Universidad de Guadalajara

Centro Universitario de Ciencias Exactas e Ingenierías

Carlos Eduardo Rodríguez García

216101729

Materia: Seminario de solución de problemas de arquitectura de computadoras

Actividad: Proyecto Final

Maestro: López Arce Delgado Jorge Ernesto

Sección: D12

Introducción:

Este proyecto final se enfoca en crear un programa con las instrucciones que han estado presente a lo largo de este ciclo las cuales son las instrucciones de Tipo R, de Tipo I (a la cual se le agrego; SLTI, BEQ, BNE) y de Tipo J.

Entre las características más importantes para el uso de nuestro procesador MIPS son:

- *PC* que se encargara de mandar la dirección requerida de la memoria de instrucciones.

- *Memoria de instrucciones* de aquí nace todo, pues es la que se encarga de mandar la instrucción a todos los módulos y regir la dirección de los bits.

- *Banco de Registros* se encarga de almacenar "momentáneamente" los datos que utilizaremos pues pueden ser remplazados en cualquier momento algo como una memoria caché.

- *Memoria de Datos* nos servirá para almacenar un dato por más tiempo que en el banco de registros introduciendo y sacando datos a nuestra merced desde y hacia el banco de registros respectivamente.

- *Unidad de Control* es la que rige si hay escritura o lectura en ciertos módulos, incluso también funciona para decidir que dato pasará por un multiplexor.

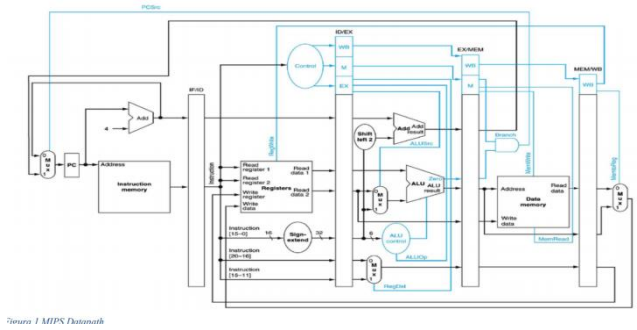
- *ALU* encargada de hacer operaciones lógicas y aritméticas.

- *Unidad de Control ALU* parte también de nuestra Unidad de Control, es la encargada de comunicar a ALU la operación a realizar que viene denotada desde la instrucción la cual llego a Unidad de Control como un Opcode.

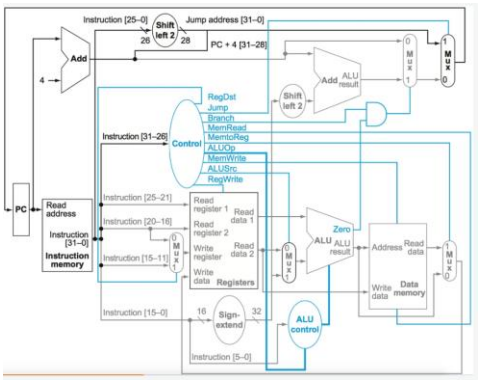
- *Buffers* nos ayudarán a establecer los tiempos de comunicación entre los módulos por cada fase del Pipelined.

Mencionar también que en este proyecto hace falta el uso de un conjunto de registros homogéneos, es decir, permitir que cualquier registro sea utilizado en cualquier contexto, pues podemos manejar datos enteros, más no de coma flotante.

Para la implementación de este proyecto se utilizó la combinación de los siguientes dos diagramas:



Datapath 1.



Datapath 2.

Teorema del binomio; es una fórmula que proporciona el desarrollo de la potencia *n*-ésima (siendo *n*, entero positivo) de un binomio. Una expresión algebraica que contiene dos términos se denomina expresión binomial. La forma general de una expresión binomial es (x + y) y la expansión $(x+y)^n$ se le denomina teorema del binomio.

De acuerdo con el teorema, es posible expandir la potencia $(x+y)^n$ en una suma que implica términos de la forma $ax^b\,y^c$, donde los exponentes *b* y *c* son números naturales con *b* + *c* = *n*, y el coeficiente *a* de cada término es un número entero positivo que depende de *n* y *b*. Cuando un exponente es cero, la correspondiente potencia es usualmente omitida del término.

El teorema del binomio sólo será válido en términos de una potencia entera y positiva de un binomio. Y solo se utilizará coeficientes positivos enteros debido a las limitaciones del programa en cuanto a los signos.

Lo primero que se necesitará es comprender que cuando elevamos un binomio a una cierta potencia, observaremos el comportamiento siguiente:

$$(x+y)^n = x^n + a\,x^{n-1}y + b\,x^{n-2}y^2 + \cdots + b\,x^2y^{n-2} + a\,xy^{n-1} + y^n$$

Lo que vamos a observar es que el exponente del primer término, x, empieza con el valor de la potencia del binomio y se va restando uno en uno hasta que su exponente llega a valer cero. Ocurre lo contrario con el segundo término del binomio, y, su exponente comienza en cero (ya que un número elevado a la cero es igual a uno) y va aumentando uno en uno el exponente hasta que llega a valer lo que vale el exponente del binomio.

Objetivos:

En esta actividad se busca que con la implementación del diseño de la actividad 12 y agregando los módulos al datapath de dos “Shift left 2”, dos “sumadores” para el PC, un “PC” encargado de mandar la dirección de la siguiente instrucción que estará en la “Memoria de instrucciones”, un “AND” que servirá para la implementación de Branch el proyecto sea capaz de implementar instrucciones nuevas como las Tipo J y algunas otras también de Tipo I como BEQ, para así realizar un programa previamente definido, que en este caso será el teorema del binomio.

Como no se tendrá interacción con algún usuario lo que haremos será establecer coeficientes y potencias dentro de nuestras instrucciones y ver que los resultados que tengamos sean los esperados, comprobando así que las operaciones se realizaron correctamente.

Finalmente, la idea es que tengamos algunos valores, por ejemplo: $(5+3)^2$ y como resultado tengamos $25x^2+30xy+9y^2$, mostrando como resultado únicamente 25, 30 y 9 que serán los resultados elevados a la potencia establecida, correspondiente a la fórmula mostrada con anterioridad.

Desarrollo:

Para el desarrollo del proyecto utilizamos todos los módulos que implementamos en la actividad 12, implementando cambios en algunos de ellos y agregando otros completamente nuevos.

Banco de Registros:

```
module BR(
    input [4:0]AR1,
    input [4:0]AR2,
    input [4:0]AW,
    input enwr,clk,
    input [31:0]DW,
    output reg [31:0]DR1, DR2
);
 reg [31:0]MEM[0:31];

 always@(negedge  clk)
begin
    if(enwr)
       begin
          MEM[AW] <= DW;
       end
    DR1 <= MEM[AR1];
    DR2 <= MEM[AR2];
    end
 initial
    begin
       MEM[0]=32'd0;
       MEM[1]=32'd0;
       MEM[2]=32'd0;
       MEM[3]=32'd0;
       MEM[4]=32'd0;
    end

 endmodule
```

Figura 1.

Para el Banco de Registros utilizamos el que siempre habíamos estado implementando, únicamente con dos cambios significativos, que son el ciclo de reloj en negedge y los registros inicializados en ceros, los cuales cambiaremos únicamente con una suma inmediata, más adelante.

Registros del BR	Datos
0	0
1	0
2	0
3	0
4	0

Figura 2.

ALU:

La unidad aritmética-lógica también se le hicieron algunos cambios:

```
module ALU(
input [31:0] e1, e2,
input [3:0] sel,
output reg zf,
output reg [31:0] res
);
always @(*)
   begin
      case (sel)
         4'b0000:
            begin
               res= e1&e2;
            end
         4'b0001:
            begin
               res= e1|e2;
            end
         4'b0010:
            begin
               res= e1^e2;
            end
         4'b0011:
            begin
               res= ~(e1&e2);
            end
         4'b0100:
            begin
               res= e1+e2;
            end
         4'b0101:
            begin
               res= e1-e2;
            end
         4'b0110:
            begin
               res= e1*e2;
            end
         4'b0111:
            begin
               res= e1/e2;
            end
         4'b1000:
            begin
               res= e1<e2;
            end
         4'b1001:
            begin
               res= e1==e2;
            end
      endcase
      case (res)
         32'b0:
            begin
               zf=1;
            end
         default:
            zf=0;
      endcase
   end

endmodule
```

Figura 3.

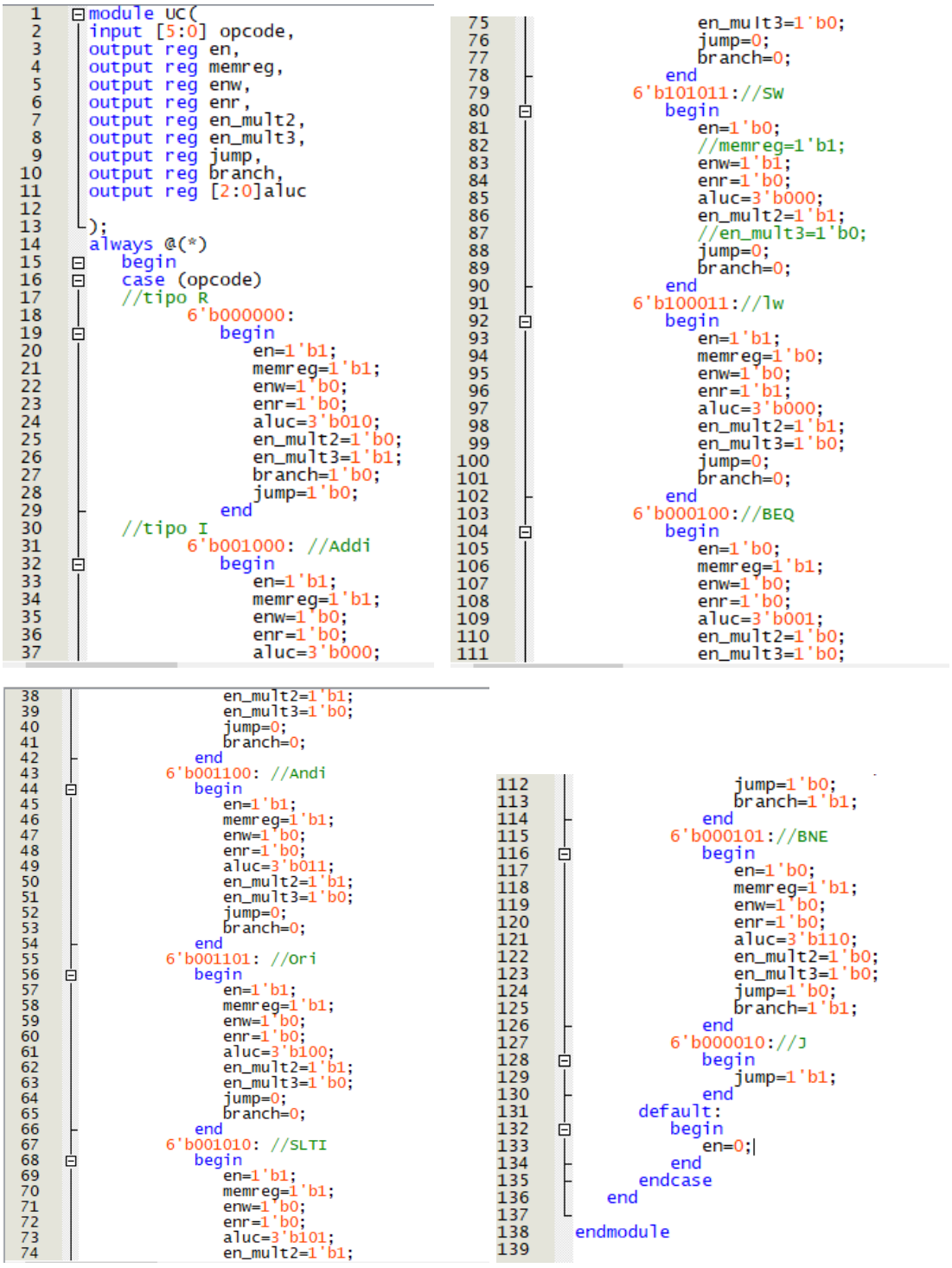
El principal cambio fue agregar un bit más a la entrada selectora, precisamente para agregar una nueva operación, para el uso del branch y el menor que.

Selectora	Operación
0000	AND
0001	OR
0010	XOR
0011	NAND
0100	SUMA
0101	RESTA
0110	MULTIPLICACIÓN
0111	DIVISIÓN
1000	MENOR QUE
1001	COMPARACIÓN IGUALES

Figura 4.

Unidad de control:

A la unidad de control también se le agrego un bit más para poder mandar a la alu control más señales. El código es el siguiente:



```
module UC(
input [5:0] opcode,
output reg en,
output reg memreg,
output reg enw,
output reg enr,
output reg en_mult2,
output reg en_mult3,
output reg jump,
output reg branch,
output reg [2:0]aluc

);
always @(*)
   begin
   case (opcode)
   //tipo R
         6'b000000:
            begin
               en=1'b1;
               memreg=1'b1;
               enw=1'b0;
               enr=1'b0;
               aluc=3'b010;
               en_mult2=1'b0;
               en_mult3=1'b1;
               branch=1'b0;
               jump=1'b0;
            end
   //tipo I
         6'b001000: //Addi
            begin
               en=1'b1;
               memreg=1'b1;
               enw=1'b0;
               enr=1'b0;
               aluc=3'b000;
               en_mult2=1'b1;
               en_mult3=1'b0;
               jump=0;
               branch=0;
            end
         6'b001100: //Andi
            begin
               en=1'b1;
               memreg=1'b1;
               enw=1'b0;
               enr=1'b0;
               aluc=3'b011;
               en_mult2=1'b1;
               en_mult3=1'b0;
               jump=0;
               branch=0;
            end
         6'b001101: //Ori
            begin
               en=1'b1;
               memreg=1'b1;
               enw=1'b0;
               enr=1'b0;
               aluc=3'b100;
               en_mult2=1'b1;
               en_mult3=1'b0;
               jump=0;
               branch=0;
            end
         6'b001010: //SLTI
            begin
               en=1'b1;
               memreg=1'b1;
               enw=1'b0;
               enr=1'b0;
               aluc=3'b101;
               en_mult2=1'b1;
               en_mult3=1'b0;
               jump=0;
               branch=0;
            end
         6'b101011://SW
            begin
               en=1'b0;
               //memreg=1'b1;
               enw=1'b1;
               enr=1'b0;
               aluc=3'b000;
               en_mult2=1'b1;
               //en_mult3=1'b0;
               jump=0;
               branch=0;
            end
         6'b100011://lw
            begin
               en=1'b1;
               memreg=1'b0;
               enw=1'b0;
               enr=1'b1;
               aluc=3'b000;
               en_mult2=1'b1;
               en_mult3=1'b0;
               jump=0;
               branch=0;
            end
         6'b000100://BEQ
            begin
               en=1'b0;
               memreg=1'b1;
               enw=1'b0;
               enr=1'b0;
               aluc=3'b001;
               en_mult2=1'b0;
               en_mult3=1'b0;
               jump=1'b0;
               branch=1'b1;
            end
         6'b000101://BNE
            begin
               en=1'b0;
               memreg=1'b1;
               enw=1'b0;
               enr=1'b0;
               aluc=3'b110;
               en_mult2=1'b0;
               en_mult3=1'b0;
               jump=1'b0;
               branch=1'b1;
            end
         6'b000010://J
            begin
               jump=1'b1;
            end
       default:
          begin
             en=0;
          end
       endcase
   end

endmodule
```

Figura 5.

Instrucción	Opcode	en	memreg	enw	enr	en_mult2	en_mult3	jump	branch	aluc
Tipo R	000000	1	1	0	0	0	1	0	0	010
ADDI	001000	1	1	0	0	1	0	0	0	000
ANDI	001100	1	1	0	0	1	0	0	0	011
ORI	001101	1	1	0	0	1	0	0	0	100
SLTI	001010	1	1	0	0	1	0	0	0	101
SW	101011	0	x	1	0	1	x	0	0	000
LW	100011	1	0	0	1	1	0	0	0	000
BEQ	000100	0	1	0	0	0	0	0	1	001
BNE	000101	0	1	0	0	0	0	0	1	110
J	000010	x	x	x	x	x	x	1	x	x

Figura 6.

En la figura 5 se puede ver como comentario a la instrucción que corresponde cada opcode.

Unidad de control ALU:

Para la unidad de control solo agregamos la instrucción de Tipo I; STLI y el BNE

```
module UCA(
input [2:0]aluc,
input [5:0]func,
output reg[3:0]sal_alu
);
always @(*)
   begin
      case(aluc)
         3'b010://TIPO R
            begin
               case(func)
                  6'b100000://suma
                     begin
                        sal_alu=4'b0100;
                     end
                  6'b100010://resta
                     begin
                        sal_alu=4'b0101;
                     end
                  6'b011000://multiplicacion
                     begin
                        sal_alu=4'b0110;
                     end
                  6'b011010://division
                     begin
                        sal_alu=4'b0111;
                     end
                  6'b101010://menor que
                     begin
                        sal_alu=4'b1000;
                     end
                  6'b100100://and
                     begin
                        sal_alu=4'b0000;
                     end
                  6'b100101://or
                     begin
                        sal_alu=4'b0001;
                     end
               endcase
            end
         3'b000: //TIPO I
               begin
                  sal_alu=4'b0100;//suma
               end
         3'b001:
               begin
                  sal_alu=4'b0101;//resta
               end
         3'b011:
               begin
                  sal_alu=4'b0000;//and
               end
         3'b100:
               begin
                  sal_alu=4'b0001;//or
               end
         3'b101:
               begin
                  sal_alu=4'b1000;//slt
               end
         3'b110:
               begin
                  sal_alu=4'b1001;//No Iguales
               end
      endcase
   end
endmodule
```

Figura 7.

TIPO R		
ALU code	**Function**	**Operación en ALU**
010	100000	SUMA
	100010	RESTA
	011000	MULTIPLICACIÓN
	011010	DIVISIÓN
	101010	MENOR QUE
	100100	AND

	100101	OR
Tipo I		
000	X	SUMA
001	X	RESTA (BEQ)
011	X	AND
100	X	OR
101	X	SLT
110	X	COMPARACIÓN IGUALES (BNE)

Figura 8.

Una vez aclarados los cambios, mencionaré a continuación los otros módulos que fueron necesarios para el Datapath.

PC:

Para el cual se utilizó el siguiente código:

```
1   module pc(
2   input [8:0] e1,
3   input clk,
4   output reg [8:0] out
5   );
6
7   always@(posedge clk)
8   begin
9       out=e1;
10  end
11  initial
12      begin
13          out=9'd0;
14      end
15
16  endmodule
```

Figura 9.

El cual se encargará de estar mandando la dirección de la siguiente instrucción a la memoria de datos en múltiplos de 4. Este deberá inicializar en un 0 la salida para darle tiempo al reloj de que la primera instrucción se mande correctamente.

Memoria de Instrucciones:

```
module memoria_instrucciones(
input clk,
input [8:0]dir,
output reg [31:0] instruccion
);

reg [7:0] mem [0:511];

always@(*)
begin
    instruccion<={mem[dir],mem[dir+1],mem[dir+2],mem[dir+3]};
end
initial
   begin
      $readmemb("Instrucciones.mif",mem);
   end

endmodule
```

Figura 10.

Está estará mandando una instrucción que le diga la dirección que provenga del PC, pero la estará armando hasta obtener 32 bits y mandarlos todos juntos, pues en nuestro archivo de texto las instrucciones están separadas de 8 bits en 8 bits.

BUFFERS:

Los buffers se encargarán de controlar el tiempo en el que los datos llegan a los diferentes módulos. Estos serán 4 y estarán también para separar las fases del programa.

```
module BUFER_ID_EX(
input  [31:0] e1,e2,e3,e4,e5,e6,e7,e8,e9,e10,e11,e12,e13,e14,e15,e16,e17,
input clk,
output reg[31:0] out1,out2,out3,out4,out5,out6,out7,out8,out9,out10,out11,out12,out13,out14,out15,out16,out17
);

 always@(posedge clk)
begin
    out1<=e1;
    out2<=e2;
    out3<=e3;
    out4<=e4;
    out5<=e5;
    out6<=e6;
    out7<=e7;
    out8<=e8;
    out9<=e9;
    out10<=e10;
    out11<=e11;
    out12<=e12;
    out13<=e13;
    out14<=e14;
    out15<=e15;
    out16<=e16;
    out17<=e17;
end
 endmodule
```

Figura 11.

Aquí tenemos el código de la fase ID/EX todos los buffers son lo mismo solo cambia el número de entradas y por consiguiente también el número de salidas.

ALUS:

También se agregaron 2 ALUS una se encargará de siempre sumarle un 4 al número de dirección que arroje el PC para así seguir aumentando direcciones.

```
1  module SUM_4(
2  input  [31:0] op1,
3  output [31:0] SUMA
4  );
5
6
7  assign SUMA=op1+9'd4;
8
9  endmodule
```

Figura 12.

Y otra que sumará el resultado de la ALU anterior y un dato que saldrá del módulo shif left 2.

```
1  module SUM(
2  input  [31:0] op1, op2,
3  output [31:0] SUMA
4  );
5
6
7  assign SUMA=op1+op2;
8
9  endmodule
```

Figura 13.

Shift left 2:

Lo necesitaremos para desplazar dos lugares el bit más significativo e insertar 0´s en el menos significativo.

```
1  module SHIFT2(
2  input [31:0] e1,
3  output reg [31:0] out1
4  );
5  always@(*)
6  begin
7  out1 = e1 << 2;
8  end
9  endmodule
```

Figura 14.

AND:

Se necesitará una compuerta AND para la implementación de Branch.

```
module MA(
input  op1,op2,
output AND
);

assign AND=op1&op2;

endmodule
```

Figura 15.

Datapath:

```
module toplevel(
input clk,
output [4:0]Direccion_1,Direccion_2,Direccion_escribir,
output [31:0]Escribir_Dato,Leer_DR1,Leer_DR2,instruccion,Entrada1,Entrada2, Resultado,Direccion_memdato,Escribir_dato,
output [8:0] numero_de_instruccion,
output [3:0] selectora,
output ZF,Escribir,Leer
);
//SECCIÓN 1

wire [8:0] w1;
wire [31:0] w2;
wire [31:0] w3;

//////////////////////

//SECCIÓN 2
wire [31:0] w4, inst;
wire w5,w6,w7,w8,w9,w10,w11,w12,w60;
wire [2:0] w13;
wire [4:0] w14,w61,w62,w63,w64;
wire [31:0] w15,w16,w17;
wire [27:0] w18;
//////////////////////

//SECCIÓN 3

wire [27:0] w19;
wire [31:0] w20,w28,w29,w30,w32,w35,w36,w37;
wire w21,w22,w23,w24,w25,w26,w34,w38;
wire [2:0] w27;
wire [5:0] w31;
wire [3:0] w33;

//////////////////////

//SECCIÓN 4
```

```
//SECCIÓN 4
wire [27:0] w39;
wire [31:0] w40,w41,w47,w48,w50,w51;
wire w42,w43,w44,w45,w46,w49;

//////////////////////

//SECCIÓN 5

wire w52,w55;
wire [27:0]w53;
wire [31:0]w54,w56,w57,w58,w59;

wire w70,w71,w72;
//////////////////////

///pruebas
//Br

assign Direccion_1 = inst[25:21];
assign Direccion_2 = inst[20:16];
assign Direccion_escribir = w64;
assign Escribir_Dato = w59;
assign Leer_DR1 = w16;
assign Leer_DR2 = w17;
//instruccion
assign numero_de_instruccion = w2;
assign instruccion = w6;

/////////ALU
assign Entrada1 = w28;
assign Entrada2 = w32;
assign selectora = w33;
assign Resultado = w35;
assign ZF = w34;
///////Memoria datos
assign Escribir = w43;
```

```
74    assign Leer = w45;
75    assign Direccion_memdato = w47;
76    assign Escribir_dato = w48;
77
78    ////SECCIÓN 1/////////
79
80    pc inst11(
81    .e1(w58),
82    .clk(clk),
83    .out(w1)
84    );
85
86
87    memoria_instrucciones inst12(
88    .clk(clk),
89    .dir(w1),
90    .instruccion(w3)
91    );
92
93    SUM_4 inst13(
94    .op1({23'd0,w1}),
95    .SUMA(w2)
96    );
97
98    BUFER_IF_ID inst16(
99    .e1(w2),
100   .e2(w3),
101   .clk(clk),
102   //////SECCIÓN 2////////
103   .out1(w4),
104   .out2(inst)
105   );
106
107   SHIFT2 inst21(
108   .e1(inst[25:0]),
109   .out1(w18)
110   );
111
112   UC inst2(
113   .opcode(inst[31:26]),
114   .en(w5),
115   .memreg(w12),
116   .enr(w7),
117   .enw(w8),
118   .branch(w9),
119   .jump(w10),
120   .en_mult2(w11),
121   .en_mult3(w6),
122   .aluc(w13)
123   );
124
125   BR inst1(
126   .AR1(inst[25:21]),
127   .AR2(inst[20:16]),
128   .AW(w64),
129   .clk(clk),
130   .enwr(w72),
131   .DW(w59),
132   .DR1(w16),
133   .DR2(w17)
134   );
135
136   sign_ext inst10(
137   .unextend(inst[15:0]),
138   .extended(w15)
139   );
140
141   BUFER_ID_EX inst17(
142   .e1(w18),
143   .e2(w4),
144   .e3(w10),
145   .e4(w9),
146   .e5(w7),
```

```
147     .e6(w12),
148     .e7(w13),
149     .e8(w8),
150     .e9(w11),
151     .e10(w16),
152     .e11(w17),
153     .e12(w15),
154     .e13(inst[5:0]),
155     .e14(w6),
156     .e15(inst[20:16]),
157     .e16(inst[15:11]),
158     .e17(w5),
159     .clk(clk),
160     ///////SECCIÓN 3//////
161     .out1(w19),
162     .out2(w20),
163     .out3(w21),
164     .out4(w22),
165     .out5(w23),
166     .out6(w24),
167     .out7(w27),
168     .out8(w25),
169     .out9(w26),
170     .out10(w28),
171     .out11(w29),
172     .out12(w30),
173     .out13(w31),
174     .out14(w60),
175     .out15(w61),
176     .out16(w62),
177     .out17(w70)
178     );
179
180   mult3 inst8(
181     .in_uc(w60),
182     .in_TI(w61),
183     .in_TR(w62),
184     .out_AW(w14)
185     );
186
187   ALU inst3(
188     .e1(w28),
189     .e2(w32),
190     .sel(w33),
191     .zf(w34),
192     .res(w35)
193     );
194
195   UCA inst4(
196     .aluc(w27),
197     .func(w31),
198     .sal_alu(w33)
199     );
200
201   mult2 inst7(
202     .in_uc(w26),
203     .in_DR2(w29),
204     .in_inst(w30),
205     .out_e2(w32)
206     );
207
208   SHIFT2 inst20(
209     .e1(w30),
210     .out1(w36)
211     );
212
213   SUM inst14(
214     .op1(w20),
215     .op2(w36),
216     .SUMA(w37)
217     );
218
219
```

```
219   BUFER_EX_MEM inst18(
220   .e1(w21),
221   .e2(w19),
222   .e3(w20),
223   .e4(w37),
224   .e5(w22),
225   .e6(w23),
226   .e7(w24),///memtoreg
227   .e8(w25),//write
228   .e9(w34),//zf
229   .e10(w35),//res alu
230   .e11(w29),//
231   .e12(w14),
232   .e13(w70),
233   .clk(clk),
234   //////SECCIÓN 4//////
235   .out1(w38),
236   .out2(w39),
237   .out3(w40),
238   .out4(w41),
239   .out5(w42),
240   .out6(w43),
241   .out7(w44),
242   .out8(w45),
243   .out9(w46),
244   .out10(w47),
245   .out11(w48),
246   .out12(w63),
247   .out13(w71)
248   );
249
250   mult5 inst22(
251   .in_and(w49),
252   .in_x(w40),
253   .in_y(w41),
254   .out_z(w50)
255   );|
```

```
257   mem_dat inst5(
258   .AR(w47),
259   .enw(w45),
260   .enr(w43),
261   .DR(w51),
262   .DW(w48)
263   );
264
265   MA inst15(
266   .op1(w42),
267   .op2(w46),
268   .AND(w49)
269   );
270
271   BUFER_MEM_WB inst19(
272   .e1(w38),
273   .e2(w39),
274   .e3(w50),
275   .e4(w44),
276   .e5(w51),
277   .e6(w47),
278   .e7(w63),
279   .e8(w71),
280   .clk(clk),
281   //////SECCIÓN 5//////
282   .out1(w52),
283   .out2(w53),
284   .out3(w54),
285   .out4(w55),
286   .out5(w56),
287   .out6(w57),
288   .out7(w64),
289   .out8(w72)
290   );
291
292   mult4 inst9(
293   .in_jump(w52),
```

```
294     .in_x(w54),
295     .in_y(w53),
296     .out_z(w58)
297     );
298
299   mult inst6(
300     .in_uc(w55),
301     .in_DR(w56),
302     .in_res(w57),
303     .out(w59)
304     );
305
306   endmodule
```

Figura 16.

Recordando que para el diseño del datapath se utilizaron los dos diagramas expuestos en le introducción (*"Datapath 1"* y *"Datapath 2"*).

Se secciono con comentarios los cables y las fases para una elaboración más clara y a la vez facilitar la conexión entre los módulos. Los assign que se encuentran entre las secciones de los cables y la primera sección del datapath, fueron esenciales para facilitar la solución de errores, pues para no buscar cable por cable decidí mejor asignarlos a salidas para poder ver sus datos.

Por ejemplo, uno de los problemas que me ayudaron a solucionar fue que el banco de registro no escribía los datos que le mandaba y me di cuenta gracias a esto que era porque el bit de escritura no se mantenía activo mucho tiempo, así que cuando llegaba la instrucción ya estaba en 0 por lo que tenía que hacer que durara más tiempo pasándolo por los buffers.

Intente conservar estas asignaciones en el test bench, pero me causaron problemas y como el profe menciono que el TB solo debía contener en ciclo de reloj opte por quitarlas.

El set de instrucciones que puede usar el datapath es:

TIPO	INSTRUCCION	FORMATO ENSAMBLADOR	SIGNIFICADO
R	SUMA	add $rd, $rs, $rt	$rd = $rs + $rt
	RESTA	sub $rd, $rs, $rt	$rd = $rs - $rt
	MULTIPLICACIÓN	mult $rd, $rs, $rt	$rd = $rs * $rt
	DIVISION	div $rd, $rs, $rt	$rd = $rs / $rt
	AND	and $rd, $rs, $rt	$rd = $rs & $rt
	OR	or $rd, $rs, $rt	$rd = $rs | $rt
	MENOR QUE	slt $rd, $rs, $rt	If($rs < $rt) $rd=1 ó $rd=0
I	SUMA INMEDIATA	addi $rd, $rs, #inm	$rd = $rs + #inm
	AND INMEDIATO	andi $rd, $rs, #inm	$rd = $rs & #inm

	OR INMEDIATO	ori $rd, $rs, #inm	$rd = $rs | #inm
	LOAD WORD	lw $rd, $rs, #inm	$rt=Memoria[$rt+#inm]
	STORE WORD	sw $rd, $rs, #inm	Memoria[$rs+#inm] =$rs
	MENOR QUE INMEDIATO	slt $rt, $rs, #inm	if($rs<#inm) $rt=1; else $rt=0
	Branch on equal	beq $rt, $rs, #inm	if($rt==$rs) salta a PC+4+#inm
	Branch on not equal	bne $rt, $rs, #inm	if($rt!=$rs) salta a PC+4+#inm
J	JUMP	J #inm	Saltar a dirección #inm

Figura 17.

Test Bench:

Como ya mencioné antes el test bench solo tiene el ciclo de reloj pues las instrucciones están en el archivo de texto que se ejecutará en la memoria de datos (figura 10), por lo que el código es el siguiente:

```
1   `timescale 1ns/1ps
2   module toplevel_tb();
3   reg clk_tb;
4
5   toplevel duv(
6   .clk(clk_tb)
7   );
8
9   initial clk_tb =1'b0;
10  always #5 clk_tb =~clk_tb;
11
12
13  initial
14  begin
15  #1350;
16  $stop;
17  #400;
18  $stop;
19  end
20  endmodule
```

Figura 18.

Dividido en dos por un $stop las primeras instrucciones corresponden a la ejecución de mi programa del teorema del binomio y las otras instrucciones corresponden a la lw, stl, beq y j, ya que se tienen que ejecutar mínimo una vez.

Instrucciones:

Para implementar el teorema en mi programa utilice $(2+4)^2$ y $(3+6)^3$ tomando en cuenta el orden (x+y) así entonces realice lo que dicta el teorema, es decir:

Para el primer caso:

$$x^2 + 2xy + y^2$$

Para el segundo caso:

$$x^3 + 3x^2y + 3xy^2 + y^3$$

Sustituyendo:

Para el primer caso:

$$2^2 + 2(2 * 4) + 4^2$$

Para el segundo caso:

$$3^3 + 3(3^2 * 6) + 3(3 * 6^2) + 6^3$$

Resultados esperados:

Para el primer caso:

$$4x^2 + 16xy + 16y^2$$

Para el segundo caso:

$$27x^3 + 162x^2y + 324xy^2 + 216y^3$$

Cabe aclarar que lo único que esperamos como resultado son los coeficientes.

A continuación, tenemos las instrucciones para obtener los resultados esperados, a partir de la dirección 108 tenemos el segundo set de instrucciones donde veremos que lw, slt, beq y j se ejecutan.

Dirección de instrucción	Instrucción en ensamblador	Instrucción en binario	Comentario
0	Addi $1, $0, #2	001000_00000_00001_0000000000000010	#2 en BR $1
4	Addi $2, $0, #2	001000_00000_00010_0000000000000010	#2 en BR $2
8	Mult $3, $1, $2	000000_00001_00010_00011_00000_011000	$1*$2 en BR $3

12	Sw $3, $0, #1	101011_00000_00011_0000000000000001	Se guarda el resultado ($3) en MemDatos[$0+1]
16	Addi $4, $0, #4	001000_00000_00100_0000000000000100	#4 en BR $4
20	Addi $5, $0, #4	001000_00000_00101_0000000000000100	#4 en BR $5
24	Mult $6, $4, $5	000000_00100_00101_00110_00000_011000	$4*$5 en BR $6
28	Sw $6, $0, #3	101011_00000_00110_0000000000000011	Se guarda el resultado ($6) en MemDatos[$0+3]
32	Mult $7, $1, $4	000000_00001_00100_00111_00000_011000	$1*$4 en BR $7
36	Mult $8, $7, $1	000000_00111_00001_01000_00000_011000	$1*$7 en BR $8
40	Sw $8, $0, #2	101011_00000_01000_0000000000000010	Se guarda el resultado ($8) en MemDatos[$0+2]
44	Addi $2, $0, #3	001000_00000_00010_0000000000000011	#3 en BR $2
48	Addi $3, $0, #3	001000_00000_00011_0000000000000011	#3 en BR $3
52	Mult $4, $2, $3	000000_00010_00011_00100_00000_011000	$2*$3 en BR $4
56	Mult $5, $4, $3	000000_00100_00011_00101_00000_011000	$4*$3 en BR $5
60	Sw $5, $0, #5	101011_00000_00101_0000000000000101	Se guarda el resultado ($5) en MemDatos[$0+5]
64	Addi $6, $0, #6	001000_00000_00110_0000000000000110	#6 en BR $6
68	Addi $7, $0, #6	001000_00000_00111_0000000000000110	#6 en BR $7
72	Mult $8, $4, $6	000000_00100_00110_01000_00000_011000	$4*$6 en BR $8
76	Mult $9, $8, $2	000000_01000_00010_01001_00000_011000	$8*$2 en BR $9
80	Sw $9, $0, #6	101011_00000_01001_0000000000000110	Se guarda el resultado ($9) en MemDatos[$0+6]
84	Mult $10, $6, $7	000000_00110_00111_01010_00000_011000	$6*$7 en BR $10

88	Mult $11, $10, $7	000000_01010_00111_01011_00000_011000	$10*$7 en BR $11
92	Sw $11, $0, #8	101011_00000_01011_0000000000001000	Se guarda el resultado ($11) en MemDatos[$0+8]
96	Mult $12, $2, $10	000000_00010_01010_01100_00000_011000	$2*$10 en BR $12
100	Mult $13, $12, $2	000000_01100_00010_01101_00000_011000	$12*$2 en BR $13
104	Sw $13, $0, #7	101011_00000_01101_0000000000000111	Se guarda el resultado ($13) en MemDatos[$0+7]
108	Lw $16, $0, #7	100011_00000_10000_0000000000000111	Se guarda el MemDatos[$0+7] en BR $16
112	J #30	000010_00000000000000000000011110	saltar a ori
116	Andi $19, $1, #0101	001100_00001_10011_0000000000000101	Guardar en BR $19= 001(2 en decimal)
120	ori $20, $5, # 01001	001101_00101_10100_0000000000001001	Guardar en BR $20= 00011011(27 en decimal)
124	Beq $3, $4, #2	000100_00011_00010_0000000000000010	Compara $3==$4 si son iguales salta 2 direcciones (136)
128	Bne $1, $0 #4	000101_00001_00000_0000000000000100	Compara $1==$0 si son diferentes salta 4 direcciones.
132	Lw $17, $0, #6	100011_00000_10001_0000000000000110	Se guarda el MemDatos[$0+6] en BR $17
136	Stl $22, $5, $6	000000_00101_00110_10110_00000_101010	$5 < $6 = en BR $22 = 0

Figura 20.

Veamos la simulación en la cual deberíamos de ver los coeficientes antes mencionados:

0	x
1	4
2	16
3	16
4	x
5	27
6	162
7	324
8	216
9	x
10	x
11	x
12	x
13	x
14	x
15	x
16	x
17	x
18	x
19	x
20	x
21	x
22	x
23	x
24	x
25	x
26	x
27	x
28	x
29	x
30	x
31	x

$4x^2 + 16xy + 16y^2$

$27x^3 + 162x^2y + 324xy^2 + 216y^3$

Figura 21.

Vemos que los datos fueron guardados satisfactoriamente en la memoria de datos.

Así queda el banco de registros al final:

0	0
1	2
2	3
3	3
4	9
5	27
6	6
7	6
8	54
9	162
10	36
11	216
12	108
13	324
14	x
15	x
16	x
17	x
18	x
19	x
20	x
21	x
22	x
23	x
24	x
25	x
26	x
27	x
28	x
29	x
30	x
31	x

Figura 22.

Ahora en la segunda parte del TB esperamos ver en el BR que en el registro 16 un 324 que sacamos con un lw de la memoria de datos, un 00011011 (27) en el registro 20 y un 0 en el registro 22. Todas estas instrucciones las podemos ver en la *Figura 20*.

0	0
1	2
2	3
3	3
4	9
5	27
6	6
7	6
8	54
9	162
10	36
11	216
12	108
13	324
14	x
15	x
16	324
17	x
18	x
19	x
20	27
21	x
22	0
23	x
24	x
25	x
26	x
27	x
28	x
29	x
30	x
31	x

Figura 23.

Confirmamos que los resultados son correctos y los esperados, concluyendo así que las instrucciones fueron hechas satisfactoriamente.

CONCLUSION:

El programa se me hizo muy laborioso y confuso, más que nada por tanta conexión entre los diferentes módulos y los buffers, también me costó trabajo en cuestión de los tiempos entre las transferencias de datos, como lo menciono en el Datapath.

La materia realmente sí se me hace útil pues nos prepara conociendo lo más esencial de una computadora, el hardware y al menos para mí me sirvió mucho pues aprendí demasiado sobre cómo funciona la comunicación entre los componentes de una computadora, conocimientos que antes desconocía totalmente.

Finalmente quiero agradecer al maestro Jorge Ernesto López Arce Delgado por su disposición de enseñar y siempre contestar mis dudas de una manera clara y consistente.

Referencias

David a. patterson, j. l. (2014). *computer organization and design.* Miami: Morgan Kaufman.

MIPS® Architecture for Programmers. (2016).

RBJLabs. (s.f.). *RBJLabs*. Obtenido de RBJLabs: https://www.rbjlabs.com/algebra/teorema-del-binomio/