PROYECTO FINAL FASE-1

JAUREGUI RABELERO HUGO JOHNATHAN
HERMOSILLO GUIZAR IRVIN YAEL
CRUZ SANCHEZ JUAN PABLO
COVARRUBIAS BECERRA JOSE ANTONIO

15/11/2023

Investigación

Procesador MIPS de 32 bits

El procesador MIPS (Microprocessor without Interlocked Pipeline Stages) de 32 bits es una arquitectura de procesador basada en RISC (Reduced Instruction Set Computing). A continuación se detallan algunos de sus elementos y características generales:

1. Unidad de control (Control Unit): es la parte del procesador que se encarga de controlar el flujo de datos y de instrucciones a través de las diferentes unidades del procesador.
2. Unidad aritmético-lógica (Arithmetic Logic Unit, ALU): es la unidad encargada de realizar las operaciones aritméticas y lógicas básicas, como sumar, restar, multiplicar, dividir, comparar y realizar operaciones booleanas
3. Arquitectura RISC: El procesador MIPS utiliza una arquitectura RISC (Reduced Instruction Set Computing) que se caracteriza por tener un conjunto de instrucciones reducido y simple. Esto permite que el procesador pueda ejecutar instrucciones de manera más rápida y eficiente
4. Memoria caché: el procesador MIPS utiliza una memoria caché para acelerar el acceso a los datos almacenados en memoria. La memoria caché es una memoria de acceso rápido que almacena copias de los datos más utilizados en la memoria principal.
5. FPU (Floating Point Unit): es la unidad encargada de realizar operaciones aritméticas en punto flotante, como sumas, restas, multiplicaciones y divisiones.
6. Arquitectura de canalización: El procesador MIPS utiliza una arquitectura de canalización para aumentar la eficiencia en la ejecución de instrucciones. La canalización permite que varias instrucciones se ejecuten al mismo tiempo, 1 reduciendo el tiempo de espera entre instrucciones.
7. Registro de propósito general: El procesador MIPS tiene 32 registros de propósito general de 32 bits, que se utilizan para almacenar datos y direcciones de memoria.

Set de instrucciones

INSTRUCCION ADD: La instrucción "add" es una instrucción de tipo R utilizada en la arquitectura de computadoras y en el lenguaje de programación ensamblador. La instrucción "add" se utiliza para sumar dos operandos y almacenar el resultado en un registro de destino.

INSTRUCCION SUB: La instrucción "sub" es una instrucción de tipo R utilizada en la arquitectura de computadoras y en el lenguaje de programación ensamblador. La

instrucción "sub" se utiliza para restar dos operandos y almacenar el resultado en un registro de destino.

INSTRUCCION OR: La instrucción "or" es una instrucción de tipo R utilizada en la arquitectura de computadoras y en el lenguaje de programación ensamblador. La instrucción "or" se utiliza para realizar una operación lógica OR a nivel de bit entre dos operandos y almacenar el resultado en un registro de destino

INSTRUCCION AND: La instrucción "and" es una instrucción de tipo R utilizada en la arquitectura de computadoras y en el lenguaje de programación ensamblador. La instrucción "and" se utiliza para realizar una operación lógica AND a nivel de bit entre dos operandos y almacenar el resultado en un registro de destino.

INSTRUCCION SLT: La instrucción "slt" es una instrucción de tipo R utilizada en la arquitectura de computadoras y en el lenguaje de programación ensamblador. La instrucción "slt" se utiliza para comparar dos valores y almacenar el resultado de la comparación en un registro de destino.

INSTRUCCION NOP: La instrucción "nop" es una instrucción de tipo R utilizada en la arquitectura de computadoras y en el lenguaje de programación ensamblador. La instrucción "nop" (abreviatura de "no operation" o "sin operación" en español) se utiliza para indicar que no se realizará ninguna operación en la ejecución de una instrucción y se utiliza comúnmente como un espacio reservado para futuras expansiones del programa

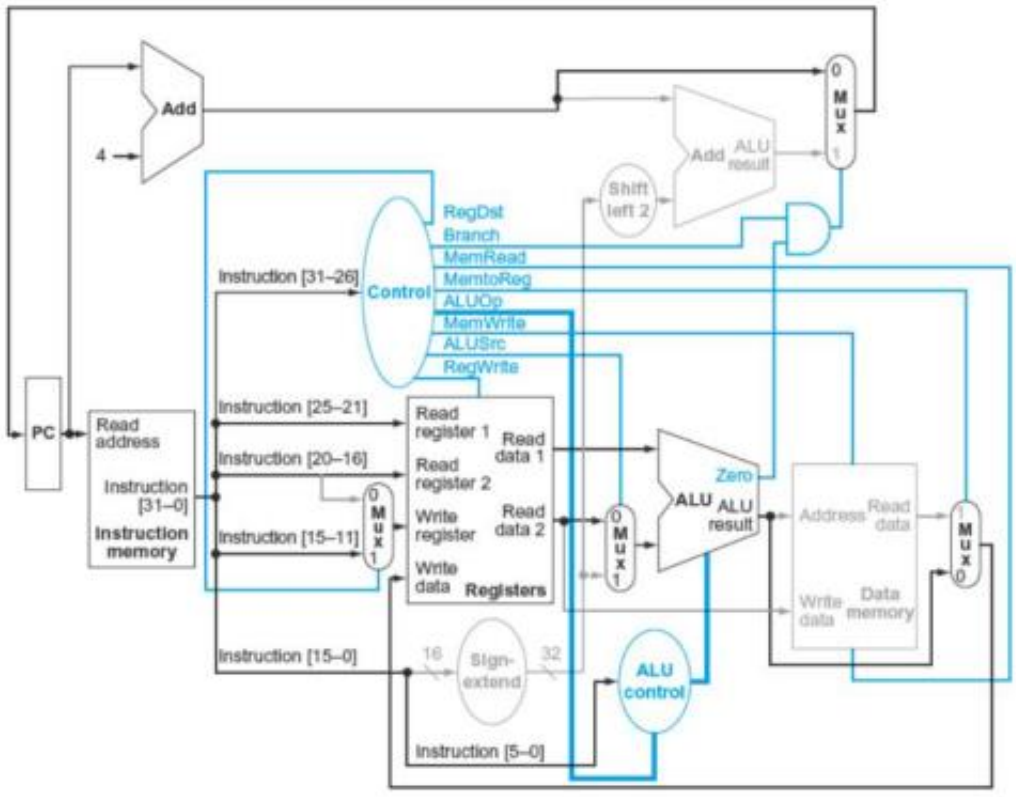
Personajes

LJUBISA BAJIC: Ljubisa Bajic es un ingeniero en computación que ha trabajado en el diseño de microprocesadores durante más de 30 años. Una de sus contribuciones más importantes en microarquitectura ha sido en la técnica de pipeline en paralelo, que permite que el procesador ejecute múltiples instrucciones simultáneamente. Bajic también ha trabajado en la mejora de la eficiencia energética de los procesadores y ha desarrollado técnicas para reducir el consumo de energía de los circuitos de procesamiento de datos.

JIM KELLER: Jim Keller es otro ingeniero en computación que ha sido fundamental en la evolución de los microprocesadores. Keller ha trabajado en el diseño de procesadores de alta gama durante más de 20 años y ha liderado equipos de diseño en empresas como AMD, Apple e Intel. Una de sus contribuciones más importantes ha sido en la arquitectura de los procesadores x86 de 64 bits, que se utilizan en la mayoría de los ordenadores personales modernos. Keller también ha trabajado en la optimización de la eficiencia energética de los procesadores, la mejora de la velocidad de reloj y la reducción del tamaño de los transistores en los circuitos integrados

Desarrollo

Se implementaron módulos correspondientes a la siguiente imagen:



En código se ve de la siguiente manera:

Adder:

```
1   `timescale 1ns/1ns
2   module Adder(
3       input [31:0] suma,
4       input [31:0] numprimero,
5       output reg [31:0] fuera
6   );
7
8   assign fuera = numprimero + suma;
9
10  endmodule
```

ALU:

```
module alu(
        input [31:0]op1,
        input [31:0]op2,
        input [3:0]sel,
        output reg [31:0]Resultado,
        output reg zeroflag
);



always @*
begin
        case (sel)
                4'b0000:
                        Resultado= op1&op2;
                4'b0001:
                        Resultado = op1|op2;
                4'b0010:
                        Resultado = op1+op2;
                4'b0110:
                        Resultado = op1-op2;
                4'b0111:
                        Resultado = op1<op2?1:0;
                4'b1100:
                        Resultado = ~(op1 | op2);
                default:
                        Resultado =0;
        endcase
        if(Resultado==0)
                zeroflag=1;
```

Alu control:

```
module ALU_Control(

        input [5:0]functionfield,
        input [1:0]Alu_op,
        output reg [3:0]operacion
);

always@*
begin
        case(Alu_op)
        2'b10:
                case(functionfield)
                6'b100000:
            operacion=4'b0010;
        6'b100010:
            operacion=4'b0110;
        6'b100101:
            operacion=4'b0001;
        6'b100100:
            operacion=4'b0000;
        6'b101010:
            operacion=4'b0111;
        endcase
    endcase
end

endmodule
```

Memoria:

```
`timescale 1ns/1ns
module Memoria(
        input Wen,//Senial para escribir
        input [31:0]Adress,//La direccion de la memoria
        input [31:0]DataW,//Escribir un valor nuevo en la memoria
        input Ren,//Senial para leer
        output reg [31:0]DataR//Valor para leer
);

reg [31:0] ROM[0:255];

always@*
begin
        if(Wen)
                ROM[Adress]=DataW;
        if(Ren)
                DataR=ROM[Adress];
        if(Wen&Ren)
                DataR=0;
end
endmodule
```

Mux 5 bits:

```
`timescale 1ns/1ns
module Mux5bits(
    input [4:0]Valoren1,
    input [4:0]Valoren0,
    input selector,
    output reg [4:0]WriteReg
);

always @*
    begin
        case(selector)
        1'b1:
            WriteReg=Valoren1;
        1'b0:
            WriteReg=Valoren0;
        endcase
    end
endmodule
```

Y así se creó modulo por modulo hasta que cada uno de los módulos se instancio en un TestBench.

Resultados:

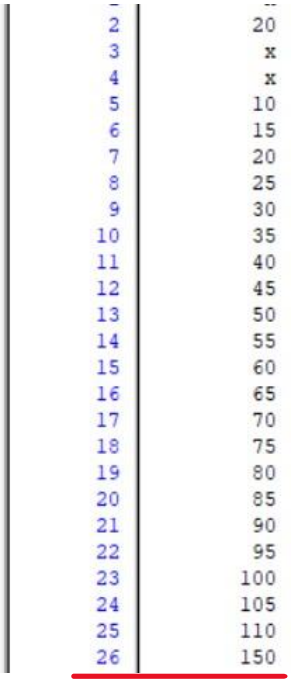
Una vez creado el TestBench e instanciado los módulos se corrió la simulación y lo primero fue cargar la memoria de instrucciones

ADD:

```
5   000000_10//ADD $26, $17, $19 70+80=150
6   001_10011
7   11010_000
8   00_100000
9
10
```

Diciéndole que vamos a sumar lo que este en la posición 17,19 y se lo vamos a agregar a la posición 26.

Corroboramos los resultados de revisando nuestro banco de registros:



Referencias:

MIPS Architecture. (n.d.). Retrieved from https://mips.com/

LjubisaBajic, Founder and CEO, Tenstorrent - Topio Networks. https://www.topionetworks.com/people/ljubisa-bajic-588b09eb2c537740e300002a

Cutress, I. (2021, May 27). An Interview with Tenstorrent: CEO Ljubisa Bajic and CTO Jim Keller. AnandTech.

https://www.anandtech.com/show/16709/an-interview-with-tenstorrent-ceo-ljubisa-bajic-and-cto-jim-keller