The RTL description allows us to capture the functional and structural aspects of a digital system before the actual implementation at the gateway level.

RTL descriptions are used at various stages of hardware development, such as simulation, synthesis, and verification, to ensure that the intended functionality of a digital system is correctly represented and implemented in the hardware.

Tasks completed in our department (Department of):

* Describe the processes and data transfer within the digital system
* Describe the operations carried out by records and the transfer of data between them.
* Determine how registers capture data on high or low clock edges and how signals propagate between clock domains.
* Determine how to perform arithmetic and logical operations on data
* Describe the behavior of finite state machines

In the RTL section, we wrote the VHDL code for (Single-Cycle Datapath and Control) by dividing the work into several units (module).

**1] Single Cycle CPU module (main module):**

**1. Module Declaration:**

This is the top-level module for the Single-Cycle CPU. It takes several inputs and provides one output. The inputs include instruction (`Inst`), reset signal (`Reset`), run signal (`Run`), clock signal (`Clock`), and a done signal (`Done`). This module represents the entire CPU.

single-cycle processor architecture where each instruction is executed in one clock cycle. The control unit generates signals that guide the operation of different functional units in the CPU.

**3. Internal Wires:**

- `address[31:0]`: Instruction memory address.

- `instruction[31:0]`: Current instruction.

- Various wires for opcode, register addresses, immediate values, etc.

- `Read\_Data1[31:0], Read\_Data2[31:0]`: Data read from registers.

- `ALU\_out[31:0]`: ALU operation result.

- `ALU\_zeroFlag`: Flag indicating zero result from the ALU.

- `imm\_after\_extend[31:0]`: Sign-extended immediate value.

- `out\_DataMemMux[31:0]`: Output from the Data Memory multiplexer.

- `Signal[9:0]`: Control signals.

- `operation[2:0]`: ALU operation signal.

- `PCSrc`: Program counter source signal.

- `Pc\_address[31:0]`: Program counter address.

**4. Functional Blocks:**

- Program Counter (`Program\_Counter`): Generates the next program counter address.

- Register (`PC`): Holds the program counter value.

- Instruction Memory (`Inst\_Mem`): Reads instructions from memory based on the program counter address.

- Control Unit (`ControlUnit`): Generates control signals based on the opcode.

- Multiplexer (`RegDst`): Selects the destination register based on the control signal.

- Register File (`reg\_file`): Reads from and writes to the register file.

- Multiplexer (`AluSrc`): Selects the second ALU operand based on the control signal.

- ALU Control Unit (`Alu\_Control`): Generates ALU control signals.

- ALU (`alu\_operation`): Performs ALU operations.

- Sign Extender (`SignExtend`): Sign-extends immediate values.

- Data Memory (`data\_Memory`): Reads from and writes to data memory.

- Multiplexer (`DataMemMux`): Selects data for write-back based on the control signal.

**5. Operations:**

- The CPU executes instructions in a single cycle.

- Control signals are generated by the control unit (`ControlUnit`) based on the opcode.

- The program counter is updated using the `Program\_Counter` module.

- Instructions are fetched from memory (`Inst\_Mem`).

- The register file (`reg\_file`) is read from and written to.

- ALU operations are performed (`alu\_operation`).

- Data memory operations are executed (`data\_Memory`).

- Write-back is determined by the `DataMemMux` multiplexer.

**6. Control Signals:**

- Control signals are generated by the `ControlUnit` module and influence various components in the processor.

**2] Register module (regn module):**

Registers are digital storage elements that store data.

32-bit register with asynchronous reset and enable control. It updates its content based on the rising edge of the clock signal. When the reset is active, the register is cleared to all zeros, and when the enable signal is active, it loads data from the input signal `regn\_Data`.

**3] Instruction memory module:**

This module is responsible for storing and retrieving instructions based on the specified address.

instruction memory module with the ability to store and retrieve instructions based on a provided memory address. When a reset is active, the memory is initialized with default values. During normal operation, it reads instructions from memory based on the provided address and outputs the fetched instruction. This is a basic building block in a digital system, commonly used in microprocessor designs to store and fetch program instructions.

**4] Multiplexer module:**

A multiplexer is a digital circuit that selects one of its input signals based on a control input.

defines a 32-bit multiplexer that selects one of the two 32-bit input signals (`mux\_inp0` or `mux\_inp1`) based on the control signal `Sel`. If `Sel` is 0, the output `Mux\_Out` carries the value of `mux\_inp0`, and if `Sel` is 1, the output carries the value of `mux\_inp1`. This is a basic building block used in digital circuits to switch between multiple data inputs.

**5] Register file module:**

register file module with 32 registers, capable of reading data from two specified registers and writing data to a specified register. It operates on rising clock edges and can be reset to initialize all registers to zero. This is a core component in microprocessor design, where registers store data for processing.

**6] Arithmetic Logic Unit (ALU)** **module:**

The ALU performs various arithmetic and logic operations on two operands based on the specified operation code (`aluOp`).

ALU module capable of performing various arithmetic and logic operations on two 32-bit operands. It determines the operation based on the 3-bit operation code (`aluOp`) and calculates the result, along with a zero flag indicating whether the result is zero.

**7] Sign extension** **module:**

performs sign extension from a 16-bit input to a 32-bit output.

to extend the sign bit of a smaller value to create a larger value without changing the value's sign.

by replicating the sign bit to create the additional 16 bits while preserving the original value's sign.

performs sign extension from a 16-bit input to a 32-bit output by replicating the sign bit to create the additional 16 bits while preserving the original value's sign.

**8] Data Memory module**

This module is responsible for reading and writing data to a memory array based on the specified address.

module with 1024 entries, each 32 bits wide, capable of reading and writing data based on the specified address. It operates on the rising edge of the clock signal and can be controlled with the `read\_En` and `write\_En` signals.

**9] Control unit** **module**

This module interprets opcode signals and generates control signals to coordinate the various components of a processor during the execution of different instruction formats, such as R-format, load/store, branch, and J-format instructions.

Bit assignments:

Bit 0 (RegDst): Set for R-format instructions.

Bit 1 (ALUSrc): Set for load and store instructions.

Bit 2 (MemtoReg): Set for load instructions.

Bit 3 (RegWrite): Set for R-format and load instructions.

Bit 4 (MemRead): Set for load instructions.

Bit 5 (MemWrite): Set for store instructions.

Bit 6 (Branch): Set for branch instructions.

Bit 7 (ALUOp1): Set for R-format instructions.

Bit 8 (ALUOp0): Set for branch instructions.

Bit 9 (Jump): Set for J-format instructions.

**10] ALU Control module**

Alu\_Control model is responsible for generating control signals (`Operation`) for an Arithmetic Logic Unit (ALU) based on input signals (`alu\_fun`, `aluop0`, and `aluop1`) to generate control signals (`Operation`) that dictate the specific operation to be performed by the ALU. The control signals are based on the specific combination of ALU function bits and control signals, ensuring proper coordination of ALU operations.

**11] Program Counter** **module**

Program\_Counter Model is designed to implement a program counter for a processor.

this module facilitates the generation of the next program counter address based on the current program counter, a potential branch or jump instruction, and relevant control signals. It incorporates an adder and multiplexers to handle different scenarios and ensure proper program flow.