Centro universitario: CUCEI

Materia: Seminario de solución de problemas de arquitectura de computadoras

Alumno: Ramos Preciado Alan Josafat

Código: 218130165

Carrera: Ingeniería en informática (INNI)

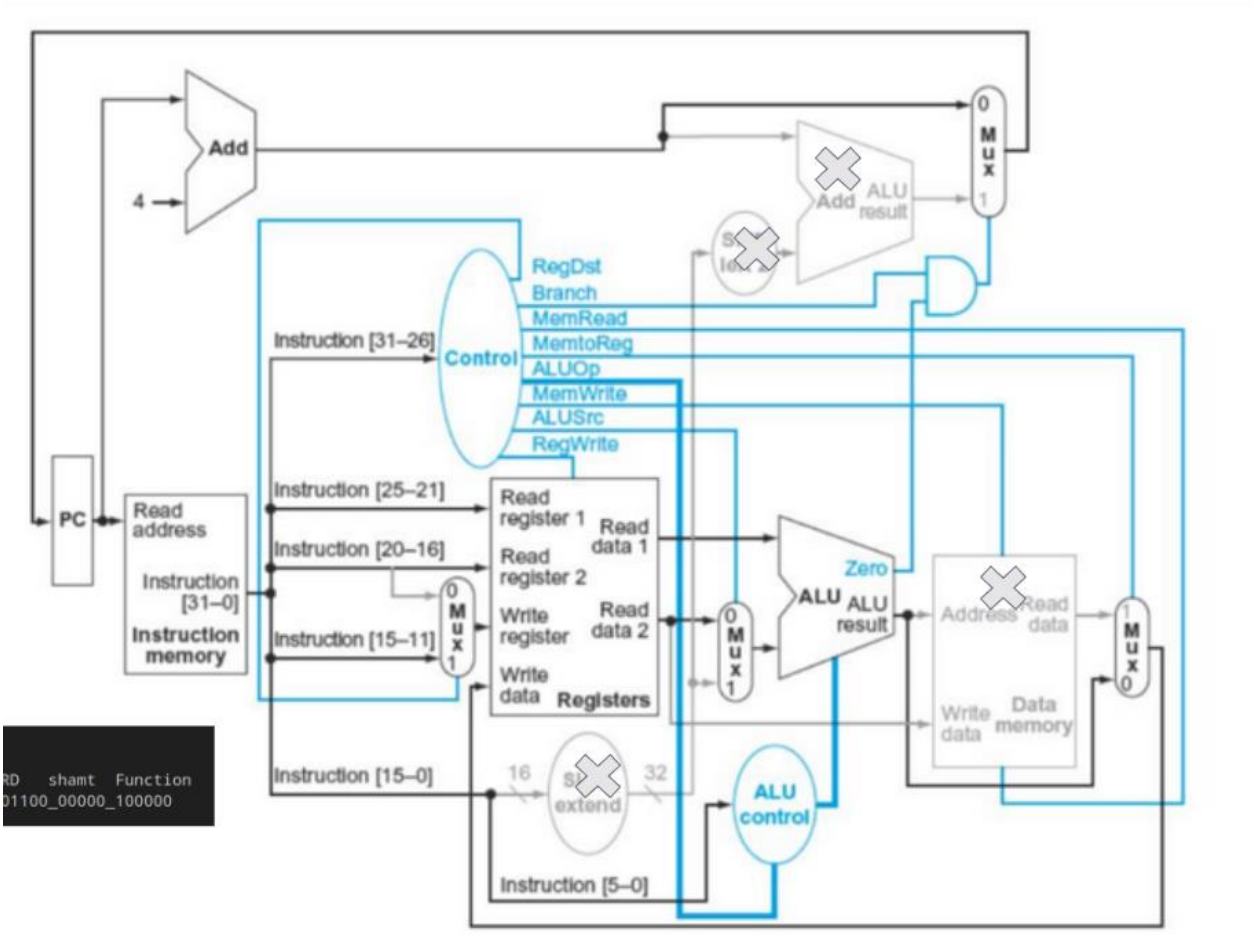
Título: fase 2

INTRODUCCIÓN

Los procesadores manejan diferentes instrucciones estas instrucciones se dividen en diferentes categorías, en mips existe un estándar el cual se describe cual instrucción corresponde a cada actividad que tiene que realizar el procesador

OBJETIVO

Desarrollar el siguiente diagrama que funcione para las diferentes instrucciones

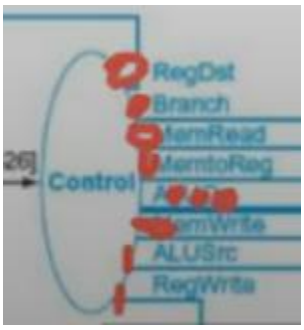


DESARROLLO

Esa es la intruccion que se agregara

Es la intrucciomn de suma inmediata

Lo que se tiene que hacer es agregar en nuedstra alu control como manejaremos todo

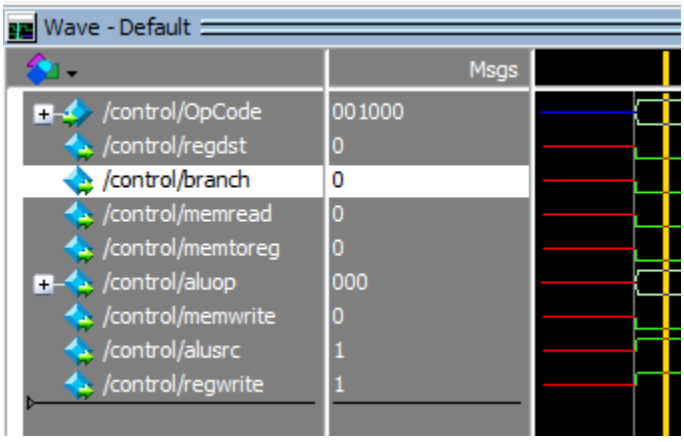


Llo que queremos qes que al final nos quede así

Entonces

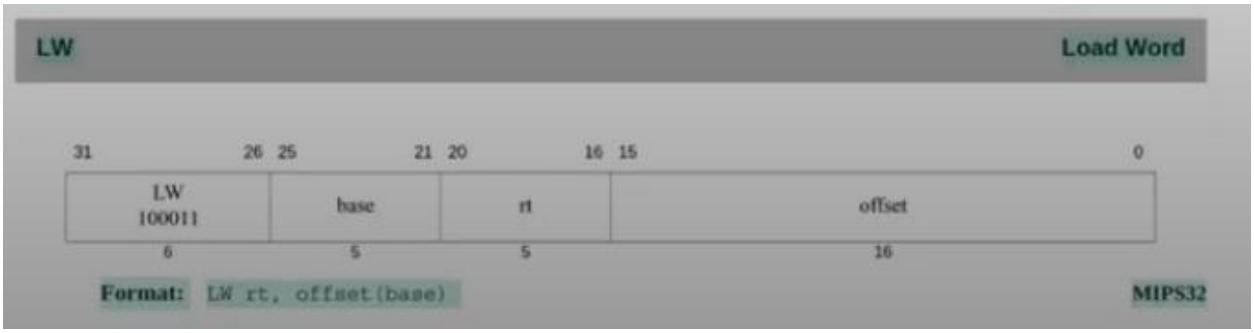
```
6'b001000: //type  addi
begin

    regdst = 1'b0;
    branch = 1'b0;
    memread = 1'b0;
    memtoreg = 1'b0;
    aluop = 3'b000;
    memwrite = 1'b0;
    alusrc = 1'b1;
    regwrite = 1'b1;
```

Así quedaría nuestra nueva control



Así comprobamnos que efectivamente funciona

Se hará lo mismo para todas las nuestras instrucciones.

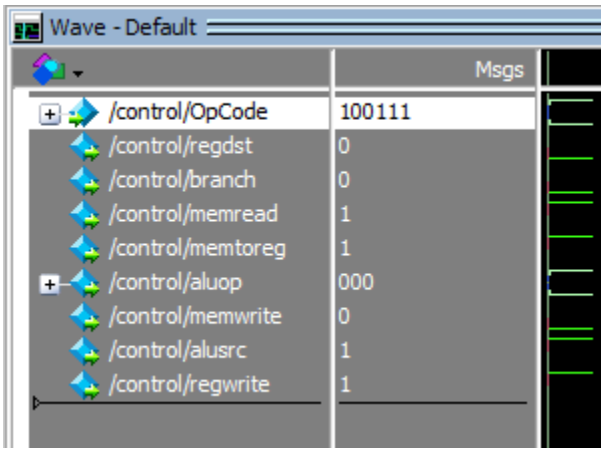


LW

Que lo que recibe control seria

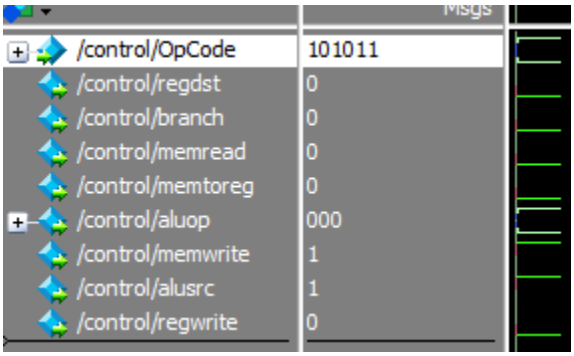
100111

Salida del wave de lw



Para el store Word necesitamos la siguiente op

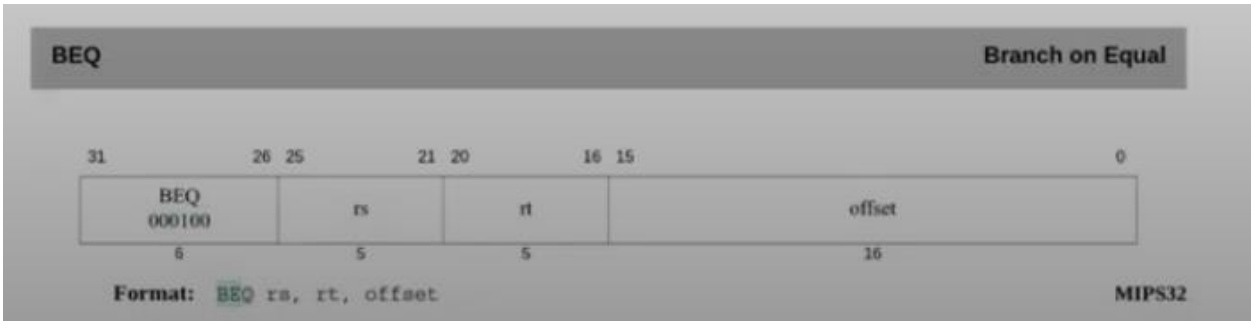
Y nuestra saldia la siguiente



La saldia fue correcta

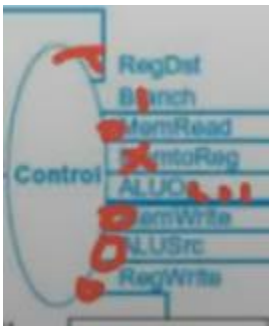
```
    6'b101011: //type store word
begin

    regdst = 1'b0;
    branch = 1'b0;
    memread = 1'b0;
    memtoreg = 1'b0;
    aluop = 3'b000;
    memwrite = 1'b1;
    alusrc = 1'b1;
    regwrite = 1'b0;

en
```

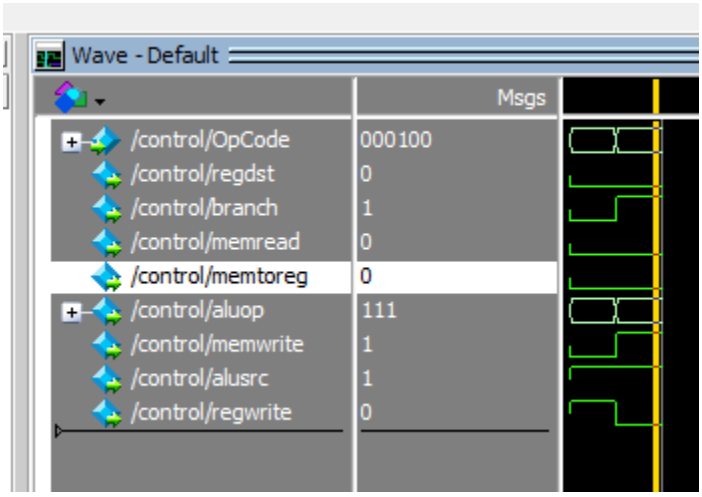
La ultima instrucción es



000100

Saldia esperada





Después de tener todos los controles de todas las instrucciones sigue hacer los módulos que no teníamos con la instrucción de tipo r

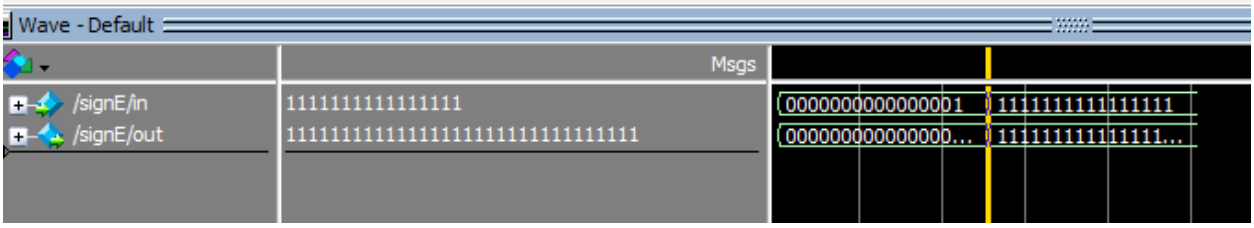
En nuestras instrucciones anteriores no utilizábamos lo anterior

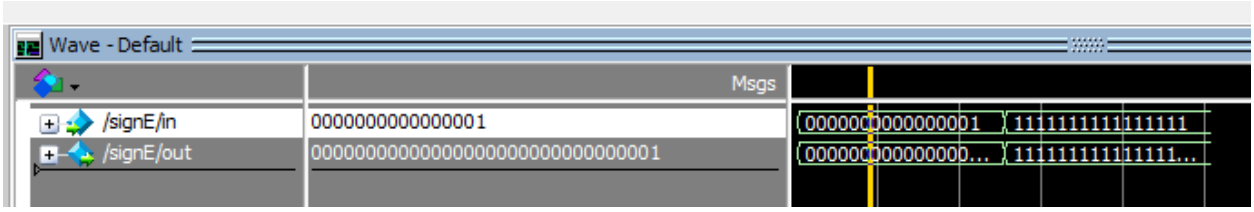
Pero aquí si lo necesitamos se para un numero de 16 bits y lo hacemos de 32 sea negativo o positivo

C:/Users/alan_/Desktop/Fase1/signoE.v - Default

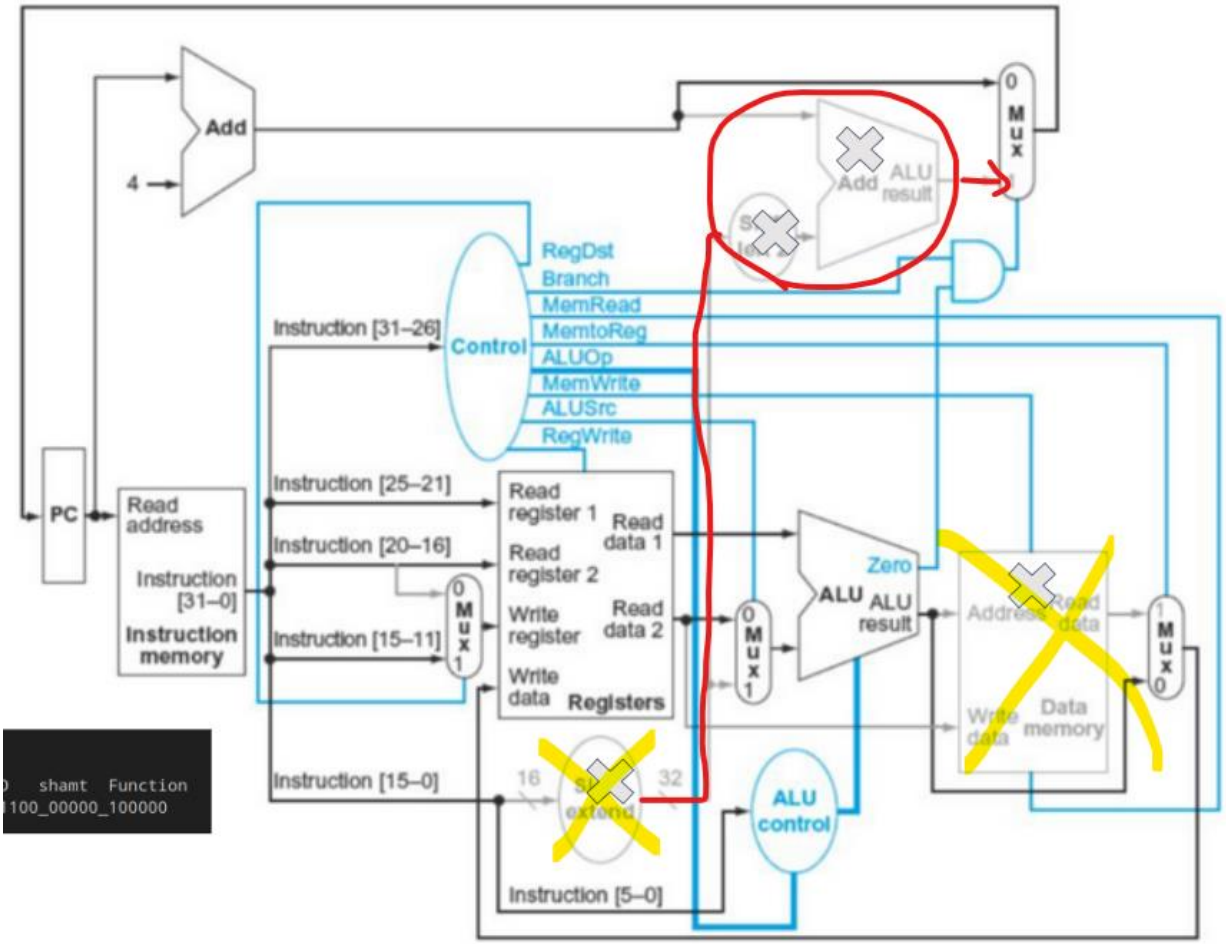
```
Ln#
1   module signE (
2       input wire [15:0] in,  // Entrada de 16 bits
3       output wire [31:0] out // Salida de 32 bits
4   );
5       assign out = { {16{in[15]}}, in };
6       // {16{in[15]}} replica el bit más significativo 16 veces para la extensión
7   endmodule
8
9
```

Comoprobamos en el wave





Ahora lo único que flaltaria seria lo siguiente

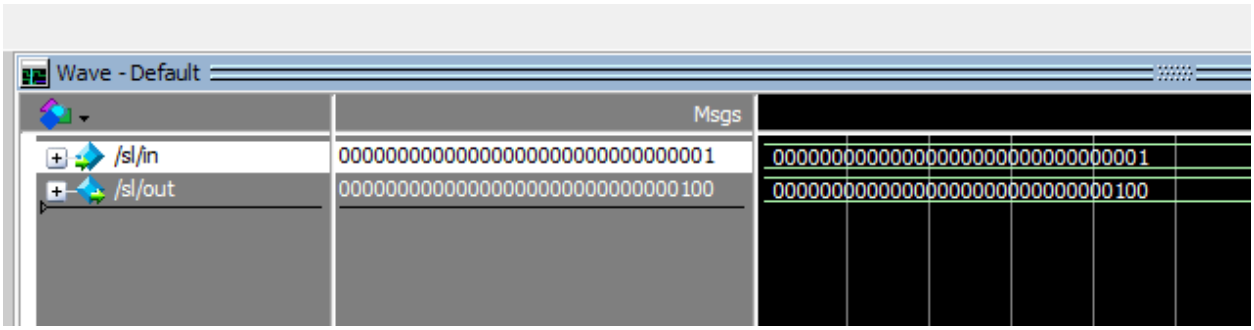


Ya se tiene todo lo que se agrego en esta sergunda fase son lo que esta tachado en amarillo y lo que falta de agregar es el que mueve los signos a la izquierda y la suma de alu con eso

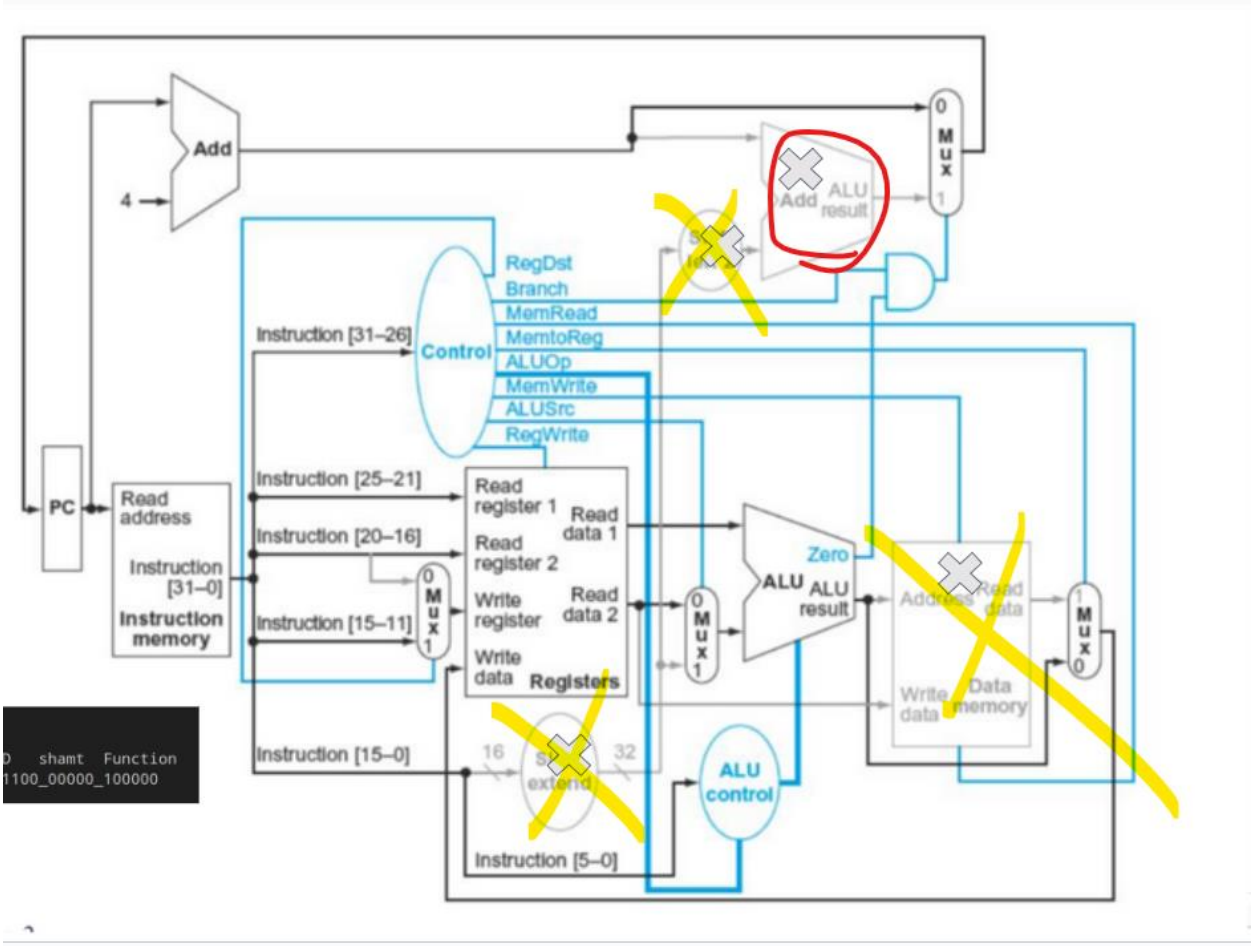
C:/Users/alan_/Desktop/Fase1/sl.v - Default

```
Ln#
1  module sl (
2      input wire [31:0] in,    // Entrada de 32 bits
3      output wire [31:0] out   // Salida de 32 bits
4  );
5      assign out = in << 2; // Desplazamiento a la izquierda por 2 bits
6  endmodule
7
8
```

Aquí se muestra el que mueve los signos



Ya funciona como debería



Tachamos y ya solo queda pendiente el alu result

Que es un add que ya tenemos

madd.v

C:\Users\alan_\Desktop\Fase1\madd.v

```
2
3   module madd(
4       input [31:0] madd_in,
5       input [31:0] madd_four,
6       output reg [31:0] out1  // 'out1' ahora es un registro
7   );
8
9       // Bloque always que se ejecuta cuando 'madd_in' o 'madd_four'
10      always @ * begin
11          out1 = madd_in + madd_four;
12      end
13
14  endmodule
15
```

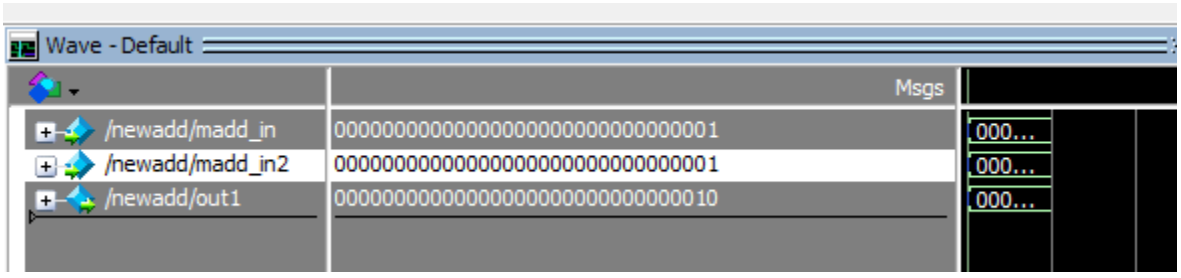
Solo que este add si tiene dos entradas que se darán por cable

Así queda el add new

C:/Users/alan_/Desktop/Fase1/newadd.v - Default *

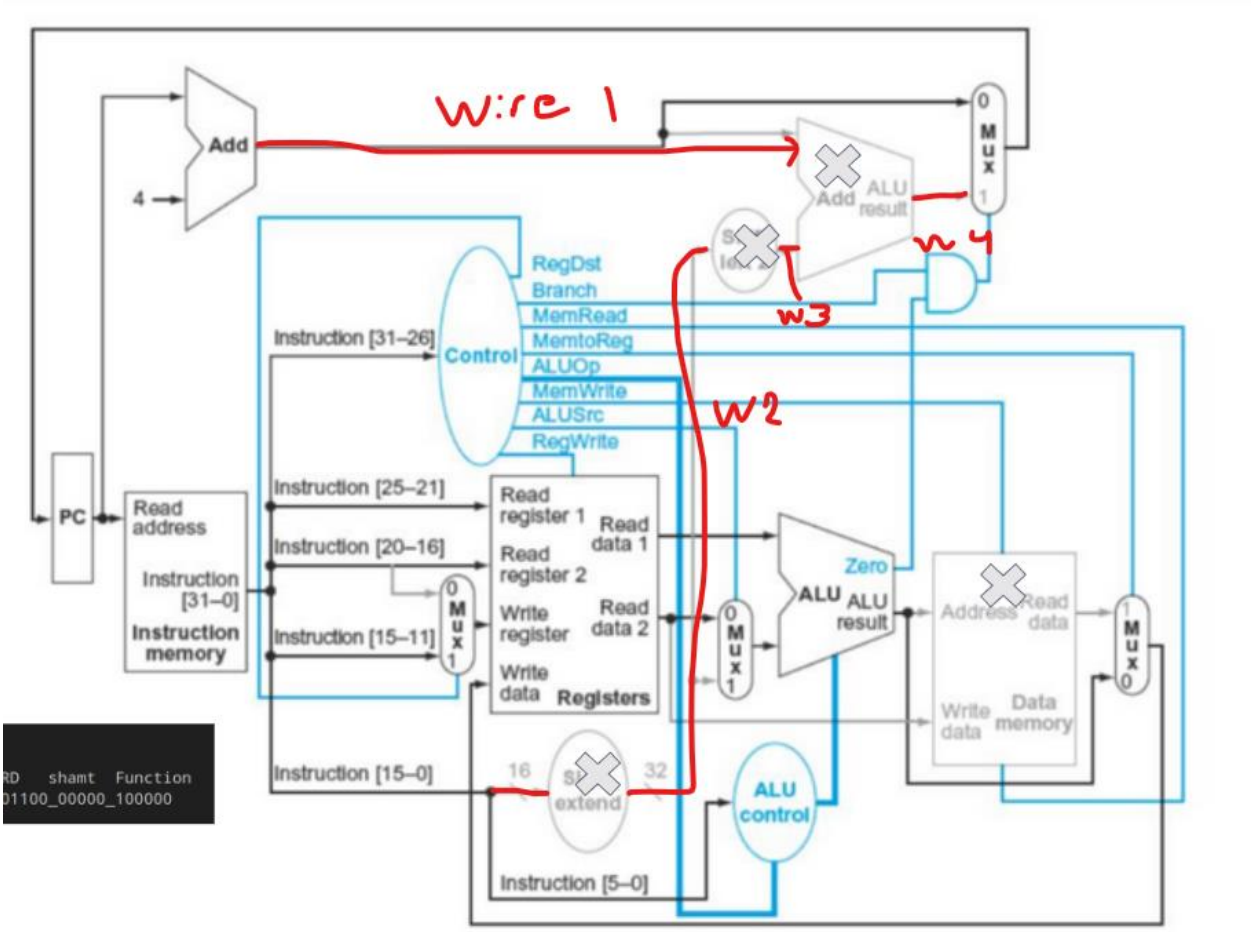
```
Ln#
1   `timescale 1ns/1ns
2
3   module newadd(
4       input [31:0] madd_in,
5       input [31:0] madd_in2,
6       output reg [31:0] out1  // 'out1' ahora es un regis
7   );
8
9
10      always @ * begin
11          out1 = madd_in + madd_in2;
12      end
13
14  endmodule
```

Comprobamos



Ahora lo que faltaría seria crear los módulos

Ahora los cables y los módulos

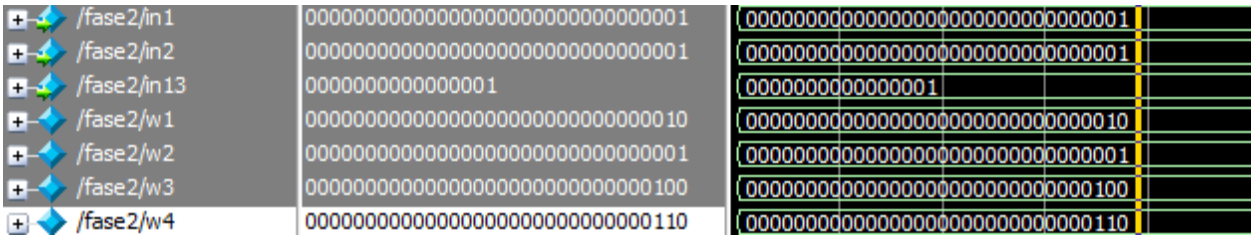



```
wire [31:0] w1;
wire [31:0] w2;
wire [31:0] w3;
wire [31:0] w4;
```

Creamos sus modulos e instancias

```
// Instancia de madd
madd instancemad(
    .madd_in(in1),       // Puerto correcto
    .madd_four(in2),
    .out1(w1)
);

// Instancia de singE
signE sEinstance(
    .in(in13),           // Uso de in13 en lugar de in3
    .out(w2)
);

// Instancia de sl
sl slintance(
    .in(w2),             // Eliminada la coma extra
    .out(w3)
);

// Instancia de newadd
madd nainstance(
    .madd_in(w1),       // Corregidos los nombres de los puertos
    .madd_four(w3),
    .out1(w4)
);
```

Hacemos su prueba



Y ya quedarian listos todos los módulos

CONCLUSIÓN

Me costo mucho trabajo hacer que el modulo completo agregando todas las partes funcionara porque me marcaba datos erróneos. Por eso mismo hice todo en módulos mas pequeños.

Conclusión se me hizo una buena materia para ser estudiante de informática y no haber visto código ensamblador o algo así. Se me hizo de complementario.

REFERENCIAS

MIPS64 Architecture Volume II:The MIPS64 Instruction Set

MIPS® Architecture for Programmers Volume II-A: The MIPS32® Instruction Set Manual