

CINVESTAV

Alumno: Efraín Arrambide Barrón

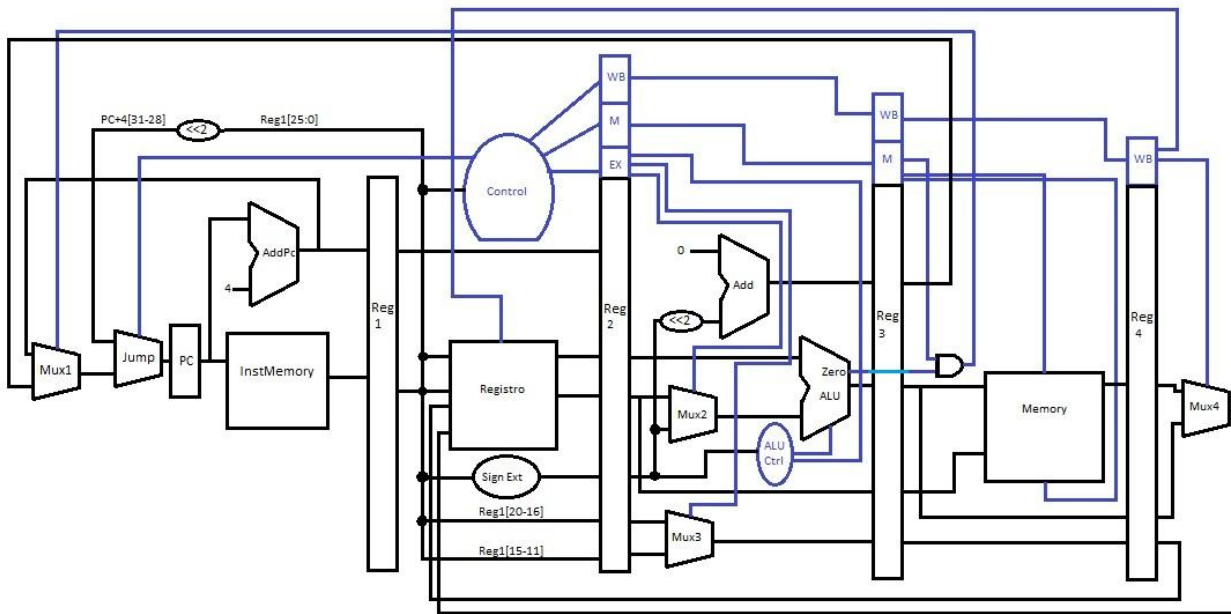
Tarea 4: Pipeline

Docente: Doc. Mariano Aguirre

PADTS 19



a) Diagrama a bloques de la arquitectura implementada, realizado por el alumno.



ADD RD, RS, RT

RD = RS + RT

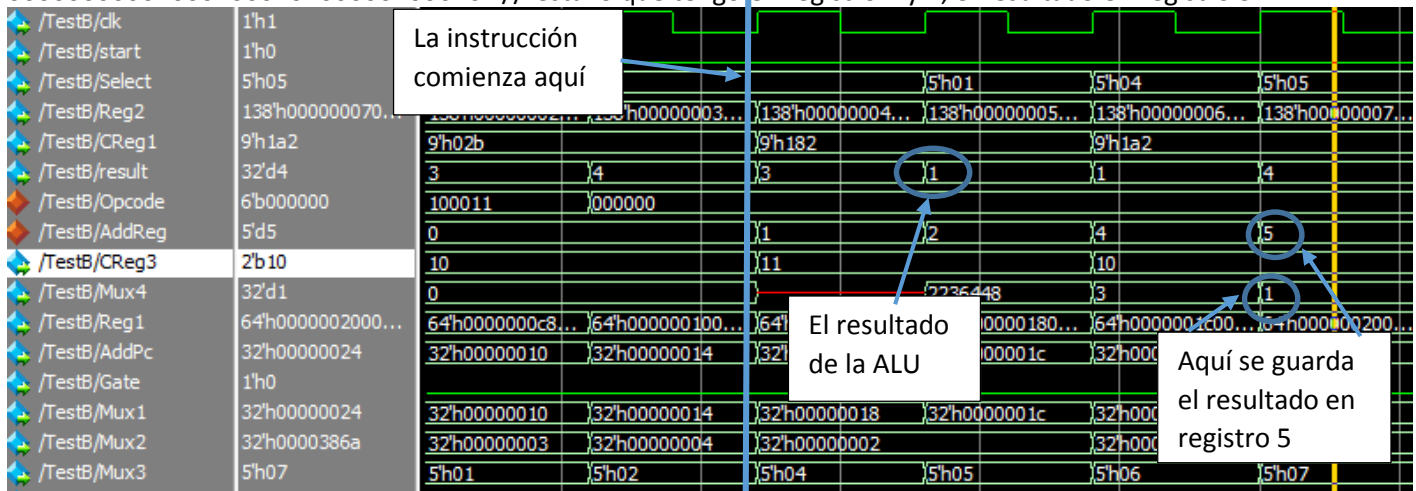
0000000000100010001000000100000 //suma lo que tengo en registro 1 y 2, el resultado en registro 4



SUB RD, RS, RT

RD = RS - RT

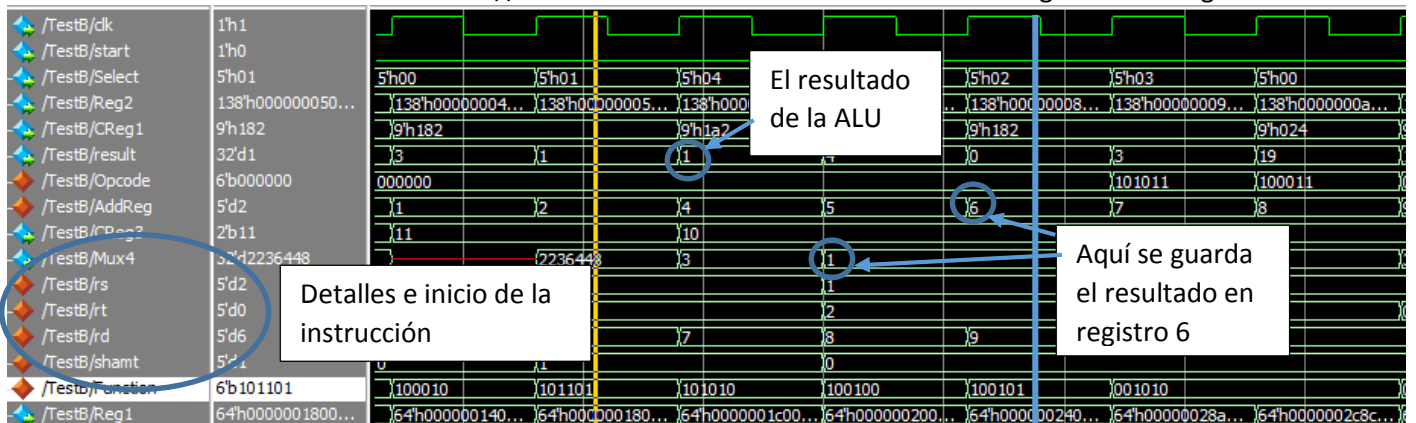
00000000001000100100000100010 //resta lo que tengo en registro 1 y 2, el resultado en registro 5



SRL RD, RS, SHIFT5

RD = RS >> SHIFT5

00000000010000000011000001101101 // corrimiento hacia la derecha recorre 2>>1 guardar en registro 6



SLL RD, RS, SHIFT5

RD = RS << SHIFT5

00000000010000000011100001101010 // corrimiento hacia la izquierda SLL RD, RS, SHIFT5 2<<1 guardar en registro 7



AND RD, RS, RT

RD = RS & RT

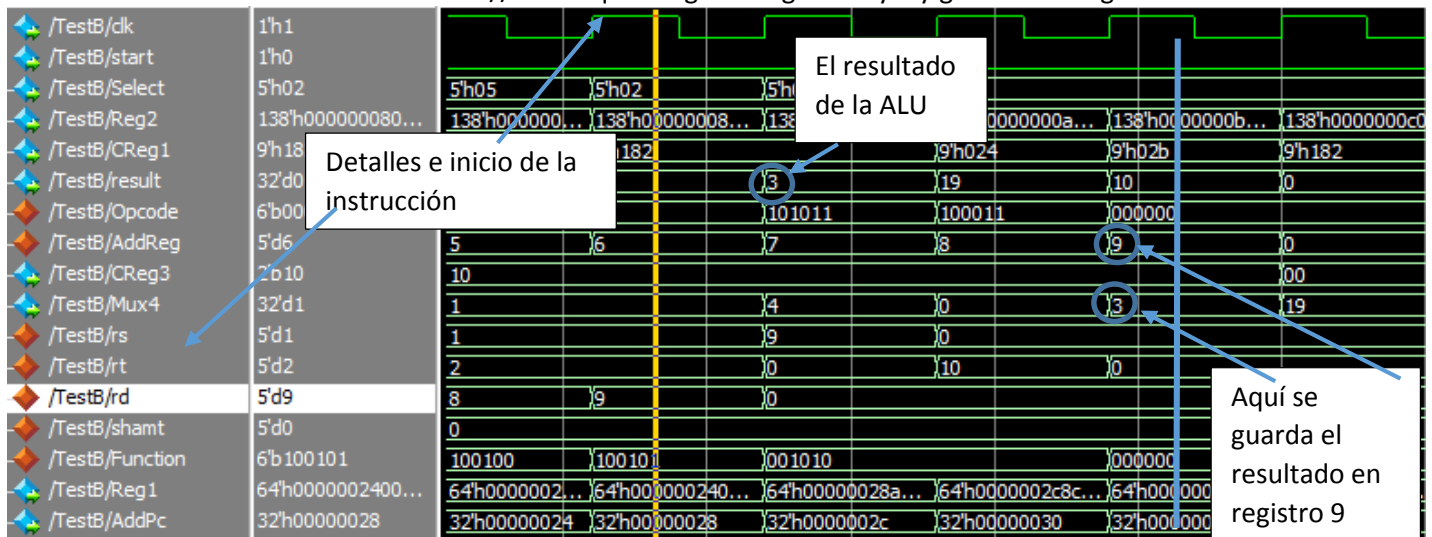
00000000001000100100000000100100 // and lo que tengo en registro 1 y 2 y guardar en registro 8



XOR RD, RS, RT

RD = RS ^ RT

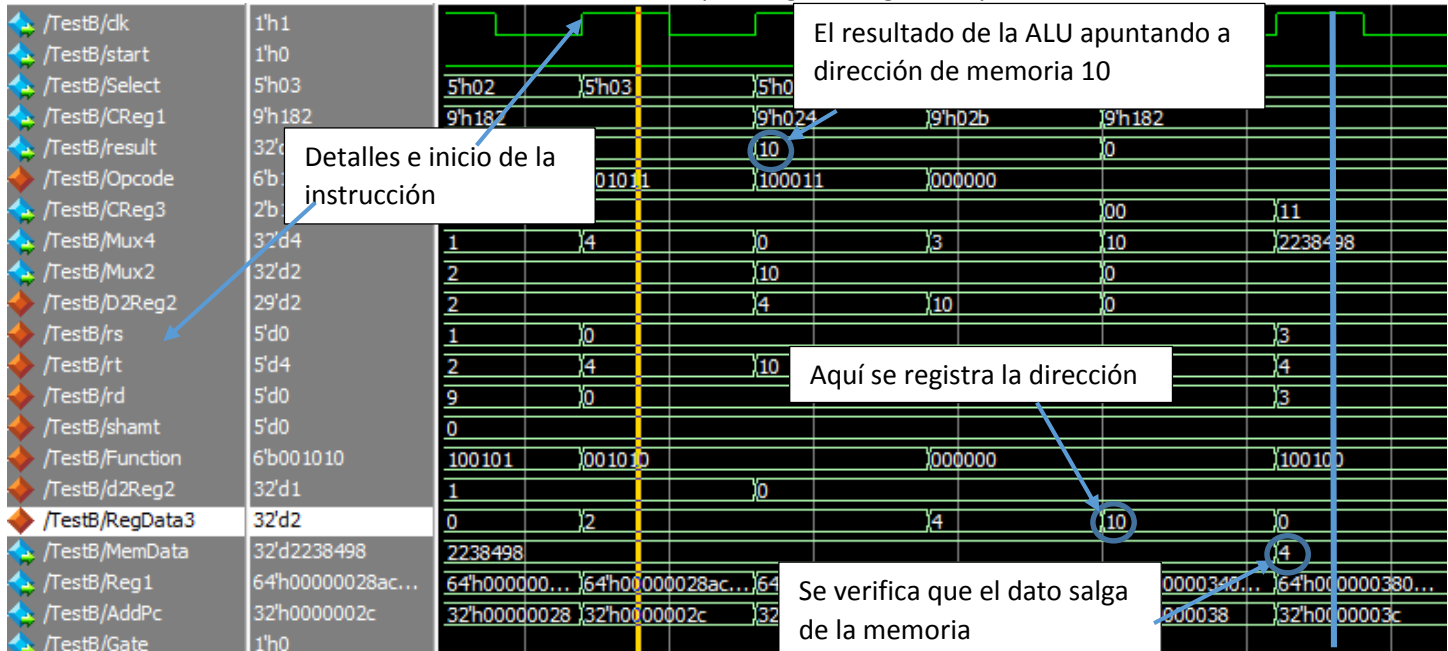
00000000001000100100100000100101 // xor lo que tengo en registro 1 y 2 y guardar en registro 9



SW RS, OFF16(RT)

$MEM32(RT + OFF16_{\pm}) = RS$

1010110000000100000000000001010 // almacenar lo que tengo en registro 4 y colocarlo mem[10]

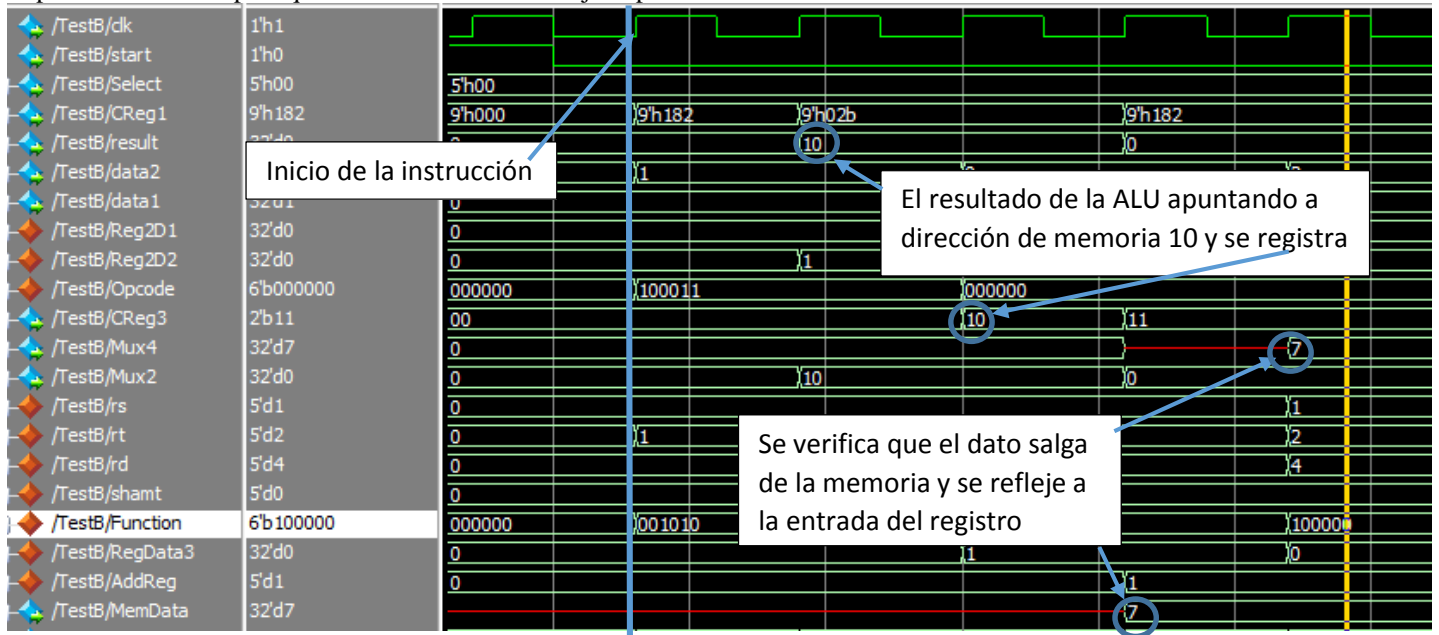


LW RD, OFF16(RS)

$RD = MEM32(RS + OFF16_{\pm})$

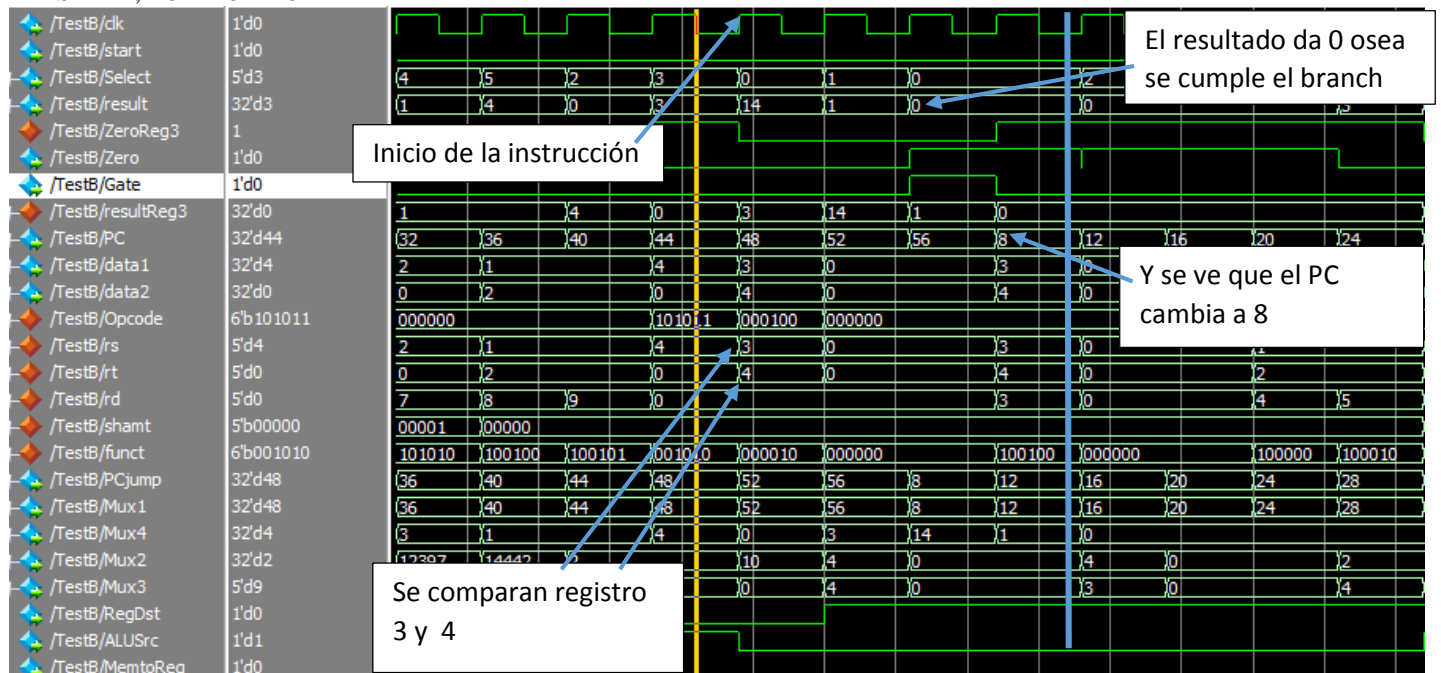
1000110000000001000000000001010 // carga lo que tengo en localidad de memoria 10 al registro 1.

Repetí la instrucción para que el resultado se vea reflejado por la lectura síncrona.



BEQ RS, RT, OFF18

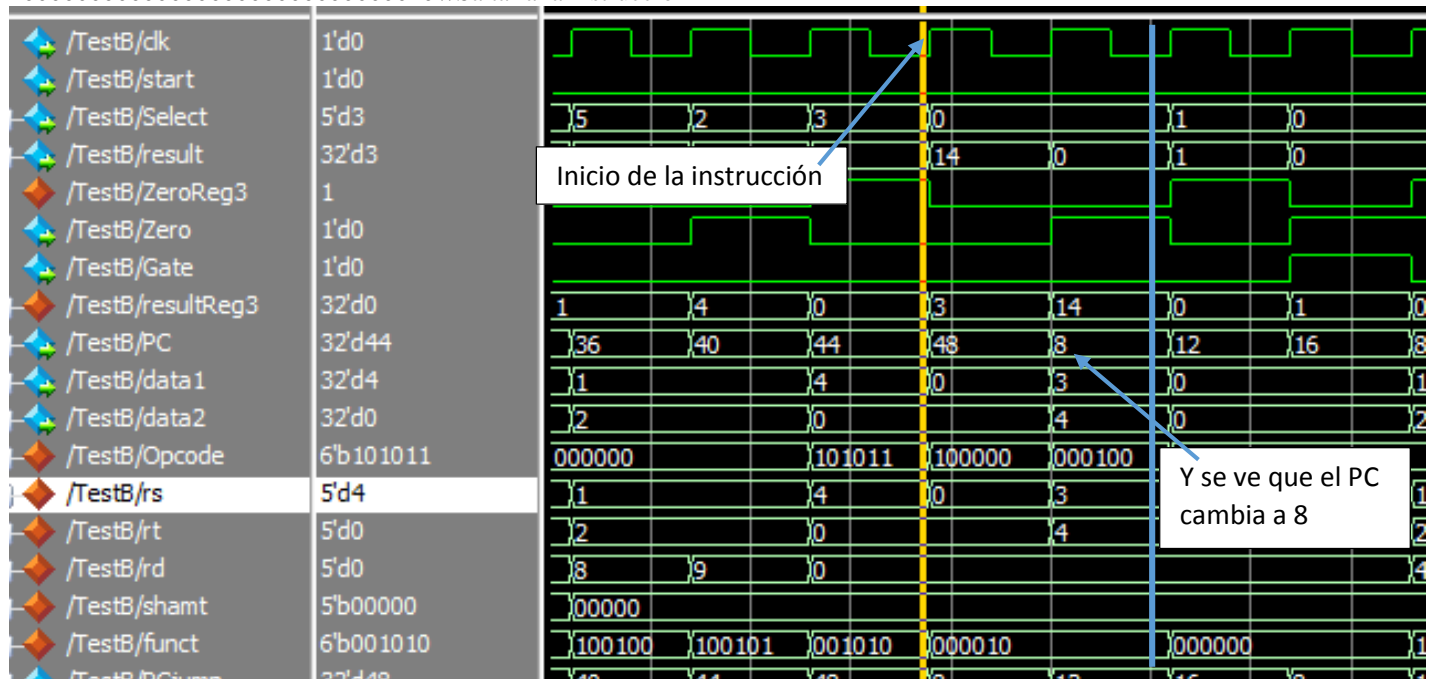
IF RS = RT, PC += OFF18±



J ADDR28

PC = PC[31: 28] :: ADDR28

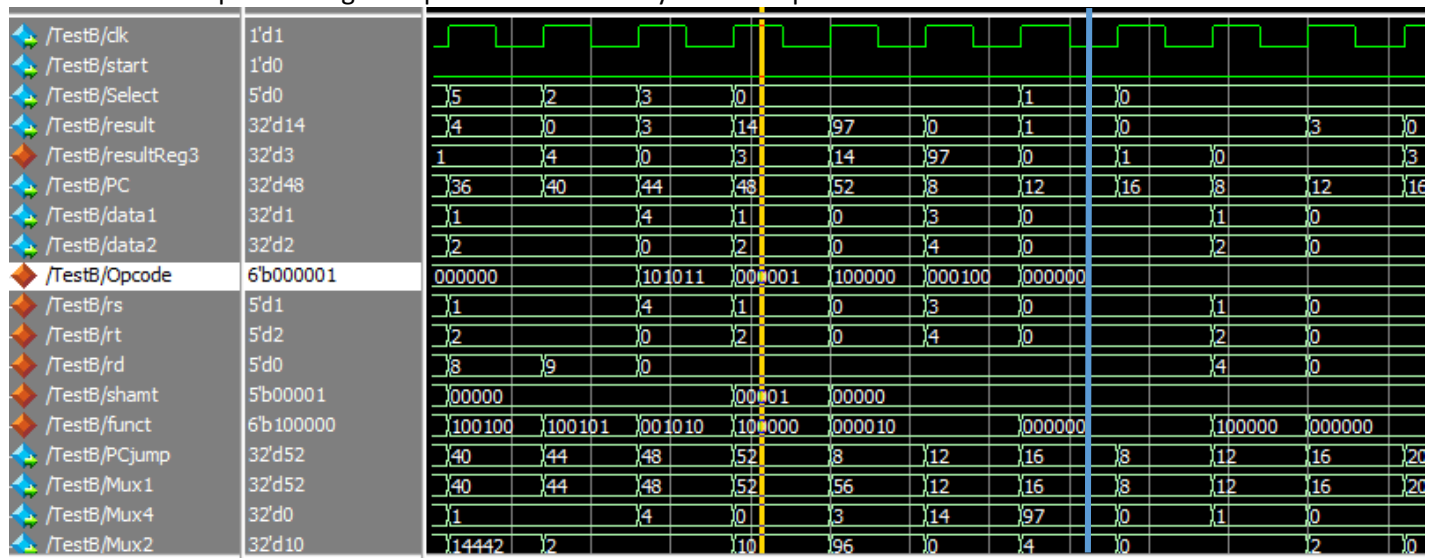
10000000000000000000000000000010 //Saltar a la instrucción 2



ADDI RD, RS, CONST16

$RD = RS + CONST16_{\pm}$

00000100001000100000000001100000 //suma lo que hay en registro 1 y offset(96), el resultado colócalo en registro 2
Si se verifica el primer diagrama podrá entender muy bien este paso



Slow 1200mV 0C Model Fmax Summary

| | Fmax | Restricted Fmax | Clock Name | Note |
|---|------------|-----------------|------------|------|
| 1 | 114.44 MHz | 114.44 MHz | clk | |

Flow Summary

| | |
|------------------------------------|--|
| Flow Status | Successful - Sat Oct 25 02:30:10 2014 |
| Quartus II 64-Bit Version | 13.0.0 Build 156 04/24/2013 SJ Web Edition |
| Revision Name | Pipeline |
| Top-level Entity Name | Pipeline |
| Family | Cyclone IV E |
| Device | EP4CE115F29C7 |
| Timing Models | Final |
| Total logic elements | 1,122 / 114,480 (< 1 %) |
| Total combinational functions | 984 / 114,480 (< 1 %) |
| Dedicated logic registers | 544 / 114,480 (< 1 %) |
| Total registers | 544 |
| Total pins | 292 / 529 (55 %) |
| Total virtual pins | 0 |
| Total memory bits | 2,048 / 3,981,312 (< 1 %) |
| Embedded Multiplier 9-bit elements | 0 / 532 (0 %) |
| Total PLLs | 0 / 4 (0 %) |

```
module Mux1(input [106:0]Reg3,input [31:0]AddPc, input Gate, output [31:0]Mux1);
```

```
assign Mux1=(Gate)?Reg3[101:70]:AddPc;
```

```
endmodule
```

```
module PC(input [31:0]PCjump, input clk,start, output logic [31:0]PC);
```

```
always@(posedge clk or posedge start)
```

```
    if(start)
```

```
        PC <= 0;
```

```
    else
```

```
        PC <= PCjump;
```

```
Endmodule
```

```
module AddPc(input[31:0]PC,output logic [31:0]AddPc);
```

```
always@(PC)
```

```
    AddPc=PC+4;
```

```
Endmodule
```

```
module InstMem (output [31:0]Instruction,input [31:0] PC);
```

```
reg [31:0] InstMemory [0:31];
```

```
    initial $readmemb("Instruction.dat",InstMemory);
```

```
    assign Instruction = InstMemory [PC[6:2]];
```

```
endmodule
```

```
module Reg1(input [31:0]AddPc,Instruction, input clk,start, output logic [63:0]Reg1);
```

```
always@(posedge clk or posedge start)
```

```
    if(start)Reg1<=0;
```

```
    else      Reg1<={AddPc,Instruction};
```

```
endmodule
```

```
module Registro (output [31:0]data1,data2,input [31:0]Mux4,input [63:0]Reg1,input [70:0]Reg4,input clk,input [31:0]R30In,output [31:0]R31Out);
```

```
reg [31:0] Registro [0:31];
```

```
initial $readmemb("Registro.dat",Registro);
```

```
    assign R31Out = Registro[31];
```

```
    assign data1 = Registro[Reg1[25:21]];
```

```
    assign data2 = Registro[Reg1[20:16]];
```

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

```
    if ((Reg4[70]==1) && (Reg4!=0) && (Reg4!=30))
```

```
        Registro[Reg4] <= Mux4;
```

```
    else if(Reg4==0)
```

```
        Registro[Reg4] <= 0;
```

```
    else if(Reg4==30)
```

```

        Registro[Reg4] <= R30In;
end
endmodule

module Reg2(input [31:0]data1,data2,input [63:0]Reg1, input
clk,start,RegDst,ALUOp1,ALUOp0,ALUSrc,Branch,MemRead,MemWrite,RegWrite,MemtoReg, output logic [146:0]Reg2);

    always@(posedge clk or posedge start)

        if(start)Reg2<=0;
        else
            Reg2<={RegDst,ALUOp1,ALUOp0,ALUSrc,Branch,MemRead,MemWrite,RegWrite,MemtoReg,Reg1[63:32],data1,
data2,{16{Reg1[15]}},Reg1[15:0]},Reg1[20:11]};

endmodule

module Add(input [146:0]Reg2, output logic [31:0]Add);
    always@(*)begin

        Add = Reg2[41:10]<<2;
    end

endmodule

module Mux2(input [146:0]Reg2,output [31:0]Mux2);

    assign Mux2=(Reg2[143])?Reg2[41:10]:Reg2[73:42];

endmodule

module Mux3(input [146:0]Reg2, output [4:0]Mux3);

    assign Mux3=(Reg2[146])?Reg2[4:0]:Reg2[9:5];

endmodule

module ALU (input [4:0]Select,input [31:0]Mux2,input [146:0]Reg2, output logic [31:0]result,output Zero );

    assign Zero=(result==0)?1'b1:1'b0;

    always@(Select,Reg2,Mux2)
        begin
            case (Select)
                0:      result = Reg2[105:74]+Mux2;
                1:      result = Mux2-Reg2[105:74];
                2:      result = Reg2[105:74]&Mux2;
                3:      result = Reg2[105:74]^Mux2;
                4:      result = Reg2[105:74]>>Mux2[10:6];
                5:      result = Reg2[105:74]<<Mux2[10:6];
                default: result = Reg2[105:74]+Mux2;
            endcase
        end

endmodule:ALU

```

```

module ALUControl (input [146:0]Reg2,output logic[4:0]Select);
logic [1:0]ALUOp;
assign ALUOp={Reg2[145],Reg2[144]};

always@(ALUOp or Reg2[15:10])
begin
    if ( ALUOp == 2'b10)
        begin
            case(Reg2[15:10])
                6'b100000: Select = 0;//Suma ALU
                6'b100010: Select = 1;//Resta ALU
                6'b100100: Select = 2;//AND
                6'b100101: Select = 3;//XOR
                6'b101101: Select = 4;//Corrimiento Dere
                6'b101010: Select = 5;//Corrimiento Izq
                default: Select = 0;
            endcase
        end
    else if (ALUOp == 2'b01)
        Select = 1;
    else
        Select = 0;
end

endmodule

module Reg3(input [31:0]result,Add,input [4:0]Mux3, input Zero, input [146:0]Reg2, input clk,start,
output logic [106:0]Reg3);

always@(posedge clk or posedge start)

    if(start)Reg3<=0;
    else
        Reg3<={Reg2[142:138],Add,Zero,result,Reg2[73:42],Mux3};

endmodule

module Memory(input clk, output logic [31:0]MemData, input [106:0]Reg3);
reg [31:0] mem [0:31];
initial $readmemb("memoria.dat",mem);

always@(posedge clk)
    if(Reg3[104]) mem[Reg3[41:37]] <= Reg3[36:5];
always@(posedge clk)
    if(Reg3[105]) MemData <=mem[Reg3[41:37]];

endmodule

```

```
module Reg4(input [31:0]MemData, input [106:0]Reg3, input clk,start, output logic [70:0]Reg4);
```

```
always@(posedge clk or posedge start)
```

```
    if(start)Reg4<=0;
    else      Reg4<={Reg3[103:102],MemData,Reg3[68:37],Reg3[4:0]};
```

```
endmodule
```

```
module Mux4(input [70:0]Reg4,output [31:0]Mux4);
```

```
assign Mux4=(Reg4[69])?Reg4[68:37]:Reg4[36:5];
```

```
endmodule
```

```
module Control (input [63:0]Reg1,output logic RegDst, ALUSrc, MemtoReg, RegWrite, MemWrite, Branch, ALUOp1,
ALUOp0, Jump, MemRead);
```

```
always_comb
```

```
begin
```

```
    unique case(Reg1[31:26])
```

```
        6'b000000: begin // R-format
```

```
            RegDst=1;
```

```
            ALUOp1=1;
```

```
            ALUOp0=0;
```

```
            ALUSrc=((Reg1[5:0]==6'b101101) || (Reg1[5:0]==6'b101010))?1'b1:1'b0;
```

```
            Branch=0;
```

```
            MemtoReg=0;
```

```
            RegWrite=1;
```

```
            MemWrite=0;
```

```
            Jump=0;
```

```
            MemRead=0;
```

```
        end
```

```
        6'b100011: begin // Opcion de Load
```

```
            RegDst=0;
```

```
            ALUSrc=1;
```

```
            MemtoReg=1;
```

```
            RegWrite=1;
```

```
            MemWrite=0;
```

```
            Branch=0;
```

```
            ALUOp1=0;
```

```
            ALUOp0=0;
```

```
            Jump=0;
```

```
            MemRead=1;
```

```
        end
```

```
        6'b101011: begin // Opcion de Store
```

```
            RegDst=0;
```

```
            ALUSrc=1;
```

```
            MemtoReg=0;
```

```
            RegWrite=0;
```

```
            MemWrite=1;
```

```
            Branch=0;
```

```
            ALUOp1=0;
```

```

        ALUOp0=0;
        Jump=0;
        MemRead=0;
        end
    6'b000100: begin//Opcion de branch
        RegDst=0;
        ALUSrc=0;
        MemtoReg=0;
        RegWrite=0;
        MemWrite=0;
        Branch=1;
        ALUOp1=0;
        ALUOp0=1;
        Jump=0;
        MemRead=0;
        end
    6'b100000: begin //Opcion de Jump
        RegDst=0;
        ALUSrc=0;
        MemtoReg=0;
        RegWrite=0;
        MemWrite=0;
        Branch=0;
        ALUOp1=0;
        ALUOp0=0;
        Jump=1;
        MemRead=0;
        end
    6'b000001: begin // R-format
        RegDst=0;
        ALUOp1=1;
        ALUOp0=0;
        ALUSrc=1;
        Branch=0;
        MemtoReg=0;
        RegWrite=1;
        MemWrite=0;
        Jump=0;
        MemRead=0;
        end

endcase
end
endmodule

```

```

module Gate(input [106:0]Reg3,input Zero, output logic Gate);
assign Gate=Zero&Reg3[106];
endmodule

```

```
module IJump(input Jump,input [63:0]Reg1,output [31:0]PCjump,input [31:0]Mux1);
```

```
assign PCjump=(Jump)?{Reg1[63:60]},{Reg1[25:0]<<2}):Mux1;
```

```
endmodule
```

```
module DataPath(input clk,start,input [31:0]R30In,output [31:0]R31Out,input [4:0]Select,  
input RegDst,ALUSrc,MemtoReg,RegWrite,MemWrite,Branch,Jump,ALUOp1,ALUOp0,MemRead,  
output logic [63:0]Reg1,output logic [146:0]Reg2);  
logic [31:0]AddPc; logic Gate; logic [31:0]Mux1,Mux4,Mux2; logic [4:0]Mux3; logic [31:0]PCjump; logic [31:0]PC;  
logic [31:0]Instruction; logic [106:0]Reg3; logic [70:0]Reg4; logic [31:0]Add; logic [31:0]MemData; logic [31:0]result;  
logic Zero; logic [31:0]data1,data2;
```

```
Mux1          M1(. *);  
PC             M2(. *);  
AddPc  M3(. *);  
InstMem        M4(. *);  
Reg1           M5(. *);  
Registro M6(. *);  
Reg2           M7(. *);  
Add            M8(. *);  
Mux2           M9(. *);  
Mux3           M10(. *);  
ALU            M11(. *);  
Reg3           M12(. *);  
Memory         M13(. *);  
Reg4           M14(. *);  
Mux4           M15(. *);  
Gate           M18(. *);  
IJump          M20(. *);
```

```
Endmodule
```

```
module MainControl(input [146:0]Reg2,output logic[4:0]Select,input [63:0]Reg1,  
output RegDst,ALUSrc,MemtoReg,RegWrite,MemWrite,Branch,ALUOp1,ALUOp0,Jump,MemRead);
```

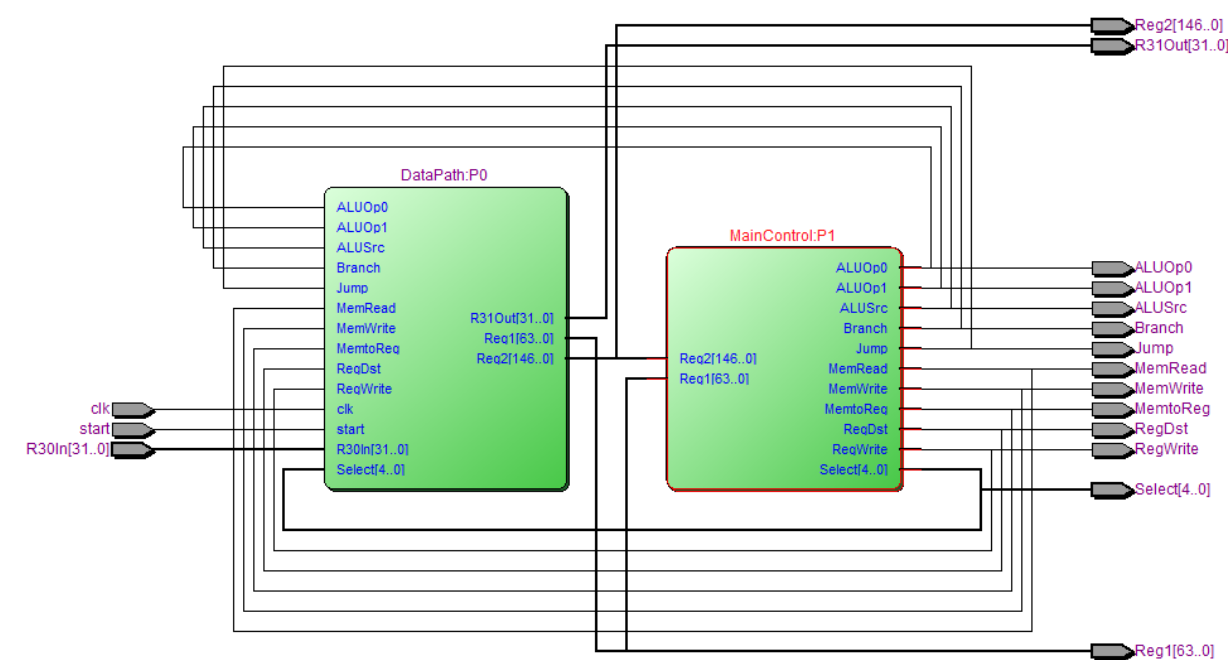
```
ALUControl    C0(. *);  
Control       C1(. *);  
Endmodule
```

```
module Pipeline(input clk,start,input [31:0]R30In,  
output [31:0]R31Out,output logic [4:0]Select,  
output logic RegDst,ALUSrc,MemtoReg,RegWrite,MemWrite,Branch,Jump,ALUOp1,ALUOp0,MemRead,  
output logic [146:0]Reg2,output logic [63:0]Reg1);
```

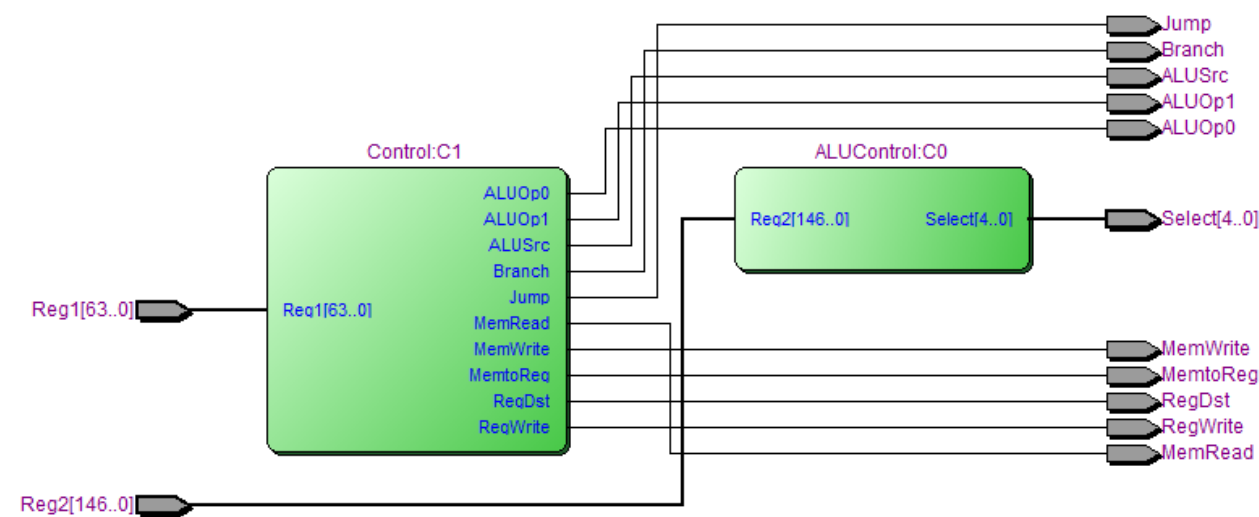
```
DataPath      P0(. *);  
MainControl P1(. *);
```

```
Endmodule
```

Herarquia Pipeline



Main Control



DataPath

