

UNIVERSIDAD DE GUADALAJARA

Centro Universitario de Ciencias Exactas e Ingenierías.



Reporte

Alumno:

Espinoza Sucilla Samuel 214017739

García Guevara Ángel Damián Raúl 220791063

Abraham Magaña Hernández 220791217

Profesor:

Jorge Ernesto López Arce Delgado.

Materia:

Seminario de Solución de problemas de Arquitectura de Computadoras.

Fase 1.

Introducción

Dentro de este reporte se condensarán los objetivos de la fase 1, el desarrollo de los objetivos de la misma, la investigación previa que se requiere para llevar a cabo los objetivos propuestos, las respectivas conclusiones de la labor hecha y al final su bibliografía.

Elementos y características generales del procesador MIPS (32 bits):

Podemos empezar el reporte definiendo de manera breve pues lo que es el procesador de tipo MIPS como recordatorio. Los procesadores de arquitectura MIPS (*Microprocessor Without Interlocked Pipelines Stages*) son procesadores que implementan una arquitectura de tipo RISC. El diseño de esta arquitectura de procesadores tiene como objetivo la optimización al segmentar las unidades de control y también busca facilitar la generación de código máquina por parte de los compiladores.

Cómo las características principales podemos decir que tiene las siguientes:

- El tamaño de las instrucciones es de 32 bits por instrucción
- Cuenta con tres tipos de instrucciones: R, I, J
- Las instrucciones contienen tres operandos: los dos registros fuente y el registro destino
- Arquitectura tipo RISC
- Bajo consumo energético

Cómo elementos principales podemos decir que tiene:

- ALU
- Unidad de control
- Banco de registros
- Memoria de datos
- Memoria de instrucciones

Objetivos de la fase 1:

- El diseño y la implementación en Verilog de los módulos necesarios dentro del *Data Path* para llevar a cabo operaciones de tipo R en un procesador MIPS.
- Generar una propuesta de algoritmo en ensamblador para revisión y posterior aprobación (bajo los cambios necesarios) del profesor.
- Generar una presentación donde se exponga el trabajo elaborado durante la fase uno del proyecto, exponiendo el correcto funcionamiento del *Data Path* a implementar, la elaboración del reporte en donde se condense el trabajo hecho.

Desarrollo:

Set de instrucciones:

Instrucción en ensamblador	Op. Code	rs	rt	rd	shamt	funct
ADD	000000	XXXXXX	XXXXXX	XXXXXX	00000	100000
SUB	000000	XXXXXX	XXXXXX	XXXXXX	00000	100010
MUL	000000	XXXXXX	XXXXXX	XXXXXX	00000	011000
AND	000000	XXXXXX	XXXXXX	XXXXXX	00000	100100
OR	000000	XXXXXX	XXXXXX	XXXXXX	00000	100101
SLT	000000	XXXXXX	XXXXXX	XXXXXX	00000	101010
NOP	000000	XXXXXX	XXXXXX	XXXXXX	00000	000000

Módulos:

Top Level:

Código:

```

1  `timescale 1ns/1ns
2
3  module TopLevel(
4      input clk
5  );
6
7      // Unidad de Control
8      wire C_MemToReg,C_MemToWrite,C_RegWrite,C_RegDst,C_Branch,C_MemRead,C_AlusSrc,C_shift_out;
9      wire[2:0]C_AlOp;
10     // ALU
11     wire [31:0]C_i_op1,C_i_op2,C_mux_i_op2,C_r_out,C_Rdata,C_mux_out;
12     wire [3:0]C_Alus;
13     wire C_Zflag;
14     // Ciclo Fetch
15     wire [31:0]C_PC_in,C_PC_out,C_suma_out,C2_suma_out,C_extend_out;
16     wire C_branch_out;
17     // Memoria de Instrucciones
18     wire [31:0]C_InsOut;
19     wire [4:0]C2_mux_out;

```

Instancias del Top Level:

```

20
21  PC PC1(
22      .PC_in(C_PC_in),
23      .clk(clk),
24      .PC_out(C_PC_out)
25  );
26  MemIns MI1(
27      .InsDir(C_PC_out),
28      .InsOut(C_InsOut)
29  );
30  UnidadDeControl UDC(
31      .op(C_InsOut[31:26]),
32      .MemToReg(C_MemToReg),
33      .MemToWrite(C_MemToWrite),
34      .AluOp(C_AlOp),
35      .RegWrite(C_RegWrite),
36      .RegDst(C_RegDst),
37      .Branch(C_Branch),
38      .MemRead(C_MemRead),
39      .AlusSrc(C_AlusSrc)
40  );
41  BancoReg BR(
42      .RegWrite(C_RegWrite),
43      .RA1(C_InsOut[25:21]),
44      .RA2(C_InsOut[20:16]),
45      .WriteData(C_mux_out),
46      .AW(C2_mux_out),
47      .DR1(C_i_op1),
48      .DR2(C_mux_i_op2)
49  );
50  SignExtend SE(
51      .extend_in(C_InsOut[15:0]),
52      .extend_out(C_extend_out)
53  );
54  ShiftLeft SL(
55      .shift_in(C_extend_out),
56      .shift_out(C_shift_out)
57  );

```

```

58  v AluControl AC(
59      .Aop(C_AluOp),
60      .Func(C_InsOut[5:0]),
61      .AluS(C_AluS)
62  );
63  v Alu ALU1(
64      .i_op1(C_i_op1),
65      .i_op2(C_i_op2),
66      .Sel(C_AluS),
67      .Zflag(C_Zflag),
68      .r_out(C_r_out)
69  );
70  v Mem Mem(
71      .MemWrite(C_MemToWrite),
72      .MemRead(C_MemRead),
73      .Adress(C_r_out),
74      .WriteD(C_mux_i_op2),
75      .Rdata(C_Rdata)
76  );
77  v Sumador4 S1(
78      .sum_in1(C_PC_out),
79      .suma_out(C_suma_out)
80  );
81  v Sumador S2(
82      .sum_in1(C_suma_out),
83      .sum_in2(C_shift_out),
84      .suma_out(C2_suma_out)
85  );
86  v Branch B1(
87      .branch(C_Branch),
88      .zeroflag(C_Zflag),
89      .branch_out(C_branch_out)
90  );

```

```

91  v Mux2_1_32 M1(
92      .mux_in1(C_mux_i_op2),
93      .mux_in2(C_extend_out),
94      .mux_s(C_AluSrc),
95      .mux_out(C_i_op2)
96  );
97  v Mux2_1_32 M2(
98      .mux_in1(C_suma_out),
99      .mux_in2(C2_suma_out),
100     .mux_s(C_branch_out),
101     .mux_out(C_PC_in)
102  );
103  v Mux2_1_32 M3(
104     .mux_in1(C_Rdata),
105     .mux_in2(C_r_out),
106     .mux_s(C_MemToReg),
107     .mux_out(C_mux_out)
108  );
109  v Mux2_1_5 M4(
110     .mux_in1(C_InsOut[20:16]),
111     .mux_in2(C_InsOut[15:11]),
112     .mux_s(C_RegDst),
113     .mux_out(C2_mux_out)
114  );
115
116  endmodule

```

Unidad de control:

Código:

```
1  `timescale 1ns/1ns
2
3  module UnidadDeControl(
4      input [5:0]op,
5      output reg MemToReg,
6      output reg MemToWrite,
7      output reg [2:0]AluOp,
8      output reg RegWrite,
9      output reg RegDst,
10     output reg Branch,
11     output reg MemRead,
12     output reg AluSrc
13 );
14
15 always @*
16 begin
17     case(op)
18         6'b000000:
19             begin
20                 RegDst = 1;
21                 Branch = 0;
22                 MemRead = 0;
23                 MemToReg = 0;
24                 AluOp = 001;
25                 MemToWrite = 0;
26                 AluSrc = 1;
27                 RegWrite = 1;
28             end
29         endcase
30     end
31
32 endmodule
```

Tabla de instrucciones:

Tipo de instrucción :	RegDs t	Branc h	MemRea d	MemtoRe g	AluO p	Memtowrit e	Alusr c	Regwrit e
R	1	0	0	0	001	0	1	1

ALU:

Código:

```

1  `timescale 1ns/1ns
2
3  module ALU(
4      input [31:0]i_op1,
5      input [31:0]i_op2,
6      input [3:0]Sel,
7      output reg Zflag,
8      output reg [31:0]r_out
9
10 );
11
12 always @*
13 begin
14     case(Sel)
15         4'b0000:
16         begin
17             r_out = i_op1 & i_op2;
18         end
19         4'b0001:
20         begin
21             r_out = i_op1 | i_op2;
22         end
23         4'b0010:
24         begin
25             r_out <= i_op1 + i_op2;
26         end
27         4'b0110:
28         begin
29             r_out = i_op1 - i_op2;
30         end
31         4'b0111:
32         begin
33             r_out = (i_op1 < i_op2) ? 1:0;
34         end

```

```

35         4'b0011:
36     begin
37         r_out = i_op1 * i_op2;
38     end
39     4'b0000:
40     begin
41         r_out = 0;
42     end
43
44     endcase
45     if (r_out>=1)
46     begin
47         Zflag = 1'b0;
48     end
49     else if (r_out<=0)
50     Zflag = 1'b1;
51 end
52
53 endmodule

```

Test Bench ALU:

Código:

```

1  `timescale 1ns/1ns
2  module tb_ALU;
3
4      reg [31:0]op1;
5      reg [31:0]op2;
6      reg [2:0]sel;
7      wire [31:0]rel;
8      wire zeroflag;
9
10     ALU duv(op1,op2,sel,rel,zeroflag);
11
12     initial begin
13         #100;op1=32'd10;op2=32'd10;sel=3'b111;
14         #100;op1=32'd10;op2=32'd10;sel=3'b110;
15         #100;op1=32'd10;op2=32'd10;sel=3'b100;
16         #100;op1=32'd10;op2=32'd10;sel=3'b000;
17         #100;op1=32'd10;op2=32'd10;sel=3'b011;
18         #100;op1=32'd10;op2=32'd10;sel=3'b001;
19         #100;op1=32'd10;op2=32'd10;sel=3'b010;
20         #100;op1=32'd10;op2=32'd10;sel=3'b101;
21         #100;$stop;
22     end
23
24 endmodule

```

ALU Control:

Código:


```

1  `timescale 1ns/1ns
2
3  module AluControl(
4      input [2:0]Aop,
5      input [5:0]Func,
6      output reg [3:0]AluS
7  );
8
9  always @*
10 begin
11     case(Aop)
12         3'b001:
13             begin
14                 case (Func)
15                     6'b100000:
16                         begin
17                             AluS = 4'b0010;
18                         end
19                     6'b100010:
20                         begin
21                             AluS = 4'b0110;
22                         end
23                     6'b100100:
24                         begin
25                             AluS = 4'b0000;
26                         end
27                     6'b101010:
28                         begin
29                             AluS = 4'b0111;
30                         end
31                     6'b100101:
32                         begin
33                             AluS = 4'b0001;
34                         end
35                     6'b011000:

```

```

36         begin
37         |   AluS = 4'b0011;
38         end
39         6'b000000:
40         begin
41         |   AluS = 4'b0000;
42         end
43     endcase
44 end
45
46 endcase
47 end
48
49 endmodule
50

```

Instrucciones:

Instrucción en ensamblador	Entrada 'sel', Módulo ALU	Entrada 'Func', AluControl	Módulo
AND	0000	100100	
OR	0001	100101	
ADD	0010	100000	
SUB	0110	100010	
SLT	0111	101010	
MUL	0011	011000	
NOP	0000	000000	

Banco de registros:

Código:

```
1  `timescale 1ns/1ns
2
3  module BancoReg(
4      input RegWrite,
5      input [4:0]RA1,
6      input [4:0]RA2,
7      input [31:0]WriteData,
8      input [4:0]AW,
9      output reg[31:0]DR1,
10     output reg[31:0]DR2
11 );
12
13 reg [31:0] Reg [0:31];
14
15 initial begin
16     $readmemb("TestF1_BReg",Reg);
17 end
18
19 always @* begin
20
21     if (RegWrite == 1)
22     begin
23         Reg[AW] = WriteData;
24     end
25
26     DR1 <= Reg[RA1];
27     DR2 <= Reg[RA2];
28
29 end
30
31
32
33 endmodule
```

Test Bench Banco de Registros:

```
1  `timescale 1ns/1ns
2  module tb_BR;
3
4      reg [4:0]RA1;
5      reg [4:0]RA2;
6      reg [31:0]Di;
7      reg [4:0]Dir;
8      reg Regw;
9      wire [31:0]DR1;
10     wire [31:0]DR2;
11
12     BR duv(RA1,RA2,Di,Dir,Regw,DR1,DR2);
13
14     initial begin
15         #10;RA1=5'd0;RA2=5'd15;Di=32'd255;Dir=5'd1;Regw=1'b1;
16         #10;RA1=5'd1;RA2=5'd16;Di=32'd356;Dir=5'd2;Regw=1'b1;
17         #10;RA1=5'd2;RA2=5'd17;Di=32'd646;Dir=5'd3;Regw=1'b1;
18         #10;RA1=5'd3;RA2=5'd18;Di=32'd149;Dir=5'd1;Regw=1'b0;
19         #10;RA1=5'd4;RA2=5'd19;Di=32'd506;Dir=5'd1;Regw=1'b0;
20         #10;RA1=5'd5;RA2=5'd20;Di=32'd105;Dir=5'd1;Regw=1'b0;
21         #10;RA1=5'd6;RA2=5'd21;Di=32'd10;Dir=5'd1;Regw=1'b0;
22         #10;RA1=5'd7;RA2=5'd22;Di=32'd856;Dir=5'd1;Regw=1'b0;
23         #10;RA1=5'd8;RA2=5'd23;Di=32'd256;Dir=5'd1;Regw=1'b0;
24         #10;RA1=5'd9;RA2=5'd24;Di=32'd205;Dir=5'd1;Regw=1'b0;
25         #10;$stop;
26     end
27
28 endmodule
```

Datos precargados al Banco de Registros:

[illegible]

Memoria de datos:

```
1  `timescale 1ns/1ns
2
3  module Mem(
4      input MemWrite,
5      input MemRead,
6      input [31:0]Adress,
7      input [31:0]WriteD,
8      output reg[31:0]Rdata
9
10 );
11
12 reg [31:0] T [0:31];
13
14 always @* begin
15     T[Adress] <= WriteD;
16
17     if (MemWrite == 1)
18     begin
19         Rdata <= T[Adress];
20     end
21
22 end
23
24 endmodule
25
```

Memoria de Instrucciones:

Código:

```
1  `timescale 1ns/1ns
2
3  module MemIns(
4      input  [31:0]InsDir,
5      output reg[31:0]InsOut
6  );
7
8      reg [7:0] MemIns [0:399];
9
10     initial begin
11         $readmemb("TestF1_MemInst",MemIns);
12     end
13
14     always @* begin
15         InsOut<={MemIns[InsDir],MemIns[InsDir+1]}
16     end
17
18     endmodule
```

Instrucciones precargadas a la memoria:

```
1  00000000
2  00000001
3  11111000
4  00100000
5  00000000
6  01100100
7  11110000
8  00100010
9  00000000
10 10100110
11 11101000
12 00100100
13 00000000
14 11101000
15 11100000
16 00100101
17 00000001
18 00101010
19 11011000
20 00101010
21 00000001
22 01101011
23 11010000
24 00100110
25 00000001
26 10001101
27 11001000
28 00011010
29 00000001
30 11001111
31 11000000
32 00011000
33 00000010
34 00010001
35 10111000
36 00010000
```

Multiplexor 5 bits:

```
1  `timescale 1ns/1ns
2
3  module Mux2_1_5(
4      input [4:0]mux_in1,
5      input [4:0]mux_in2,
6      input mux_s,
7      output reg[4:0]mux_out
8  );
9
10
11  always @*
12  begin
13      case(mux_s)
14          1'b0:
15              begin
16                  mux_out <= mux_in2;
17              end
18          1'b1:
19              begin
20                  mux_out <= mux_in1;
21              end
22      endcase
23  end
24
25  endmodule
```

Multiplexor de 32 Bits:

```
1  `timescale 1ns/1ns
2
3  module Mux2_1_32(
4      input [31:0]mux_in1,
5      input [31:0]mux_in2,
6      input mux_s,
7      output reg[31:0]mux_out
8  );
9
10
11  always @*
12  begin
13      case(mux_s)
14          1'b0:
15              begin
16                  mux_out <= mux_in2;
17              end
18          1'b1:
19              begin
20                  mux_out <= mux_in1;
21              end
22      endcase
23  end
24
25  endmodule
```


Sumador:

```
1  `timescale 1ns/1ns
2
3  module Sumador(
4      input [31:0]sum_in1,sum_in2,
5      output [31:0]suma_out
6  );
7
8      assign suma_out=sum_in1+sum_in2;
9
10 endmodule
11
```

Sumador constante:

```
1  `timescale 1ns/1ns
2
3  module Sumador4(
4      input [31:0]sum_in1,
5      output [31:0]suma_out
6  );
7
8      assign suma_out=sum_in1+4;
9
10 endmodule
11
```

Sign Extend:

```
1  `timescale 1ns/1ns
2
3  module SignExtend(
4      input [15:0]extend_in,
5      output reg[31:0]extend_out
6  );
7
8      always @*
9  begin
10         extend_out[31:0]<=$signed(extend_in);
11     end
12
13 endmodule
```

Señal de reloj:

```
1  `timescale 1ns/1ns
2
3  module PC(
4      input [31:0]PC_in,
5      input clk,
6      output reg[31:0]PC_out
7  );
8
9      always @(posedge clk)
10  begin
11      PC_out<=PC_in;
12  end
13
14  endmodule
```

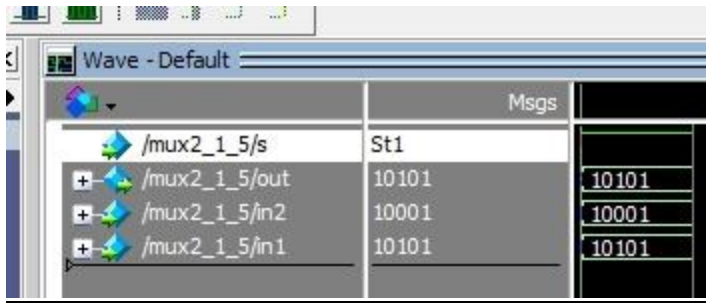
Shift Left:

```
1  `timescale 1ns/1ns
2
3  module ShiftLeft(
4      input [31:0]shift_in,
5      output [31:0]shift_out
6  );
7
8      assign shift_out=shift_in<<2;
9
10  endmodule
```

Branch:

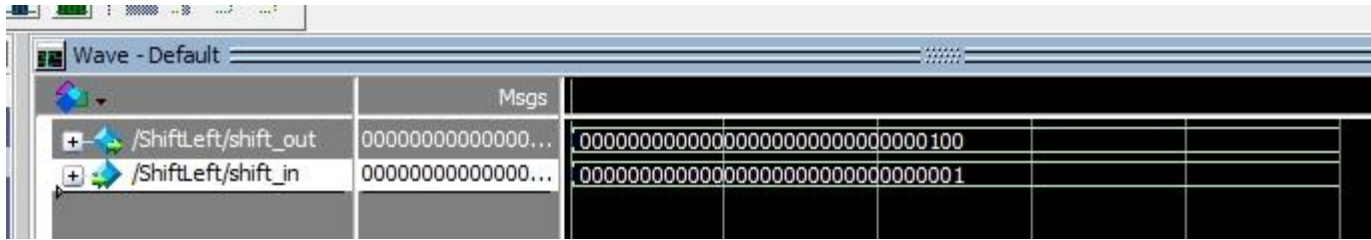
```
1  `timescale 1ns/1ns
2
3  module Branch(
4      input branch,zeroflag,
5      output branch_out
6  );
7
8      assign branch_out = branch & zeroflag;
9
10  endmodule
```

Simulación multiplexor:



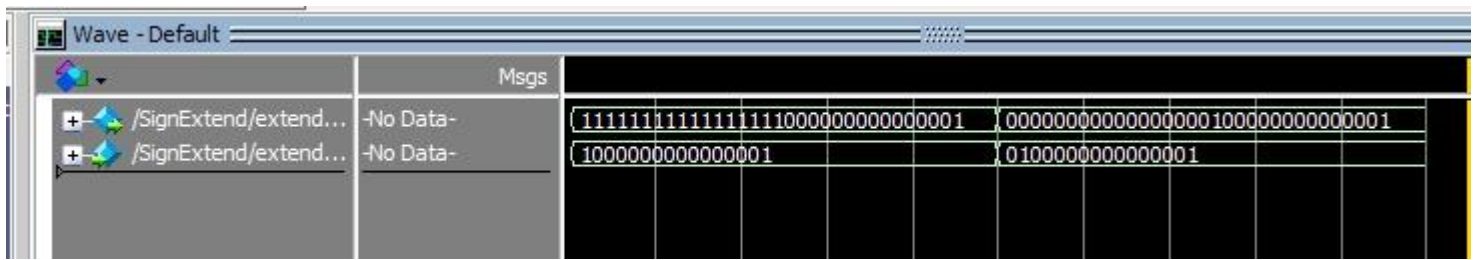
Component	Msgs
/mux2_1_5/s	St1
/mux2_1_5/out	10101
/mux2_1_5/in2	10001
/mux2_1_5/in1	10101

Simulación Shift Left:



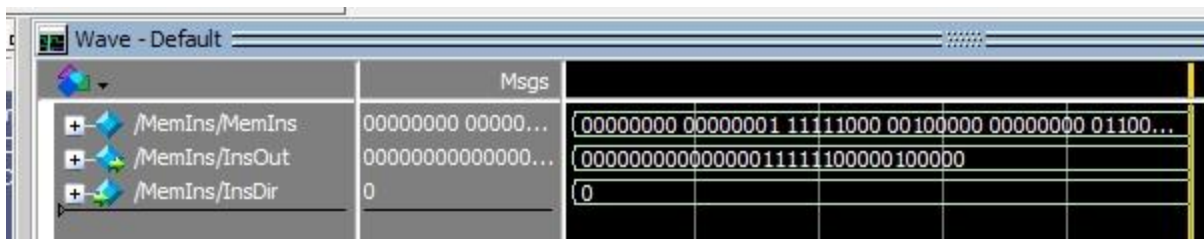
Component	Msgs
/ShiftLeft/shift_out	0000000000000000...
/ShiftLeft/shift_in	0000000000000000...

Simulación Sign Extend:



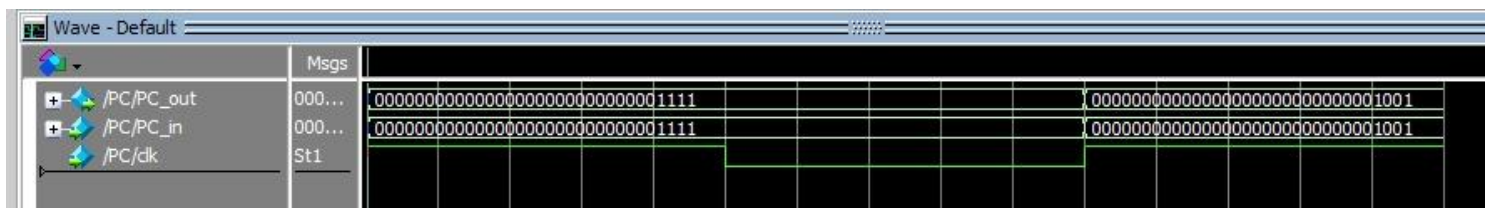
Component	Msgs
/SignExtend/extend...	-No Data-
/SignExtend/extend...	-No Data-

Simulación Memoria de Instrucciones:



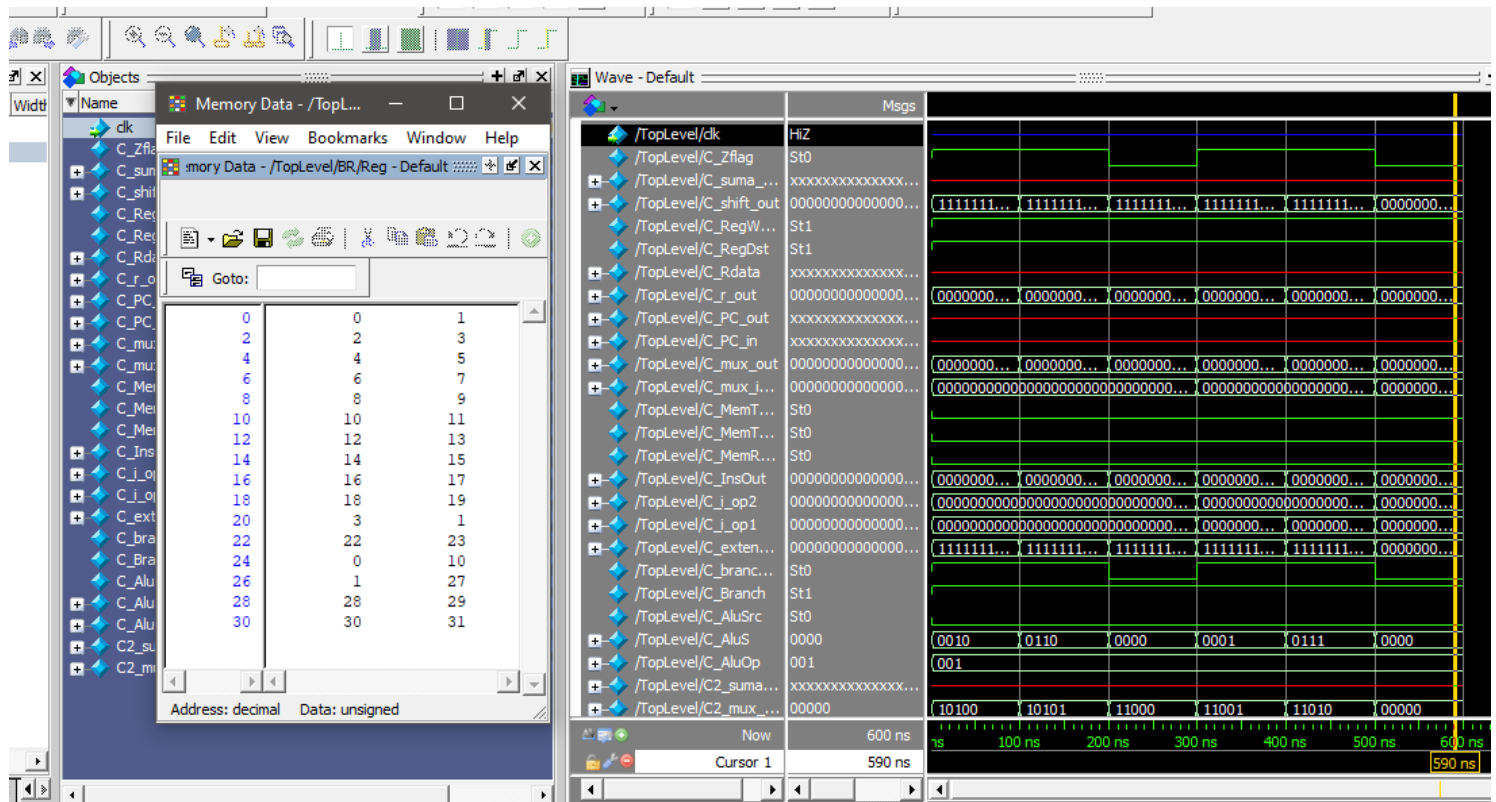
Component	Msgs
/MemIns/MemIns	00000000 00000...
/MemIns/InsOut	0000000000000000...
/MemIns/InsDir	0

Simulación de la señal de reloj:



Component	Msgs
/PC/PC_out	000...
/PC/PC_in	000...
/PC/dk	St1

Simulación del top level:



Conclusión fase 1:

Ángel Damián Raúl García Guevara:

Siguiendo el diseño mostrado en los lineamientos para la fase uno, se implementó cada uno de los módulos dentro del proyecto a lo largo de tres reuniones que se extendieron por 5 días, las minutas de cada reunión están registradas dentro de la Wiki del repositorio con los temas de los cuales se habló durante esas reuniones. Hubo varios inconvenientes a la hora de implementar ciertos módulos y al inicio en la comprensión de cómo funcionaba exactamente el *data path* mostrado, pero a lo largo de las reuniones y con el avance de las clases se respondieron esas dudas. Personalmente considera que la clase respondió bastante bien las dudas que tuvimos a lo largo del desarrollo de la fase 1, algunos aspectos a mejorar por parte mía serán ponerme al corriente con algunos conceptos que sigo sin comprender del todo e involucrarme más dentro del desarrollo de las siguientes fases.

Samuel Espinoza Sucilla

Al principio de del proyecto en mi opinión los tres lo vimos muy grande y de poco tiempo a realizar, ya pasando el tiempo ayude a instanciar y a ver los cables con el compañero de código en llamada para que no se le hiciera tan pesado, aun así tuvimos que checar los códigos por separado después porque aún no estaba bien conectado hasta que hice funcionar algunos elementos del código y el reto aquí está en pasar todas las instrucciones para que funcione el ciclo fetch bien que es lo único que nos faltaría en esta fase de retocar para que quede bien, en cuanto la documentación solo respondí algunas dudas del compañero pero en si no se vio mucho esta parte ya que le dimos más importancia al código y sobre la propuesta para el programa en ensamblador no tenemos algo tan claro pero esperemos la propuesta sea del agrado o que nos propongan una nueva y las diapositivas lo hice lo más sencillo posible para no abarcar mucho tiempo y ser directos con el tema.

Abraham Magaña Hernández

Mi conclusión sobre la clase, bueno, no hay mucho que decir, el profesor es muy bueno siento que he aprendido mucho más en esta clase que es el seminario que en la otra materia relacionada, es muy entretenido, sin embargo como todos he tenido dolores de cabeza a veces con el código, realmente me ha presentado retos, he dado algunas soluciones, el proyecto final es el mejor ejemplo de esto, me toco hacer el código en la fase uno y vaya que ha sido algo que realmente me ha puesto a recurrir a varias personas, dar soluciones revisando horas pero al final se ha logrado superar la mayoría de ellos, por lo cual estoy contento con el trabajo.

Fase 2.

Introducción:

En esta fase se recomienda avanzar desde tres aspectos,

- Código Verilog:

Implementación del “single datapath” de MIPS de 32 bits para ejecutar Instrucciones tipo I . (Debe ejecutar correctamente el archivo de validación adjunto).

a) Debe Completar la tabla 1 y tabla 2 con los tipos de instrucción que se indiquen (agregar 3 instrucciones tipo R, 3 instrucciones tipo I y 1 instrucción tipo J).

- Reporte: Redacción y descripción del desarrollo de los módulos que tienen el datapath y los elementos nuevos para instrucciones tipo I.

- Programa ensamblador:

a) Una vez aprobada la propuesta de algoritmo, comenzar a implementar el algoritmo elegido, Comenzar a trabajar con el decodificador de lenguaje ensamblador a código binario, la final debe crear el archivo a cargar en la memoria de instrucciones (extensión .mem o .txt), y en caso de ser necesario también el archivo de inicialización del Banco de Registro.

- Presentación (subirla a Moodle):

Esta debe ser un resumen de los tres puntos anteriores, se debe de mostrar que han investigado de la parte teorica del algoritmo a implementar en ensamblador, que han codificado en esta fase 2 así como la validación de el datapath para dicha fase.

Desarrollo:

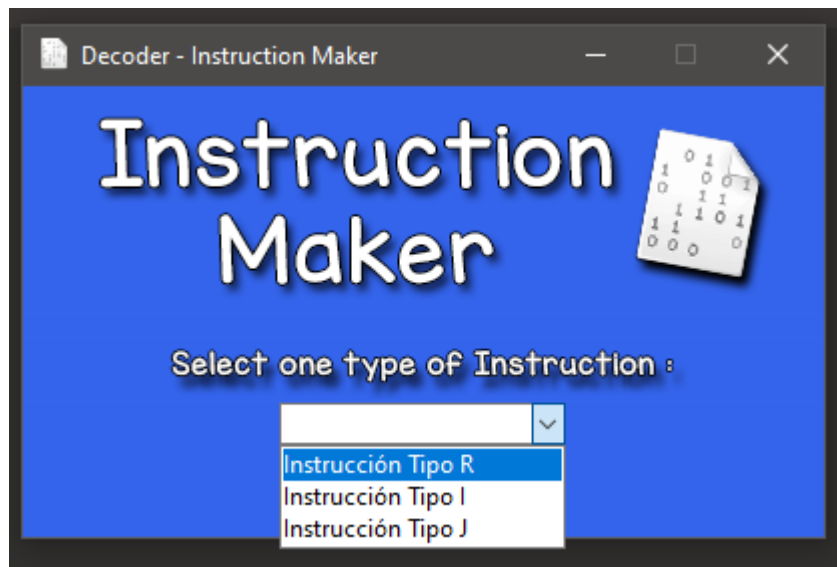
Buffer de datos

Se agregaron 4 nuevos módulos en esta fase llamado buffers el cual a continuación explicare que son y para que sirven.

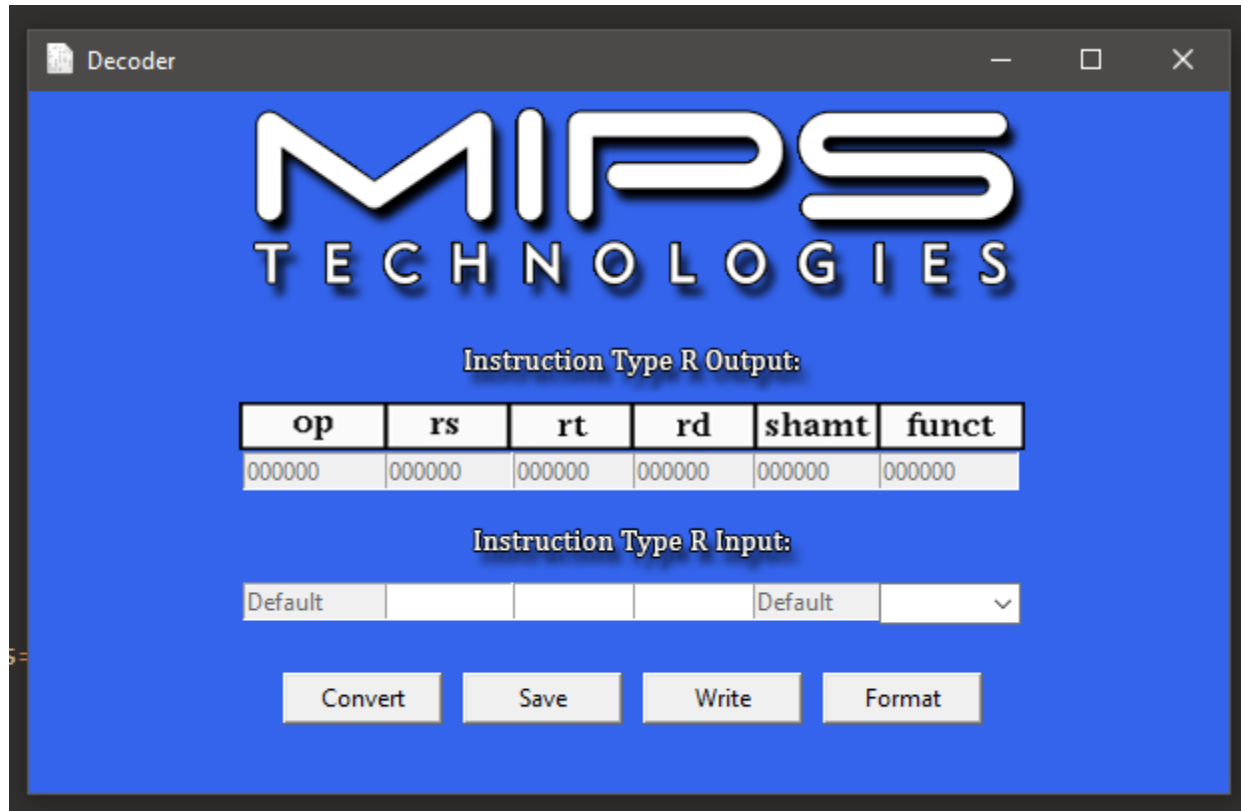
Un búfer es un espacio de [memoria](#), en el que se almacenan datos de manera temporal, normalmente para un único uso (generalmente utilizan un sistema de cola [FIFO](#)); su principal uso es para evitar que el programa o recurso que los requiere, ya sea [hardware](#) o [software](#), se quede sin datos durante una transferencia (entrada/salida) de datos irregular o por la velocidad del proceso.

Normalmente los datos se almacenan en un búfer mientras son transferidos desde un [dispositivo de entrada](#) (como un [ratón](#) o [mouse](#)) o justo antes de enviarlos a un [dispositivo de salida](#) (por ejemplo: [altavoces](#)). También puede utilizarse para transferir datos entre [procesos](#), de una forma parecida a los búferes utilizados en telecomunicaciones. Un ejemplo de esto último ocurre en una comunicación telefónica, en la que al realizar una llamada esta se almacena, se disminuye su calidad y el número de [bytes](#) a ser transferidos, y luego se envían estos datos modificados al receptor.

Se hizo un programa en Python para poder facilitar el hacer las instrucciones dependiendo el tipo que pidamos para el programa en ensamblador:



Se tomó como ejemplo aquí las instrucciones tipo R para poder tener en código maquina el tipo de instrucción:





Estas instrucciones de la unidad de control para lw, sw y beq:

Entrada o salida	Nombre de la señal	Formato R	lw	sw	beq
Entradas	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Salidas	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

Módulos implementados en fase 2:

```
1  | timescale 1ns/1ns
2  | /*Las salidas/salidas de la unidad de control que
3  | se ecuentran aquí vienen todas del bufer "ID_EX" */
4  | module Ex_Mem (
5  |     //entrada de la señal de reloj
6  |     input clk,
7  |     //Entradas de otros modulos
8  |     input [31:00] Add_in,
9  |     input [31:00] ALU_in,
10 |     input [31:00] B2_in,
11 |     input [4:0] Mux_in,
12 |     input ZF_in,
13 |     //Entradas unidad de control
14 |     input ExMem_Branch_in,
15 |     input ExMem_MemWrite_in,
16 |     input ExMem_MemRead_in,
17 |     input ExMem_Regwrite_in,
18 |     input ExMem_MemToReg_in,
19 |     //Salidas de la unidad de control
20 |     output reg ExMem_Branch_out,
21 |     output reg ExMem_MemWrite_out,
22 |     output reg ExMem_MemRead_out,
23 |     output reg ExMem_Regwrite_out,
24 |     output reg ExMem_MemToReg_out,
25 |     //Salidas de otros modulos
26 |     output reg [31:00] Add_out,
27 |     output reg [31:00] ALU_out,
28 |     output reg [31:00] B2_out,
29 |     output reg [4:0] Mux_out,
30 |     output reg ZF_out
31 | );
32 |
33 | initial
34 | begin
35 |     Add_out = 0;
36 |     ALU_out = 0;
37 |     B2_out = 0;
38 |     Mux_out = 0;
39 |     ZF_out = 0;
```

Ex_Mem:

```

40 | ExMem_Branch_out = 0;
41 | ExMem_MemWrite_out = 0;
42 | ExMem_MemRead_out = 0;
43 | ExMem_Regwrite_out = 0;
44 | ExMem_MemToReg_out = 0;
45 | end
46 |
47 | always @(posedge clk)
48 | begin
49 |     Add_out = (Add_in) ? Add_in : 32'b00;
50 |     ALU_out = (ALU_in) ? ALU_in : 32'b00;
51 |     B2_out = (B2_in) ? B2_in : 32'b00;
52 |     Mux_out = (Mux_in) ? Mux_in : 5'b00;
53 |     ZF_out = (ZF_in) ? ZF_in : 1'b0;
54 |     ExMem_MemRead_out = (ExMem_MemRead_in) ? ExMem_MemRead_in : 1'b0;
55 |     ExMem_Branch_out = (ExMem_Branch_in) ? ExMem_Branch_in : 1'b0;
56 |     ExMem_MemWrite_out = (ExMem_MemWrite_in) ? ExMem_MemWrite_in : 1'b0;
57 |     ExMem_Regwrite_out = (ExMem_Regwrite_in) ? ExMem_Regwrite_in : 1'b0;
58 |     ExMem_MemToReg_out = (ExMem_MemToReg_in) ? ExMem_MemToReg_in : 1'b0;
59 | end
60 |
61 | endmodule

```

ID_EX:

```

1 | timescale 1ns/1ns
2 | module ID_EX (
3 |     //Entrada de la señal de reloj
4 |     input clk,
5 |     //Entradas y salidas de otros modulos
6 |     input [31:00] Read_D1_in,
7 |     input [31:00] Read_D2_in,
8 |     input [31:00] Sign_Extend_in,
9 |     input [31:00] PC_adder_in,
10 |    input [4:0] Ins_2016_in,
11 |    input [4:0] Ins_1511_in,
12 |    //Entradas y salidas de la Unidad de control
13 |    input IDEX_MemToReg_in,
14 |    input IDEX_MemToWrite_in,
15 |    input [2:0] IDEX_AluOp_in,

```

```

15     input [2:0] IDEX_AluOp_in,
16     input IDEX_RegWrite_in,
17     input IDEX_RegDst_in,
18     input IDEX_Branch_in,
19     input IDEX_MemRead_in,
20     input IDEX_Alusrc_in,
21     //Salidas de la Unidad de control que van directo hacia módulos
22     output reg IDEX_Alusrc_out,
23     output reg [2:0] IDEX_AluOp_out,
24     output reg IDEX_RegDst_out,
25     //Salidas de la unidad de control que van a los demás búfers
26     output reg IDEX_Branch_out,
27     output reg IDEX_MemRead_out,
28     output reg IDEX_MemToReg_out,
29     output reg IDEX_MemToWrite_out,
30     output reg IDEX_RegWrite_out,
31     //salidas de otros modulos
32     output reg [31:00] Read_D1_out,
33     output reg [31:00] Read_D2_out,
34     output reg [31:00] Sign_Extend_out,
35     output reg [31:00] PC_adder_out,
36     output reg [4:0] Ins_2016_out,
37     output reg [4:0] Ins_1511_out
38 );
39

```

```

40  initial
41  begin
42      //Unidad De Control
43      IDEX_Alusrc_out = 1'b0;
44      IDEX_AluOp_out = 3'b0;
45      IDEX_RegDst_out = 1'b0;
46      IDEX_Branch_out = 1'b0;
47      IDEX_MemRead_out = 1'b0;
48      IDEX_MemToReg_out = 1'b0;
49      IDEX_MemToWrite_out = 1'b0;
50      IDEX_RegWrite_out = 1'b0;
51      //Otros Modulos
52      Read_D1_out = 32'b0;
53      Read_D2_out = 32'b0;
54      Sign_Extend_out = 32'b0;
55      PC_adder_out = 32'b0;
56      Ins_2016_out = 5'b0;
57      Ins_1511_out = 5'b0;
58  end
59
60  always @(posedge clk)
61  begin
62      //Unidad de Control
63      IDEX_AluOp_out = (IDEX_AluOp_in) ? IDEX_AluOp_in : 3'b0;
64      IDEX_Alusrc_out = (IDEX_Alusrc_in) ? IDEX_Alusrc_in : 1'b0;
65      IDEX_RegDst_out = (IDEX_RegDst_in) ? IDEX_RegDst_in : 1'b0;
66      IDEX_Branch_out = (IDEX_Branch_in) ? IDEX_Branch_in : 1'b0;
67      IDEX_MemRead_out = (IDEX_MemRead_in) ? IDEX_MemRead_in : 1'b0;
68      IDEX_MemToReg_out = (IDEX_MemToReg_in) ? IDEX_MemToReg_in : 1'b0;
69      IDEX_MemToWrite_out = (IDEX_MemToWrite_in) ? IDEX_MemToWrite_in : 1'b0;
70      IDEX_RegWrite_out = (IDEX_RegWrite_in) ? IDEX_RegWrite_in : 1'b0;
71      //Resto de modulos
72      Read_D1_out = (Read_D1_in) ? Read_D1_in : 32'b0;
73      Read_D2_out = (Read_D2_in) ? Read_D2_in : 32'b0;
74      Sign_Extend_out = (Sign_Extend_in) ? Sign_Extend_in : 32'b0;
75      PC_adder_out = (PC_adder_in) ? PC_adder_in : 32'b0;
76      Ins_2016_out = (Ins_2016_in) ? Ins_2016_in : 5'b0;
77      Ins_1511_out = (Ins_1511_in) ? Ins_1511_in : 5'b0;
78  end

```

IF_ID:

```
1 | timescale 1ns/1ns
2
3 | module IF_ID (
4 |     input [31:00] PC_Adder_in,
5 |     input [31:00] Mem_Inst_in,
6 |     input clk,
7 |     output reg [31:00] PC_Adder_out,
8 |     output reg [31:00] Mem_Inst_out
9 | );
10
11 | initial
12 | begin
13 |     PC_Adder_out = 0;
14 |     Mem_Inst_out = 0;
15 | end
16
17 | always @(posedge clk)
18 | begin
19 |     PC_Adder_out = (PC_Adder_in) ? PC_Adder_in : 32'b0;
20 |     Mem_Inst_out = (Mem_Inst_in) ? Mem_Inst_in : 32'b0;
21 | end
22
23 | endmodule
```

Mem_WB:

```
1  `timescale 1ns/1ns
2  /*Las salidas/salidas de la unidad de control que
3  se ecuentran aquí vienen todas del bufer "Ex_Mem" */
4  module Mem_WB (
5      //Entrada de la señal de reloj
6      input clk,
7      //Entradas de otros modulos
8      input [31:00] ReadData_in,
9      input [31:00] B3Dir_in,
10     input [4:0] B3Mux_in,
11     //Entradas de la Unidad de control
12     input MemWB_RegWrite_in,
13     input MemWB_MemToReg_in,
14     //salidas de la unidad de control
15     output reg MemWB_RegWrite_out,
16     output reg MemWB_MemToReg_out,
17     //Salidas de otros modulos
18     output reg [31:00] ReadData_out,
19     output reg [31:00] B3Dir_out,
20     output reg [4:0] B3Mux_out
21 );
22
23 initial
24 begin
25     ReadData_out = 0;
26     B3Dir_out = 0;
27     B3Mux_out = 0;
28     MemWB_MemToReg_out = 0;
29     MemWB_RegWrite_out = 0;
30 end
31
32 always @(posedge clk)
33 begin
34     ReadData_out = (ReadData_in) ? ReadData_in : 32'b0;
35     B3Dir_out = (B3Dir_in) ? B3Dir_in : 32'b0;
36     B3Mux_out = (B3Mux_in) ? B3Mux_in : 5'b0;
37     MemWB_RegWrite_out = (MemWB_RegWrite_in) ? MemWB_RegWrite_in : 1'b0;
38     MemWB_MemToReg_out = (MemWB_MemToReg_in) ? MemWB_MemToReg_in : 1'b0;
39 end
```

Unidad de Control .- Hicimos cambios en esta para que funcionaran las instrucciones de tipo I:

```
30 //Addi
31 6'b001000:
32 begin
33     RegDst = 0;
34     Branch = 0;
35     MemRead = 0;
36     MemToReg = 0;
37     AluOp = 3'b000;
38     MemToWrite = 0;
39     AluSrc = 1;
40     RegWrite = 1;
41 end
42 //Andi
43 6'b001100:
44 begin
45     RegDst = 0;
46     Branch = 0;
47     MemRead = 0;
48     MemToReg = 0;
49     AluOp = 3'b011;
50     MemToWrite = 0;
51     AluSrc = 1;
52     RegWrite = 1;
53 end
54 //Ori
55 6'b001101:
56 begin
57     RegDst = 0;
58     Branch = 0;
59     MemRead = 0;
60     MemToReg = 0;
61     AluOp = 3'b100;
62     MemToWrite = 0;
63     AluSrc = 1;
64     RegWrite = 1;
65 end
66 //Slti
67 6'b001010:
68 begin
```

```

67 6'b001010:|
68 begin
69     RegDst = 0;
70     Branch = 0;
71     MemRead = 0;
72     MemToReg = 0;
73     AluOp = 3'b010;
74     MemToWrite = 0;
75     AluSrc = 1;
76     RegWrite = 1;
77 end
78 //Lw
79 6'b100011:
80 begin
81     RegDst = 0;
82     Branch = 0;
83     MemRead = 1;
84     MemToReg = 1;
85     AluOp = 3'b000;
86     MemToWrite = 0;
87     AluSrc = 1;
88     RegWrite = 1;
89 end
90 //Sw
91 6'b101011:
92 begin
93     //RegDst = 1'bx;
94     Branch = 0;
95     MemRead = 1;
96     //MemToReg = 1'bx;
97     AluOp = 3'b000;
98     MemToWrite = 1;
99     AluSrc = 1;
100    RegWrite = 0;
101 end
102 //Beq
103 6'b000100:
104 begin
105    //RegDst = 1'bx;

```



```
102 //Beq
103 6'b000100:
104 begin
105     //RegDst = 1'bx;
106     Branch = 1;
107     MemRead = 0;
108     //MemToReg = 1'bx;
109     AluOp = 3'b101;
110     MemToWrite = 0;
111     AluSrc = 0;
112     RegWrite = 1;
113 end
114 endcase
115 end
116
117 endmodule
118
```

AluControl .- se agregaron en el primer case de la alu control para que agarrara las instrucciones inmediatas:

```
42         end
43     endcase
44 end
45 //lw, Sw, Addi, beq
46 3'b000:
47 begin
48     AluS = 4'b0010;
49 end
50 //slti
51 3'b010:
52 begin
53     AluS = 4'b0111;
54 end
55 //andi
56 3'b011:
57 begin
58     AluS = 4'b0000;
59 end
60 //ori
61 3'b100:
62 begin
63     AluS = 4'b0001;
64 end
65 3'b101:
66 begin
67     AluS = 4'b0110;
68 end
69 default:
70 begin
71 end
72 endcase
73
74 end
75
76 endmodule
77
```

Simulador de buffer::

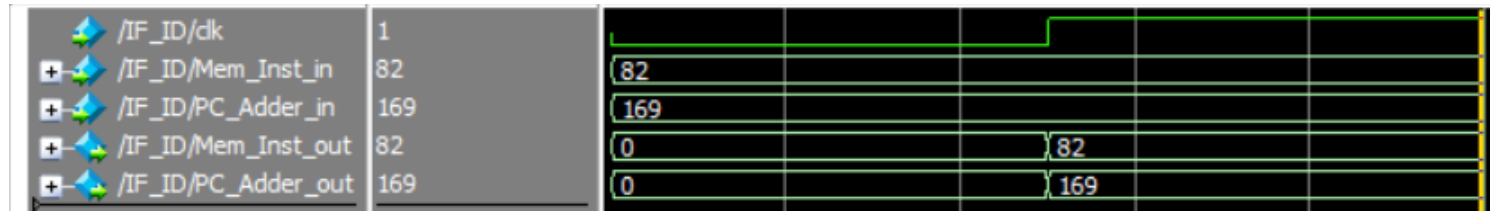
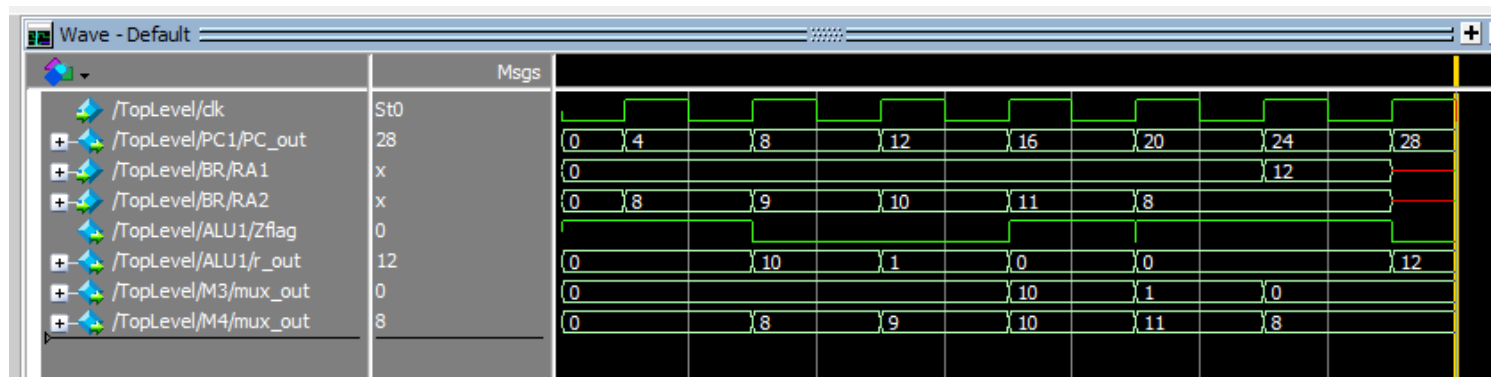


Tabla de resultados:

PC_OUT	Rd1	Rd2	ALUR	ZFLAG	C_MUX_OUT	ALU MUX
12	10	8	10	0	10	10
16	18	18	18	0	18	8
20	18	9	0	1	0	10
24	0	0	0	1	0	9
28	0	10	0	1	0	10
32	0	0	0	1	0	10
36	0	11	10	0	10	10

Simulación con la memoria de instrucciones que el profesor nos proporcione:



Banco de registros

Memoria de datos

Memory Data - /TopLevel/BR/Reg		Memory Data - /TopLevel/Mem1/Memoria	
0	0	0	10
1	1	1	x
2	2	2	x
3	3	3	x
4	4	4	x
5	5	5	x
6	6	6	x
7	7	7	x
8	10	8	x
9	1	9	x
10	0	10	x
11	0	11	x
12	12	12	10
13	13	13	x
14	14	14	x
15	15	15	x
16	16	16	x
17	17	17	x
18	18	18	x
19	19	19	x
20	20	20	x
21	21	21	x
22	22	22	x
23	23	23	x
24	24	24	x
25	25	25	x
26	26	26	x
27	27	27	x
28	28	28	x
29	29	29	x
30	30	30	x

Conclusión fase 2:

Ángel Damián Raúl García Guevara:

Siguiendo el diseño mostrado en los lineamientos para la fase uno, se implementó cada uno de los módulos dentro del proyecto a lo largo de tres reuniones que se extendieron por 5 días, las minutas de cada reunión están registradas dentro de la Wiki del repositorio con los temas de los cuales se habló durante esas reuniones. Hubo varios inconvenientes a la hora de implementar ciertos módulos y al inicio en la comprensión de cómo funcionaba exactamente el *data path* mostrado, pero a lo largo de las reuniones y con el avance de las clases se respondieron esas dudas. Personalmente considera que la clase respondió bastante bien las dudas que tuvimos a lo largo del desarrollo de la fase 1, algunos aspectos a mejorar por parte mía serán ponerme al corriente con algunos conceptos que sigo sin comprender del todo e involucrarme más dentro del desarrollo de las siguientes fases.//

Samuel Espinoza Sucilla

Al principio de del proyecto en mi opinión los tres lo vimos muy grande y de poco tiempo a realizar, ya pasando el tiempo ayude a instanciar y a ver los cables con el compañero de código en llamada para que no se le hiciera tan pesado, aun así tuvimos que checar los códigos por separado después porque aún no estaba bien conectado hasta que hice funcionar algunos elementos del código y el reto aquí está en pasar todas las instrucciones para que funcione el ciclo fetch bien que es lo único que nos faltaría en esta fase de retocar para que quede bien, en cuanto la documentación solo respondí algunas dudas del compañero pero en si no se vio mucho esta parte ya que le dimos más importancia al código y sobre la propuesta para el programa en ensamblador no tenemos algo tan claro pero esperemos la propuesta sea del agrado o que nos propongan una nueva y las diapositivas lo hice lo más sencillo posible para no abarcar mucho tiempo y ser directos con el tema.//

Abraham Magaña Hernández

Mi conclusión sobre la clase, bueno, no hay mucho que decir, el profesor es muy bueno siento que he aprendido mucho más en esta clase que es el seminario que en la otra materia relacionada, es muy entretenido, sin embargo como todos he tenido dolores de cabeza a veces con el código, realmente me ha presentado retos, he dado algunas soluciones, el proyecto final es el mejor ejemplo de esto, me toco hacer el código en la fase uno y vaya que ha sido algo que realmente me ha puesto a recurrir a varias personas, dar soluciones revisando horas pero al final se ha logrado superar la mayoría de ellos, por lo cual estoy contento con el trabajo.//

Bibliografía:

- Patterson, D. A., & Henessy, J. L. (2014). Computer Organization And Design (5.a ed., Vol. 1) [Libro electrónico]. Horgan Kaufman. https://doc-00-bg-apps-viewer.googleusercontent.com/viewer/secure/pdf/r6aeht2l0quge2g96vkq3nuggr576sqk/jt9usvab8i8oq50kb86sralemcn8joa6/1622010000000/drive/01214495260069733672/ACFrOgB2E-ncD1gQ3x100pQlaaqyhwbxsREj5k5z8k_W5MowpA2VKMvc_vPRg8CYmps3Tr_vNmzypF0BcKNESqZGe4vCS3pEtg1v8YCDi-hN7hgZd9vrad2UZAKBSv-ky3FSmtC8IHWZwd53fH8P?print=true&nonce=llh5g91227cis&user=01214495260069733672&hash=9okhdvmof3n4msqlhue29oous7cijejd
- Hernández Cerezo, A., Universidad de Valladolid, & Tejedor García, C. (s. f.). Arquitectura MIPS. Arquitectura MIPS. Recuperado 26 de mayo de 2021, de https://www.infor.uva.es/~bastida/OC/TRABAJO1_MIPS.pdf
- MIPS Technologies. (s. f.). The MIPS32® Instruction Set Manua (Revisado ed., Vol. 6) [Libro electrónico]. MIPS Technologies. https://www.cs.cornell.edu/courses/cs3410/2008fa/MIPS_Vol2.pdf