**Eastern Mediterranean University**

**Computer Engineering Department**

**CMSE 222 Introduction to Computer Organization– Lab. 8**

**Single Clock Data Path for 16-bit R-type Instructions in ALTERA QUARTUS VHDL Environment**

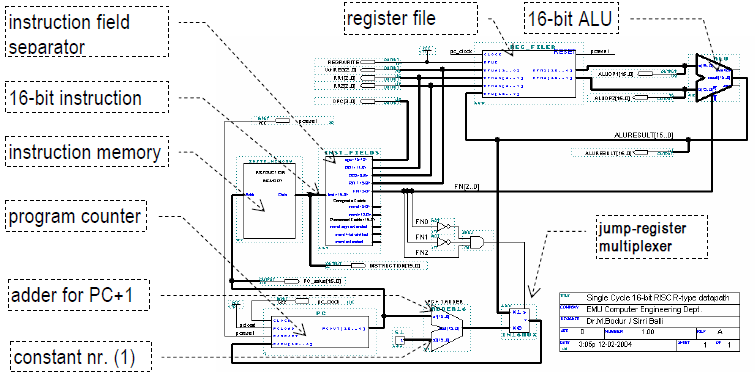
1. **Objective:**

To familiarize with the Single-Clock VHDL implementation for a set of 16-bit R-type instructions, and to measure the response time of several building components in a RISC datapath.

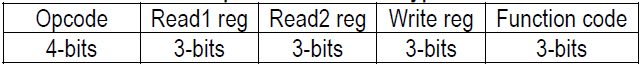
1. **Introduction**

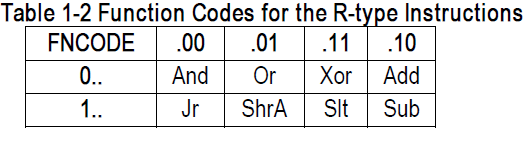
You have already worked to construct a simple 16-bit instruction set for your project/homework. A typical simple 16-bit instruction set and a corresponding R-type datapath is provided in this experiment. You will be asked to take several measurements that can lead you to a conclusion about the typical speed constraints, and possible improvements of the similar circuits.

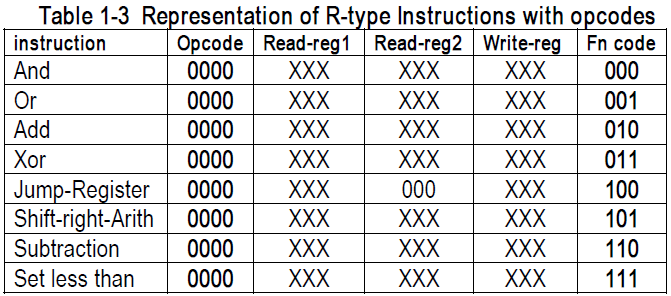
In this experiment, we will focus only on the R-type instructions, so that we can observe typical properties of some of the basic building blocks such as the Program-Counter, the Instruction-Memory, the Register-File, and the Adder for updating the Program-Counter.



The complete set of the 16-bit instructions are summarized in the following tables.







1. **QUARTUS and VHDL Code**

In this section you should learn how to implement a VHDL code for single cycle data path. Before we go any further, let’s define some of the terms that we use throughout the book.

**Entity:** All designs are expressed in terms of entities. An entity is the most basic building block in a design. The uppermost level of the design is the top-level entity. If the design is hierarchical, then the top-level description will have lower-level descriptions contained in it. These lower-level descriptions will be lower-level entities contained in the top-level entity description.

**Architecture:** All entities that can be simulated have an architecture description. The architecture describes the behavior of the entity. A single entity can have multiple architectures. One architecture might be behavioral while another might be a structural description of the design.

**Configuration:** A configuration statement is used to bind a component instance to an entity-architecture pair. A configuration can be considered like a parts list for a design. It describes which behavior to use for each entity, much like a parts list describes which part to use for each part in the design.

**Package:** A package is a collection of commonly used data types and subprograms used in a design. Think of a package as a toolbox that contains tools used to build designs.

**Driver:** This is a source on a signal. If a signal is driven by two sources, then when both sources are active, the signal will have two drivers.

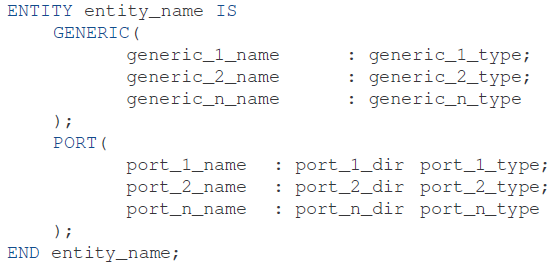
**Bus:** The term “bus” usually brings to mind a group of signals or a particular method of communication used in the design of hardware. In VHDL, a bus is a special kind of signal that may have its drivers turned off.

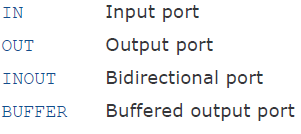
**Attribute:** An attribute is data that are attached to VHDL objects or predefined data about VHDL objects. Examples are the current drive capability of a buffer or the maximum operating temperature of the device.

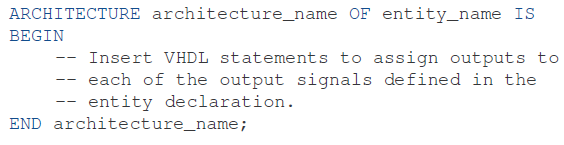
**Generic:** A generic is VHDL’s term for a parameter that passes information to an entity. For instance, if an entity is a gate level model with a rise and a fall delay, values for the rise and fall delays could be passed into the entity with generics.

**Process:** A process is the basic unit of execution in VHDL. All operations that are performed in a simulation of a

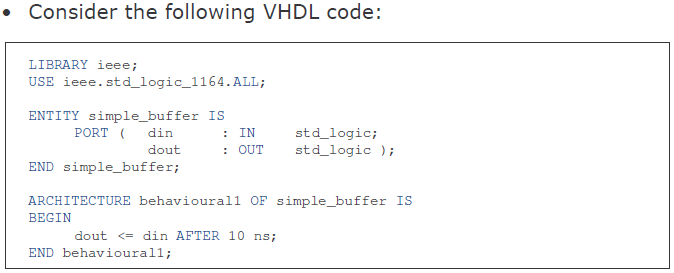
Thus, let us get our feet wet and look at VHDL structure briefly as follow:







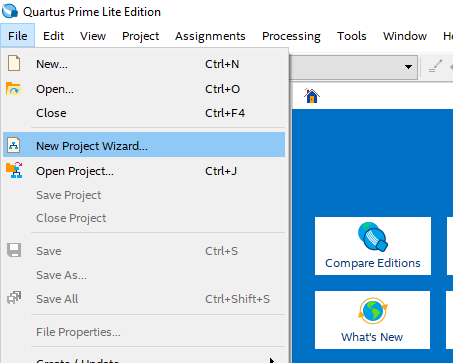
This is a simple example of VHDL code that just work as buffer and send input to output after 10 ns:



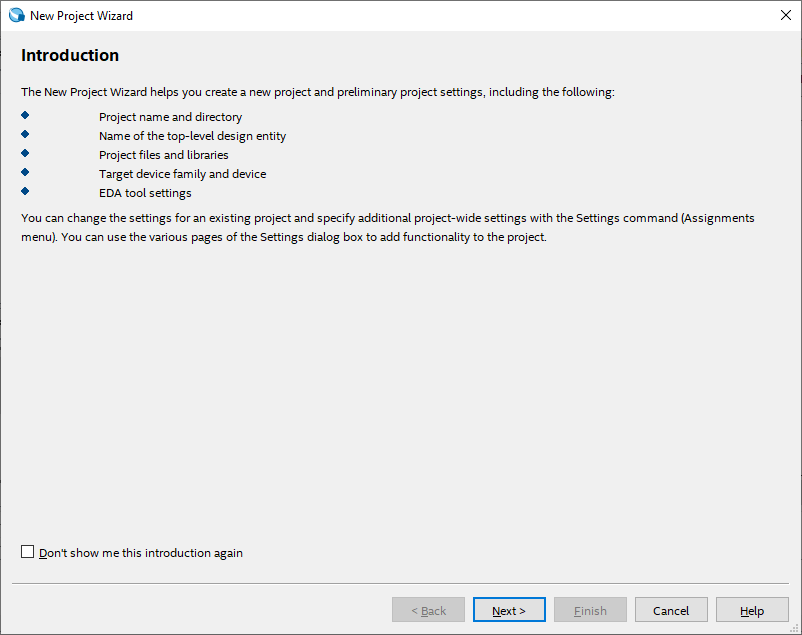
**First simple program using VHDL:**

Step 1. Create a Project: Every new design you make in the Quartus program must be under a project name. After opening the program, to create a project, click on:

[File] -> [New Project Wizard]



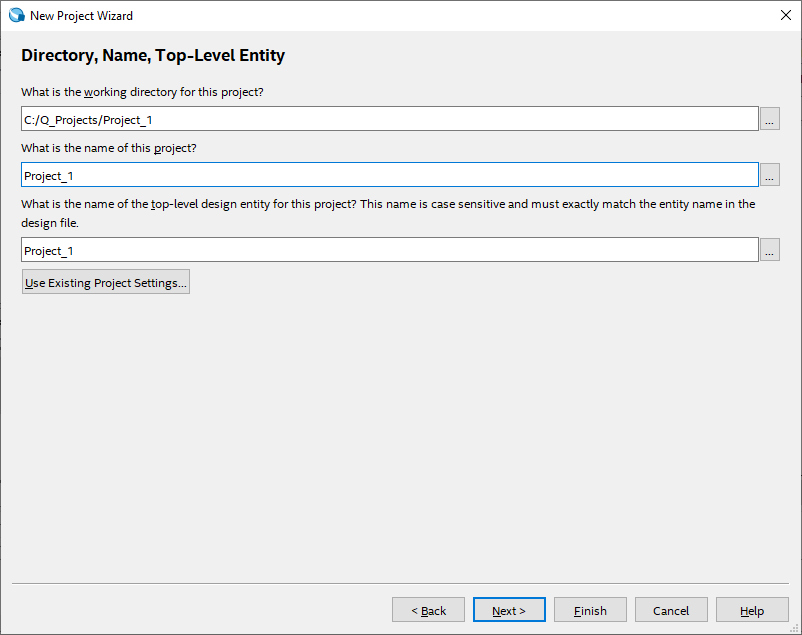
After this stage you will see the following project builder screen.



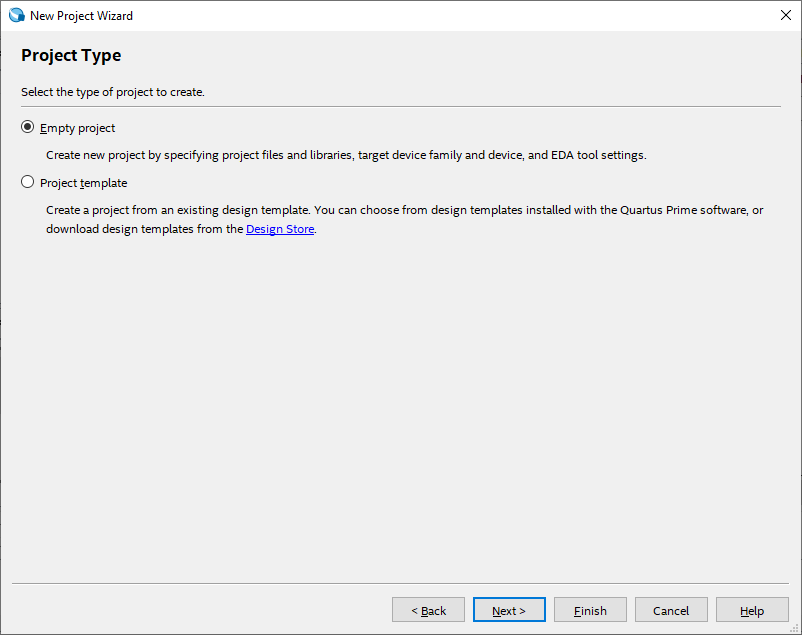
This screen is the home screen of the project builder. Here you will see which stages of your project are formed and what you can organize.

Here, press [Next >] button to continue.

On the new screen, you will see the input sections that ask us to enter the directory and the name of the related project. Please fill in the appropriate information here. (Project names for the Quartus program is very important.)



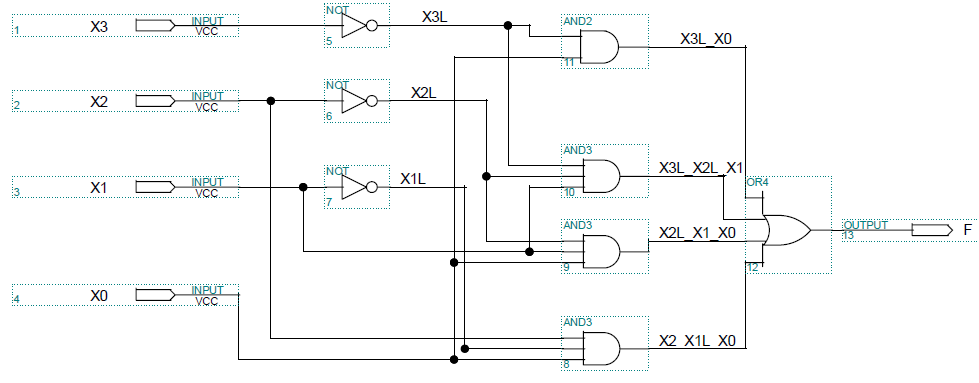
The project type display gives you additional setting options for what hardware the project will be compiled and implemented. Since, we'll simulate our projects on the program and we won't run on any real hardware, we'll choose the [Empty Project] option here, and then click the [Finish] button to create our project.



Once your project is created, you will see a screen like this:



**Step2:** Look at the program using VHDL code, here we have a practical circuit, implement it in Quartus (destine / compile/ simulate/show waveform), you can review pervious lab experiment and follow the steps.



**Step3:** In order to create a VHDL file follow the steps:

**File🡪New🡪VHDL File**

This is the VHDL code of above program:

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

ENTITY lab6 IS

PORT (x0, x1, x2,X3 : IN STD\_LOGIC;

F : OUT STD\_LOGIC);

END lab6;

ARCHITECTURE Arch\_NelistStruct of lab6 is

SIGNAL X3L, X2L, X1L, X3L\_X0, X3L\_X2L\_X1, X2L\_X1\_X0, X2\_X1L\_X0:STD\_LOGIC;

BEGIN

X3L <= not X3;

X2L <= not X2;

X1L <= not X1;

X3L\_X0 <= X3L and X0;

X3L\_X2L\_X1 <= X3L and X2L and X1;

X2L\_X1\_X0 <= X2L and X1 and X0;

X2\_X1L\_X0 <= X2 and X1 and X0;

F <= x3L\_X0 or X3L\_X2L\_X1 or X2L\_X1\_X0 or x2\_X1L\_X0;

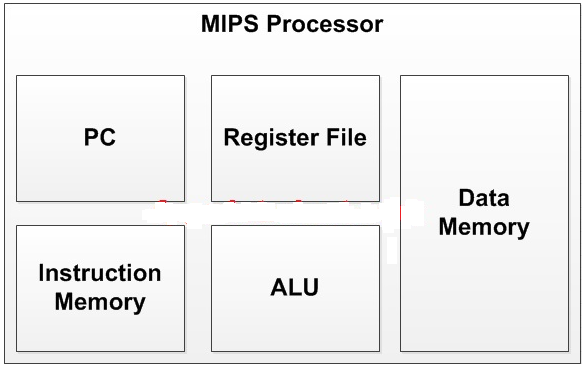
END Arch\_NelistStruct;

**Note:** Trace the program line by line, now you are ready to learn single cycle data path using VHDL code.

1. **Experimental Practice**

Note that our instructions have no mnemonics at this stage, because an assembler has not been written yet for this brand new processor. We use the binary codes of the instructions to write them into the instruction memory.

The VHDL code for the MIPS Processor will be presented. A simple VHDL testbench for the MIPS processor will be also provided for simulation purposes.



**Note:** Be careful in your project name and VHDL file name and entity name must be same.

**VHDL code for Data Memory of the MIPS processor:**

-- fpga4student.com: [FPGA projects](https://www.fpga4student.com/p/fpga-projects.html), [Verilog projects](https://www.fpga4student.com/p/verilog-project.html), [VHDL projects](https://www.fpga4student.com/p/vhdl-project.html)

-- VHDL project: VHDL code for single-cycle MIPS Processor

**library** **IEEE**;

**use** **IEEE.STD\_LOGIC\_1164.ALL**;

**USE** **IEEE.numeric\_std.all**;

-- VHDL code for the data Memory of the MIPS Processor

**entity** **Data\_Memory\_VHDL** **is**

**port** (

clk: **in** **std\_logic**;

mem\_access\_addr: **in** **std\_logic\_Vector**(**15** **downto** **0**);

mem\_write\_data: **in** **std\_logic\_Vector**(**15** **downto** **0**);

mem\_write\_en,mem\_read:**in** **std\_logic**;

mem\_read\_data: **out** **std\_logic\_Vector**(**15** **downto** **0**)

);

**end** **Data\_Memory\_VHDL**;

**architecture** **Behavioral** **of** **Data\_Memory\_VHDL** **is**

**signal** i: **integer**;

**signal** ram\_addr: **std\_logic\_vector**(**7** **downto** **0**);

**type** data\_mem **is** **array** (**0** **to** **255** ) **of** **std\_logic\_vector** (**15** **downto** **0**);

**signal** RAM: data\_mem :=((**others**=> (**others**=>'0')));

**begin**

ram\_addr <= mem\_access\_addr(**8** **downto** **1**);

**process**(clk)

**begin**

**if**(rising\_edge(clk)) **then**

**if** (mem\_write\_en='1') **then**

ram(to\_integer(unsigned(ram\_addr))) <= mem\_write\_data;

**end** **if**;

**end** **if**;

**end** **process**;

mem\_read\_data <= ram(to\_integer(unsigned(ram\_addr))) **when** (mem\_read='1') **else** x"0000";

**end** **Behavioral**;

**VHDL code for**[**ALU**](https://www.fpga4student.com/2017/06/vhdl-code-for-arithmetic-logic-unit-alu.html)**of the MIPS processor:**

-- fpga4student.com: FPGA projects, Verilog projects, VHDL projects

-- VHDL project: VHDL code for single-cycle MIPS Processor

**library** **IEEE**;

**use** **IEEE.STD\_LOGIC\_1164.ALL**;

**use** **IEEE.STD\_LOGIC\_signed.all**;

-- VHDL code for ALU of the MIPS Processor

**entity** **ALU\_VHDL** **is**

**port**(

a,b : **in** **std\_logic\_vector**(**15** **downto** **0**); -- src1, src2

alu\_control : **in** **std\_logic\_vector**(**2** **downto** **0**); -- function select

alu\_result: **out** **std\_logic\_vector**(**15** **downto** **0**); -- ALU Output Result

zero: **out** **std\_logic** -- Zero Flag

);

**end** **ALU\_VHDL**;

**architecture** **Behavioral** **of** **ALU\_VHDL** **is**

**signal** result: **std\_logic\_vector**(**15** **downto** **0**);

**begin**

**process**(alu\_control,a,b)

**begin**

**case** alu\_control **is**

**when** "000" =>

result <= a + b; -- add

**when** "001" =>

result <= a - b; -- sub

**when** "010" =>

result <= a **and** b; -- and

**when** "011" =>

result <= a **or** b; -- or

**when** "100" =>

**if** (a<b) **then**

result <= x"0001";

**else**

result <= x"0000";

**end** **if**;

**when** **others** => result <= a + b; -- add

**end** **case**;

**end** **process**;

zero <= '1' **when** result=x"0000" **else** '0';

alu\_result <= result;

**end** **Behavioral**;

**VHDL code for**[**ALU Control Unit**](https://www.fpga4student.com/2017/01/basic-digital-blocks-in-verilog.html)**of the MIPS processor:**

-- fpga4student.com: FPGA projects, Verilog projects, VHDL projects

-- VHDL project: VHDL code for single-cycle MIPS Processor

**library** **IEEE**;

**use** **IEEE.STD\_LOGIC\_1164.ALL**;

-- VHDL code for ALU Control Unit of the MIPS Processor

**entity** **ALU\_Control\_VHDL** **is**

**port**(

ALU\_Control: **out** **std\_logic\_vector**(**2** **downto** **0**);

ALUOp : **in** **std\_logic\_vector**(**1** **downto** **0**);

ALU\_Funct : **in** **std\_logic\_vector**(**2** **downto** **0**)

);

**end** **ALU\_Control\_VHDL**;

**architecture** **Behavioral** **of** **ALU\_Control\_VHDL** **is**

**begin**

**process**(ALUOp,ALU\_Funct)

**begin**

**case** ALUOp **is**

**when** "00" =>

ALU\_Control <= ALU\_Funct(**2** **downto** **0**);

**when** "01" =>

ALU\_Control <= "001";

**when** "10" =>

ALU\_Control <= "100";

**when** "11" =>

ALU\_Control <= "000";

**when** **others** => ALU\_Control <= "000";

**end** **case**;

**end** **process**;

**end** **Behavioral**;

**VHDL code for Register File of the MIPS processor:**

-- fpga4student.com: FPGA projects, Verilog projects, VHDL projects

-- VHDL project: VHDL code for single-cycle MIPS Processor

**library** **IEEE**;

**use** **IEEE.STD\_LOGIC\_1164.ALL**;

**USE** **IEEE.numeric\_std.all**;

-- VHDL code for the register file of the MIPS Processor

**entity** **register\_file\_VHDL** **is**

**port** (

clk,rst: **in** **std\_logic**;

reg\_write\_en: **in** **std\_logic**;

reg\_write\_dest: **in** **std\_logic\_vector**(**2** **downto** **0**);

reg\_write\_data: **in** **std\_logic\_vector**(**15** **downto** **0**);

reg\_read\_addr\_1: **in** **std\_logic\_vector**(**2** **downto** **0**);

reg\_read\_data\_1: **out** **std\_logic\_vector**(**15** **downto** **0**);

reg\_read\_addr\_2: **in** **std\_logic\_vector**(**2** **downto** **0**);

reg\_read\_data\_2: **out** **std\_logic\_vector**(**15** **downto** **0**)

);

**end** **register\_file\_VHDL**;

**architecture** **Behavioral** **of** **register\_file\_VHDL** **is**

**type** reg\_type **is** **array** (**0** **to** **7** ) **of** **std\_logic\_vector** (**15** **downto** **0**);

**signal** reg\_array: reg\_type;

**begin**

**process**(clk,rst)

**begin**

**if**(rst='1') **then**

reg\_array(**0**) <= x"0001";

reg\_array(**1**) <= x"0002";

reg\_array(**2**) <= x"0003";

reg\_array(**3**) <= x"0004";

reg\_array(**4**) <= x"0005";

reg\_array(**5**) <= x"0006";

reg\_array(**6**) <= x"0007";

reg\_array(**7**) <= x"0008";

**elsif**(rising\_edge(clk)) **then**

**if**(reg\_write\_en='1') **then**

reg\_array(to\_integer(unsigned(reg\_write\_dest))) <= reg\_write\_data;

**end** **if**;

**end** **if**;

**end** **process**;

reg\_read\_data\_1 <= x"0000" when reg\_read\_addr\_1 = "000" **else** reg\_array(to\_integer(unsigned(reg\_read\_addr\_1)));

reg\_read\_data\_2 <= x"0000" when reg\_read\_addr\_2 = "000" **else** reg\_array(to\_integer(unsigned(reg\_read\_addr\_2)));

**end** **Behavioral**;

**VHDL code for Control Unit of the MIPS processor:**

- fpga4student.com: FPGA projects, Verilog projects, VHDL projects

-- VHDL project: VHDL code for single-cycle MIPS Processor

**library** **IEEE**;

**use** **IEEE.STD\_LOGIC\_1164.ALL**;

-- VHDL code for Control Unit of the MIPS Processor

**entity** **control\_unit\_VHDL** **is**

**port** (

opcode: **in** **std\_logic\_vector**(**2** **downto** **0**);

reset: **in** **std\_logic**;

reg\_dst,mem\_to\_reg,alu\_op: **out** **std\_logic\_vector**(**1** **downto** **0**);

jump,branch,mem\_read,mem\_write,alu\_src,reg\_write,sign\_or\_zero: **out** **std\_logic**

);

**end** **control\_unit\_VHDL**;

**architecture** **Behavioral** **of** **control\_unit\_VHDL** **is**

**begin**

**process**(reset,opcode)

**begin**

**if**(reset = '1') **then**

reg\_dst <= "00";

mem\_to\_reg <= "00";

alu\_op <= "00";

jump <= '0';

branch <= '0';

mem\_read <= '0';

mem\_write <= '0';

alu\_src <= '0';

reg\_write <= '0';

sign\_or\_zero <= '1';

**else**

**case** opcode **is**

**when** "000" => -- add

reg\_dst <= "01";

mem\_to\_reg <= "00";

alu\_op <= "00";

jump <= '0';

branch <= '0';

mem\_read <= '0';

mem\_write <= '0';

alu\_src <= '0';

reg\_write <= '1';

sign\_or\_zero <= '1';

**when** "001" => -- sliu

reg\_dst <= "00";

mem\_to\_reg <= "00";

alu\_op <= "10";

jump <= '0';

branch <= '0';

mem\_read <= '0';

mem\_write <= '0';

alu\_src <= '1';

reg\_write <= '1';

sign\_or\_zero <= '0';

**when** "010" => -- j

reg\_dst <= "00";

mem\_to\_reg <= "00";

alu\_op <= "00";

jump <= '1';

branch <= '0';

mem\_read <= '0';

mem\_write <= '0';

alu\_src <= '0';

reg\_write <= '0';

sign\_or\_zero <= '1';

**when** "011" =>-- jal

reg\_dst <= "10";

mem\_to\_reg <= "10";

alu\_op <= "00";

jump <= '1';

branch <= '0';

mem\_read <= '0';

mem\_write <= '0';

alu\_src <= '0';

reg\_write <= '1';

sign\_or\_zero <= '1';

**when** "100" =>-- lw

reg\_dst <= "00";

mem\_to\_reg <= "01";

alu\_op <= "11";

jump <= '0';

branch <= '0';

mem\_read <= '1';

mem\_write <= '0';

alu\_src <= '1';

reg\_write <= '1';

sign\_or\_zero <= '1';

**when** "101" => -- sw

reg\_dst <= "00";

mem\_to\_reg <= "00";

alu\_op <= "11";

jump <= '0';

branch <= '0';

mem\_read <= '0';

mem\_write <= '1';

alu\_src <= '1';

reg\_write <= '0';

sign\_or\_zero <= '1';

**when** "110" => -- beq

reg\_dst <= "00";

mem\_to\_reg <= "00";

alu\_op <= "01";

jump <= '0';

branch <= '1';

mem\_read <= '0';

mem\_write <= '0';

alu\_src <= '0';

reg\_write <= '0';

sign\_or\_zero <= '1';

**when** "111" =>-- addi

reg\_dst <= "00";

mem\_to\_reg <= "00";

alu\_op <= "11";

jump <= '0';

branch <= '0';

mem\_read <= '0';

mem\_write <= '0';

alu\_src <= '1';

reg\_write <= '1';

sign\_or\_zero <= '1';

**when** **others** =>

reg\_dst <= "01";

mem\_to\_reg <= "00";

alu\_op <= "00";

jump <= '0';

branch <= '0';

mem\_read <= '0';

mem\_write <= '0';

alu\_src <= '0';

reg\_write <= '1';

sign\_or\_zero <= '1';

**end** **case**;

**end** **if**;

**end** **process**;

**end** **Behavioral**;

**VHDL code for Instruction Memory of the MIPS processor:**

-- fpga4student.com: FPGA projects, Verilog projects, VHDL projects

-- VHDL project: VHDL code for single-cycle MIPS Processor

**library** **IEEE**;

**use** **IEEE.STD\_LOGIC\_1164.ALL**;

**USE** **IEEE.numeric\_std.all**;

-- VHDL code for the Instruction Memory of the MIPS Processor

**entity** **Instruction\_Memory\_VHDL** **is**

**port** (

pc: **in** **std\_logic\_vector**(**15** **downto** **0**);

instruction: **out** **std\_logic\_vector**(**15** **downto** **0**)

);

**end** **Instruction\_Memory\_VHDL**;

**architecture** **Behavioral** **of** **Instruction\_Memory\_VHDL** **is**

**signal** rom\_addr: **std\_logic\_vector**(**3** **downto** **0**);

-- lw $3, 0($0) -- pc=0

-- Loop: sltiu $1, $3, 11= pc = 2

-- beq $1, $0, Skip = 4 --PCnext=PC\_current+2+BranchAddr

-- add $4, $4, $3 = 6

-- addi $3, $3, 1 = 8

-- beq $0, $0, Loop--a

-- Skip c = 12 = 4 + 2 + br

**type** ROM\_type **is** **array** (**0** **to** **15** ) **of** **std\_logic\_vector**(**15** **downto** **0**);

**constant** rom\_data: ROM\_type:=(

"1000000110000000",

"0010110010001011",

"1100010000000011",

"0001000111000000",

"1110110110000001",

"1100000001111011",

"0000000000000000",

"0000000000000000",

"0000000000000000",

"0000000000000000",

"0000000000000000",

"0000000000000000",

"0000000000000000",

"0000000000000000",

"0000000000000000",

"0000000000000000"

);

**begin**

rom\_addr <= pc(**4** **downto** **1**);

instruction <= rom\_data(to\_integer(unsigned(rom\_addr))) **when** pc < x"0020" else x"0000";

**end** **Behavioral**;

1. **Reporting**

Before the Lab-time is over, show the simulation result to your lab assistant, in order to grading.

Grading:

Lab Performance:

Asst. Observations: