Universidad Autónoma de Zacatecas
Programa de Ingeniería en Robótica y Mecatrónica

Microcontroladores

Remberto Sandoval Aréchiga

Microprocesador

Alumno	Matricula
Karla Lorena Loera Benitez	35165366
Jonathan Adrián García Guerrero	35164932
Allan Uriel Valencia Esparza	35164771

Índice

Resumen

En el siguiente reporte se verá el diseño de un microprocesador de 8 bits en el cual para su creación es necesario contar con conocimientos anteriormente vistos de sistemas digitales y descripción del hardware. Se verán los diagramas de caja negra, código en verilog y la descripción de sus señales.

Diagrama caja negra

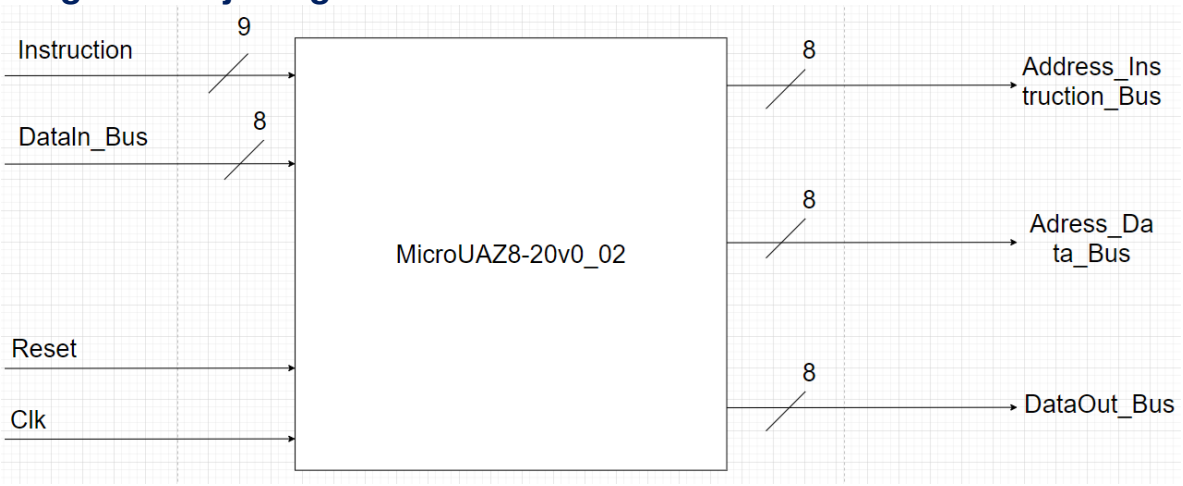

Ilustración 1 Diagrama caja negra microprocesador

Descripción de entradas y salidas

Señal	Numero de bits	Descripción
Instruction	9	Entrada de 9 bits que indica la instrucción a realizar extraída desde la memoria ROM hasta el microprocesador.
Data_InBus	8	Entrada de 8 bits que transmite la información de la memoria de datos al microprocesador.
Clk	8	Entrada de 8 bits referencia de tiempo. Debe ser una señal periódica con frecuencia de 100 MHz.
Reset	8	Entrada de 8 bits que establece el sistema en un estado inicial.
Adress_Instruction_Bus	8	Entrada de 8 bits que lleva las solicitudes de instrucciones de instrucciones a la memoria ROM.
Adress_Data_Bus	8	Entrada de 8 bits encargada de llevar las direcciones de los datos a la memoria de datos.
DataOut_Bus	8	Entrada de 8 bits que se encarga de llevar los datos que se desea escribir en la memoria de datos.

Set de instrucciones

Instruction	Arguments	Description	Comments
LOAD	RX,#NUM	Load #Num to register X	#Num is 3 bits [0,7]
LOAD	RX,[RY]	Load data at address [RY] from memory	RY and RX are 3 bits[0,7]
STORE	#NUM	Store #Num to [RX] address memory	#Num is 3 bits [0,7]
STORE	[RX],RY	Stores data at Register RY in [RX] memory address	RY and RX are 3 bits [0,7]
MOVE	RX,RY	Move data from register RY to RX	RY and RX are 3 bits [0,7]
MATH	RX,OP	DO MATH OPERATION WITH RX, AND STORES RESULT IN R0	OP: 0: R0=R0+RX 1: R0=R0-RX 2: R0= R0<<RX 3: R0= R0>>RY 4: R0=~RX 5: R0=R0&RX 6: R0 = R0|RX 7: R0=R0^RX
JUMP	[RX],COND	JUMP PC TO [RX] ADDRESS IF COND IS TRUE	COND: 0:NO CONDITION 1: SAVE PC IN R7 2: Z FLAG IS TRUE 3: Z FLAG IS FALSE 4: C FLAG IS TRUE 5: C FLAG IS FALSE 6: N FLAG IS TRUE 7: N FLAG IS FALSE
NOP		NO OPERATION	

Diagrama caja blanca

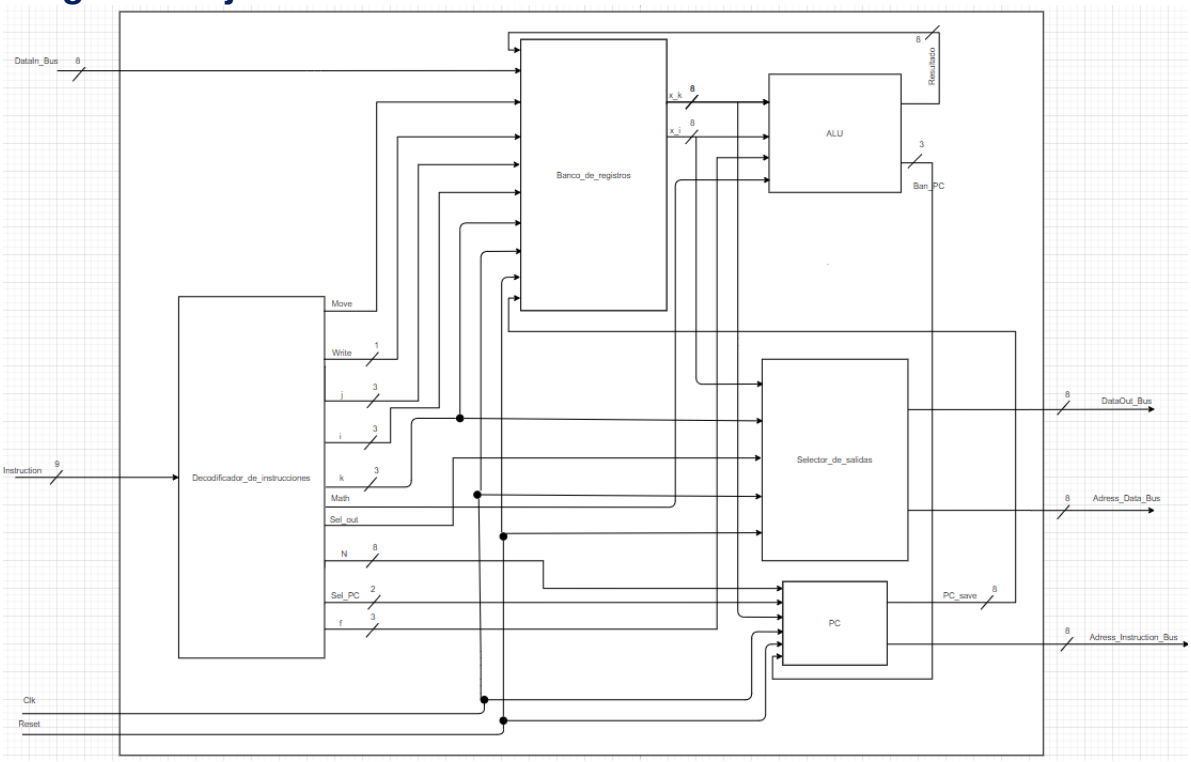

Ilustración 2 Diagrama caja blanca Microprocesador

Codigo verilog.

```
module Pana(
input [7:0] DataIn_Bus,
input [8:0] Instruction,
input Clk,
input Reset,
output [7:0] DataOut_Bus,
output [7:0] Adress_Data_Bus,
output [7:0] Adress_Instruction_Bus
    );

 wire Move;
 wire Math;
 wire Write;
 wire [2:0] k;
 wire [2:0] i;
 wire [2:0] j;
 wire Sel_out;
 wire [7:0] N;
 wire [1:0] Sel_PC;
 wire [2:0] f;
 wire [7:0] Resultado;
 wire [7:0] x_k;
 wire [7:0] x_i;
 wire [2:0] Ban_PC;
 wire [7:0] PC_save;

Decodificador_de_instrucciones M1(
```

```
.Instruction(Instruction),
.Move(Move),
.Write(Write),
.j(j),
.i(i),
.k(k),
.Math(Math),
.Sel_out(Sel_out),
.N(N),
.Sel_PC(Sel_PC),
.f(f)
);

Banco_de_registros M2(
.Resultado(Resultado),
.DataIn_Bus(DataIn_Bus),
.Move(Move),
.Write(Write),
.j(j),
.i(i),
.k(k),
.Reset(Reset),
.Clk(Clk),
.PC_save(PC_save),
.x_k(x_k),
.x_i(x_i)
);

ALU M3(
.Math(Math),
.x_k(x_k),
.x_i(x_i),
.f(f),
.Resultado(Resultado),
.Ban_PC(Ban_PC)
);

Selector_de_salidas M4(
.x_i(x_i),
.k(k),
.Sel_out(Sel_out),
.Clk(Clk),
.Reset(Reset),
.DataOut_Bus(DataOut_Bus),
.Adress_Data_Bus(Adress_Data_Bus)
);
PC M5(
.N(N),
.Sel_PC(Sel_PC),
.x_k(x_k),
.Clk(Clk),
.Reset(Reset),
.Ban_PC(Ban_PC),
.PC_save(PC_save),
.Adress_Instruction_Bus(Adress_Instruction_Bus)
);
Endmodule
```

Descripción de los bloques

PC

Diagrama caja negra.

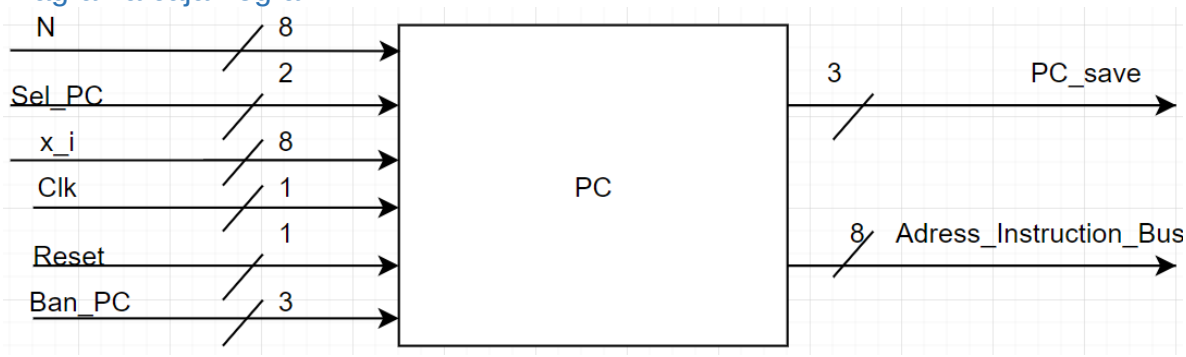

Ilustración 3 Diagrama caja negra PC

Descripción funcional.

Bloque encargado de indicar a la memoria qué instrucción debe proporcionarle al microprocesador.

Descripción de entradas y salidas

Señal	Dirección	Ancho (Bits)	Descripción
clk	Entrada	1	Señal de reloj
reset	Entrada	1	Señal útil para reiniciar
N	Entrada	8	Número de la instrucción a la que saltará
X_i	Entrada	8	Registro a revisar para los saltos positivos y negativos
Sel_PC	Entrada	2	Parte de la instrucción dedicada a seleccionar el tipo de salto de instrucción
Ban_PC	Entrada	3	Señal que sirve para entregarle las banderas al pc
PC_save	Salida	3	Guardar el pc en R7

Codigo verilog.

```
module PC(
input [2:0]Ban_PC,
input [7:0] N,
input [1:0] Sel_PC,
input [7:0] x_k,
input Clk,
input Reset,
output [7:0] PC_save,
```

```
output [7:0] Adress_Instruction_Bus
    );

reg[7:0] Cuenta;

always@(posedge Clk)
     if (!Reset)
        Cuenta = Cuenta + 1;
    else
        Cuenta = 0;

assign Adress_Instruction_Bus = Cuenta;
assign PC_save = Cuenta;
endmodule
```

Decodificador_de_instrucciones

Diagrama caja negra.

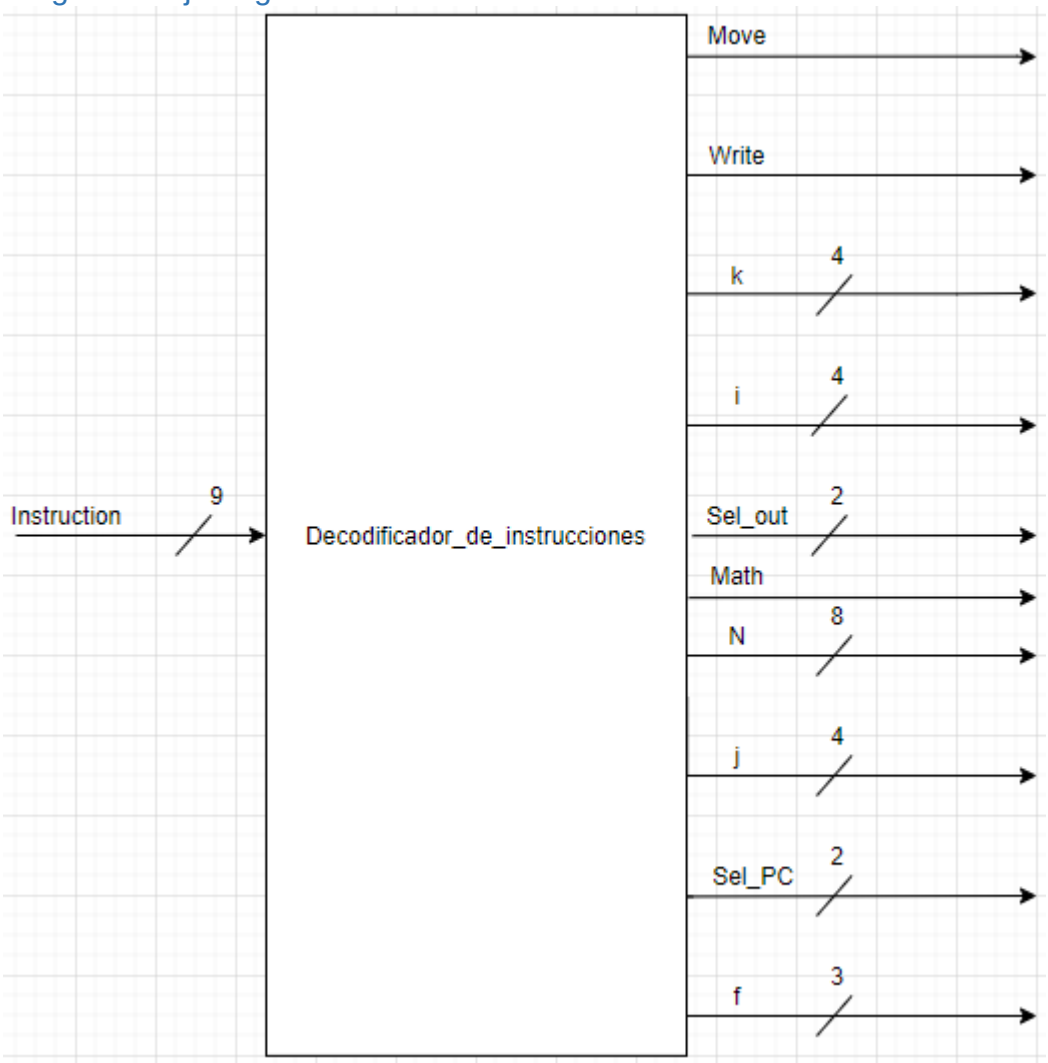

Ilustración 4 Diagrama caja negra Decodificador_de_instrucciones

Descripción funcional.

Este bloque es el encargado de tomar la instrucción de 9 bits proporcionada y en base a lo que esta contenga dar a los demás bloques las señales que son necesarias para actuar en consecuencia.

Descripción de entradas y salidas

Señal	Dirección	Ancho (Bits)	Descripción
Instruccion	Entrada	9	Instrucción que ejecutará el microprocesador
Move	Salida	1	Señal encargada de indicar si se van a mover los registros
Write	Salida	1	Señal que indica al banco de registros si debe escribir en el registro seleccionado o no
k	Salida	4	Señal usada para seleccionar registros internos
i	Salida	4	Señal usada para seleccionar el registro con el que se trabajará
Sel_out	Salida	2	Parte de la instrucción dedicada a seleccionar el valor que se pondrá en la salida
N	Salida	8	Número de la instrucción a la que se puede saltar
Sel_PC	Salida	2	Parte de la instrucción dedicada a seleccionar el valor que se pondrá en la salida
f	Salida	3	Señal usada para indicarle a la ALU si debe sumar o restar
Math	Salida	1	Señal utilizada para indicarle a la ALU cuando debe hacer matematicas

Código verilog.

```
module Decodificador_de_instrucciones(
input [8:0] Instruction,
output Move,
output Write,
output [2:0] j,
output [2:0] k,
output [2:0] i,
output Math,
output Sel_out,
output [7:0] N,
output [1:0] Sel_PC,
output [2:0] f
    );
reg Move_i;
```

```
reg Write_i;
reg [2:0] j_i;
reg [2:0] k_i;
reg [2:0] i_i;
reg Math_i;
reg [1:0] Sel_out_i;
reg [7:0] N_i;
reg [1:0] Sel_PC_i;
reg [2:0] f_i;

always @(Instruction)
begin

   case(Instruction[2:0])
//Load RX #NUM
3'b000: begin

//RX
    case(Instruction[5:3])
    //0
        3'b000: begin
   k_i = 0;
    end
    //1
        3'b001: begin
    k_i = 1;
    end
    //2
        3'b010: begin
     k_i = 2;
    end
    //3
        3'b011: begin
     k_i = 3;
    end
    //4
        3'b100: begin
     k_i = 4;
    end
    //5
        3'b101: begin
     k_i = 5;
    end
    //6
        3'b110: begin
     k_i = 6;
    end
    //7
        3'b111: begin
     k_i = 7;
    end
endcase

 //#NUM
    case(Instruction[8:6])
    //0
        3'b000: begin
```

```
    j_i = 0;
    end
    //1
        3'b001: begin
    j_i = 1;
    end
    //2
        3'b010: begin
    j_i = 2;
    end
    //3
        3'b011: begin
    j_i = 3;
    end
    //4
        3'b100: begin
    j_i = 4;
    end
    //5
        3'b101: begin
    j_i = 5;
    end
    //6
        3'b110: begin
    j_i = 6;
    end
    //7
        3'b111: begin
    j_i = 7;
    end
endcase

Move_i = 0;
Write_i = 0;
i_i = 0;
Math_i = 0;
Sel_out_i = 0;
N_i = 0;
Sel_PC_i = 0;
f_i = 0;


end
//Load RX [RY]
3'b001: begin

//RX
    case(Instruction[5:3])
    //R0+RX
        3'b000: begin
    k_i = 0;
    end
    //R0-RX
        3'b001: begin
    k_i = 1;
    end
    //R0<<RX
```

```
        3'b010: begin
    k_i = 2;
    end
    //R0>>RY
        3'b011: begin
    k_i = 3;
    end
    //~RX
        3'b100: begin
    k_i = 4;
    end
    //R0&RX
        3'b101: begin
    k_i = 5;
    end
    // R0|RX
        3'b110: begin
    k_i = 6;
    end
    //R0^RX
        3'b111: begin
    k_i = 7;
    end
endcase

 //#RY
    case(Instruction[8:6])
    //R0+RX
        3'b000: begin
    i_i = 0;
    end
    //R0-RX
        3'b001: begin
    i_i = 1;
    end
    //R0<<RX
        3'b010: begin
    i_i = 2;
    end
    //R0>>RY
        3'b011: begin
    i_i = 3;
    end
    //~RX
        3'b100: begin
    i_i = 4;
    end
    //R0&RX
        3'b101: begin
    i_i = 5;
    end
    // R0|RX
        3'b110: begin
    i_i = 6;
    end
    //R0^RX
        3'b111: begin
```

```
    i_i = 7;
    end
endcase

Move_i = 0;
Write_i = 0;
j_i = 0;
Math_i = 0;
Sel_out_i = 0;
N_i = 0;
Sel_PC_i = 0;
f_i = 0;

end
//Store #Num
3'b010: begin

//RX
    case(Instruction[5:3])
    //0
        3'b000: begin
    k_i = 0;
    end
    //1
        3'b001: begin
    k_i = 1;
    end
    //2
        3'b010: begin
    k_i = 2;
    end
    //3
        3'b011: begin
    k_i = 3;
    end
    //4
        3'b100: begin
    k_i = 4;
    end
    //5
        3'b101: begin
    k_i = 5;
    end
    //6
        3'b110: begin
    k_i = 6;
    end
    //7
        3'b111: begin
    k_i = 7;
    end
endcase

 //#NUM
    case(Instruction[8:6])
    //0
        3'b000: begin
```

```
    j_i = 0;
    end
    //1
        3'b001: begin
    j_i = 1;
    end
    //2
        3'b010: begin
    j_i = 2;
    end
    //3
        3'b011: begin
    j_i = 3;
    end
    //4
        3'b100: begin
    j_i = 4;
    end
    //5
        3'b101: begin
    j_i = 5;
    end
    //6
        3'b110: begin
    j_i = 6;
    end
    //7
        3'b111: begin
    j_i = 7;
    end
endcase

Move_i = 0;
Write_i = 1;
i_i = 0;
Math_i = 0;
Sel_out_i = 0;
N_i = 0;
Sel_PC_i = 0;
f_i = 0;


end
//Store RX [RY]
3'b011: begin
//RX
    case(Instruction[5:3])
    //0
        3'b000: begin
    k_i = 0;
    end
    //1
        3'b001: begin
    k_i = 1;
    end
    //2
        3'b010: begin
```

```
    k_i = 2;
    end
    //3
        3'b011: begin
    k_i = 3;
    end
    //4
        3'b100: begin
    k_i = 4;
    end
    //5
        3'b101: begin
    k_i = 5;
    end
    //6
        3'b110: begin
    k_i = 6;
    end
    //7
        3'b111: begin
    k_i = 7;
    end
endcase

 //RY
    case(Instruction[8:6])
    //0
        3'b000: begin
    i_i = 0;
    end
    //1
        3'b001: begin
    i_i = 1;
    end
    //2
        3'b010: begin
    i_i = 2;
    end
    //3
        3'b011: begin
    i_i = 3;
    end
    //4
        3'b100: begin
    i_i = 4;
    end
    //5
        3'b101: begin
    i_i = 5;
    end
    //6
        3'b110: begin
    i_i = 6;
    end
    //7
        3'b111: begin
    i_i = 7;
```

```
    end
endcase

Move_i = 0;
Write_i = 1;
j_i = 0;
Math_i = 0;
Sel_out_i = 0;
N_i = 0;
Sel_PC_i = 0;
f_i = 0;


end
//Move RX RY
3'b100: begin
//RX
    case(Instruction[5:3])
    //0
        3'b000: begin
    k_i = 0;
    end
    //1
        3'b001: begin
    k_i = 1;
    end
    //2
        3'b010: begin
    k_i = 2;
    end
    //3
        3'b011: begin
    k_i = 3;
    end
    //4
        3'b100: begin
    k_i = 4;
    end
    //5
        3'b101: begin
    k_i = 5;
    end
    //6
        3'b110: begin
    k_i = 6;
    end
    //7
        3'b111: begin
    k_i = 7;
    end
endcase

 //RY
    case(Instruction[8:6])
    //R0+RX
        3'b000: begin
    i_i = 0;
```

```
    end
    //R0-RX
        3'b001: begin
    i_i = 1;
    end
    //R0<<RX
        3'b010: begin
    i_i = 2;
    end
    //R0>>RY
        3'b011: begin
    i_i = 3;
    end
    //~RX
        3'b100: begin
    i_i = 4;
    end
    //R0&RX
        3'b101: begin
    i_i = 5;
    end
    // R0|RX
        3'b110: begin
    i_i = 6;
    end
    //R0^RX
        3'b111: begin
    i_i = 7;
    end
endcase

Move_i = 1;
Write_i = 0;
j_i = 0;
Math_i = 0;
Sel_out_i = 0;
N_i = 0;
Sel_PC_i = 0;
f_i = 0;


end

//Math RX OP
3'b101: begin

//RX
    case(Instruction[5:3])
    //0
        3'b000: begin
    k_i = 0;
    end
    //1
        3'b001: begin
    k_i = 1;
    end
    //2
```

```
            3'b010: begin
        k_i = 2;
        end
        //3
            3'b011: begin
        k_i = 3;
        end
        //4
            3'b100: begin
        k_i = 4;
        end
        //5
            3'b101: begin
        k_i = 5;
        end
        //6
            3'b110: begin
        k_i = 6;
        end
        //7
            3'b111: begin
        k_i = 7;
        end
    endcase

    //OP
        case(Instruction[8:6])
        //R0+RX
            3'b000: begin
        f_i = 0;
        end
        //R0-RX
            3'b001: begin
        f_i = 1;
        end
        //R0<<RX
            3'b010: begin
        f_i = 2;
        end
        //R0>>RY
            3'b011: begin
        f_i = 3;
        end
        //~RX
            3'b100: begin
        f_i = 4;
        end
        //R0&RX
            3'b101: begin
        f_i = 5;
        end
        // R0|RX
            3'b110: begin
        f_i = 6;
        end
        //R0^RX
            3'b111: begin
```

```
    f_i = 7;
    end
    endcase


Move_i = 0;
Write_i = 0;
j_i = 0;
i_i = 0;
Math_i = 1;
Sel_out_i = 0;
N_i = 0;
Sel_PC_i = 0;


end


//Jump
3'b110: begin

//RX
    case(Instruction[5:3])
    //0
        3'b000: begin
    k_i = 0;
    end
    //1
        3'b001: begin
    k_i = 1;
    end
    //2
        3'b010: begin
    k_i = 2;
    end
    //3
        3'b011: begin
    k_i = 3;
    end
    //4
        3'b100: begin
    k_i = 4;
    end
    //5
        3'b101: begin
    k_i = 5;
    end
    //6
        3'b110: begin
    k_i = 6;
    end
    //7
        3'b111: begin
    k_i = 7;
    end
endcase
```

```
//#Cond
    case(Instruction[8:6])
    //0:NO CONDITION
        3'b000: begin
    f_i = 0;
    end
    //1: NO CONDITION SAVE PC IN R7
        3'b001: begin
    f_i = 1;
    end
    //2:Z FLAG IS TRUE
        3'b010: begin
    f_i = 2;
    end
    //3:Z FLAG IS FALSE
        3'b011: begin
    f_i = 3;
    end
    //4: C FLAG IS TRUE
        3'b100: begin
    f_i = 4;
    end
    //5: C FLAG IS FALSE
        3'b101: begin
    f_i = 5;
    end
    //6: N FLAG IS TRUE
        3'b110: begin
    f_i = 6;
    end
    //7: N FLAG IS FALSE
        3'b111: begin
    f_i = 7;
    end
endcase

Move_i = 0;
Write_i = 0;
j_i = 0;
i_i = 0;
Math_i = 0;
Sel_out_i = 0;
N_i = 0;
Sel_PC_i = 0;
f_i = 0;

end

//Nop
3'b111: begin

Move_i = 0;
Write_i = 0;
Math_i = 0;


end
```

```
endcase
end

assign Move = Move_i;
assign Write = Write_i;
assign j = j_i;
assign k = k_i;
assign i = i_i;
assign Math = Math_i;
assign Sel_out = Sel_out_i;
assign N = N_i;
assign Sel_PC = Sel_PC_i;
assign f = f_i;

endmodule
```

Banco_de_registros

Diagrama caja negra.

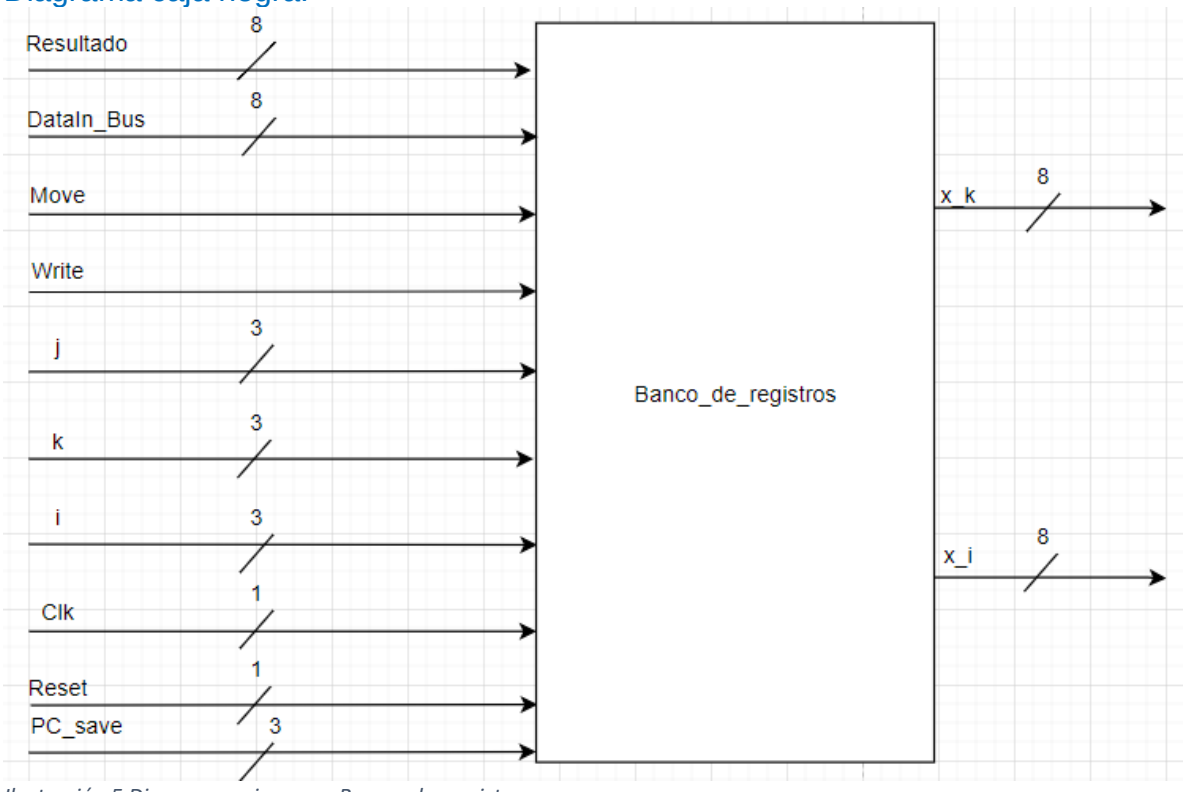

Ilustración 5 Diagrama caja negra Banco_de_registros

Descripción funcional.

Este módulo es el encargado de guardar los valores numéricos y/o presentarlos a su salida (según sea necesario).

Descripción de entradas y salidas

Señal	Dirección	Ancho (Bits)	Descripción
Clk	Entrada	1	Señal de reloj
Reset	Entrada	1	Señal de reinicio
Move	Entrada	1	Señal que indica al banco de registros donde debe mover los registros
Write	Entrada	1	Señal que indica al banco de registros si debe escribir en el registro seleccionado o no
k	Entrada	4	Señal usada para seleccionar registros internos
i	Entrada	4	Señal usada para seleccionar el registro con el que se trabajará
Sel_out	Entrada	2	Parte de la instrucción dedicada a seleccionar el valor que se pondrá en la salida
N	Entrada	8	Número de la instrucción a la que se puede saltar
Sel_PC	Entrada	2	
f	Entrada	3	Señal usada para indicarle a la ALU si debe sumar o restar
X_i	Salida	8	Señal que será utilizada en La ALU como operador A
X_j	Salida	8	Señal que será utilizada en la ALU como operador B
PC_save	Entrada	8	Guardar el pc en R7
j	Entrada	3	Valores de #num

Código verilog.

```
module Banco_de_registros(
input [7:0] Resultado,
input Move,
input [7:0] DataIn_Bus,
input Write,
input [7:0] PC_save,
input [2:0] k,
input [2:0] i,
input [2:0] j,
input Clk,
input Reset,
output [7:0] x_k,
output [7:0] x_i
    );

reg[7:0] R[0:7];
assign x_k=R[k];
assign x_i=R[i];

always@(posedge Clk)
```

```
begin
if(Reset)
begin
R[0]<=0;
R[1]<=0;
R[2]<=0;
R[3]<=0;
R[4]<=0;
R[5]<=0;
R[6]<=0;
R[7]<=0;
end

else
begin

if(Write)
begin
R[k]<= j;
R[i]<= R[i];
end

else

R[7] <= PC_save;
R[0] <= Resultado;
 end

if(!Move)
begin

end
else
R[i] <= R [k];
end

endmodule
```

ALU

Diagrama caja negra.

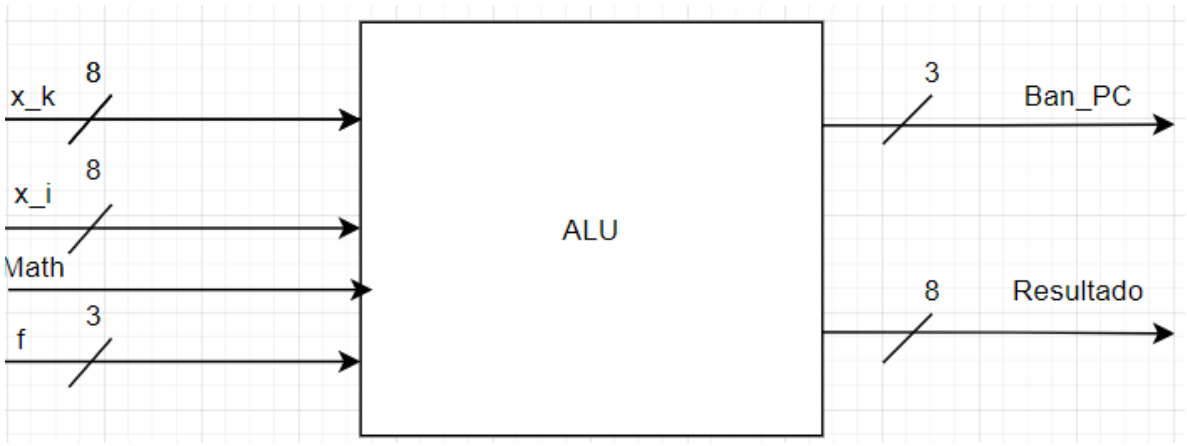

Ilustración 6 Diagrama caja negra ALU

Descripción funcional.

Bloque encargado de hacer las operaciones matemáticas (sumas y restas), en el cual también se encuentran las banderas.

Descripción de entradas y salidas

Señal	Dirección	Ancho (Bits)	Descripción
OpA	Entrada	8	Operador A
OpB	Entrada	8	Operador B
f	Entrada	1	Señal que indica a la ALU si debe sumar o restar los operadores
resultado	Salida	8	Resultado de la operación
Ban_PC	Salida	3	Señal que sirve para entregarle las banderas al PC.
Math	Entrada	1	Señal utilizada para indicarle a la ALU cuando debe hacer matemáticas

Código verilog.

```
module ALU(
input [7:0] x_k,
input [7:0] x_i,
input [2:0] f,
input Math,
output [7:0] Resultado,
output [2:0] Ban_PC
    );

reg [7:0] R0 = 0;

always @(*)
begin
case(Math)
1'b0:
R0=x_k;

1'b1:

case(f)
 0: R0 = (x_k + x_i) ;
 1: R0 = (x_k - x_i) ;
 2: R0 = (x_k << x_i) ;
 3: R0 = (x_k >> x_i) ;
 4: R0 =~ x_i;
 5: R0 = (x_k & x_i) ;
 6: R0 = (x_k | x_i) ;
 7: R0 = (x_k ^ x_i) ;
endcase

endcase
end
```

```
assign Resultado = R0;
assign Ban_PC[0] = &(~R0);      //Cero
assign Ban_PC[1] = R0[8];     //Acarreo
assign Ban_PC[2] = R0[7];     //Negativo

endmodule
```

Selector_de_salidas

Diagrama caja negra.

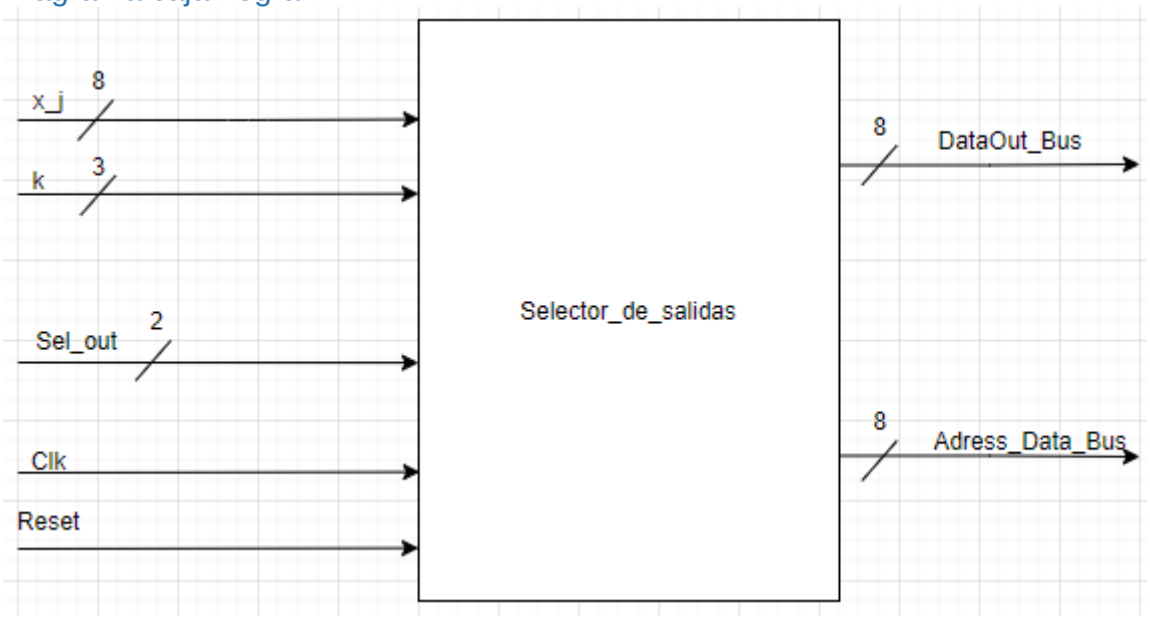

Ilustración 7 Diagrama caja negra Selector_de_salidas

Descripción funcional.

Este bloque es el encargado de seleccionar el valor que se colocará en la salida y seleccionar en que salida se colocará el mismo.

Descripción de entradas y salidas

Señal	Dirección	Ancho (Bits)	Descripción
clk	Entrada	6	Señal de reloj
reset	Entrada	1	Señal útil para reiniciar
X_j	Entrada	4	Señal usada para poner el valor de un registro en una salida
k	Entrada	2	Señal usada para mandar la dirección del dato
Sel_out	Entrada	2	Parte de la instrucción dedicada a seleccionar el valor que se pondrá en la salida
DataOut_Bus	Salida	8	Número de la instrucción a la que se puede saltar
Adress_Data_Bus	Salida	8	Salida de dirección de datos

Código verilog.

```
module Selector_de_salidas(
input [7:0] x_i,
input [2:0] k,
input Sel_out,
input Clk,
input Reset,
output [7:0] DataOut_Bus,
output [7:0] Adress_Data_Bus
    );

reg [7:0] DataOut_Bus_i;
reg [7:0] Adress_Data_Bus_i;
reg [7:0] aux;

always@(posedge Clk)
begin
 if (Reset)
 begin
 aux<=0;
 end
 else
 case(Sel_out)

 0: aux<=x_i;
 1: aux<=k;

 default: aux<=0;

 endcase

end
always@(posedge Clk)
begin
 if (Reset)
 begin
 DataOut_Bus_i<=0;
 Adress_Data_Bus_i<=0;
 end
 else

begin
 DataOut_Bus_i<=aux;
 Adress_Data_Bus_i<= k;
 end

end

assign DataOut_Bus = DataOut_Bus_i;
assign Adress_Data_Bus = Adress_Data_Bus_i;

endmodule
```

Simulación

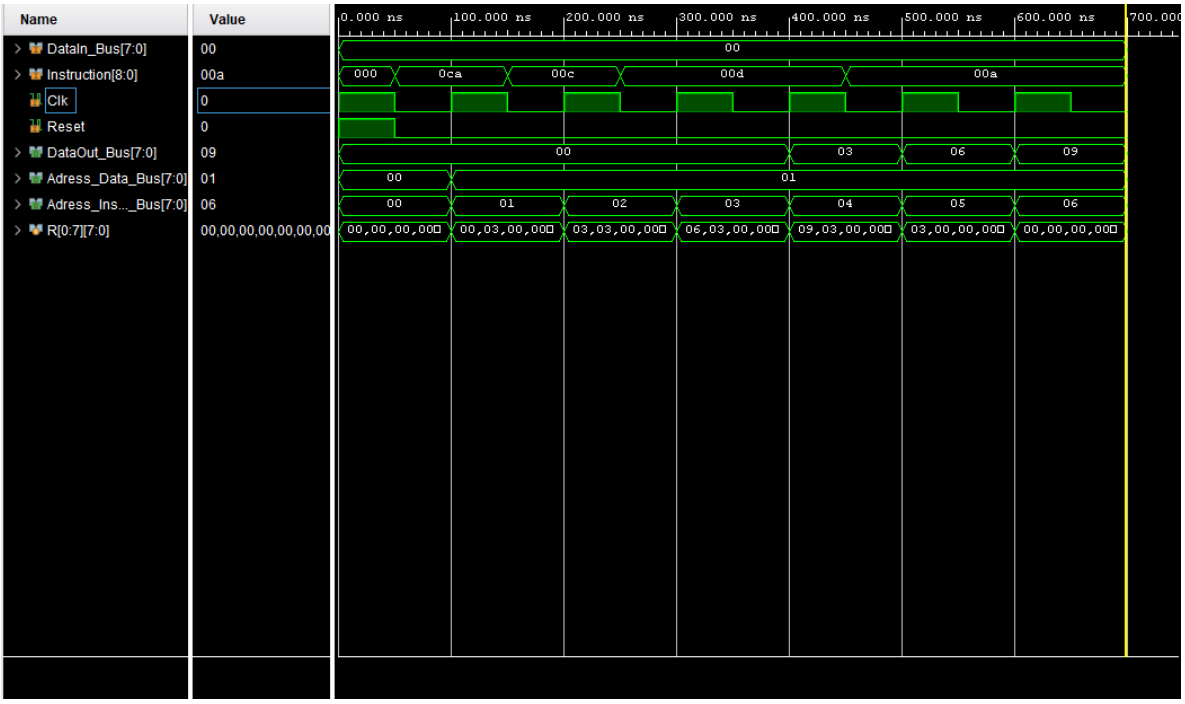

Ilustración 8 Simulación del microprocesador

Descripción de la simulación.

En la siguiente simulación se puede observar que se guardó un 3 en el registro 1, el cual se movió al registro 0 para, posteriormente realizar sumas con estos números, se sumó 3 + 3 después se le sumo al resultado 3 y después se le sumo al resultado 3, dándonos como salida 9, el cual es el resultado de multiplicar 3 por 3.