Final Lab: Single Cycle CPU-LITE

Mark Natavio

May 22, 2022

CSC 34200/34300

Table of Contents

Objective **3**

Description of Specifications and Functionality **3**

Data Memory 7
Instruction Register (IR) 7
Instruction Memory 8
Register File (RF) 8
Program Counter (PC) 9
Adder 9
5-Bit 2-to-1 Multiplexer 11
32-Bit 2-to-1 Multiplexer 12
Sign Extender 13
Logical Bitwise Operation Unit 13
Add/Sub 14
Arithmetic Logic Unit (ALU) 15
CPU Controller Unit 16
CPU Components Package 18
Central Processing Unit (CPU) 20

Simulations **22**

Conclusion **26**

Objective

The objective for this lab is to build a single-cycle CPU using VHDL by combining elements from previous labs such as the BEQ, BNE, J Lab and the Arithmetic Logic Unit (ALU) lab. The CPU is meant to be capable of executing MIPS instructions in Table 1. Then we are to compare the CPU and MIPS on MARS with each other by adding 10 numbers together.

Description of Specifications and Functionality

For this lab we are to be able to execute the MIPS instructions as laid out in Table 1. As shown in the table, there are three main instruction formats in MIPS: R-Type, I-Type, and J-Type. This is important to point out as each instruction is represented by 32 bits and organized differently based on its type, which the CPU needs to break down accordingly.

Name	Mnemonic	Format	Operation
Add	add	R	R[rd] = R[rs] + R[rt]
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]
Subtract	sub	R	R[rd] = R[rs] - R[rt]
Subtract Unsigned	subu	R	R[rd] = R[rs] - R[rt]
AND	and	R	R[rd] = R[rs] & R[rt]
AND Immediate	andi	I	R[rt] = R[rs] & ZreoExtImm
NOR	nor	R	R[rd] = ~ (R[rs] | R[rt])
OR Immediate	ori	I	R[rt] = R[rs] | ZeroExtImm
Shift Left	sll	R	R[rd] = R[rt] << shamt
Shift Right	srl	R	R[rd] = R[rt] >> shamt
Shift Right Arith	sra	R	R[rd] = R[rt] >>> shamt
Store Word	sw	I	M[R[rs] + SignExtImm] = R[rt]
Load Word	lw	I	R[rt] = M[R[rs] + SignExtImm]
Branch on Equal	beq	I	If R[rs] == R[rt] : PC = PC + 4 + [SignExtImm, 2b00]
Branch on Not Equal	bne	I	If R[rs] != R[rt] : PC = PC + 4
Jump	j	J	PC = PC + JumpAddress + 00

Table 1. All MIPS instructions required

Instruction Types Breakdown

There are three main instruction formats in MIPS: R-Type, I-Type, and J-Type. Each of these is composed of different parts which indicate different values that correspond to a specific instruction. These parts are OPCode, RS Address, RT Address, RD Address, Shamt, Funct, and Immediate.

R-Type Instructions

R-Type instructions have the OPCode of 000000 and are differentiated by their funct value. The funct value is the last 6 bits of the 32-bits and they are displayed on Table 2. All R-Types (except for the shift instructions sll, srl, and sra) use registers. They are formatted as follows:

31	26	21	16	11	6 0
OPCode	RS Address	RT Address	RD Address	Shamt	Funct
6 Bits	5 Bits	5 Bits	5 Bits	5 Bits	6 Bits

Instruction	Funct Value
add	000000
addu	000001
sub	000010
subu	000011
and	000100
nor	000101
sll	000110
srl	000111
sra	001000

Table 2. All R-Type Instruction Funct Values

I-Type Instructions

I-Type instructions have an OPCode greater than 3. They all have a 16-bit immediate value that is sign extended to 32-bits (except for and, or, and xor). Each instruction is differentiated by their OPCode as displayed on Table 3. It is worth mentioning that the Branch instruction will multiply the immediate by 4 to get the offset value. They are formatted as follows:

31	26	21	16 0
OPCode	RS Address	RT Address	Immediate
6 Bits	5 Bits	5 Bits	16 Bits

J-Type Instructions

The jump J-Type instructions have an OPCode of 2 and 3. Since we are only using instruction J, the OPCode for this instruction will be 2. J-Type instructions are characterized by their OPCode and contain a 26 bit Immediate value which is the target instruction address. J-Type instructions are formatted as follows:

31	26 … 0
OPCode	Immediate
6 bits	26 bits

Since the Jump target address is 26 bits and an address is 32 bits, the Immediate value is shifted to the left by 2 bits and then the first 4 bits are the result of PC + 4. It looks as follows:

31	28 … 2	0
PC + 4	Immediate	00
4 bits	26 bits	2 bits

CPU Design Breakdown

Now that we know how the instructions are broken down, I began to design the CPU that I will be building. According to the lab instructions, the CPU design will require a CPU Controller Operation, Data Memory, Instruction memory, Dual Ported Register File, PC Register, Instruction Register, Arithmetic Logic Unit (ALU), Adders, 2-to-1 Multiplexers, and 16-to-32 bit extender. All of these components have been used in previous labs with the exception of the CPU Controller Operation.

CPU Controller

The CPU Controller organizes and controls the data paths for each instruction by managing control signals (i.e. RegWr, RegDst, ExtOp, ALUSrc, ALUCtr, MemWr, MemToReg, Branch, and Jump). It will take an instruction from the Instruction Memory and get the OPCode, RT, RS, RD, Shamt, Funct, and Imm16 values as shown in Diagram 1.

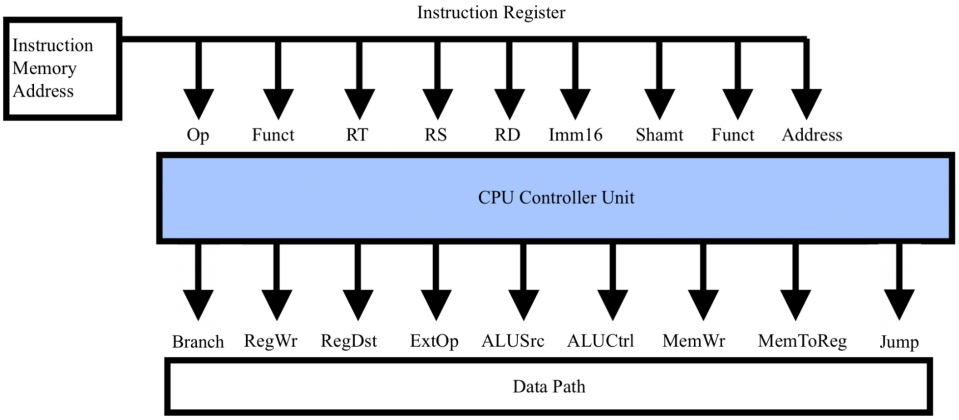


Diagram 1. CPU Controller Design

Control Signals Breakdown

The control signals that the CPU Controller controls are RegWr, RegDst, ExtOp, ALUSrc, ALUCtr, MemWr, MemToReg, Branch, and Jump. Every control signal for every instruction is shown in Table 3. This table is based on what was made during a lab exercise. However, during the lab session I made many mistakes as I had a limited time to analyze the control signals. Now with more time I have updated the table and corrected values of the control signals.

Instruction	OPCode	RegWr	RegDst	ExtOp	ALUSrc	ALUCtrl	MemWr	MemRd	MemToReg	Jump	Branch
add	000000	1	1	X	0	0000	0	0	0	0	0
addi	000100	1	0	1	1	0001	0	0	0	0	0
addiu	000101	1	0	1	1	0001	0	0	0	0	0
addu	000000	1	1	X	0	0000	0	0	0	0	0
sub	000000	1	1	X	0	0010	0	0	0	0	0
subu	000000	1	1	X	0	0010	0	0	0	0	0
and	000000	1	1	X	0	0011	0	0	0	0	0
andi	000110	1	0	0	1	0100	0	0	0	0	0
ori	000111	1	0	0	1	0101	0	0	0	0	0
nor	000000	1	1	X	0	0110	0	0	0	0	0
sll	000000	1	1	X	0	0111	0	0	0	0	0
srl	000000	1	1	X	0	1000	0	0	0	0	0
sra	000000	1	1	X	0	1001	0	0	0	0	0
sw	001000	0	X	1	1	0000	1	0	X	0	0
lw	001001	1	0	1	1	0000	0	1	1	0	0
beq	001010	0	X	1	0	0010	0	0	X	0	1
bne	001011	0	X	1	0	0010	0	0	X	0	1
j	000010	0	X	X	X	XXXX	0	0	X	1	0

Table 3. All Instruction Control Signals

Data Memory

This code is a code design for the Data Memory Component. This component has a memory size of 64 bytes and the data word size is 32 bits. This component is used for inputting data to be processed into a data memory block. When the clock is enabled, the data inputted is processed into the mem_arr of the MemAddress. Once the clock is enabled and wren is disabled, the value is set to the Out value.

Natavio_May_22_2022_Data_Memory.vhd | Compilation Report - Natavio_May_22_2022_Data_Memory

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY Natavio_May_22_2022_Data_Memory IS
   PORT(Natavio_May_22_2022_clk     : IN std_logic;
        Natavio_May_22_2022_wren    : IN std_logic;
        Natavio_May_22_2022_rden    : IN std_logic;
        Natavio_May_22_2022_data    : IN std_logic_vector (31 DOWNTO 0);
        Natavio_May_22_2022_address : IN std_logic_vector (31 DOWNTO 0);
        Natavio_May_22_2022_Out     : OUT std_logic_vector (31 DOWNTO 0));
END Natavio_May_22_2022_Data_Memory;

ARCHITECTURE arch OF Natavio_May_22_2022_Data_Memory IS
   SIGNAL Natavio_May_22_2022_MemAddress  : std_logic_vector (31 DOWNTO 0);
   TYPE Natavio_May_22_2022_mem IS ARRAY(0 TO 63) OF std_logic_vector (31 DOWNTO 0);
   SIGNAL Natavio_May_22_2022_mem_arr  : Natavio_May_22_2022_mem := ((others => (others => '0')));

   BEGIN
      Natavio_May_22_2022_MemAddress <= Natavio_May_22_2022_address(31 DOWNTO 0);
      PROCESS (Natavio_May_22_2022_clk)
         BEGIN
            IF (rising_edge(Natavio_May_22_2022_clk)) THEN
               IF (Natavio_May_22_2022_wren = '1') THEN
                  Natavio_May_22_2022_mem_arr(to_integer(unsigned(Natavio_May_22_2022_MemAddress))) <= Natavio_May_22_2022_data;
               END IF;
            END IF;
      END PROCESS;

   Natavio_May_22_2022_Out <= Natavio_May_22_2022_mem_arr(to_integer(unsigned(Natavio_May_22_2022_MemAddress))) WHEN (Natavio_May_22_2022_rden = '1') ELSE x"00000000";
END arch;
```

Instruction Register (IR)

This code is a code design for the Instruction Register (IR). This component is meant to act as a 32-bit register that stores the instruction being run by the CPU during execution. The current instruction is saved in the mem signal. Once the clock and wren signals are enabled, the instruction in mem will be outputted into IR_q. Otherwise, the IR_q is loaded with ‘Z’.

Natavio_May_22_2022_Data_Memory.vhd | Natavio_May_22_2022_IR.vhd | Compilation Report - Natavio_May_22_2022_Data_Memory

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY Natavio_May_22_2022_IR IS
   PORT (Natavio_May_22_2022_clk           : IN std_logic;
         Natavio_May_22_2022_wren          : IN std_logic;
         Natavio_May_22_2022_rden          : IN std_logic;
         Natavio_May_22_2022_chen          : IN std_logic;
         Natavio_May_22_2022_Instruction   : IN std_logic_vector (31 DOWNTO 0);
         Natavio_May_22_2022_IR_q          : OUT std_logic_vector (31 DOWNTO 0));
END Natavio_May_22_2022_IR;

ARCHITECTURE arch OF Natavio_May_22_2022_IR IS
   SIGNAL Natavio_May_22_2022_mem : std_logic_vector (31 DOWNTO 0);

   BEGIN
      Natavio_May_22_2022_p1  : PROCESS(Natavio_May_22_2022_clk)
         BEGIN
            IF (rising_edge(Natavio_May_22_2022_clk) AND Natavio_May_22_2022_wren = '1') THEN
               Natavio_May_22_2022_mem <= Natavio_May_22_2022_Instruction;
            END IF;
      END PROCESS Natavio_May_22_2022_p1;

      Natavio_May_22_2022_p2  : PROCESS(Natavio_May_22_2022_rden, Natavio_May_22_2022_chen, Natavio_May_22_2022_mem)
         BEGIN
            IF (Natavio_May_22_2022_rden = '1' AND Natavio_May_22_2022_chen = '1') THEN
               Natavio_May_22_2022_IR_q <= Natavio_May_22_2022_mem;
            ELSIF (Natavio_May_22_2022_chen = '0') THEN
               Natavio_May_22_2022_IR_q <= (OTHERS => 'Z');
            END IF;
      END PROCESS Natavio_May_22_2022_p2;
END arch;
```

Instruction Memory

This code is a code design for the Instruction Memory component, which contains the MIPS instructions to be executed for testing later. It will split the instruction into the 7 different instruction parts.

Natavio_May_22_2022_Data_Memory.vhd | Natavio_May_22_2022_IR.vhd | Natavio_May_22_2022_Instruction_Memory.vhd | Compilation Report - Natavio_M

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY Natavio_May_22_2022_Instruction_Memory IS
   PORT (Natavio_May_22_2022_inst_addr : IN std_logic_vector (31 DOWNTO 0);
         Natavio_May_22_2022_OPCode    : OUT std_logic_vector (5 DOWNTO 0);
         Natavio_May_22_2022_RS        : OUT std_logic_vector (4 DOWNTO 0);
         Natavio_May_22_2022_RT        : OUT std_logic_vector (4 DOWNTO 0);
         Natavio_May_22_2022_RD        : OUT std_logic_vector (4 DOWNTO 0);
         Natavio_May_22_2022_shamt     : OUT std_logic_vector (4 DOWNTO 0);
         Natavio_May_22_2022_funct     : OUT std_logic_vector (5 DOWNTO 0);
         Natavio_May_22_2022_Imm16     : OUT std_logic_vector (15 DOWNTO 0);
         Natavio_May_22_2022_J_Addr    : OUT std_logic_vector (31 DOWNTO 0);
         Natavio_May_22_2022_Inst      : OUT std_logic_vector (31 DOWNTO 0));
END Natavio_May_22_2022_Instruction_Memory;

ARCHITECTURE arch OF Natavio_May_22_2022_Instruction_Memory IS
   TYPE Natavio_May_22_2022_mem IS ARRAY (0 to 127) OF std_logic_vector (31 DOWNTO 0);

   SIGNAL Natavio_May_22_2022_mem_arr : Natavio_May_22_2022_mem;
   SIGNAL Natavio_May_22_2022_Instruction : std_logic_vector (31 DOWNTO 0);

   BEGIN
      Natavio_May_22_2022_mem_arr(0)  <= x"10100001"; -- addi $s0, 1
      Natavio_May_22_2022_mem_arr(4)  <= x"12100002"; -- addi $s0, 2
      Natavio_May_22_2022_mem_arr(8)  <= x"12100003"; -- addi $s0, 3
      Natavio_May_22_2022_mem_arr(12) <= x"12100004"; -- addi $s0, 4
      Natavio_May_22_2022_mem_arr(16) <= x"12100005"; -- addi $s0, 5

      Natavio_May_22_2022_Instruction <= Natavio_May_22_2022_mem_arr(to_integer(unsigned(Natavio_May_22_2022_inst_addr)));

      Natavio_May_22_2022_OPCode <= Natavio_May_22_2022_Instruction(31 DOWNTO 26);
      Natavio_May_22_2022_RS <= Natavio_May_22_2022_Instruction(25 DOWNTO 21);
      Natavio_May_22_2022_RT <= Natavio_May_22_2022_Instruction(20 DOWNTO 16);
      Natavio_May_22_2022_RD <= Natavio_May_22_2022_Instruction(15 DOWNTO 11);
      Natavio_May_22_2022_shamt <= Natavio_May_22_2022_Instruction(10 DOWNTO 6);
      Natavio_May_22_2022_funct <= Natavio_May_22_2022_Instruction(5 DOWNTO 0);
      Natavio_May_22_2022_Imm16 <= Natavio_May_22_2022_Instruction(15 DOWNTO 0);
      Natavio_May_22_2022_J_Addr <= Natavio_May_22_2022_inst_addr(31 DOWNTO 28) & Natavio_May_22_2022_Instruction(25 DOWNTO 0) & "00";
      Natavio_May_22_2022_Inst <= Natavio_May_22_2022_Instruction(31 DOWNTO 0);
END arch;
```

Register File (RF)

This code is a code design for the Register File (RF). It is dual ported with two reads and one additional write port. This RF is made up of 32 registers of size 32 bits each.

Natavio_May_22_2022_RF.vhd | Compilation Report - Natavio_May_22_2022_Data_Memory

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY Natavio_May_22_2022_RF IS
   PORT (Natavio_May_22_2022_clk    : IN std_logic;
         Natavio_May_22_2022_wren   : IN std_logic;
         Natavio_May_22_2022_data   : IN std_logic_vector (31 DOWNTO 0);
         Natavio_May_22_2022_wr     : IN std_logic_vector (4 DOWNTO 0);
         Natavio_May_22_2022_rdA    : IN std_logic_vector (4 DOWNTO 0);
         Natavio_May_22_2022_rdB    : IN std_logic_vector (4 DOWNTO 0);
         Natavio_May_22_2022_qA     : OUT std_logic_vector (31 DOWNTO 0);
         Natavio_May_22_2022_qB     : OUT std_logic_vector (31 DOWNTO 0));
END Natavio_May_22_2022_RF;

ARCHITECTURE arch OF Natavio_May_22_2022_RF IS
   TYPE Natavio_May_22_2022_reg IS ARRAY(0 TO 31) OF std_logic_vector (31 DOWNTO 0);
   SIGNAL Natavio_May_22_2022_reg_arr : Natavio_May_22_2022_reg;

   BEGIN
      PROCESS (Natavio_May_22_2022_clk, Natavio_May_22_2022_wren)
         BEGIN
            IF (rising_edge(Natavio_May_22_2022_clk)) THEN
               IF (Natavio_May_22_2022_wren = '1') THEN
                  Natavio_May_22_2022_reg_arr(to_integer(unsigned(Natavio_May_22_2022_wr))) <= Natavio_May_22_2022_data;
               END IF;
            END IF;
      END PROCESS;

      Natavio_May_22_2022_qA <= x"00000000" WHEN Natavio_May_22_2022_rdA = "00000" ELSE
                                Natavio_May_22_2022_reg_arr(to_integer(unsigned(Natavio_May_22_2022_rdA)));

      Natavio_May_22_2022_qB <= x"00000000" WHEN Natavio_May_22_2022_rdB = "00000" ELSE
                                Natavio_May_22_2022_reg_arr(to_integer(unsigned(Natavio_May_22_2022_rdB)));

END arch;
```

Program Counter (PC)

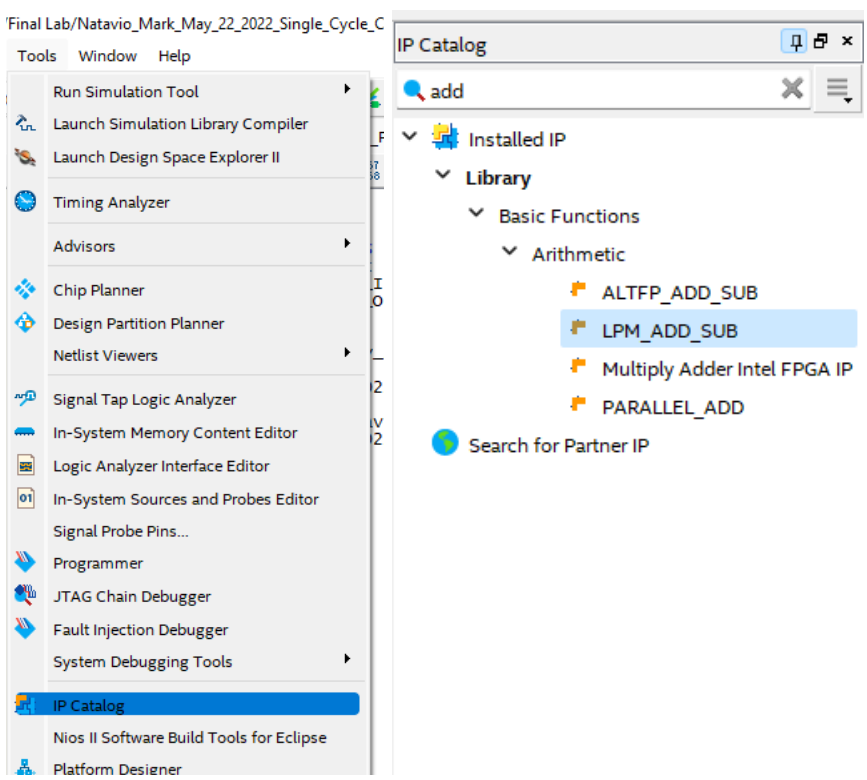
This code is a code design for the Program Counter (PC). This component is a 32-bit register that stores the address of the instruction that is currently being executed.

Natavio_May_22_2022_PC.vhd

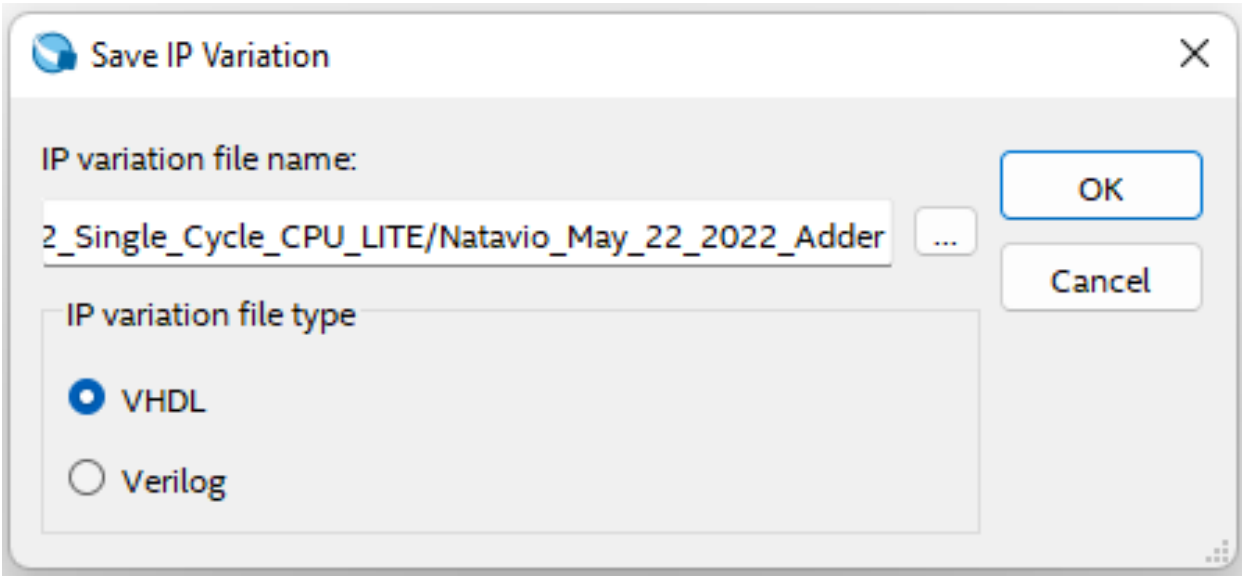
```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Natavio_May_22_2022_PC IS
   PORT (Natavio_May_22_2022_clk     : IN std_logic;
         Natavio_May_22_2022_PC_I    : IN std_logic_vector (31 DOWNTO 0);
         Natavio_May_22_2022_PC_O    : OUT std_logic_vector (31 DOWNTO 0));
END Natavio_May_22_2022_PC;

ARCHITECTURE arch OF Natavio_May_22_2022_PC IS
   BEGIN
      PROCESS (Natavio_May_22_2022_clk)
         BEGIN
            IF (rising_edge(Natavio_May_22_2022_clk)) THEN
               Natavio_May_22_2022_PC_O <= Natavio_May_22_2022_PC_I;
            END IF;
      END PROCESS;
END arch;
```

Adder

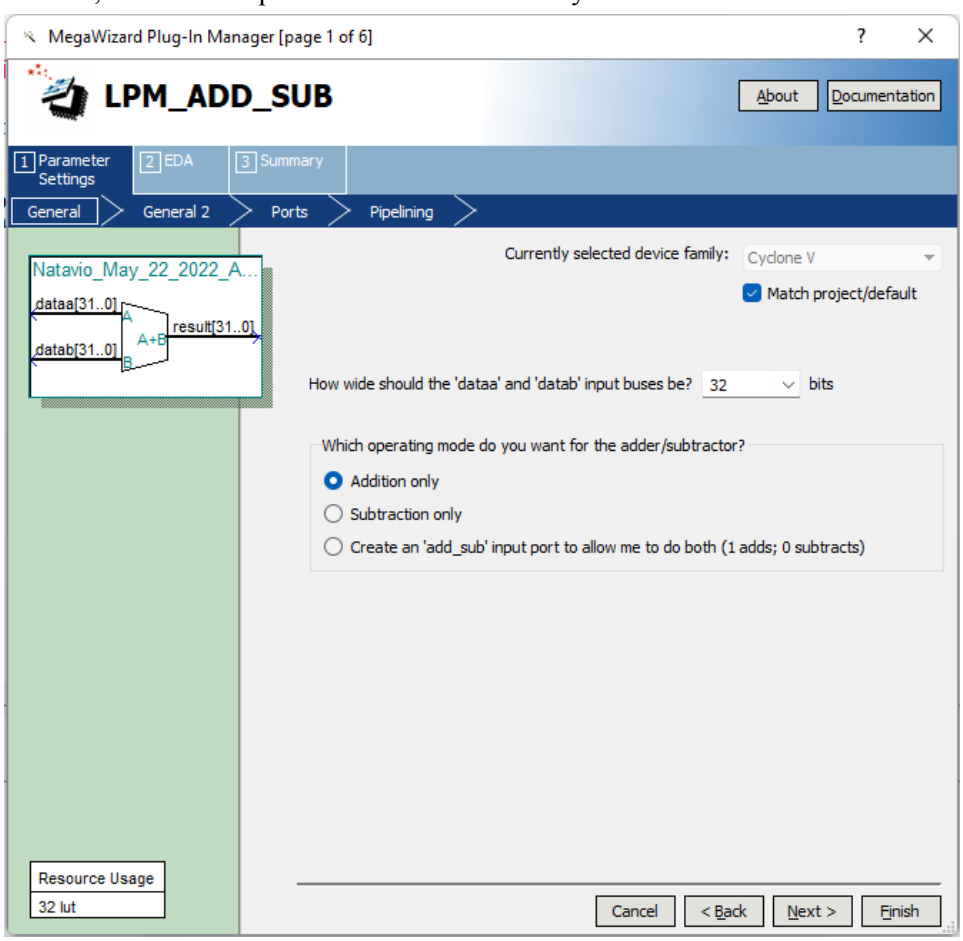
Now I began creating the Adder Component by using the IP Catalog and adding an LPM_ADD_SUB component into the project.



I set the name of the component to be Natavio_May_22_2022_Adder as per the instructions all files must have the correct prefix. I also set the IP Variation file type to be VHDL and then clicked OK.



I then set the adder to be a 32-bit Adder. Since this component is exclusively supposed to be working as an adder, I set the component to do 'Addition only'.



From here on out all settings were left as default so I clicked on Finish and got the following VHDL code. I made some minor changes to add my last name as a prefix to all signal values.

Natavio_May_22_2022_PC.vhd | Natavio_May_22_2022_Adder.vhd | Co

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY lpm;
USE lpm.all;

ENTITY Natavio_May_22_2022_Adder IS
   PORT (Natavio_May_22_2022_dataa     : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
         Natavio_May_22_2022_datab     : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
         Natavio_May_22_2022_result    : OUT STD_LOGIC_VECTOR (31 DOWNTO 0));
END Natavio_May_22_2022_Adder;

ARCHITECTURE SYN OF natavio_may_22_2022_adder IS

   SIGNAL Natavio_May_22_2022_sub_wire0   : STD_LOGIC_VECTOR (31 DOWNTO 0);

   COMPONENT lpm_add_sub
   GENERIC (
      lpm_direction     : STRING;
      lpm_hint    : STRING;
      lpm_representation      : STRING;
      lpm_type    : STRING;
      lpm_width      : NATURAL
   );
   PORT (dataa : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
         datab : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
         result   : OUT STD_LOGIC_VECTOR (31 DOWNTO 0));
   END COMPONENT;

BEGIN
   Natavio_May_22_2022_result <= Natavio_May_22_2022_sub_wire0(31 DOWNTO 0);

   LPM_ADD_SUB_component : LPM_ADD_SUB
   GENERIC MAP (
      lpm_direction => "ADD",
      lpm_hint => "ONE_INPUT_IS_CONSTANT=NO,CIN_USED=NO",
      lpm_representation => "UNSIGNED",
      lpm_type => "LPM_ADD_SUB",
      lpm_width => 32
   )
   PORT MAP (
      dataa => Natavio_May_22_2022_dataa,
      datab => Natavio_May_22_2022_datab,
      result => Natavio_May_22_2022_sub_wire0);

END SYN;
```

5-Bit 2-to-1 Multiplexer

This code is a code design for the 5-bit 2-to-1 MUX. This multiplexer has 5-bit inputs and outputs. It is based on the multiplexer design I made for the BEQ, BNE, J MIPS Operations Lab.

Natavio_May_22_2022_PC.vhd | Natavio_May_22_2022_Adder.vhd | Natavio_May_22_2022_5_bits_2_1_Mux.vhd | Compilati

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Natavio_May_22_2022_5_bits_2_1_Mux IS
   PORT (Natavio_May_22_2022_A       : IN std_logic_vector (4 DOWNTO 0);
         Natavio_May_22_2022_B       : IN std_logic_vector (4 DOWNTO 0);
         Natavio_May_22_2022_SEL     : IN std_logic;
         Natavio_May_22_2022_Out     : OUT std_logic_vector (4 DOWNTO 0));
END Natavio_May_22_2022_5_bits_2_1_Mux;

ARCHITECTURE arch OF Natavio_May_22_2022_5_bits_2_1_Mux IS
   BEGIN
      Natavio_May_22_2022_Out(0) <= ((Natavio_May_22_2022_A(0) AND (NOT Natavio_May_22_2022_SEL)) OR
                                     (Natavio_May_22_2022_B(0) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(1) <= ((Natavio_May_22_2022_A(1) AND (NOT Natavio_May_22_2022_SEL)) OR
                                     (Natavio_May_22_2022_B(1) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(2) <= ((Natavio_May_22_2022_A(2) AND (NOT Natavio_May_22_2022_SEL)) OR
                                     (Natavio_May_22_2022_B(2) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(3) <= ((Natavio_May_22_2022_A(3) AND (NOT Natavio_May_22_2022_SEL)) OR
                                     (Natavio_May_22_2022_B(3) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(4) <= ((Natavio_May_22_2022_A(4) AND (NOT Natavio_May_22_2022_SEL)) OR
                                     (Natavio_May_22_2022_B(4) AND Natavio_May_22_2022_SEL));
END arch;
```

32-Bit 2-to-1 Multiplexer

This code is a code design for the 32-bit 2-to-1 MUX. This multiplexer has 32-bit inputs and outputs. Once again, it is based on the multiplexer design I made for the BEQ, BNE, J MIPS Operations Lab.

Natavio_May_22_2022_32_bit_2_1_Mux.vhd | Compilation Report - Natavio_May_22_2022_Data_Memory

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Natavio_May_22_2022_32_bit_2_1_Mux IS
   PORT (Natavio_May_22_2022_A      : IN std_logic_vector (31 DOWNTO 0);
         Natavio_May_22_2022_B      : IN std_logic_vector (31 DOWNTO 0);
         Natavio_May_22_2022_SEL    : IN std_logic;
         Natavio_May_22_2022_Out    : OUT std_logic_vector (31 DOWNTO 0));
END Natavio_May_22_2022_32_bit_2_1_Mux;

ARCHITECTURE arch OF Natavio_May_22_2022_32_bit_2_1_Mux IS
   BEGIN
      Natavio_May_22_2022_Out(0) <= ((Natavio_May_22_2022_A(0) AND (NOT Natavio_May_22_2022_SEL)) OR
                                     (Natavio_May_22_2022_B(0) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(1) <= ((Natavio_May_22_2022_A(1) AND (NOT Natavio_May_22_2022_SEL)) OR
                                     (Natavio_May_22_2022_B(1) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(2) <= ((Natavio_May_22_2022_A(2) AND (NOT Natavio_May_22_2022_SEL)) OR
                                     (Natavio_May_22_2022_B(2) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(3) <= ((Natavio_May_22_2022_A(3) AND (NOT Natavio_May_22_2022_SEL)) OR
                                     (Natavio_May_22_2022_B(3) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(4) <= ((Natavio_May_22_2022_A(4) AND (NOT Natavio_May_22_2022_SEL)) OR
                                     (Natavio_May_22_2022_B(4) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(5) <= ((Natavio_May_22_2022_A(5) AND (NOT Natavio_May_22_2022_SEL)) OR
                                     (Natavio_May_22_2022_B(5) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(6) <= ((Natavio_May_22_2022_A(6) AND (NOT Natavio_May_22_2022_SEL)) OR
                                     (Natavio_May_22_2022_B(6) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(7) <= ((Natavio_May_22_2022_A(7) AND (NOT Natavio_May_22_2022_SEL)) OR
                                     (Natavio_May_22_2022_B(7) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(8) <= ((Natavio_May_22_2022_A(8) AND (NOT Natavio_May_22_2022_SEL)) OR
                                     (Natavio_May_22_2022_B(8) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(9) <= ((Natavio_May_22_2022_A(9) AND (NOT Natavio_May_22_2022_SEL)) OR
                                     (Natavio_May_22_2022_B(9) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(10) <= ((Natavio_May_22_2022_A(10) AND (NOT Natavio_May_22_2022_SEL)) OR
                                      (Natavio_May_22_2022_B(10) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(11) <= ((Natavio_May_22_2022_A(11) AND (NOT Natavio_May_22_2022_SEL)) OR
                                      (Natavio_May_22_2022_B(11) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(12) <= ((Natavio_May_22_2022_A(12) AND (NOT Natavio_May_22_2022_SEL)) OR
                                      (Natavio_May_22_2022_B(12) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(13) <= ((Natavio_May_22_2022_A(13) AND (NOT Natavio_May_22_2022_SEL)) OR
                                      (Natavio_May_22_2022_B(13) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(14) <= ((Natavio_May_22_2022_A(14) AND (NOT Natavio_May_22_2022_SEL)) OR
                                      (Natavio_May_22_2022_B(14) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(15) <= ((Natavio_May_22_2022_A(15) AND (NOT Natavio_May_22_2022_SEL)) OR
                                      (Natavio_May_22_2022_B(15) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(16) <= ((Natavio_May_22_2022_A(16) AND (NOT Natavio_May_22_2022_SEL)) OR
                                      (Natavio_May_22_2022_B(16) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(17) <= ((Natavio_May_22_2022_A(17) AND (NOT Natavio_May_22_2022_SEL)) OR
                                      (Natavio_May_22_2022_B(17) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(18) <= ((Natavio_May_22_2022_A(18) AND (NOT Natavio_May_22_2022_SEL)) OR
                                      (Natavio_May_22_2022_B(18) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(19) <= ((Natavio_May_22_2022_A(19) AND (NOT Natavio_May_22_2022_SEL)) OR
                                      (Natavio_May_22_2022_B(19) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(20) <= ((Natavio_May_22_2022_A(20) AND (NOT Natavio_May_22_2022_SEL)) OR
                                      (Natavio_May_22_2022_B(20) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(21) <= ((Natavio_May_22_2022_A(21) AND (NOT Natavio_May_22_2022_SEL)) OR
                                      (Natavio_May_22_2022_B(21) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(22) <= ((Natavio_May_22_2022_A(22) AND (NOT Natavio_May_22_2022_SEL)) OR
                                      (Natavio_May_22_2022_B(22) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(23) <= ((Natavio_May_22_2022_A(23) AND (NOT Natavio_May_22_2022_SEL)) OR
                                      (Natavio_May_22_2022_B(23) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(24) <= ((Natavio_May_22_2022_A(24) AND (NOT Natavio_May_22_2022_SEL)) OR
                                      (Natavio_May_22_2022_B(24) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(25) <= ((Natavio_May_22_2022_A(25) AND (NOT Natavio_May_22_2022_SEL)) OR
                                      (Natavio_May_22_2022_B(25) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(26) <= ((Natavio_May_22_2022_A(26) AND (NOT Natavio_May_22_2022_SEL)) OR
                                      (Natavio_May_22_2022_B(26) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(27) <= ((Natavio_May_22_2022_A(27) AND (NOT Natavio_May_22_2022_SEL)) OR
                                      (Natavio_May_22_2022_B(27) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(28) <= ((Natavio_May_22_2022_A(28) AND (NOT Natavio_May_22_2022_SEL)) OR
                                      (Natavio_May_22_2022_B(28) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(29) <= ((Natavio_May_22_2022_A(29) AND (NOT Natavio_May_22_2022_SEL)) OR
                                      (Natavio_May_22_2022_B(29) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(30) <= ((Natavio_May_22_2022_A(30) AND (NOT Natavio_May_22_2022_SEL)) OR
                                      (Natavio_May_22_2022_B(30) AND Natavio_May_22_2022_SEL));
      Natavio_May_22_2022_Out(31) <= ((Natavio_May_22_2022_A(31) AND (NOT Natavio_May_22_2022_SEL)) OR
                                      (Natavio_May_22_2022_B(31) AND Natavio_May_22_2022_SEL));
END arch;
```

Sign Extender

This code is a code design for the Sign Extender. It takes in a 16-bit input as well as the ExtOp signal, and it outputs the sign extended to 32 bits. If the ExtOp is 0, then the sign extended is unsigned. On the other hand, if ExtOp is 1 then the sign extended is signed.

Natavio_May_22_2022_32_bit_2_1_Mux.vhd | Natavio_May_22_2022_Sign_Ext.vhd | Compilation Report - N

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY Natavio_May_22_2022_Sign_Ext IS
   PORT (Natavio_May_22_2022_16_bits   : IN std_logic_vector (15 DOWNTO 0);
         Natavio_May_22_2022_ExtOp     : IN std_logic;
         Natavio_May_22_2022_32_bits   : OUT std_logic_vector (31 DOWNTO 0));
END Natavio_May_22_2022_Sign_Ext;

ARCHITECTURE arch OF Natavio_May_22_2022_Sign_Ext IS

   BEGIN
      PROCESS (Natavio_May_22_2022_ExtOp)
         BEGIN
            IF (Natavio_May_22_2022_ExtOp = '0') THEN
               Natavio_May_22_2022_32_bits <= std_logic_vector(resize(unsigned(Natavio_May_22_2022_16_bits), 32));
            ELSE
               Natavio_May_22_2022_32_bits <= std_logic_vector(resize(signed(Natavio_May_22_2022_16_bits), 32));
            END IF;
      END PROCESS;
END arch;
```

Logical Bitwise Operation Unit

This code is a code design for the Logical Bitwise Operation Unit. It is meant to compute all arithmetic operations handled by the Arithmetic Logic Unit (ALU) component. It is the same design as I had on the ALU Lab. The only change made is the prefixes.

Natavio_May_22_2022_Sign_Ext.vhd | Natavio_May_22_2022_Logical_Bitwise_Op.vhd | Natavio_May_22_2022_addsub_ZNO.vhd | Natavio_May

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;
USE ieee.numeric_std.all;

ENTITY Natavio_May_22_2022_Logical_Bitwise_Op IS
   PORT (Natavio_May_22_2022_OP     : IN std_logic_vector (3 DOWNTO 0);
         Natavio_May_22_2022_A      : IN std_logic_vector (31 DOWNTO 0);
         Natavio_May_22_2022_B      : IN std_logic_vector (31 DOWNTO 0);
         Natavio_May_22_2022_Ext    : IN std_logic_vector (31 DOWNTO 0);
         Natavio_May_22_2022_Shamt  : IN std_logic_vector (15 DOWNTO 0);
         Natavio_May_22_2022_Res    : OUT std_logic_vector (31 DOWNTO 0));
END Natavio_May_22_2022_Logical_Bitwise_Op;

ARCHITECTURE arch OF Natavio_May_22_2022_Logical_Bitwise_Op IS
   BEGIN
      PROCESS (Natavio_May_22_2022_OP, Natavio_May_22_2022_A, Natavio_May_22_2022_B)
         BEGIN
            IF (Natavio_May_22_2022_OP = "0011") THEN -- AND
               Natavio_May_22_2022_Res <= Natavio_May_22_2022_A AND Natavio_May_22_2022_B;
            ELSIF (Natavio_May_22_2022_OP = "0100") then -- ANDi
               Natavio_May_22_2022_Res <= Natavio_May_22_2022_A AND Natavio_May_22_2022_Ext;
            ELSIF (Natavio_May_22_2022_OP = "0110") then -- NOR
               Natavio_May_22_2022_Res <= Natavio_May_22_2022_A NOR Natavio_May_22_2022_B;
            ELSIF (Natavio_May_22_2022_OP = "0101") then -- ORi
               Natavio_May_22_2022_Res <= Natavio_May_22_2022_A OR Natavio_May_22_2022_ext;
            ELSIF (Natavio_May_22_2022_OP = "0111") THEN -- SLL
               Natavio_May_22_2022_Res <= std_logic_vector(shift_left(unsigned(Natavio_May_22_2022_B),
                                                 to_integer(unsigned(Natavio_May_22_2022_Shamt))));
            ELSIF (Natavio_May_22_2022_OP = "1000") THEN -- SRL
               Natavio_May_22_2022_Res <= std_logic_vector(shift_right(unsigned(Natavio_May_22_2022_B),
                                                 to_integer(unsigned(Natavio_May_22_2022_Shamt))));
            ELSIF (Natavio_May_22_2022_OP = "1001") THEN -- SRA
               Natavio_May_22_2022_Res <= std_logic_vector(shift_right(signed(Natavio_May_22_2022_B),
                                                 to_integer(unsigned(Natavio_May_22_2022_Shamt))));
            ELSE
               Natavio_May_22_2022_Res <= x"00000000";
            END IF;
      END PROCESS;
END arch;
```

Add/Sub

This code is a code design for the Arithmetic Logic Unit's Add/Sub component. It is meant to compute addition and subtraction. Once again, this is the exact same design as I had on the ALU Lab. The only change made is the prefixes.

Natavio_May_22_2022_addsub_ZNO.vhd | Compilation Report - Natavio_May_22_2022_Data_Memory

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Natavio_May_22_2022_addsub_ZNO IS
    GENERIC (Natavio_May_22_2022_N    :  integer := 32);
    PORT (Natavio_May_22_2022_OP      :  IN std_logic;
          Natavio_May_22_2022_A       :  IN std_logic_vector (Natavio_May_22_2022_N-1 DOWNTO 0);
          Natavio_May_22_2022_B       :  IN std_logic_vector (Natavio_May_22_2022_N-1 DOWNTO 0);
          Natavio_May_22_2022_Result  :  OUT std_logic_vector (Natavio_May_22_2022_N-1 DOWNTO 0);
          Natavio_May_22_2022_COUT    :  OUT std_logic;
          Natavio_May_22_2022_Overflow, Natavio_May_22_2022_Zero, Natavio_May_22_2022_Negative   :  OUT std_logic);
END Natavio_May_22_2022_addsub_ZNO;

Architecture arch OF Natavio_May_22_2022_addsub_ZNO is
    SIGNAL Natavio_May_22_2022_index : std_logic_vector(Natavio_May_22_2022_N DOWNTO 0);
    SIGNAL Natavio_May_22_2022_Z        : std_logic_vector(Natavio_May_22_2022_N DOWNTO 0);
    SIGNAL Natavio_May_22_2022_var      : std_logic_vector(2 DOWNTO 0);
    SIGNAL Natavio_May_22_2022_out      : std_logic_vector(Natavio_May_22_2022_N DOWNTO 0);
    SIGNAL Natavio_May_22_2022_Z_flag   : std_logic;
    SIGNAL Natavio_May_22_2022_p0, Natavio_May_22_2022_p1,
           Natavio_May_22_2022_p2 : std_logic_vector(Natavio_May_22_2022_N DOWNTO 0);

    BEGIN
       PROCESS(Natavio_May_22_2022_OP, Natavio_May_22_2022_A, Natavio_May_22_2022_B, Natavio_May_22_2022_index,
               Natavio_May_22_2022_p0, Natavio_May_22_2022_p1, Natavio_May_22_2022_p2, Natavio_May_22_2022_out)
          BEGIN
             Natavio_May_22_2022_index(0) <= Natavio_May_22_2022_OP;
             Natavio_May_22_2022_Z(0) <= '1';

             FOR i IN 0 TO Natavio_May_22_2022_N-1 LOOP
                Natavio_May_22_2022_p0(i) <= Natavio_May_22_2022_B(i) XOR Natavio_May_22_2022_OP;
                Natavio_May_22_2022_p1(i) <= Natavio_May_22_2022_p0(i) XOR Natavio_May_22_2022_A(i);
                Natavio_May_22_2022_Result(i) <= Natavio_May_22_2022_p1(i) XOR Natavio_May_22_2022_index(i);
                Natavio_May_22_2022_out(i) <= Natavio_May_22_2022_p1(i) XOR Natavio_May_22_2022_index(i);
                Natavio_May_22_2022_Z(i+1) <= Natavio_May_22_2022_Z(i) AND (NOT ((Natavio_May_22_2022_p1(i) XOR
                                                                                  Natavio_May_22_2022_index(i))));

                Natavio_May_22_2022_p2(i) <= Natavio_May_22_2022_index(i) AND Natavio_May_22_2022_p1(i);
                Natavio_May_22_2022_index(i+1) <= ((Natavio_May_22_2022_A(i) AND Natavio_May_22_2022_p0(i)) OR
                                                   Natavio_May_22_2022_p2(i));
             END LOOP;

             IF (Natavio_May_22_2022_out = "00000000000000000000000000000000") THEN
                Natavio_May_22_2022_Z_flag <= '1';
             ELSE
                Natavio_May_22_2022_Z_flag <= '0';
             END IF;

       END PROCESS;

       Natavio_May_22_2022_COUT <= Natavio_May_22_2022_index(Natavio_May_22_2022_N);

       Natavio_May_22_2022_Zero <= Natavio_May_22_2022_Z_flag;

       Natavio_May_22_2022_Overflow <= (Natavio_May_22_2022_index(Natavio_May_22_2022_N) XOR
                                        Natavio_May_22_2022_index(Natavio_May_22_2022_N-1));

       Natavio_May_22_2022_var(0) <= (Natavio_May_22_2022_A(Natavio_May_22_2022_N-1) OR
                                      (Natavio_May_22_2022_B(Natavio_May_22_2022_N-1) XOR Natavio_May_22_2022_OP));

       Natavio_May_22_2022_var(1) <= (Natavio_May_22_2022_index(Natavio_May_22_2022_N) XOR
                                      Natavio_May_22_2022_index(Natavio_May_22_2022_N-1));

       Natavio_May_22_2022_var(2) <= (Natavio_May_22_2022_p1(Natavio_May_22_2022_N-1) XOR
                                      Natavio_May_22_2022_index(Natavio_May_22_2022_N-1));

       Natavio_May_22_2022_Negative <= Natavio_May_22_2022_var(0) AND (Natavio_May_22_2022_var(1) XOR
                                                                       Natavio_May_22_2022_var(2));
END arch;
```

Arithmetic Logic Unit (ALU)

This code is a code design for the Arithmetic Logic Unit (ALU). This component is meant to handle all arithmetic instructions. This code design is based on the ALU lab previously done. Since the registers are handled by the CPU and not the ALU in this case, I did not add those components. Instead the ALU is now made up of the Sign Extender, Bitwise Logic Operation Unit, and Add/Sub components. First, the ALU will get the sign extension. Then, it will carry out all arithmetic instructions (i.e. add, addi, addiu, addu, sub, subu). After this, it will carry out the Logical operations (i.e. and, andi, nor, ori, sll, srl, and sra). Finally, it will output the result of the instruction carried out based on the instruction's ALUCtrl value, which is set in the CPU Controller.

Natavio_May_22_2022_Sign_Ext.vhd | Natavio_May_22_2022_Logical_Bitwise_Op.vhd | Natavio_May_22_2022_addsub_ZNO.vhd | Natavio_May_22_2022_ALU.vhd

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Natavio_May_22_2022_ALU IS
   PORT (Natavio_May_22_2022_ALUCtrl   : IN std_logic_vector (3 DOWNTO 0);
         Natavio_May_22_2022_Operand1  : IN std_logic_vector (31 DOWNTO 0);
         Natavio_May_22_2022_Operand2  : IN std_logic_vector (31 DOWNTO 0);
         Natavio_May_22_2022_upper_res : OUT std_logic_vector (31 DOWNTO 0);
         Natavio_May_22_2022_lower_res : OUT std_logic_vector (31 DOWNTO 0);
         Natavio_May_22_2022_Overflow  : OUT std_logic;
         Natavio_May_22_2022_Zero      : OUT std_logic;
         Natavio_May_22_2022_Negative  : OUT std_logic;
         Natavio_May_22_2022_Cout      : OUT std_logic);
END Natavio_May_22_2022_ALU;

ARCHITECTURE arch OF Natavio_May_22_2022_ALU IS
   SIGNAL Natavio_May_22_2022_res   : std_logic_vector (63 DOWNTO 0);
   SIGNAL Natavio_May_22_2022_ext, Natavio_May_22_2022_ext2 : std_logic_vector (31 DOWNTO 0);
   SIGNAL Natavio_May_22_2022_imm_o : std_logic_vector (15 DOWNTO 0);
   SIGNAL Natavio_May_22_2022_ext0, Natavio_May_22_2022_ext1   : std_logic_vector (15 DOWNTO 0);

   -- All arithmetic flags
   SIGNAL Natavio_May_22_2022_O0, Natavio_May_22_2022_Z0, Natavio_May_22_2022_N0,
          Natavio_May_22_2022_O1, Natavio_May_22_2022_Z1, Natavio_May_22_2022_N1,
          Natavio_May_22_2022_O2, Natavio_May_22_2022_Z2, Natavio_May_22_2022_N2 : std_logic := '0';

   -- All arithmetic carryouts
   SIGNAL Natavio_May_22_2022_Cout0, Natavio_May_22_2022_Cout1, Natavio_May_22_2022_Cout2 : std_logic := '0';

   -- All Opertaion outputs
   SIGNAL Natavio_May_22_2022_temp0, Natavio_May_22_2022_temp1,
          Natavio_May_22_2022_temp2, Natavio_May_22_2022_temp3 : std_logic_vector (31 DOWNTO 0);

                                        -- Components
      -- Sign Extension Unit
   COMPONENT Natavio_May_22_2022_Sign_Ext
      PORT (Natavio_May_22_2022_16_bits   : IN std_logic_vector (15 DOWNTO 0);
            Natavio_May_22_2022_Extop     : IN std_logic;
            Natavio_May_22_2022_32_bits   : OUT std_logic_vector (31 DOWNTO 0));
   END COMPONENT;

   -- Logical Bitwise Operation Unit
   COMPONENT Natavio_May_22_2022_Logical_Bitwise_Op
      PORT (Natavio_May_22_2022_OP     : IN std_logic_vector (3 DOWNTO 0);
            Natavio_May_22_2022_A      : IN std_logic_vector (31 DOWNTO 0);
            Natavio_May_22_2022_B      : IN std_logic_vector (31 DOWNTO 0);
            Natavio_May_22_2022_Ext    : IN std_logic_vector (31 DOWNTO 0);
            Natavio_May_22_2022_Shamt  : IN std_logic_vector (15 DOWNTO 0);
            Natavio_May_22_2022_Res    : OUT std_logic_vector (31 DOWNTO 0));
   END COMPONENT;

   -- Add/Sub Unit W/ Zero, Negative, & Overflow Flags
   component Natavio_May_22_2022_addsub_ZNO
      GENERIC (Natavio_May_22_2022_N   :  integer := 32);
```

```vhdl
      PORT (Natavio_May_22_2022_OP     :  IN std_logic;
            Natavio_May_22_2022_A      :  IN std_logic_vector (Natavio_May_22_2022_N-1 DOWNTO 0);
            Natavio_May_22_2022_B      :  IN std_logic_vector (Natavio_May_22_2022_N-1 DOWNTO 0);
            Natavio_May_22_2022_Result :  OUT std_logic_vector (Natavio_May_22_2022_N-1 DOWNTO 0);
            Natavio_May_22_2022_COUT   :  OUT std_logic;
            Natavio_May_22_2022_Overflow, Natavio_May_22_2022_Zero, Natavio_May_22_2022_Negative   :  OUT std_logic);
   END COMPONENT;

   BEGIN
      Natavio_May_22_2022_imm_o <= Natavio_May_22_2022_Operand2(15 DOWNTO 0);

      -- Extension
      Natavio_May_22_2022_extension : Natavio_May_22_2022_Sign_Ext PORT MAP (Natavio_May_22_2022_imm_o, '0', Natavio_May_22_2022_ex

      Natavio_May_22_2022_ext0 <= Natavio_May_22_2022_ext(31 DOWNTO 16);
      Natavio_May_22_2022_ext1 <= Natavio_May_22_2022_imm_o;
      Natavio_May_22_2022_ext2 <= Natavio_May_22_2022_ext0 & Natavio_May_22_2022_ext1;

      -- Arithmetic Operations
      Natavio_May_22_2022_add : Natavio_May_22_2022_addsub_ZNO GENERIC MAP (Natavio_May_22_2022_N => 32) -- add/addu
                                                               PORT MAP ('0', Natavio_May_22_2022_Operand1,
                                                                         Natavio_May_22_2022_Operand2, Natavio_May_22_2022_temp0,
                                                                         Natavio_May_22_2022_Cout0, Natavio_May_22_2022_O0,
                                                                         Natavio_May_22_2022_Z0, Natavio_May_22_2022_N0);

      Natavio_May_22_2022_addi : Natavio_May_22_2022_addsub_ZNO GENERIC MAP (Natavio_May_22_2022_N => 32) --addi/addiu
                                                               PORT MAP ('0', Natavio_May_22_2022_Operand1,
                                                                         Natavio_May_22_2022_ext2, Natavio_May_22_2022_temp1,
                                                                         Natavio_May_22_2022_Cout1, Natavio_May_22_2022_O1,
                                                                         Natavio_May_22_2022_Z1, Natavio_May_22_2022_N1);

      Natavio_May_22_2022_sub : Natavio_May_22_2022_addsub_ZNO GENERIC MAP (Natavio_May_22_2022_N => 32) --sub/subu
                                                               PORT MAP ('1', Natavio_May_22_2022_Operand1,
```

```
                                                                    Natavio_May_22_2022_Operand2, Natavio_May_22_2022_temp2,
                                                                    Natavio_May_22_2022_Cout2, Natavio_May_22_2022_O2,
                                                                    Natavio_May_22_2022_Z2, Natavio_May_22_2022_N2);

    -- Logical Operations
    Natavio_May_22_2022_bit_op : Natavio_May_22_2022_Logical_Bitwise_Op PORT MAP (Natavio_May_22_2022_ALUCtrl,
                                                                                  Natavio_May_22_2022_Operand1,
                                                                                  Natavio_May_22_2022_Operand2,
                                                                                  Natavio_May_22_2022_ext2,
                                                                                  Natavio_May_22_2022_imm_o,
                                                                                  Natavio_May_22_2022_temp3);

    PROCESS (Natavio_May_22_2022_ALUCtrl, Natavio_May_22_2022_temp0, Natavio_May_22_2022_temp1, Natavio_May_22_2022_temp2,
             Natavio_May_22_2022_temp3, Natavio_May_22_2022_ext2, Natavio_May_22_2022_imm_o,
             Natavio_May_22_2022_O0, Natavio_May_22_2022_Z0, Natavio_May_22_2022_N0, Natavio_May_22_2022_Cout0,
             Natavio_May_22_2022_O1, Natavio_May_22_2022_Z1, Natavio_May_22_2022_N1, Natavio_May_22_2022_Cout1,
             Natavio_May_22_2022_O2, Natavio_May_22_2022_Z2, Natavio_May_22_2022_N2, Natavio_May_22_2022_Cout2)
        BEGIN
           CASE Natavio_May_22_2022_ALUCtrl IS
              WHEN "0000" => -- add/addu/sw/lw
```

```
                 Natavio_May_22_2022_upper_res <= x"00000000";
                 Natavio_May_22_2022_lower_res <= Natavio_May_22_2022_temp0;
                 Natavio_May_22_2022_Overflow <= Natavio_May_22_2022_O0;
                 Natavio_May_22_2022_Zero <= Natavio_May_22_2022_Z0;
                 Natavio_May_22_2022_Negative <= Natavio_May_22_2022_N0;
                 Natavio_May_22_2022_Cout <= Natavio_May_22_2022_Cout0;

              WHEN "0001" => -- addi/addiu
                 Natavio_May_22_2022_upper_res <= x"00000000";
                 Natavio_May_22_2022_lower_res <= Natavio_May_22_2022_temp1;
                 Natavio_May_22_2022_Overflow <= Natavio_May_22_2022_O1;
                 Natavio_May_22_2022_Zero <= Natavio_May_22_2022_Z1;
                 Natavio_May_22_2022_Negative <= Natavio_May_22_2022_N1;
                 Natavio_May_22_2022_Cout <= Natavio_May_22_2022_Cout1;

              WHEN "0010" => -- sub/subu/beq/bne
                 Natavio_May_22_2022_upper_res <= x"00000000";
                 Natavio_May_22_2022_lower_res <= Natavio_May_22_2022_temp2;
                 Natavio_May_22_2022_Overflow <= Natavio_May_22_2022_O2;
                 Natavio_May_22_2022_Zero <= Natavio_May_22_2022_Z2;
                 Natavio_May_22_2022_Negative <= Natavio_May_22_2022_N2;
                 Natavio_May_22_2022_Cout <= Natavio_May_22_2022_Cout2;

              WHEN "0011" | "0100" | "0101" | "0110" | "0111" | "1000" | "1001" => -- and/andi/ori/nor/sll/srl/sra
                 Natavio_May_22_2022_upper_res <= x"00000000";
                 Natavio_May_22_2022_lower_res <= Natavio_May_22_2022_temp3;

              WHEN others => -- base case
                 Natavio_May_22_2022_upper_res <= (others => 'Z');
                 Natavio_May_22_2022_lower_res <= (others => 'Z');
           END CASE;
    END PROCESS;
END arch;
```

CPU Controller Unit

This code is a code design for the CPU Controller. As explained throughout the report, this component will extract the current instruction being read and then determine the control signals for the instruction. These control signals will indicate to the CPU what the instruction is and what the desired outputs are to be produced.

Natavio_May_22_2022_CPU_Controller.vhd Compilation Report - Natavio_May_22_202

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Natavio_May_22_2022_CPU_Controller IS
   PORT (Natavio_May_22_2022_OPCode     : in std_logic_vector (5 DOWNTO 0);
         Natavio_May_22_2022_Funct      : in std_logic_vector (5 DOWNTO 0);
         Natavio_May_22_2022_ExtOp      : out std_logic;
         Natavio_May_22_2022_ALUCtrl    : out std_logic_vector (3 DOWNTO 0);
         Natavio_May_22_2022_RegWr      : out std_logic;
         Natavio_May_22_2022_RegDst     : out std_logic;
         Natavio_May_22_2022_ALUSrc     : out std_logic;
         Natavio_May_22_2022_MemToReg   : out std_logic;
         Natavio_May_22_2022_MemWr      : out std_logic;
         Natavio_May_22_2022_Branch     : out std_logic;
         Natavio_May_22_2022_Jump       : out std_logic);
END Natavio_May_22_2022_CPU_Controller;

ARCHITECTURE arch OF Natavio_May_22_2022_CPU_Controller IS
   SIGNAL Natavio_May_22_2022_ALU_OP    : std_logic_vector (2 DOWNTO 0);

   BEGIN
      PROCESS(Natavio_May_22_2022_OPCode)
         BEGIN
            CASE Natavio_May_22_2022_OPCode IS
               -- R-Type Instructions
               WHEN "000000" => -- add/addu/sub/subu/and/nor/sll/srl/sra
                  Natavio_May_22_2022_ALU_OP <="000";
                  Natavio_May_22_2022_RegWr <= '1';
                  Natavio_May_22_2022_RegDst <= '1';
```

```
30                  Natavio_May_22_2022_ExtOp <= '0';
31                  Natavio_May_22_2022_ALUSrc <= '0';
32                  Natavio_May_22_2022_MemWr <= '0';
33                  Natavio_May_22_2022_MemToReg <= '0';
34                  Natavio_May_22_2022_Jump <= '0';
35                  Natavio_May_22_2022_Branch <= '0';
36
37               -- I-Type Instructions
38               WHEN "000100" | "000101" => -- addi/addiu
39                  Natavio_May_22_2022_ALU_OP <="001";
40                  Natavio_May_22_2022_RegWr <= '1';
41                  Natavio_May_22_2022_RegDst <= '0';
42                  Natavio_May_22_2022_ExtOp <= '1';
43                  Natavio_May_22_2022_ALUSrc <= '1';
44                  Natavio_May_22_2022_MemWr <= '0';
45                  Natavio_May_22_2022_MemToReg <= '0';
46                  Natavio_May_22_2022_Jump <= '0';
47                  Natavio_May_22_2022_Branch <= '0';
48               WHEN "000110" => -- andi
49                  Natavio_May_22_2022_ALU_OP <="010";
50                  Natavio_May_22_2022_RegWr <= '1';
51                  Natavio_May_22_2022_RegDst <= '0';
52                  Natavio_May_22_2022_ExtOp <= '0';
53                  Natavio_May_22_2022_ALUSrc <= '1';
54                  Natavio_May_22_2022_MemWr <= '0';
```

```
55                  Natavio_May_22_2022_MemToReg <= '0';
56                  Natavio_May_22_2022_Jump <= '0';
57                  Natavio_May_22_2022_Branch <= '0';
58               WHEN "000111" => -- ori
59                  Natavio_May_22_2022_ALU_OP <="011";
60                  Natavio_May_22_2022_RegWr <= '1';
61                  Natavio_May_22_2022_RegDst <= '0';
62                  Natavio_May_22_2022_ExtOp <= '0';
63                  Natavio_May_22_2022_ALUSrc <= '1';
64                  Natavio_May_22_2022_MemWr <= '0';
65                  Natavio_May_22_2022_MemToReg <= '0';
66                  Natavio_May_22_2022_Jump <= '0';
67                  Natavio_May_22_2022_Branch <= '0';
68               WHEN "001000" => -- sw
69                  Natavio_May_22_2022_ALU_OP <="100";
70                  Natavio_May_22_2022_RegWr <= '0';
71                  Natavio_May_22_2022_RegDst <= '0';
72                  Natavio_May_22_2022_ExtOp <= '1';
73                  Natavio_May_22_2022_ALUSrc <= '1';
74                  Natavio_May_22_2022_MemWr <= '1';
75                  Natavio_May_22_2022_MemToReg <= '0';
76                  Natavio_May_22_2022_Jump <= '0';
77                  Natavio_May_22_2022_Branch <= '0';
78               WHEN "001001" => -- lw
79                  Natavio_May_22_2022_ALU_OP <="101";
80                  Natavio_May_22_2022_RegWr <= '1';
81                  Natavio_May_22_2022_RegDst <= '0';
82                  Natavio_May_22_2022_ExtOp <= '1';
83                  Natavio_May_22_2022_ALUSrc <= '1';
84                  Natavio_May_22_2022_MemWr <= '0';
85                  Natavio_May_22_2022_MemToReg <= '1';
86                  Natavio_May_22_2022_Jump <= '0';
87                  Natavio_May_22_2022_Branch <= '0';
88               WHEN "001010" | "001011" => -- beq/bne
89                  Natavio_May_22_2022_ALU_OP <="110";
90                  Natavio_May_22_2022_RegWr <= '0';
91                  Natavio_May_22_2022_RegDst <= '0';
92                  Natavio_May_22_2022_ExtOp <= '1';
93                  Natavio_May_22_2022_ALUSrc <= '0';
94                  Natavio_May_22_2022_MemWr <= '0';
95                  Natavio_May_22_2022_MemToReg <= '0';
96                  Natavio_May_22_2022_Jump <= '0';
97                  Natavio_May_22_2022_Branch <= '1';
98
99               -- J-Type Instruction
100              WHEN "000010" => -- j
101                 Natavio_May_22_2022_ALU_OP <="111";
102                 Natavio_May_22_2022_RegWr <= '0';
103                 Natavio_May_22_2022_RegDst <= '0';
104                 Natavio_May_22_2022_ExtOp <= '0';
105                 Natavio_May_22_2022_ALUSrc <= '0';
106                 Natavio_May_22_2022_MemWr <= '0';
107                 Natavio_May_22_2022_MemToReg <= '0';
```

```
108                 Natavio_May_22_2022_Jump <= '1';
109                 Natavio_May_22_2022_Branch <= '0';
110              WHEN others => -- base case
111                 Natavio_May_22_2022_ALU_OP <="ZZZ";
112                 Natavio_May_22_2022_RegWr <= 'Z';
113                 Natavio_May_22_2022_RegDst <= 'Z';
114                 Natavio_May_22_2022_ExtOp <= 'Z';
115                 Natavio_May_22_2022_ALUSrc <= 'Z';
116                 Natavio_May_22_2022_MemWr <= 'Z';
117                 Natavio_May_22_2022_MemToReg <= 'Z';
118                 Natavio_May_22_2022_Jump <= 'Z';
119                 Natavio_May_22_2022_Branch <= 'Z';
120           END CASE;
121        END PROCESS;
```

```
122
123         PROCESS(Natavio_May_22_2022_ALU_OP, Natavio_May_22_2022_Funct)
124            BEGIN
125               -- R-Type Instructions
126               IF (Natavio_May_22_2022_ALU_OP = "000" AND (Natavio_May_22_2022_Funct = "000000" OR
127                                                          Natavio_May_22_2022_Funct = "000001")) THEN
128                  Natavio_May_22_2022_ALUCtrl <= "0000"; -- add/addu
129               ELSIF (Natavio_May_22_2022_ALU_OP = "000" AND (Natavio_May_22_2022_Funct = "000010" OR
130                                                            Natavio_May_22_2022_Funct = "000011")) THEN
131                  Natavio_May_22_2022_ALUCtrl <= "0010"; -- sub/subu
132               ELSIF (Natavio_May_22_2022_ALU_OP = "000" AND Natavio_May_22_2022_Funct = "000100") THEN
133                  Natavio_May_22_2022_ALUCtrl <= "0011"; -- and
134               ELSIF (Natavio_May_22_2022_ALU_OP = "000" AND Natavio_May_22_2022_Funct = "000101") THEN
135                  Natavio_May_22_2022_ALUCtrl <= "0110"; -- nor
136               ELSIF (Natavio_May_22_2022_ALU_OP = "000" AND Natavio_May_22_2022_Funct = "000110") THEN
137                  Natavio_May_22_2022_ALUCtrl <= "0111"; -- sll
138               ELSIF (Natavio_May_22_2022_ALU_OP = "000" AND Natavio_May_22_2022_Funct = "000111") THEN
139                  Natavio_May_22_2022_ALUCtrl <= "1000"; -- srl
140               ELSIF (Natavio_May_22_2022_ALU_OP = "000" AND Natavio_May_22_2022_Funct = "001000") THEN
141                  Natavio_May_22_2022_ALUCtrl <= "1001"; -- sra
142
143               -- I-Type Instructions
144               ELSIF (Natavio_May_22_2022_ALU_OP = "001") THEN
145                  Natavio_May_22_2022_ALUCtrl <= "0001"; -- addi/addiu
146               ELSIF (Natavio_May_22_2022_ALU_OP = "010") THEN
147                  Natavio_May_22_2022_ALUCtrl <= "0100"; -- andi
148               ELSIF (Natavio_May_22_2022_ALU_OP = "011") THEN
149                  Natavio_May_22_2022_ALUCtrl <= "0101"; -- ori
150               ELSIF (Natavio_May_22_2022_ALU_OP = "100" OR Natavio_May_22_2022_ALU_OP = "101") THEN
151                  Natavio_May_22_2022_ALUCtrl <= "0000"; -- sw/lw
152               ELSIF (Natavio_May_22_2022_ALU_OP = "110") THEN
153                  Natavio_May_22_2022_ALUCtrl <= "0010"; -- beq/bne
154               END IF;
155         END PROCESS;
156      END arch;
```

CPU Components Package

Now almost all components needed for the CPU have been created. To make the code shorter on the final CPU VHDL file, I made a components package, which contains every single component created.

Natavio_May_22_2022_CPU_Controller.vhd | Natavio_May_22_2022_CPU_Components.vhd | Compilation Report - Natavio_May_22_2022_Data_Memory

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   PACKAGE Natavio_May_22_2022_CPU_Components IS
5      -- Data Memory
6      COMPONENT Natavio_May_22_2022_Data_Memory IS
7         PORT(Natavio_May_22_2022_clk     : IN std_logic;
8              Natavio_May_22_2022_wren    : IN std_logic;
9              Natavio_May_22_2022_rden    : IN std_logic;
10             Natavio_May_22_2022_data    : IN std_logic_vector (31 DOWNTO 0);
11             Natavio_May_22_2022_address : IN std_logic_vector (31 DOWNTO 0);
12             Natavio_May_22_2022_Out     : OUT std_logic_vector (31 DOWNTO 0));
13     END COMPONENT;
14
15     -- Instruction Register (IR)
16     COMPONENT Natavio_May_22_2022_IR IS
17        PORT (Natavio_May_22_2022_clk          : IN std_logic;
18              Natavio_May_22_2022_wren         : IN std_logic;
19              Natavio_May_22_2022_rden         : IN std_logic;
20              Natavio_May_22_2022_chen         : IN std_logic;
21              Natavio_May_22_2022_Instruction  : IN std_logic_vector (31 DOWNTO 0);
22              Natavio_May_22_2022_IR_q         : OUT std_logic_vector (31 DOWNTO 0));
23     END COMPONENT;
24
25     -- Instruction Memory
26     COMPONENT Natavio_May_22_2022_Instruction_Memory IS
27        PORT (Natavio_May_22_2022_inst_addr : IN std_logic_vector (31 DOWNTO 0);
28              Natavio_May_22_2022_OPCode    : OUT std_logic_vector (5 DOWNTO 0);
29              Natavio_May_22_2022_RS        : OUT std_logic_vector (4 DOWNTO 0);
30              Natavio_May_22_2022_RT        : OUT std_logic_vector (4 DOWNTO 0);
31              Natavio_May_22_2022_RD        : OUT std_logic_vector (4 DOWNTO 0);
32              Natavio_May_22_2022_shamt     : OUT std_logic_vector (4 DOWNTO 0);
33              Natavio_May_22_2022_funct     : OUT std_logic_vector (5 DOWNTO 0);
34              Natavio_May_22_2022_Imm16     : OUT std_logic_vector (15 DOWNTO 0);
35              Natavio_May_22_2022_J_Addr    : OUT std_logic_vector (31 DOWNTO 0);
36              Natavio_May_22_2022_Inst      : OUT std_logic_vector (31 DOWNTO 0));
37     END COMPONENT;
38
39     -- Register File (RF)
40     COMPONENT Natavio_May_22_2022_RF IS
41        PORT (Natavio_May_22_2022_clk     : IN std_logic;
42              Natavio_May_22_2022_wren    : IN std_logic;
43              Natavio_May_22_2022_data    : IN std_logic_vector (31 DOWNTO 0);
44              Natavio_May_22_2022_wr      : IN std_logic_vector (4 DOWNTO 0);
45              Natavio_May_22_2022_rdA     : IN std_logic_vector (4 DOWNTO 0);
46              Natavio_May_22_2022_rdB     : IN std_logic_vector (4 DOWNTO 0);
47              Natavio_May_22_2022_qA      : OUT std_logic_vector (31 DOWNTO 0);
48              Natavio_May_22_2022_qB      : OUT std_logic_vector (31 DOWNTO 0));
49     END COMPONENT;
50
```

```
    -- Program Counter (PC)
    COMPONENT Natavio_May_22_2022_PC IS
        PORT (Natavio_May_22_2022_clk     : IN std_logic;
              Natavio_May_22_2022_PC_I    : IN std_logic_vector (31 DOWNTO 0);
              Natavio_May_22_2022_PC_O    : OUT std_logic_vector (31 DOWNTO 0));
    END COMPONENT;

    -- Adder
    COMPONENT Natavio_May_22_2022_Adder IS
        PORT (Natavio_May_22_2022_dataa      : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
              Natavio_May_22_2022_datab      : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
              Natavio_May_22_2022_result     : OUT STD_LOGIC_VECTOR (31 DOWNTO 0));
    END COMPONENT;

    -- 5-Bit 2:1 MUX
    COMPONENT Natavio_May_22_2022_5_bits_2_1_Mux IS
        PORT (Natavio_May_22_2022_A       : IN std_logic_vector (4 DOWNTO 0);
              Natavio_May_22_2022_B       : IN std_logic_vector (4 DOWNTO 0);
              Natavio_May_22_2022_SEL     : IN std_logic;
              Natavio_May_22_2022_Out     : OUT std_logic_vector (4 DOWNTO 0));
    END COMPONENT;

    -- 32-Bit 2:1 MUX
    COMPONENT Natavio_May_22_2022_32_bit_2_1_Mux IS
        PORT (Natavio_May_22_2022_A       : IN std_logic_vector (31 DOWNTO 0);
              Natavio_May_22_2022_B       : IN std_logic_vector (31 DOWNTO 0);
              Natavio_May_22_2022_SEL     : IN std_logic;
              Natavio_May_22_2022_Out     : OUT std_logic_vector (31 DOWNTO 0));
    END COMPONENT;

    -- Sign Extender
    COMPONENT Natavio_May_22_2022_Sign_Ext IS
        PORT (Natavio_May_22_2022_16_bits    : IN std_logic_vector (15 DOWNTO 0);
              Natavio_May_22_2022_ExtOp      : IN std_logic;
              Natavio_May_22_2022_32_bits    : OUT std_logic_vector (31 DOWNTO 0));
    END COMPONENT;

    -- Logical Bitwise Operation Unit
    COMPONENT Natavio_May_22_2022_Logical_Bitwise_Op IS
        PORT (Natavio_May_22_2022_OP      : IN std_logic_vector (3 DOWNTO 0);
              Natavio_May_22_2022_A       : IN std_logic_vector (31 DOWNTO 0);
              Natavio_May_22_2022_B       : IN std_logic_vector (31 DOWNTO 0);
              Natavio_May_22_2022_Ext     : IN std_logic_vector (31 DOWNTO 0);
              Natavio_May_22_2022_Shamt   : IN std_logic_vector (15 DOWNTO 0);
              Natavio_May_22_2022_Res     : OUT std_logic_vector (31 DOWNTO 0));
    END COMPONENT;

    -- Add/Sub
    COMPONENT Natavio_May_22_2022_addsub_ZNO IS
        GENERIC (Natavio_May_22_2022_N    :  integer := 32);
        PORT (Natavio_May_22_2022_OP      :  IN std_logic;
              Natavio_May_22_2022_A       :  IN std_logic_vector (Natavio_May_22_2022_N-1 DOWNTO 0);
              Natavio_May_22_2022_B       :  IN std_logic_vector (Natavio_May_22_2022_N-1 DOWNTO 0);
              Natavio_May_22_2022_Result  :  OUT std_logic_vector (Natavio_May_22_2022_N-1 DOWNTO 0);
              Natavio_May_22_2022_COUT    :  OUT std_logic;
              Natavio_May_22_2022_Overflow, Natavio_May_22_2022_Zero, Natavio_May_22_2022_Negative    :  OUT std_logic);
    END COMPONENT;

    -- Arithmetic Logic Unit (ALU)
    COMPONENT Natavio_May_22_2022_ALU IS
        PORT (Natavio_May_22_2022_ALUCtrl    : IN std_logic_vector (3 DOWNTO 0);
              Natavio_May_22_2022_Operand1   : IN std_logic_vector (31 DOWNTO 0);
              Natavio_May_22_2022_Operand2   : IN std_logic_vector (31 DOWNTO 0);
              Natavio_May_22_2022_upper_res  : OUT std_logic_vector (31 DOWNTO 0);
              Natavio_May_22_2022_lower_res  : OUT std_logic_vector (31 DOWNTO 0);
              Natavio_May_22_2022_Overflow   : OUT std_logic;
              Natavio_May_22_2022_Zero       : OUT std_logic;
              Natavio_May_22_2022_Negative   : OUT std_logic;
              Natavio_May_22_2022_Cout       : OUT std_logic);
    END COMPONENT;

    -- CPU Controller
    COMPONENT Natavio_May_22_2022_CPU_Controller IS
        PORT (Natavio_May_22_2022_OPCode     : in std_logic_vector (5 DOWNTO 0);
              Natavio_May_22_2022_Funct      : in std_logic_vector (5 DOWNTO 0);
              Natavio_May_22_2022_ExtOp      : out std_logic;
              Natavio_May_22_2022_ALUCtrl    : out std_logic_vector (3 DOWNTO 0);
              Natavio_May_22_2022_RegWr      : out std_logic;
              Natavio_May_22_2022_RegDst     : out std_logic;
              Natavio_May_22_2022_ALUSrc     : out std_logic;
              Natavio_May_22_2022_MemToReg   : out std_logic;
              Natavio_May_22_2022_MemWr      : out std_logic;
              Natavio_May_22_2022_Branch     : out std_logic;
              Natavio_May_22_2022_Jump       : out std_logic);
    END COMPONENT;

END Natavio_May_22_2022_CPU_Components;
```

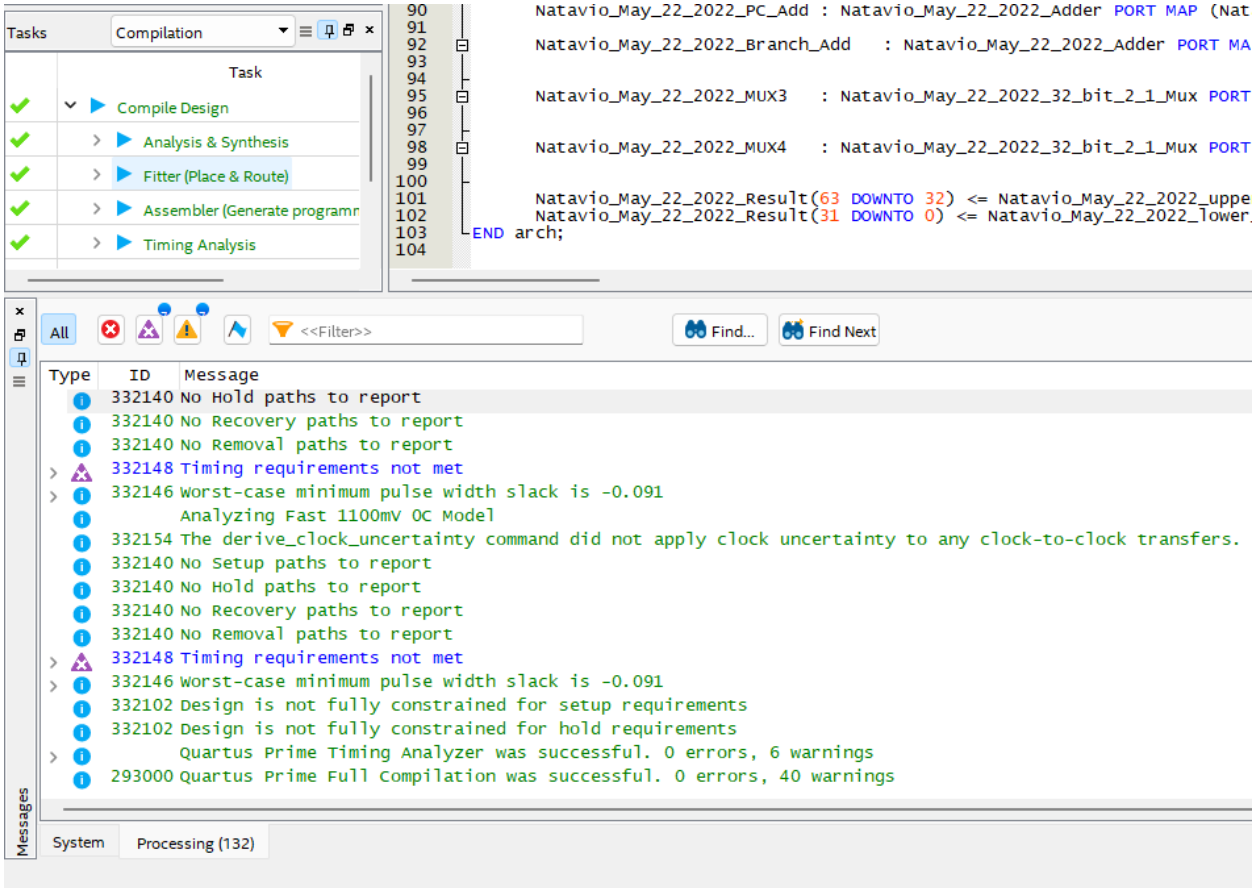
Central Processing Unit (CPU)

This code is a code design for the Single Cycle CPU. It begins by importing the CPU components package I previously made. The input is the clock signal and the result is a 64-bit string. Since this CPU is single cycled, that means that one instruction is carried out each time. The design of the CPU follows that of the diagram in the instructions.

Natavio_May_22_2022_CPU_Controller.vhd | Natavio_May_22_2022_CPU_Components.vhd | Natavio_May_22_2022_CPU.vhd | Compilation Report - Natavio_May_22_2022_Data_Memory

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE ieee.std_logic_unsigned.all;
USE WORK.Natavio_May_22_2022_CPU_Components.all;

ENTITY Natavio_May_22_2022_CPU IS
   PORT (Natavio_May_22_2022_clk    : in STD_LOGIC;
         Natavio_May_22_2022_Result : OUT std_logic_vector (63 DOWNTO 0));
END Natavio_May_22_2022_CPU;

ARCHITECTURE arch OF Natavio_May_22_2022_CPU IS
   SIGNAL Natavio_May_22_2022_OPCode, Natavio_May_22_2022_Funct   : std_logic_vector (5 DOWNTO 0);
   SIGNAL Natavio_May_22_2022_RS, Natavio_May_22_2022_RT, Natavio_May_22_2022_RD, Natavio_May_22_2022_Shamt : std_logic_vector (4 DOWNTO 0);
   SIGNAL Natavio_May_22_2022_Imm   : std_logic_vector (15 DOWNTO 0);
   SIGNAL Natavio_May_22_2022_JmpAddr, Natavio_May_22_2022_Inst, Natavio_May_22_2022_IR_Q : std_logic_vector (31 DOWNTO 0);

   -- CPU Controller Control Signals
   SIGNAL Natavio_May_22_2022_ExtOp, Natavio_May_22_2022_RegWr, Natavio_May_22_2022_RegDst, Natavio_May_22_2022_ALUSrc,
          Natavio_May_22_2022_MemWr, Natavio_May_22_2022_MemRd, Natavio_May_22_2022_MemToReg, Natavio_May_22_2022_Branch,
          Natavio_May_22_2022_Jump  : std_logic;
   SIGNAL Natavio_May_22_2022_PCSrc : std_logic;
   SIGNAL Natavio_May_22_2022_ALUCtrl  : std_logic_vector (3 DOWNTO 0);

   -- ALU Signals
   SIGNAL Natavio_May_22_2022_A, Natavio_May_22_2022_B, Natavio_May_22_2022_lower_res, Natavio_May_22_2022_upper_res,
          Natavio_May_22_2022_BusB  : std_logic_vector (31 DOWNTO 0);
   SIGNAL Natavio_May_22_2022_Overflow, Natavio_May_22_2022_Zero, Natavio_May_22_2022_Negative, Natavio_May_22_2022_Cout   : std_logic;

   -- MUX signals
   SIGNAL Natavio_May_22_2022_Rw : std_logic_vector (4 DOWNTO 0);
   SIGNAL Natavio_May_22_2022_BusW, Natavio_May_22_2022_NextAddr  : std_logic_vector (31 DOWNTO 0);

   -- Data Memory signals
   SIGNAL Natavio_May_22_2022_Mem_Data : std_logic_vector (31 DOWNTO 0);

   -- Instruction signals
   SIGNAL Natavio_May_22_2022_ImmExt, Natavio_May_22_2022_ImmExtShft, Natavio_May_22_2022_Adder1, Natavio_May_22_2022_Adder2,
          Natavio_May_22_2022_PC_O, Natavio_May_22_2022_CurrAdder, Natavio_May_22_2022_CurrAdderTmp : std_logic_vector (31 DOWNTO 0);

   BEGIN
      Natavio_May_22_2022_Program_Count   : Natavio_May_22_2022_PC PORT MAP (Natavio_May_22_2022_clk, Natavio_May_22_2022_CurrAdder,
                                                                            Natavio_May_22_2022_NextAddr);

      Natavio_May_22_2022_Inst_Mem  : Natavio_May_22_2022_Instruction_Memory PORT MAP (Natavio_May_22_2022_NextAddr, Natavio_May_22_2022_OPCode,
                                                                                       Natavio_May_22_2022_RS, Natavio_May_22_2022_RT,
                                                                                       Natavio_May_22_2022_RD, Natavio_May_22_2022_Shamt,
                                                                                       Natavio_May_22_2022_Funct, Natavio_May_22_2022_Imm,
                                                                                       Natavio_May_22_2022_JmpAddr, Natavio_May_22_2022_Inst);

      Natavio_May_22_2022_InstReg   : Natavio_May_22_2022_IR PORT MAP (Natavio_May_22_2022_clk, '1', '1', '1', Natavio_May_22_2022_Inst,
                                                                       Natavio_May_22_2022_IR_Q);

      Natavio_May_22_2022_CPUC   : Natavio_May_22_2022_CPU_Controller PORT MAP (Natavio_May_22_2022_OPCode, Natavio_May_22_2022_Funct,
                                                                                Natavio_May_22_2022_ExtOp, Natavio_May_22_2022_ALUCtrl,
                                                                                Natavio_May_22_2022_RegWr, Natavio_May_22_2022_RegDst,
                                                                                Natavio_May_22_2022_ALUSrc, Natavio_May_22_2022_MemToReg,
                                                                                Natavio_May_22_2022_MemWr, Natavio_May_22_2022_Branch,
                                                                                Natavio_May_22_2022_Jump);

      Natavio_May_22_2022_MUX0   : Natavio_May_22_2022_5_bits_2_1_Mux PORT MAP (Natavio_May_22_2022_RT, Natavio_May_22_2022_RD,
                                                                                Natavio_May_22_2022_RegDst, Natavio_May_22_2022_Rw);

      Natavio_May_22_2022_RegF   : Natavio_May_22_2022_RF PORT MAP (Natavio_May_22_2022_clk, Natavio_May_22_2022_RegWr, Natavio_May_22_2022_BusW,
                                                                    Natavio_May_22_2022_Rw, Natavio_May_22_2022_RS, Natavio_May_22_2022_RT,
                                                                    Natavio_May_22_2022_A, Natavio_May_22_2022_BusB);

      Natavio_May_22_2022_MUX1   : Natavio_May_22_2022_32_bit_2_1_Mux PORT MAP (Natavio_May_22_2022_BusB, Natavio_May_22_2022_ImmExt,
                                                                                Natavio_May_22_2022_ALUSrc, Natavio_May_22_2022_B);

      Natavio_May_22_2022_ALU_P  : Natavio_May_22_2022_ALU PORT MAP (Natavio_May_22_2022_ALUCtrl, Natavio_May_22_2022_A, Natavio_May_22_2022_B,
                                                                     Natavio_May_22_2022_upper_res, Natavio_May_22_2022_lower_res,
                                                                     Natavio_May_22_2022_Overflow, Natavio_May_22_2022_Zero,
                                                                     Natavio_May_22_2022_Negative, Natavio_May_22_2022_Cout);

      Natavio_May_22_2022_PCSrc <= Natavio_May_22_2022_Branch AND Natavio_May_22_2022_Zero;

      Natavio_May_22_2022_DM  : Natavio_May_22_2022_Data_Memory PORT MAP (Natavio_May_22_2022_clk, Natavio_May_22_2022_MemWr, Natavio_May_22_2022_MemRd,
                                                                          Natavio_May_22_2022_BusB, Natavio_May_22_2022_upper_res,
                                                                          Natavio_May_22_2022_Mem_Data);

      Natavio_May_22_2022_MUX2   : Natavio_May_22_2022_32_bit_2_1_Mux PORT MAP (Natavio_May_22_2022_lower_res, Natavio_May_22_2022_Mem_Data,
                                                                                Natavio_May_22_2022_MemToReg, Natavio_May_22_2022_BusW);

      Natavio_May_22_2022_SignExt   : Natavio_May_22_2022_Sign_Ext PORT MAP (Natavio_May_22_2022_Imm, Natavio_May_22_2022_ExtOp,
                                                                             Natavio_May_22_2022_ImmExt);

      Natavio_May_22_2022_ImmExtShft <= to_stdlogicvector(to_bitvector(Natavio_May_22_2022_ImmExt) sll 2);

      Natavio_May_22_2022_PC_Add : Natavio_May_22_2022_Adder PORT MAP (Natavio_May_22_2022_NextAddr, x"00000004", Natavio_May_22_2022_Adder1);

      Natavio_May_22_2022_Branch_Add   : Natavio_May_22_2022_Adder PORT MAP (Natavio_May_22_2022_Adder1, Natavio_May_22_2022_ImmExtShft,
                                                                             Natavio_May_22_2022_Adder2);

      Natavio_May_22_2022_MUX3   : Natavio_May_22_2022_32_bit_2_1_Mux PORT MAP (Natavio_May_22_2022_Adder1, Natavio_May_22_2022_Adder2,
                                                                                Natavio_May_22_2022_PCSrc, Natavio_May_22_2022_CurrAdderTmp);

      Natavio_May_22_2022_MUX4   : Natavio_May_22_2022_32_bit_2_1_Mux PORT MAP (Natavio_May_22_2022_CurrAdderTmp, Natavio_May_22_2022_JmpAddr,
                                                                                Natavio_May_22_2022_Jump, Natavio_May_22_2022_PC_O);

      Natavio_May_22_2022_Result(63 DOWNTO 32) <= Natavio_May_22_2022_upper_res;
      Natavio_May_22_2022_Result(31 DOWNTO 0) <= Natavio_May_22_2022_lower_res;
END arch;
```

Compiling


Tasks
Compilation
Task
Compile Design
Analysis & Synthesis
Fitter (Place & Route)
Assembler (Generate programm
Timing Analysis
90 Natavio_May_22_2022_PC_Add : Natavio_May_22_2022_Adder PORT MAP (Nat
91
92 Natavio_May_22_2022_Branch_Add : Natavio_May_22_2022_Adder PORT MA
93
94
95 Natavio_May_22_2022_MUX3 : Natavio_May_22_2022_32_bit_2_1_Mux PORT
96
97
98 Natavio_May_22_2022_MUX4 : Natavio_May_22_2022_32_bit_2_1_Mux PORT
99
100
101 Natavio_May_22_2022_Result(63 DOWNTO 32) <= Natavio_May_22_2022_uppe
102 Natavio_May_22_2022_Result(31 DOWNTO 0) <= Natavio_May_22_2022_lower
103 END arch;
104
All
<<Filter>>
Find...
Find Next
Type ID Message
332140 No Hold paths to report
332140 No Recovery paths to report
332140 No Removal paths to report
332148 Timing requirements not met
332146 Worst-case minimum pulse width slack is -0.091
Analyzing Fast 1100mV 0C Model
332154 The derive_clock_uncertainty command did not apply clock uncertainty to any clock-to-clock transfers.
332140 No Setup paths to report
332140 No Hold paths to report
332140 No Recovery paths to report
332140 No Removal paths to report
332148 Timing requirements not met
332146 Worst-case minimum pulse width slack is -0.091
332102 Design is not fully constrained for setup requirements
332102 Design is not fully constrained for hold requirements
Quartus Prime Timing Analyzer was successful. 0 errors, 6 warnings
293000 Quartus Prime Full Compilation was successful. 0 errors, 40 warnings
Messages
System
Processing (132)

Simulations

To test the CPU design, we are to execute the following expression in both platforms:

$$Z = \sum_{k=1}^{5} X_k$$

MIPS Code & Simulation

The following is my MIPS code to add 5 numbers from 1 to 5. So for this code I essentially added all numbers from 1 to 5 at a time using addi.

Edit | Execute

Testing.asm

```
.text
.globl main

main:
addi    $s0, $s0, 1     # add $s0 by 1
addi    $s0, $s0, 2     # add $s0 by 2
addi    $s0, $s0, 3     # add $s0 by 3
addi    $s0, $s0, 4     # add $s0 by 4
addi    $s0, $s0, 5     # add $s0 by 5
```

Edit | Execute

Text Segment

Bkpt	Address	Code	Basic	
	0x00400000	0x22100001	addi $16,$16,0x0000...	5: addi $s0, $s0, 1 # add $s0 by 1
	0x00400004	0x22100002	addi $16,$16,0x0000...	6: addi $s0, $s0, 2 # add $s0 by 2
	0x00400008	0x22100003	addi $16,$16,0x0000...	7: addi $s0, $s0, 3 # add $s0 by 3
	0x0040000c	0x22100004	addi $16,$16,0x0000...	8: addi $s0, $s0, 4 # add $s0 by 4
	0x00400010	0x22100005	addi $16,$16,0x0000...	9: addi $s0, $s0, 5 # add $s0 by 5

In these screenshots I show how the value of $s0 changes from 0x0 to 0x037 which is 55 in decimal. This code clearly works as all numbers from 1 to 10 added result in 55.

Registers | Coproc 1 | Coproc 0

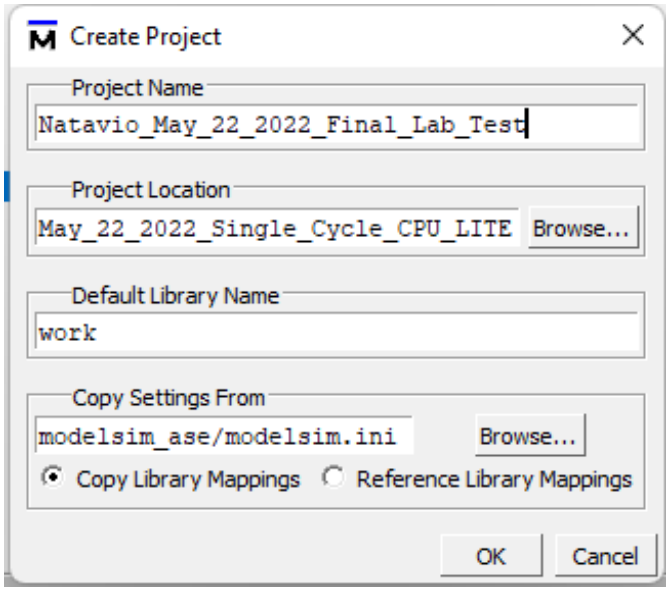
Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000

Registers | Coproc 1 | Coproc 0

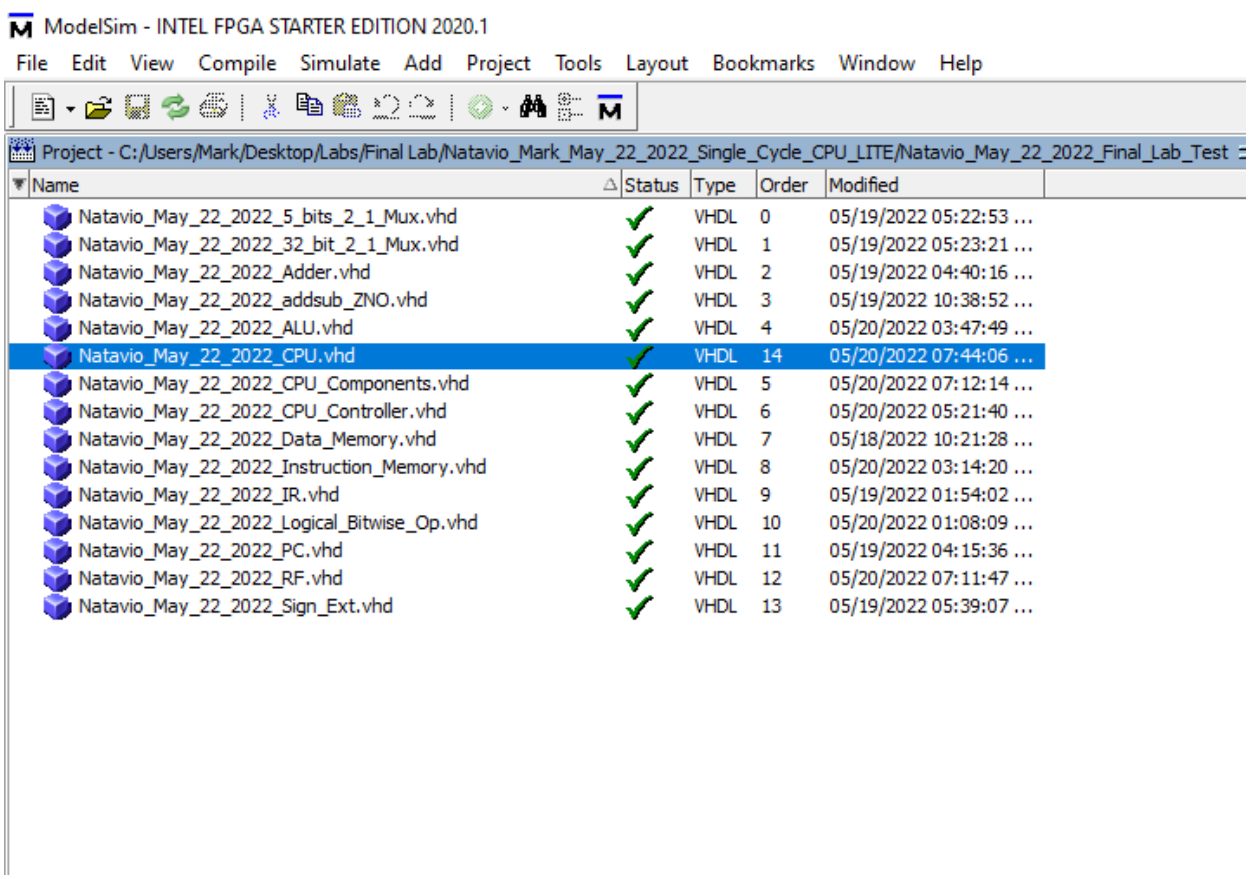
Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x0000000f
$s1	17	0x00000000

ModelSim Simulation

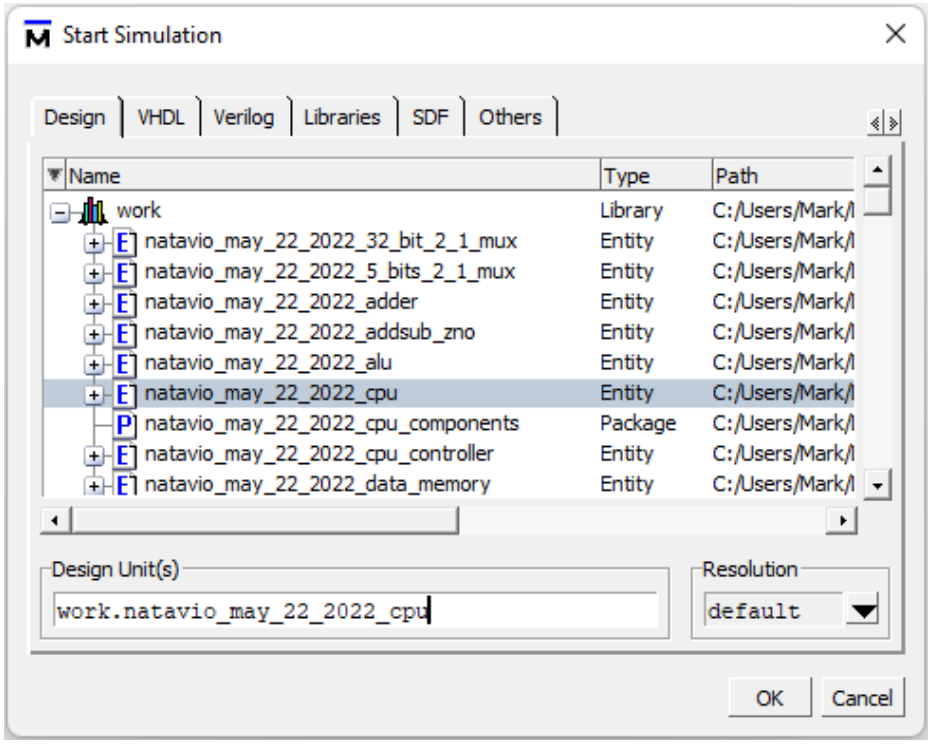
In order to simulate the CPU design I used ModelSim. I created a new project and titled it Natavio_May_22_2022_Final_Lab_Test.



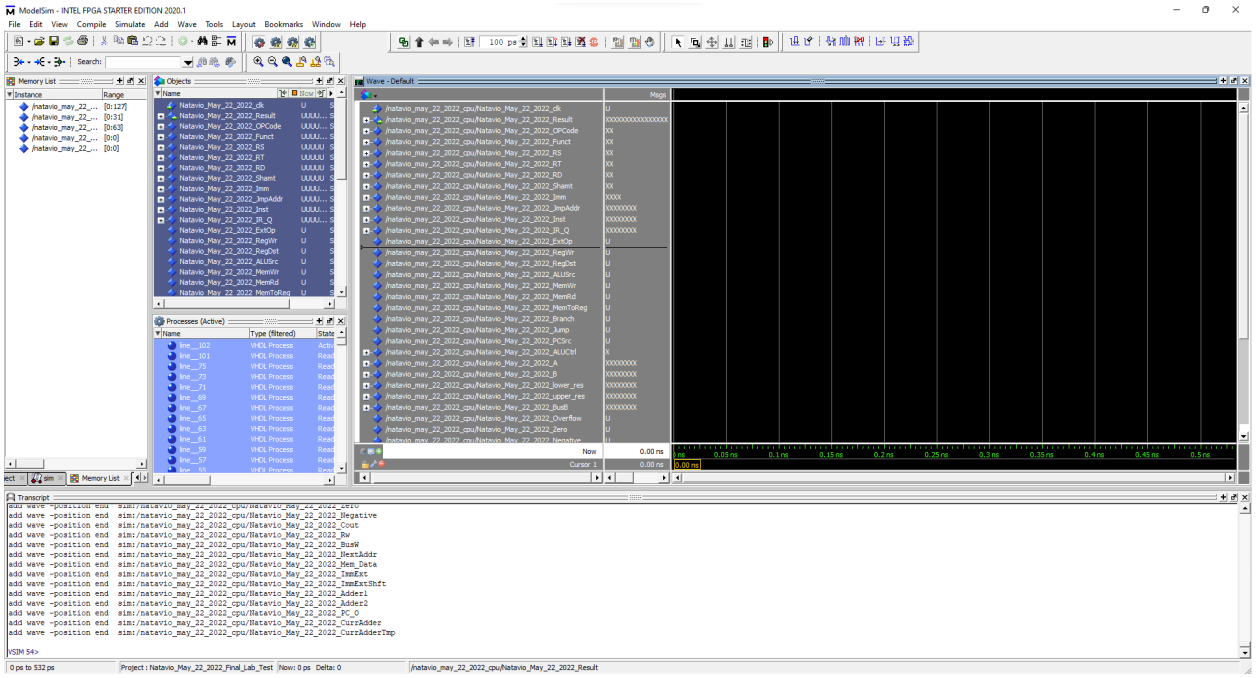
Then I added all vhd files from the project into the ModelSim Project. Then, I compiled it to make sure that all files are working in ModelSim.



I clicked on Simulate > Start Simulation and chose the Natavio_May_22_2022_CPU file, as shown in the screenshot below.



After pressing OK I added all CPU signal values to the waveform as seen in the screenshot below. Now I was ready to simulate the code based on the instructions in the Instruction Memory component.



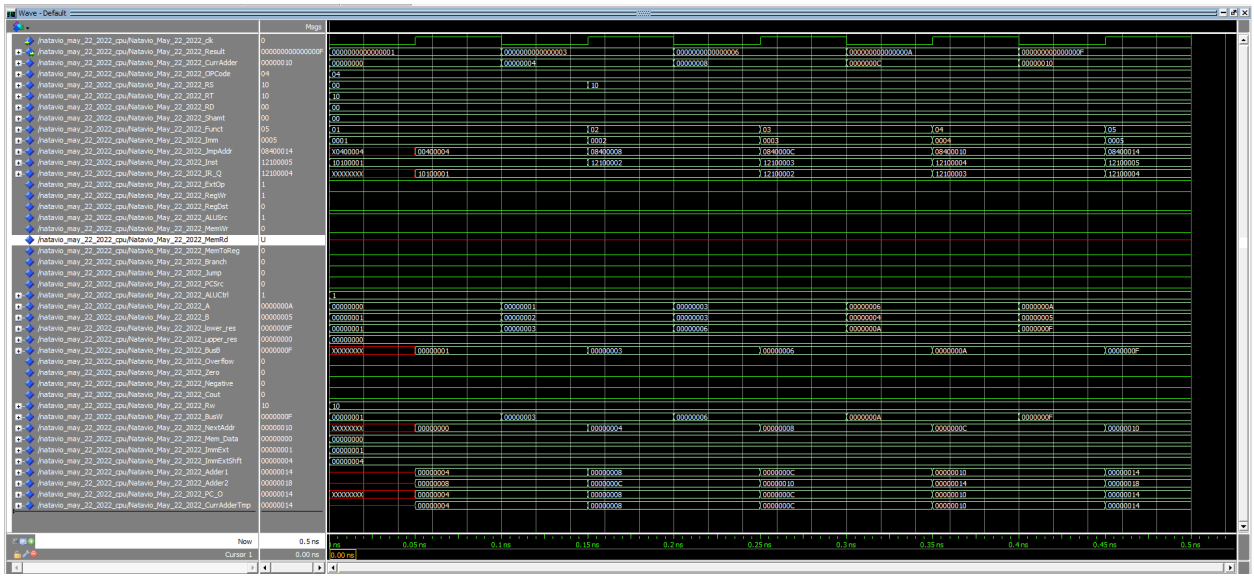
The desired instructions loaded into the Instruction Memory component can be seen in the screenshot below.

```
BEGIN
   Natavio_May_22_2022_mem_arr(0)  <= x"10100001"; -- addi $s0, 1
   Natavio_May_22_2022_mem_arr(4)  <= x"12100002"; -- addi $s0, 2
   Natavio_May_22_2022_mem_arr(8)  <= x"12100003"; -- addi $s0, 3
   Natavio_May_22_2022_mem_arr(12) <= x"12100004"; -- addi $s0, 4
   Natavio_May_22_2022_mem_arr(16) <= x"12100005"; -- addi $s0, 5
```

I know that these instructions are the same as my instructions in MIPS by breaking it down. Once again, looking at the I-type instruction format and the addi formula (R[rt] = R[rs] + SignExtImm), we can see that

- 10100001: 00010000000100000000000000000001
  - OPCode: 000100 -> addi which is an I-type instruction
  - RS: 00000 -> register address 0x0 which is loaded with 0x0 by default
  - RT: 10000 -> register address 0xF which is loaded with 0x0 by default
  - Imm: 0000000000000001 -> 0x1
- 12100002: 00010010000100000000000000000010
  - OPCode: 000100 -> addi which is an I-type instruction
  - RS: 10000 -> register address 0xF which is now loaded with 0x1
  - RT: 10000 -> register address 0xF which is now loaded with 0x1
  - Imm: 0000000000000010 -> 0x2
- 12100003: 00010010000100000000000000000011
  - OPCode: 000100 -> addi which is an I-type instruction
  - RS: 10000 -> register address 0xF which is now loaded with 0x3
  - RT: 10000 -> register address 0xF which is now loaded with 0x3
  - Imm: 0000000000000010 -> 0x3
- 12100004: 00010010000100000000000000000100
  - OPCode: 000100 -> addi which is an I-type instruction
  - RS: 10000 -> register address 0xF which is now loaded with 0x6
  - RT: 10000 -> register address 0xF which is now loaded with 0x6
  - Imm: 0000000000000010 -> 0x4
- 12100005: 00010010000100000000000000000101
  - OPCode: 000100 -> addi which is an I-type instruction
  - RS: 10000 -> register address 0xF which is now loaded with 0xA
  - RT: 10000 -> register address 0xF which is now loaded with 0xA
  - Imm: 0000000000000010 -> 0x5

After this instruction is executed, 0xA is added by the immediate 0x5, which results in 15 or 0xF in hexadecimal. Therefore giving us the correct result of adding all numbers from 1 to 5. This is reflected in the simulation waveform below. Since I am using addi instructions, there is no need to set any initial data into the memory. Instead the instruction will access the data in the RS memory and RT memory, which by default start with the value 0x00000000. Then, I loaded the first instruction to the Program Counter (PC). In the screenshot below, this is the CurrAdder signal. I began by loading it with instruction 0 as per the screenshot above. The CPU is designed to fetch the instruction taken from the instruction memory and load it into the Instruction Register (IR), as shown in the Inst signal.



The screenshot above shows the full simulation waveform. As we can see the clock changes from ‘0’ to ‘1’ once every 100 ps. As shown by the Result signal, we are constantly adding the numbers from 1 to 5 in accordance to my prior explanation. The Result signal is finalized with the value of 0x0F. This means that my CPU test was successful.

Conclusion

In this lab I learned how to create a single cycle CPU that is capable of carrying out R-type, I-type, and J-type instructions. More importantly, I learned how the CPU works more in detail and all components that are used to build one. What I found most interesting about this lab is the usage of previous labs to build this one. This not only meant that it was required for me to adapt my previous labs to this one, but also planning out how each component will work with one another. In the end, my test was successful, which shows that the CPU was properly made.