LAB 7- 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 primera parte, donde se construirá un simplificado MIPS Single-Cycle Processor usando uno provisto en System Verilog. Para ello, combinaremos el ALU que realizamos en el Laboratorio 5 con el código del resto del procesador tomado del libro Harris Harris Ch.7.3 'Microarchitecture - *Digital Design and Computer Architecture*'. Luego se probará con un programa de testeo para verificar que el sistema funciona. Seguidamente, se implementarán dos nuevas instrucciones, y se escribirá un nuevo programa de testeo que confirme que las nuevas instrucciones trabajan bien. Para el final de esta primera parte, entenderemos por completo el funcionamiento del MIPS Single-Cycle Processor.

Para empezar el desarrollo del laboratorio, se nos recomendó estar familiarizados con el capítulo del libro HH de donde se sacó el código y esquemático del sistema usado en este laboratorio. Esta versión del MIPS Single-Cycle Processor puede ejecutar las siguientes instrucciones: `add`, `sub`, `and`, `or`, `slt`, `lw`, `sw`, `beq`, `addi` y `j`. El modelo divide la máquina en dos unidades mayores: *the unit control* y *the datapath*. Cada unidad está constituida por varios bloques funcionales.

3. MIPS Single-Cycle Processor:

En el archivo Lab07.tar se dispone del MIPS Single-Cycle Processor implementado en el libro HH Ch.7 . Se recomienda estudiar arduamente los archivos provistos, asimismo relacionar los contenidos y nombres del código en System Verilog con los cables del esquemático. Luego de entender el **módulo controller** y sus implicados por completo, se puede ahora ver el **módulo datapath** que a su vez contiene otros **sub-módulos**. Se deberá entender a profundidad estos también, para llegar al **módulo alu** que no está definido. Ahí se copiará el ALU realizado en el Laboratorio 5, asegurándose de que los nombres de los módulos, inputs y outputs coincidan, en orden y nombres, con los del módulo datapath. El módulo top incluye las instrucciones y datos de la memoria, así como los procesadores. Cada una de las memories son una matriz de 64 palabrasx32 bits. La memoria de instrucciones debe contener algunos valores iniciales que representan el programa. El código de lenguaje de máquina para el programa se almacena en **memfile.dat**.

4. Testing the single-cycle MIPS processor:

Luego de haber traducido el MIPS Single-Cycle Processor de System Verilog a Verilog y habiendo agregado el ALU, se procederá a testear el programa. Para esto, empezaremos prediciendo los resultados que deberá botar el programa antes de correrlo. Los resultados supuestos se incluirán en la Tabla 1.

¿Qué dirección escribirá la instrucción final sw y qué valor será? Esto se responderá con los waveforms en el desarrollo.

La waveform final deberá contener las siguientes señales en el correspondiente orden: `clk`, `reset`, `pc`, `instr`, `aluout`, `writedata`, `memwrite`, `readdata`. Todos los valores deberán mostrarse en hexadecimal y ser visibles en las imágenes.

5. Modifying and Testing the MIPS single-cycle processor:

Ahora se modificará el MIPS Single-Cycle Processor adicionando las instrucciones de `ori` y `bne`. Primero se recomienda **modificar el esquema** del procesador para mostrar

los cambios que fueron necesarios. Se dibujarán directamente los cambios realizados. Luego, se modificará el **main decoder** y el **ALU decoder** como sea requerido. Asimismo, se modificará el código provisto como sea necesario para que incluya las modificaciones. Se muestran los cambios en las tablas al final del laboratorio. Para el testeo, se hará un programa de testeo que verifique que el nuevo procesador funcione. El programa revisa el funcionamiento de las nuevas instrucciones sin que las anteriores fallen. Usar el test2.asm.

```
# test2.asm
# Test MIPS instructions.

#Assembly Code
main:       ori  $t0, $0,  0x8000
            addi $t1, $0,  -32768
            ori  $t2, $t0, 0x8001
            beq  $t0, $t1, there
            slt  $t3, $t1, $t0
            bne  $t3, $0,  here
            j    there
here:       sub  $t2, $t2, $t0
            ori  $t0, $t0, 0xFF
there:      add  $t3, $t3, $t2
            sub  $t0, $t2, $t0
            sw   $t0, 82($t3)
```

Figure 1. MIPS assembly program: test2.asm

Convertir el programa a instrucciones en lenguaje máquina y escribirlos en **memfile2.dat.**

Modificar el imem para cargar el archivo. Modificar, también, el testbench para que revise las direcciones y valores de datos apropiados indicando que la simulación fue exitosa. Correr el programa, verificar los resultados, corregir donde sea necesario. Cuando todo ello se haya hecho, imprimir las waveforms, indicando la dirección y data value escrito por la instrucción `sw`.

DESARROLLO:

1. Versión completa de las Tablas.

A continuación se muestra la Tabla 1, correspondiente al primer test del programa. Esta se llenó con 16 cycles, instrucciones y señales como el reset, pc, branch, srca, srcb, aluout, zero, pcsrc, writedate, memwrite y readdata. Para llenar las columnas de las señales se usaron las waveforms generadas al realizar el test 1. Las instrucciones en hexadecimal se disponían en el archivo memfile.dat, se usó un programa que convirtiese estos números a mips instruction, para una mejor especificación de las instrucciones realizadas. (El link se incluye en las referencias al final del informe).

Predicciones: los datos llenados están en decimales.

Cycle	Reset	pc	instr	branch	srca	srcb	aluout	zero	pcsrc	writedata	memwrite	read data
1	1	0	addi $2, $0, 5 20020005	0	0	5	5	0	0	0	0	x
2	0	4	addi $3, $0, 12 200300c	0	0	12	12	0	0	0	0	x
3	0	8	addi $7, $3, -9 20067fff7	0	12	-9	3	0	0	0	0	x
4	0	c	or $a0,$a3,$v0 00e22025	0	3	5	7	0	0	0	0	x
5	0	10	and $a1,$v1,$a0 00642824	0	12	7	4	0	0	0	0	x
6	0	14	add $a1,$a1,$a0 00a42820	0	4	7	11	0	0	0	0	x
7	0	18	beq $a1,$a3,0x000a 10a7000a	1	11	3	8	0	0	0	0	x
8	0	1c	slt $a0,$v1,$a0 0064202a	0	12	7	0	1	0	0	0	x
9	0	20	beq $a0,$zero,0x0001 10800001	1	0	0	0	1	1	0	0	x
10	0	28	slt $a0,$a3,$v0 00e2202a	0	3	5	1	0	0	7	0	x
11	0	2c	add $a3,$a0,$a1 00853820	0	1	11	12	0	0	0	0	x
12	0	30	sub $a3,$a3,$v0 00e23822	0	12	5	7	0	0	0	0	x
13	0	34	sw $a3,0x0044,$v1 ac670044	0	12	68	80	0	0	7	1	x

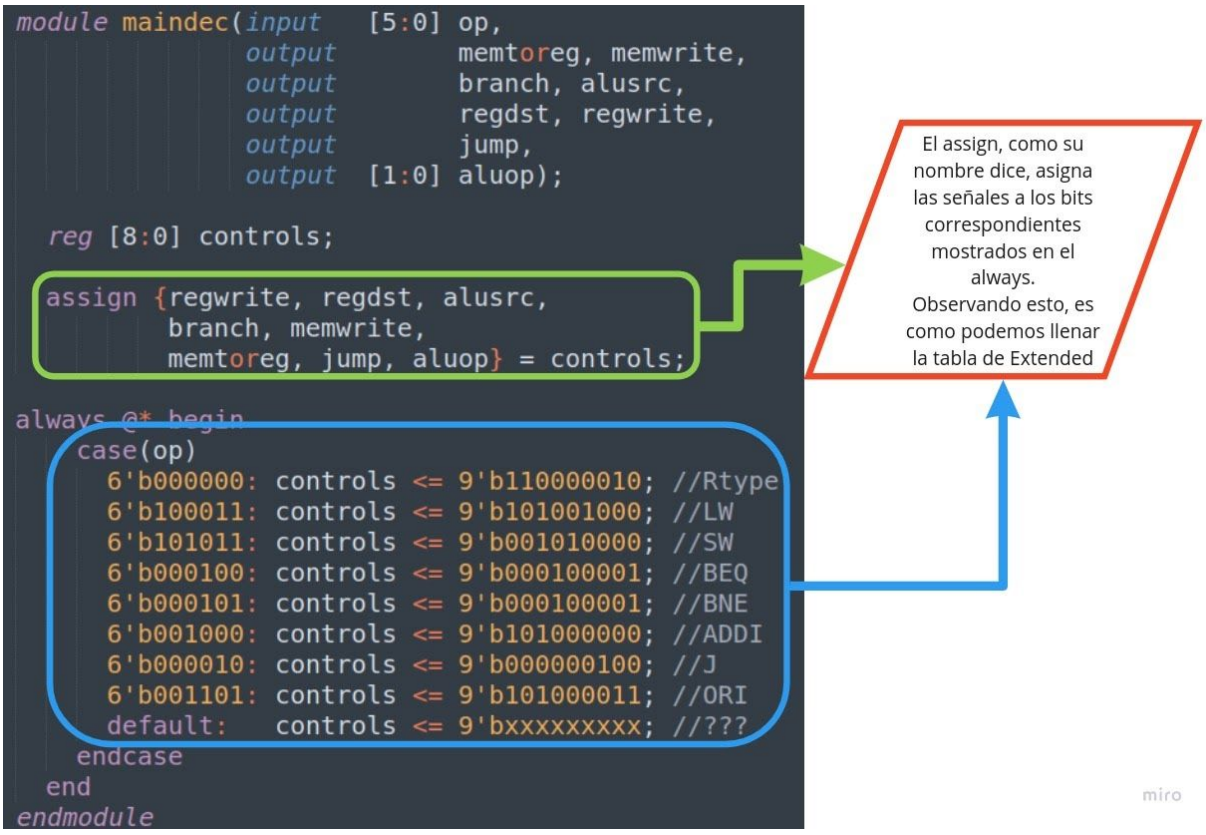
14	0	38	lw $v0,0x0050,$zero 8c020050	0	0	80	80	0	0	0	0	7
15	0	3c	j 0x0000011 08000011	0	0	0	0	1	0	0	0	x
16	0	44	sw $v0 0x0054 $zero ac020054	0	0	84	84	0	0	7	1	x

Datos Obtenidos en la simulación: en hexadecimal

Cycle	Reset	pc	instr	branch	srca	srcb	aluout	zero	pcsrc	writedata	memwrite	read data
1	1	0	addi $2, $0, 5 20020005	0	0	5	5	0	0	0	0	x
2	0	4	addi $3, $0, 12 200300c	0	0	12	12	0	0	0	0	x
3	0	8	addi $7, $3, -9 20067fff7	0	12	-9	3	0	0	0	0	x
4	0	c	or $a0,$a3,$v0 00e22025	0	3	5	7	0	0	5	0	x
5	0	10	and $a1,$v1,$a0 00642824	0	12	7	4	0	0	7	0	x
6	0	14	add $a1,$a1,$a0 00a42820	0	4	7	b	0	0	7	0	x
7	0	18	beq $a1,$a3,0x000a 10a7000a	1	11	3	8	0	0	3	0	x
8	0	1c	slt $a0,$v1,$a0 0064202a	0	12	7	0	1	0	7	0	x
9	0	20	beq $a0,$zero,0x0001 10800001	1	0	0	0	1	1	0	0	x
10	0	28	slt $a0,$a3,$v0 00e2202a	0	3	5	1	0	0	5	0	x
11	0	2c	add $a3,$a0,$a1 00853820	0	1	11	12	0	0	b	0	x
12	0	30	sub $a3,$a3,$v0 00e23822	0	12	5	7	0	0	5	0	x
13	0	34	sw $a3,0x0044,$v1 ac670044	0	12	68	80	0	0	7	1	x
14	0	38	lw $v0,0x0050,$zero 8c020050	0	12	80	80	0	0	5	0	7
15	0	3c	j 0x0000011 08000011	0	0	0	0	1	0	0	0	x
16	0	44	sw $v0 0x0054 $zero ac020054	0	0	84	84	0	0	7	1	x

Observamos que las señales obtenidas difirieron con las predicciones hechas. Esto sin embargo no significaba que nuestro código estuviese errado, sino que writedata si bien recibe valores, estos no son usados, pues el memwrite decide cuando se activa el data memory. Finalmente, comprobamos que el código fue apropiadamente traducido de System Verilog a Verilog, y nuestro ALU funcionaba correctamente.

La siguiente tabla corresponde al Extended Functionality del Main Decoder para llenar esta usamos el código del módulo *maindec* en el mips.v.



Extended functionality. Main Decoder:

Instruction	Op5:0	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	ALUOp1:0	Jump
R-type	000000	1	1	0	0	0	0	00	0
`lw`	100011	1	0	1	0	0	1	00	0
`sw`	101011	0	X	1	0	1	X	00	0
`beq`	000100	0	X	0	1	0	X	01	0
`addi`	001000	1	0	1	0	0	0	00	0
`j`	000010	0	X	X	X	0	X	XX	1
`ori`	001101	1	0	1	0	0	0	11	0
`bne`	000101	0	0	0	1	0	0	01	0

La siguiente tabla corresponde al Extended Functionality del Main Decoder para llenar esta usamos el código del módulo *aludec* en el mips.v.

```
module aludec(input   [5:0] funct,
              input   [1:0] aluop,
              output reg [3:0] alucontrol);

  always @* begin
    case(aluop)
      2'b00: alucontrol <= 4'b0000;  // add
      2'b01: alucontrol <= 4'b0010;  // sub
      2'b11: alucontrol <= 4'b0101;  // or

      default: case(funct)          // RTYPE
          6'b100000: alucontrol <= 4'b0000; // ADD
          6'b100010: alucontrol <= 4'b0010; // SUB
          6'b100100: alucontrol <= 4'b0100; // AND
          6'b100101: alucontrol <= 4'b0101; // OR
          6'b100111: alucontrol <= 4'b0111; // NOR
          6'b100110: alucontrol <= 4'b0110; // XOR
          6'b101010: alucontrol <= 4'b1010; // SLT
          default:   alucontrol <= 4'bxxxx; // ???
      endcase
    endcase
  end
endmodule
```

Extended functionality. ALU Decoder:

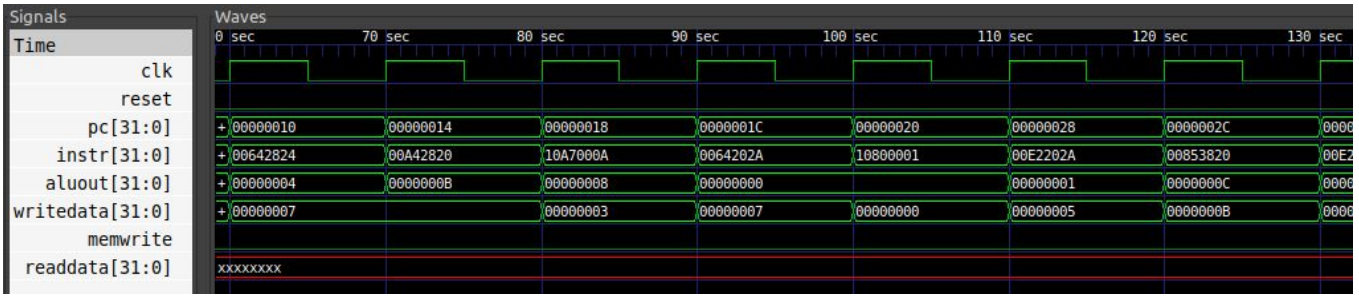
ALUOp1:0	Meaning
00	Add
01	Substract
10	Look at `funct` field
11	Or

2. Waveforms test 1.

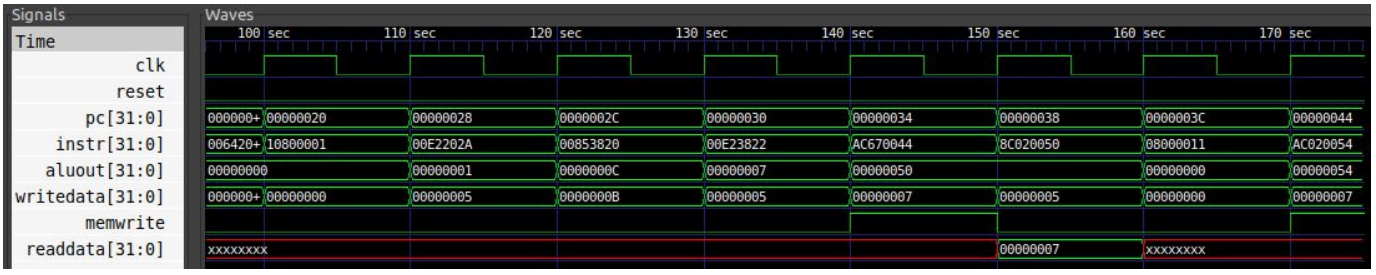
Los waveforms fueron demasiado largos como para mostrarlos en una sola captura de pantalla, por ello se sacaron las capturas por intervalos como se muestra continuación:

➔ [0 , 70 sec]:

➔ [70, 130 sec]:



➔ [130, 175 sec]



~~~~~~

¿Qué dirección escribirá la instrucción final sw y qué valor será?

La dirección de la instrucción final se obtiene del aluout, esta es 84 en decimal. El valor está mostrado por el writedata, este es 7.

~~~~~~

3. `ori` and `bne`.

➔ Para la implementación de **ori**, el código se modificó como se muestra:

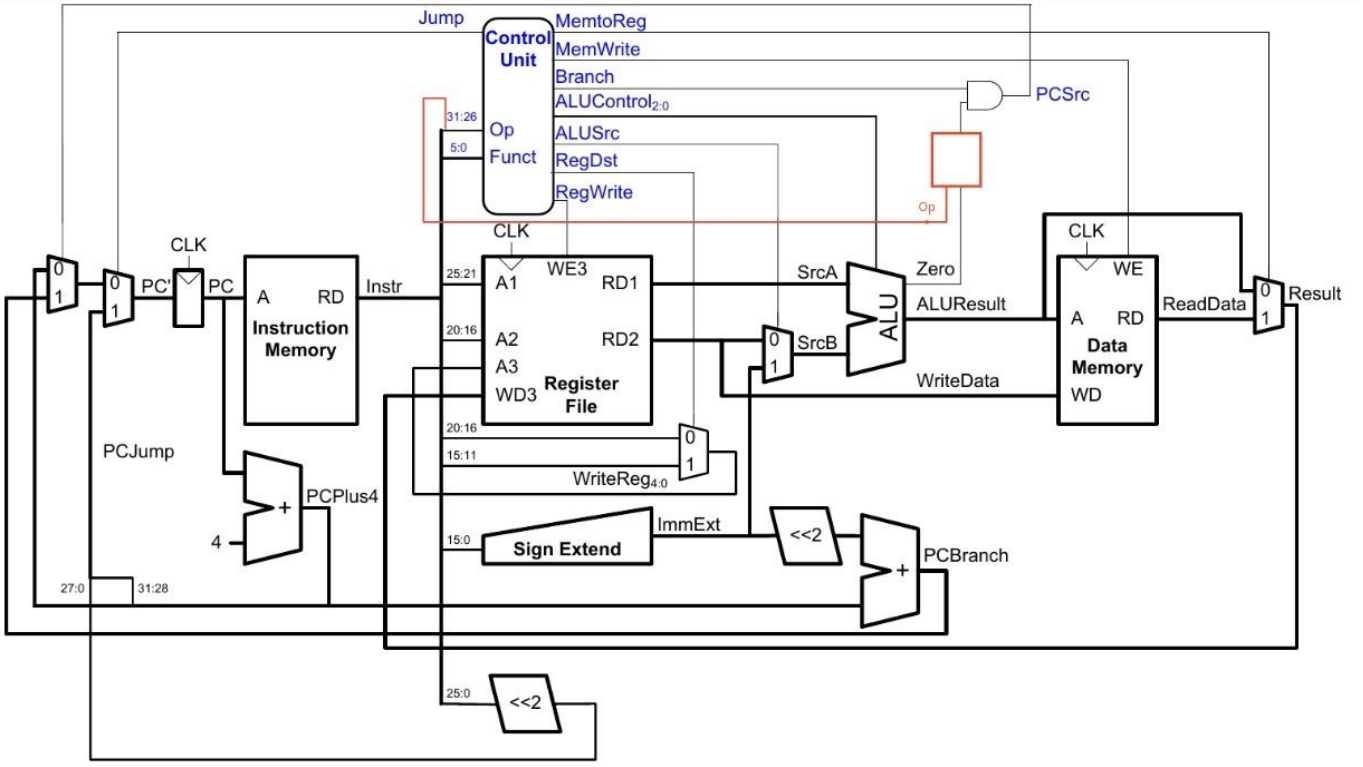
```
module maindec(input   [5:0] op,
               output        memtoreg, memwrite,
               output        branch, alusrc,
               output        regdst, regwrite,
               output        jump,
               output  [1:0] aluop);

  reg [8:0] controls;

  assign {regwrite, regdst, alusrc,
          branch, memwrite,
          memtoreg, jump, aluop} = controls;

always @* begin
    case(op)
      6'b000000: controls <= 9'b110000010; //Rtype
      6'b100011: controls <= 9'b101001000; //LW
      6'b101011: controls <= 9'b001010000; //SW
      6'b000100: controls <= 9'b000100001; //BEQ
      6'b000101: controls <= 9'b000100001; //BNE
      6'b001000: controls <= 9'b101000000; //ADDI
      6'b000010: controls <= 9'b000000100; //J
      6'b001101: controls <= 9'b101000011; //ORI
      default:   controls <= 9'bXXXXXXXXX; //???
    endcase
  end
endmodule
```

➔ Para la implementación del bne se requirió modificar el esquemático de la siguiente manera: (lo agregado se muestra en rojo y la imagen se adjuntará aparte en caso de que no se aprecie del todo bien en el informe)



Single-cycle MIPS processor

➔ En el código, el cambio realizado fue:

```
module controller(input   [5:0] op, funct,
                  input         zero,
                  output        memtoreg, memwrite,
                  output        pcsrc, alusrc,
                  output        regdst, regwrite,
                  output        jump,
                  output  [3:0] alucontrol);
  wire [1:0] aluop;
  wire       branch;

  maindec md(op, memtoreg, memwrite, branch,
             alusrc, regdst, regwrite, jump,
             aluop);
  aludec  ad(funct, aluop, alucontrol);

  assign pcsrc = branch & ((op == 6'b000101)? ~zero : zero);
endmodule
```

4. `memfile2.dat` and test 2.

Para el memfile2.dat se tenía que pasar las instrucciones mostradas en el test2.asm, para esto usamos la herramienta Mars.

test2.asm

```
1  # test2.asm
2  # Test MIPS instructions
3
4  # Assembly Code
5  main:   ori $t0, $0, 0x8000
6          addi $t1, $0, -32768
7          ori $t2, $t0, 0x8001
8          beq $t0, $t1, there
9          slt $t3, $t1, $t0
10         bne $t3, $0, here
11         j there
12 here:   sub $t2, $t2, $t0
13         ori $t0, $t0, 0xFF
14 there:  add $t3, $t3, $t2
15         sub $t0, $t2, $t0
16         sw $t0, 82($t3)
```

Luego se simuló el código para obtener las instrucciones en hexadecimal desde el Text Segment de la emulación.

Text Segment

Bkpt	Address	Code	Basic	Source
	12288	0x34088000	ori $8,$0,32768	5: main: ori $t0, $0, 0x8000
	12292	0x20098000	addi $9,$0,-32768	6: addi $t1, $0, -32768
	12296	0x350a8001	ori $10,$8,32769	7: ori $t2, $t0, 0x8001
	12300	0x11090005	beq $8,$9,5	8: beq $t0, $t1, there
	12304	0x0128582a	slt $11,$9,$8	9: slt $t3, $t1, $t0
	12308	0x15600001	bne $11,$0,1	10: bne $t3, $0, here
	12312	0x08000c09	j 12324	11: j there
	12316	0x01485022	sub $10,$10,$8	12: here: sub $t2, $t2, $t0
	12320	0x350800ff	ori $8,$8,255	13: ori $t0, $t0, 0xFF
	12324	0x016a5820	add $11,$11,$10	14: there: add $t3, $t3, $t2
	12328	0x01484022	sub $8,$10,$8	15: sub $t0, $t2, $t0
	12332	0xad680052	sw $8,82($11)	16: sw $t0, 82($t3)

Finalmente pasamos las instrucciones en hexadecimal al archivo memfile2.dat que luego de haber modificado el código, se testea y generan los waveforms siguientes.

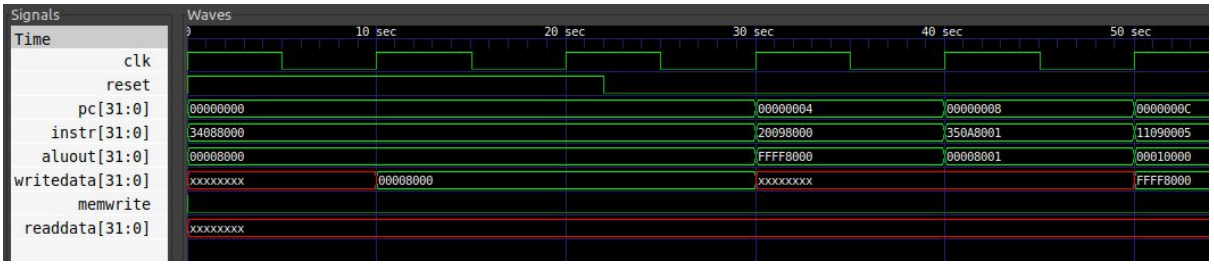
memfile2.dat

```
1  34088000
2  20098000
3  350a8001
4  11090005
5  0128582a
6  15600001
7  08000009
8  01485022
9  350800ff
10 016a5820
11 01484022
12 ad680052
```

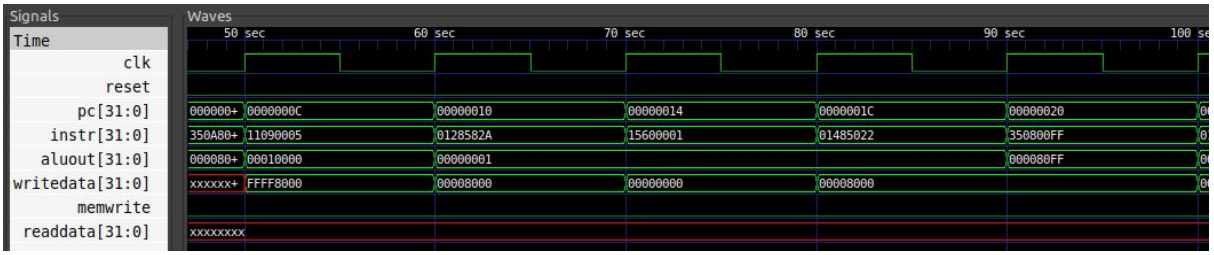
5. Waveforms test 2.

Los waveforms fueron demasiado largos como para mostrarlos en una sola captura de pantalla, por ello se sacaron las capturas por intervalos como se muestra continuación:

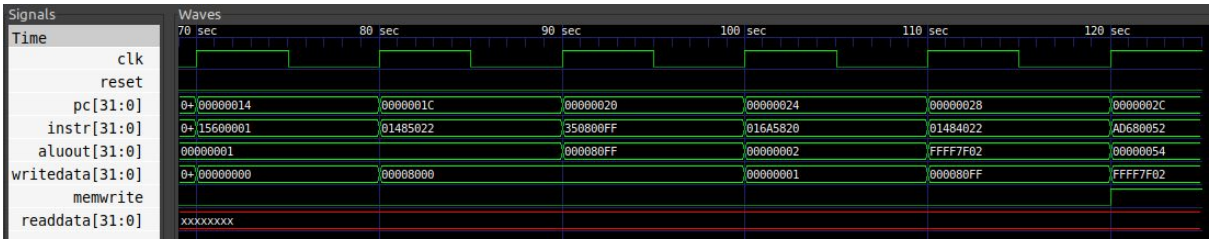
➔ [0, 50 sec]



➔ [50, 90 sec]



➔ [90, 125 sec]



6. Referencias:

➢ Harris D. y Harris S. (2007). *Microarchitecture. En MacFadden N., Digital desing and computer architecture* (pp 363 - 461). San Francisco, Estados Unidos: Elsevier

➢ MIPS Converter: https://www.eg.bucknell.edu/~csci320/mips_web/

GRACIAS! :D

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