Proyecto Final - Fase 1

Meza Najera Ximena
Ibañez Hernandez Moises Antonio
Hernandez Lomeli Diego Armando
Rivera Reos Fernando de Jesús

04/05/2023

Arquitectura de Computadoras

INVESTIGACIÓN

MIPS

El procesador MIPS de 32 bits es una arquitectura de procesador RISC que se utiliza en una amplia gama de aplicaciones, desde dispositivos integrados hasta supercomputadoras. La sigla MIPS significa "Million Instructions Per Second" (Millones de Instrucciones por Segundo), que se refiere a la capacidad del procesador para ejecutar instrucciones de manera eficiente. A continuación, se presentan algunas características generales y elementos del procesador MIPS de 32 bits:

1. Arquitectura RISC: El procesador MIPS utiliza una arquitectura RISC (Reduced Instruction Set Computing) que se caracteriza por tener un conjunto de instrucciones reducido y simple. Esto permite que el procesador pueda ejecutar instrucciones de manera más rápida y eficiente.
2. Conjunto de instrucciones: El conjunto de instrucciones del procesador MIPS consta de aproximadamente 100 instrucciones, lo que lo hace más simple que otros procesadores. Además, el procesador MIPS utiliza un formato de instrucción de 32 bits, lo que permite que las instrucciones sean más compactas.
3. Registro de propósito general: El procesador MIPS tiene 32 registros de propósito general de 32 bits, que se utilizan para almacenar datos y direcciones de memoria.
4. Unidad de control: La unidad de control del procesador MIPS es responsable de buscar, decodificar y ejecutar las instrucciones.
5. Unidad aritmética y lógica: La unidad aritmética y lógica (ALU) del procesador MIPS es responsable de realizar operaciones matemáticas y lógicas, como sumas, restas, multiplicaciones, divisiones y comparaciones.
6. Caché: El procesador MIPS de 32 bits utiliza una caché de datos y una caché de instrucciones para mejorar el rendimiento y reducir la latencia en el acceso a memoria.
7. Arquitectura de canalización: El procesador MIPS utiliza una arquitectura de canalización para aumentar la eficiencia en la ejecución de instrucciones. La canalización permite que varias instrucciones se ejecuten al mismo tiempo,

reduciendo el tiempo de espera entre instrucciones.

Algunas aplicaciones típicas del procesador MIPS de 32 bits incluyen dispositivos de redes, sistemas embebidos, routers, consolas de videojuegos y supercomputadoras. El procesador MIPS también se utiliza en algunas versiones de la consola de juegos PlayStation de Sony.

SET DE INSTRUCCIONES

Un conjunto de instrucciones (también conocido como ISA, por sus siglas en inglés) es un conjunto de instrucciones que un procesador es capaz de ejecutar. En el caso de la arquitectura MIPS, el conjunto de instrucciones consiste en un conjunto de operaciones básicas que el procesador puede llevar a cabo.

El conjunto de instrucciones MIPS consta de alrededor de 100 instrucciones diferentes, que se utilizan para realizar operaciones aritméticas y lógicas, transferencias de datos y operaciones de control de flujo. Cada instrucción tiene un código de operación (conocido como opcode), que indica la operación que se debe realizar. Además, cada instrucción puede tener uno o más operandos, que son los valores que se utilizan en la operación.

Por ejemplo, una de las instrucciones básicas del conjunto de instrucciones MIPS es "add", que se utiliza para sumar dos números enteros. La sintaxis de la instrucción es la siguiente:

add $dest, $src1, $src2

En esta instrucción, "$dest" es el registro de destino donde se almacenará el resultado de la suma, "$src1" y "$src2" son los registros que contienen los operandos que se sumarán.

Otra instrucción útil en el conjunto de instrucciones MIPS es "lw", que se utiliza para cargar un valor de una dirección de memoria. La sintaxis de la instrucción es la siguiente:

lw $dest, offset($src)

En esta instrucción, "$dest" es el registro donde se almacenará el valor cargado, "$src" es el registro que contiene la dirección de memoria base, y "offset" es un valor inmediato que se suma a la dirección base para obtener la dirección final.

Ljubisa Bajic

Profesionista radicado en Toronto,Ontario, Canadá. Cofundador y director de tecnología de Tenstorrent, empresa líder en inteligencia artificial y semiconductores RISC-V. Fundo Tenstorrent en 2016 junto a Ivan Hamer y Milos Trajkovic, y supervisó el crecimiento de la empresa hasta el estatus de unicornio. Dejó su cargo a tiempo completo en Tenstorrent y sigue involucrado en calidad de asesor.

Ljubisa Bajic es un veterano de la industria de semiconductores y ha trabajado extensamente en diseño y depuración de VLSI, así como en arquitectura de aceleradores y experiencia en software. Pasó una década en AMD como arquitecto ASIC trabajando en diseño de gestión de energía y DSP, antes de pasar a NVIDIA como arquitecto senior y luego regresar a AMD durante un par de años trabajando con Jim en el hardware que impulsa el balance de AMD hoy en día. Dejó AMD para fundar Tenstorrent con otros dos cofundadores en 2016, ha recaudado $240 millones en tres rondas de financiación y ha creado tres generaciones de procesadores, la última de las cuales se enviará este año. Al menos dos generaciones más están en camino.

Jim Keller

Jim Keller es un ingeniero de microprocesadores estadounidense conocido por su trabajo en AMD y Apple. Fue el arquitecto principal de la microarquitectura AMD K8 (incluyendo el Athlon 64 original) y participó en el diseño de los procesadores Athlon (K7) y Apple A4 / A5. También fue coautor de las especificaciones para el conjunto de instrucciones x86-64 y el interconexión HyperTransport. Desde 2012 hasta 2015, regresó a AMD para trabajar en las microarquitecturas Zen y K12 de AMD.

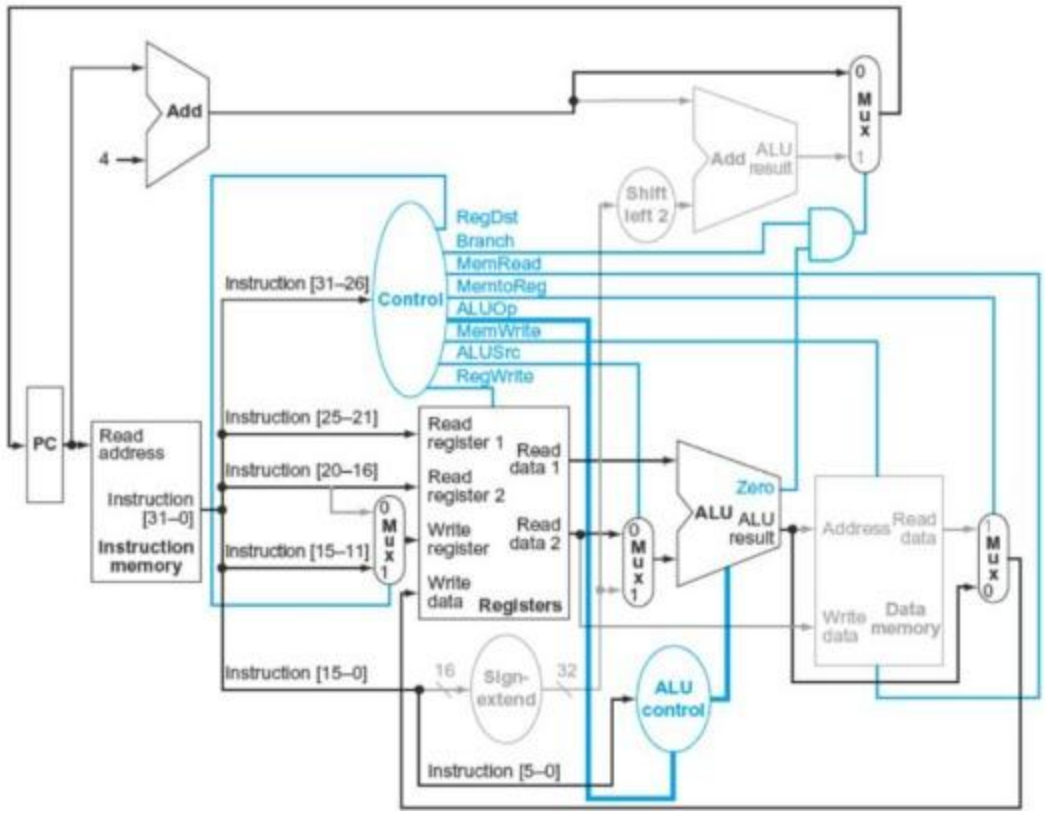
DESARROLLO

Para el caso de este proyecto el conjunto de instrucciones seria una version reducida que se veria de esta manera:

Instrucción	Tipo	Sintaxis		Instrucción	Tipo	Sintaxis
Add	R	Add $rd, $rs, $rt		Lw	I	lw $rt, offset($rs)
Sub	R	sub $rd, $rs, $rt		Sw	I	sw $rt, offset($rs)
Mul	R	mul $rd, $rs, $rt		Slt	R	slt $rd, $rs, $rt
Div	R	div $rs, $rt		Slti	I	slti $rt, $rs, imm
Or	R	or $rd, $rs, $rt		Beq	I	beq $rs, $rt, offset
And	R	and $rd, $rs, $rt		Bne	I	bne $rs, $rt, offset
Addi	I	addi $rt, $rs, imm		j	J	j target
Subi	I	subi $rt, $rs, imm		Nop	R	nop
Ori	I	ori $rt, $rs, imm		Bgtz	I	bgtz $rs, offset
Andi	I	andi $rt, $rs, imm				

Módulos y código

Se implementó el single datapath con sus módulos correspondientes acorde a la siguiente imagen:



Los cuales en código se verían de la siguiente manera:

ALU:

```
1  `timescale 1ns/1ns
2  module ALU (
3      input [31:0]op1,
4      input [31:0]op2,
5      input [3:0]selOp,
6      output reg [31:0] resultado,
7      output reg zeroFlag
8  );
9      initial begin
10         resultado <= 0;
11         zeroFlag <= 0;
12     end
13     always @* begin
14         case (selOp)
15             4'b0000: begin //And
16                 resultado <= op1 & op2;
17             end
18             4'b0001: begin// or
19                 resultado <= op1 | op2;
20             end
21             4'b0010: begin//suma
22                 resultado <= op1 + op2;
23             end
24             4'b0110: begin//resta
25                 resultado <= op1 - op2;
```

```
26             end
27             4'b0111: begin//SLT
28                 resultado <= op1 < op2
29                         ? 32'd1
30                         : 32'd0;
31             end
32             4'b1100: begin//Nor
33                 resultado <= ~(op1 | op2);
34             end
35             default:  begin
36
37             end
38
39         endcase
40         zeroFlag = resultado == 32'd0;
41     end
42 endmodule
```

Adder:

```
1  `timescale 1ns/1ns
2  module Adder (
3      input [31:0] add,
4      input [31:0] originalNumber,
5      output [31:0] out
6  );
7      assign out = originalNumber + add;
8  endmodule
```

AluControl:

```
1  /* Todo el desarrollo de este modulo esta basado en el libro
2  "Computer organization and design" página 260 Figura 4.12.
3  Esta sujeto a cambios para complemtenar las funcionalidades */
4  `timescale 1ns/1ns
5  module AluControl (
6      input [5:0] functionField, //Obtenida directamente de la insturcción actual
7      input [1:0] AluOp,//Obtenido desde la unidad de control
8      output reg [3:0] operation//Operación que debe realizar ahora la ALU
9  );
10     initial operation = 4'd0;
11     /*
12     Cuando AluOp en su indice 1 contenga 1, será una instrucción de tipo R, por ello solo evaluamos ese campo
13      */
14     always @* begin
15         if (AluOp == 2'b10) begin
16             case (functionField)
17                 6'b100000: begin//Add operation
18                     operation <= 4'b0010;
19                 end
20                 6'b100010: begin//subtract operation
21                     operation <= 4'b0110;
22                 end
23                 6'b100100: begin//And operation
24                     operation <= 4'b0000;
25                 end
```

```
26                  6'b100101: begin//Or operation
27                      operation <= 4'b0001;
28                  end
29                  6'b101010: begin//Set on less
30                      operation <= 4'b0111;
31                  end
32                  6'b000000: begin//Nop operation (no operation)
33                      operation <= 4'b1000;
34                  end
35              endcase
36          end
37          /* en los siguientes campos, el campo "functionField" no
38          tiene relevancia, es parte de una condición "don´t care" */
39          else begin
40              /* Cuando AluOp en su indice 0 tiene 1 */
41              if (AluOp[0] == 1'b1) begin
42
43              end else begin
44                  /* Cuado no se cumpla ninguna condición anterior,
45                  entonces es una instrucción LW, SW */
46              end
47          end
48
49      end
50  endmodule
```

ControlUnit:

```
1   `timescale 1ns/1ns
2   module ControlUnit (
3       input [5:0] instruction,
4       output reg regDst,
5       output reg branch,
6       output reg memRead,
7       output reg memToReg,
8       output reg [1:0]ALUop,
9       output reg memWrite,
10      output reg ALUSrc,
11      output reg RegWrite
12  );
13      always @(instruction) begin
14          if (instruction == 6'b000_000) begin
15              /* Formato de instrucciones R
16              No almacena datos en memoria, solo en banco de registros*/
17              regDst <= 1;
18              branch <= 0;
19              memRead <= 0;
20              memToReg <= 0;
21              ALUop <= 10;
22              memWrite <= 0;
23              ALUSrc <= 0;
24              RegWrite <= 1;
25          end
26      end
27  endmodule //ControlUnit
```

DataMemory:

```
1   `timescale 1ns/1ns
2   module DataMemory(
3       input [31:0] dataEN,
4       input [5:0] d,
5       input write,
6       input read,
7       output reg[31:0] data
8   );
9       reg [31:0] ram [0:63];
10      always @* begin
11          if (write == 1'b1) begin
12              //Escribir
13              data <= ram[d];
14          end
15          if (read == 1'b1) begin
16              //leer
17              data <= ram[d];
18          end
19      end
20
21
22  endmodule
```

MInstructions:

```
1
2   `timescale 1ns/1ns
3   module MInstructions(
4       input [31:0] rdAccess,
5       output reg[31:0] data
6   );
7       reg [7:0] ram [0:255];
8       initial begin
9           //Lectura de datos del archivo
10          $readmemb("ram.txt", ram);
11      end
12
13
14      always @* begin
15          //leer
16          data <= {ram[rdAccess], ram[rdAccess+1],ram[rdAccess+2], ram[rdAccess+3]};
17      end
18  endmodule
```

Mux01_31Bits

```
1   `timescale 1ns/1ns
2   module Mux01_31Bits (
3       input [31:0] valOnSel0,
4       input [31:0] valOnSel1,
5       input selector,
6       output reg [31:0]dataOut
7   );
8
9       assign dataOut = selector == 1'b1
10                      ? valOnSel1
11                      : valOnSel0;
12
13  endmodule //Mux01
```

Mux01_31_33

```
1
2   `timescale 1ns/1ns
3   module Mux01_31_33 (
4       input [31:0] valOnSel0,
5       input [33:0] valOnSel1,
6       input selector,
7       output reg [33:0]dataOut
8   );
9
10      always @(selector) begin
11          dataOut <= selector == 1'b1
12                      ? valOnSel1
13                      //Esta concatenación es unicamente para igualar a los bits de salida
14                      : {2'b00,valOnSel0};
15      end
16  endmodule //Mux01
```

Mux01_5Bits

```
1   `timescale 1ns/1ns
2   /* El primer input es el que se devuelve si el
3   selector tienen 1'b1 de valor */
4   module Mux01_5Bits (
5       input [4:0] valOnSel0,
6       input [4:0] valOnSel1,
7       input selector,
8       output reg [4:0]dataOut
9   );
10
11      assign dataOut = selector == 1'b1
12                      ? valOnSel1
13                      : valOnSel0;
14
15  endmodule //Mux01
```

PC

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

Resultados

Cada uno de los módulos se instancio en un TestBench y se corrió la simulación, dando los siguientes resultados:

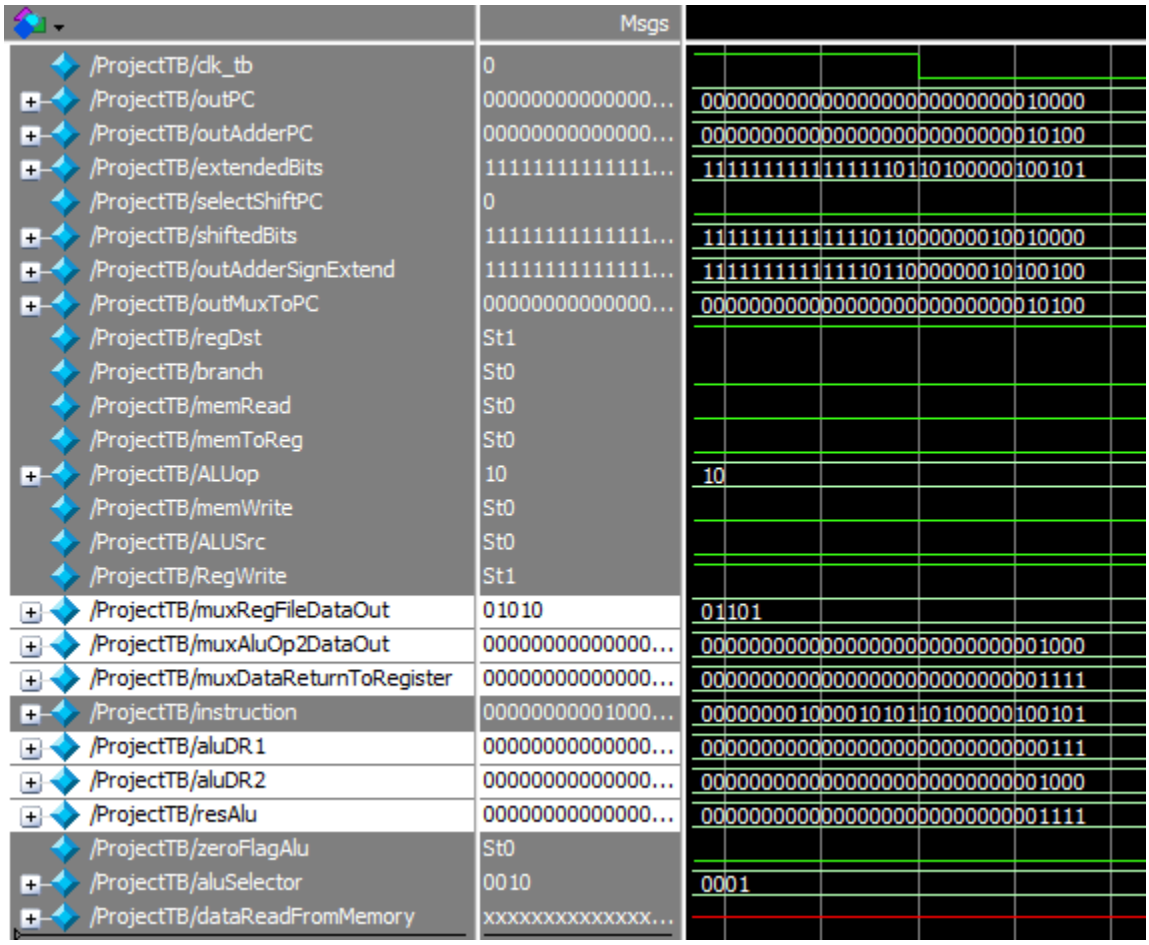
Se inicializa la memoria de instrucciones:

```
000000_00//No op
000_00000
00000_000
00_000000
000000_00//ADD $10, $1, $3
001_00011
01010_000
00_100000
000000_00//SLT $11, $4, $5
100_00101
01011_000
00_101010
000000_00//AND $12, $3, $4
011_00100
01100_000
00_100100
000000_00//OR $13, $4, $5
100_00101
01101_000
00_100101
000000_00//No op
000_00000
00000_000
00_000000      You, 4 hours
```

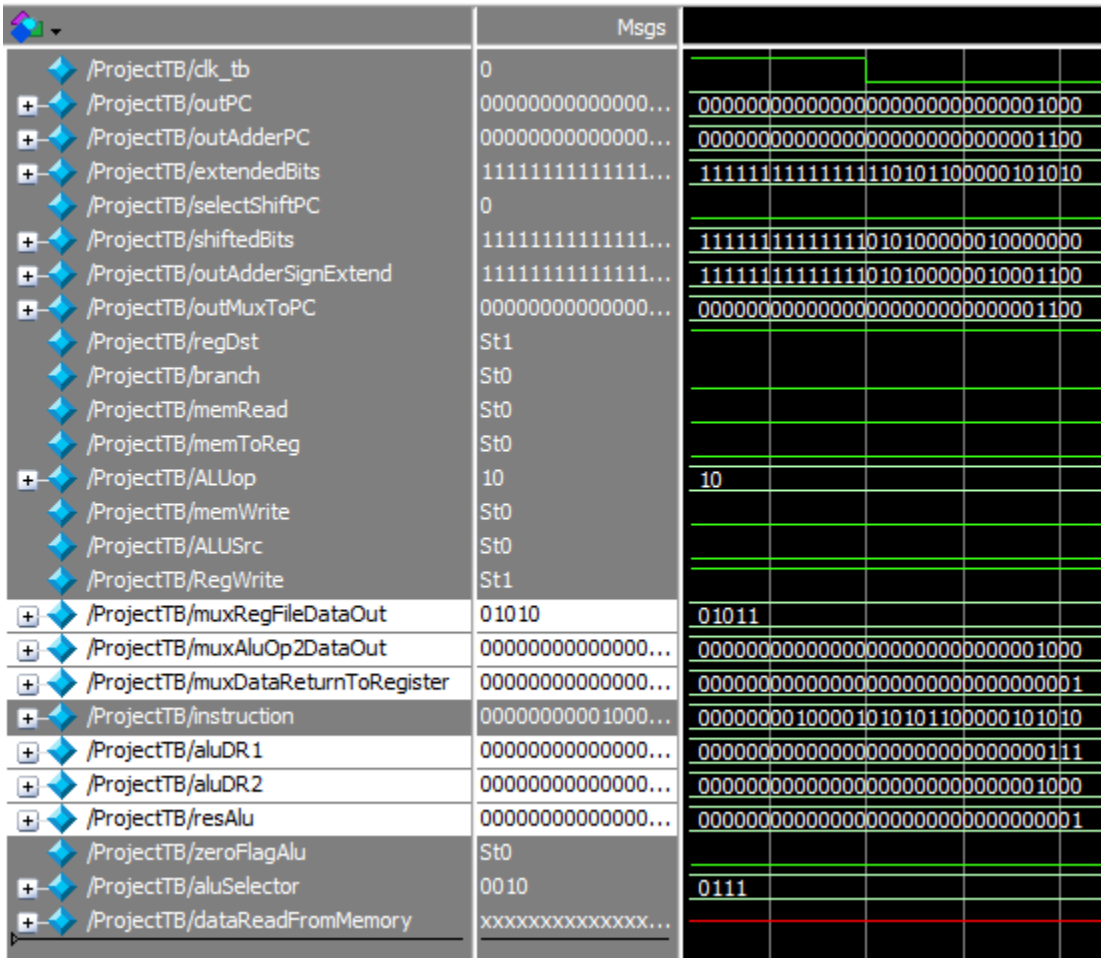
Se precarga el banco de registros:

```
00011001
10010001
11001000
0101
0111
1000
```

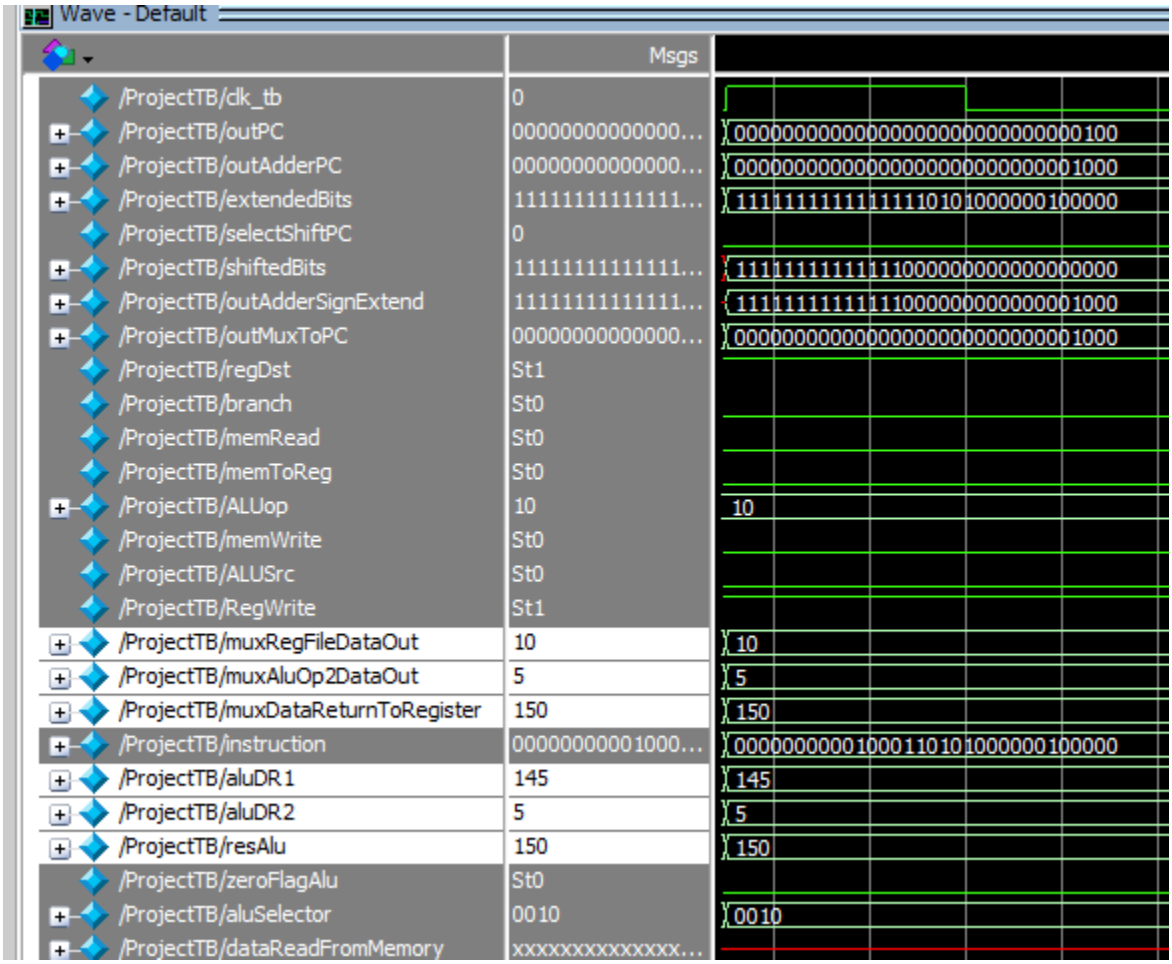
Resultados de la operación OR



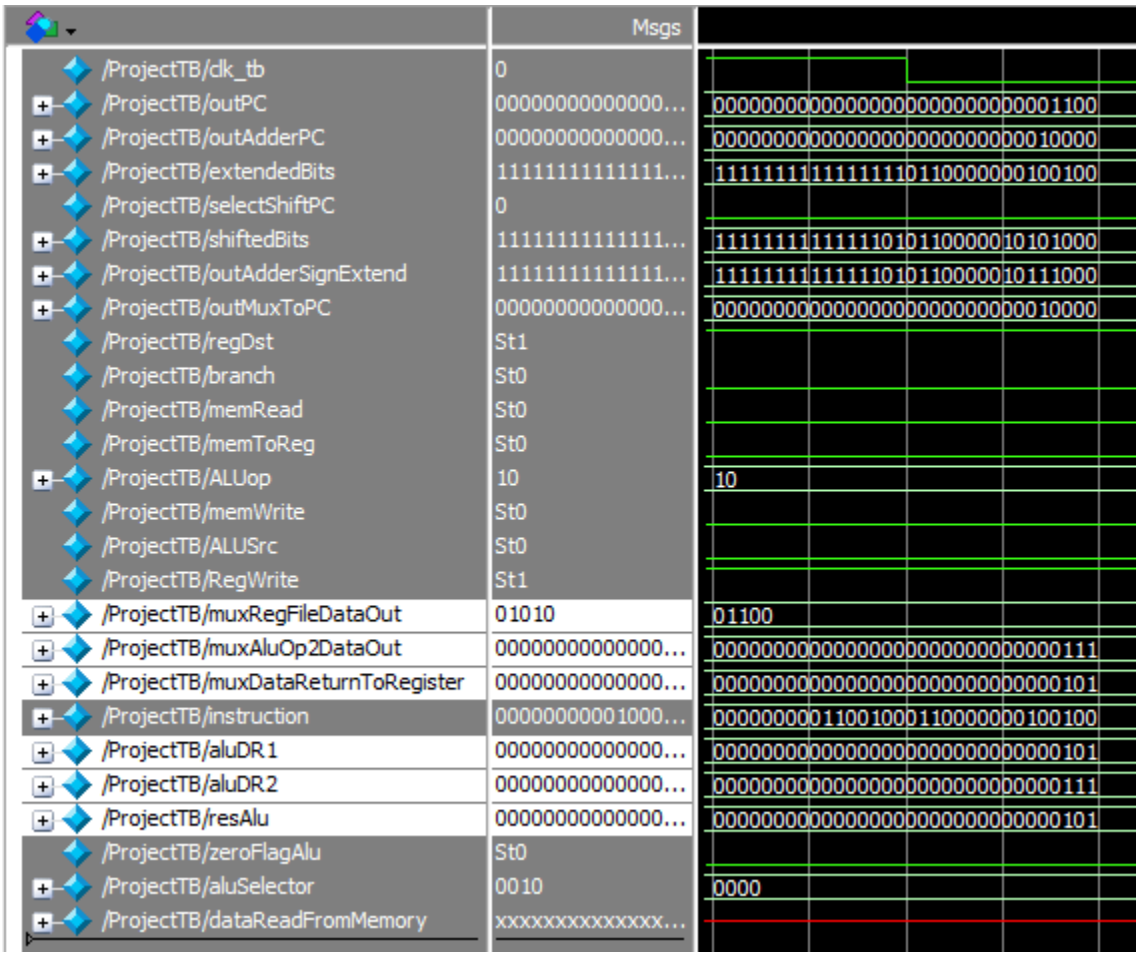
Resultados de la operación SLT:



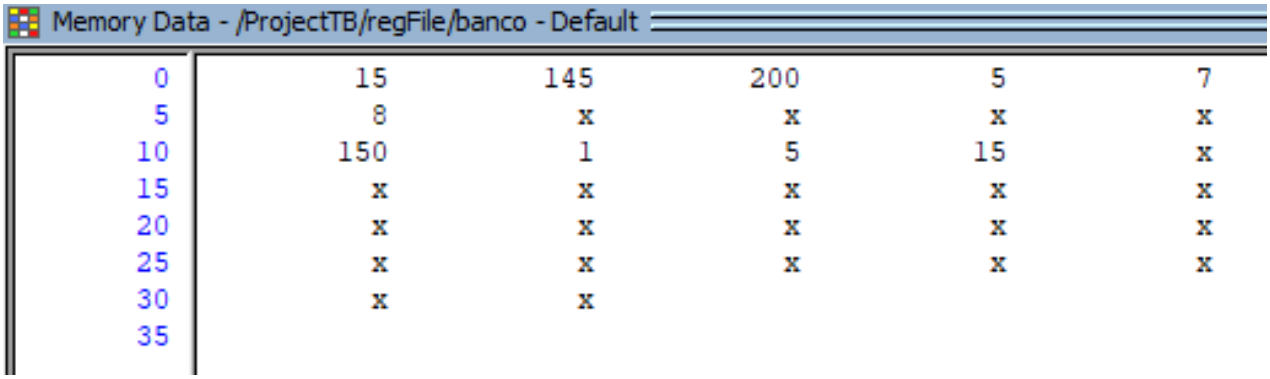
Resultados de la operación Add:



Resultados de la operación AND:



Resultados en banco de registros en decimal:



REFERENCIAS

- "MIPS architecture", Wikipedia, consultado el 2 de mayo de 2023, https://en.wikipedia.org/wiki/MIPS_architecture.
- An Interview with Tenstorrent: CEO Ljubisa Bajic and CTO - AnandTech. https://www.anandtech.com/show/16709/an-interview-with-tenstorrent-ceo-ljubisa-bajic-and-cto-jim-keller.
- Tenstorrent Shifts Leadership Roles - Forbes. https://www.forbes.com/sites/karlfreund/2023/01/18/tenstorrent-shifts-leadership-roles/
- Ljubisa Bajic, Founder and CEO, Tenstorrent - Topio Networks. https://www.topionetworks.com/people/ljubisa-bajic-588b09eb2c537740e300002a
- Jim Keller (engineer) - Wikipedia. https://en.wikipedia.org/wiki/Jim_Keller_%28engineer%29