Lab 1: Datapath

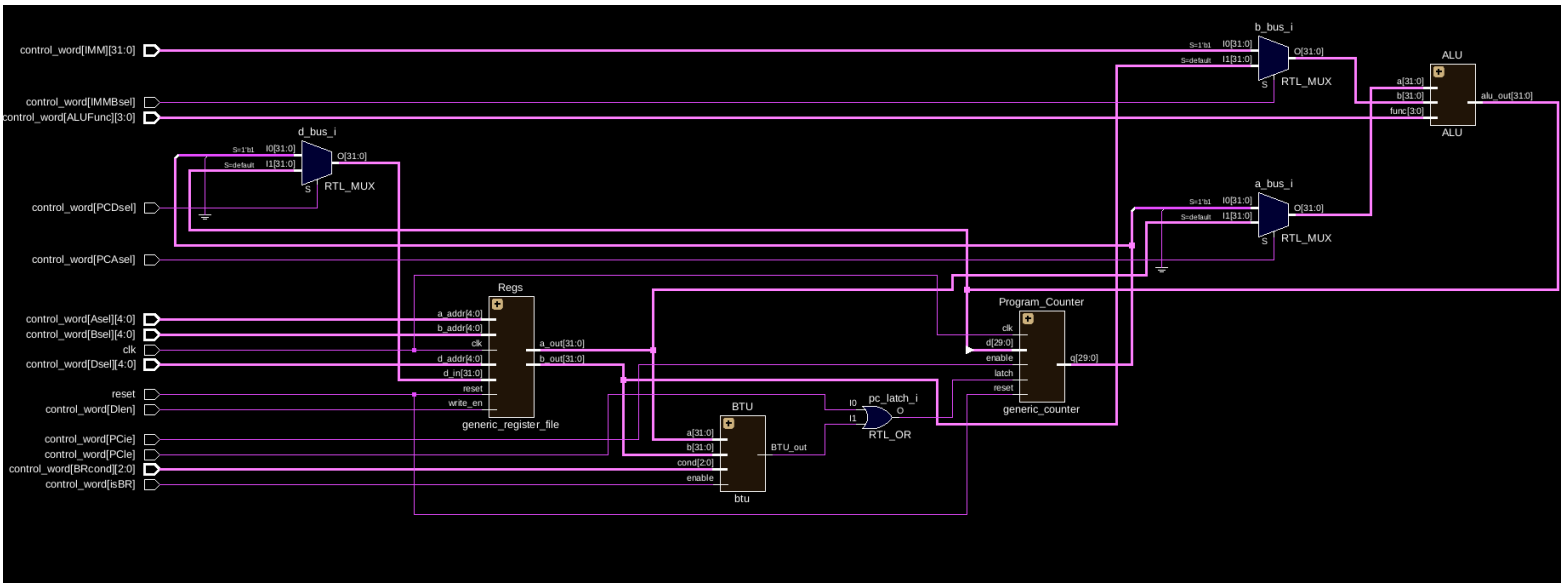
-Overview-

The purpose of this lab was to design and implement the datapath for a basic RISC-V (RV32I) processor core using VHDL. The datapath includes a register file, ALU, program counter, and branch test unit, with their operation controlled by signals generated from what will be the instruction decoder. These signals directed operand selection, data writing, ALU operations, immediate handling, and program counter updates, enabling support for Register-Register, Register-Immediate, Jump, and Branch instructions.

-Design-

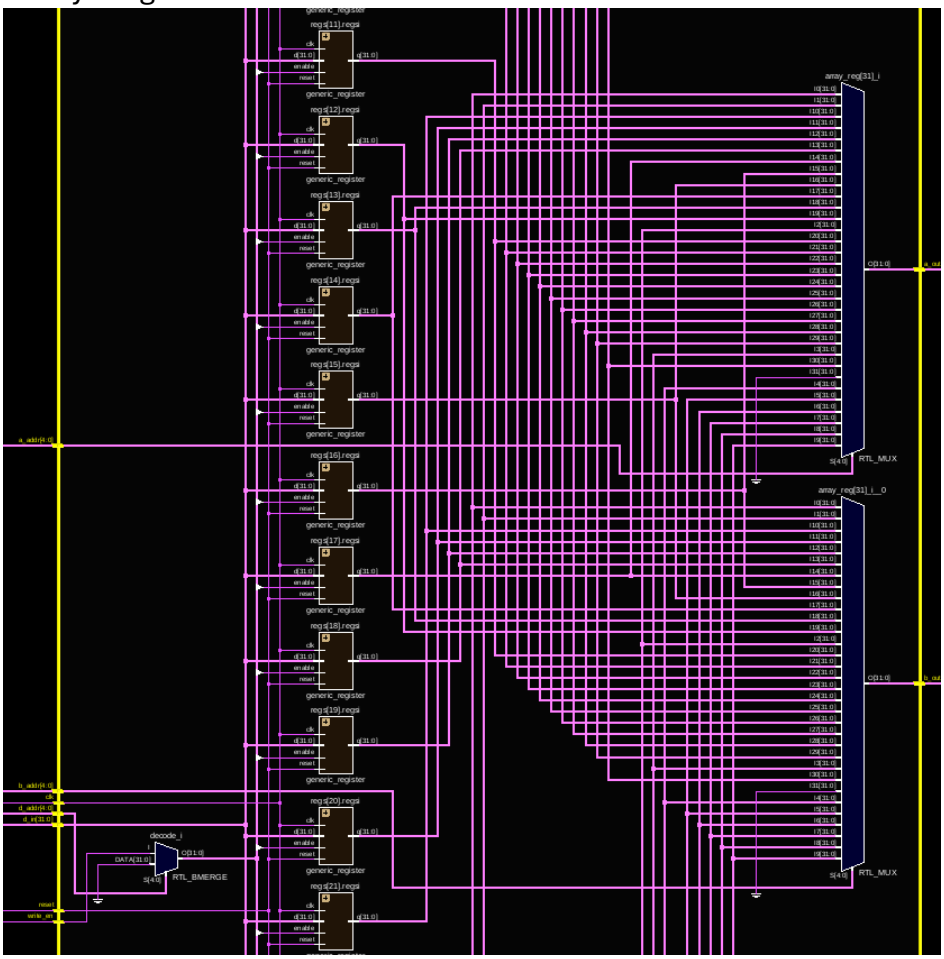
High Level Overview of the Datapath

From this level, you can see that the datapath is controlled by a record type called “control_word”. These signals will both control how the data flows though the datapath, but also what values are being sent in though the A/B register select and Immediate value (IMM). Of interest on this view is: “d_bus_i” selects weather to take the ALU output or PC output to be stored in the register file; “pc_latch_i” that takes either an external signal, or a signal from the Branch Test Unit (BTU) to latch the PC with a new address value; “b_bus_i” selects with the IMM values or value form the B register to support both register-register and register-immediate functions; a_bus_i selects between the PC or A register so math can be done on the PC.



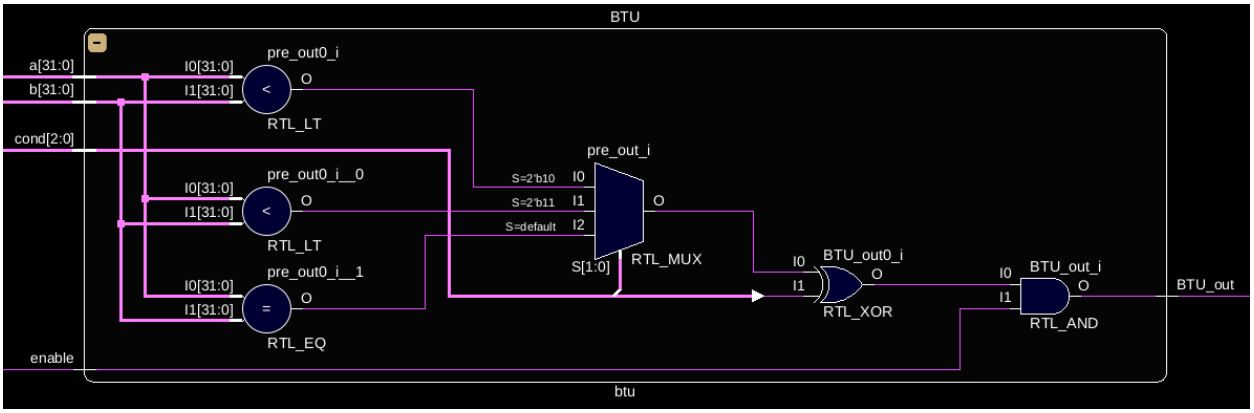
Register File

Nothing significant here is just a standard register file with 2 muxes. Only this is that the 0 register on the mux's are grounded for x0, and the decode value for x0 is not connected to anything.



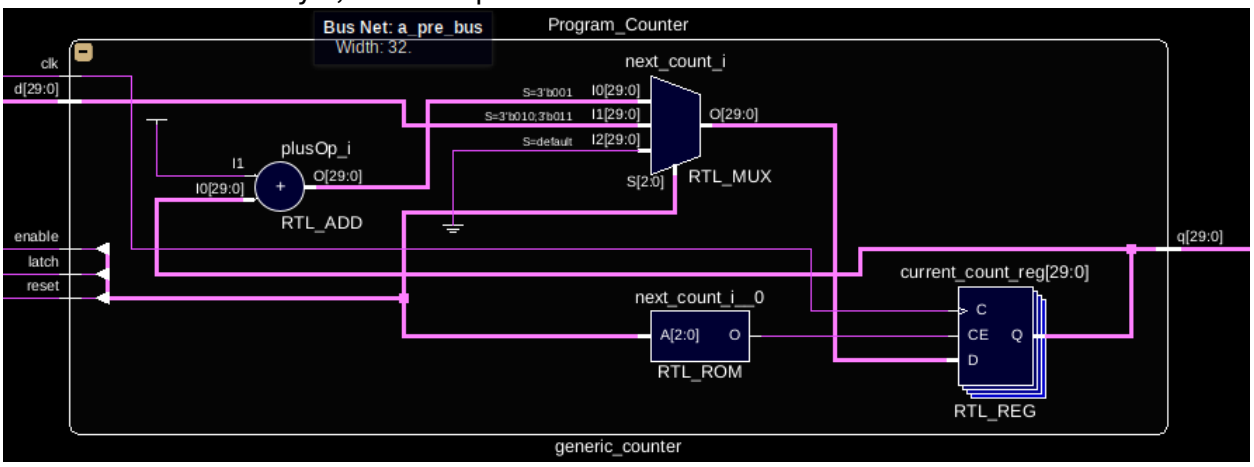
Branch Test Unit (BTU)

The 3 comparators in the front handle =, <, and <=. These signals are selected by mux. Then the value is inverted based on a control bit as != is inverted =, >= is inverted < and > is inverted <=.



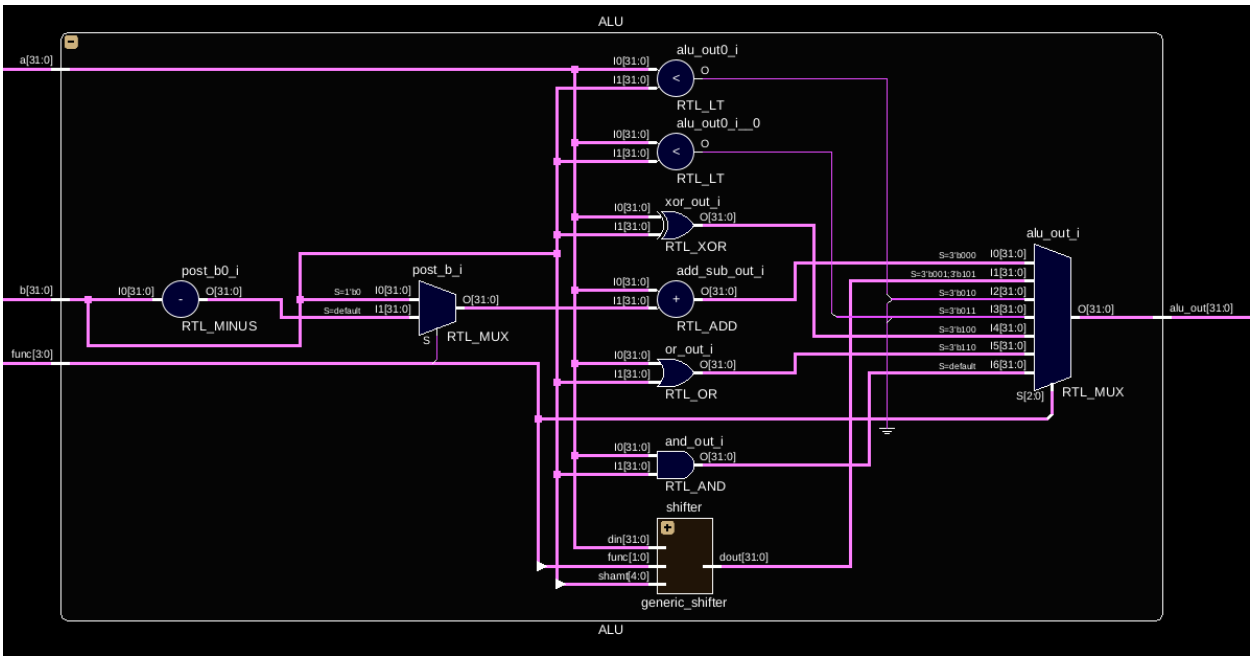
Program Counter (PC)

The PC is only 29 bits, as whenever the PC is used the bottom 2 bits are just grounded. This means that when loading, the bottom 2 bits are also just cut off. When the counter increments by 1, it is interpreted as a increment of 4.



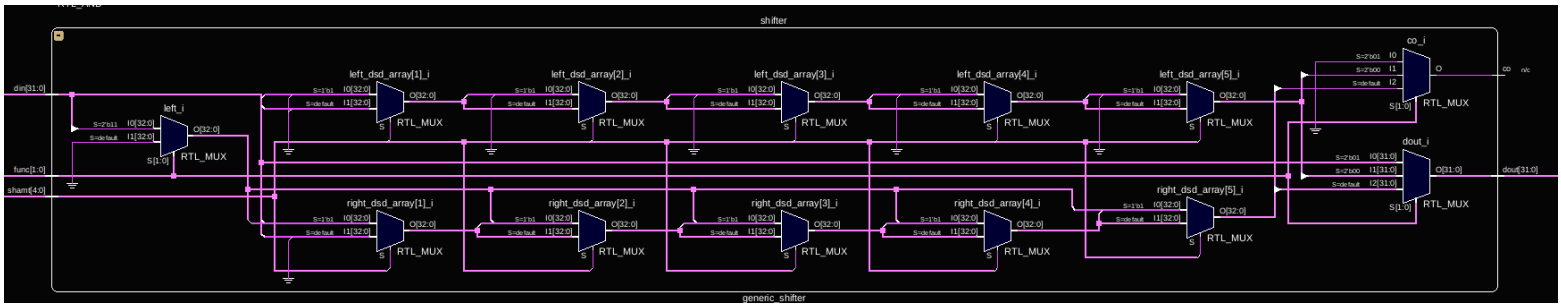
Arithmetic Logic Unit (ALU)

In the center the alu covers: set less than (signed and unsigned), xor, or, and, shift, and add/sub. For subtract, B is 2's complimented and selected by the "post_b_i" mux. The output is selected by the mux on the right controlled directly by the function from the instruction.



Shifter

This is basically 2 barrel shifters: left shift and right shift (top and bottom respectively). For right shift, a mux in the beginning chooses between 0 or the msb for logical and arithmetic shifts. The output alu selects between left, right, and no shift. There is a carry out that could be used for future use if needed, else it can be removed.



-Simulation-

These are all the tests I performed to check the datapath. I tried to have at least one test each to test the functionality of instructions that will be run on the datapath. I have confirmed that all my tests result in correct outputs.

Reset ~ 10 ns: reset and all registers go to 0.

Adding/Subtracting ~ 30 ns: adding immediate to x0 and store into x1.

40ns: try to do same thing but without Dlen selected so no values get stored.

50ns: try to store into x0 register, so nothing happens again.

70ns: add x1 and x2 and store in x31. 80ns: sub x1 and x2 and store in x31.

90ns: sub x1 and immediate value and store in x1.

Shifting (all are stored to x31) ~ 100ns: shift x0 left by immediate (which just stores 0).

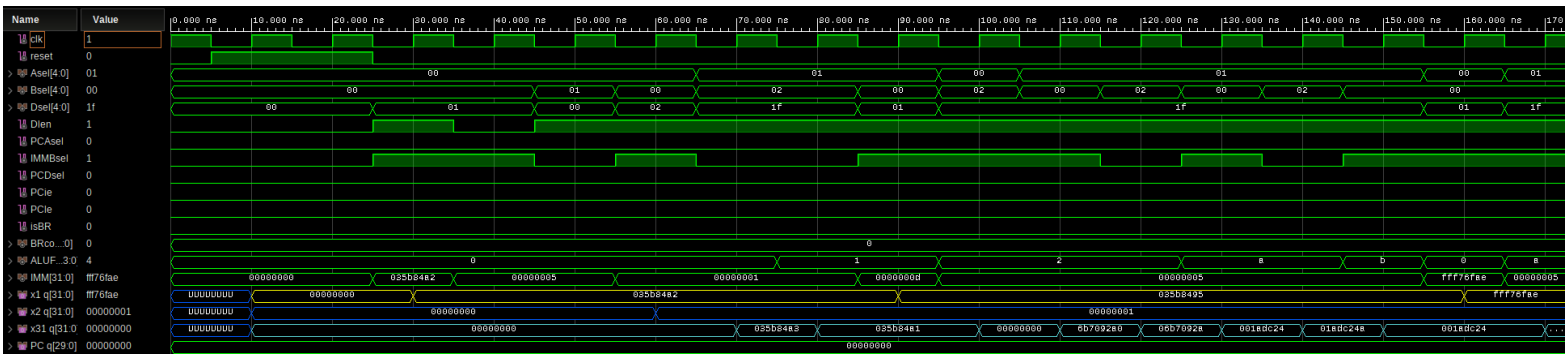
110ns: shift x1 left by immediate (5).

120ns: shift x1 left by value in x2 (1).

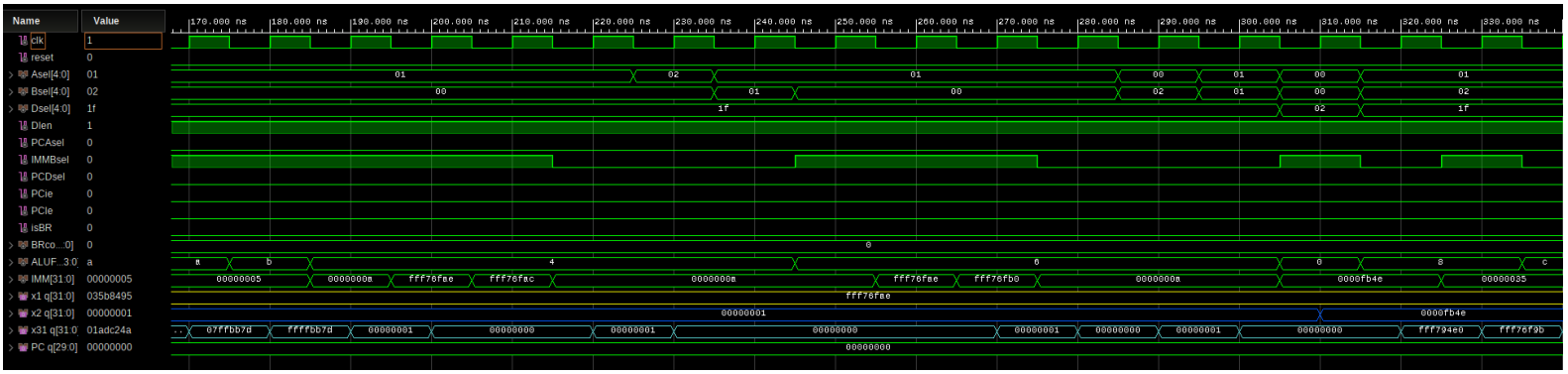
130ns: shift x1 right logical by immediate (5).

140ns: shift x1 right logical by value in x2 (1).

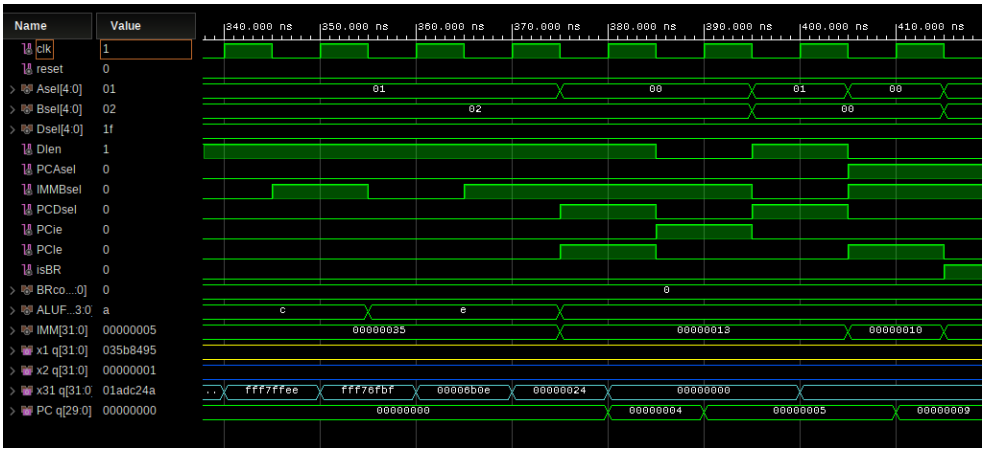
150ns: shift x1 right arithmetic when x1 starts with 0.



Shifting cont. (all are stored to x31) ~ 170ns: shift x1 right logical when x1 starts with a 1.
180ns: shift x1 right arithmetic when x1 starts with a 1.
Set less then (all are stored to x31) ~ 190ns: signed set x1 (-561,234) less than 10.
200ns: signed set x1 (negative) less than -561,234(same value as x1).
210ns: signed set x1 (negative) less than -561,236(less than x1).
220ns: signed set x1 less than x0.
230ns: signed set x2 (1) less than x1.
240ns: signed set x1 less than x1 (0 no matter what).
250ns: unsigned x1(big positive) less than 10.
260ns: unsigned set x1 less than same as x1.
270ns: unsigned set x1 less than a number slight more than x1.
280ns: set unsigned x1 less than x0.
290ns: unsigned set x0 less than x2.
300ns: unsigned set x1 less than x1.
Logical (all are stored to x31) ~ 320ns: x1 xor x2. 330 ns: x1 xor immediate 53.

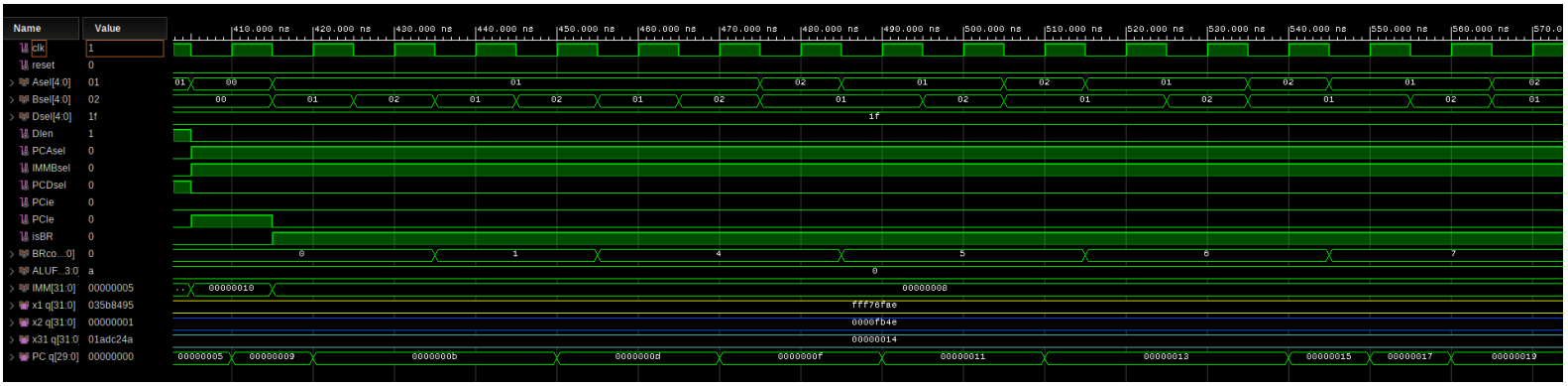


Logical cont. ~ 340ns: x1 or x2. 350ns: x1 or immediate 53.
360ns: x1 and x2. 370ns: x1 and immediate 53.
Program Counter Manipulation ~ 380ns: store 19 into PC (should only store as 16 which will be 4 in the register) and store the PC into x31.
390ns: increment PC (only +1 in the counter itself but will be interpreted as +4).
400ns: PC plus x0 store to x31.
410ns: PC plus immediate 16.



Conditional Branches (branches are too PC + 8) ~ 420ns: x1 equals x1.

430ns: x1 equals x2.
440ns: x1 is not equal to x2.
450ns: x1 is not equal x2.
460ns: signed x1 less than x1.
470ns: signed x1(-) less than x2(+).
480ns: signed x2(+) less than x1(-).
490ns: signed x1 greater than or equal to x1.
500ns: signed x1(-) greater than or equal to x2(+).
510ns: signed x2(+) greater than or equal to x1(-).
520ns: unsigned x1 less than x1.
530ns: unsigned x1(big) less than x2(small).
540ns: unsigned x2(small) less than x1(big).
550ns: unsigned x1 greater than or equal to x1.
560ns: unsigned x1(big) greater than or equal to x2(big).
570ns: unsigned x2(small) greater than or equal to x1(small).



-Conclusion-

Luckily for me, all of my previous parts like the barrel shifter and register file all worked from last year. They only required a little modification to seamlessly fit in the design like making the register 0 a zero register. Other than that, I made a mistake shifting my lefts right and right left, but that was an easy switch in the shifter output mux. I believe I have done a sufficient amount of testing and can be confident that it will function properly when we start bring in the other pieces of the processor.

-Appendix-

Main VHDL

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

package RISCV_package is
constant XLEN_BITS : integer := 5;
constant XLEN : integer := 2**XLEN_BITS;
constant REGS_ADDR_BITS : integer := 5;

type control_word is record
    Asel : std_logic_vector(4 downto 0);
    Bsel : std_logic_vector(4 downto 0);
    Dsel : std_logic_vector(4 downto 0);
    Dlen : std_logic;
    PCAsel : std_logic;
    IMMBsel : std_logic;
    PCDsel : std_logic;
    PCie : std_logic;
    PCle : std_logic; --might not needed
    isBR : std_logic;
    BRcond : std_logic_vector(2 downto 0);
    ALUFunc : std_logic_vector(3 downto 0);
    IMM : std_logic_vector(31 downto 0);
end record control_word;

function decode (signal sel: std_logic_vector; signal enable: std_logic ) return std_logic_vector;
function vector_and ( signal to_and : std_logic_vector ) return std_logic;
end RISCV_package;

package body RISCV_package is

function decode (signal sel: std_logic_vector; signal enable: std_logic ) return std_logic_vector is
    variable decoded : std_logic_vector ((2**sel'length)-1 downto 0);
begin
        decoded := (others => '0');
        decoded( to_integer( unsigned( sel ))) := enable;
        return decoded;
end function;
```

Figure 1: RISC_package

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


entity btu is
    Port ( a : in STD_LOGIC_VECTOR (XLEN-1 downto 0);
           b : in STD_LOGIC_VECTOR (XLEN-1 downto 0);
           cond : in STD_LOGIC_VECTOR (2 downto 0); -- bit 2: 1 = less then: 0 = eq, bit 1: 1 = signed: 2 = signed, bit 0: 1 = invert result
           enable : in STD_LOGIC;
           BTU_out : out STD_LOGIC);
end btu;

architecture Behavioral of btu is
    signal pre_out, equal, lt, slt : std_logic;
begin
    equal <= '1' when a = b else '0';
    lt <= '1' when unsigned(a) <  unsigned(b) else '0';
    slt <= '1' when signed(a) < signed(b) else '0';

    with cond(2 downto 1) select pre_out <=
        slt when "10",
        lt when "11",
        equal when others;

    BTU_out <= (pre_out xor cond(0)) and enable;

end Behavioral;

```

Figure 2: Branch Test Unit

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

entity datapath is
    port (
        clk, reset: in std_logic;
        control_word: in control_word
        );
end datapath;

architecture Behavioral of datapath is
    signal d_bus, a_pre_bus, b_pre_bus, a_bus, b_bus, pc, alu_out : std_logic_vector(XLEN-1 downto 0);
    signal BTU_out, pc_latch : std_logic;

begin

    Regs: entity work.generic_register_file (Behavioral)
    generic map (word_len => XLEN, addr_bits => REGS_ADDR_BITS)
    port map ( clk => clk, reset => reset,
               write_en => control_word.Dlen,
               d_addr =>control_word.Dsel,
               d_in => d_bus,
               a_addr => control_word.Asel,
               a_out => a_pre_bus,
               b_addr => control_word.Bsel,
               b_out => b_pre_bus);

    ALU: entity work.ALU (Behavioral)
    port map ( a => a_bus, b => b_bus, alu_out => alu_out, func => control_word.ALUfunc );

    Program_Counter: entity work.generic_counter (Behavioral)
    generic map ( bits => XLEN - 2)
    port map (clk => clk, reset => reset, latch => pc_latch, enable => control_word.PCie, d => alu_out(31 downto 2), q => pc(31 downto 2));
    pc(1 downto 0) <= "00"; --hard set last 2 bits of pc to 00

    BTU : entity work.BTU (Behavioral)
    port map ( a => a_pre_bus, b => b_pre_bus, cond => control_word.BRcond, enable => control_word.isBR, BTU_out => BTU_out);




    a_bus <= pc when control_word.PCAsel = '1' else
             a_pre_bus;
    b_bus <= control_word.IMM when control_word.IMMBsel = '1' else
             b_pre_bus;
    d_bus <= pc when control_word.PCDsel = '1' else
             alu_out;
    pc_latch <= control_word.PCle or BTU_out; --might not needed. BTU_out might just be PCle


end Behavioral;
```

Figure 3: Datapath

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

entity ALU is
    port (
        a, b: in std_logic_vector(XLEN-1 downto 0);
        alu_out: out std_logic_vector(XLEN-1 downto 0); --bit 0: 1 = right/sub: 0 = left/add, --bit 3 to 1: 000 =
        func : in std_logic_vector(3 downto 0)
        );
end ALU;

architecture Behavioral of ALU is
    signal add_sub_out, shift_out, slt_unsigned_out, slt_signed_out, xor_out, or_out, and_out : std_logic_vector(XLEN-1 downto 0);
    signal post_b: std_logic_vector(XLEN-1 downto 0);
    signal shift_func : std_logic_vector (1 downto 0);
begin
    --add/subtract
    post_b <= b when func(0) = '0' else
              std_logic_vector(-signed(b));
    add_sub_out <= std_logic_vector(unsigned(a) + unsigned(post_b));

    --shift
    shift_func <= func(3) & func(0);
    shifter: entity work.generic_shifter (Behavioral)
    generic map (shamt_bits => XLEN_BITS)
    port map ( din => a,
               dout => shift_out,
               shamt => b(4 downto 0),
               func => shift_func);
               --co =>

    --set less than
    slt_unsigned_out <= std_logic_vector(to_unsigned(1,XLEN)) when unsigned(a) < unsigned(b) else (others => '0');
    slt_signed_out <= std_logic_vector(to_unsigned(1,XLEN)) when signed(a) < signed(b) else (others => '0');

    --Logicals
    xor_out <= a xor b;
    or_out <= a or b;
    and_out <= a and b;


    --out
    with func(3 downto 1) select alu_out <=
        add_sub_out when "000",
        shift_out when "001"|"101",
        slt_signed_out when "010",
        slt_unsigned_out when "011",
        xor_out when "100",
        or_out when "110",
        and_out when others;


end Behavioral;
```

Figure 5: ALU

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


entity generic_register_file is
    generic(
        word_len: integer := 32;
        addr_bits: integer := 5
        );
    port (
        clk, reset: in std_logic;
        write_en: in std_logic;
        d_addr: in std_logic_vector ( addr_bits-1 downto 0 );
        d_in: in std_logic_vector ( word_len-1 downto 0 );
        a_addr: in std_logic_vector ( addr_bits-1 downto 0 );
        a_out : out std_logic_vector ( word_len-1 downto 0 );
        b_addr: in std_logic_vector ( addr_bits-1 downto 0 );
        b_out : out std_logic_vector ( word_len-1 downto 0 )
        );
end generic_register_file;

architecture Behavioral of generic_register_file is
    type signal_array is array ( 2**addr_bits-1 downto 0 ) of std_logic_vector ( word_len-1 downto 0 );
    signal array_reg: signal_array;
    signal decoded : std_logic_vector ( 2**addr_bits-1 downto 0 );
begin
    decoded <= decode( d_addr, write_en );

    regs: for i in 1 to 2**addr_bits - 1 generate --only 2^addr_bits - 1 register (so 31 for 5 addrress) as x0 is going to be grounded
        regsi: entity work.generic_register ( Behavioral )
        generic map ( bits => word_len )
        port map ( clk => clk, reset => reset, enable => decoded(i), d => d_in, q => array_reg(i));
    end generate regs;
    array_reg(0) <= ( others => '0'); --x0 = 0
    a_out <= array_reg ( to_integer( unsigned( a_addr )));
    b_out <= array_reg ( to_integer( unsigned( b_addr )));
end Behavioral;

```

Figure 4: Register File

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;
use IEEE.math_real.ALL;

entity generic_shifter is
    Generic ( shamt_bits : natural := 2 );
    Port ( din : in STD_LOGIC_VECTOR ( (2**shamt_bits)-1 downto 0);
           dout : out STD_LOGIC_VECTOR ((2**shamt_bits)-1 downto 0);
           shamt : in STD_LOGIC_VECTOR (shamt_bits - 1 downto 0);
           func : in STD_LOGIC_vector (1 downto 0); --01 = no shift, 00 = left, 10 = right loigcal, 11 right arithmetic
           co : out STD_LOGIC);
end generic_shifter;

architecture Behavioral of generic_shifter is
    type t_array is array ( integer range <> ) of std_logic_vector( (2**shamt_bits) downto 0 );
    signal right_dsd_array : t_array (shamt_bits downto 0);
    signal left_dsd_array : t_array (shamt_bits downto 0);
    signal left : std_logic_vector ( (2**shamt_bits) downto 0);
    signal right : std_logic_vector ( (2**shamt_bits) downto 0);

begin
    left_dsd_array(0) <= '0' & din;
    right_dsd_array(0) <= din & '0';
    with func select left <=
        (others => din( (2**shamt_bits)-1 ) ) when "11",
        (others => '0') when others;
    right <= (others => '0');

    left_shift_loop: for i in shamt_bits-1 downto 0 generate
        left_dsd_array(i+1) <=  left_dsd_array(i)(((2**shamt_bits)-(2**i)) downto 0) & right (2**i-1 downto 0) when shamt(i) = '1' else left_dsd_array(i);
    end generate left_shift_loop;

    right_shift_loop: for i in shamt_bits-1 downto 0 generate
        right_dsd_array(i+1) <=  left(2**i-1 downto 0) &  right_dsd_array(i)((2**shamt_bits) downto (2**i) ) when shamt(i) = '1' else  right_dsd_array(i);
    end generate right_shift_loop;

    with func select dout <=
        din when "01",
        left_dsd_array(shamt_bits)((2**shamt_bits)-1 downto 0) when "00",
        right_dsd_array(shamt_bits)((2**shamt_bits) downto 1) when others;

    with func select co <=
        '0' when "01",
        left_dsd_array(shamt_bits)(2**shamt_bits) when "00",
        right_dsd_array(shamt_bits)((0)) when others;
end Behavioral;
```

Figure 7: Barrel Shifter

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;


entity generic_counter is
    generic ( bits : integer := 4 );
    Port ( clk, reset, latch, enable : in STD_LOGIC;
           d : in std_logic_vector ( bits-1 downto 0);
           q : out STD_LOGIC_VECTOR ( bits-1 downto 0 ));
end generic_counter;

architecture Behavioral of generic_counter is
    signal current_count : std_logic_vector ( bits-1 downto 0 ) := ( others => '0' );
    signal next_count : std_logic_vector ( bits-1 downto 0 ) := ( others => '0' );
    signal ctrl : std_logic_vector ( 2 downto 0 );
begin
    --declare memory
    current_count <= next_count when rising_edge( clk );
    --next state logic
    ctrl <= reset & latch & enable;
    with ctrl select next_count <=
        current_count when "000",
        std_logic_vector ( unsigned ( current_count ) + 1 ) when "001",
        d when "010"|"011",
        (others => '0') when others;

    --output logic
    q <= current_count;

end Behavioral;
```

Figure 6: Generic Counter

Testbench

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


entity datapath_tb is
end datapath_tb;

architecture Behavioral of datapath_tb is
    signal clk: std_logic := '1';
    signal reset: std_logic := '0';
    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'; --might not needed
    signal isBR : std_logic:= '0';
    signal BRcond : std_logic_vector(2 downto 0):= "000";
    signal ALUFunc : std_logic_vector(3 downto 0):= "0000";
    signal IMM : std_logic_vector(31 downto 0) := "00000000000000000000000000000000";
begin

  uut: 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;
```

```
test: process
  begin
   wait for 5 ns;
    --reset in beginning
    reset <= '1';
    wait for 20 ns;
    reset <= '0';

    --add x0 + 56329378 -> x1
    Asel    <= "00000";
    Bsel    <= "00000";
    Dsel    <= "00001";
    Dlen    <= '1';
    PCAsel  <= '0';
    IMMBsel <= '1';
    PCDsel  <= '0';
    PCie    <= '0';
    PCle    <= '0';
    isBR    <= '0';
    BRcond  <= "000";
    ALUFunc <= "0000";
    IMM     <= std_logic_vector(to_unsigned(56329378,32));
    wait for 10 ns;

    --add x0 + 5 -> x1 (but no Dlen so nothing happens)
    Dlen    <= '0';
    IMM     <= std_logic_vector(to_unsigned(5,32));
    wait for 10 ns;

    --add x0 + x1 -> x0 (shouldn't do anything)
    Bsel    <= "00001";
    Dsel    <= "00000";
    Dlen    <= '1';
    IMMBsel <= '0';
    wait for 10 ns;
```

```
    --add x0 + 1 -> x2
    Bsel    <= "00000";
    Dsel    <= "00010";
    IMMBsel <= '1';
    IMM     <= std_logic_vector(to_unsigned(1,32));
    wait for 10 ns;

    --add x1 + x2 -> x31
    Asel    <= "00001";
    Bsel    <= "00010";
    Dsel    <= "11111";
    IMMBsel <= '0';
    wait for 10 ns;

    --sub x1 - x2 -> x31
    ALUFunc <= "0001";
    wait for 10 ns;

    --sub x1 - 13 -> x1
    Bsel    <= "00000";
    Dsel    <= "00001";
    IMMBsel <= '1';
    IMM     <= std_logic_vector(to_unsigned(13,32));
    wait for 10 ns;

    --shift x0 left 5 -> x31 (should be 0) --100 ns
    Asel    <= "00000";
    Bsel    <= "00010";
    Dsel    <= "11111";
    ALUFunc <= "0010";
    IMM     <= std_logic_vector(to_unsigned(5,32));
    wait for 10 ns;
```

```
118         --shift x1 left 5 -> x31
119         Asel    <= "00001";
120         Bsel    <= "00000";
121         wait for 10 ns;
122
123         --shift x1 left x2 -> x31
124         Bsel    <= "00010";
125         IMMBsel <= '0';
126         wait for 10 ns;
127
128         --shift x1 right 5 -> x31
129         Bsel    <= "00000";
130         IMMBsel <= '1';
131         ALUFunc <= "1010";
132         wait for 10 ns;
133
134         --shift x1 right x2 -> x31
135         Bsel    <= "00010";
136         IMMBsel <= '0';
137         wait for 10 ns;
138
139         --shift x1 right arithmetic 5 -> x31
140         Bsel    <= "00000";
141         IMMBsel <= '1';
142         ALUFunc <= "1011";
143         wait for 10 ns;
144
145         --load number with 1 in msb in x1
146         Asel    <= "00000";
147         Dsel    <= "00001";
148         ALUFunc <= "0000";
149         IMM     <= std_logic_vector(to_signed(-561234,32));
150         wait for 10 ns;
151
152         --shift x1 right logical 5 -> x31
153         Asel    <= "00001";
154         Dsel    <= "11111";
155         ALUFunc <= "1010";
156         IMM     <= std_logic_vector(to_unsigned(5,32));
157         wait for 10 ns;
158
159         --shift x1 right arithmetic 5 -> x31
160         ALUFunc <= "1011";
161         wait for 10 ns;
162
163         --set less than signed x1 < 10 -> x31
164         ALUFunc <= "0100";
165         IMM     <= std_logic_vector(to_signed(10,32));
166         wait for 10 ns;
167
168         --set less than signed x1 < -561234 -> x31 --200 ns
169         IMM     <= std_logic_vector(to_signed(-561234,32));
170         wait for 10 ns;
171
172         --set less than signed x1 < -561236 -> x31
173         IMM     <= std_logic_vector(to_signed(-561236,32));
174         wait for 10 ns;
175
176         --set less than signed x1 < x0 -> x31
177         IMMBsel <= '0';
178         IMM     <= std_logic_vector(to_signed(10,32));
179         wait for 10 ns;
180
181         --set less than signed x2 < x0 -> x31
182         Asel    <= "00010";
183         wait for 10 ns;
184
185         --set less than signed x1 < x1 -> x31
186         Asel    <= "00001";
187         Bsel    <= "00001";
188         wait for 10 ns;
189
```

```
190         --set less than unsigned x1 < 10 -> x31 --250 ns
191         Bsel    <= "00000";
192         IMMBsel <= '1';
193         ALUFunc <= "0110";
194         IMM     <= std_logic_vector(to_unsigned(10,32));
195         wait for 10 ns;
196
197         --set less than unsigned x1 < 4294406062 -> x31
198         IMM     <= std_logic_vector(to_signed(-561234,32));
            --just leaving as signed so its same value
199         wait for 10 ns;
200
201         --set less than unsigned x1 < -4294406064 -> x31
202         IMM     <= std_logic_vector(to_signed(-561232,32));
            --is greater than -561234 when interpreted unsigned
203         wait for 10 ns;
204
205         --set less than unsigned x1 < x0 -> x31
206         IMMBsel <= '0';
207         IMM     <= std_logic_vector(to_signed(10,32));
208         wait for 10 ns;
209
210         --set less than unsigned x0 < x2 -> x31
211         Asel    <= "00000";
212         Bsel    <= "00010";
213         wait for 10 ns;
214
215         --set less than unsigned x1 < x1 -> x31 --300 ns
216         Asel    <= "00001";
217         Bsel    <= "00001";
218         wait for 10 ns;
219
220         --add x0 + 64334 -> x2
221         Asel    <= "00000";
222         Bsel    <= "00000";
223         Dsel    <= "00010";
224         IMMBsel <= '1';
225         ALUFunc <= "0000";
226         IMM     <= std_logic_vector(to_unsigned(64334,32));
227         wait for 10 ns;
228
229         -- x1 xor x2 -> x31
230         Asel    <= "00001";
231         Bsel    <= "00010";
232         Dsel    <= "11111";
233         IMMBsel <= '0';
234         ALUFunc <= "1000";
235         wait for 10 ns;
236
237         -- x1 xor 53 -> x31
238         IMMBsel <= '1';
239         IMM     <= std_logic_vector(to_unsigned(53,32));
240         wait for 10 ns;
241
242         -- x1 or x2 -> x31
243         IMMBsel <= '0';
244         ALUFunc <= "1100";
245         wait for 10 ns;
246
247         -- x1 or 53 -> x31 --350 ns
248         IMMBsel <= '1';
249         wait for 10 ns;
250
251         -- x1 and x2 -> x31
252         IMMBsel <= '0';
253         ALUFunc <= "1110";
254         wait for 10 ns;
255
256         -- x1 and 53 -> x31
257         IMMBsel <= '1';
258         wait for 10 ns;
259
```

```
        -- add x0 and 19 -> PC (should only be 16 as takes of bottom 2 bits) and store PC -> x31
        Asel    <= "00000";
        PCDsel  <= '1';
        PCle    <= '1';
        ALUFunc <= "0000";
        IMM     <= std_logic_vector(to_unsigned(19,32));
        wait for 10 ns;

        -- increment PC
        Dlen    <= '0';
        PCDsel  <= '0';
        PCie    <= '1';
        PCle    <= '0';
        wait for 10 ns;

        -- PC + x0 -> x31 --400 ns
        Asel    <= "00001";
        Bsel    <= "00000";
        Dlen    <= '1';
        IMMBsel <= '0';
        PCDsel  <= '1';
        PCie    <= '0';
        wait for 10 ns;

        -- PC + 16 -> PC
        Asel    <= "00000";
        Dlen    <= '0';
        PCAsel  <= '1';
        IMMBsel <= '1';
        PCDsel  <= '0';
        PCle    <= '1';
        IMM     <= std_logic_vector(to_unsigned(16,32));
        wait for 10 ns;

        --branch to PC + 8 if x1 equal x1
        Asel    <= "00001";
        Bsel    <= "00001";
        PCle    <= '0';
        isBR    <= '1';
        IMM     <= std_logic_vector(to_unsigned(8,32));
        wait for 10 ns;

        --branch to PC + 8 if x1 equal x2
        Bsel    <= "00010";
        wait for 10 ns;

        --branch to PC + 8 if x1 not equal x1
        Bsel    <= "00001";
        BRcond  <= "001";
        wait for 10 ns;

        --branch to PC + 8 if x1 not equal x2 --450 ns
        Bsel    <= "00010";
        wait for 10 ns;

         --branch to PC + 8 if x1 less than x1 (signed)
        Bsel    <= "00001";
        BRcond  <= "100";
        wait for 10 ns;

        --branch to PC + 8 if x1 less than x2 (signed)
        Bsel    <= "00010";
        wait for 10 ns;

        --branch to PC + 8 if x2 less than x1 (signed)
        Asel    <= "00010";
        Bsel    <= "00001";
        wait for 10 ns;
```

```
        --branch to PC + 8 if x1 greater than or equal to x1 (signed)
        Asel    <= "00001";
        BRcond  <= "101";
        wait for 10 ns;

        --branch to PC + 8 if x1 greater than or equal to x2 (signed) --500 ns
        Bsel    <= "00010";
        wait for 10 ns;

        --branch to PC + 8 if x2 greater than or equal to x1 (signed)
        Asel    <= "00010";
        Bsel    <= "00001";
        wait for 10 ns;

        --branch to PC + 8 if x1 less than x1 (unsigned)
        Asel    <= "00001";
        BRcond  <= "110";
        wait for 10 ns;

        --branch to PC + 8 if x1 less than x2 (unsigned)
        Bsel    <= "00010";
        wait for 10 ns;

        --branch to PC + 8 if x2 less than x1 (unsigned)
        Asel    <= "00010";
        Bsel    <= "00001";
        wait for 10 ns;

        --branch to PC + 8 if x1 greater than or equal to x1 (unsigned) --550 ns
        Asel    <= "00001";
        BRcond  <= "111";
        wait for 10 ns;

        --branch to PC + 8 if x1 greater than or equal to x2 (unsigned)
        Bsel    <= "00010";
        wait for 10 ns;

        --branch to PC + 8 if x2 greater than or equal to x1 (unsigned)
        Asel    <= "00010";
        Bsel    <= "00001";
        wait for 10 ns;

        wait;
    end process;

end Behavioral;
```