Table of Contents

[Abstract 3](#_Toc125141444)

[Objective: 4](#_Toc125141445)

[Introduction: 4](#_Toc125141446)

[The basic elements of a processor 4](#_Toc125141447)

[CPU Operations:- 5](#_Toc125141448)

[Components and how CPUs work:- 5](#_Toc125141449)

[Types:- 5](#_Toc125141450)

[MODULES:- 6](#_Toc125141451)

[Multiplexer: 6](#_Toc125141452)

[Program counter: 6](#_Toc125141453)

[Accumulator: 7](#_Toc125141454)

[Arithmetic Logic Unit: 7](#_Toc125141455)

[Counter increment: 8](#_Toc125141456)

[Control unit: 8](#_Toc125141457)

[Data memory: 9](#_Toc125141458)

[Instruction Memory: 10](#_Toc125141459)

[Data Path: 11](#_Toc125141460)

[Central Processing Unit (CPU): 11](#_Toc125141461)

[Test Bench: 12](#_Toc125141462)

[Output: 13](#_Toc125141463)

[RTL Schematic Diagram: 13](#_Toc125141464)

[Conclusion: 14](#_Toc125141465)

**ACKNOWLEDGEMENT**

I would like to express my sincere gratitude to Dr. Wasiq Ali for imparting his invaluable knowledge and expertise in the field of DSD (Digital Systems Design). His passion for teaching and dedication to his students has greatly contributed to my understanding and appreciation of the subject. His guidance and support throughout the course have been instrumental in my learning experience. I am grateful for the opportunity to have been taught by such a knowledgeable and experienced instructor. Thank you, Dr. Wasiq Ali, for your dedication and commitment to education. Your contributions to the field and to your students will not be forgotten.

I would like to extend my sincerest appreciation to my friend Kaleem for his invaluable help and support throughout this project. His willingness to share his knowledge and expertise has been instrumental in my understanding and progress in the field.

I would also like to express my gratitude to Miss Ayesha for her guidance and support. Her dedication to her students and her expertise in the field have greatly contributed to my understanding and progress. Her encouragement and support have been a constant source of motivation, and I am grateful for the opportunity to have had her as a mentor and guide.

Thank you, my friend Kaleem and Miss Ayesha, for your unwavering support and guidance. Your contributions to my learning and development will not be forgotten.

At last we would like to acknowledge that this project was completed entirely by group and not by someone else.

# Abstract

The project's primary goal is to simulate and synthesize an 8-bit RISC CPU based on MIPS. The project entails modelling and designing a straightforward RISC-V processor. A microprocessor known as a Reduced Instruction Set Compiler (RISC) was created to execute a condensed set of instructions in order to boost the processor's overall speed. In this paper, we examine the RISC (Reduced Instruction Set Computer) CPU instruction set's MIPS instruction structure, instruction data flow, decoder module function, and design theory. Additionally, we successfully simulate using a pipeline design technique that uses 8-bit CPU modules based on the RISC CPU instruction set for the instruction fetch (IF), instruction decoder (ID), execution (EXE), data memory (MEM), and write back (WB) functions. The IF module's primary duty is to retrieve instructions from memory. The purpose of the ID stage is to transmit control commands, or instructions, to the control unit for decoding. The EXE stage performs math operations. ALU is the primary element of the EXE stage. If the instruction is not a memory/IO instruction, the result is forwarded to the WB stage. The MEM stage is used to retrieve data from memory and store data in memory. The last step of the WB process involves writing the results, storing data, and entering data into a register file. WB stage's goal is to write data to the destination register. The goal of this project was to build a RISC processor in VHDL that could be readily integrated into a bigger design in the future. In systems where an issue is simple to solve in software but complex to tackle using control logic, it will be helpful. However, it is simpler to implement the function in software when the complexity is large. For logical verification in this simulation project, we employ Modalism. We then synthesize it on the Xilinx-ISE tool with the target technology and carry out placement and routing operations for system verification. The tools needed are Simulation XILINX-ISE 10.1 - Synthesis and the language we used was HDL. Bottling plants and autonomous robot control are the applications.

**8-BIT Single Cycle MIPS PROCESSOR**

# Objective:

 The objective of the project is to simulate the operation of MIPS processor, and develop a graphical representation of it in Verilog language.  In this project, the [data path](http://www-ee.eng.hawaii.edu/~sasaki/EE361/Fall99/ChrisChan/Report.html#Datapath) with the entire module for different implementations of the MIPS instruction set was constructed, and an implementation that includes a subset of the core [MIPS instruction](http://www-ee.eng.hawaii.edu/~sasaki/EE361/Fall99/ChrisChan/Report.html#MIPS%20Instructions) set was also simulated.  And of course after doing the project, the following main points would be clearly understood.

* The main module of the MIPS processor.
* Different modules used as construction block for designing MIPS processor.
* Modules required when running a particular instruction of each type of MIPS.

# Introduction:

 MIPS is an instruction set developed by Sony, Nintendo, and NEC. MIPS, which stands  for  Million Instructions  Per  Second,  is a  rating  of  a  Central Processing Unit (CPU), that refers to how many low-level machine code instructions a processor can execute in one second.

A processor (CPU) is the logic circuitry that responds to and processes the basic [instructions](https://www.techtarget.com/whatis/definition/instruction) that drive a computer. The CPU is seen as the main and most crucial integrated circuitry (IC) chip in a computer, as it is responsible for interpreting most of computers commands. CPUs will perform most basic arithmetic, logic and I/O operations, as well as allocate commands for other chips and components running in a computer.

The term processor is [used interchangeably](https://searchservervirtualization.techtarget.com/tip/CPU-vs-microprocessor-What-are-the-differences) with the term central processing unit ([CPU](https://www.techtarget.com/whatis/definition/processor)), although strictly speaking, the CPU is not the only processor in a computer. The [GPU](https://www.techtarget.com/searchvirtualdesktop/definition/GPU-graphics-processing-unit) (graphics processing unit) is the most notable example, but the hard drive and other devices within a computer also perform some processing independently. Nevertheless, the term processor is generally understood to mean the CPU.

Processors can be found in PCs, smart phones, tablets and other computers. The two main competitors in the processor market are Intel and [AMD](https://searchservervirtualization.techtarget.com/definition/AMD).

## The basic elements of a processor

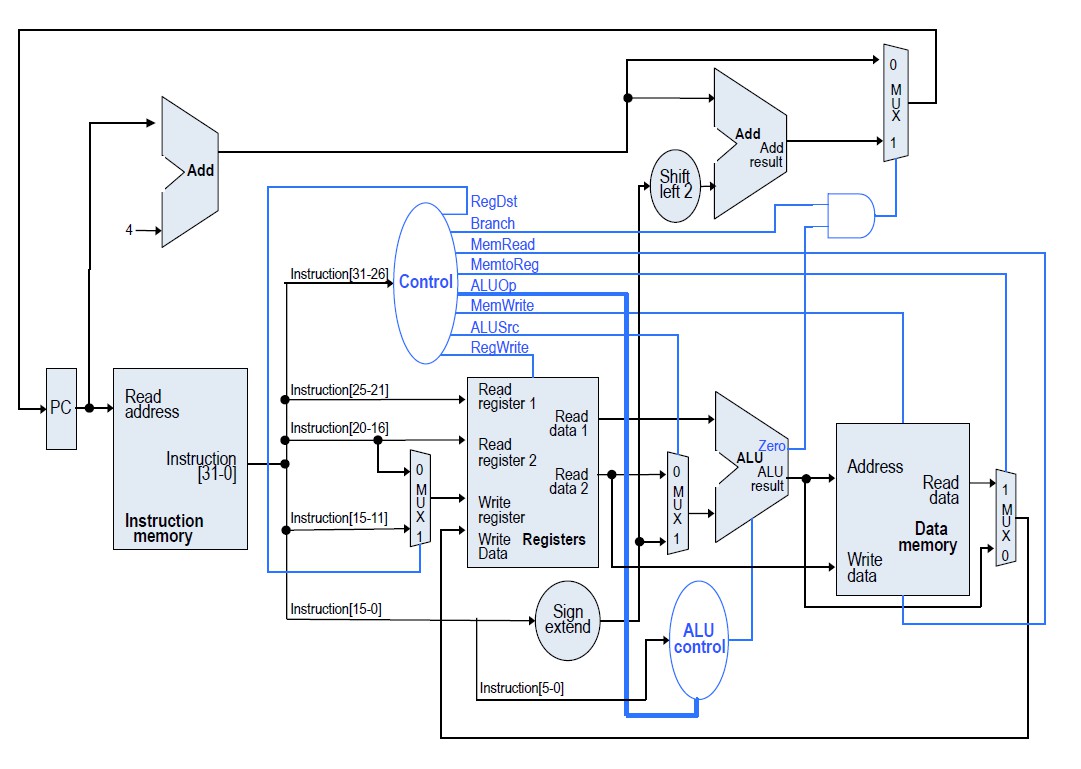
The basic elements of a processor include:

The arithmetic logic unit ([ALU](https://www.techtarget.com/whatis/definition/arithmetic-logic-unit-ALU)), which carries out arithmetic and logic [operations](https://www.techtarget.com/whatis/definition/operator) on the [operands](https://www.techtarget.com/whatis/definition/operand) in [instructions](https://www.techtarget.com/whatis/definition/instruction).

[Registers](https://www.techtarget.com/whatis/definition/register), which hold instructions and other data. Registers supply operands to the ALU and store the results of operations.

[L1 and L2](https://www.techtarget.com/whatis/definition/L1-and-L2) [cache memory](https://www.techtarget.com/searchstorage/definition/cache-memory). Their inclusion in the CPU saves time compared to having to get data from random access memory ([RAM](https://www.techtarget.com/searchstorage/definition/RAM-random-access-memory)).

Control unit, which controls all the operations and flow of data and instructions.



## CPU Operations:-

The four primary [functions](https://www.techtarget.com/whatis/definition/function) of a processor are [fetch](https://searchsqlserver.techtarget.com/definition/fetch), decode, execute and write back. Fetch- is the operation which receives instructions from program memory from systems RAM. Decode- is where the instruction is converted to understand which other parts of the CPU are needed to continue the operation. This is performed by the instruction decoder. Execute- is where the operation is performed. Each part of the CPU that is needed is activated to carry out the instructions.

## Components and how CPUs work:-

The main components of a CPU are the ALU, registers and control unit. The basic functions of the ALU and register are labeled in the above “basic elements of a processor section.” The control unit is what operates the fetching and execution of instructions.

The processor in a personal computer or embedded in small devices is often called a microprocessor. That term means that the processor's elements are contained in a single IC chip. Some computers will operate using a [multi-core processor](https://www.techtarget.com/searchdatacenter/definition/multi-core-processor)—a chip containing more than one CPU. A CPU is typically a small device with pins on it facing down in a motherboard. CPUs can also be attached to a motherboard with a [heat sink](https://www.techopedia.com/definition/2211/heat-sink) and a fan to dissipate heat.

## Types:-

Most processors today are [multi-core](https://www.techtarget.com/searchdatacenter/definition/multi-core-processor), which means that the IC contains two or more [processors](https://www.techtarget.com/whatis/definition/processor) for enhanced performance, reduced power consumption and more efficient simultaneous processing of multiple tasks (see: [parallel processing](https://www.techtarget.com/searchdatacenter/definition/parallel-processing)). Multi-core set-ups are similar to having multiple, separate processors installed in the same computer, but because the processors are actually plugged into the same socket, the connection between them is faster.

Most computers may have up to two-four cores; however, this number can increase up to 12 cores, for example. If a CPU can only process a single set of instructions at one time, then it is considered as a single-core processor. If a CPU can process two sets of instructions at a time it is called a dual-core processor; four sets would be considered a [quad-core processor](https://www.techtarget.com/whatis/definition/quad-core-processor). The more cores, the more instructions at a time a computer can handle. Some processors use multi-threading, which uses virtualized processor cores. Virtualized processor cores are called CPUs. These are not as powerful as physical cores but can be used to [improve performance](https://www.techtarget.com/searchdatacenter/tip/Improvements-in-CPU-features-help-shape-selection) in virtual machines ([VMs](https://searchservervirtualization.techtarget.com/definition/virtual-machine)). However, adding unnecessary CPUs can hurt consolidation ratios, so there should be about four-six CPUs per physical core.

# MODULES:-

## Multiplexer:

In electronics, a multiplexer (or mux; spelled sometimes as multiplexor), also known as a data selector, is a device that selects between several analog or digital input signals and forwards the selected input to a single output line. The selection is directed by a separate set of digital inputs known as select lines.

|  |  |
| --- | --- |
| module Mux2to1\_8Bit  (  input [7:0]i0,i1,input sel,  output [7:0]mux\_out  );  assign mux\_out =sel?i1:i0;  endmodule | module Mux2to1\_6Bit  (  input [4:0] i0, i1,input sel,  output[4:0] mux\_out  );  assign mux\_out = sel ? i1 : i0;  endmodule |

## Program counter:

The program counter (PC) is a register that manages the memory address of the instruction to be executed next.

The address specified by the PC will be + n (+1 for a 1-word instruction and +2 for a 2-word instruction) each time one instruction is executed.  
However, in the case of an interrupt instruction, etc., the jump destination address is stored.

The CPU reads the address where the instruction to be executed next is stored from the PC and executes it in sequence.

|  |
| --- |
| module ProgramCounter  (  input [4:0]d\_in,  input reset, clk,  output reg [4:0] d\_out  );  always @(posedge clk)  if (reset)  d\_out <= 5'b00000;  else  d\_out <= d\_in;  endmodule |

## Accumulator:

An accumulator is a type of [register](https://www.computerhope.com/jargon/r/register.htm) included in a [CPU](https://www.computerhope.com/jargon/c/cpu.htm). It acts as a temporary storage location which holds an intermediate value in mathematical and logical calculations. Intermediate results of an operation are progressively written to the accumulator, overwriting the previous value. For example, in the operation "3 + 4 + 5," the accumulator would hold the value 3, then the value 7, then the value 12. The benefit of an accumulator is that it does not need to be explicitly referenced, which conserves data in the operation statement.

|  |
| --- |
| module Accumulator(  input [7:0] d\_in,  input load, clk,  output reg [7:0] d\_out  );  always @(posedge clk)  if (load)  d\_out <= d\_in;  initial  d\_out=8'h00;  endmodule |

## Arithmetic Logic Unit:

An arithmetic logic unit (ALU) is a digital circuit used to perform arithmetic and logic operations. It represents the fundamental building block of the central processing unit (CPU) of a computer. Modern CPUs contain very powerful and complex ALUs. In addition to ALUs, modern CPUs contain a control unit (CU).

Most of the operations of a CPU are performed by one or more ALUs, which load data from input registers. A register is a small amount of storage available as part of a CPU. The control unit tells the ALU what operation to perform on that data, and the ALU stores the result in an output register. The control unit moves the data between these registers, the ALU, and memory.

|  |
| --- |
| module ALU(  input [7:0]a, input [7:0]b,input [2:0]opcode,  output reg [7:0]alu\_out  );  always @(opcode,a,b)  case(opcode)  3'b000:alu\_out = a + b;  3'b001:alu\_out = a - b;  3'b010:alu\_out = a&b;  3'b011:alu\_out = a|b;  3'b100:alu\_out = ~b;  3'b101:alu\_out = a^b;  3'b110:alu\_out = a~^b;  default:alu\_out = 0;  endcase  endmodule |

## Counter increment:

In digital logic and computing, a counter is a device which stores (and sometimes displays) the number of times a particular event or process has occurred, often in relationship to a clock. The most common type is a sequential digital logic circuit with an input line called the clock and multiple output lines.

|  |
| --- |
| module CounterIncrement  (  input [4:0]a, input [4:0]b,  output[4:0] adder\_out  );  assign adder\_out = a + b;  endmodule |

## Control unit:

The control unit (CU) is a component of a computer's [central processing unit](https://en.wikipedia.org/wiki/Central_processing_unit) (CPU) that directs the operation of the processor. A CU typically uses a [binary decoder](https://en.wikipedia.org/wiki/Binary_decoder) to convert coded instructions into timing and control signals that direct the operation of the other units (memory, [arithmetic logic unit](https://en.wikipedia.org/wiki/Arithmetic_logic_unit) and input and output devices, etc.).

Most computer resources are managed by the CU. It directs the flow of data between the CPU and the other devices. [John von Neumann](https://en.wikipedia.org/wiki/John_von_Neumann) included the control unit as part of the [Von Neumann architecture](https://en.wikipedia.org/wiki/Von_Neumann_architecture).[[1]](https://en.wikipedia.org/wiki/Control_unit#cite_note-1) In modern computer designs, the control unit is typically an internal part of the [CPU](https://en.wikipedia.org/wiki/Central_processing_unit) with its overall role and operation unchanged since its introduction.

|  |
| --- |
| module Controller  (  input [2:0] opcode,  output reg rd\_mem,wr\_mem,ac\_src,ld\_ac,pc\_src,jmp\_uncond);  always @(opcode)  begin  rd\_mem = 1'b0;  wr\_mem = 1'b0;  ac\_src = 1'b0;  pc\_src = 1'b0;  ld\_ac = 1'b0;  jmp\_uncond=1'b0;  case (opcode)  3'b000: //load accumulator from memory  begin  rd\_mem = 1'b1;  wr\_mem = 1'b0;  ld\_ac = 1'b1;  ac\_src = 1'b0;  end  3'b001:  begin  rd\_mem = 1'b1;  wr\_mem = 1'b0;  ld\_ac = 1'b1;  ac\_src = 1'b0;//SUBTRACT  end  3'b010:  begin  rd\_mem = 1'b1;  wr\_mem = 1'b0;  ld\_ac = 1'b1;  ac\_src = 1'b0;//AND  end  3'b011:  begin  rd\_mem = 1'b1;  wr\_mem = 1'b0;  ld\_ac = 1'b1;  ac\_src = 1'b0;//OR  end  3'b100:  begin  rd\_mem = 1'b1;  wr\_mem = 1'b0;  ld\_ac = 1'b1;  ac\_src = 1'b0;//NOT  end  3'b101:  begin  rd\_mem = 1'b1;  wr\_mem = 1'b0;  ld\_ac = 1'b1;  ac\_src = 1'b0;//XOR  end  3'b110:  begin  rd\_mem = 1'b1;  wr\_mem = 1'b0;  ld\_ac = 1'b1;  ac\_src = 1'b0;//XNOR  end  3'b111:  begin  rd\_mem = 1'b0;  wr\_mem = 1'b0;  ld\_ac = 1'b0;  ac\_src = 1'b0;  pc\_src=1'b1;  jmp\_uncond=1'b1;//JUMP  end  default:  begin  rd\_mem = 1'b0;  wr\_mem = 1'b0;  ac\_src = 1'b0;  pc\_src = 1'b0;  ld\_ac = 1'b0;  end  endcase //end case  end //end always  endmodule |

## Data memory:

The data memory modules store frames, which are managed by the data [memory management](https://www.sciencedirect.com/topics/computer-science/memory-management) units, with one DMMU per memory module. Each DMM is organized logically with a three-level hierarchy: blocks, frames, and queues. Memory is organized as a sequence of fixed size blocks, which store frame data. Because a frame may need more than one block to be stored, several blocks are used for a frame and are organized logically with a linked list that indicates the sequence of data bytes within the frame. Frames are organized in logical queues, depending on several of their parameters, such as priority, multicast or [unicast](https://www.sciencedirect.com/topics/computer-science/unicast) requirements, outgoing NAU, etc.

|  |
| --- |
| module DataMemory  (  input rd, wr,  input [4:0] abus,  input [7:0] in\_dbus,  output reg [7:0] out\_dbus);  reg [7:0] dm\_array [0:31];  always @(rd,abus)  begin  if (rd)  out\_dbus = dm\_array [abus];  end  always @(wr,in\_dbus) //always @(wr or abus or in\_dbus)  begin  if (wr)  dm\_array [abus] = in\_dbus;  end  initial  begin  dm\_array[0] = 8'h01;  dm\_array[1] = 8'h02;  dm\_array[2] = 8'h03;  dm\_array[3] = 8'h04;  dm\_array[4] = 8'h05;  end  endmodule |

## Instruction Memory:

The Instruction Memory (IM) stores all the prefetch instructions. It is composed of 5 major components: [the PC (Program Counter) Unit](https://www.ece.rice.edu/Courses/422/1997/mingfai/elec422/imfd.html#PC), [PC Decoder](https://www.ece.rice.edu/Courses/422/1997/mingfai/elec422/imfd.html#pcdec), [INBUF (an input buffer to the IM)](https://www.ece.rice.edu/Courses/422/1997/mingfai/elec422/imfd.html#inbuf), [IM Storage](https://www.ece.rice.edu/Courses/422/1997/mingfai/elec422/imfd.html#im), [OUTBUF (an output buffer to the internal bus)](https://www.ece.rice.edu/Courses/422/1997/mingfai/elec422/imfd.html#outbuf). All the memory elements (registers/latches) are qualified with clka.

|  |
| --- |
| module InstructionMemory  (input [4:0] abus, output reg [7:0] dbus);  reg [7:0] im\_array [0:12];  always @(abus)  dbus = im\_array [abus];  initial  begin  im\_array[0]= 8'h00; // Initialize Accumulator with 0 and do addition with content of DataMemory at address 0.  im\_array[1]= 8'h21; // Subtract content of accumulator with content of DataMemory at address 1.  im\_array[2]= 8'h42; // Logical AND of accumulator with content of DataMemory at address 2.  im\_array[3]= 8'h63; // Logical OR of accumulator with content of DataMemory at address 3.  im\_array[4]= 8'h84; // Logical NOT of accumulator with content of DataMemory at address 4.  im\_array[5]= 8'hA4; // Logical XOR of accumulator with content of DataMemory at address 4.  im\_array[6]= 8'hC4; // Logical XNOR of accumulator with content of DataMemory at address 4.  im\_array[7]= 8'hEA; // Unconditional Jump to 01010 address of Instruction memory.  im\_array[10]= 8'h00; // Addition with content of DataMemory at address 0.  im\_array[11]= 8'hE0; // Unconditional Jump to 00000 address of Instruction memory.  end  endmodule |

## Data Path:

A datapath is a collection of [functional units](https://en.wikipedia.org/wiki/Functional_unit) such as [arithmetic logic units](https://en.wikipedia.org/wiki/Arithmetic_logic_unit) or [multipliers](https://en.wikipedia.org/wiki/Binary_multiplier) that perform data processing operations, [registers](https://en.wikipedia.org/wiki/Processor_register), and [buses](https://en.wikipedia.org/wiki/Bus_(computing)).[[1]](https://en.wikipedia.org/wiki/Datapath#cite_note-NullLobur2006-1) Along with the [control unit](https://en.wikipedia.org/wiki/Control_unit) it composes the [central processing unit (CPU)](https://en.wikipedia.org/wiki/Central_processing_unit).[[1]](https://en.wikipedia.org/wiki/Datapath#cite_note-NullLobur2006-1) A larger datapath can be made by joining more than one datapaths using [multiplexers](https://en.wikipedia.org/wiki/Multiplexers). A data path is the ALU, the set of registers, and the CPU's internal bus(es) that allow data to flow between them.

|  |
| --- |
| module DataPath(  input reset,ld\_ac, ac\_src, pc\_src, clk,  output [2:0] opcode,  output [4:0] im\_abus,  input [7:0] im\_dbus,  output [4:0] dm\_abus,  output [7:0] dm\_in\_dbus,  input [7:0] dm\_out\_dbus,  output [7:0] ac\_out,alu\_out);  //wire [7:0] ac\_out,alu\_out,mux2\_out;  wire [7:0]mux2\_out;  wire [4:0] pc\_out, adder\_out,mux1\_out;  ProgramCounter pc(.d\_in(mux1\_out),.reset(reset),.clk(clk),.d\_out(pc\_out)); //instantiationof all module  CounterIncrement adder(.a(pc\_out),.b(5'b00001),.adder\_out(adder\_out));  Mux2to1\_6Bit mux1(.i0(adder\_out),.i1(im\_dbus[4:0]),.sel(pc\_src),.mux\_out(mux1\_out));  Accumulator ac(.d\_in(mux2\_out),.load(ld\_ac),.clk(clk),.d\_out(ac\_out));  ALU alu(.a(ac\_out),.b(dm\_out\_dbus),.opcode(opcode),.alu\_out(alu\_out));  Mux2to1\_8Bit mux2(.i0(alu\_out),.i1(dm\_out\_dbus),.sel(ac\_src),.mux\_out(mux2\_out));  assign im\_abus = pc\_out; //assign im\_abus = 6'b000000;  assign opcode = im\_dbus [7:5];  assign dm\_abus = im\_dbus [4:0]; //abus for DataMemory.  assign dm\_in\_dbus=ac\_out;  endmodule |

## Central Processing Unit (CPU):

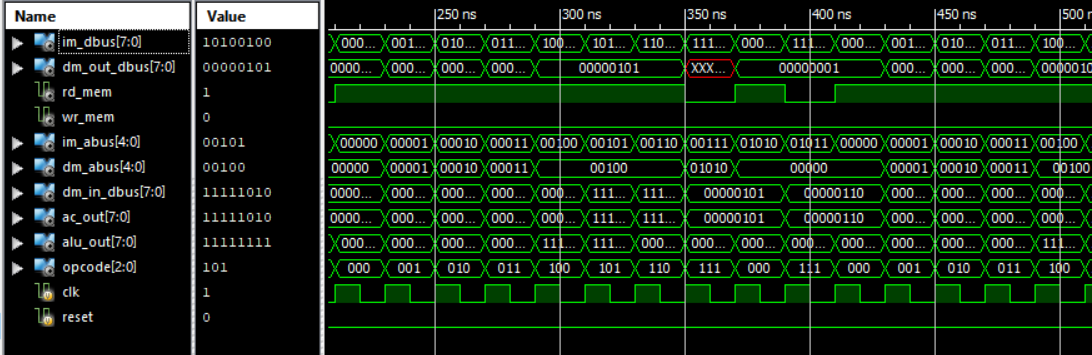
The Central Processing Unit (CPU) is the primary component of a computer that acts as its “control center.” The CPU, also referred to as the “central” or “main” processor, is a complex set of electronic circuitry that runs the machine’s operating system and apps. The CPU interprets, processes and executes instructions, most often from the hardware and software programs running on the device.The CPU performs arithmetic, logic, and other operations to transform data input into more usable information output. While the CPU must contain at least one processing core, many contain multiple cores. A server with two hexa-core CPUs, for example, will have a total of 12 processors.

|  |
| --- |
| module CPU( //The CPU  input clk,reset,  output rd\_mem,wr\_mem,  output [4:0] im\_abus, input [7:0] im\_dbus,  output [4:0] dm\_abus, output [7:0] dm\_in\_dbus,  input [7:0] dm\_out\_dbus,  output [7:0] ac\_out,alu\_out,  output [2:0] opcode);  //wire [2:0] opcode;  wire ac\_src,ld\_ac, pc\_src,jmp\_uncond;  DataPath dpu  (.reset(reset),.ld\_ac(ld\_ac),.ac\_src(ac\_src),.pc\_src(pc\_src),.clk(clk),.opcode(opcode)  ,.im\_abus(im\_abus),.im\_dbus(im\_dbus),.dm\_abus(dm\_abus),.dm\_in\_dbus(dm\_in\_dbus),.dm\_out\_dbus(dm\_out\_dbus),.ac\_out(ac\_out),.alu\_out(alu\_out));//dj  Controller cu  (.opcode(opcode),.rd\_mem(rd\_mem),.wr\_mem(wr\_mem),.ac\_src(ac\_src),.ld\_ac(ld\_ac),  .pc\_src(pc\_src),.jmp\_uncond(jmp\_uncond));  endmodule |

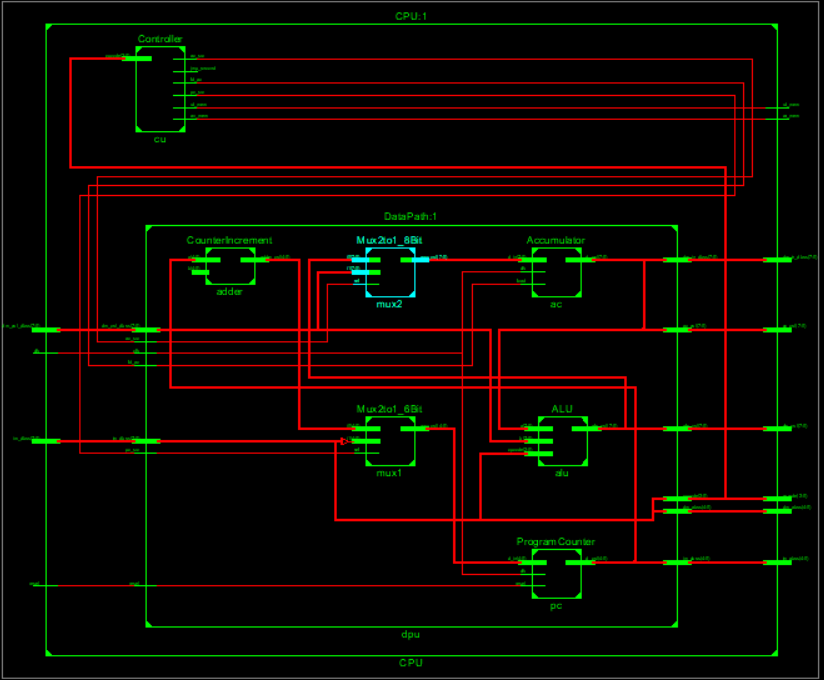
## Test Bench:

|  |
| --- |
| module testBench;  reg clk;  reg reset;  wire [7:0] im\_dbus;  wire [7:0] dm\_out\_dbus;  wire rd\_mem;  wire wr\_mem;  wire [4:0] im\_abus;  wire [4:0] dm\_abus;  wire [7:0] dm\_in\_dbus;  wire [7:0] ac\_out,alu\_out;  wire [2:0] opcode;  CPU uut (  .clk(clk),.reset(reset),.rd\_mem(rd\_mem),.wr\_mem(wr\_mem),  .im\_abus(im\_abus),.im\_dbus(im\_dbus),.dm\_abus(dm\_abus),  .dm\_in\_dbus(dm\_in\_dbus),.dm\_out\_dbus(dm\_out\_dbus),.ac\_out(ac\_out),.alu\_out(alu\_out),.opcode(opcode));  InstructionMemory IM (.abus(im\_abus),.dbus(im\_dbus));  DataMemory DM  (.rd(rd\_mem),.wr(wr\_mem),.abus(dm\_abus),.in\_dbus(dm\_in\_dbus),.out\_dbus(dm\_out\_dbus)  );  initial  begin  clk = 0; reset = 1;//im\_dbus =8'hxx;dm\_out\_dbus = 8'b00000000;  #20 reset = 1'b0;  #500 $finish;  end  always  #10 clk = ~clk;  endmodule |

# Output:



# RTL Schematic Diagram:



# Conclusion:

The processor is a chip or a logical circuit that responds and processes the basic instructions to drive a particular computer. The main functions of the processor are fetching, decoding, executing, and write back the operations of an instruction. The processor is also called the brain of any system which incorporates computers, laptops, smartphones, embedded systems, etc. The ALU (Arithmetic Logic Unit) and CU (Control Unit) are the two parts of the processors. The Arithmetic Logic Unit performs all mathematical operations such as additions, multiplications, subtractions, divisions, etc. and the control unit works like traffic police, it manages the command or the operation of the instructions. The processor communicates with the other components also they are input/output devices and memory/storage devices.