Implementación y Simulación de un "Pipelined Processing Unit" para la Arquitectura POWER

Fase I

1. Simulación de Componentes

1.1 Objetivos

1. Familiarizar al estudiante con la implementación de circuitos digitales mediante el uso del leguaje de desarrollo de hardware Verilog.

2. Desarrollar componentes de hardware que eventualmente serán integrados al circuito del procesador que será implementado como parte del proyecto.

1.2 Tarea

En esta fase cada integrante del grupo, independientemente, implementará y simulará en Verilog uno de los siguientes tres grupos de componentes:

1. **Arithmetic Logic Unit y Comparador** (sección 2.1)
2. **Register File y Register Handler** (sección 2.2)
3. **PC y Target Address Generator** (sección 2.3)

Cada estudiante es solo responsable del grupo de componentes que le toque. En casos de grupos de proyecto de menos de tres integrantes no serán responsables del grupo de componentes que se quede sin asignar. Eventualmente los tendrán que implementar, pero no serán responsables de ellos en esta etapa.

1.3 Reglas de Juego Para la Demostración

Además de mostrar lo que se solicita para cada caso, puedo pedirles que alteren su código de Verilog para mostrar otras cosas. Es muy probable que les haga preguntas sobre el código que desarrollaron. Si usted no puede contestar algunas preguntas sobre el código, que alguien que lo desarrolló debería contestar, entonces por deducción lógica se cuestionará la autoría del código y como consecuencia se asignará una puntuación de cero para esta fase.

Los/las estudiantes que no completen esta fase antes de la fecha de entrega de la Fase II se les adjudicará una nota de cero y quedaran desasociados de su grupo original. Podrán

continuar con el proyecto de manera independiente o formando grupo con otros/as estudiantes en iguales circunstancias. Sin embargo, deben completar esta fase, no mas tardar, tres días antes de la fecha de entrega de la Fase III.

1.4 Entrega

1. Subir un documento de texto (puro texto) con el código Verilog de los módulos del/los componentes que le ha tocado y un módulo de prueba del mismo. El nombre de los archivos debe ir precedido por el keyword F1 seguido de su primer apellido, segundo apellido y su nombre, unidos con el signo de underscore. Si yo fuese a someter un módulo del RAM el nombre del file sería el siguiente:

   F1_Rodriguez_Rivera_Nestor_ram.v

2. Demonstrar la operación de el/los componentes según se indique en las instrucciones correspondientes a el/los componentes.

1.5 Rúbrica de Evaluación

- Se adjudicarán 20 puntos si el componente opera correctamente.

- Se podrían adjudicar puntos parciales en caso de que el componente no funcione correctamente dependiendo de cuán avanzado esté el diseño y la simulación del mismo.

- Se asignará una puntuación de cero a cada componente para el que no se someta código en Verilog.

- Se descontará un punto por cada día lectivo que transcurra sin que se demuestre la operación correcta del (los) componentes correspondientes a partir de la fecha que se establezca para realizar la demostración.

2. Componentes

2.1 Arithmetic Logic Unit y Comparador

2.1.1 Tarea

Implementar el circuito que se muestra a continuación.

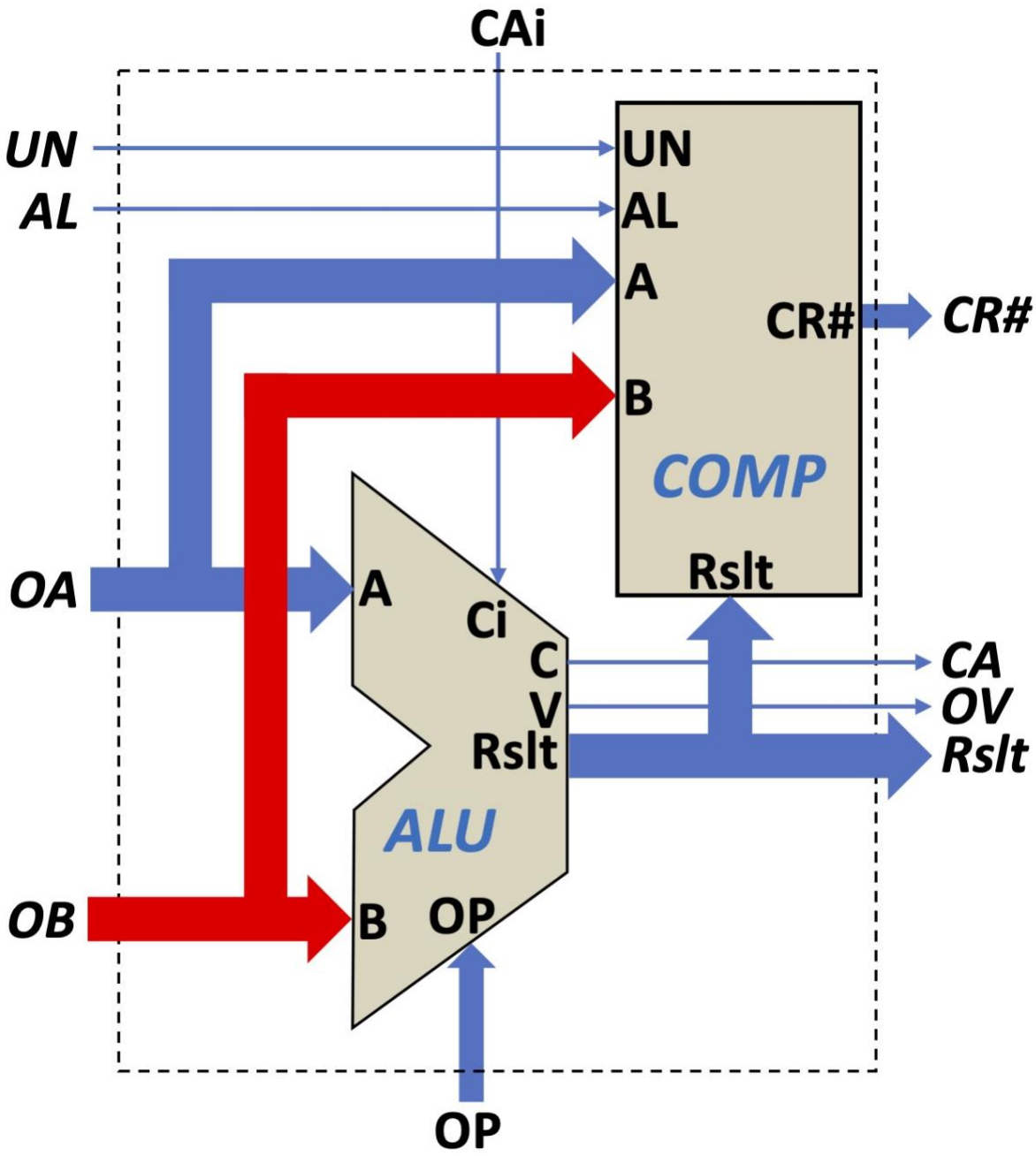

Figura 1. Diagrama de bloque del Circuito del Arithmetic Logic Unit y el Comparador

2.1.2 Descripción de señales

OA - Operando A (32 bits)

OB - Operando B (32 bits)

OP - Código que indica la operación que debe realizar el **ALU** (4 bits)

UN - Señal que indica si la comparación es de números sin signo (***UN***=1) o con signo (***UN***=0) (1 bit)

AL - Señal que indica si la comparación se hace para instrucciones aritméticas o lógicas (***AL***=1) o no (***AL***=0) (1 bit)***Rslt*** - Resultado de operación del **ALU** (32 bits)

CAi - Carry de entrada al **ALU** (1 bit)

Rslt - Resultado de la operación que realiza el **ALU** (32 bits).

CR# - Resultado de la comparación que realiza el comparado **COMP** (3 bits).

CA - Carry bit generado por el **ALU** (1 bit)

OV - Overflow bit generado por el **ALU** (1 bit)

2.1.3 Operación del ALU

El **ALU** toma dos números de 32 bits (**A** y **B**) y un bit de carry (**Ci**) y realiza operaciones con los mismos, determinadas por la señal **OP.** El resultado de las operaciones es un número (**Rslt)** y tres banderas de condiciones (**N**, **C** y **V**). En la Tabla 1 se muestran las operaciones que realiza el ALU para cada valor de **OP**.

Tabla 1. Operaciones del *ALU* y Banderas Generadas

OP	Operación	Banderas
0000	**shift left logical** (**A**) **B**[4:0] posiciones	
0001	**shift right logical** (**A**) **B**[4:0] posiciones	
0010	**shift right arithmetic** (**A**) **B**[5:0] posiciones	
0011	**A**+**B**	**NCV**
0100	**A**+**B**+**Ci**	**NCV**
0101	**B**-**A**	**NCV**
0110	**B**-**A**+**Ci**	**NCV**
0111	**A** AND **B**	
1000	**A** OR **B**	
1001	**A** XOR **B**	
1010	**A** NOR **B**	
1011	**A** NAND **B**	
1100	**A**	
1101	**B**	
1110	Not used	
1111	Not used	

Las banderas se generan solo para los valores de OP operaciones del 3 al 6. El valor de las banderas se genera de la siguiente manera:

- El bit **N** representa el signo del resultado de la operación (**N** = **Rslt**[31]).
- **C** representa el bit de ”overflow” de operaciones de suma o resta de números sin signo. Usualmente se le denomina como “carry” para suma y como “borrow” para resta. Para la suma de dos números de n bits **C** es el bit n+1 del resultado de la suma. Para la resta de dos números (**B** - **A**) **C** será igual a 1 si **B** < **A**.
- **V** representa el bit de “overflow” de operaciones de suma o resta de números con signo. **V** es igual a 1 cuando el signo del resultado de la suma o la resta no es consistente con las reglas de asignación de signo de operaciones aritméticas de números con signo, de lo contrario es igual a cero. **V** se puede determinar mediante una ecuación booleana de los signos de **A**, **B** y **Rslt**.

  Para **A** + **B**:

  V = ˜(**A**[31] ^ **B**[31]) & (**B**[31] ^ **Out**[31]).

  Para **B** - **A**:

  V = (**A**[31] ^ **B**[31]) & (**B**[31] ^ **Out**[31]).

2.1.4 Operación de COMP

El comparador ***COMP*** tiene la función de generar los tres bits más significativos que deben guardarse en un campo del registro **CR** cuando se ejecutan instrucciones aritméticas, lógicas o de comparación. Los tres bits se generan a partir de una comparación que envuelve las señales de entrada **A**, **B** y **Rslt.** El resultado de la comparación se hace disponible en la salida **CR#.**

La comparación está condicionada por las señales **AL** y **UN** según se muestra en el pseudocódigo que sigue:

```
If AL then
        If ((signed(A)< 0) then
                        CR#=0b100
                else if  ((signed(A) > 0) then
                                CR#=0b010
                        else
                                CR#=0b001
else    if UN then
                OA = A
                OB = B
                If (OA <OB)  then CR#=0b100
                        else if  (OA  > OB) then CR#=0b010
                                else CR#=0b001
        else
                OA = signed(A)
                OB = signed(B)
                If (OA < OB)  then
                                CR#=0b100
```

```
else if  (OA  = OB) then
                    CR#=0b010
          else
                    CR#=0b001
```

2.1.5 Demostración:

Deben asignar a las entradas **OA** y **OB** los valores que se indican a continuación:

OA = 10011100000000000000000000111000
OB = 01110000000000000000000000000011

Entonces, beben asignar la secuencia de valores de las señales de entrada que se indica en la Tabla 2.

Con una instrucción de monitor deben imprimir, en una sola línea, el valor de las siguientes señales:

OP en binario
Rslt en decimal y binario
CR# en binario.
CA y **OV**

Tabla 2. Secuencia de Valores de las Señales de Entrada

OP	CAi	AL	UN
0000	0	0	0
0001	0	0	0
0010	0	0	0
0011	0	0	0
0100	0	0	1
0101	0	0	1
0110	0	0	1
0111	0	0	1
1000	0	1	0
1001	0	1	0
1010	0	1	0
1011	0	1	0
1100	0	1	1
1101	0	1	1
0011	1	1	1
0100	1	1	1
0101	1	1	1
0110	1	1	1

2.2 Register File y Register Handler

2.2.1 Tarea

Implementar el circuito que se muestra a continuación.

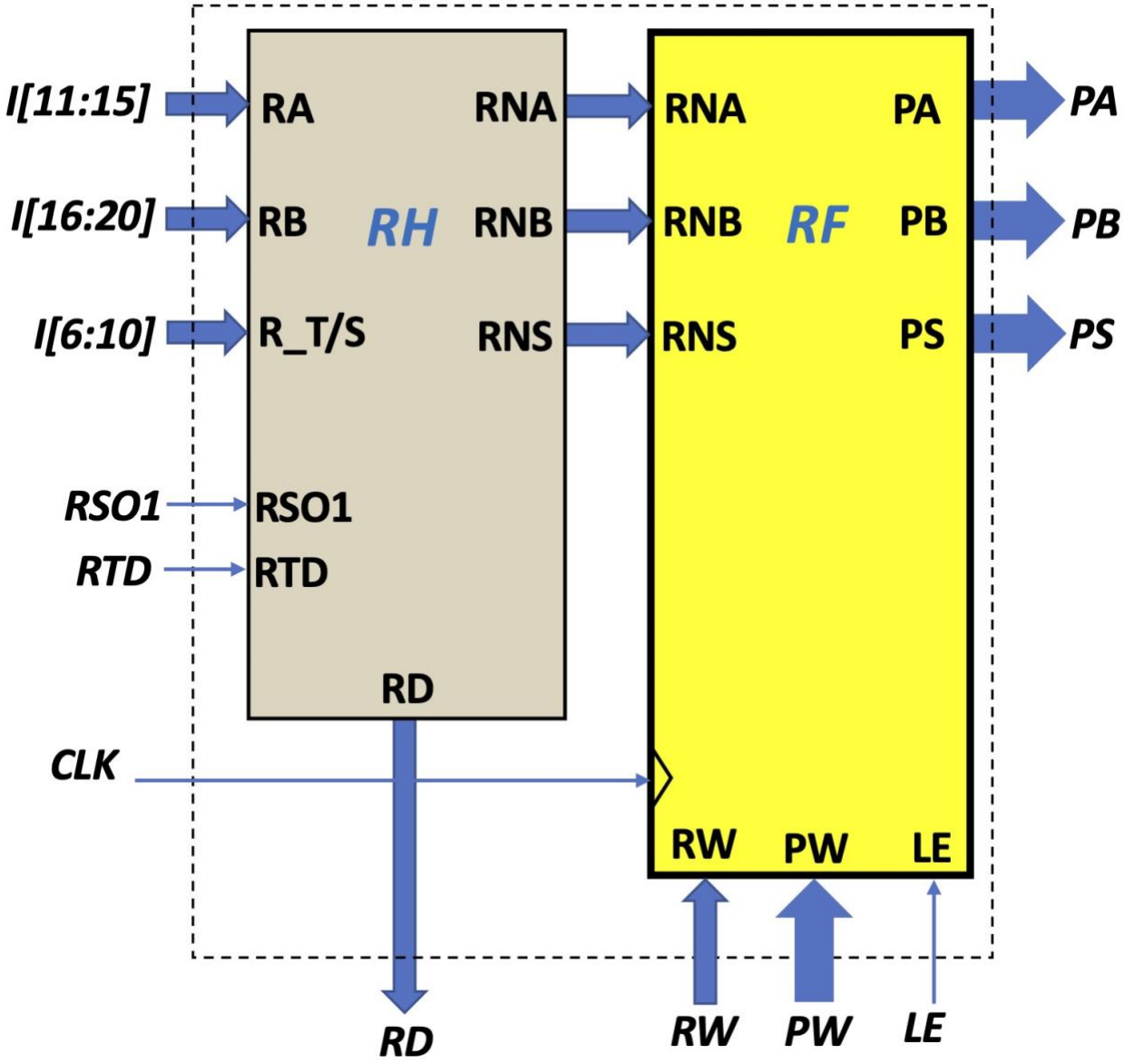

Figura 2. Diagrama de Bloque del Circuito del Register File y el Register Handler

2.2.2 Descripción de Señales:

I[11:15]- Número del registro especificado por el campo **RA** de una instrucción (5 bits)

I[16:20] Número del registro especificado por el campo **RB** de una instrucción (5 bits)

I[6:10]- Número del registro especificado por el campo **RT** o **RS** de una instrucción (5 bits)

RW- Número del registro donde se desea almacenar un operando (5 bits)

RD- Número del registro destino (donde eventualmente se almacena el operando destino de la instrucción (5 bits)

PA- Puerto de salida del primer operando fuente (32 bits)

PB- Puerto de salida del segundo operando fuente (32 bits)

PS- Puerto de salida del tercer operando fuente (32 bits)

PW- Operando que se desea almacenar en un registro (32 bits)

LE - Señal habilitadora de la escritura de un operando en un registro (1 bit)

RSO1 - Señal que indica que el campo **RS** de la instrucción especifica el registro fuente 1 (1 bit)

RTD - Señal que indica que el campo **RT** de la instrucción especifica el registro destino (1 bit)

CLK - Señal reloj del sistema (1 bit)

2.2.3 Register File (RF)

Las arquitecturas RISC, como POWER, tienen un gran número de registros de uso general que sirven para mantener operandos que se envuelven en alguna operación de un gran número de instrucciones. Esos registros también se utilizan para guardar el resultado de la operación de algunas instrucciones. En el circuito de un procesador, los registros son mantenidos en lo que se conoce como un register file. Este componente tiene señales que permiten acceder al contenido de los registros y guardar valores en los mismo.

2.2.3.1 Especificaciones

El Register File que se debe implementar contiene 32 registros de 32 bits cada uno. El mismo permite la lectura del contenido de un registro por tres puertos de salida (**PA**, **PB** y **PS**) y permite escribir un valor en un registro a través de un puerto de entrada (**PW**). Para implementar este componente pueden utilizar como modelo el diagrama del Register File de dos puertos de salida que se muestra en la Figura 3 de la página que sigue, teniendo en cuenta que el Register File que debe implementar (el de la Figura 2) es de 32 registros en vez de 16 y de tres puertos de salida en vez de dos.

El Register File se debe implementar interconectando tres tipos de componentes: binary decoder, multiplexer y registro. La operación fundamental de los primeros dos componentes se describe en la sección ***Logic Boxes*** de la lección ***Digital Circuits Fundamentals,*** mientras que la de los

registros se describe en la sección ***Edge-Triggered Registers*** de la misma lección. En los slides 67 al 69 de la lección ***Velrilog Tutorial*** se muestra código que describe cómo implementar registros sincrónicos con clear sincrónico. Sin embargo, los registros que deben implementar no deben tener una señal de clear porque no se presume que tengan un valor particular antes de ser cargados por primera vez. Entonces, pueden remover la señal **Clr** del código del registro. Para cada componente del Register File se debe crear un módulo en Verilog e instanciarlo las veces que sean necesarias (**esto es un requisito**).

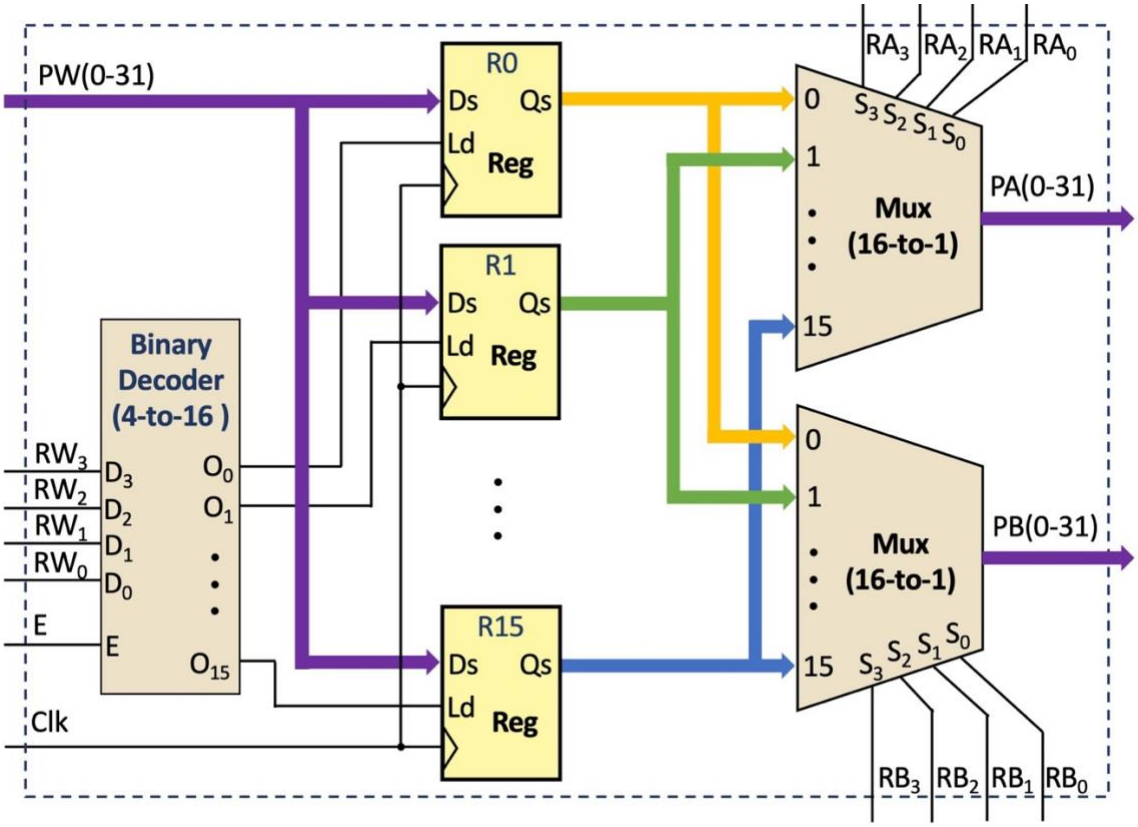

Figura 3. Register File de Dos Puertos

2.2.3.2 Operación del Register File

Lectura de registros:

El contenido de un registro se puede hacer disponible en un puerto de salida **Pi** especificando el número del registro en la correspondiente entrada **RNi**. Esta acción ocurre prácticamente de manera instantánea pues es una acción asincrónica.

Escritura en un registro:

Para poder escribir un registro se debe poner el valor que se desea escribir en la entrada **PW**, indicar el número del registro en la entrada **RW** y asignar un valor de 1 en la entrada **LE**. Entonces, el valor se escribe cuando ocurra la transición de 0 a 1 de la señal **CLK** (esta es una operación sincrónica).

2.2.4 Register Handler (RH)

En una unidad de ejecución pipelined, las instrucciones se traen de memoria en la etapa de fetch y se ubican en un registro en la etapa de decodificación. Las instrucciones de algunas arquitecturas pueden envolver de cero a tres registros fuentes (que almacenan operandos fuentes) y podría envolver uno o ningún registro destino (donde se almacena el operando destino de la instrucción). Por su función, los operandos contenidos en registros fuente se pueden categorizar como: operando fuente 1, operando fuente 2 o operando fuente 3.

El contenido de los registros fuente se puede acceder a través de los puertos de salida del register file según la categoría del operando que contienen. A continuación se indica qué operando fuente se puede se puede acceder por cuál puerto:

PA – operando fuente 1
PB – operando fuente 2
PS – operando fuente 3

Como se indicó en la descripción del Register File, para acceder al operando de un registro fuente solo se necesita indicarle al puerto el número del registro fuente por la entrada **RNi** que le corresponda. El número de cada registro fuente de una instrucción está especificado por un campo de la misma. Ese valor se debe hacer llegar a entrada **RNi** correspondiente, tomando en cuenta la categoría del operando fuente.

Para la arquitectura POWER, los campos que especifican el número del registro que contiene un operando fuente 2 y un operando fuente 3 se pueden conectar directamente a **RNB** y **RNS** respectivamente. Sin embargo, el campo que especifica el número del registro que contiene un operando fuente 1 no se puede conectar directamente a **RNA** porque la arquitectura POWER tiene las siguientes peculiaridades:

Primero: Los campos designados como **RT** y **RS** son especificados por los mismos bits de una instrucción (bits **I**[6:10]). Sin embargo, cuando ese campo se denomina como **RT,** este especifica el número del registro destino. Por otro lado, cuando ese campo se denomina **RS,** para algunas instrucciones ese campo especifica el número del registro que contiene el operando fuente 1 y para otras (como los stores) especifica el número del registro que contiene el operando fuente 3.

Segundo: El campo denominado como **RA** (bits **I**[11:15]) puede especificar un registro fuente o un registro destino. Para algunas instrucciones **RA** especifica el registro que contiene el operando fuente 1, pero para otras, especifica el número del registro destino.

Lo anterior tiene las siguientes consecuencias:

1. Los campos que especifican el número del registro que contiene un operando fuente 2 (**I**[16:20] y un operando fuente 3 (**I**[6:10]) se deben conectar directamente a las señales **RNB** y **RNS** respectivamente.

2. Cuando el número del registro fuente 1 es especificado por el campo **RS**, los bits de la instrucción que se deben conectar a **RNA** son **I**[6:10], de lo contrario, se deben conectar los bits **I**[11:15].

3. Cuando el número del registro destino es especificado por **RT**, los bits de la instrucción que se deben tomar como el registro destino (**RD**) son **I**[6:10], de lo contrario, se deben tomar los bits **I**[11:15] como el registro destino.

2.2.4.1 Operación del Register Handler

Las tres consecuencias antes mencionadas describen la operación del Register Handler.

2.2.5 Demostración

Para demostrar la operación del circuito de la Figura 1, se debe crear una señal **CLK**. Esta señal se debe inicializar a 0 en tiempo cero y luego debe cambiar de estado cada dos unidades de tiempo de manera perpetua. Entonces, la simulación debe seguir la siguiente secuencia de pasos:

1. A tiempo cero asignar los siguientes valores a las señales: **LE**=1, **PW**=30, **RW**,=0, **I[11:15]**=0, **I[16:20]**=31, **I[6:10]**=30, **RSO1**=0, **RTD**=1.

2. Incrementar por uno, cada cuatro unidades de tiempo, los valores de **PW**, **RW**, **I[11:15]**, **I[16:20]** y **I**[6:10] hasta que **I[11:15]** alcance el valor de 31.

3. Sin alterar los valores ya cargados en los registros, asignar los siguientes valores a las señales: **LE**=0, **PW**=55, **RW**,=0, **I[11:15]**=0, **I[16:20]**=31, **I[6:10]**=30, **RSO1**=1, **RTD**=0 y repetir el paso 2 pero hasta que **I[11:15]** alcance el valor de 4.

Utilizando una instrucción de monitor en el módulo de prueba de Verilog deben imprimir en una línea, en decimal, los valores de **RW**, **I[11:15]**, **I[16:20]**, **I[6:10]**, **PW**, **PA, PB, PS** y **RD**.

Como los registros no son precargados, cuando se lean sin haber sido cargados por primera vez su valor debe aparecer con x en todos sus bits. Esto no significa que tiene un error, lo que significa es que Verilog reconoce que esos registros no han sido cargados por primera vez y por lo tanto, su valor es indeterminado.

2.3 Program Counter y Generador de Target Address

2.3.1 Tarea

Implementar el circuito que se muestra a continuación.

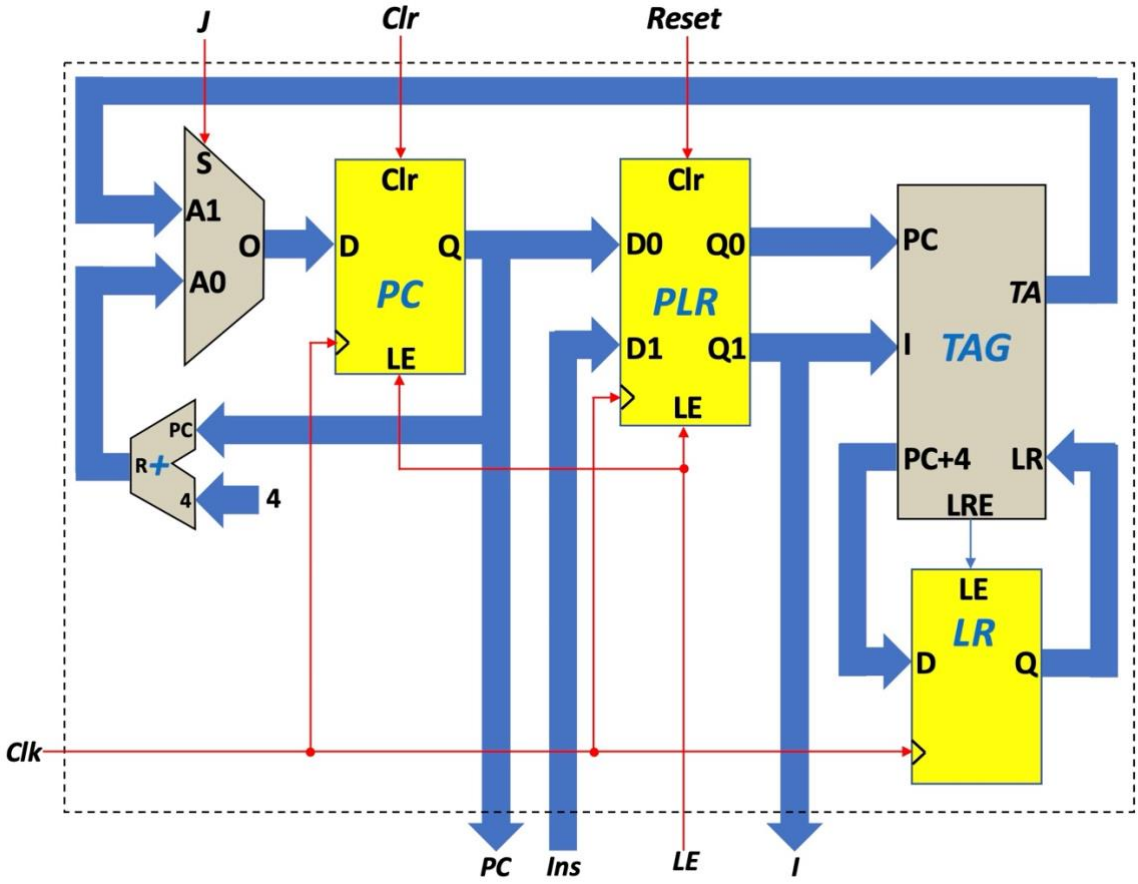

Figura 4. Diagrama de Bloque del Circuito del Program Counter y Generador de Target Address

2.3.2 Descripción de Señales:

PC- Valor del registro ***PC*** (localización de la instrucción a la que se le hace fetch) (32 bits)

I- Instrucción en la salida del Pipe Line Register (***PLR***) (32 bits)

LE - Señal habilitadora para cargar los registros **PC** y **PLR** (1 bit)

Reset – Señal de reset del sistema 1 (1 bit)

CLr - Señal de clear para el registro ***PC*** (1 bit)

J – Indica si la condición de un brinco se da (***J***=1) o no (***J***=0) (1 bit)

CLK - Señal reloj del sistema (1 bit)

2.3.3 Los Registros

Para el circuito deben implementar tres registros: Program Counter (**PC**), Pipeline Register (**PLR**) y Link Register (**LR**). ***PC*** y ***LR*** son registros de 32 bits, mientras ***PLR*** es de 64 bits. Los tres registros deben operar de manera sincrónica (se pueden cargan solo cuando ***LE*** es 1 y ocurre la transición de 0 a 1 de la señal ***CLK***). **PC** y **PLR** tienen una señal de clear (**Clr**) que actúa de manera sincrónica (si la señal es igual a 1 y ocurre la transición de 0 a 1 de la señal ***CLK*** sus bits de salida se hacen igual a cero). La operación de registros se describe en la sección ***Edge-Triggered Registers*** de la misma lección. En los slides 67 al 69 de la lección ***Velrilog Tutorial*** se muestra código que describe cómo implementar registros sincrónicos con clear sincrónico.

Los componentes del circuito que están a la izquierda del registro ***PLR*** corresponden a la etapa de fetch de un procesador pipelined, mientras que los de la derecha corresponden a la etapa de decodificación. Por ahora esto es una cuestión semántica. En las siguientes etapas del proyecto les será evidente.

Los registros tienen la siguiente función:

PC - Mantener la localización de la instrucción que se debe ejecutar (a la que se hace el fetch).
PLR – Transferir a la etapa de decodificación el contenido del ***PC*** y la instrucción que se trae de memoria como resultado del fetch (***Ins***).
LR – Cargar y mantener el valor del Link Register.

2.3.4 El Sumador y el Multiplexer

El circuito designado como ***+*** produce como resultado el valor de ***PC*** +4. La función del mux es seleccionar la dirección de la próxima instrucción que se debe ejecutar. Cuando la señal ***S*** es 0 se selecciona la salida del sumador (***PC***+4) y cuando es 1 se selecciona el Target Address que genera el circuito ***TAG***.

2.3.4 El Target Address Generator (TAG)

El Target Address Generator es un circuito combinatorial que tiene la función de generar el target address de una instrucción de brinco y controlar el Link Register. El circuito tiene las siguientes señales:

PC - Localización de la instrucción de brinco cuando se le hizo el fetch

I - Instrucción transferida desde la etapa del fetch

LR - El address almacenado en el Link Register

PC+4 **-** El valor de la señal ***PC*** + 4

TA **-** El target address que genera el circuito

LRE **-** Señal que habilita la carga del Link Register

2.3.4.1 Operación del Target Address Generator

La operación del ***TAG*** es determinada por la instrucción que se le hace llegar por su entrada ***I***. La instrucción tiene bits para identificar si una instrucción es una instrucción de brinco y si es de brinco, cómo atenderla. En los slides 62, 63 y 64 se describe la operación de las instrucciones de brinco. Las señales de salida del circuito deben generar lo siguiente:

TA - El target address según se especifica en los slides 62, 63 y 64. Cuando la instrucción no es de brinco se le debe asignar un valor de 0 a ***TA***. Para generar el address deben presumir que la condición se da.

PC+4 - Siempre tendrá el valor de la señal ***PC*** + 4.

LRE - Debe ser 1 si la instrucción es una instrucción de brinco que debe guardar un address de retorno en el Link Register, de lo contrario debe ser igual a 0.

2.5.5 Demostración

Para demostrar la operación del circuito de la Figura 1, se debe crear una señal **CLK**. Esta señal se debe inicializar a 0 en tiempo cero y luego debe cambiar de estado cada dos unidades de tiempo de manera perpetua. Entonces, se debe asignar a las señales de entrada los valores que se indican en la Tabla 3 en el tiempo indicado en la tabla.

Tabla 3. Secuencia de Valores de las Señales de Entrada

Tiempo	Ins	Reset	Clr	LE	J
0	00000111000000000000000000000000	1	1	1	0
4	01001000000000000000000000011001	0	0	1	0
8	01001000000000000000000000001110	0	0	1	1
12	00000111000000000000000000000000	0	0	1	1
16	01001111111111111111100000100001	0	0	1	0
20	00000111000000000000000000000000	0	0	1	1
24	01000011111111111111111111111110	0	0	1	0
28	00000111000000000000000000000000	0	0	1	1
32	01001111111111111111100000100000	0	0	1	0
36	00000111000000000000000000000000	0	0	1	1
40	01001000000000000000000000001110	1	0	1	0
44	00000111000000000000000000000000	0	0	1	1
48	00000111000000000000000000000000	0	0	0	0
52	01001000000000000000000000001110	0	0	1	0
56	00000111000000000000000000000000	0	0	1	0
60	00000111000000000000000000000000	0	0	1	0

Utilizando una instrucción de monitor en el módulo de prueba de Verilog deben imprimir en una línea, los valores de **PC** (en decimal), **Ins** (en binario), **Reset**, **Clr**, **LE** y **J**.

La simulación debe terminar en tiempo 63.

Como los registros no son precargados, cuando se lean sin haber sido cargados por primera vez su valor debe aparecer con x en todos sus bits. Esto no significa que tiene un error, lo que significa es que Verilog reconoce que esos registros no han sido cargados por primera vez y por lo tanto, su valor es indeterminado.

Para esta simulación les puede ser útil manejar la asignación de valores en diferentes tiempos utilizando el fork-join statement de Verilog que se describe en el slide 64 de la lección Verilog Tutorial.