**Nano Processor Design Competition**

**CS 2052 - Computer Architecture**\

# Team Members – Runtime Terror

Name of the group - Runtime Terror

|  |  |
| --- | --- |
| **Name** | **Index No.** |
| Liyanapathirana S.T. | 190359P |
| Nimesh Ariyaratne | 190050K |
| Heshan K.D.Y | 190222P |

# Introduction

We were asked to design a 4-bit processor which is capable of executing 4 instructions add, subtract, jump and negation.

This designed processor can ,

* Add and subtract signed integers
* Decode instructions to activate necessary components on the processor
* Be verified their functionality via simulation and on the development board

Since this was a team project, my team consisted of 3 students; Yasith, Nimesh and Me. As a group, we wanted to address key points which we learnt in the lectures and previous labs. Since we cannot meet physically due to the Covid-19 pandemic, we utilized zoom meetings to get to know each other. In the first meeting on 26 March 2021, we threw out ideas on what we were thinking regarding the project and split up who was going to cover which aspects of the project. Five days before the due date, we completed all the tasks which were assigned.

Overall, we think our project was a huge success. We achieved our goal of achieving the key aspects of the microprocessor and did so in a fun and interactive way.

# 4 bit add/sub

This unit is capable of adding and subtracting numbers represented using 2’s complement. To implement this unit 4-bit RCA from Lab 3 is used.

### VHDL Code of 4 bits Add/Subunit

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY Add\_Sub\_unit IS

    PORT (

        AA : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0); --first number

        BB : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0); --second number

        SS : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0); --get result on this

        Sel : IN STD\_LOGIC; --Select add or substract using this

        Zero : OUT STD\_LOGIC; -- Zero flag

        Overflow : OUT STD\_LOGIC; --Overflow flag

        Carry : OUT STD\_LOGIC; -- carry flag

        Negative : OUT STD\_LOGIC --Negative flag

    );

END Add\_Sub\_unit;

ARCHITECTURE Behavioral OF Add\_Sub\_unit IS

    -- Implemented using the Full Adder which created in Lab 3

    COMPONENT FA --defining full adder circuit with its ports

        PORT (

            A : IN STD\_LOGIC;

            B : IN STD\_LOGIC;

            C\_in : IN STD\_LOGIC;

            S : OUT STD\_LOGIC;

            C\_out : OUT STD\_LOGIC);

    END COMPONENT;

    SIGNAL FA0\_C, FA1\_C, FA2\_C, FA3\_C : STD\_LOGIC;

    --These are used to keep values for use in logical operations

    SIGNAL BBU : STD\_LOGIC\_VECTOR(3 DOWNTO 0);

    SIGNAL SSU : STD\_LOGIC\_VECTOR(3 DOWNTO 0);

    SIGNAL C : STD\_LOGIC; -- used to keep final value of C\_out

    SIGNAL Overflow\_flag : STD\_LOGIC; -- used to keep overflow flag value

BEGIN

    FA\_0 : FA --full adder 01

    PORT MAP(

        A => AA(0),

        B => BBU(0),

        C\_in => Sel, -- Get selector value to this

        S => SSU(0),

        C\_Out => FA0\_C);

    FA\_1 : FA -- full adder 02

    PORT MAP(

        A => AA(1),

        B => BBU(1),

        C\_in => FA0\_C, --input carry bit from FA\_0

        S => SSU(1),

        C\_Out => FA1\_C);

    FA\_2 : FA --full adder 03

    PORT MAP(

        A => AA(2),

        B => BBU(2),

        C\_in => FA1\_C, --input carry bit from FA\_1

        S => SSU(2),

        C\_Out => FA2\_C);

    FA\_3 : FA --full adder 04

    PORT MAP(

        A => AA(3),

        B => BBU(3),

        C\_in => FA2\_C, --input carry bit from FA\_2

        S => SSU(3),

        C\_Out => C);

    -- If SEL = 1, we have B XOR 1 = B and C\_out = 1 == perform as a subtractor

    -- If SEL = 0, we have B XOR 1 = Not(B) and C\_in = 0  == perform as an adder

    BBU(0) <= Sel XOR BB(0);

    BBU(1) <= Sel XOR BB(1);

    BBU(2) <= Sel XOR BB(2);

    BBU(3) <= Sel XOR BB(3);

    SS <= SSU;--outputs

    --This get only overflow, not sign in minus numbers

    Overflow\_flag <= ((AA(3) AND BBU(3) AND NOT(SSU(3))) OR (NOT(AA(3)) AND NOT(BBU(3)) AND SSU(3)));

    Overflow <= Overflow\_flag;

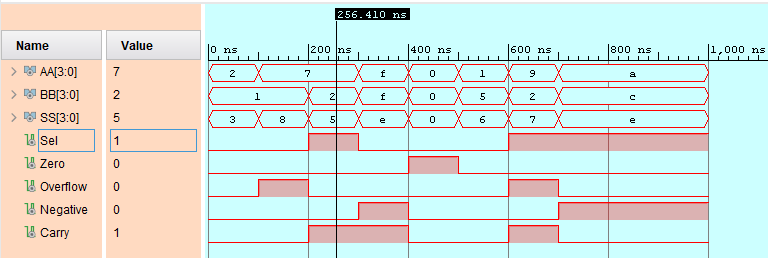
    Zero <= NOT(SSU(0) OR SSU(1) OR SSU(2) OR SSU(3) OR C); --This is the zero flag

    Negative <= SSU(3) AND NOT(Overflow\_flag); -- This is the negative flag

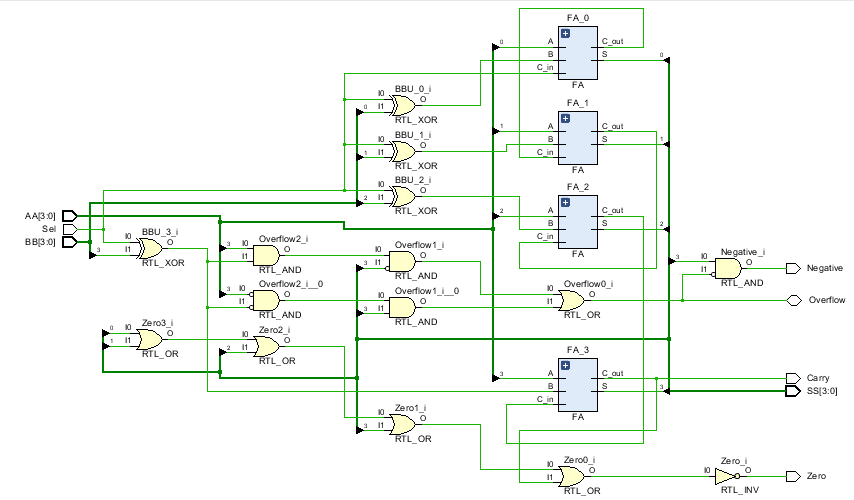
    Carry <= C; --This is the carry flag

END Behavioral;

### Timing diagram of 4 bit Add/Subunit



### Elaborated design of 4 bit Add/Subunit



# 3-bit adder

This unit is used to increment the program counter

### VHDL Code of 3-bit ADDER

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY Adder\_3\_bit IS

    PORT (

        AA : IN STD\_LOGIC\_VECTOR (2 DOWNTO 0); --first number

        SS : OUT STD\_LOGIC\_VECTOR (2 DOWNTO 0) --get result on this

    );

END Adder\_3\_bit;

ARCHITECTURE Behavioral OF Adder\_3\_bit IS

    COMPONENT FA --defining full adder circuit with its ports

        PORT (

            A : IN STD\_LOGIC;

            B : IN STD\_LOGIC;

            C\_in : IN STD\_LOGIC;

            S : OUT STD\_LOGIC;

            C\_out : OUT STD\_LOGIC);

    END COMPONENT;

    SIGNAL FA0\_C, FA1\_C, FA2\_C : STD\_LOGIC;

BEGIN

    FA\_0 : FA --full adder 01

    PORT MAP(

        A => AA(0),

        B => '1', --add 1

        C\_in => '0', -- Get selector value to this

        S => SS(0),

        C\_Out => FA0\_C);

    FA\_1 : FA -- full adder 02

    PORT MAP(

        A => AA(1),

        B => '0',

        C\_in => FA0\_C, --input carry bit from FA\_0

        S => SS(1),

        C\_Out => FA1\_C);

    FA\_2 : FA --full adder 04

    PORT MAP(

        A => AA(2),

        B => '0',

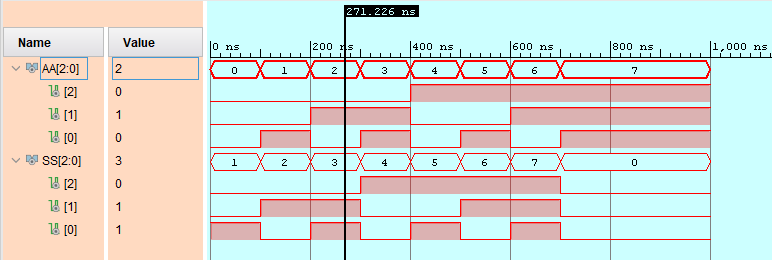
        C\_in => FA1\_C, --input carry bit from FA\_2

        S => SS(2),

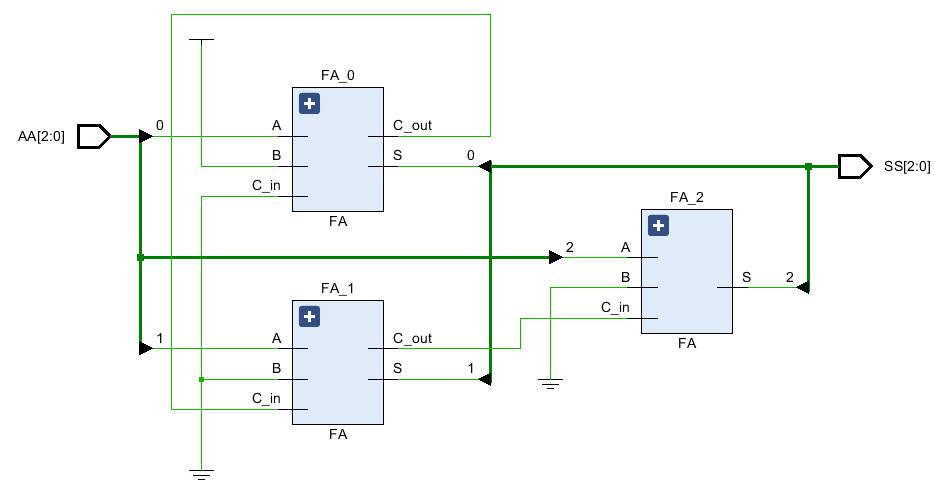
        C\_Out => FA2\_C);

END Behavioral;

### Timing diagram of 3-bit ADDER



### Elaborated design of 3-bit ADDER



# 3-bit Program Counter (PC)

Program counter is used to point out the address of the next instruction to be executed from memory. Program counter needs to be reset to 0 when required. Hence we build it using D Flip Flops with a clear/reset input from Lab 5

### VHDL Code of 3-bit program counter

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY Counter IS

    PORT (

        Next\_Ins : IN STD\_LOGIC\_VECTOR(2 DOWNTO 0); --Next INstruction

        Res : IN STD\_LOGIC; --Reset

        Clk : IN STD\_LOGIC; --Clock

        Current\_Ins : OUT STD\_LOGIC\_VECTOR(2 DOWNTO 0)); -- Current Instruction

END Counter;

ARCHITECTURE Behavioral OF Counter IS

    -- Add D Flip Flops to the Program counter

    -- Implemented using previous Lab

    COMPONENT D\_FF

        PORT (

            D : IN STD\_LOGIC;

            Res : IN STD\_LOGIC;

            Clk : IN STD\_LOGIC;

            Q : OUT STD\_LOGIC;

            Qbar : OUT STD\_LOGIC

        );

    END COMPONENT;

    -- Add Slow Clock to the program counter

    -- Implemented using previous Lab

    COMPONENT Slow\_Clock

        PORT (

            Clk\_In : IN STD\_LOGIC;

            Clk\_Out : OUT STD\_LOGIC

        );

    END COMPONENT;

    SIGNAL D0, D1, D2 : STD\_LOGIC;

    SIGNAL Clk\_slow : STD\_LOGIC;

BEGIN

    Slow\_clock0 : Slow\_Clock --Slow Clock

    PORT MAP(

        Clk\_In => Clk,

        Clk\_Out => Clk\_slow

    );

    D\_FF0 : D\_FF --First D-Flip Flop

    PORT MAP(

        D => Next\_Ins(0),

        Res => Res,

        Clk => Clk\_slow,

        Q => Current\_Ins(0)

    );

    D\_FF1 : D\_FF --Second D-Flip Flop

    PORT MAP(

        D => Next\_Ins(1),

        Res => Res,

        Clk => Clk\_slow,

        Q => Current\_Ins(1)

    );

    D\_FF2 : D\_FF --Third D-Flip Flop

    PORT MAP(

        D => Next\_Ins(2),

        Res => Res,

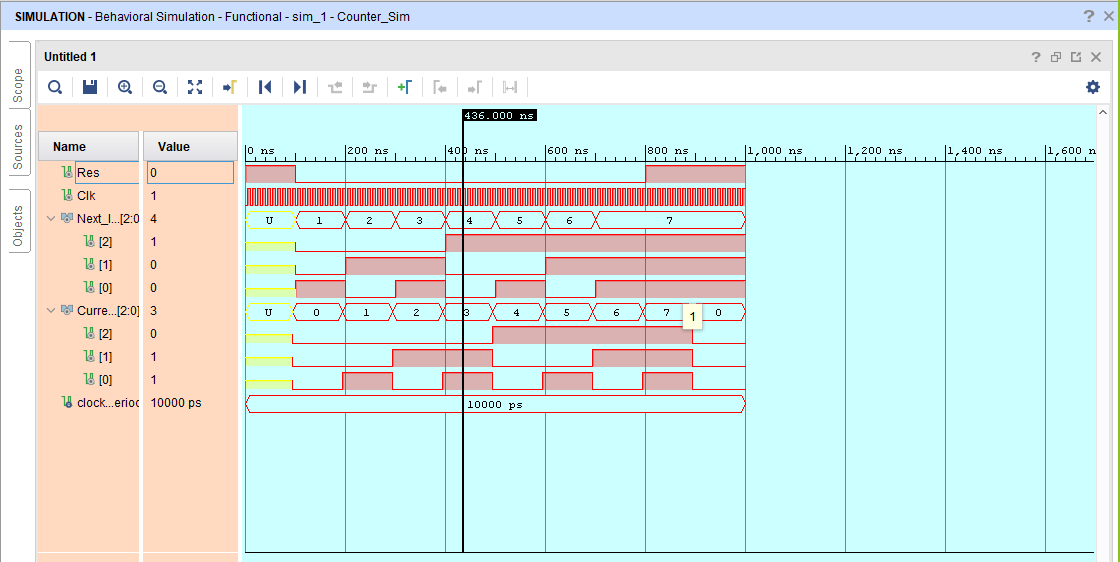
        Clk => Clk\_slow,

        Q => Current\_Ins(2)

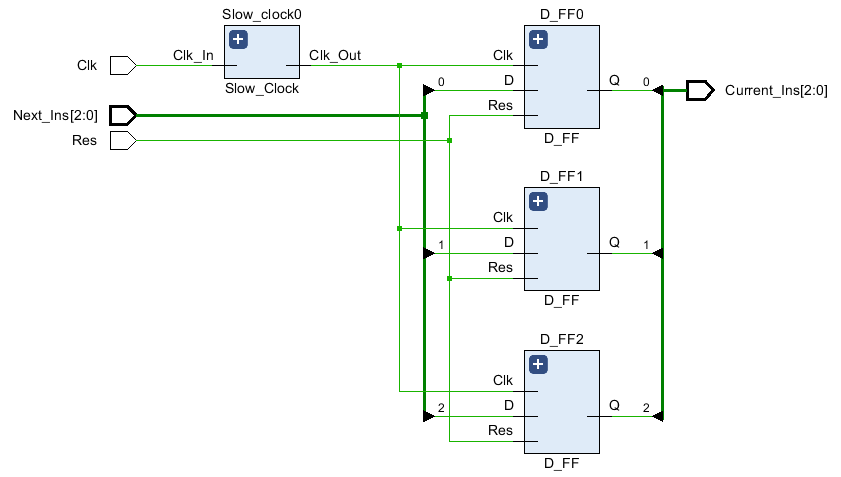
    );

END Behavioral;

### Timing diagram of 3-bit program counter



### Elaborated design of 3-bit program counter



# k-way b-bit multiplexers

A k-way b-bit multiplexer can take in k-inputs, each with b-bits, rather than a single bit, and the output is a group of b-bits. There are log2 k control bits, and these control bits are used to select one of the k groups of b bits rather than a single bit. We have implemented the component using an 8-to-1 multiplexer developed in Lab 4 o Alternatively, instead of multiplexers, we used tri-state buffers.

### VHDL Codes of 2-way 3-bit multiplexers

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY MUX\_2\_way\_3\_bit IS

    PORT (

        Adder\_3 : IN STD\_LOGIC\_VECTOR (2 DOWNTO 0);

        JUMP\_TO : IN STD\_LOGIC\_VECTOR (2 DOWNTO 0);

        Selector : IN STD\_LOGIC;

        Output : OUT STD\_LOGIC\_VECTOR(2 DOWNTO 0));

END MUX\_2\_way\_3\_bit;

ARCHITECTURE Behavioral OF MUX\_2\_way\_3\_bit IS

    COMPONENT tri\_state\_buffer\_3bit

        PORT (

            inputTri : IN STD\_LOGIC\_VECTOR (2 DOWNTO 0);

            outputTri : OUT STD\_LOGIC\_VECTOR (2 DOWNTO 0);

            EN : IN STD\_LOGIC);

    END COMPONENT;

    SIGNAL NOTSel : STD\_LOGIC;

BEGIN

    tri\_state\_buffer\_3bit\_0 : tri\_state\_buffer\_3bit

    PORT MAP(

        inputTri => Adder\_3,

        outputTri => Output,

        EN => NOTSel);

    tri\_state\_buffer\_3bit\_1 : tri\_state\_buffer\_3bit

    PORT MAP(

        inputTri => JUMP\_TO,

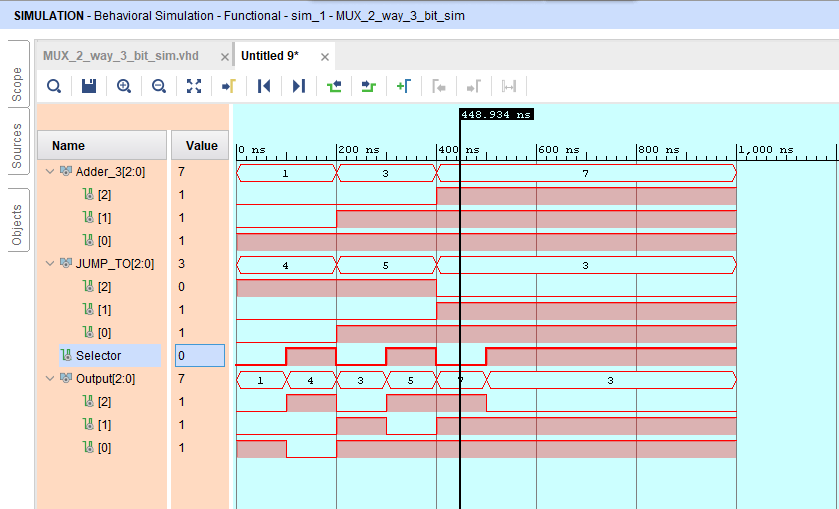
        outputTri => Output,

        EN => Selector);

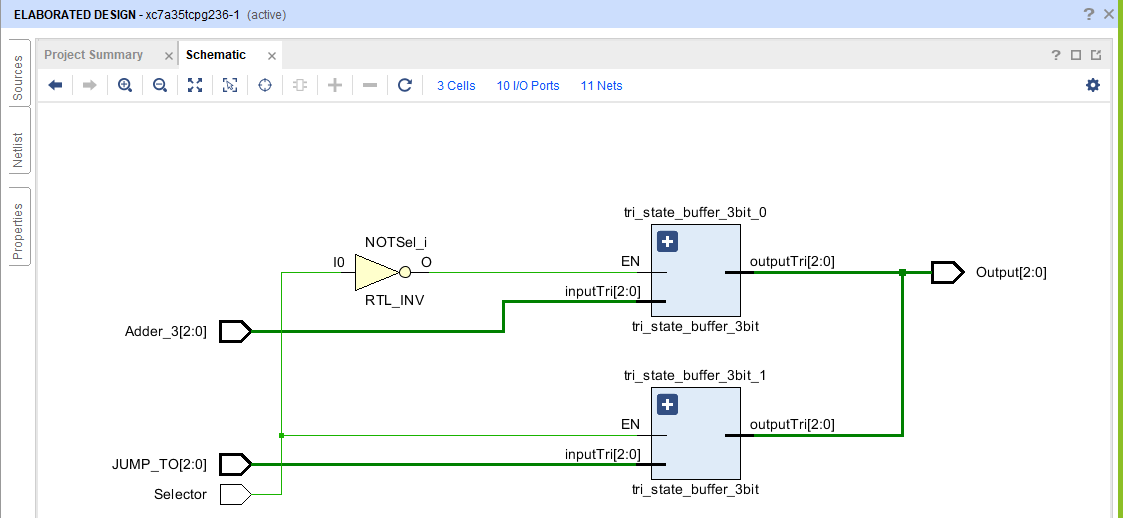
    NOTSel <= NOT Selector;

END Behavioral;

### Timing diagram of 2-way 3-bit multiplexers



### Elaborated design of 2-way 3-bit multiplexers



### VHDL Codes of 2-way 4-bit multiplexers

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY MUX\_2\_way\_4\_bit IS

    PORT (

        AddSubValue : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

        InsDecValue : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

        OutputValue : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0);

        Selector : IN STD\_LOGIC);--'0' for addsub and '1' for insdec

END MUX\_2\_way\_4\_bit;

ARCHITECTURE Behavioral OF MUX\_2\_way\_4\_bit IS

    COMPONENT tri\_state\_buffer

        PORT (

            inputTri : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

            outputTri : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0);

            EN : IN STD\_LOGIC);

    END COMPONENT;

    SIGNAL NOTSel : STD\_LOGIC;

BEGIN

    tri\_state\_buffer\_0 : tri\_state\_buffer

    PORT MAP(

        inputTri => AddSubValue,

        outputTri => OutputValue,

        EN => NOTSel);

    tri\_state\_buffer\_1 : tri\_state\_buffer

    PORT MAP(

        inputTri => InsDecValue,

        outputTri => OutputValue,

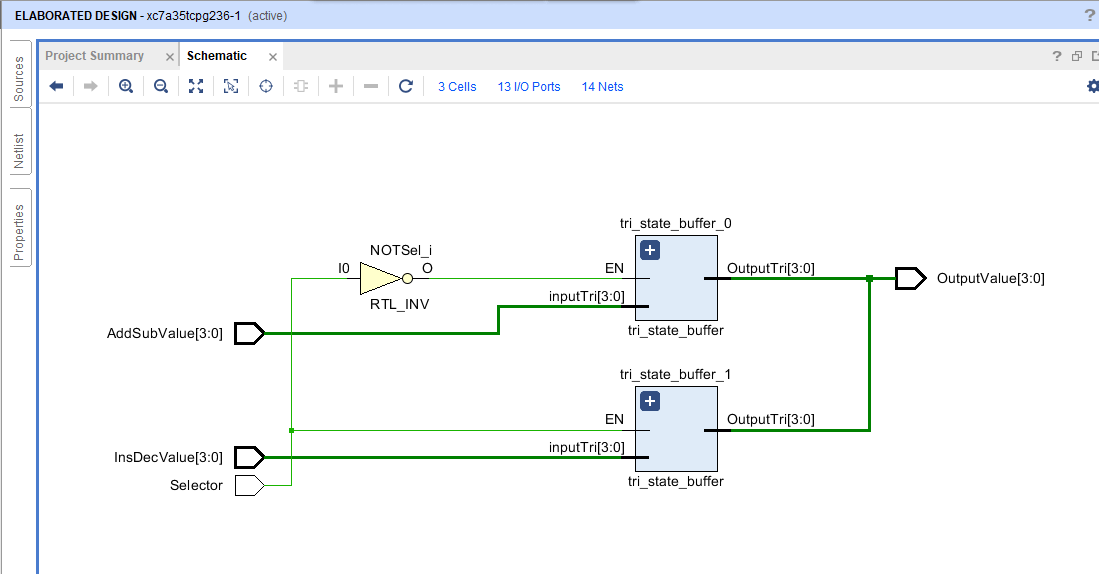
        EN => Selector);

    NOTSel <= NOT Selector;

END Behavioral;

### Timing diagram of 2-way 4-bit multiplexers

### Elaborated design of 2-way 4-bit multiplexers



### VHDL Codes of 8-way 4-bit multiplexers

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY MUX\_8\_way\_4\_bit IS

    PORT (

        Reg0 : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

        Reg1 : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

        Reg2 : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

        Reg3 : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

        Reg4 : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

        Reg5 : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

        Reg6 : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

        Reg7 : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

        RegValue : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0);

        RegSelection : IN STD\_LOGIC\_VECTOR (2 DOWNTO 0));

END MUX\_8\_way\_4\_bit;

ARCHITECTURE Behavioral OF MUX\_8\_way\_4\_bit IS

    COMPONENT Decoder\_3\_to\_8

        PORT (

            I : IN STD\_LOGIC\_VECTOR (2 DOWNTO 0);

            EN : IN STD\_LOGIC;

            Y : OUT STD\_LOGIC\_VECTOR (7 DOWNTO 0));

    END COMPONENT;

    SIGNAL YY : STD\_LOGIC\_VECTOR(7 DOWNTO 0);

    COMPONENT tri\_state\_buffer

        PORT (

            inputTri : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

            outputTri : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0);

            EN : IN STD\_LOGIC);

    END COMPONENT;

BEGIN

    Decoder\_3\_to\_8\_0 : Decoder\_3\_to\_8

    PORT MAP(

        I => RegSelection,

        EN => '1',

        Y => YY);

    tri\_state\_buffer\_0 : tri\_state\_buffer

    PORT MAP(

        inputTri => Reg0,

        EN => YY(0),

        outputTri => RegValue);

    tri\_state\_buffer\_1 : tri\_state\_buffer

    PORT MAP(

        inputTri => Reg1,

        EN => YY(1),

        outputTri => RegValue);

    tri\_state\_buffer\_2 : tri\_state\_buffer

    PORT MAP(

        inputTri => Reg2,

        EN => YY(2),

        outputTri => RegValue);

    tri\_state\_buffer\_3 : tri\_state\_buffer

    PORT MAP(

        inputTri => Reg3,

        EN => YY(3),

        outputTri => RegValue);

    tri\_state\_buffer\_4 : tri\_state\_buffer

    PORT MAP(

        inputTri => Reg4,

        EN => YY(4),

        outputTri => RegValue);

    tri\_state\_buffer\_5 : tri\_state\_buffer

    PORT MAP(

        inputTri => Reg5,

        EN => YY(5),

        outputTri => RegValue);

    tri\_state\_buffer\_6 : tri\_state\_buffer

    PORT MAP(

        inputTri => Reg6,

        EN => YY(6),

        outputTri => RegValue);

    tri\_state\_buffer\_7 : tri\_state\_buffer

    PORT MAP(

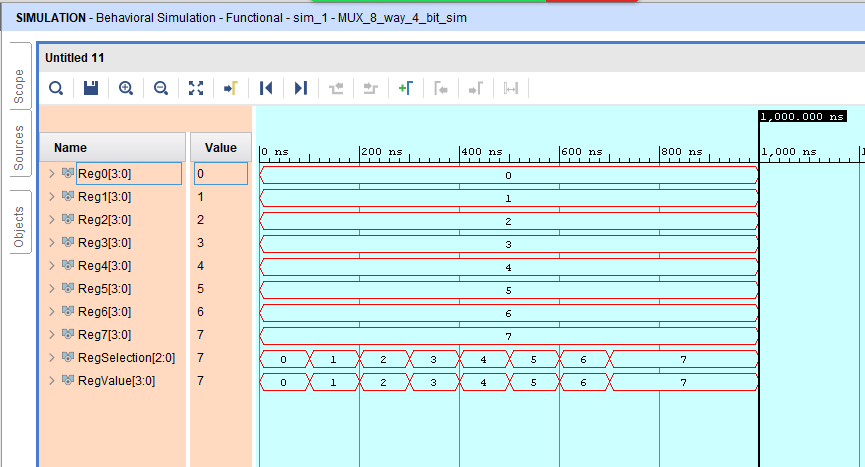
        inputTri => Reg7,

        EN => YY(7),

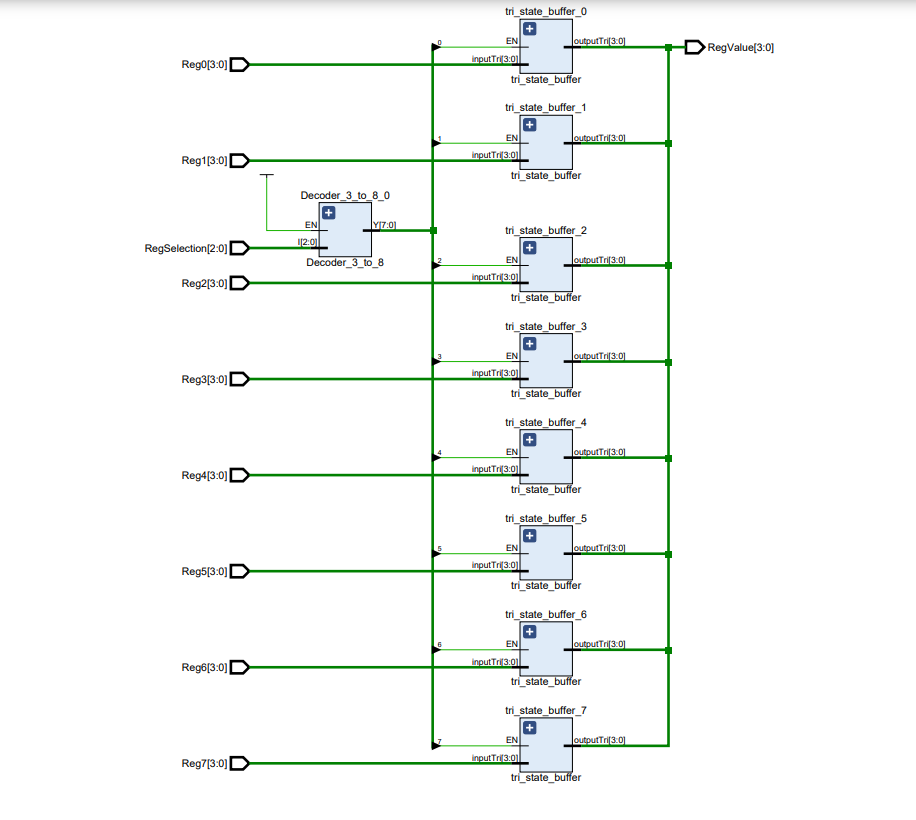
        outputTri => RegValue);

END Behavioral;

### Timing diagram of 8-way 4-bit multiplexers



### Elaborated design of 8-way 4-bit multiplexers



# Register Bank

Contains 8, 4-bit registers (named R0 to R7) o Hardcode value of R0 to all 0s. We used a 3-to-8 decoder which developed in Lab 4. As we do not have a separate instruction to reset a register, we used D Flip Flops with a reset input and connected the reset input to Reset button

### VHDL Code of Register bank

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY RegisterBank IS

    PORT (

        Clk : IN STD\_LOGIC; --Cloak

        Res : IN STD\_LOGIC; --Reset

        R0 : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0); --Register 1

        R1 : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0); --Register 2

        R2 : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0); --Register 3

        R3 : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0); --Register 4

        R4 : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0); --Register 5

        R5 : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0); --Register 6

        R6 : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0); --Register 7

        R7 : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0); --Register 8

        D : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0); --Data bus

        X : IN STD\_LOGIC\_VECTOR (2 DOWNTO 0)); --Instruction

END RegisterBank;

ARCHITECTURE Behavioral OF RegisterBank IS

    COMPONENT Reg --Register component

        PORT (

            D : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

            En : IN STD\_LOGIC;

            Clk : IN STD\_LOGIC;

            Q : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0));

    END COMPONENT;

    COMPONENT Decoder\_3\_to\_8 -- Three to Eight decoder component

        PORT (

            I : IN STD\_LOGIC\_VECTOR (2 DOWNTO 0);

            EN : IN STD\_LOGIC;

            Y : OUT STD\_LOGIC\_VECTOR (7 DOWNTO 0));

    END COMPONENT;

    SIGNAL Y0, Y1 : STD\_LOGIC\_VECTOR(7 DOWNTO 0);

    SIGNAL D\_sig : STD\_LOGIC\_VECTOR(3 DOWNTO 0);

BEGIN

    Decoder\_3\_to\_8\_0 : Decoder\_3\_to\_8 --3 to 8 decoder

    PORT MAP(

        I => X (2 DOWNTO 0),

        EN => '1',

        Y => Y1

    );

    Reg0 : Reg --Register 01

    PORT MAP(

        D => "0000",

        En => '1',

        Clk => Clk,

        Q => R0

    );

    Reg1 : Reg --Register 02

    PORT MAP(

        D => D\_sig,

        En => Y0(1),

        Clk => Clk,

        Q => R1

    );

    Reg2 : Reg --Register 03

    PORT MAP(

        D => D\_sig,

        En => Y0(2),

        Clk => Clk,

        Q => R2);

    Reg3 : Reg --Register 04

    PORT MAP(

        D => D\_sig,

        En => Y0(3),

        Clk => Clk,

        Q => R3

    );

    Reg4 : Reg --Register 05

    PORT MAP(

        D => D\_sig,

        En => Y0(4),

        Clk => Clk,

        Q => R4

    );

    Reg5 : Reg --Register 06

    PORT MAP(

        D => D\_sig,

        En => Y0(5),

        Clk => Clk,

        Q => R5

    );

    Reg6 : Reg --Register 07

    PORT MAP(

        D => D\_sig,

        En => Y0(6),

        Clk => Clk,

        Q => R6

    );

    Reg7 : Reg --Register 08

    PORT MAP(

        D => D\_sig,

        En => Y0(7),

        Clk => Clk,

        Q => R7

    );

    D\_sig <= "0000" WHEN(Res = '1') ELSE

        D;

    Y0 <= "11111111" WHEN (Res = '1') ELSE

        Y1;

END Behavioral;

### Timing diagram of Register bank

Calendar

Description automatically generated with medium confidence

### Elaborated design of Register bank

Diagram, schematic

Description automatically generated

# Program ROM

This stored our Assembly program. This was built by extending the ROM-based LUT developed in Lab 7.

### VHDL Code of Program ROM

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

USE ieee.numeric\_std.ALL;

ENTITY ProgramRom IS

    PORT (

        Memory\_select : IN STD\_LOGIC\_VECTOR (2 DOWNTO 0); -- Memory selection

        Instruction : OUT STD\_LOGIC\_VECTOR (11 DOWNTO 0)); -- Instruction

END ProgramRom;

ARCHITECTURE Behavioral OF ProgramRom IS

    TYPE rom\_type IS ARRAY (0 TO 7) OF STD\_LOGIC\_VECTOR(11 DOWNTO 0);

    SIGNAL programRom : rom\_type := (

        "101110000000",

        "101100000011",

        "101010001111",

        "001111100000",

        "001101010000",

        "111100000111",

        "110000000011",

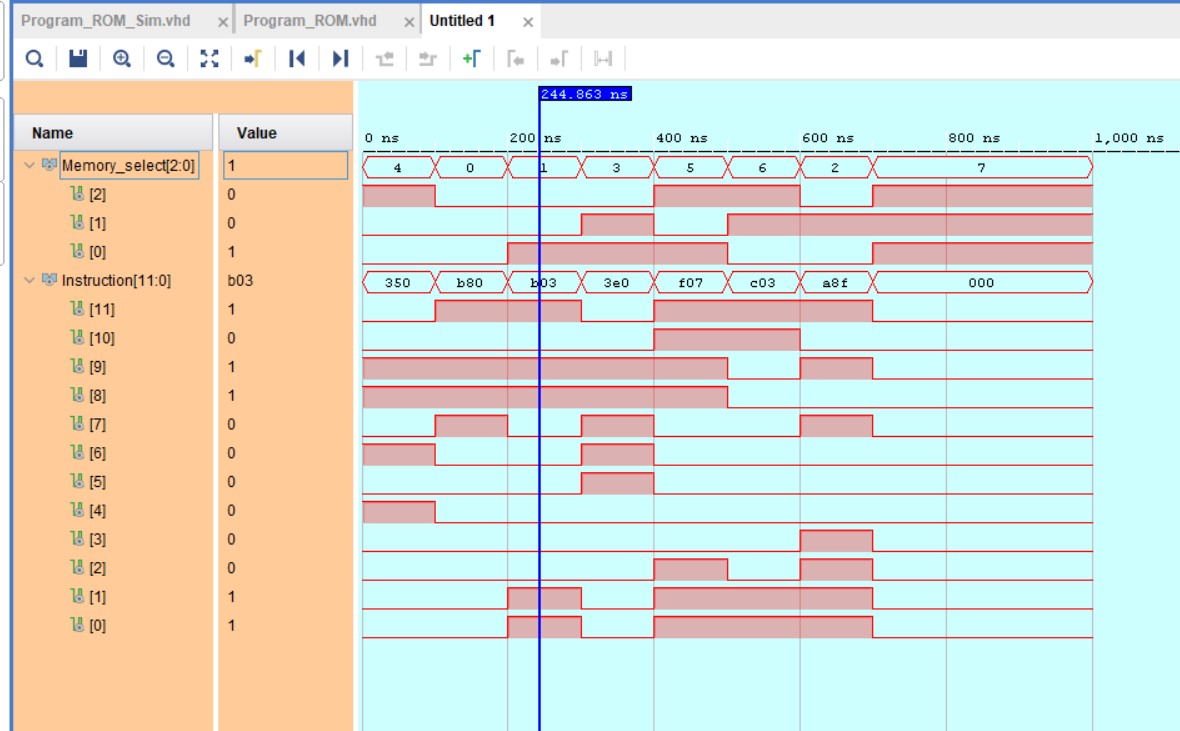
        "000000000000");

BEGIN

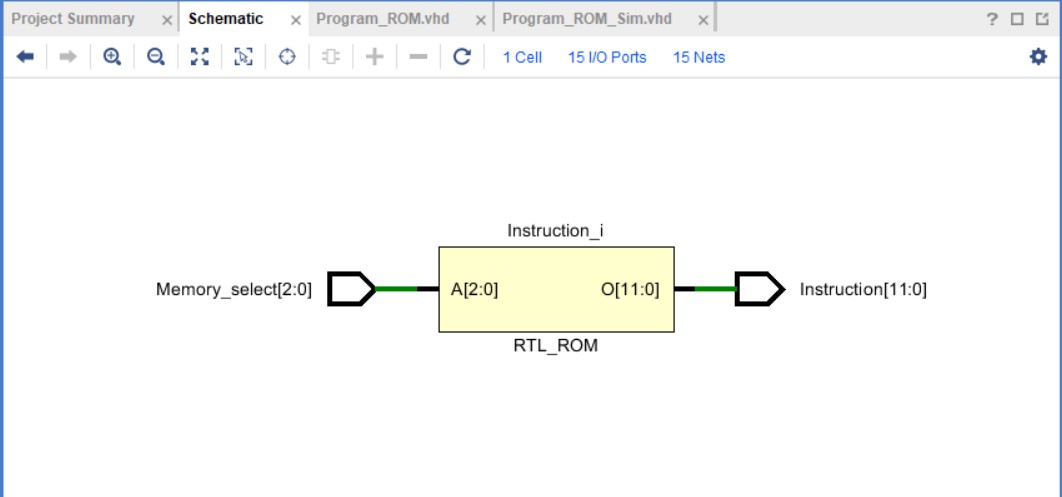
    Instruction <= programRom(to\_integer(unsigned(Memory\_select)));

END Behavioral;

### Timing diagram of Program ROM



### Elaborated design of Program ROM



# Instruction Decoder

Instruction decoder activates necessary components based on the instructions we wish to execute. We mainly execute 4 types of instructions .

* ADD
* NEG
* JZR
* MOVI

### VHDL Code of Instruction Decoder

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY Instruction\_Decoder IS

    PORT (

        INS : IN STD\_LOGIC\_VECTOR (11 DOWNTO 0); --Instruction

        Jump\_check : IN STD\_LOGIC; --Register check for jump

        Reg\_enable : OUT STD\_LOGIC\_VECTOR (2 DOWNTO 0); --Register enable

        Reg\_select\_0 : OUT STD\_LOGIC\_VECTOR (2 DOWNTO 0); --Register select one

        Reg\_select\_1 : OUT STD\_LOGIC\_VECTOR (2 DOWNTO 0); --Register select two

        Imediate\_val : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0); --Imediate value

        Add\_Sub\_select : OUT STD\_LOGIC; --add/sub select

        Load\_select : OUT STD\_LOGIC; --load select

        Jump\_flag : OUT STD\_LOGIC; --jump flag

        Jump\_address : OUT STD\_LOGIC\_VECTOR (2 DOWNTO 0)); --Address to jump

END Instruction\_Decoder;

ARCHITECTURE Behavioral OF Instruction\_Decoder IS

    --to activate only the necessary modules at instance, 2 to 4 decoder is used

    COMPONENT Decoder\_2\_to\_4

        PORT (

            I : IN STD\_LOGIC\_VECTOR (1 DOWNTO 0);

            Enable : IN STD\_LOGIC;

            Y : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0));

    END COMPONENT;

    SIGNAL MOV, ADD, NEG, JZR : STD\_LOGIC;

    SIGNAL JUMP\_FLAG\_0 : STD\_LOGIC;

BEGIN

    Decoder\_2\_to\_40 : Decoder\_2\_to\_4

    PORT MAP(

        I(0) => INS(10),

        I(1) => INS(11),

        Y(0) => ADD,

        Y(1) => NEG,

        Y(2) => MOV,

        Y(3) => JZR,

        Enable => '1');

    Add\_Sub\_select <= NEG;

    Load\_select <= MOV;

    Reg\_Enable <= INS(9 DOWNTO 7);

    Imediate\_val <= INS(3 DOWNTO 0);

    Reg\_Select\_0 <= INS(9 DOWNTO 7);

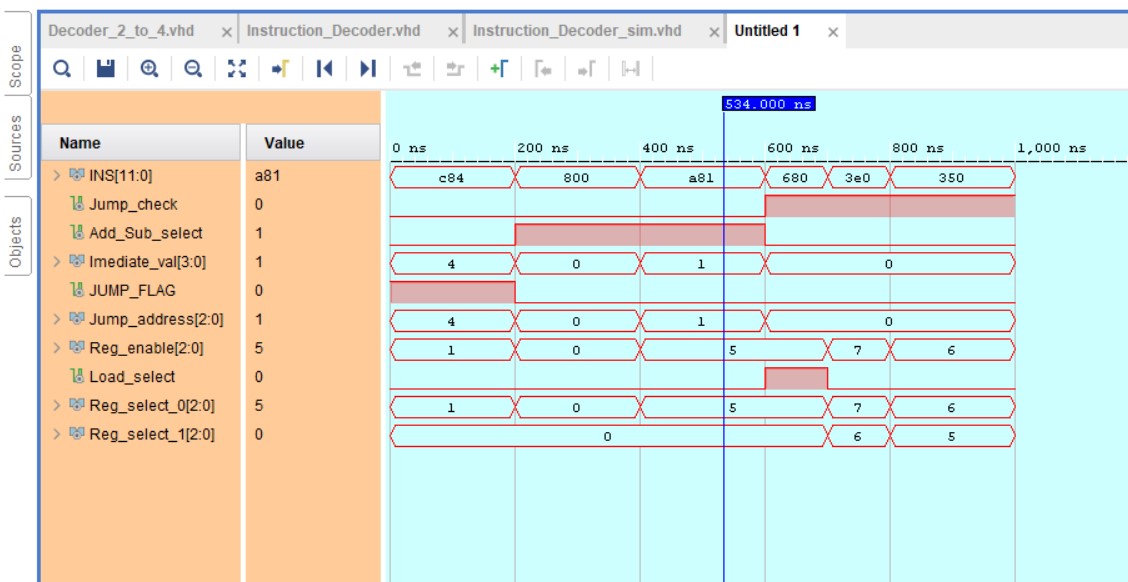
    Reg\_Select\_1 <= INS(6 DOWNTO 4);

    Jump\_flag <= JZR AND (NOT(JUMP\_CHECK));

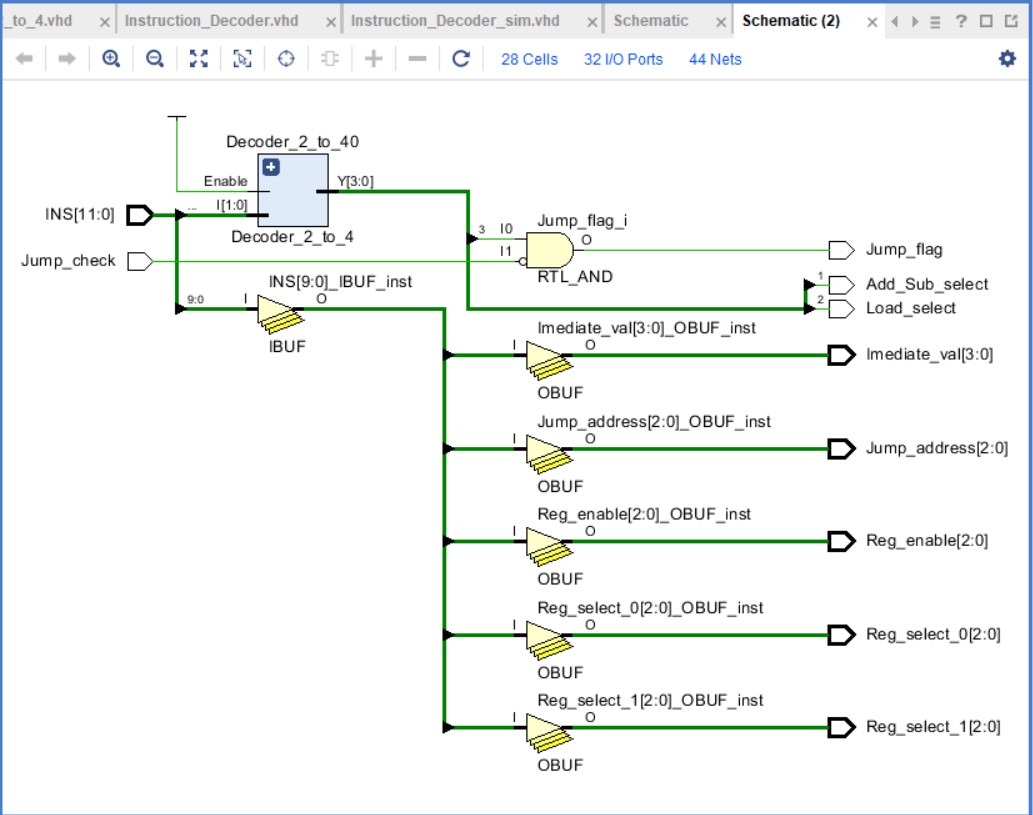
    Jump\_address <= INS(2 DOWNTO 0);

END Behavioral;

### Timing diagram of Instruction Decoder



### Elaborated design of Instruction Decoder



# Nanoprocessor

### VHDL code for Nanoprocessor

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY NanoProcessor IS

    PORT (

        Reset : IN STD\_LOGIC;                               --Reset

        Clk : IN STD\_LOGIC;                                 --Clock

        Zero\_Flag : OUT STD\_LOGIC;                          --Zero Flag

        Overflow\_Flag : OUT STD\_LOGIC;                      --Overflow flag

        Carry\_Flag : OUT STD\_LOGIC;                         --Carry flag

        Negative\_Flag : OUT STD\_LOGIC;                      --Negative Flag

        Reg0 : OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);            --Register 1

        Reg1 : OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);            --Register 2

        Reg2 : OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);            --Register 3

        Reg3 : OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);            --Register 4

        Reg4 : OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);            --Register 5

        Reg5 : OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);            --Register 6

        Reg6 : OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);            --Register 7

        Reg7 : OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);            --Register 8

        Num1 : OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);            --First Number

        Num2 : OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);            --Second Number

        Instruction\_next : OUT STD\_LOGIC\_VECTOR(2 DOWNTO 0);--Next Instruction address

        jmp\_flag : OUT STD\_LOGIC;                           --Jump flag

        instructions : OUT STD\_LOGIC\_VECTOR(11 DOWNTO 0);   --Instructions

        SD\_7\_display : OUT STD\_LOGIC\_VECTOR(6 DOWNTO 0));   --7 segment display

END NanoProcessor;

ARCHITECTURE Behavioral OF NanoProcessor IS

    COMPONENT Slow\_Clock --Slow clock

        PORT (

            Clk\_in : IN STD\_LOGIC;

            Clk\_out : OUT STD\_LOGIC);

    END COMPONENT;

    COMPONENT Counter --Counter

        PORT (

            Next\_Ins : IN STD\_LOGIC\_VECTOR(2 DOWNTO 0) := "000"; --Next INstruction

            Res : IN STD\_LOGIC; --Reset

            Clk : IN STD\_LOGIC; --Clock

            Current\_Ins : OUT STD\_LOGIC\_VECTOR(2 DOWNTO 0)); -- Current Instruction

    END COMPONENT;

    COMPONENT Adder\_3\_bit --3 bit adder

        PORT (

            AA : IN STD\_LOGIC\_VECTOR (2 DOWNTO 0); --first number

            SS : OUT STD\_LOGIC\_VECTOR (2 DOWNTO 0) --get result on this

        );

    END COMPONENT;

    COMPONENT MUX\_2\_way\_4\_bit --2 way 4 bit multiplexer

        PORT (

            AddSubValue : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

            InsDecValue : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

            OutputValue : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0);

            Selector : IN STD\_LOGIC --'0' for addsub and '1' for insdec

        );

    END COMPONENT;

    COMPONENT MUX\_2\_way\_3\_bit --2 way 3 bit multiplexer

        PORT (

            Adder\_3 : IN STD\_LOGIC\_VECTOR (2 DOWNTO 0);

            JUMP\_TO : IN STD\_LOGIC\_VECTOR (2 DOWNTO 0);

            Selector : IN STD\_LOGIC;

            Output : OUT STD\_LOGIC\_VECTOR(2 DOWNTO 0)

        );

    END COMPONENT;

    COMPONENT MUX\_8\_way\_4\_bit --8 way 4 bit multiplexer

        PORT (

            Reg0 : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

            Reg1 : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

            Reg2 : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

            Reg3 : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

            Reg4 : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

            Reg5 : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

            Reg6 : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

            Reg7 : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

            RegValue : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0);

            RegSelection : IN STD\_LOGIC\_VECTOR (2 DOWNTO 0)

        );

    END COMPONENT;

    COMPONENT ProgramRom --program ROM

        PORT (

            Memory\_select : IN STD\_LOGIC\_VECTOR (2 DOWNTO 0); -- Memory selection

            Instruction : OUT STD\_LOGIC\_VECTOR (11 DOWNTO 0) -- Instruction

        );

    END COMPONENT;

    COMPONENT Instruction\_Decoder --Instruction Decoder

        PORT (

            INS : IN STD\_LOGIC\_VECTOR (11 DOWNTO 0); --Instruction

            Jump\_check : IN STD\_LOGIC\_VECTOR(3 DOWNTO 0); --Register check for jump

            Reg\_enable : OUT STD\_LOGIC\_VECTOR (2 DOWNTO 0); --Register enable

            Reg\_select\_0 : OUT STD\_LOGIC\_VECTOR (2 DOWNTO 0); --Register select one

            Reg\_select\_1 : OUT STD\_LOGIC\_VECTOR (2 DOWNTO 0); --Register select two

            Imediate\_val : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0); --Imediate value

            Add\_Sub\_select : OUT STD\_LOGIC; --add/sub select

            Load\_select : OUT STD\_LOGIC; --load select

            Jump\_flag : OUT STD\_LOGIC; --jump flag

            Jump\_address : OUT STD\_LOGIC\_VECTOR (2 DOWNTO 0) --Address to jump

        );

    END COMPONENT;

    COMPONENT RegisterBank --Register Bank

        PORT (

            Clk : IN STD\_LOGIC; --Cloak

            Res : IN STD\_LOGIC; --Reset

            R0 : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0); --Register 1

            R1 : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0); --Register 2

            R2 : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0); --Register 3

            R3 : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0); --Register 4

            R4 : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0); --Register 5

            R5 : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0); --Register 6

            R6 : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0); --Register 7

            R7 : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0); --Register 8

            D : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0); --Data bus

            X : IN STD\_LOGIC\_VECTOR (2 DOWNTO 0) --Instruction

        );

    END COMPONENT;

    COMPONENT Add\_Sub\_unit --Add/Sub unit

        PORT (

            AA : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0); --first number

            BB : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0); --second number

            SS : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0); --get result on this

            Sel : IN STD\_LOGIC; --Select add or substract using this

            Zero : OUT STD\_LOGIC; -- Zero flag

            Overflow : OUT STD\_LOGIC; --Overflow flag

            Carry : OUT STD\_LOGIC; -- carry flag

            Negative : OUT STD\_LOGIC --Negative flag

        );

    END COMPONENT;

    COMPONENT LUT\_16\_7 -- Look up table which we designed in lab 7(7 segment display)

        PORT (

            address : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

            data : OUT STD\_LOGIC\_VECTOR (6 DOWNTO 0));

    END COMPONENT;

    SIGNAL Number01, Number02 : STD\_LOGIC\_VECTOR (3 DOWNTO 0);

    SIGNAL AddSubTotal : STD\_LOGIC\_VECTOR (3 DOWNTO 0) := "0000";

    SIGNAL Adder\_3\_value : STD\_LOGIC\_VECTOR(2 DOWNTO 0);

    SIGNAL Jump\_flag\_sel : STD\_LOGIC;

    SIGNAL Address\_to\_jump : STD\_LOGIC\_VECTOR(2 DOWNTO 0);

    SIGNAL Enable, Res : STD\_LOGIC;

    SIGNAL Next\_instruction : STD\_LOGIC\_VECTOR(2 DOWNTO 0);

    SIGNAL Current\_Instruction : STD\_LOGIC\_VECTOR(2 DOWNTO 0) := "000";

    SIGNAL Instruction\_bus : STD\_LOGIC\_VECTOR(11 DOWNTO 0);

    SIGNAL Load\_sel, Add\_Sub\_sel : STD\_LOGIC;

    SIGNAL Reg\_sel\_0, Reg\_sel\_1 : STD\_LOGIC\_VECTOR(2 DOWNTO 0);

    SIGNAL Register\_enable : STD\_LOGIC\_VECTOR(2 DOWNTO 0);

    SIGNAL Register\_receiving\_value, Imediate\_value : STD\_LOGIC\_VECTOR(3 DOWNTO 0);

    SIGNAL R0\_value, R1\_value, R2\_value, R3\_value, R4\_value, R5\_value, R6\_value, R7\_value : STD\_LOGIC\_VECTOR(3 DOWNTO 0);

    SIGNAL Clk\_out : STD\_LOGIC;

BEGIN

    clock : Slow\_Clock --slow clock

    PORT MAP(

        Clk\_in => Clk,

        Clk\_out => Clk\_out);

    Number\_01 : MUX\_8\_way\_4\_bit --first 8 way 4 bit multiplexer

    PORT MAP(

        Reg0 => R0\_value,

        Reg1 => R1\_value,

        Reg2 => R2\_value,

        Reg3 => R3\_value,

        Reg4 => R4\_value,

        Reg5 => R5\_value,

        Reg6 => R6\_value,

        Reg7 => R7\_value,

        RegValue => Number01,

        RegSelection => Reg\_sel\_0

    );

    Number\_02 : MUX\_8\_way\_4\_bit--second 8 way 4 bit multiplexer

    PORT MAP(

        Reg0 => R0\_value,

        Reg1 => R1\_value,

        Reg2 => R2\_value,

        Reg3 => R3\_value,

        Reg4 => R4\_value,

        Reg5 => R5\_value,

        Reg6 => R6\_value,

        Reg7 => R7\_value,

        RegValue => Number02,

        RegSelection => Reg\_sel\_1

    );

    add\_sub : Add\_Sub\_unit --add sub unit

    PORT MAP(

        AA => Number02, --value in second MUX

        BB => Number01, --value in first MUX

        SS => AddSubTotal,

        Sel => Add\_Sub\_sel,

        Zero => Zero\_Flag,

        Overflow => Overflow\_Flag,

        Carry => Carry\_Flag,

        Negative => Negative\_Flag

    );

    MUX\_2\_way\_4 : MUX\_2\_way\_4\_bit ----2 way 4 bit multiplexer

    PORT MAP(

        AddSubValue => AddSubTotal,

        InsDecValue => Imediate\_value,

        OutputValue => Register\_receiving\_value,

        Selector => Load\_sel

    );

    register\_bank : RegisterBank --register bank

    PORT MAP(

        Clk => Clk\_out,

        Res => Reset,

        R0 => R0\_value,

        R1 => R1\_value,

        R2 => R2\_value,

        R3 => R3\_value,

        R4 => R4\_value,

        R5 => R5\_value,

        R6 => R6\_value,

        R7 => R7\_value,

        D => Register\_receiving\_value,

        X => Register\_enable

    );

    instruction\_dec : Instruction\_Decoder --instruction decoder

    PORT MAP(

        INS => Instruction\_bus,

        Jump\_check => Number01,

        Reg\_enable => Register\_enable,

        Reg\_select\_0 => Reg\_sel\_0,

        Reg\_select\_1 => Reg\_sel\_1,

        Imediate\_val => Imediate\_value,

        Add\_Sub\_select => Add\_Sub\_sel,

        Load\_select => Load\_sel,

        Jump\_flag => Jump\_flag\_sel,

        Jump\_address => Address\_to\_jump

    );

    adder : Adder\_3\_bit --adder 3 bit

    PORT MAP(

        AA => Current\_Instruction,

        SS => Adder\_3\_value

    );

    MUX\_2\_way\_3 : MUX\_2\_way\_3\_bit--2 way 3 bit multiplexer

    PORT MAP(

        Adder\_3 => Adder\_3\_value,

        JUMP\_TO => Address\_to\_jump,

        Selector => Jump\_flag\_sel,

        Output => Next\_instruction

    );

    programme\_counter : Counter --program counter

    PORT MAP(

        Next\_Ins => Next\_instruction,

        Res => Reset,

        Clk => Clk\_out,

        Current\_Ins => Current\_Instruction

    );

    prom : ProgramRom --program rom

    PORT MAP(

        Memory\_select => Current\_Instruction,

        Instruction => Instruction\_bus

    );

    Display\_7\_segment : LUT\_16\_7 PORT MAP( --7 segment display

        address => AddSubTotal,

        data => SD\_7\_display

    );

    --selection proper port to each register

    Reg0 <= R0\_value;

    Reg1 <= R1\_value;

    Reg2 <= R2\_value;

    Reg3 <= R3\_value;

    Reg4 <= R4\_value;

    Reg5 <= R5\_value;

    Reg6 <= R6\_value;

    Reg7 <= R7\_value;

    Num1 <= Number01;

    Num2 <= Number02;

    jmp\_flag <= Jump\_flag\_sel;

    instructions <= Instruction\_bus;

    Instruction\_next <= Next\_instruction;

END Behavioral;

# Conclusions

# Member Responsibilities

|  |  |  |
| --- | --- | --- |
| Name | Responsibilities | Hours spent |
| Liyanapathirana S.T. | Program Rom  Instruction Decoder  Report writing | 36 |
| Yasith Heshan | 3-bit counter  Register bank  Report writing  NanoProcessor | 48 |
| Nimesh Ariyaratne | 3-bit adder  4-bit Add/Sub unit  K-way b-bit mux  Report writing  NanoProcessor | 48 |