Arquitectura Sparc V8

Sparc V8 architecture

Autor 1: Jairo Andres Angel Morales, *Risaralda , Universidad tecnológica de Pereira, Pereira, Colombia*
Correo-e: andres.angel1@utp.edu.co

***Resumen*— Con este articulo se pretende dar a conocer el los principios básicas para usar la arquitectura sparc y también mostrar las diferentes implementaciones de los módulos de un procesador construido a partir de la misma arquitectura ya mencionada.**

***Palabras clave*— Arquitectura Sparc, Modulos de procesamiento, Procesador, Sparc v8.**

I. INTRODUCCIÓN

SPARC es una arquitectura de conjunto de instrucciones para CPU , derivada de un linaje de computadora con conjunto reducido de instrucciones (RISC). Como arquitectura, SPARC permite un espectro de implementaciones de chip y sistema en una variedad de puntos de precio / rendimiento para una variedad de aplicaciones, incluidas las científicas / de ingeniería, programación, en tiempo real y comerciales.
La introducción puede contener:

- Es complicado entender el funcionamiento de un procesador y laborioso y mucho mas su diseño de hardware
- Al diseñar un hardware nos damos cuenta que es mas complejo de lo pensado y optamos por siempre minimizar el crédito de dicha labor
- Con dicho artículo se pretende hacer un paso a paso y explicar cada uno de los módulos de un procesador basado en la arquitectura Sparc V8 y asi poder facilitar el aprendizaje y fomentar el interés hacia los estudiantes sobre la materia de arquitectura de computadores

II. Palabras Clave

Arquitectura de computadores:

Es el diseño conceptual y la estructura operacional fundamental de un sistema de computadoras. Es decir, es un modelo y una descripción funcional de los requerimientos y las implementaciones de diseño para varias partes de una computadora, con especial interés en la forma en que la unidad central de proceso (CPU) trabaja internamente y accede a las direcciones de memoria, También suele definirse como la forma de interconectar componentes de hardware, para crear computadoras según los requerimientos de funcionalidad, rendimiento y costo.

Instrucción:

Se denomina **instrucción** en informática al conjunto de datos insertados en una secuencia estructurada o específica que el procesador interpreta y ejecuta.

Registro:

En informática, o concretamente en el contexto de una base de datos relacional, un registro (también llamado fila o tupla) representa un objeto único de datos implícitamente estructurados en una tabla. En términos simples, una tabla de una base de datos puede imaginarse formada de *filas* y *columnas* o campos. Cada fila de una tabla representa un conjunto de datos relacionados, y todas las filas de la misma tabla tienen la misma estructura.

VHDL:

(Very High Speed Integrated Circuit Hardware Description Language) Es un lenguaje de programación específico para modelar y diseñar circuitos electrónicos y componentes de Hardware. Aunque su sintaxis tiende a asemejarse a otros lenguajes de programación universales de alto nivel, no deja de ser algo complicado de usar.

III. Entrega de módulos

Para tener un mejor entendimiento del funcionamiento del procesador dividiremos el contenido del de dicho de procesador en 13 módulos los cuales son cruciales para el funcionamiento del mismo.

Entrega	Fecha
1.Modulos básicos para el funcionamiento del procesador	**10 de abril de 2018**
2.Conexion de dichos módulos para mostrar el funcionamiento del mismo	**23 de abril de 2018**

3. Agregar el modulo PSR el cual ayuda con la interpretación de los bits NZVC	**8 de mayo de 2018**
4. Agregar Windows manager para así poder usar el minimo de registros de la arquitectura sparc v8	**18 de mayo de 2018**
5.Agregar y conectar el modulo data memory para poder implementar comandos load y store	**24 de mayo de 2018**

Tabla I, descripción de las entregas y fechas

Entrega 1: Módulos básicos para el funcionamiento del procesador

Modulo 1: NPC(Next program counter), este módulo se encarga de decirle al procesador que instrucción debe leer en ese periodo de procesamiento, se inicializa en cero "0", y va incrementado de a uno en uno gracias a otro modulo llamado sumador, este modulo recibe una entrada de 31 bits y su salida es también de 32 bits que va hacia el memoria de instrucciones la cual leerá la posición de memoria dependiendo de la salida del NPC, a su vez este modulo depende de un reset y un clock o reloj que dependen de los del procesador en si.

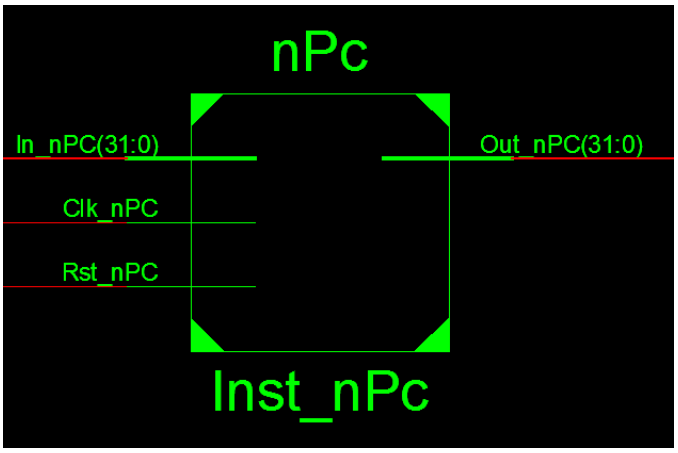


Imagen I, modulo NPC(next program counter).

Modulo 2 : El modulo 2 de nuestro procesador es el sumador el cual es una parte fundamental ya que este modulo es el encargado de incrementar las pociones del NPC y así mandar la nueva posición de lectura a la memoria de instrucciones y hacer que nuestro procesador pueda hacer múltiples tareas de manera secuencial, tiene una entrada de 32 bits la cual proviene del NPC, y esta señal se le suma uno "1", gracias a su programación este valor de adición nunca cambiara y nuestro procesador siempre leerá sus instrucciones en orden de escritura, después de sumarle uno a la entrada, la salida del sumador va otra vez al NPC para indicarle a la memoria de instrucciones que esta es la posición y que esa es la que se debe leer.

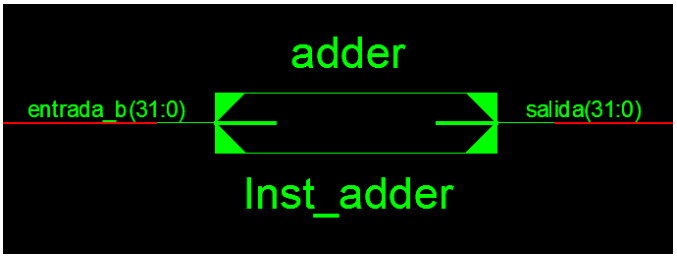


Imagen II, modulo Sumador.

Modulo 3: Intruction memory o memoria de instrucciones, este modulo es el encargado de guardar las instrucciones que le enviamos al procesador, esta memoria recibe una entrada de 32 bits la cual se denomina como dirección de el registro que va a leer el cual tiene contenido la instrucción, las instrucciones están almacenadas en un archivo externo al código, el cual lo almacenamos en lenguaje de maquina el cual es binario es decir solamente unos y ceros "1", "0", teniendo ya la instrucción que también es una salida de 32 bits ya podremos hacer uso de la misma y poner en marcha el procesador.

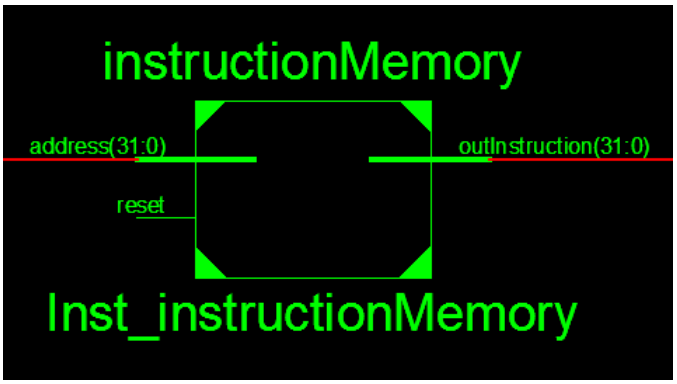
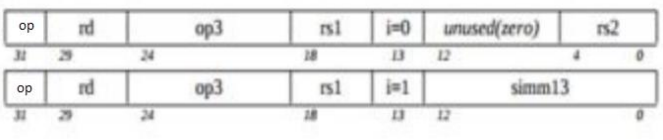


Imagen III, modulo memoria de instrucciones.

La salida de la memoria de instrucciones se divide entre varios modulo ya que la arquitectura sparc v8 los divide de la siguiente manera

Para formato 3:



Op: es el código de operación de dos bits, "2", i va desde el bit 31 al 30, y nos dira que tipo de operación es.

RD: es el registro destino, es decir el numero en binario del el registro que ocupara el resultado de la operación entre RS1 y RS2, va del 29 al 25.

OP3: es el código de la operación la cual se efectuara entre RS1 y RS2, algunas operaciones pueden tener el mismo OP3 pero dependiendo del OP inicial se tendrá en cuenta cual es el OP3 que debemos utilizar va del bit 24 al 19.

RS1: es el registro fuente uno, es decir el numero binario del registro en el cual se almacenara ya sea un numero o un valor, va desde el bit 18 al 14.

i: es el bit que si bien esta en 1 querrá decir que hay un numero inmediato por lo tanto se operara con el, y no se tendría en cuenta un registro fuente 2, ocupa solamente el bit numero 13.

> En caso tal de que no haya inmediato las bits del 14 al 5 seran ceros, y del bit 4 al 0 estara la dirección en binario del registro fuente dos o RS2.

Siimm13: es un numero entero representado en binario de 13 bits llamado inmediato, el cual se se operara con el RS1, va del bit 12 al 0.

Para el formato 2:

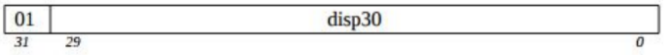
Op: es el código de operación de dos bits, “2”, i va desde el bit 31 al 30, y nos dira que tipo de operación es.

a: es un bit que varia entre 1 y 0 nos dice si se lee la línea que hay debajo de lo que el procesador actualmente antes de saltar (ya que se utilizan solamente en instrucciones de salto como un Branch), utiliza el bit 29.

Cond: es el condicional o bien un binario de 4 bits que nos dice que condición vamos a utilizar funciona igual que el OP3 en el formato 3, va del bit 28 al bit 25.

Disp22: son los espacios de desplazamiento en el register file que el procesador tendrá que hacer para leer la nueva instrucción, va desde el bit 21 al bit 0.

Para el formato 1:



Se utilizan para la instrucción Call, el cual siempre salta y lee la instrucción que esta debajo de la línea que se esta leyendo,

Disp30:, es la cantidad de casillas que debe moverse dentro de la memoria para acceder a la dirección en la cual esta la instrucción que se desea llamar.

Modulo 4: La unidad de control es el modulo que nos lee los bits del 31 al 30 es decir el OP y también los bits del 24 al 19 es decir el OP3, entonces tiene 2 entradas una de 2 bits y la otra de 6 bits, y una salida de 6 bits la cual mandaremos a la Alu con el fin de efectuar la operación correcta, para funcionar con el procesador más básico esta salida funcionaria sin embargo para el procesador que soporte las instrucciones de load y store es necesario agregar, 3 salidas mas cada una de un bit, una para decirle al register file que puede escribir dentro de si, otra para el data memory para que la deje usar sus registros y el otro para que el multiplexor del fnal del procesador nos deje pasar ya sea la respuesta de la Alu o al contenido de la memoria en caso de hacer un Load.

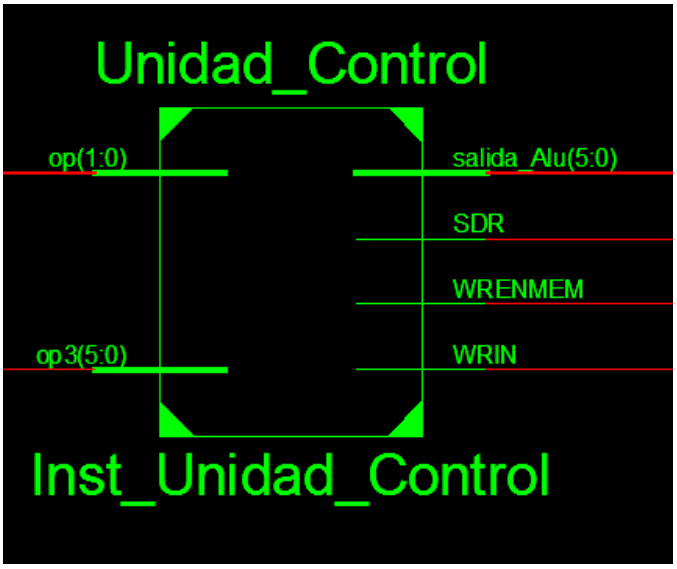


Imagen IV, modulo unidad de control.

Modulo 5: Register file (Archivos de registro), en este modulo es donde se almacenan los valores ingresados desde la memoria de instrucciones, para esto tenemos 4 entradas, registro fuente 1(RS1), registro fuente 2(RS2), registro destino(RD), y el escritor de memoria o data write(DWR), para el RS1 utilizaremos los bits 18 al 14 de la salida de la memoria de instrucciones, para el RS2 utilizamos los bits del 4 al 0, el RD del bit 29 al 25, y por ultimo el DWR que proviene de la Alu y es el resultado de la operación entre RS1 y RS2 y se almacenara en la dirección que tenga el RD, de salida tenderemos el contenido de lo que estaba almacenado dentro de la dirección del RS1 y el RS2 es decir 2 salidas, esto se tiene en cuenta para el funcionamiento mas básico del procesador, pero eventualmente se añade una entra la cual proviene de la unidad de control la cual nos permite escribir dentro del register file en caso de un load y también se añade una salida ya que siempre vamos a necesitar y tener en cuenta el contenido de lo que había en el RD en caso de hacer un store.

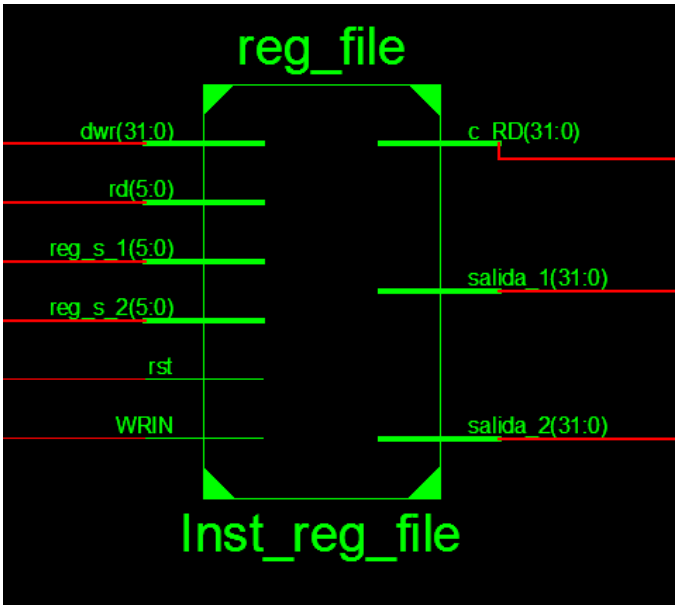


Imagen V, Register file.

Modulo 6: SEU, o unidad de extensión de signo, esta unidad le llegan los bits del 12 al 0 en caso de tener un inmediato en vez de un registro fuente 1, ya que este inmediato llega con una longitud de solamente 13 bits y la Alu especula que debe tener una entrada de 32 bits por lo tanto la SEU coge el bit mas significativo es decir el bit 12 y dependiendo si es 1 o 0 coloca del bit 31 al bit 13 del mismo modo que el valor del bit 12 para así lograr formar un inmediato de 32 bits, recibe una entrada de 13 bits y devuelve una salida de 32 bits que va hacia el multiplexor.

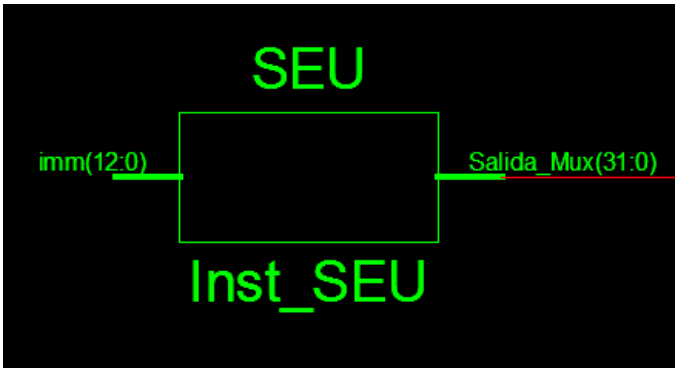


Imagen VI, unidad de extensión de signo

Modulo 7: Multiplexor, este modulo recibe el bit numero 13 de la instrucción que proviene de la memoria de instrucciones, también recibe el inmediato ya transformado a 32 bits y también la salida del Register file que corresponde al contenido del registro fuente 2, este modulo hace el papel de switch ya que si el bit 13 esta en 1 el deja pasar la salida de la SEU ya que significa que hay inmediato, en caso tal de que el bit 13 este en 0 significa que no lo hay por lo tanto dejaría pasar la salida que corresponde al contenido del registro fuente 2.

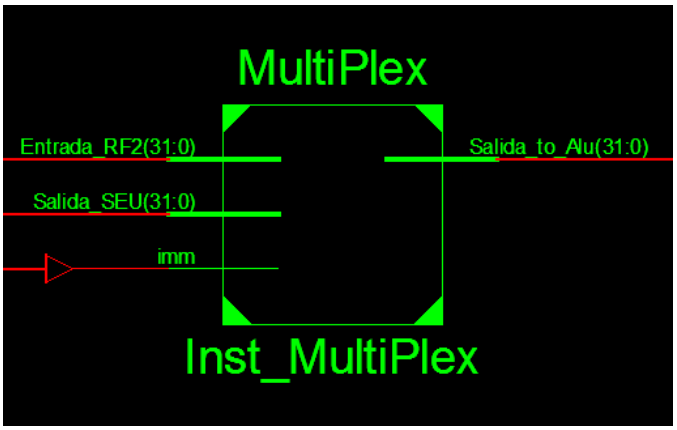


Imagen VII, multiplexor.

Modulo 8: ALU(Unidad aritmeticológica), este modulo es el que hace la operación entre el registro fuente 1 y el registro fuente 2 o inmediato de la instrucción que sale del register file, este modulo recibe 3 entradas, la salida que corresponde al contenido del registro fuente 1 , la salida del multiplexor, ambas de 32 bits y la salida de la unidad de control la cual nos indica cual será la operación que la ALU debe realizar con los dos operandos, y una salida también de 32 bits la cual corresponde al DWR que va al Register file para almacenar el resultado en el registro destino, y a su vez también es la salida o respuesta del procesador.

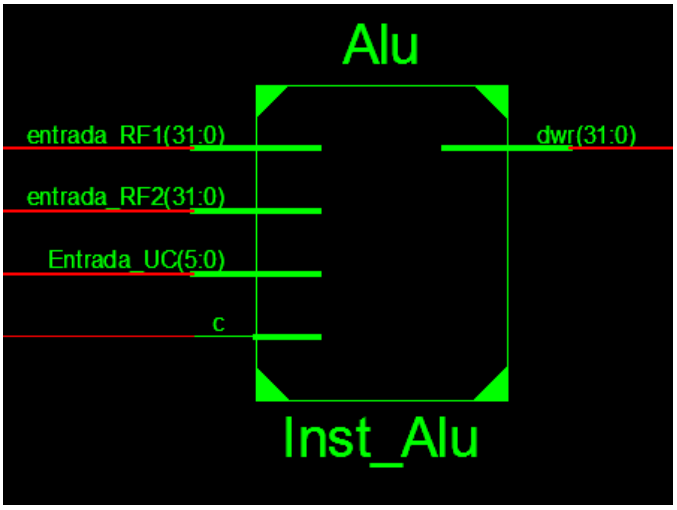


Imagen VIII, Alu(Unidad aritmeticológica)

Entrega 2: conexión de cada uno de los módulos y así formar un solo modulo el cual será el procesador como tal .

Para hacer la conexión del procesador deberemos crear otro modulo dentro de la plataforma Xilinx en la cual estamos trabajando , damos click derecho en sobre la placa que estamos usando y después le damos click en “New source”, allí abrirá una ventana en la cual escogeremos VHDL source y le pondremos un nombre.

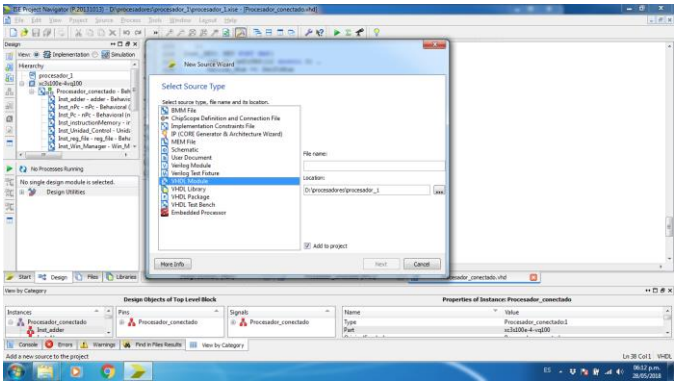


Imagen IX, creando el modulo del procesador

Después de hacer esto empezamos a agregar cada uno de los módulos a dicho modulo que seria nuestro procesador, para hacer eso denominamos cada uno de nuestros módulos como “top module”, y de esta manera aparecerá la opción de “View HDL instatiation template”, el Xilinx nos abrirá una nueva venta en la cual veremos algo asi:

```
COMPONENT nPc
PORT(
    In_nPC : IN std_logic_vector(31 downto 0);
    Rst_nPC : IN std_logic;
    Clk_nPC : IN std_logic;
    Out_nPC : OUT std_logic_vector(31 downto 0)
    );
END COMPONENT;

Inst_nPc: nPc PORT MAP(
    In_nPC => ,
    Rst_nPC => ,
    Clk_nPC => ,
    Out_nPC =>
);
```

Imagen X, témplate para conectar

Toda esta parte debemos pegarla dentro del modulo del procesador y debe ser distribuida de la siguiente manera, la parte del Component asi:

```
26
27  -- Uncomment the following library declaration if instantiating
28  -- any Xilinx primitives in this code.
29  --library UNISIM;
30  --use UNISIM.VComponents.all;
31
32  entity Procesador_conectado is
33      Port ( rst : in  STD_LOGIC;
34             clk : in  STD_LOGIC;
35             Salida_procesador : out  STD_LOGIC_VECTOR (31 downto 0));
36  end Procesador_conectado;
37
38  architecture Behavioral of Procesador_conectado is
39  COMPONENT adder
40     PORT(
41        entrada_b : IN std_logic_vector(31 downto 0);
42        salida : OUT std_logic_vector(31 downto 0)
43        );
44     END COMPONENT;
45
46     COMPONENT nPc
47     PORT(
48        In_nPC : IN std_logic_vector(31 downto 0);
49        Rst_nPC : IN std_logic;
50        Clk_nPC : IN std_logic;
51        Out_nPC : OUT std_logic_vector(31 downto 0)
52        );
53     END COMPONENT;
```

Imagen XI, componente

Después de agregar el componente debemos ir agregando las señales que nos conectaran los módulos es como decir los cables que llevan la información de un modulo al otro, se deben declarar de la siguiente manera:

```
162  signal ucToAlu, winToRF1, winToRF2, winToRFD : STD_LOGIC_VECTOR (5 downto 0);
163  signal PSRMToPSR  : STD_LOGIC_VECTOR (3 downto 0);
164  signal PSRToWin, WinToPsr : STD_LOGIC_VECTOR (1 downto 0);
165  signal PSRtoAlu, UCtoMUX0, UCtoDM, UCtoRF: STD_LOGIC;
```

Imagen XII, declaración de variables

Después de tener las variables creadas procedemos a interconectar los módulos ya que anteriormente solamente los hemos declarado, se hace de la siguiente manera:

```
166  begin
167  Inst_adder: adder PORT MAP(
168        entrada_b => npcToAdder,
169        salida => adderToNpc
170     );
171
172  Inst_nPc: nPc PORT MAP(
173        In_nPC => adderToNpc,
174        Rst_nPC => rst ,
175        Clk_nPC => clk,
176        Out_nPC => npcToAdder
177     );
178  Inst_Pc: nPc PORT MAP(
179        In_nPC => npcToAdder,
180        Rst_nPC => rst ,
181        Clk_nPC => clk,
182        Out_nPC => npcToIm
183     );
184  Inst_instructionMemory: instructionMemory PORT MAP(
185        address => npcToIm,
186        reset => rst,
187        outInstruction => imToURS
188     );
273  end Behavioral;
```

Imagen XIII, estableciendo las conexiones

Después de hacer esto con cada uno de los módulos terminaremos nuestro procesador 1 y tendremos que hacer las conexiones como plantea el siguiente diagrama

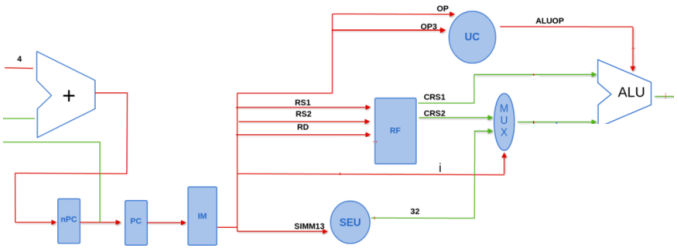


Imagen XIV, diagrama de procesador conectado.

Entrega numero 3: se agregan los módulos PSR y el PSR modifier al procesador.

Modulo 9: PSR modifier, esta modulo es la que nos modifica los 4 bits NZVC los cuales le dan agregado al valor de la Alu es decir nos indicando que :

N: que el numero es negativo

Z: que el resultado de la ALU es cero "0".

V: que el resultado tiene overflow, es decir que el resultado supera las capacidad del procesador y no puede generar una respuesta tan grande o tan pequeña.

C: que la operación que se acabe de efectuar lleva carrie es decir que al resultado de la próxima instrucción se le deberá sumar uno dado que la operación que recién se efectuó lleva uno de mas como lo seria la suma de dos números negativos en binario.

Para saber que operación se necesitaba efectuar debimos consulta el libro **The SPARC Architecture Manual,** en las paginas 173, 174, 175, este modulo recibe 4 entradas las cuales proviene la salida que corresponde a la salida al contenido del registro fuente 1, la salida de la unidad de control, la salida del multiplexor y la salida de la ALU, también consta de una salida de 4 bits la cual corresponde a los bits NZVC y van hacia el modulo PSR.

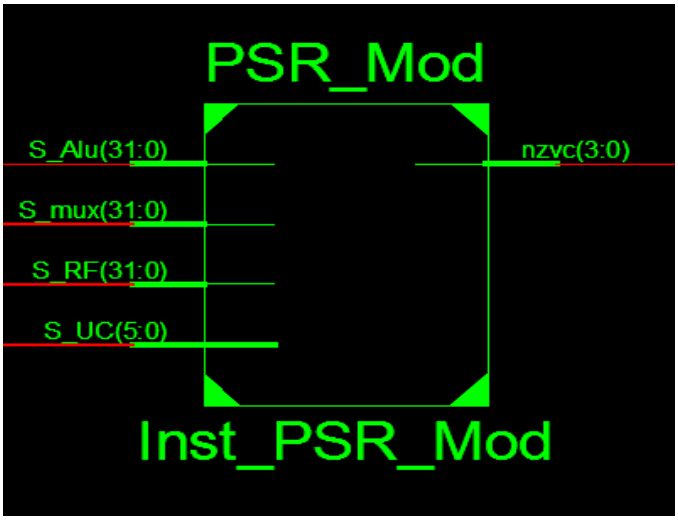


Imagen XV, PSR modifier.

Modulo 10: PSR, en este modulo se almacenan los bits NCVZ de la ultima instrucción realizada por el procesador, consta de 1 entrada de 4 bits proveniente del PSR modifier, y una salida de 1 bit que va hacia la Alu ya que es la que debe tener en cuenta si hay carrie o no para asi sumarlo en caso tal de que esa sea la situación, así funcionaria sin la función de ventaneo ya que para la próxima conexión debemos añadir una entrada la cual es el CWP(Current Windows pointer) la cual es de 2 bits.

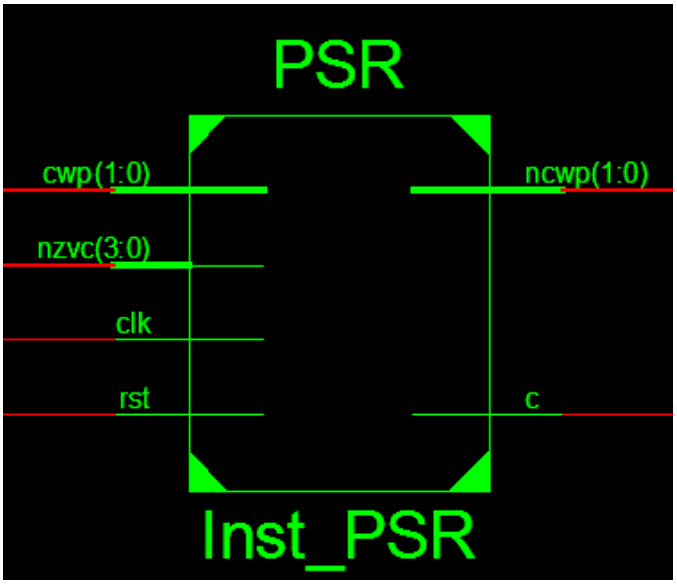


Imagen XVI, modulo PSR

Después de haber hecho los dos módulos, PSR y PSR modifier, podemos conectar dichos módulos al procesador que ya habíamos implementado, de la misma manera que se muestra en la entrega 2, el diagrama de la conexión debe quedar de la siguiente manera.

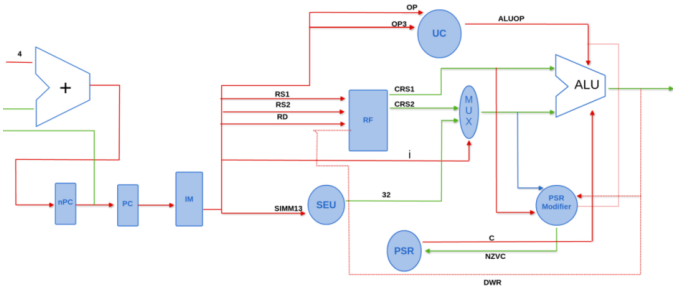


Imagen XVII, Diagrama de la entrega 3

Entrega numero 4: se agrega el modulo de Windows manager.

En esta entrega se elabora el Windows manager para poder asi hacer uso del minimo de registros de la arquitectura sparc V8 los cuales son 40, y como en este momento usamos registros de 5 bits solamente podremos utilizar 32, lo que debemos hacer es crear dicho modulo para así cambiar el numero de bits a 6 y poder organizar los registros de la siguiente manera:

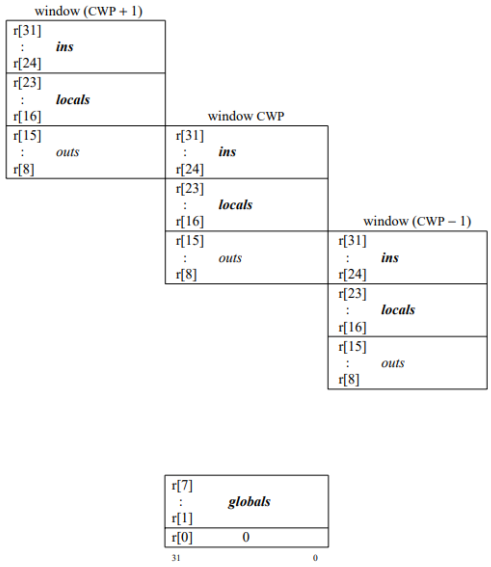


Imagen XVIII, registros correspondientes a ventanas.

También podremos movernos entre ventanas con las instrucciones save y restore, pero debemos tener en cuenta la siguiente tabla para hacer posible el movimiento

	Ventana 1	Ventana 2
In [7] – In [0]	R [31] – R [24]	R [15] – R [8]
Local [7] – Local [0]	R [23] – R [16]	R [39] – R [32]
out [7] – out [0]	R [15] – R [8]	R [31] – R [24]
global [7] – global [0]	R [7] – R [0]	R [7] – R [0]

Tabla II, registros en las ventanas.

Se debe tener en cuenta ya se resta o suma a los registros de a 16 desde la ventana donde estoy dependiendo si es un save o un restore.

Modulo 11: Windows Manager, o manejador de ventanas, es el que con la ayuda del PSR me dice en que ventana estoy y en que registro debo escribir dentro del register file dependiendo en que ventana se encuentra el procesador, esto nos permite manejar 40 registros con dos ventas y mejorar el rendimiento del procesador ya que tenemos mas registros, esto conlleva que hay que hacerle una modificación al Register file ya que antes estaba hecho para recibir registros de 5 bits ahora se debe hacer la modificación para que almacene registros o direcciones de 6 bits, y también se debe modificar el timpo de memoria para que ya no tenga solamente 32 registros sino 40, también se debe modificar el PSR ya que debe tener una entrada y una salida mas para que se le envié el NCWP(current Windows pointger) y una salida la cual se le envía al Windows manager para saber en que ventana esta parada y cuales registros debe acceder el Register file, este modulo cuenta con 6 entradas las cuales son todas las salidas del instruction memory, mas la señal del CWP proveniente del PSR, se requiere saber el op y el op3 para tener en cuenta en caso tal de que se haga un cambio de ventana, las salidas son los nuevos registros 1, 2 y el nuevo RD convertidos a 6 bits cada uno, también sale el nuevo CWP hacia el PSR en caso tal de que se haga un cambio de ventana.

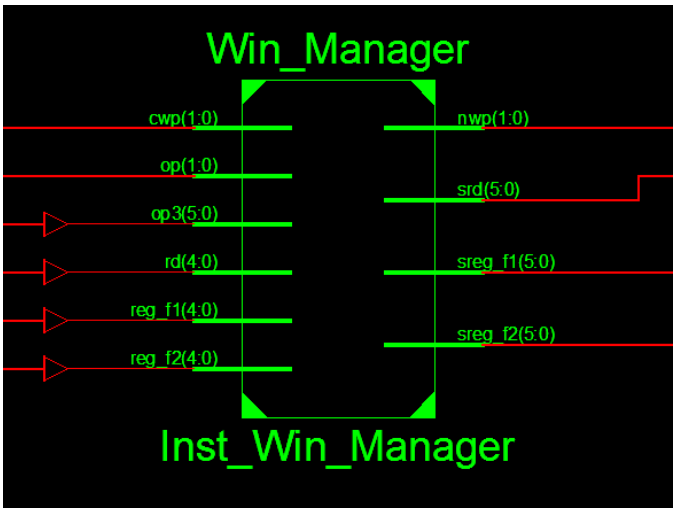


Imagen XIX, Windows manager.

Después de conectar siempre de la misma forma que en la primera entrega el diagrama del procesador debería verse de la siguiente manera:

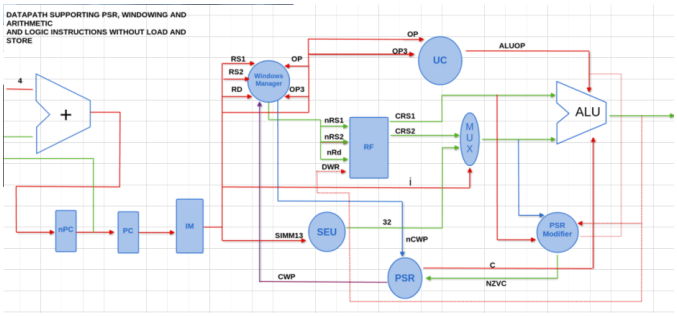


Imagen XX, procesador con Windows manager.

Entrega numero 5: se agregan los modulos de data memory y un nuevo multiplexor el cual nos permite ejecutar instrucciones de memoria como load y store.

Para esta entrega se realizan los últimos dos modulos para poder completar nuestro procesador, ya que podremos crear una memoria de datos en la cual podremos almacenar nuestros registros y de esa manera poder mejorar el rendimiento de nuestro procesador ya que podremos tenr muchos datos en cuentas y ahorrar instrucciones.

Antes de empezar a realizar nuestros dos modulos es necesario hacer unas modificaciones significativas a nuestro procesador, lo primero que debemos modificar es nuestra unidad de control añadiendo 3 salidas mas de un bit cada una, esto nos servirá para mandarlas al register file, al data memory y al nuevo multiplexor, esto con el fin de hacerlos funcionar de diferente manera en caso de un load o un store.

Tambien es necesario agregar una entrada al Register file para que la nueva salida de la unidad de control se integre al modulo y crear una condición de que en caso de que se este haciendo un Load se le permita escribir en el Register file y en caso de un save que no lo deje escribir , también se le adiciona una nueva salida la cual es el contenido del registro destino, ya que allí esta el valor que se quiere guardar el la memoria de datos.

Modulo 12: Data memory, este modulo nos permite tener una memoria muy similar al Register file con la diferencia de que la utilizaremos cuando creamos que es necesario, y podrá tener el espacio que deseemos o consideremos crucial para nuestro proyecto, este consta de 3 entradas, las cuales provienen del resultado de la Alu la cual nos dara la dirección del registro en el cual el procesador deberá guardar el dato, el contendí del registro destino el cual proviene de del register file y es el dato que guardaremos en la dirección de registro que la Alu nos provea, ambas señales de 32 bits, por ultimo es la señal que proviene de la unidad de control y es únicamente de un bit, este bit dependiendo de su valor nos permitirá escribir en la memoria de datos en caso de ser un store, o nos dejara leer dichos registros y mandarlos a la salida de 32 bits en caso de ser un load.

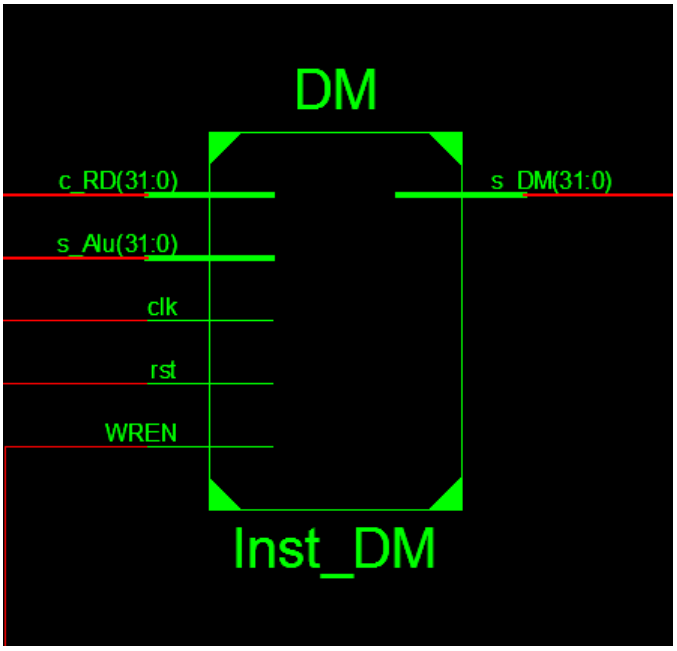


Imagen XXI, modulo de memoria de datos.

Modulo 13: Multiplexor de la memoria de datos, este es el ultimo componente de nuestro procesador y funciona exactamente igual al anterior multiplexor fabricado, ya que recibe una señal de un bit proveniente de la unidad de control, y dependiendo si es un store o un load el deja pasar como respuesta la respuesta de la memoria de datos en caso de ser un load o un save, o deja pasar la señal de la alu en caso de ser una operación aritmeticológica.

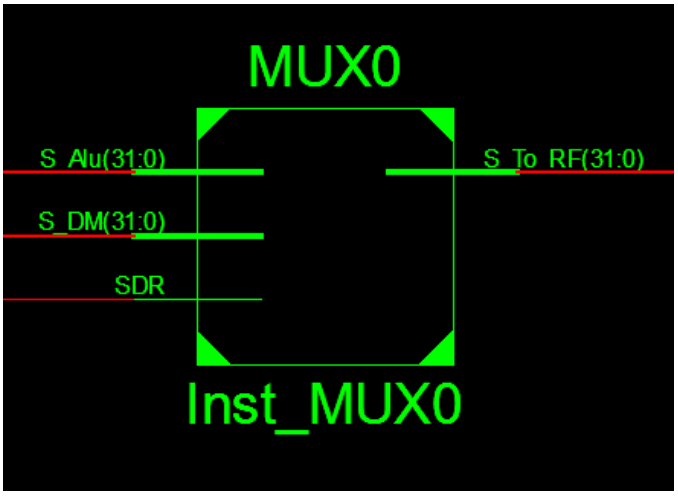


Imagen XXII, multiplexor de la memoria de datos.

Teniendo ya todos los módulos implementados podemos conectarlos siempre de la misma manera explicada en la entrega 1, tras hacerlo el procesador debe lucir de la siguiente manera

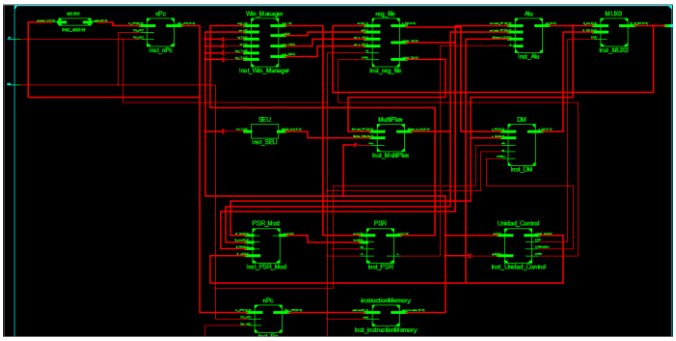
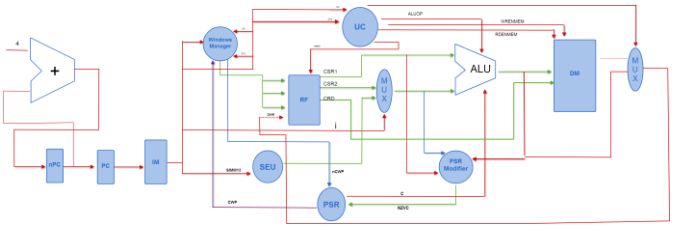


Imagen XXIII, procesador conectado(img Xilinx)

O también se puede apreciar con el siguiente diagrama.



Teniendo correctamente conectado el procesador podremos hacer cualquier tipo operación ya sea de memoria, cambio de ventana, o aritmeticológica con resultados satisfactorios, finalmente se recomienda también utilizar las instrucciones dedicadas a la arquitectura sparc v8 que se pueden encontrar en el link http://moss.csc.ncsu.edu/~mueller/codeopt/codeopt00/notes/sparc.html, en el cual se podrá encontrar toda la documentación necesaria para poder implementar dichas instrucciones, en el link también se encuentra las expresiones del lenguaje ensamblador, el cual nos ayudara a tener un mejor concepto de como es que funciona esta arquitectura.

Bibliografia

Manual de la arquitectura:
https://www.gaisler.com/doc/sparcv8.pdf

Xilinx:
https://www.xilinx.com/

Xilinx fórums:
https://forums.xilinx.com/

Conexión de procesador :

https://www.youtube.com/watch?v=mjBEF5mwWYM

https://www.educreations.com/lesson/view/procesador-sparc-v8-soportando-windowing/13230159/

https://www.youtube.com/watch?v=mDLdiAA1d_E&authuser=3

https://www.youtube.com/watch?v=qifLAgU77uE&t=4s&authuser=3

Instrucciones Sparc:
http://moss.csc.ncsu.edu/~mueller/codeopt/codeopt00/notes/sparc.html

Sparc V8 Assembly Book:
https://en.wikibooks.org/wiki/SPARC_Assembly

Sparc Technical documentation:
https://sparc.org/technical-documents/

Design and verification of sparc V8 book:

https://books.google.com.co/books/about/Design_and_Verification_of_a_Variable_la.html?id=H8OArgEACAAJ&redir_esc=y

Wikipedia:
https://es.wikipedia.org/wiki/Sun_SPARC