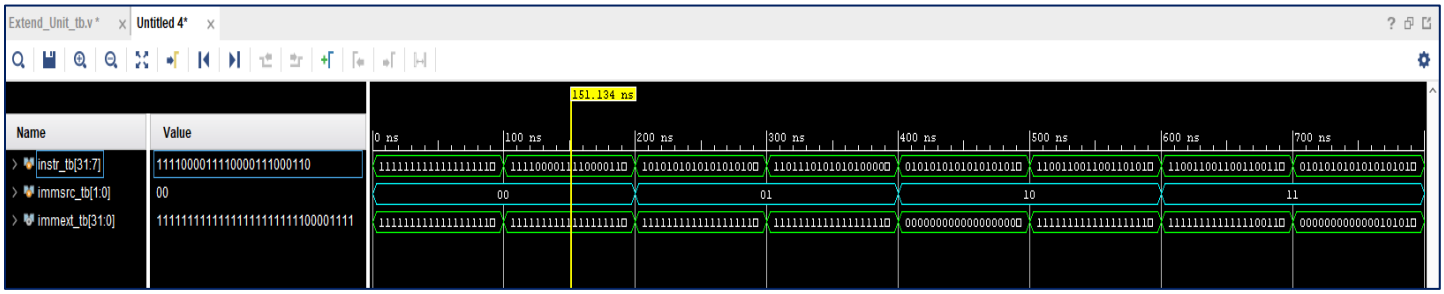
←نتيجة الـ **Simulation** :

⇨I-Type instruction :-



instr : 11111111111111111111111111

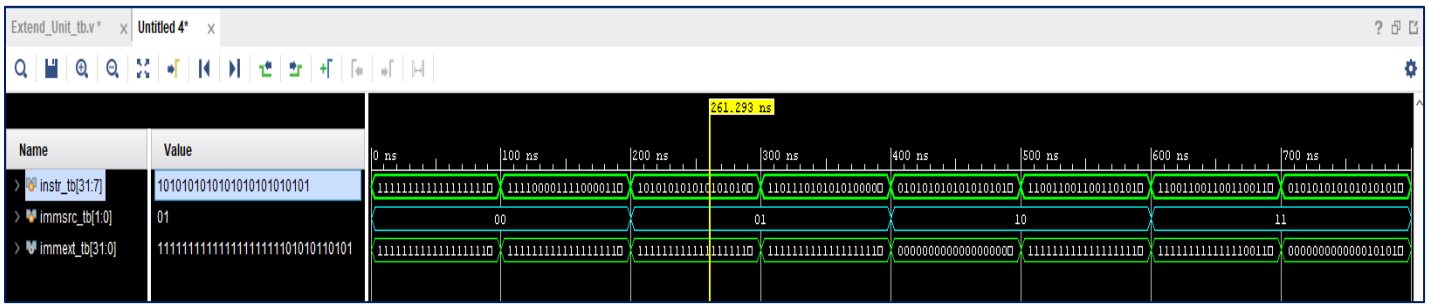
immex: 11111111111111111111111111111111



instr : 11110000111100001110000110

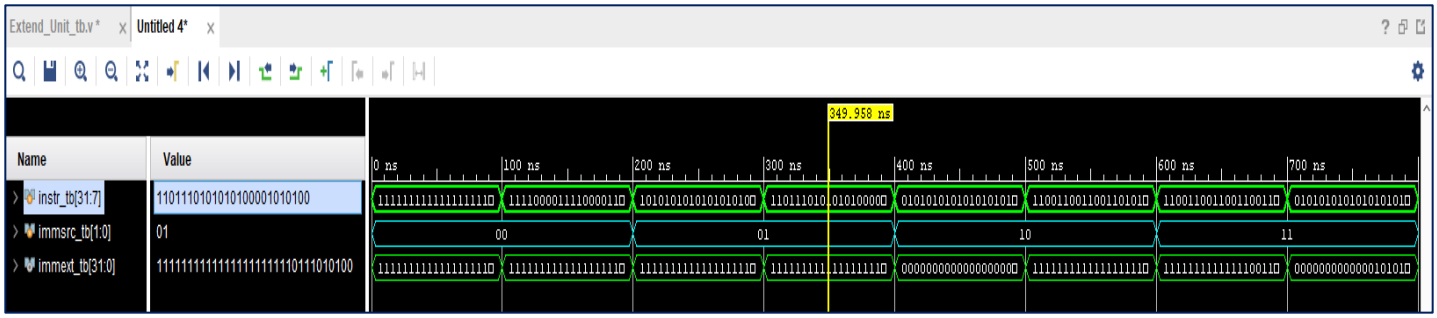
immex: 11111111111111111111111111100001111

⇨S-Type instruction :-



instr : 10101010101010101010101010101

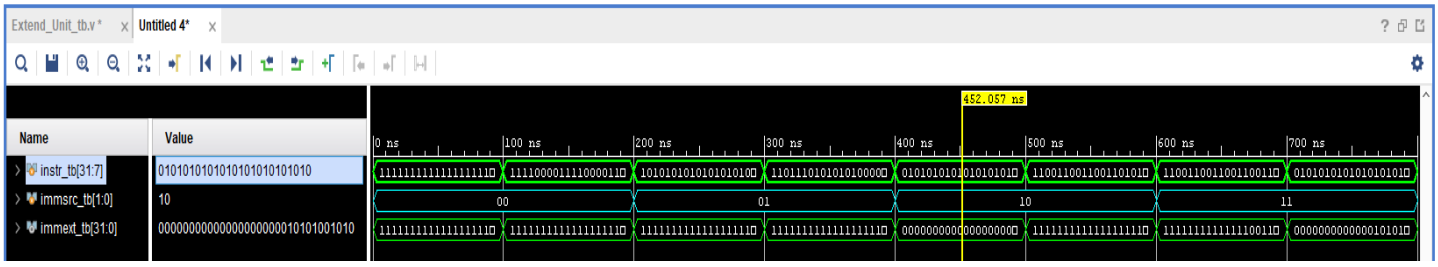
immex: 11111111111111111111101010110101



instr : 11011101010101000010101 00

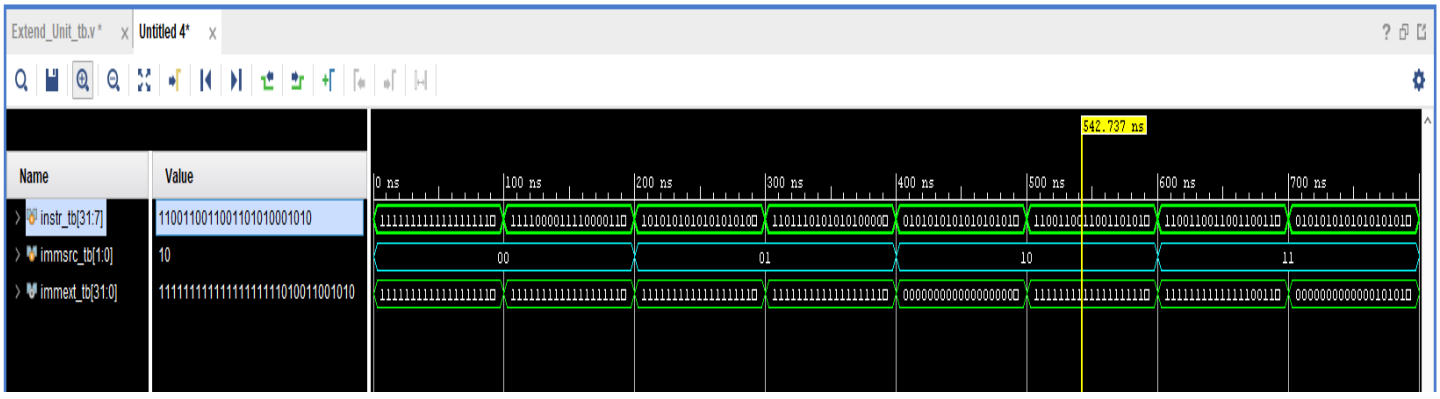
immex : 11111111111111111111011101010 0

⇨B-Type instruction :-



instr : 01010101010101010101010 10

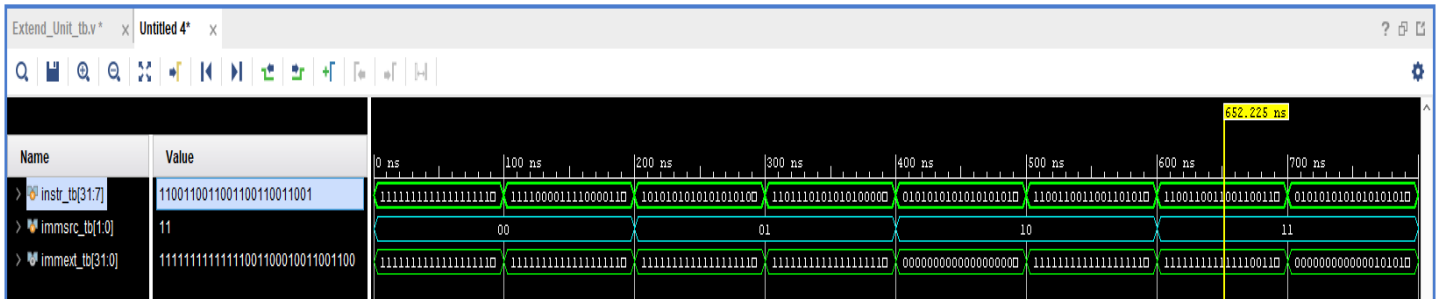
immex : 00000000000000000000101010010 10



instr : 11001100110011010100010 10

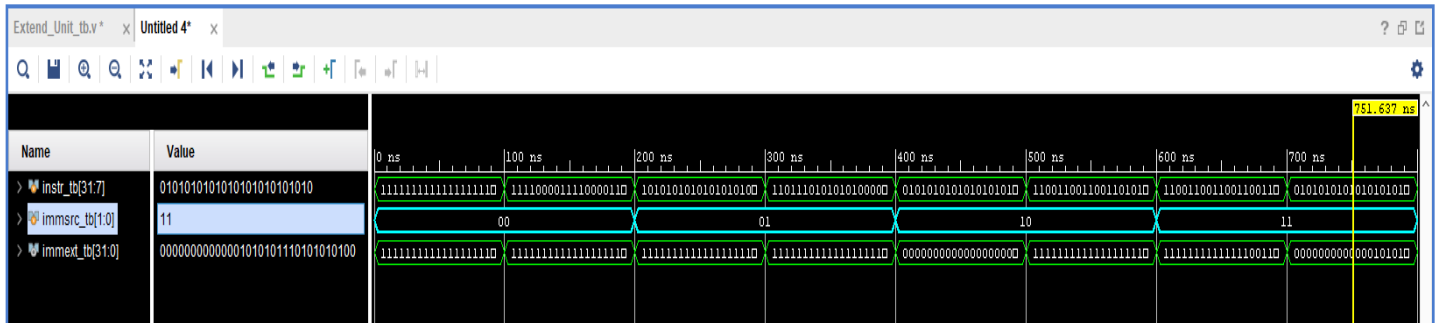
immex : 11111111111111111111010011001010

⇨J-Type instruction :-



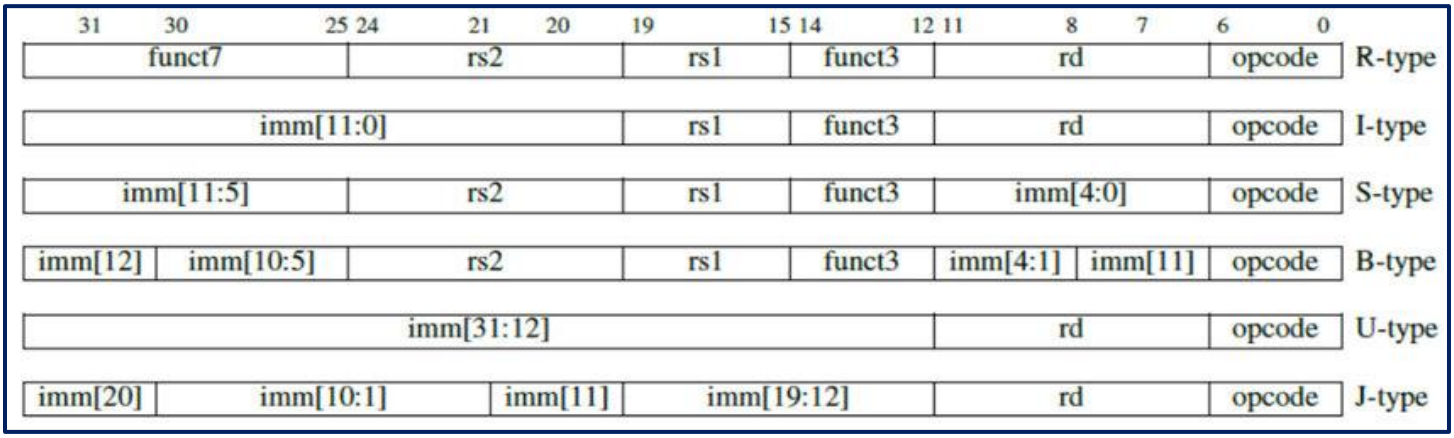
instr : 11001100110011001100111001

immex: 11111111111111001100000110011 00



instr : 01010101010101010101001010

immex: 00000000000010101011101010100



Made by: Hossam Ahmed Seyam