<!--Arquitectura de Computadoras Prof.:Ernesto Lopez -->

Actividad 6 {

<Por="Humberto Peña, Rebeca Hernandez, Elizabeth Arroyo"/>

<!--UdeG CUCEI ICOM-->

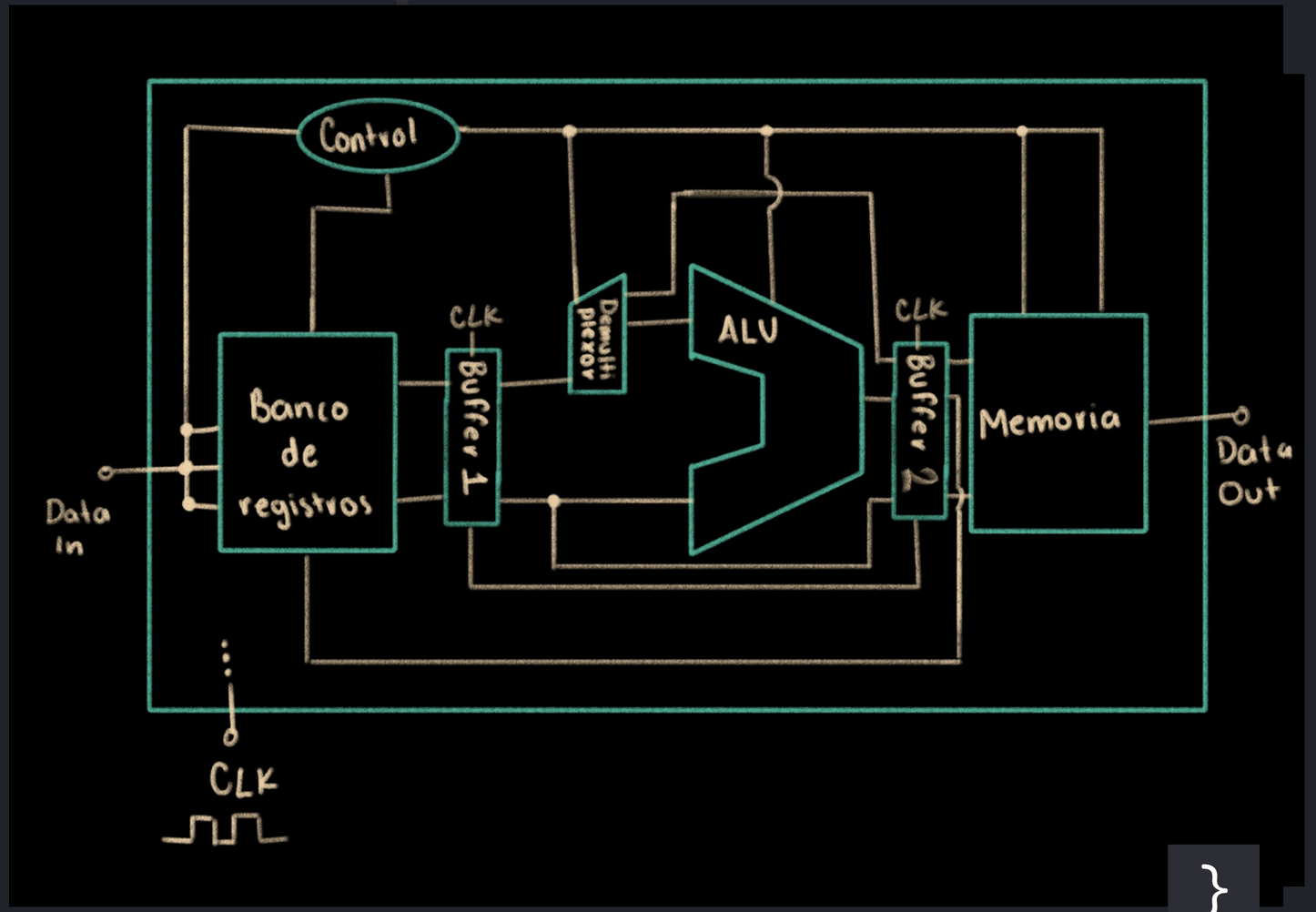
}

Contenidos

Introducción {

El diseño y el analisis de un procesador es simple por medio de sus módulos los cuales cumplen funciones especificas en el sistema permitiendo el buen funcionamiento de operaciones aritmeticas y de control.

El proyecto utiliza un diseño facil de entender separando por módulos como la ALU, Banco de Registros y la Memoria.



}

Módulos {

BANCO DE REGISTROS

ALU

MEMORIA DE DATOS

DEMULTIPLEXOR

BUFFER

}

Banco de Registros {

Es una memoria interna que guarda datos temporales en registros de 32 bits. El cuál permite leer y reescribir los valores dentro de ella.

¿Porqué un Banco de Registros y no una RAM?
El Banco de Registros se caracteriza por sus pequeñas memorias de acceso ultrarrápido que almacenan datos temporales durante su ejecución y su acceso es mucho más rápido que la RAM.

Entradas

RA1,RA2 Son las direcciones de lectura
WA Es la dirección de escritura
WD es el dato de 32 bits que se guardara en la dirección

Salidas

RD1, RD2 Los valores leídos del registro de RA1 y RA2

}

ALU {

La ALU es la Unidad Aritmético Lógica encargada de tales operaciones en el procesador, en nuestro proyecto utilizamos: AND, OR, ADD, SUB, SLT y NOR

- Las cuales sus entradas con dos valores de 32 bits y el código de operación.
- Su salida es el resultado de la operacion y una señal de estado.

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

}

Memoria de Datos {

La memoria de datos se encarga de leer y escribir los valores obtenidos de un archivo .txt

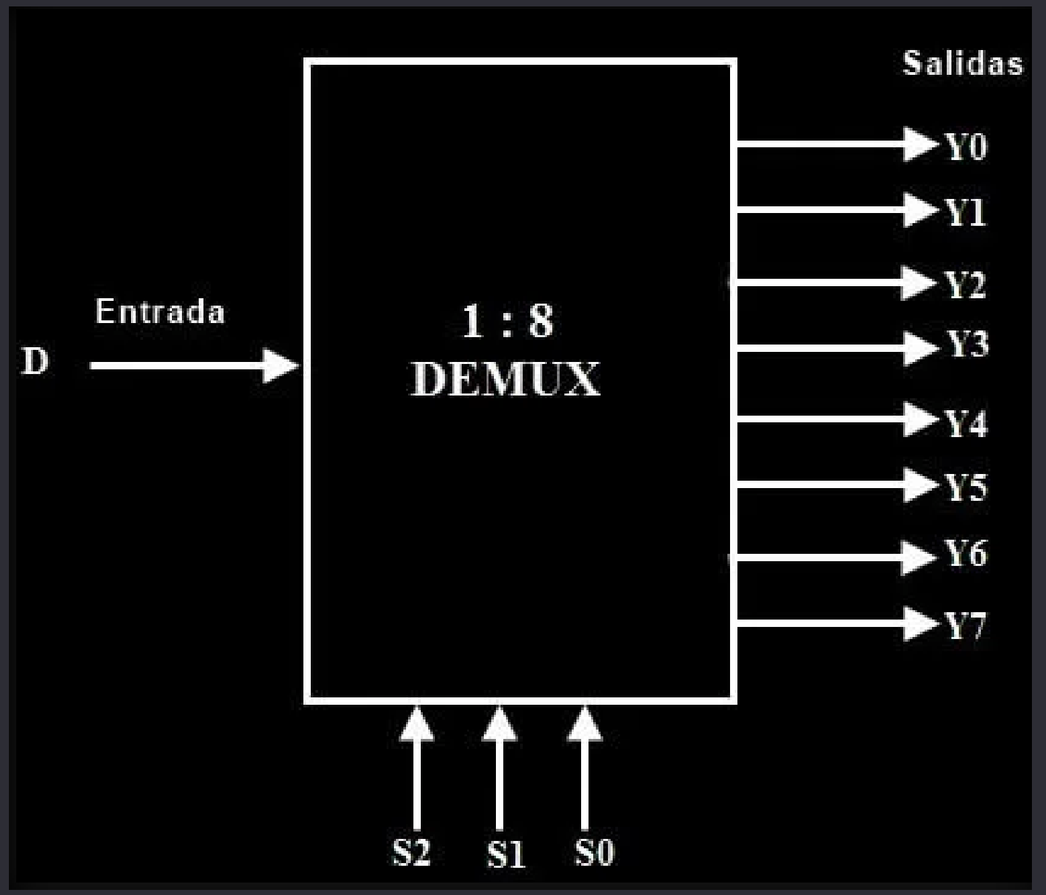
> 1) memoria de 32 bits: usando un índice para acceder a las posiciones
>
> 2) WEN y REN: usadas para activar su lectura y escritura

3) Utilizando un acceso asíncrono en la memoria permitiendo a las operaciones de lectura y de escritura ejecutarse de forma combinacional

}

Demultiplexor {

El Demultiplexor DEMUX toma la entrada y la lleva a una de varias salidas en señal de selección. La cual permite una sola linea de datos a varios destinos.
Se usa para direccionar datos en el sistema
Sus entradas IN y SEL determinan a donde llevan el dato.
Y como salidas son variadas pueden ser OUT1, OUT2, OUTn ya que solo una salida será la receptora de la información.



}

Buffer {

- Es fácil de sincronizar
- Guarda datos antes de su proceso
- Evita bloqueos

El buffer de datos funciona como una memoria temporal que guarda información en tránsito entre procesos, buscando un mejor rendimiento y evitando la pérdida de datos.

}

Módulo de Control {

- Decodifica la instrucción en función del código de opción de 4 bits.
- Genera señales de control necesarias para el procesador.

El módulo de Control interpreta el opcode de la instrucción y genera las señales necesarias para su ejecución. Define la operación de la ALU (aluOp), habilita la escritura en registros (regWrite) y controla el acceso a la memoria (memWrite), asegurando la correcta coordinación del procesador.

}

Principal (Jericalla Evolucionada) {

Este módulo es el encargado de dirigir a todos los módulos anteriores en un orden secuencial para que cumplan con su funcionamiento correctamente.

La secuencia comienza en la lectura de valores en el Banco de Registros mandandolos a la ALU para realizar una operación y despues redireccionarlos y almacenarlos en la memoria para que de ahi se pueda leer al final del procesador.

```
module Jericalla_evolucion(
    input wire clk,
    input wire [4:0] rs1, rs2, rd,
    input wire [2:0] alu_sel,
    input wire we, mem_we, buf_we, demux_en,
    output wire [31:0] alu_out, mem_out, buf_out
);

    wire [31:0] reg_out1, reg_out2;

    BR banco_registros (.clk(clk),
.we(we),.RA1(rs1),.RA2(rs2),.WA(rd),.DW(alu_out),.
DR1(reg_out1),.DR2(reg_out2));

    ALU alu_unit ( .A(reg_out1), .B(reg_out2),
.sel(alu_sel), .sal(alu_out));

    MemD memoria_datos (.clk(clk), .we(mem_we),
.ADDR(alu_out[4:0]), .DW(reg_out2), .RD(mem_out));

    Buffer buffer_unit (.clk(clk),
.enable(buf_we), .d_entrada(mem_out),
.d_salida(buf_out));

    Demux demux_unit (.data_in(buf_out),
.sel(alu_sel[1:0]), .enable(demux_en),.out0(),
.out1(), .out2(), .out3());

endmodule
```

}

Pruebas {

Antes de ejecutar

0	1	111	100	200	333	11	1	20
8	21	22	0	0	0	0	0	0
16	x	x	x	x	x	x	x	x
24	x	x	x	x	x	x	x	x

Despues de ejecutar

0	x	x	x	x	x	x	x	x
8	x	x	x	x	x	x	x	x
16	x	x	x	x	333	11	1	x
24	x	x	x	x	x	x	x	x

}

Errores durante el proyecto {

Existieron varios errrores que nos complicaron un como la ejecucion del código, uno de los principales errores fue el agregar en `$readmemb` dentro de el módulo de memoria de datos.

Este error ocurrió pensando que era necesario leer desde RAM la lista de datos seleccionada, pero en realidad esa función tiene que realizarse desde el TestBench.

```
$readmemb("dato.txt")
```

Error en módulo Jericalla

```
instancia en RAM
.wd(buffer2Out)
```

El módulo de Jericalla existió un error donde el wd instanciaba a buffer1Out, este error se corrigió revisando el diagrama del proyecto.

}

Conclusión {

Al realizar dividir el código para poder llevarlo a cabo de forma organizada observamos las distintas funciones que son fundamentales para el manejo de la memoria un claro ejemplo es la memoria de datos donde mediante un archivo de texto obtuvimos los datos de entrada que se usaron para leer y escribir los valores con direcciones de memoria específicas y en la lectura de memoria se buscó no depender del reloj para que se garantizara el acceso inmediato de los datos.

Esta práctica ayuda a reforzar el funcionamiento de la ALU y la manera de direccionar datos usando un banco de registros y una memoria, procesando datos y haciendo pruebas.

}

<!--Fin de presentación-->

Gracias {

}