Fase II: Diagrama de Bloque del PPU

Tarea:

En esta fase deben completar un diagrama de bloque circuito del PPU que se implementará para apoyar un sub conjunto de instrucciones de la arquitectura SPARC. Pueden utilizar como modelo el PPU descrito en la lección "Pipelined Processing Unit" que fue desarrollado para implementar un sub conjunto de instrucciones de la arquitectura ARM. Con algunas modificaciones ese PPU puede también implementar un sub conjunto de instrucciones de la arquitectura SPARC. Sin embargo, hay dos asuntos que deben tomarse en cuenta y atenderse.

1. Como se indica en la lección "Pipelined Processing Unit" hay detalles que han sido omitidos porque no eran relevantes para el entendimiento de su operación.

2. Necesita apoyar funciones específicas de SPARC.

Detalles ocultos:

1. Las instrucciones stores representan un caso especial ya que su registro destino es realmente utilizado como un tercer operando fuente que contiene el valor que se quiere guardar en memoria. Por lo tanto, se requiere un register file con tres puertos de salida como el especificado en la Fase I.

2. Se puede prescindir del mux en la etapa WB ya que el de la etapa MEM hace el mismo trabajo de seleccionar la fuente del operando que se escribirá en el register file en la etapa WB. Entonces, enviando la salida de este mux a la etapa WB hace el trabajo.

3. El logic box Shifter/Sign Extender debe ser sustituido por el **Source Operand2 Handler** descrito en la Fase I del proyecto. Se puede prescindir del mux de la etapa EX conectando la salida del **Source Operand2 Handler** a la entrada **B** del **ALU**.

4. La detección de la condición de brinco para una instrucción de brinco condicional parte de la suposición de que cuando la instrucción de brinco está en la etapa ID la instrucción que se encuentra en la etapa EX está autorizada a alterar los condition flags. Como es una suposición se puede dar el caso de que la instrucción en la etapa EX no tenga autorización para alterar los condition flags. Entonces, los condition flags que genera el ALU para esa instrucción no son los que se deben utilizar para la detección de la condición del brinco. Para atender este caso lo que necesitan saber es ¿dónde están los condition flags que se deben utilizar para detectar la condición del brinco?

5. El **ALU** genera conditon flags, no los almacena.

Asuntos específicos de SPARC:

1. SPARC utiliza dos registros, **nPC** y **PC**, para manejar "delayed branches". Bajo condiciones normales de ejecución de instrucciones estos dos registros se actualizan al mismo tiempo de la siguiente manera:

   PC <= **nPC**, **nPC** <= n**PC** + 4

2. Cuando una instrucción **call** o una instrucción **b** que brinca llega a la etapa ID, los registros **nPC** y **PC** se deben actualizar al mismo tiempo de la siguiente manera:

   PC <= **Brach_Target_Address**, **nPC** <= **Brach_Target_Address** +4

   El **Brach_Target_Address** se calcula cuando la instrucción **call** o **b** están en la etapa ID. Por lo tanto, cuando esas instrucciones están en la etapa IF es necesario enviar el valor del **PC** a la etapa ID en vez de **PC** + 4 como era el caso del PPU para la arquitectura ARM.

3. Cuando una instrucción de brinco condicional se encuentra en la etapa ID y se determina que la condición no se da, si el bit **a** de la instrucción (bit 29) es igual a 1, la instrucción en la etapa IF se debe cancelar activando un reset para el **Pipeline Register IF/ID**.

4. Cuando se ejecuta un brinco con una instrucción **jmpl** se genera un control Hazard cuando la instrucción llega a la etapa EX. Por lo tanto, la instrucción en la etapa IF debe ser cancelada aplicando un reset al **Pipeline Register IF/ID**. Entonces, **nPC** y **PC** se actualizan al mismo tiempo de la siguiente manera:

   PC <= **ALU_out**, **nPC** <= **ALU_out** + 4

5. La ejecución de la instrucción **jmpl** requiere guardar en su registro destino la localización de memoria en que se encuentra esa instrucción (el valor del **PC** cuando la instrucción se encuentra en la etapa IF). Por lo tanto, el valor del **PC** se debe propagar a las siguientes etapas hasta llegar a la etapa MEM donde se debe convertir en el operando destino que debe enviarse a la etapa WB.

6. Al igual que para jmpl, la ejecución de la instrucción **call** requiere guardar la localización de memoria de esa instrucción (el valor del **PC**) en un registro por lo que el **PC** se debe propagar a las siguientes etapas hasta llegar a la etapa MEM donde se debe convertir en el operando destino que debe enviarse a la etapa WB. A diferencia de jmpl, el registro donde se debe guardar el PC es específicamente el registro R15. Para todos fines práctico **call** tiene un registro destino, aunque ese registro no se especifica en la instrucción porque es implícito. Para atender esta situación, cuando se detecte una instrucción **call** en la etapa de decodificación, a los bits que se envían a la etapa EX, para identificar el número del registro destino, se les debe asignar un valor de 15.

7. La memoria de data debe manejar lectura y escritura de bytes, halfwords y words. En el caso de la lectura de bytes y halfword la misma se puede hacer con signo o sin signo. Esto requiere la conexión de varias señales que no son mostradas en el PPU de ARM pero que si están contempladas en la descripción de la memoria de data en la Fase I.

Guía para representación de diagramas de bloque:

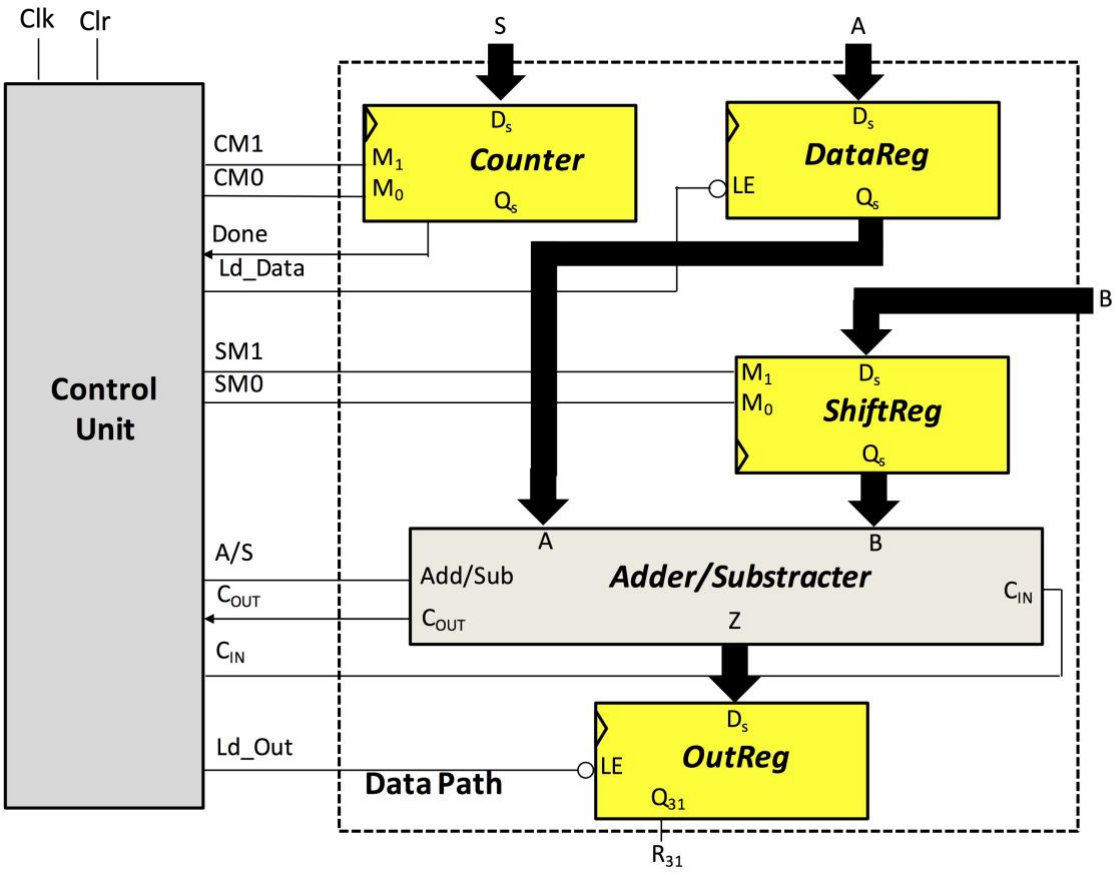
Los diferentes componentes se representan como cajitas rectangulares identificadas con un keyword y con las señales de entrada y salida apropiadamente identificadas dentro de la cajita. Las señales que se indican dentro de la cajita no tienen que tener nombres únicos. Por ejemplo, un circuito puede tener más de un componente con una señal nombrada **D**. Por el contrario, las señales externas que se utilizan para interconectar cajitas si tienen que tener nombres únicos. Si una señal de interconexión nombrada **Z** aparece en diferentes secciones de interconexión esta representa la misma señal y debe tener el mismo valor donde quiera que esté conectada. Las señales externas que se conectan a una cajita, pueden pero no tienen, que tener los mismos nombres internos de la cajita.

Las señales que consisten de mas de un bit se representan con un keyword. Por ejemplo **A** se puede utilizar para representar una señal de 32 bits. Sin embargo, si se quiere hacer referencia al byte menos significativo de **A** este se puede representar de las siguiente manera: $\mathbf{A}_{7:0}$ o **A**(7:0). De igual manera si se quiere hacer referencia al bit 27 de **A** se puede representar de la siguiente manera: $\mathbf{A}_{27}$. o **A**(27).

Los buses se representan como franjas o líneas gruesas con punta de flecha indicando dirección de la transmisión de la data.Las señales de un bit se representan como líneas finas con punta de flecha indicando dirección de la transmisión de la data.

Las señales de las cajitas que tienen aserción negativa (un cero las activa) se identifican con un pequeño círculo.

A continuación se muestra un ejemplo de un diagrama de bloque de un circuito:



Entrega:

Subir a NEO un documento en pdf del diagrama. El nombre del archivo debe seguir el siguiente formato: PF2 nombres de los integrantes del grupo separados con un espacio. (Ejemplo: PF2 Rosa Rene Pedro.pdf)

Rúbrica de evaluación:

- Se adjudicarán 3 puntos si el diagrama cumple con la guía para representación de diagramas de bloque que se describe en este documento.
- Se adjudicarán 4 puntos si el diagrama implementa un hardware altamente capaz de operar correctamente.
- Se adjudicarán 8 puntos si el diagrama atiende los *detalles ocultos* y los *detalles específicos de SPARC* que se describen en este documento.
- Se podrán asignar puntos parciales dependiendo del potencial de operación del PPU que el diagrama sometido evidencie.

La calificación de esta esta fase se adjudicará cuando se someta la versión final de los documentos requeridos en la Fase IV.