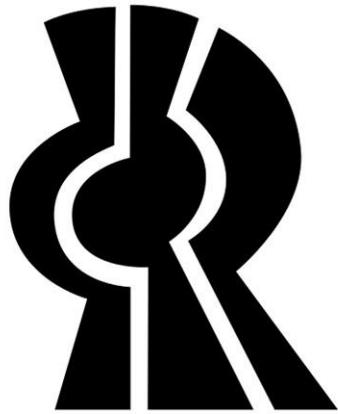




CENTRO UNIVERSITARIO DE
CIENCIAS EXACTAS E INGENIERÍAS



IL365 – Arquitectura de Computadoras

Proyecto Final

Profesor: Jorge Ernesto López Arce Delgado

Ciclo: 2025 B **Horario:** lunes y miércoles de 9-11 Am.

NRC: 200204 **SEC:** D03

Fecha: 01/12/2025

Integrantes:

Jiménez Sánchez Antonio

Díaz López Iris Vianney

Introducción.

En este proyecto se busca hacer un diseño de un procesador MIPS cumpliendo con sus requerimientos de sus diferentes etapas, buffers y decodificador hecho con Python el cual consiste en llegar a la parte de lenguaje ensamblador para que decodifique el lenguaje ordinario a lenguaje binario, cumpliendo con sus diferentes tipos de instrucciones que se solicitaron como la I, J, R.

El pipeline implementado sigue el modelo clásico MIPS con las etapas: IF (Instruction Fetch), ID (Instruction Decode), EX (Execute), MEM (Memory Access) y WB (Write Back), soportando un conjunto de instrucciones que incluye operaciones aritméticas, lógicas, acceso a memoria y control de flujo.

Objetivo general.

Se busca integrar el procesamiento Pipeline MIPS que funcione a base de verilog.

Usando las diferentes etapas requeridas para su funcionamiento.

Junto con su decodificador de instrucciones hecho con Python para generar los archivos .txt y ponerlos en el mips para más automatización y eficiencia .

Objetivos particulares.

1.Verilog

-Diseñar módulos y componentes para el funcionamiento del programa como: buffers, procesos aritméticos, bancos de registro, memorias etc...

2.Python

-Se busca desarrollar un decodificador de instrucciones para convertir textos en lenguaje común a binario para inyectarlos en el proyecto los datos para la eficiencia del mismo.

Desarrollo.

El proyecto consta de diferentes etapas como;

Memoria_datos.v

Es una memoria que tiene 256 espacios para almacenar información donde guarda números de 32 bits como máximo de alcance.

Carga la información desde un archivo llamado datos_memoria.txt que contiene la información en binario.

Parte del código:

```
`timescale 1ns/1ns
```

```
module memoria_datos(
```

```
    input clk,
```

```
    input mem_escribir,
```

```
    input mem_leer,
```

```
    input [31:0] direccion,
```

```
    input [31:0] dato_escribir,
```

```
    output reg [31:0] dato_leer
```

```
);
```

```
reg [31:0] memoria [0:255];
```

```
initial begin
```

```
    $readmemb("datos_memoria.txt", memoria);
```

```
end
```

```
always @ (posedge clk) begin
```

```
    if (mem_escribir) begin
```

```
        memoria[direccion[7:0]] <= dato_escribir;
```

```
    end
```

Ciclo fetch es como el corazón de la cpu basica por decirlo asi, busca continuamente la información que se debe usar para ejecutar las instrucciones del procesador.

Tiene un program counter que guarda la dirección de memoria y un ciclo de reloj que suma 4 a la dirección para pasar a la siguiente instrucción.

```
`timescale 1ns/1ns
```

```
module ciclo_fetch(
```

```
    input clk,
```

```
    input rst_tb,
```

```
output [31:0] instrucion_fetch,  
output [31:0] pc_plus4  
);  
wire [31:0] cable1,cable2;
```

```
program_counter inst1(  
.clk_pc(clk),  
.reset(rst_tb),  
.pc_in(cable2),  
.pc_out(cable1)  
);
```

```
adicion inst2(  
.opIn(cable1),  
.opIn2(32'd4),  
.opNext(cable2)  
);
```

```
assign pc_plus4 = cable2;
```

```
memoria_instrucciones inst3(  
.dir(cable1[7:0]),  
.inst(instrucion_fetch)  
);  
Endmodule
```

Banco de registro

Funciona con la memoria rápida del procesador con 32 registros de 32 bits cada uno ,teniendo dos modos de uso lectura y escritura , con lectura entrega los datos de forma rs y rt.

Para escritura se utiliza el rd durante el flanco de subida del reloj.

```
`timescale 1ns/1ns
```

```
module bancoRegistros(
```

```
    input clk,  
    input reg_escribir,  
    input [4:0] rs,  
    input [4:0] rt,  
    input [4:0] rd,  
    input [31:0] dato_escribir,
```

```

    output reg [31:0] dr1,
    output reg [31:0] dr2
);

reg [31:0] registros [0:31];

initial begin
    $readmemb("datos2.txt", registros);
end

always @(*) begin
    dr1 = registros[rs];
    dr2 = registros[rt];
end

always @(posedge clk) begin
    if (reg_escribir && rd != 0) begin
        registros[rd] <= dato_escribir;
    end
end

endmodule

```

Los buffers son como un procesador segmentado, sincroniza y almacena la información entre las etapas de las fases.

```

`timescale 1ns/1ns
module buf1(
    input clk_buf1,
    input reset_buf1,

    input [31:0] instrucion_in,
    input [31:0] pc_plus4_in,

    output reg [31:0] instrucion_out,
    output reg [31:0] pc_plus4_out
);

always @(posedge clk_buf1 or posedge reset_buf1) begin
    if (reset_buf1) begin

```

```

    instrucion_out <= 32'b0;
    pc_plus4_out   <= 32'b0;
end
else begin
    instrucion_out <= instrucion_in;
    pc_plus4_out   <= pc_plus4_in;
end
end

endmodule

```

La unidad de control interpreta el código de cada instrucción y manda las señales al procesador para que sepa que es lo que tiene que hacer .

`timescale 1ns/1ns

```

module unidad_control(
    input [5:0] codigo_operacion,
    output reg destino_reg,
    output reg branch,
    output reg mem_leer,
    output reg mem_a_reg,
    output reg [1:0] alu_operacion,
    output reg mem_escribir,
    output reg alu_fuente,
    output reg reg_escribir,
    output reg salto
);

```

Consistió en el desarrollo de 5 etapas (IF, ID, EX, MEM, WB), implementado en lenguaje verilog acompañado de su decodificador Python con interfaz gráfica para que sea de forma más fácil de usar por el usuario.

Conclusiones.

Se logro implementar de forma exitosa el proyecto hecho en verilog de 5 etapas con sus instrucciones y decodificador.

El desarrollo de este proyecto fue algo realmente retador para el equipo ya que se tuvo que adaptar cosas e investigar etapas que llevaban un grado de complejidad realmente grande.

Referencias.

<https://www.fpga4student.com/2016/11/verilog-microcontroller-code.html>

<https://www.chipverify.com/verilog/verilog-testbench-simulation>

<https://www.quora.com/Whats-the-Verilog-code-of-64-bit-instruction-memory>

<https://medium.com/@raoumer160903/8-bit-arithmetic-logic-unit-alu-implementation-in-verilog-38b545869a37>

<https://forum.digikey.com/t/fifo-buffer-module-with-watermarks-verilog-and-vhdl/13182>

<https://ithy.com/article/microprocessor-design-6y3rr4wp>

<https://medium.com/@LambdaMamba/building-a-mips-5-stage-pipeline-processor-in-verilog-6d627a31127c>