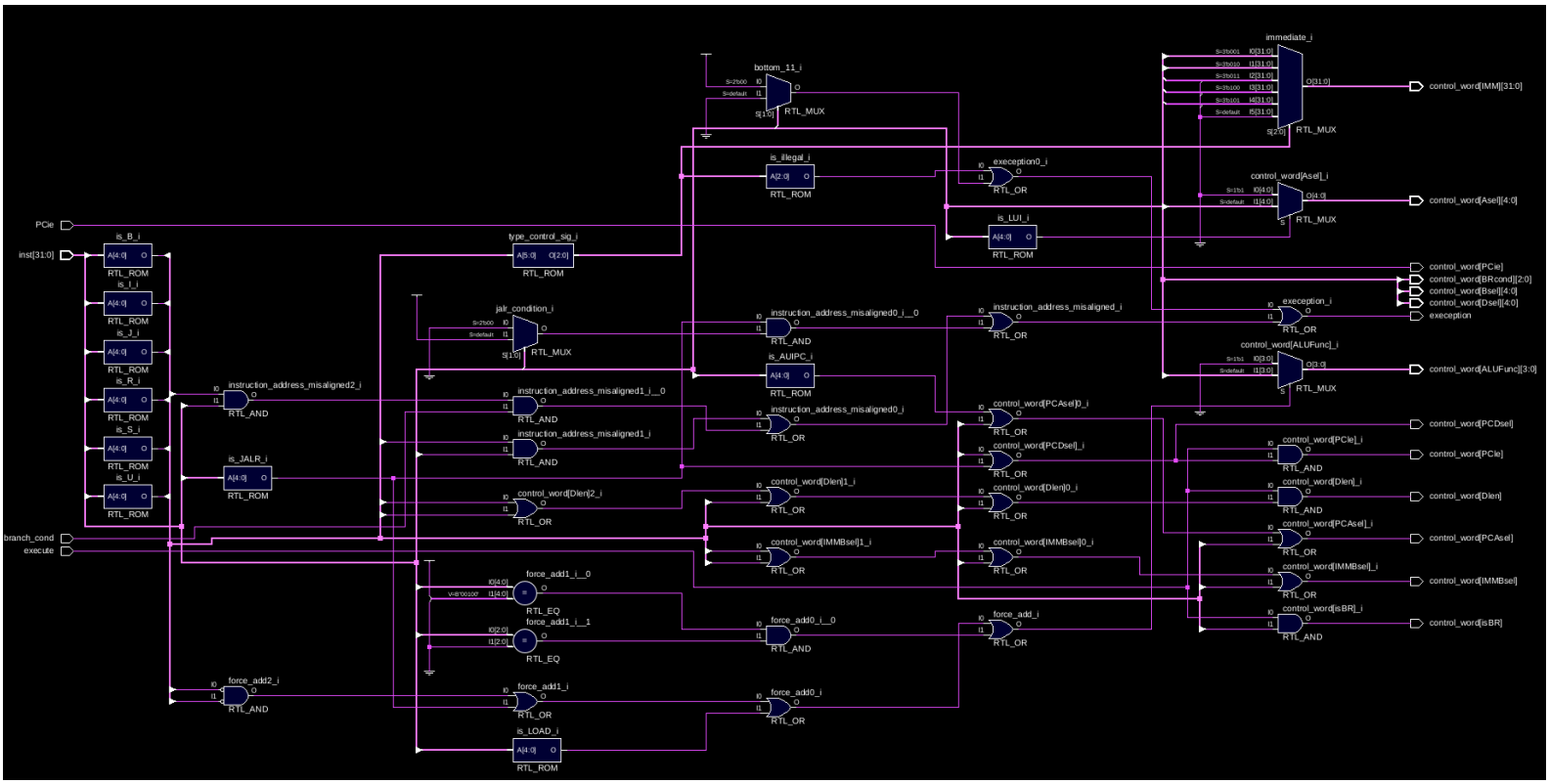
Lab 2: Instruction Decoder

-Overview-

This lab focuses on constructing the instruction decoder for a basic RISC-V (RV32I) core in VHDL. The decoder must correctly classify and decode all RV32I instructions, detect illegal instructions, and output a control word as defined in Lab 1.

-Design-

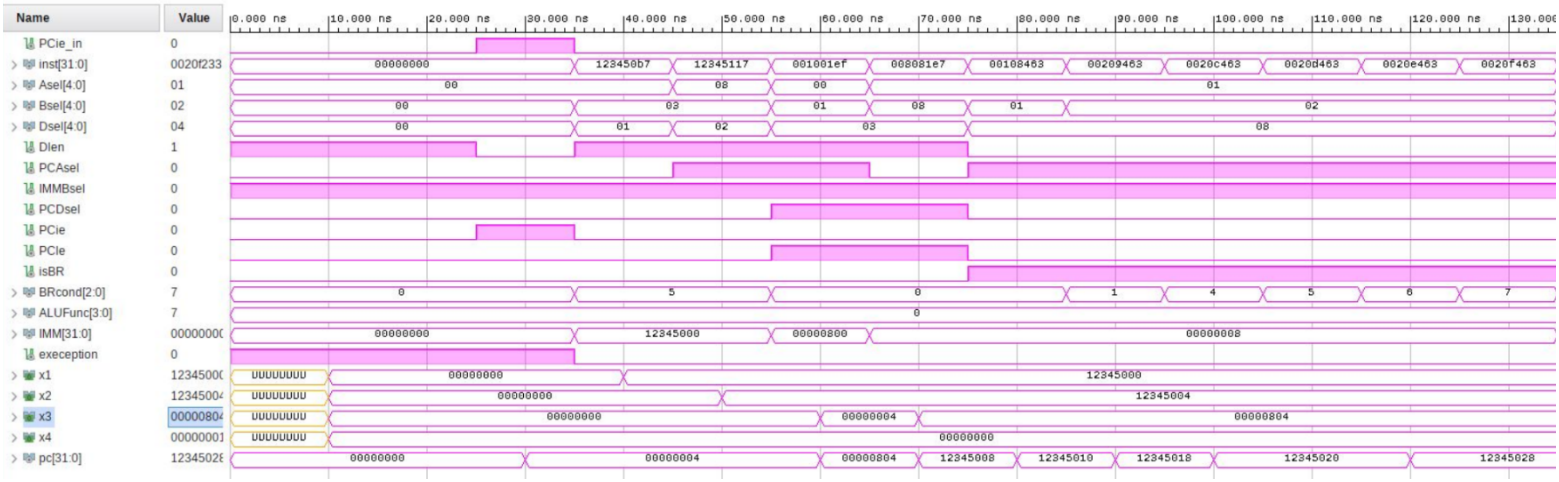
I started my decoder by first identifying what type of instruction it is. On the left, you can see some ROMs, each that identify one type: B, I, J, R, S, and U. Throughout the design, you can see more ROMs that identify specific instructions: JALR, AUIPC, LOAD, and LUI. On the top right, there is a mux that assembles the immediate based on the instruction type. I feed the types of signals into an encoder to decrease the control word on the immediate mux from 5 to 3. The A select is muxed with is_LUI, so the A is forced to 0 if it is a LUI. The exception line goes high if the following: the bottom 2 bits of instruction aren't 11, the opcode isn't any of the types that are allowed, or and address is misaligned in an instruction. All enable outputs are ANDed with the execute signal to lock datapath. The rest of the logic is just using the type of instruction, it is to decode the rest of the signals appropriately



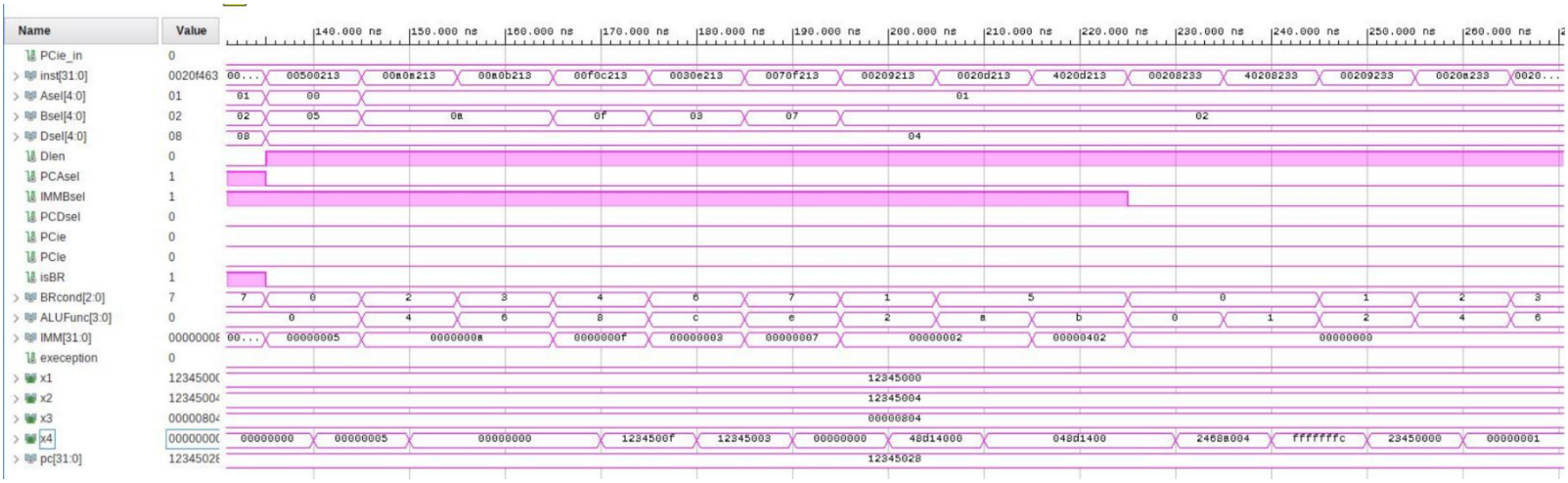
-Simulation-

I first started by testing the decoder on its own in the testbench provided by Dr. Pyeatt. I had to modify it a bit too fit my control word structure, and I added some test lines for illegal instructions. All the assertions came out good. This testbench I break down is with both the datapath and decoder.

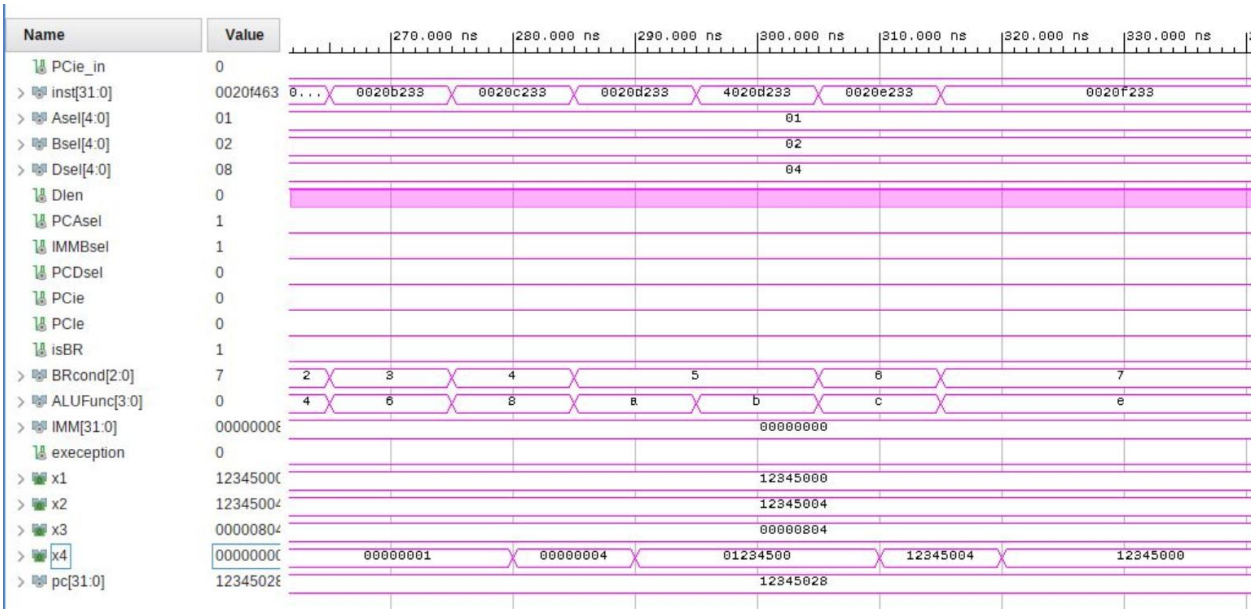
10-20 ns: reset
30ns: increment PC
40 ns: LUI x1 with 0x12345. Is padded with 12 zeros after 0x12345
50ns: AUIPC x2 with 0x12345. PC is 0 so it effectively load x2 with 0x12345000
60ns: JAL with x3 as return address and jump to plus 0x800
70ns: JALR with x3 as return address and jump to x1 plus 8
80ns: Branch if x1 = x1 to PC+8
90ns: Branch if x1 != x2 to PC+8
100ns: Branch if x1< x2 to PC+8
110ns: Branch if x1 >= x2 to PC+8
120ns: Branch if x1 < x2 to PC+8 (unsigned)
130ns: Branch if x1 >= x2 to PC+8 (unsigned)



140ns: ADDI x0 with 4 into x4
150ns: SLTI x1 < 10 into x4
160ns: SLTIU x1 < 10 into x4
170ns: XORI x1 ^ 15 into x4
180ns: ORI x1 or 3 into x4
190ns: ANDI x1 and 7 into x4
200ns: SLLI x1 left by 2 into x4
210ns: SRLI x1 right by 2 into x4
220ns: SRAI x1 right by 2 into x3
230ns: ADD x1 + x2 into x4
240ns: SUB x1 – x2 into x4
250ns: SLL x1 by x2 (lower 5 bits) into x4
260ns: SLT x1 < x2 into x4



270ns: SLTU x1 < x2 into x4
280ns: XOR x1 ^ x2 into x4
290ns: SRL x1 by x2 (bottom 5 bits) into x4
300ns: SRA x1 by x2 (bottom 5 bits) into x4
310ns: OR x1 or x2 into x4
320ns: AND x1 and x2 into x4



-Conclusion-

I didn’t take the lots of mux route like was shown in the example. I think identifying the basic types, and then only identifying specific instructions when needed was a decent strategy. I’m not sure if it is the most efficient, but it worked. I tried to implement as many signals as possible to go right from the instruction into the datapath. Overall, it wasn’t the hardest lab ever, and I learned a lot about each instruction.

-Appendix-

Opcode Encodings

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1		Asel	Bsel	Dsel	Dlen	PCAsel	IMMBsel	PCDsel	Pcle	isBR	Brcond	ALUFunc	IMM				
2	LUI (U)	0	X	rd	1	0	1	0	0	0	x	0000,	[31:12] +12 0's				
3	AUIPC (U)	X	X	rd	1	1	1	0	0	0	x	0000,	[31:12] +12 0's	*illegal if not adress aligned to 4 bytes			
4	JAL (J)	X	X	rd	1	1	1	1	1	0	x	0000,	[31s][19:12][20][30:21	*illegal if not adress aligned to 4 bytes			
5	JALR (I)	rs1	X	rd	1	0	1	1	1	0	x	0000,	[31s:20]	*illegal if not adress aligned to 4 bytes			
6	BXX (B)	rs1	rs2	X	0	1	1	X	0	1	[14:12]	0000,	[31s][7][30:25][11:6]				
7	L (I)	rs1	X	rd	1	0	1	0	0	0	x	0000,	[31s:20]				
8	S (S)	rs1	rs2	X	0	0	1	X	0	0	x	0000,	[31s:5][11:7]				
9	ADDI (I)	rs1	X	rd	1	0	1	0	0	0	x	[14:12][0]	[31s:20]				
10	SLTI	rs1	X	rd	1	0	1	0	0	0	x	[14:12][X]	[31s:20]				
11	SLTIU	rs1	X	rd	1	0	1	0	0	0	x	[14:12][X]	[31s:20]				
12	XORI	rs1	X	rd	1	0	1	0	0	0	x	[14:12][X]	[31s:20]				
13	ORI	rs1	X	rd	1	0	1	0	0	0	x	[14:12][X]	[31s:20]				
14	ANDI	rs1	X	rd	1	0	1	0	0	0	x	[14:12][X]	[31s:20]				
15	SLLI	rs1	X	rd	1	0	1	0	0	0	x	[14:12][30]	[31s:20]				
16	SRLI	rs1	X	rd	1	0	1	0	0	0	x	[14:12][30]	[31s:20]				
17	SRAI	rs1	X	rd	1	0	1	0	0	0	x	[14:12][30]	[31s:20]				
18	ADD	rs1	X	rd	1	0	1	0	0	0	x	[14:12][30]	X				
19	SUB	rs1	rs2	rd	1	0	1	0	0	0	x	[14:12][30]	X				
20	SLLI	rs1	rs2	rd	1	0	1	0	0	0	x	[14:12][30]	X				
21	SLT	rs1	rs2	rd	1	0	0	0	0	0	x	[14:12][30]	X				
22	SLTU	rs1	rs2	rd	1	0	0	0	0	0	x	[14:12][30]	X				
23	XOR	rs1	rs2	rd	1	0	0	0	0	0	x	[14:12][30]	X				
24	SRL	rs1	rs2	rd	1	0	0	0	0	0	x	[14:12][30]	X				
25	SRA	rs1	rs2	rd	1	0	0	0	0	0	x	[14:12][30]	X				
26	OR	rs1	rs2	rd	1	0	0	0	0	0	x	[14:12][30]	X				
27	AND	rs1	rs2	rd	1	0	0	0	0	0	x	[14:12][30]	X				
28																	

Decoder VHDL

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.RISCV_package.all;

entity Instruction_Decoder is
    Port ( execute, PCie, branch_cond : in std_logic;
           inst : in std_logic_vector (XLEN-1 downto 0);
           control_word : out control_word;
           exeception: out std_logic);
end Instruction_Decoder;

architecture Behavioral of Instruction_Decoder is
    signal immediate: std_logic_vector(XLEN-1 downto 0);
    signal type_control_sig: std_logic_vector(2 downto 0);
    signal jalr_condition, instruction_address_misaligned, bottom_11: std_logic;
    signal force_add: std_logic;
    signal is_LOAD, is_LUI, is_AUIPC, is_JALR, is_ADDI, is_R, is_I, is_S, is_B, is_U, is_J, is_illegal: std_logic;
begin
    --throw expection when inst(1 downto 0) != 0
    bottom_11 <= '1' when inst(1 downto 0) = "00" else '0';

    --jump/branch alligment
    jalr_condition <= '0' when inst(21 downto 20) = "00" else '1';
    instruction_address_misaligned <= (is_J and inst(21)) or
                                      (is_B and inst(8) and branch_cond) or
                                      (is_JALR and jalr_condition);

    --opcode interpretation
    is_LOAD <= '1' when inst(6 downto 2) = "00000" else '0';
    is_LUI <= '1' when inst(6 downto 2) = "01101" else '0';
    is_AUIPC <= '1' when inst(6 downto 2) = "00101" else '0';
    is_JALR <= '1' when inst(6 downto 2) = "11001" else '0';
    is_ADDI <= '1' when inst(6 downto 2) = "00100" and inst(14 downto 12) = "000" else '0';

    --MISC-MEM aka FENCE inst(6 downto 2) = "00011"
    --STYSTEM aka ECALL and EBreak inst(6 downto 2) = "11100"
    is_R <= '1' when inst(6 downto 2) = "01100" else '0';
    with inst(6 downto 2) select is_I <=
            '1' when "00100" | "00000" | "11001",
            '0' when others;
    is_S <= '1' when inst(6 downto 2) = "01000" else '0';
    is_B <= '1' when inst(6 downto 2) = "11000" else '0';
    --is_U <= is_LUI or is_AUIPC;
    with inst(6 downto 2) select is_U <=
            '1' when "01101" | "00101",
            '0' when others;
    is_J <= '1' when inst(6 downto 2) = "11011" else '0';
```

```vhdl
    --encode these into a 3 bit control word
    with std_logic_vector'(is_R & is_I & is_S & is_B & is_U & is_J) select type_control_sig <=
        "000" when "100000",  --R
        "001" when "010000",  --I
        "010" when "001000",  --S
        "011" when "000100",  --B
        "100" when "000010",  --U
        "101" when "000001",  --J
        "111" when others;

    is_illegal <= '1' when type_control_sig = "111" else '0';


    --immediate assembly
    with type_control_sig select immediate <=
        (31 downto 11 => inst(31)) & inst(30 downto 20) when "001", --I
        (31 downto 11 => inst(31)) & inst(30 downto 25) & inst(11 downto 7) when "010", --S
        (31 downto 12 => inst(31)) & inst(7) & inst(30 downto 25) & inst(11 downto 8) & '0' when "011", --B
        inst(31) & inst (30 downto 12) & "000000000000" when "100", --U
        (31 downto 20 => inst(31)) & inst(19 downto 12) & inst(20) & inst(30 downto 21) & '0' when "101", --J
        (others => '0') when others;

    --all the functions that need a add
    force_add <= (not is_R and not is_I) or is_JALR or is_LOAD or is_ADDI;

    control_word.Asel <= "00000" when is_LUI = '1' else inst(19 downto 15);
    control_word.Bsel <= inst(24 downto 20);
    control_word.Dsel <= inst(11 downto 7);
    control_word.Dlen <= execute and (is_R or is_I or is_U or is_J);
    control_word.PCAsel <= is_AUIPC or is_J or is_B;
    control_word.IMMBsel <= is_S or is_I or is_U or is_J or is_B;
    control_word.PCDsel <= is_J or is_JALR;
    control_word.PCie <= PCie;
    control_word.PCle <= execute and (is_J or is_JALR);
    control_word.isBR <= execute and (is_B);
    control_word.BRcond <= inst(14 downto 12);
    control_word.ALUFunc <= "0000" when force_add = '1' else inst(14 downto 12) & (inst(30));
    control_word.IMM <= immediate;

    exeception <= is_illegal or bottom_11 or instruction_address_misaligned;
end Behavioral;
```

Just Decoder Test Bench

```
entity Instruction_Decoder_tb is
end Instruction_Decoder_tb;

architecture Behavioral of Instruction_Decoder_tb is
    signal execute : std_logic := '1';
    signal PCie_in : std_logic := '0';
    signal branch_cond : std_logic := '0';
    signal inst : std_logic_vector(31 downto 0) := "00000000000000000000000000000000";
    signal Asel : std_logic_vector(REGS_ADDR_BITS-1 downto 0) := "00000";
    signal Bsel : std_logic_vector(REGS_ADDR_BITS-1 downto 0) := "00000";
    signal Dsel : std_logic_vector(REGS_ADDR_BITS-1 downto 0) := "00000";
    signal Dlen : std_logic := '0';
    signal PCAsel : std_logic := '0';
    signal IMMBsel : std_logic := '0';
    signal PCDsel : std_logic := '0';
    signal PCie : std_logic := '0';
    signal PCle : std_logic := '0';
    signal isBR : std_logic:= '0';
    signal BRcond : std_logic_vector(2 downto 0):= "000";
    signal ALUFunc : std_logic_vector(3 downto 0):= "0000";
    signal pre_ALUFunc : std_logic_vector(3 downto 0):= "0000";
    signal IMM : std_logic_vector(31 downto 0) := "00000000000000000000000000000000";
    signal exeception : std_logic;
begin

  uut: entity work.Instruction_Decoder (Behavioral)
      port map( execute => execute,
                PCie => PCie_in,
                branch_cond => branch_cond,
                inst => inst,
                control_word.Asel => Asel,
                control_word.Bsel => Bsel,
                control_word.Dsel => Dsel,
                control_word.Dlen => Dlen,
                control_word.PCAsel => PCAsel,
                control_word.IMMBsel => IMMBsel,
                control_word.PCDsel => PCDsel,
                control_word.PCie => PCie,
                control_word.PCle => PCle,
                control_word.isBR => isBR,
                control_word.BRcond => BRcond,
                control_word.ALUFunc => pre_ALUFunc,
                control_word.IMM => IMM,
                exeception => exeception
                );
ALUFunc <= pre_ALUFunc(0) & pre_ALUFunc(3 downto 1);
  test: process
test: process
  begin
    inst <= B"0000000_00010_01000_111_11001_0110111";
    wait for 10 ns;
    assert unsigned(Dsel) = 25 and signed(IMM) = 2387968 and unsigned(Asel) = 0 and ALUfunc = "0000" report "LUI #1 not working" severity warning;

    inst <= B"0000000_01000_00010_000_01111_0110111";
    wait for 10 ns;
    assert unsigned(Dsel) = 15 and signed(IMM) = 8454144 and unsigned(Asel) = 0 and ALUfunc = "0000" report "LUI #2 not working" severity warning;

    inst <= B"0000101_11110_11010_000_10111_0110111";
    wait for 10 ns;
    assert unsigned(Dsel) = 23 and signed(IMM) = 200081408 and unsigned(Asel) = 0 and ALUfunc = "0000" report "LUI #3 not working" severity warning;


    inst <= B"0001010_00010_11001_010_11101_0110111";
    wait for 10 ns;
    assert unsigned(Dsel) = 29 and signed(IMM) = 338468864 and unsigned(Asel) = 0 and ALUfunc = "0000" report "LUI #4 not working" severity warning;


    inst <= B"0000100_10101_11100_101_11000_0110111";
    wait for 10 ns;
    assert unsigned(Dsel) = 24 and signed(IMM) = 157175808 and unsigned(Asel) = 0 and ALUfunc = "0000" report "LUI #5 not working" severity warning;

    inst <= B"0000000_10101_00010_001_10100_0010111";
    wait for 10 ns;
    assert unsigned(Dsel) = 20 and signed(IMM) = 22089728 and ALUfunc = "0000" report "AUIPC #1 not working" severity warning;

    inst <= B"0000000_00000_11001_000_10111_0010111";
    wait for 10 ns;
    assert unsigned(Dsel) = 23 and signed(IMM) = 819200 and ALUfunc = "0000" report "AUIPC #2 not working" severity warning;

    inst <= B"1111111_00101_11111_111_00000_1101111";
    wait for 10 ns;
    assert unsigned(Dsel) = 0 and signed(IMM) = -28 and ALUfunc = "0000" report "JAL #1 not working" severity warning;

    inst <= B"0000000_11000_00000_000_00101_1101111";
    wait for 10 ns;
    assert unsigned(Dsel) = 5 and signed(IMM) = 24 and ALUfunc = "0000" report "JAL #2 not working" severity warning;

    inst <= B"1111111_01100_00001_000_00000_1100111";
    wait for 10 ns;
    assert unsigned(Dsel) = 0 and unsigned(Asel) = 1 and signed(IMM) = -20 and ALUfunc = "0000" report "JALR #1 not working" severity warning;

```

```
97      inst <= B"0001001_01100_00010_000_00000_1100111";
98      wait for 10 ns;
99      assert unsigned(Dsel) = 0 and unsigned(Asel) = 2 and signed(IMM) = 300 and ALUfunc = "0000" report "JALR #2 not working" severity warning;
100
101     inst <= B"0000000_00000_00000_000_01100_1100011";
102     wait for 10 ns;
103     assert Dlen = '0' and unsigned(Asel) = 0 and unsigned(Bsel) = 0 and signed(IMM) = 12 and ALUfunc = "0000" report "BRANCH #1 not working" severity warning;
104
105     inst <= B"0000000_00010_00001_001_01000_1100011";
106     wait for 10 ns;
107     assert Dlen = '0' and unsigned(Asel) = 1 and unsigned(Bsel) = 2 and signed(IMM) = 8 and ALUfunc = "0000" report "BRANCH #2 not working" severity warning;
108
109     inst <= B"0000000_00100_00011_100_00100_1100011";
110     wait for 10 ns;
111     assert Dlen = '0' and unsigned(Asel) = 3 and unsigned(Bsel) = 4 and signed(IMM) = 4 and ALUfunc = "0000" report "BRANCH #3 not working" severity warning;
112
113     inst <= B"0000000_00101_00100_101_00000_1100011";
114     wait for 10 ns;
115     assert Dlen = '0' and unsigned(Asel) = 4 and unsigned(Bsel) = 5 and signed(IMM) = 0 and ALUfunc = "0000" report "BRANCH #4 not working" severity warning;
116
117     inst <= B"1111111_00111_00110_110_11101_1100011";
118     wait for 10 ns;
119     assert Dlen = '0' and unsigned(Asel) = 6 and unsigned(Bsel) = 7 and signed(IMM) = -4 and ALUfunc = "0000" report "BRANCH #5 not working" severity warning;
120
121     inst <= B"1111111_01000_00111_111_11001_1100011";
122     wait for 10 ns;
123     assert Dlen = '0' and unsigned(Asel) = 7 and unsigned(Bsel) = 8 and signed(IMM) = -8 and ALUfunc = "0000" report "BRANCH #6 not working" severity warning;
124
125     inst <= B"0100111_00101_01010_000_01001_0000011";
126     wait for 10 ns;
127     assert unsigned(Dsel) = 9 and unsigned(Asel) = 10 and signed(IMM) = 1253 and ALUfunc = "0000" report "LOAD #1 not working" severity warning;
128
129     inst <= B"1101110_01101_01100_001_01011_0000011";
130     wait for 10 ns;
131     assert unsigned(Dsel) = 11 and unsigned(Asel) = 12 and signed(IMM) = -563 and ALUfunc = "0000" report "LOAD #2 not working" severity warning;
132
133     inst <= B"0001100_10011_01110_010_01101_0000011";
134     wait for 10 ns;
135     wait for 10 ns;
136     assert unsigned(Dsel) = 13 and unsigned(Asel) = 14 and signed(IMM) = 403 and ALUfunc = "0000" report "LOAD #3 not working" severity warning;
137
138     inst <= B"0001100_00010_10000_100_01111_0000011";
139     wait for 10 ns;
140     assert unsigned(Dsel) = 15 and unsigned(Asel) = 16 and signed(IMM) = 386 and ALUfunc = "0000" report "LOAD #4 not working" severity warning;
141
142     inst <= B"0001001_01111_10010_101_10001_0000011";
143     wait for 10 ns;
144     assert unsigned(Dsel) = 17 and unsigned(Asel) = 18 and signed(IMM) = 303 and ALUfunc = "0000" report "LOAD #5 not working" severity warning;
145
146     inst <= B"0001010_10011_10100_000_11100_0100011";
147     wait for 10 ns;
148     assert Dlen = '0' and unsigned(Asel) = 20 and unsigned(Bsel) = 19 and signed(IMM) = 348 and ALUfunc = "0000" report "STORE #1 not working" severity warnin(
149
150     inst <= B"0011010_10101_10110_001_00011_0100011";
151     wait for 10 ns;
152     assert Dlen = '0' and unsigned(Asel) = 22 and unsigned(Bsel) = 21 and signed(IMM) = 835 and ALUfunc = "0000" report "STORE #2 not working" severity warnin(
153
154     inst <= B"0011101_10111_11000_010_01000_0100011";
155     wait for 10 ns;
156     assert Dlen = '0' and unsigned(Asel) = 24 and unsigned(Bsel) = 23 and signed(IMM) = 936 and ALUfunc = "0000" report "STORE #3 not working" severity warnin(
157
158     inst <= B"0011110_01000_11010_000_11001_0010011";
159     wait for 10 ns;
160     assert unsigned(Dsel) = 25 and unsigned(Asel) = 26 and signed(IMM) = 968 and ALUfunc = "0000" report "ADDI #1 not working" severity warning;
161
162     inst <= B"1110110_10101_11100_000_11011_0010011";
163     wait for 10 ns;
164     assert unsigned(Dsel) = 27 and unsigned(Asel) = 28 and signed(IMM) = -299 and ALUfunc = "0000" report "ADDI #2 not working" severity warning;
165
166     inst <= B"1111111_11111_00000_000_11110_0010011";
167     wait for 10 ns;
168     assert unsigned(Dsel) = 30 and unsigned(Asel) = 0 and signed(IMM) = -1 and ALUfunc = "0000" report "ADDI #3 not working" severity warning;
169
170     inst <= B"0111111_11111_00001_000_11100_0010011";
171     wait for 10 ns;
172     assert unsigned(Dsel) = 28 and unsigned(Asel) = 1 and signed(IMM) = 2047 and ALUfunc = "0000" report "ADDI #4 not working" severity warning;
173
174     inst <= B"1110000_01100_10001_010_01111_0010011";
175     wait for 10 ns;
176     assert unsigned(Dsel) = 15 and unsigned(Asel) = 17 and signed(IMM) = -500 and ALUfunc(2 downto 0) = "010" report "SLTI not working" severity warning;
177
178     inst <= B"1110110_11100_10010_011_00111_0010011";
179     wait for 10 ns;
180     assert unsigned(Dsel) = 7 and unsigned(Asel) = 18 and signed(IMM) = -292 and ALUfunc(2 downto 0) = "011" report "SLTIU not working" severity warning;
181
182     inst <= B"1111100_11100_00010_100_00001_0010011";
183     wait for 10 ns;
184     assert unsigned(Dsel) = 1 and unsigned(Asel) = 2 and signed(IMM) = -100 and ALUfunc(2 downto 0) = "100" report "XORI not working" severity warning;
185
186     inst <= B"0001111_10100_00100_110_00011_0010011";
187     wait for 10 ns;
188     assert unsigned(Dsel) = 3 and unsigned(Asel) = 4 and signed(IMM) = 500 and ALUfunc(2 downto 0) = "110" report "ORI not working" severity warning;
```

```
        inst <= B"1110101_11111_00110_111_00101_0010011";
        wait for 10 ns;
        assert unsigned(Dsel) = 5 and unsigned(Asel) = 6 and signed(IMM) = -321 and ALUfunc(2 downto 0) = "111" report "ANDI not working" severity warning;

        inst <= B"0000000_00001_01000_001_00111_0010011";
        wait for 10 ns;
        assert unsigned(Dsel) = 7 and unsigned(Asel) = 8 and signed(IMM) = 1 and ALUfunc = "0001" report "SLLI #1 not working" severity warning;

        inst <= B"0000000_11101_01010_001_01001_0010011";
        wait for 10 ns;
        assert unsigned(Dsel) = 9 and unsigned(Asel) = 10 and signed(IMM) = 29 and ALUfunc = "0001" report "SLLI #2 not working" severity warning;

        inst <= B"0000000_00010_01100_101_01011_0010011";
        wait for 10 ns;
        assert unsigned(Dsel) = 11 and unsigned(Asel) = 12 and signed(IMM) = 2 and ALUfunc = "0101" report "SRLI #1 not working" severity warning;

        inst <= B"0000000_11111_01110_101_01101_0010011";
        wait for 10 ns;
        assert unsigned(Dsel) = 13 and unsigned(Asel) = 14 and signed(IMM) = 31 and ALUfunc = "0101" report "SRLI #2 not working" severity warning;

        inst <= B"0100000_00011_10000_101_01111_0010011";
        wait for 10 ns;
        assert unsigned(Dsel) = 15 and unsigned(Asel) = 16 and unsigned(IMM(4 downto 0)) = 3 and ALUfunc = "1101" report "SRAI #1 not working" severity warning;

        inst <= B"0100000_11110_10010_101_10001_0010011";
        wait for 10 ns;
        assert unsigned(Dsel) = 17 and unsigned(Asel) = 18 and unsigned(IMM(4 downto 0)) = 30 and ALUfunc = "1101" report "SRAI #2 not working" severity warning;

        inst <= B"0000000_00010_00001_000_00000_0110011";
        wait for 10 ns;
        assert unsigned(Dsel) = 0 and unsigned(Asel) = 1 and unsigned(Bsel) = 2 and ALUfunc = "0000" report "ADD not working" severity warning;

        inst <= B"0100000_00100_00011_000_00010_0110011";
        wait for 10 ns;
        assert unsigned(Dsel) = 2 and unsigned(Asel) = 3 and unsigned(Bsel) = 4 and ALUfunc = "1000" report "SUB not working" severity warning;

        inst <= B"0000000_00111_00110_001_00101_0110011";
        wait for 10 ns;
        assert unsigned(Dsel) = 5 and unsigned(Asel) = 6 and unsigned(Bsel) = 7 and ALUfunc = "0001" report "SLL not working" severity warning;

        inst <= B"0000000_01010_01000_010_01000_0110011";
        wait for 10 ns;
        assert unsigned(Dsel) = 8 and unsigned(Asel) = 8 and unsigned(Bsel) = 10 and ALUfunc = "0010" report "SLT not working" severity warning;

        inst <= B"0000000_01011_01010_011_01001_0110011";
        wait for 10 ns;
        assert unsigned(Dsel) = 9 and unsigned(Asel) = 10 and unsigned(Bsel) = 11 and ALUfunc = "0011" report "SLTU not working" severity warning;
        inst <= B"0000000_01101_01100_100_01011_0110011";
        wait for 10 ns;
        assert unsigned(Dsel) = 11 and unsigned(Asel) = 12 and unsigned(Bsel) = 13 and ALUfunc = "0100" report "XOR not working" severity warning;

        inst <= B"0000000_10000_01111_101_01110_0110011";
        wait for 10 ns;
        assert unsigned(Dsel) = 14 and unsigned(Asel) = 15 and unsigned(Bsel) = 16 and ALUfunc = "0101" report "SRL not working" severity warning;

        inst <= B"0100000_10011_10010_101_10001_0110011";
        wait for 10 ns;
        assert unsigned(Dsel) = 17 and unsigned(Asel) = 18 and unsigned(Bsel) = 19 and ALUfunc = "1101" report "SRA not working" severity warning;

        inst <= B"0000000_10110_10101_110_10100_0110011";
        wait for 10 ns;
        assert unsigned(Dsel) = 20 and unsigned(Asel) = 21 and unsigned(Bsel) = 22 and ALUfunc = "0110" report "OR not working" severity warning;

        inst <= B"0000000_11010_11000_111_10111_0110011";
        wait for 10 ns;
        assert unsigned(Dsel) = 23 and unsigned(Asel) = 24 and unsigned(Bsel) = 26 and ALUfunc = "0111" report "AND not working" severity warning;

        inst <= B"1111011_00000_00000_000_10001_1100011";
        wait for 10 ns;
        assert Dlen = '0' and unsigned(Asel) = 0 and unsigned(Bsel) = 0 and signed(IMM) = -144 and ALUfunc = "0000" report "BRANCH #7 not working" severity warning

        inst <= B"1111101_11001_11111_111_00000_1101111";
        wait for 10 ns;
        assert unsigned(Dsel) = 0 and ALUfunc = "0000" and signed(IMM) = -72 report "JAL #3 not working" severity warning;

        inst <= B"0100000_00000_00010_000_00001_0010011";
        wait for 10 ns;
        assert unsigned(Dsel) = 1 and unsigned(Asel) = 2 and signed(IMM) = 1024 and ALUfunc = "0000" report "ADDI #5 not working" severity warning;
        wait;

        --not end in 11
        inst <= B"0100000_00000_00010_000_00001_0010001";
        wait for 10 ns;
        assert exeception = '1' report "ILLEGAL #1 not working" severity warning;

        --jump to illegal address
        inst <= B"1111111_00101_11111_111_00000_1101111";
        wait for 10 ns;
        assert exeception = '1' report "ILLEGAL #2 not working" severity warning;

        --illegal opcode
        inst <= B"0100000_00000_00010_000_00001_1101011";
        wait for 10 ns;
        assert exeception = '1' report "ILLEGAL #3 not working" severity warning;
```

Decoder + Datapath Test Bench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.RISCV_package.all;


entity Decoder_Plus_Datapath_tb is
end Decoder_Plus_Datapath_tb;

architecture Behavioral of Decoder_Plus_Datapath_tb is
    signal clk: std_logic := '1';
    signal reset: std_logic := '0';
    signal execute : std_logic := '1';
    signal PCie_in : std_logic := '0';
    signal branch_cond : std_logic := '0';
    signal inst : std_logic_vector(31 downto 0) := "00000000000000000000000000000000";
    signal Asel : std_logic_vector(REGS_ADDR_BITS-1 downto 0) := "00000";
    signal Bsel : std_logic_vector(REGS_ADDR_BITS-1 downto 0) := "00000";
    signal Dsel : std_logic_vector(REGS_ADDR_BITS-1 downto 0) := "00000";
    signal Dlen : std_logic := '0';
    signal PCAsel : std_logic := '0';
    signal IMMBsel : std_logic := '0';
    signal PCDsel : std_logic := '0';
    signal PCie : std_logic := '0';
    signal PCle : std_logic := '0';
    signal isBR : std_logic:= '0';
    signal BRcond : std_logic_vector(2 downto 0):= "000";
    signal ALUFunc : std_logic_vector(3 downto 0):= "0000";
    signal pre_ALUFunc : std_logic_vector(3 downto 0):= "0000";
    signal IMM : std_logic_vector(31 downto 0) := "00000000000000000000000000000000";
    signal exeception : std_logic;
begin

uut1: entity work.Instruction_Decoder (Behavioral)
    port map( execute => execute,
              PCie => PCie_in,
              branch_cond => branch_cond,
              inst => inst,
              control_word.Asel => Asel,
              control_word.Bsel => Bsel,
              control_word.Dsel => Dsel,
              control_word.Dlen => Dlen,
              control_word.PCAsel => PCAsel,
              control_word.IMMBsel => IMMBsel,
              control_word.PCDsel => PCDsel,
              control_word.PCie => PCie,
              control_word.PCle => PCle,
              control_word.isBR => isBR,
              control_word.BRcond => BRcond,
              control_word.ALUFunc =>ALUFunc,
              control_word.IMM => IMM,
              exeception => exeception
              );

uut2: entity work.Datapath (Behavioral)
    port map( clk => clk,
              reset =>reset,
              control_word.Asel => Asel,
              control_word.Bsel => Bsel,
              control_word.Dsel => Dsel,
              control_word.Dlen => Dlen,
              control_word.PCAsel => PCAsel,
              control_word.IMMBsel => IMMBsel,
              control_word.PCDsel => PCDsel,
              control_word.PCie => PCie,
              control_word.PCle => PCle,
              control_word.isBR => isBR,
              control_word.BRcond => BRcond,
              control_word.ALUFunc => ALUFunc,
              control_word.IMM => IMM
              );

 --clk
 clk <= not clk after 5 ns;

```

```
76    test: process
77      begin
78
79        wait for 5 ns;
80        --reset in beginning
81        reset <= '1';
82        wait for 20 ns;
83        reset <= '0';
84
85      -- inc PC
86      execute <= '0';
87      PCie_in <= '1';
88      wait for 10 ns;
89      execute <= '1';
90      PCie_in <= '0';
91
92
93      -- LUI x1, 0x12345
94      inst <= B"00010010001101000101_00001_0110111";
95      wait for 10 ns;
96
97      -- AUIPC x2, 0x12345
98      inst <= B"00010010001101000101_00010_0010111";
99      wait for 10 ns;
100
101     -- JAL x3, +16
102     inst <= B"00000000000100000000_00011_1101111";
103     wait for 10 ns;
104
105     -- JALR x3, x1, 8
106     inst <= B"000000001000_00001_000_00011_1100111";
107     wait for 10 ns;
108
109     -- BEQ x1, x1, +8
110     inst <= B"0000000_00001_00001_000_01000_1100011";
111     wait for 10 ns;
112
113     -- BNE x1, x2, +8
114     inst <= B"0000000_00010_00001_001_01000_1100011";
115     wait for 10 ns;
116
117     -- BLT x1, x2, +8
118     inst <= B"0000000_00010_00001_100_01000_1100011";
119     wait for 10 ns;
120
125      -- BLTU x1, x2, +8
126      inst <= B"0000000_00010_00001_110_01000_1100011";
127      wait for 10 ns;
128
129      -- BGEU x1, x2, +8
130      inst <= B"0000000_00010_00001_111_01000_1100011";
131      wait for 10 ns;
132
133      -- ADDI x4, x0, 5
134      inst <= B"000000000101_00000_000_00100_0010011";
135      wait for 10 ns;
136
137      -- SLTI x4, x1, 10
138      inst <= B"000000001010_00001_010_00100_0010011";
139      wait for 10 ns;
140
141      -- SLTIU x4, x1, 10
142      inst <= B"000000001010_00001_011_00100_0010011";
143      wait for 10 ns;
144
145      -- XORI x4, x1, 15
146      inst <= B"000000001111_00001_100_00100_0010011";
147      wait for 10 ns;
148
149      -- ORI x4, x1, 3
150      inst <= B"000000000011_00001_110_00100_0010011";
151      wait for 10 ns;
152
153      -- ANDI x4, x1, 7
154      inst <= B"000000000111_00001_111_00100_0010011";
155      wait for 10 ns;
156
157      -- SLLI x4, x1, 2
158      inst <= B"0000000_00010_00001_001_00100_0010011";
159      wait for 10 ns;
160
161      -- SRLI x4, x1, 2
162      inst <= B"0000000_00010_00001_101_00100_0010011";
163      wait for 10 ns;
164
165      -- SRAI x4, x1, 2
166      inst <= B"0100000_00010_00001_101_00100_0010011";
167      wait for 10 ns;
```

```
169         -- ADD x4, x1, x2
170         inst <= B"0000000_00010_00001_000_00100_0110011";
171         wait for 10 ns;
172
173         -- SUB x4, x1, x2
174         inst <= B"0100000_00010_00001_000_00100_0110011";
175         wait for 10 ns;
176
177         -- SLL x4, x1, x2
178         inst <= B"0000000_00010_00001_001_00100_0110011";
179         wait for 10 ns;
180
181         -- SLT x4, x1, x2
182         inst <= B"0000000_00010_00001_010_00100_0110011";
183         wait for 10 ns;
184
185         -- SLTU x4, x1, x2
186         inst <= B"0000000_00010_00001_011_00100_0110011";
187         wait for 10 ns;
188
189         -- XOR x4, x1, x2
190         inst <= B"0000000_00010_00001_100_00100_0110011";
191         wait for 10 ns;
192
193         -- SRL x4, x1, x2
194         inst <= B"0000000_00010_00001_101_00100_0110011";
195         wait for 10 ns;
196
197         -- SRA x4, x1, x2
198         inst <= B"0100000_00010_00001_101_00100_0110011";
199         wait for 10 ns;
200
201         -- OR x4, x1, x2
202         inst <= B"0000000_00010_00001_110_00100_0110011";
203         wait for 10 ns;
204
205         -- AND x4, x1, x2
206         inst <= B"0000000_00010_00001_111_00100_0110011";
207         wait for 10 ns;
208         wait;
209     end process;
210 end Behavioral;
211
```