Arquitectura de Computadoras
LAB 6 parte 2

Docente: Jorge Gonzales Reaño

Integrantes:

Nombre y apellidos	Código	Correo
Jose Leandro Machaca	202110202	jose.machaca.s@utec.edu,pe
Diego Pacheco Ferrell	202010159	diego.pacheco@utec.edu.pe
Dimael Rivas Chavez	202110307	dimael.rivas@utec.edu.pe
Anderson Carcamo Vargas	202020025	anderson.carcamo@utec.edu.pe

Cycle	Reset	PC	Instr	(FSM) state	SrcA	SrcB	ALUResult
1	1	00	0	FETCH	0	4	4
2	0	04	SUB E04F000F	DECODE	4	4	8
3	0	04		EXECUTER	8	8	0
4	0	04		ALUWB	x	x	x
5	0	04		FETCH	4	4	8
6	0	08	ADD E2802005	DECODE	8	4	C
7	0	08		EXECUTEI	0	5	5
8	0	08		ALUWB	x	x	x
9	0	08		FETCH	8	4	C
10	0	12	ADD E280300C	DECODE	C	4	10
11	0	12		EXECUTEI	0	C	C
12	0	12		ALUWB	x	x	x
13	0	12		FETCH	C	4	10
14	0	10	SUB E2437009	DECODE	10	4	14
15	0	10		EXECUTEI	C	9	3
16	0	10		ALUWB	C	X	X
17	0	10		FETCH	10	4	14
18	0	14	ORR E1874002	DECODE	14	04	18
19	0	14		EXECUTER	3	5	7
20	0	14		ALUWB	3	5	8
21	0	14		FETCH	14	4	18
22	0	18	AND E0035004	DECODE	18	4	1C
23	0	18		EXECUTER	C	7	4
24	0	18		ALUWB	C	7	13
25	0	18		FETCH	18	4	1C
26	0	1C	ADD E0855004	DECODE	1C	4	20
27	0	1C		EXECUTER	4	7	B
28	0	1C		ALUWB	4	7	B
				FETCH	1C	4	20
29	0	20	SUBS	DECODE	20	4	24

			E0558007				
30	0	20		EXECUTER	B	3	8
31	0	20		ALUWB	B	3	E
32	0	20		FETCH	20	4	24
33	0	24	BEQ 0A00000C	DECODE	24	4	28
34	0	24		BRANCH	28	30	58
35	0	24		FETCH	24	4	28
37	0	28	SUBS E0538004	DECODE	28	4	2C
38	0	28		EXECUTER	C	7	5
39	0	28		ALUWB	C	7	13
40	0	28		FETCH	28	4	2C
41	0	2C	BGE AA000000	DECODE	2C	4	30
42	0	2C		BRANCH	30	0	30
44	0	2C		FETCH	30	4	34
45	0	30	SUBS E0578002	DECODE	34	4	38
46	0	30		EXECUTER	3	5	FFFFFFFE
47	0	30		ALUWB	3	5	8
48	0	30		FETCH	34	4	8
49	0	34	ADDLT B2857001	DECODE	38	4	3C
50	0	34		EXECUTEI	B	1	C
51	0	34		ALUWB	B	X	X
52	0	34		FETCH	38	4	3C
53	0	38	SUB E0477002	DECODE	3C	4	40
54	0	38		EXECUTER	C	5	7
55	0	38		ALUWB	C	5	11
56	0	38		FETCH	3C	4	40
57	0	3C	STR E5837054	DECODE	40	4	44
58	0	3C		MEMADR	C	54	60
59	0	3C		MEMWRITE	C	7	13
60	0	3C		FETCH	40	4	44
61	0	40	LDR E5902060	DECODE	44	4	48

62	0	40		MEMADR	0	60	60
63	0	40		MEMREAD	0	5	5
64	0	40		MEMWB	0	5	5
64	0	40		FETCH	44	4	48
65	0	44	ADD E08FF000	DECODE	48	4	4C
66	0	44		EXECUTER	4C	0	4C
67	0	44		ALUWB	4C	0	4C
68	0	44		FETCH	4C	4	50
69	0	48	B EA000001	DECODE	50	4	54
70	0	48		BRANCH	54	4	58
71	0	48		FETCH	58	4	5C
73	0	4C	STR E5802064	DECODE	5C	4	60
74	0	4C		MEMADR	0	64	64
74	0	4C		MEMWRITE	0	7	7

Table 1. Expected Instruction Trace

Datapath:

Cabe mencionar que el bloque mem no se instancia dentro del datapath, debido a que este ya se encuentra dentro del módulo top.

```
module datapath (

clk,

reset,

Adr,

WriteData,

ReadData,

Instr,

ALUFlags,

PCWrite,

RegWrite,

IRWrite,

AdrSrc,

RegSrc,
```

```
ALUSrcA,
ALUSrcB,
ResultSrc,
ImmSrc,
ALUControl
);
input wire clk;
input wire reset;
output wire [31:0] Adr;
output wire [31:0] WriteData;
input wire [31:0] ReadData;
output wire [31:0] Instr;
output wire [3:0] ALUFlags;
input wire PCWrite;
input wire RegWrite;
input wire IRWrite;
input wire AdrSrc;
input wire [1:0] RegSrc;
input wire [1:0] ALUSrcA;
input wire [1:0] ALUSrcB;
input wire [1:0] ResultSrc;
input wire [1:0] ImmSrc;
input wire [1:0] ALUControl;
wire [31:0] PCNext;
wire [31:0] PC;
wire [31:0] ExtImm;
wire [31:0] SrcA;
wire [31:0] SrcB;
wire [31:0] Result;
wire [31:0] Data;
wire [31:0] RD1;
wire [31:0] RD2;
```

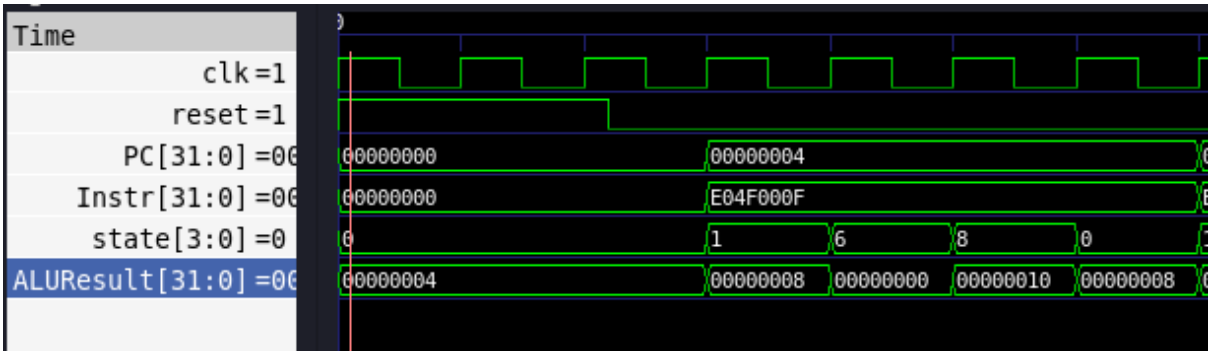
```
wire [31:0] A;
wire [31:0] ALUResult;
wire [31:0] ALUOut;
wire [3:0] RA1;
wire [3:0] RA2;

parameter reg15 = 4'b1111;
parameter const4 = 4;

flopenr #(32) PCreg(
.clk(clk),
.reset(reset),
.en(PCWrite),
.d(Result),
.q(PC)
);

mux2 #(32) ADRmux(
.d0(PC),
.d1(Result),
.s(AdrSrc),
.y(Adr)
);

flopenr #(32) REGinstr(
.clk(clk),
.reset(reset),
.en(IRWrite),
.d(ReadData),
.q(Instr)
);
flopr #(32) REGdata(
```

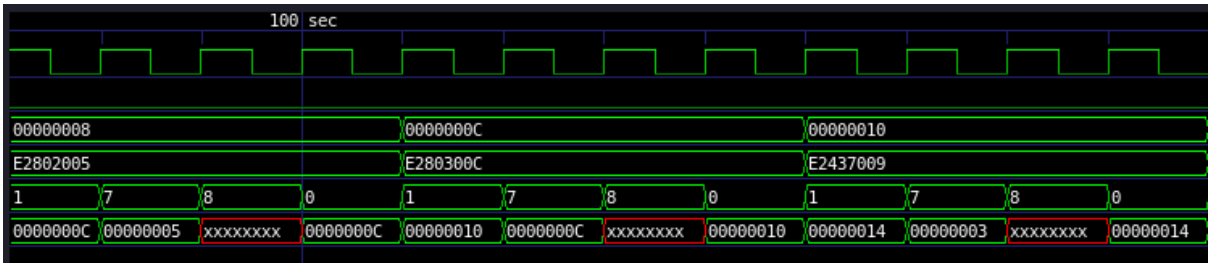
```
.clk(clk),
.reset(reset),
.d(ReadData),
.q(Data)
);
mux2 #(4) RA1mux(
.d0(Instr[19:16]),
.d1(reg15),
.s(RegSrc[0]),
.y(RA1)
);
mux2 #(4) RA2mux(
.d0(Instr[3:0]),
.d1(Instr[15:12]),
.s(RegSrc[1]),
.y(RA2)
);
extend BloqueExtend(
.Instr(Instr[23:0]),
.ImmSrc(ImmSrc),
.ExtImm(ExtImm)
);
regfile RegisterFile(
.clk(clk),
.we3(RegWrite),
.ra1(RA1),
.ra2(RA2),
.wa3(Instr[15:12]),
.wd3(Result),
.r15(Result),
.rd1(RD1),
.rd2(RD2)
```

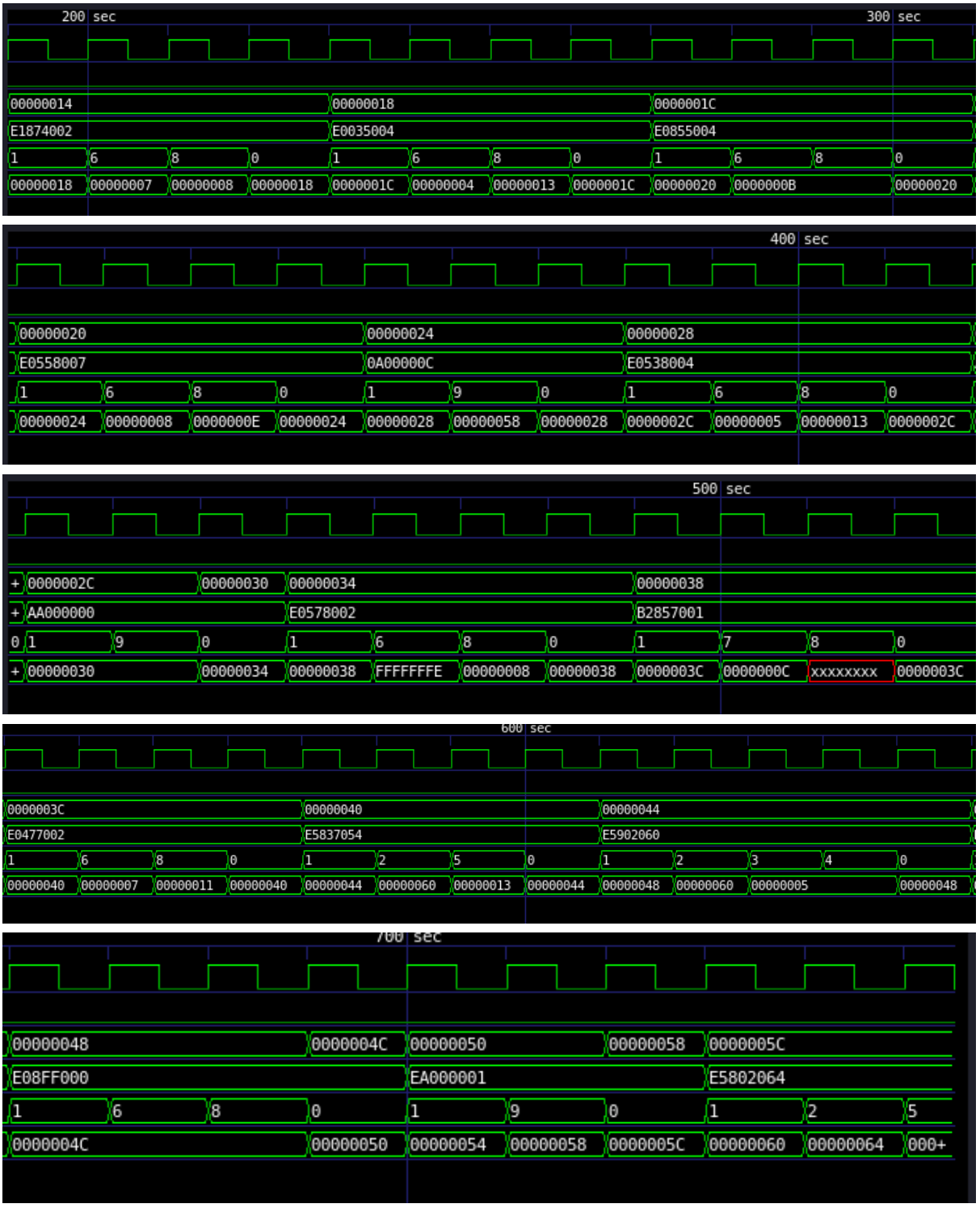
```
);
flopr #(32) RegisterA1(
.clk(clk),
.reset(reset),
.d(RD1),
.q(A)
);
flopr #(32) RegisterA0(
.clk(clk),
.reset(reset),
.d(RD2),
.q(WriteData)
);
mux3 #(32) SrcAmux(
.d0(A),
.d1(PC),
.d2(ALUOut),
.s(ALUSrcA),
.y(SrcA)
);
mux3 #(32) SrcBmux(
.d0(WriteData),
.d1(ExtImm),
.d2(const4),
.s(ALUSrcB),
.y(SrcB)
);
alu BloqueAlu(
.SrcA(SrcA),
.SrcB(SrcB),
.ALUControl(ALUControl),
.ALUResult(ALUResult),
```

```
.ALUFlags(ALUFlags)

);

flopr #(32) ALUOUTreg(

.clk(clk),

.reset(reset),

.d(ALUResult),

.q(ALUOut)

);

mux3 #(32) ResultMux(

.d0(ALUOut),

.d1(Data),

.d2(ALUResult),

.s(ResultSrc),

.y(Result)

);

endmodule
```

Waveforms:







Un aspecto importante que mencionar es en el proceso de debugging nos encontramos con 2 problemas:

1. El programa indicaba “Simulation Failed”.
   Solución: Dentro del controller, para las señales habían ciertos errores en el output logic (controls), ya que las señales que habíamos dejado prendidas, no se cambiaban si el estado actual no requería su cambio. Es por eso que cambiamos a 0 a las señales que no cambiaban. Este es el nuevo output logic:

FETCH: controls = 13'b1000101001100;
DECODE: controls = 13'b0000001001100;
EXECUTER: controls = 13'b0000000000001;
EXECUTEI: controls = 13'b0000000000011;
ALUWB: controls = 13'b0001000000000;
MEMADR: controls = 13'b0000000000010;
MEMWRITE:controls = 13'b0010010000000;
MEMREAD: controls = 13'b0000010000000;
MEMWB: controls = 13'b0001000100000;
BRANCH: controls = 13'b0100001010010;

2. El waveform no mostraba el último estado de la última instrucción.
   Solución: cambiar el $Stop por un $finish