Pregunta 1

Sin responder aún

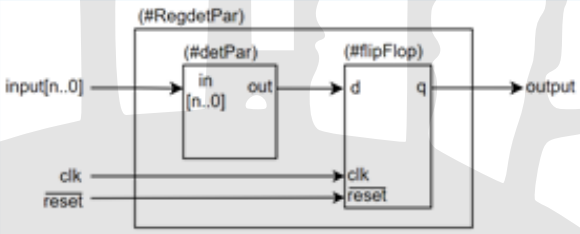
Puntúa como 1,00

Marcar pregunta

a) Diseñar en SystemVerilog un detector de paridad parametrizable (cantidad de bits de la entrada **in** = n, por defecto n=4) como el de la figura (#detPar). La salida **out** del circuito debe valer '1' cuando la cantidad de unos en el vector de entrada es par y '0' en caso contrario.

b) Diseñar en SystemVerilog un Flip-Flop D de 1 bit, con reset síncrono, activo por bajo, según el diagrama dado (#flipFlop). Entradas= **d, clk, reset**. Salida= **q**.

c) Diseñar en SystemVerilog un módulo (#RegdetPar) que registre la salida del detector de paridad en cada flanco de subida del clock. Entradas= **input, clk, reset**. Salida= **output**.



Para tener en cuenta:

- Respetar los nombres de los módulos, y puertos de entradas y salidas de forma LITERAL (respetando mayúsculas y minúsculas).
- Los archivos de SystemVerilog (.sv) deben tener el mismo nombre que los módulos.
- Las señales de entrada y salida deben ser del tipo logic y se deben declarar en el orden en que están listadas.
- El entregable de este ejercicio es el proyecto completo de Quartus, con los tres módulos desarrollados, en un archivo comprimido de nombre <Ej1.zip>

Información

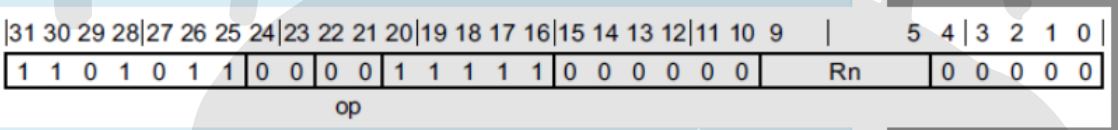
Marcar pregunta

Para que un procesador pueda ejecutar correctamente las tareas asociadas a un vector de excepciones real, es indispensable que el mismo sea capaz de cambiar completamente de contexto de ejecución. Eso incluye poder saltar a cualquier bloque de código contenida en memoria de programa. Es por esto que se propone en este ejercicio la modificación del procesador de un ciclo (el desarrollado en los **Prácticos 1 y 2 - SIN excepciones**) a fin que permita la ejecución de la instrucción **BR (Branch to Register).**

Los datos para la implementación de esta instrucción (OpCode, Tipo de instrucción, comportamiento, etc) deben obtenerse de la *Reference Data LEGv8* (GreenCard LEGv8) del libro "Computer Organization and Design - ARM Edition". Es necesario observar que esta instrucción posee un error de typo en la GreenCard, tal como se señala a continuación:

Branch to Register	BR	R	6B0	PC = R[Rt]

De esta forma, el campo de la instrucción utilizado para indicar el registro que contiene la dirección de salto es en realidad Rn, en lugar de Rt, como lo indica la tarjeta. De esta forma, el binario de la instrucción completa, queda conformado de la siguiente manera:



Se debe observar que la instrucción BR es del tipo R y pertenece al grupo de instrucciones de saltos INCONDICIONALES. Para la implementación se deben realizar todas las modificaciones que se consideren necesarias tanto en el bloque #datapath, como en #controller.

El entregable de este ejercicio es el proyecto en SystemVerilog del micro completo modificado (tomando como base el desarrollado en los prácticos 1 y 2), con TODOS sus módulos, los registros inicializados de 0 a 30 y el test bench del micro, que permita la ejecución de un programa en assembler que contenga la instrucción BR. El proyecto se entrega en un único archivo comprimido de nombre <Ej2.zip>

Información

Marcar pregunta

El objetivo de este ejercicio es poner en práctica los conocimientos desarrollados en el Práctico 3 sobre el comportamiento de las excepciones de nuestro procesador de un ciclo, mediante la escritura de un código en assembler que implemente uno de los procesos asociados al vector de excepciones, suponiendo que en nuestro procesador corre un Sistema Operativo (OS) multitarea. Este proceso debe, ante la ocurrencia de una excepción por OpCode invalido, determinar qué tarea de las que actualmente gestiona el OS generó la excepción, eliminarla del planificador (Scheduler) y continuar con la ejecución de la próxima tarea disponible de prioridad más alta. Para implementar este código se dispone de la ISA COMPLETA LEGv8, más las instrucciones que incorporamos en el Práctico 3 (ERET y MRS). Esto quiere decir que NO estamos limitados sólo a las instrucciones implementadas en el procesador de un ciclo de nuestros proyectos en SystemVerilog.

Como ocurre en la mayoría de los OS multitarea, este realiza la gestión de las tareas existentes a través de estructuras de datos residentes en memoria llamadas Bloques de Control de Tareas (TBC, Task Block Control). Cada vez que se crea una tarea en el OS, se crea un TBC asociada a la misma. De igual forma, cuando una tarea se elimina del planificador, su bloque TBC también se elimina.

En nuestro OS imaginario, cada TBC de 48 bytes está formado por una estructura de 6 elementos de 64 bits (8 bytes) c/u, tal como se detalla en la siguiente tabla:

Dirección	Campo TBC (tamaño)	Descripción
[Task n + 00h]	Task ID (8 bytes)	Número único que identifica cada tarea
[Task n + 08h]	Prioridad (8 bytes)	Número de prioridad asignado a cada tarea. Un valor de prioridad único por tarea. El número más bajo representa el valor de prioridad más alto.
[Task n + 10h]	Task Status (8 bytes)	Valores permitidos: 0: Tarea eliminada 1: Tarea no asignada al planificador 2: Tarea bloqueada 3: Tarea pausada 4: Tarea en ejecución 5: Tarea lista para ejecutar
[Task n + 18h]	Puntero Inicio (8 bytes)	Dirección de memoria que contiene la primer instrucción de la tarea
[Task n + 20h]	Puntero Final (8 bytes)	Dirección de memoria que contiene la última instrucción de la tarea
[Task n + 28h]	Puntero Link (8 bytes)	Dirección de retorno, que apunta a la siguiente instrucción de la tarea que debe ejecutarse una vez que la tarea pase a estado "en ejecución".
[Task n+1 + 00h]	...	...

La dirección de memoria de inicio del primer TBC (correspondiente a la Task 0) se encuentra en la dirección **0x20400**. Todos los TBC están alojados en forma contigua en memoria. Esto quiere decir que el próximo TBC se encuentra en la dirección 0x20430... 0x20460...etc. Un TBC con ID = 64'd0 indica el final de la lista de TBC's.

Implementacion del codigo:

Se debe implementar una sección de código assembler (su ubicación en memoria es indistinta) que realice el procesamiento equivalente al de una excepción por OpCode Invalido por parte de un OS. Dicho en otras palabras, esta sección de código forma parte del OS y su ejecución se produce cada vez que el procesador genera una excepción por instrucción inválida.

En forma resumida, el código a implementar debe realizar las siguientes acciones:

1- Haciendo uso de los registros especiales de excepción, determinar si la excepción a procesar es la de un OpCode invalido. Si fuera de cualquier otro tipo, no se realiza ninguna acción y se retorna del vector de excepciones.

2- Determinar qué tarea contiene el OpCode Invalido que causó la excepción. Para esto debe corroborarse no solo que la instrucción esté contenida en el rango de direcciones de la tarea, sino que además esta tarea se encuentre en estado "en ejecución".

3- Una vez identificada la tarea se debe cambiar el Status de la misma a "Tarea eliminada".

4- Determinar la próxima tarea a ejecutar, en base al valor de Status de las mismas (solo se tendrán en consideración aquellas tareas con Status igual a "Tarea lista para ejecutar") y su prioridad asignada.

5- Cambiar el status de la nueva tarea a "en ejecución".

6- Saltar a la dirección de retorno de la nueva tarea, contenido en campo "Puntero Link".

Con fines de claridad analicemos el siguiente ejemplo: Supongamos que la instrucción contenida en la posición 0x08348 posee un OpCode invalido, y al momento que el procesador intenta ejecutarla genera una excepción. En ese momento los bloques TBC estaban como se muestran a continuación (considerar organizacion *little endian*):

Dirección	Campo TBC	Contenido antes exc	Contenido después exc
0x20400	Task ID	00 D0 FF 00 00 00 00 00	00 D0 FF 00 00 00 00 00
0x20408	Prioridad	02 00 00 00 00 00 00 00	02 00 00 00 00 00 00 00
0x20410	Task Status	05 00 00 00 00 00 00 00	04 00 00 00 00 00 00 00
0x20418	Puntero Inicio	00 10 08 00 00 00 00 00	00 10 08 00 00 00 00 00
0x20420	Puntero Final	FF 2F 08 00 00 00 00 00	FF 2F 08 00 00 00 00 00
0x20428	Puntero Link	0F 1E 08 00 00 00 00 00	0F 1E 08 00 00 00 00 00
0x20430	Task ID	00 C0 CA 00 00 00 00 00	00 C0 CA 00 00 00 00 00
0x20438	Prioridad	05 00 00 00 00 00 00 00	05 00 00 00 00 00 00 00
0x20440	Task Status	04 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0x20448	Puntero Inicio	00 30 08 00 00 00 00 00	00 30 08 00 00 00 00 00
0x20450	Puntero Final	FF DF 08 00 00 00 00 00	FF DF 08 00 00 00 00 00
0x20458	Puntero Link	0F 3E 08 00 00 00 00 00	0F 3E 08 00 00 00 00 00
0x20460	Task ID	00 CA FE 00 00 00 00 00	00 CA FE 00 00 00 00 00
0x20468	Prioridad	03 00 00 00 00 00 00 00	03 00 00 00 00 00 00 00
0x20470	Task Status	05 00 00 00 00 00 00 00	05 00 00 00 00 00 00 00
0x20478	Puntero Inicio	00 00 09 00 00 00 00 00	00 00 09 00 00 00 00 00
0x20480	Puntero Final	FF 2F 09 00 00 00 00 00	FF 2F 09 00 00 00 00 00
0x20488	Puntero Link	0F 0E 09 00 00 00 00 00	0F 0E 09 00 00 00 00 00
0x20490	Task ID	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00

Una vez que se determina que se trata de una excepción por OpCode invalido, se determina que la instrucción corresponde a la tarea ID 0xCAC000, ya que la dirección que generó la excepción está dentro del rango entre **0x083000 y 0x08DFFF** y el Status es **0x4**. A continuación debe eliminarse esta tarea cambiando el Status por 0x00.

Una vez concluido esto, se debe determinar cuál será la próxima tarea a ejecutarse. En nuestro ejemplo, las otras dos tareas presentes se encuentran "listas para ejecutarse" porque tienen Status 0x5, sin embargo se selecciona la tarea ID 0xFFD000 ya que posee prioridad 0x02. Por ellos se debe cambiar su Status por el valor 0x4 ("en ejecución") y se debe retornar a la dirección 0x081E0F (valor contenido en Puntero Link).

El entregable de este ejercicio es un archivo Ej3.asm con el código generado, que puede escribirse a partir de los template usados en los Prácticos 2 o 3.

Pregunta 2

Sin responder aún

Puntúa como 1,00

Marcar pregunta

Dados los siguientes fragmentos de código de instrucciones LEGv8:

```
1>>         XOR X4, X4, X4
2>> L1:     LSL X9, X4, #3
3>>         ADD X9, X9, X6
4>>         LDUR X10, [X9, #0]
5>>         LDUR X11, [X9, #8]
6>>         SUB X9, X10, X11
7>>         ADDI X4, X4, #1
8>>         CBNZ X9, L1
9>>         SUBI X5, X4, #1
10>>        XOR X4, X4, X4
11>>        XOR X6, X6, X6
```

El segmento de memoria utilizado se encuentra inicializado de la siguiente forma:

Dirección	Contenido
[X6+00h]	C0 CA C0 1A CA FE CA FE
[X6+08h]	CA FE CA FE C0 CA C0 1A
[X6+10h]	CA FE CA FE C0 CA C0 1A
[X6+18h]	BE BA CA FE 60 0D BE E2

a) Analizar en el código las dependencias y completar la siguiente tabla:

Instrucciones involucradas (# instr. 1, #instr 2)	Operando en conflicto	Tipo de dependencia (Datos, control, estructural)

Aclaración: el número de líneas en la tabla no es un indicador de la cantidad de dependencias existentes

b) Mostrar el orden de ejecución de las instrucciones utilizando un procesador con *forwarding stall* suponiendo que el contenido de la memoria es el mostrado en la tabla contigua. Considerar que el procesador es *always not taken*. Notar que el contenido de la memoria se muestra con organización *little endian*.

c) Cuántos ciclos de clock toma la ejecución del código del punto anterior? Considerando un clock de 3GHz, calcular el tiempo de ejecución, considerando que el "pipeline está vacío".

La respuesta a esta pregunta debe enviarse en un único archivo en formato pdf con el nombre ej4.pdf, pueden ser fotos del ejercicio resuelto en papel (fotos de buena calidad y resuelto en forma prolija) o puede ser resuelto en computadora.