LAB 9- ARCH - 2020-I

Integrantes: Códigos:

- Calixto Rojas, Neftali 201910092
- Lapa Romero, Julisa 201910200

1. Indicaciones:

1. Fecha de entrega: 21 de Julio.
2. Entregables: Informe y archivos todo en un comprimido '`.tar`'.

2. Resumen de Contenidos:

Este informe corresponde al Proyecto Final del curso de Arquitectura de Computadores, consta de tres partes, MIPS Single-Cycle Processor, MIPS Multi-Cycle Processor 1 y MIPS Multi-Cycle Processor 2.

Esta es la tercera parte, donde completamos nuestro propio MIPS Multi-Cycle Processor(basado en el diseño de el Harris Harris). Para este laboratorio nos referiremos al laboratorio anterior. En el laboratorio 8, creamos el control unit. En este laboratorio diseñaremos el datapath, unidades mem y testear tu multicycle anterior.

Esta vez, usaremos el mismo testbench y partes generales del MIPS Single-Cycle. Copiaremos nuestro archivo mipstest.v, mipsparts.v y el alu_xx.sv del laboratorio 7 a nuestra carpeta Lab9_xx. Les daremos un vistazo general a los archivos y veremos cómo trabajan.

3. Testeando el Programa:

Usaremos el archivo `memfile.dat` del laboratorio 7. Cómo en el lab7, será muy útil predecir los resultados del testeo antes de correrlo para poder descubrir y hacer seguimiento de errores o discrepancias. La **Tabla 1**, que está parcialmente completada, lista el rastreo de las instrucciones esperadas mientras corre el testeo. Completar el resto de la tabla. Hacer esto antes de ejecutar las simulaciones, para tener un grupo de datos

esperados que comparar con los resultados obtenidos; de otra forma, es fácil engañarse creyendo que las simulaciones erróneas son correctas.

Notar que la instrucción(`instr`) es obtenida durante el state 0 y por ello no se actualiza hasta el state 1 de cada instrucción.

Cuando el ALUResult no sea usado(e.g. en el estado Decode de una instrucción nonbranch, o el estado Writeback de cualquier instrucción), se pondrá una 'x' para los don't care en vez de predecir valores no utilizados por el procesador al simular.

4. Diseño del Datapath:

Referimos la Figura 1 en el Laboratorio 8 para los módulos de hardware que necesitaremos hacer para el datapath. Diseñar la unidad Datapath en Verilog.

Recordar que debemos reusar hardware de anteriores laboratorios (e.g. ALU, multiplexores, registers, sign-extension hardware modules, register file, etc) donde sea posible.

Todos los registers deben tomar un *Reset* input para reiniciarse del valor inicial a un estado conocido (0). El Instruction Register y PC también requerirán inputs de enable.

Ser cuidadosos a las conexiones del bus, son un buen lugar para cometer errores.

Simular el procesador usando el testbench brindado (`mipstest.v`). La señal de reset está encendida al inicio. Mostrar, cuando esté apagada o al mínimo, el PC, Instr, FSM state(dentro del módulo `controller`), SrcA y SrcB (dentro del datapath), ALUResult, Zero y el Control Word. Verificar que los resultados coincidan con lo predicho en la Table 1. Si hay discrepancias, revise el diseño y arregle los errores.

Cuando esto haya terminado...~~FELICIDADES~~...Han construído un microprocesador por tu cuenta y has probado su capacidad en microarquitectura, Verilog, FSMs y diseño lógico.

5. Entregables:

Presentar los siguientes elementos **en el siguiente orden.** Detallar el orden.

- Una versión completa de la Tabla 1 indicando la salida esperada al correr el programa.
- Todos los códigos en Verilog del Datapath.
- Waveforms del procesador, mostrando (en el orden dado): *CLK*, *Reset*, *state*, *PC*, *Funct*, *ALUResult*. Mostrar todas las señales en hexadecimal o decimal, pero asegurar se que se legible, **¿concuerdan con las predicciones?, ¿El programa indica Simulation Succeeded?**

DESARROLLO:

1. Tabla 1: Expected Instruction Trace

Cycle	Reset	pc	instr	(FSM) state	srca	srcb	ALUResult	zero	Control Word
1	1	00	0	0	00	04	04	0	5010
2	0	04	addi 20020005	1	04	x	x	0	0030
3	0	04	addi 20020005	9	00	05	05	0	0420
4	0	04	addi 20020005	10	x	x	x	0	0800
5	0	04	addi 20020005	0	04	04	08	0	5010
6	0	08	addi 200300c	1	08	x	x	0	0030
7	0	08	addi 200300c	9	00	12	12	0	0420
8	0	08	addi 200300c	10	x	12	x	0	0800
9	0	08	addi 200300c	0	00	x	x	0	5010
10	0	12	addi 20067fff7	1	00	12	-24	0	0030
11	0	12	addi 20067fff7	9	12	x	x	0	0420
12	0	12	addi 20067fff7	10	12	x	x	x	0800
13	0	12	addi 20067fff7	0	12	x	x	0	5010
14	0	16	or 00e22025	1	12	3	32932	0	0030
15	0	16	or 00e22025	6	3	5	7	0	0402
16	0	16	or 00e22025	7	3	5	21	0	0840
17	0	16	or 00e22025	0	3	5	20	0	5010
18	0	20	and 00642824	1	3	5	41124	0	0030
19	0	20	and 00642824	6	12	7	4	0	0402
20	0	20	and 00642824	7	12	7	27	0	0840
21	0	20	and 00642824	0	12	7	24	0	5010
22	0	24	add 00a42820	1	12	7	41112	0	0030

23	0	24	add 00a42820	6	4	7	11	0	0402
24	0	24	add 00a42820	7	4	7	31	0	0840
25	0	24	add 00a42820	0	4	7	28	0	5010
26	0	28	beq 10a7000a	1	11	7	68	0	0030
27	0	28	beq 10a7000a	8	11	3	8	0	0605
28	0	28	beq 10a7000a	0	11	3	32	0	5010
29	0	32	slt 0064202a	1	11	3	32968	0	0030
30	0	32	slt 0064202a	6	12	7	0	1	0402
31	0	32	slt 0064202a	7	12	7	39	0	0840
32	0	32	slt 0064202a	0	12	7	36	0	5010
33	0	36	beq 10800001	1	12	0	40	0	0030
34	0	36	beq 10800001	8	0	0	0	1	0605
35	0	36	beq 10800001	0	0	0	44	0	5010
36	0	44	slt 00e2202a	1	0	0	32980	0	0030
37	0	44	slt 00e2202a	6	3	5	1	0	0402
38	0	44	slt 00e2202a	7	3	5	49	0	0840
39	0	44	slt 00e2202a	0	3	5	48	0	5010
40	0	48	add 00853820	1	3	5	57520	0	0030
41	0	48	add 00853820	6	1	11	12	0	0402
42	0	48	add 00853820	7	1	11	59	0	0840
43	0	48	add 00853820	0	1	11	52	0	5010
44	0	52	sub 00e23822	1	1	11	57532	0	0030
45	0	52	sub 00e23822	6	12	5	7	0	0402
46	0	52	sub 00e23822	7	12	5	57	0	0840
47	0	52	sub 00e23822	0	12	5	56	0	5010
48	0	56	sw ac670044	1	7	5	328	0	0030
49	0	56	sw ac670044	2	12	7	80	0	0420
50	0	56	sw ac670044	5	12	7	63	0	2100
51	0	56	sw ac670044	0	12	7	60	0	5010
52	0	60	lw 8c020050	1	12	7	380	0	0030
53	0	60	lw 8c020050	2	0	5	80	0	0420
54	0	60	lw 8c020050	3	0	5	65	0	0100
55	0	60	lw 8c020050	4	0	5	65	0	0880

56	0	60	lw 8c020050	0	0	5	64	0	5010
57	0	64	j 08000011	1	0	7	132	0	0030
58	0	64	j 08000011	11	0	0	64	0	4008
59	0	64	j 08000011	0	0	0	72	0	5010
60	0	72	sw ac020054	1	0	0	408	0	0030
61	0	72	sw ac020054	2	0	7	84	0	0420
62	0	72	sw ac020054	5	0	7	79	0	2100

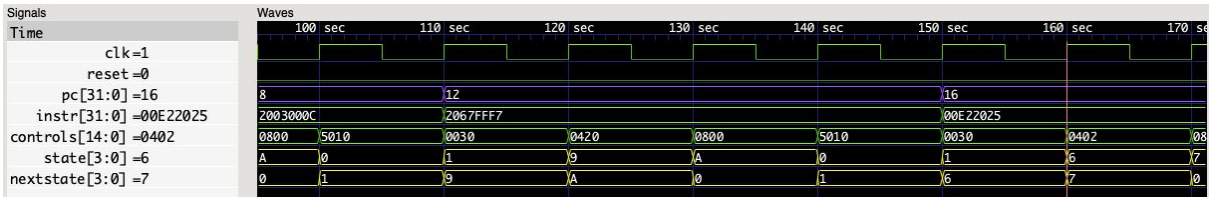
2. CÓDIGOS EN VERILOG

➔ DATAPATH:

```
// it is composed of instances of its sub-modules.  For example,
// the instruction register is instantiated as a 32-bit flopenr.
// The other submodules are likewise instantiated.

module datapath(input          clk, reset,
                input          pcen, irwrite, regwrite,
                input          alusrca, iord, memtoreg, regdst,
                input   [1:0]  alusrcb, pcsrc,
                input   [3:0]  alucontrol,
                output  [5:0]  op, funct,
                output         zero,
                output  [31:0] adr, writedata,
                input   [31:0] readdata);

  // Below are the internal signals of the datapath module.

  wire [4:0]  writereg;
  wire [31:0] pcnext, pc;
  wire [31:0] instr, data, srca, srcb;

  wire [31:0] A ,B;
  wire [31:0] aluresult, aluout;
  wire [31:0] signimm;   // the sign-extended immediate
  wire [31:0] signimmsh;        // the sign-extended immediate shifted left by 2
  wire [31:0] wd3, rd1, rd2;
  wire [31:0] pcjump;

  // op and funct fields to controller
  assign op = instr[31:26];
  assign funct = instr[5:0];
  assign writedata=B;
  assign pcjump={pc[31:28],{pc[25:0],2'b00}};
  // Your datapath hardware goes below.  Instantiate each of the submodules
  // that you need.  Remember that alu's, mux's and various other
  // versions of parameterizable modules are available in mipsparts.sv
  // from Lab 9. You'll likely want to include this verilog file in your
  // simulation.
  flopenr #(32)         pcreg(clk, reset, pcen, pcnext, pc);
  mux2 #(32)            muxfetch(pc, aluout, iord, adr);

  flopenr #(32)         intrdecode(clk, reset, irwrite, readdata, instr);
  flopr #(32)           rdata(clk, reset, readdata, data);
  mux2 #(5)             muxrt(instr[20:16], instr[15:11], regdst, writereg);
  mux2 #(32)            wdmux(aluout, data, memtoreg, wd3);

  regfile               rf(clk, regwrite, instr[25:21], instr[20:16], writereg, wd3, rd1, rd2);
  signext               sign(instr[15:0], signimm);

  floprx2 #(32)         readata12(clk,reset,rd1,rd2,A,B);
  //assign                  writedata=B;

  sl2                   sll2(signimm, signimmsh);
  mux4 #(32)            mux4x1(B, 32'b100, signimm, signimmsh, alusrcb, srcb);
  mux2 #(32)            muxsrca(pc, A, alusrca, srca);

  Alu                   alu(.A(srca), .B(srcb), .AluOp(alucontrol), .Result(aluresult), .zero(zero));
```

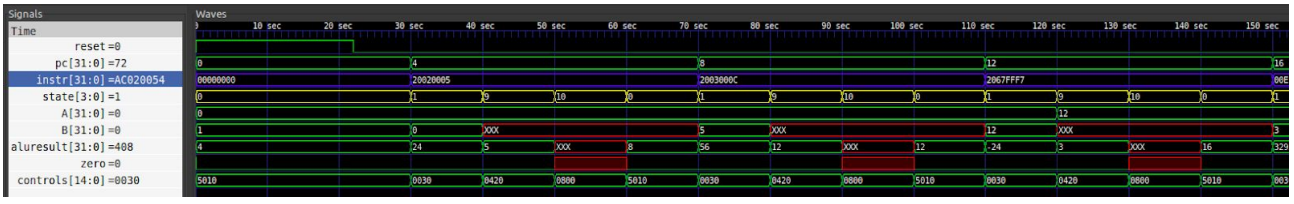
```
237
238   Alu                   alu(.A(srca), .B(srcb), .AluOp(alucontrol), .Result(aluresult), .zero(zero));
239
240   //assign                pcjump={pc[31:28],pc[25:0],2'b00};
241   flopr #(32)           alures(clk, reset, aluresult, aluout);
242
243   mux3 #(32)            mux3x1(aluresult, aluout, pcjump, pcsrc, pcnext);
244
245   // We've included parameterizable 3:1 and 4:1 muxes below for your use.
246
247   // Remember to give your instantiated modules applicable names
248   // such as pcreg (PC register), wdmux (Write Data Mux), etc.
249   // so it's easier to understand.
250
251   // ADD CODE HERE
252
253   // datapath
254
255 endmodule
```

3. WAVEFORMS:

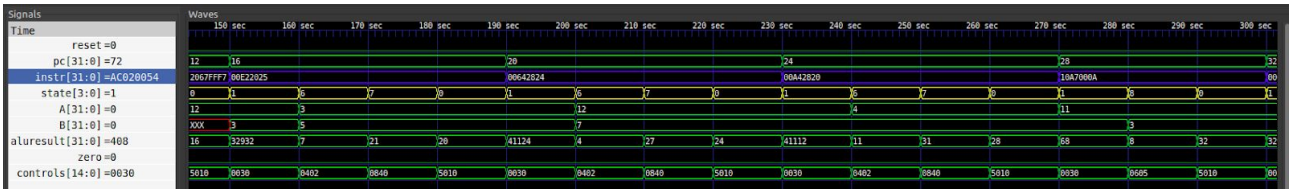


->Waveforms usados para llenar la Tabla1.

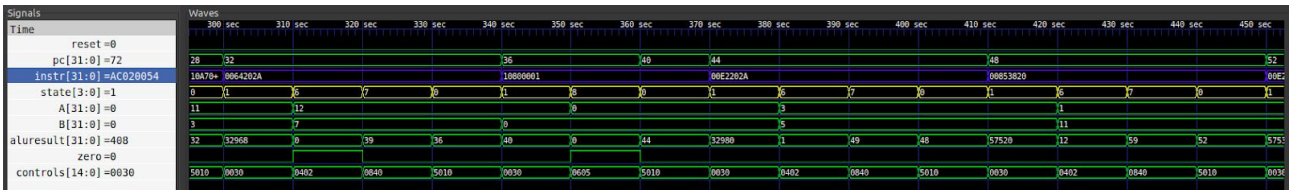
[0, 150sec]



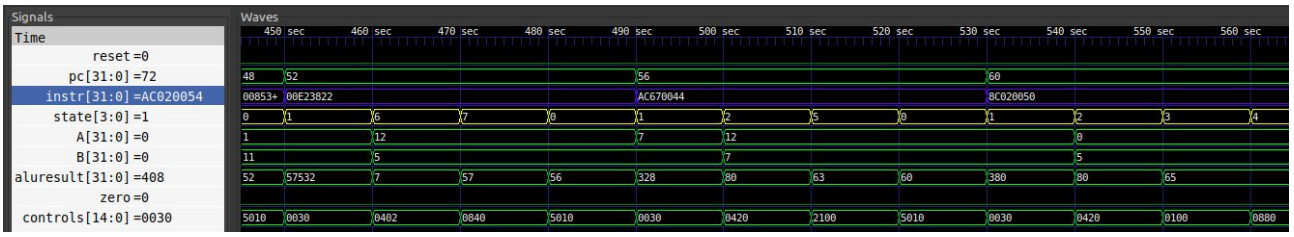
[150, 300sec]



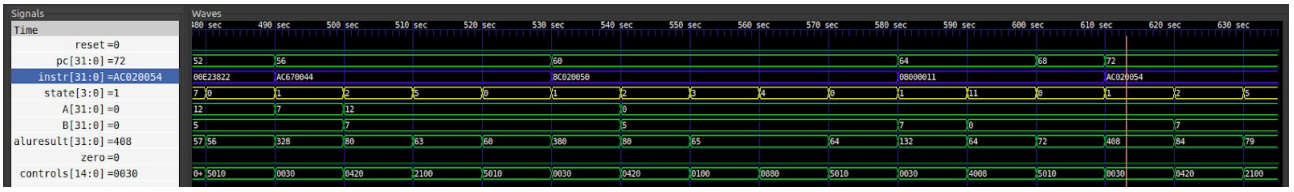
[300, 450sec]



[450, 550sec]



[550, 635sec]



GRACIAS! :D

Los demás informes estarán en la carpeta respectiva de cada laboratorio…