Proyecto de Ingeniería de Computadores (PEC)

Etapa 8: TLB (Translation Lookaside Buffer)

Finalmente, añadiremos soporte para la memoria virtual, añadiendo dos TLB (*Translation Lookaside Buffer*): uno para las direcciones de instrucciones y el otro para las direcciones de datos. Esto permitirá que el sistema operativo pueda desarrollar una tabla de páginas con soporte de hardware.

Tenemos que añadir el conjunto de instrucciones que operan con el TLB de datos y de instrucciones: **`WRPI`**, **`WRVI`**, **`WRPD`** y **`WRVD`**. Las instrucciones que terminan con una 'I' escriben en el TLB de instrucciones, mientras que las instrucciones terminadas con una 'D' lo hacen en el TLB de datos. Las que tienen una 'P' escriben en el TAG físico, en la posición indicada por el registro del operando A y las que tienen una 'V', escriben en el TAG virtual.

Por último, tendremos la instrucción **`FLUSH`**: Esta instrucción, en función del valor en Ra, vacía por completo el contenido de las caches (que nuestro procesador no tiene) de instrucciones y/o datos y/o elimina todas las traducciones existentes en el TLB de datos y/o instrucciones.

Para poder decodificar correctamente las instrucciones es necesario saber cómo se codifican. La siguiente tabla muestra la codificación de las instrucciones **`WRPI`**, **`WRVI`**, **`WRPD`**, **`WRVD`**, y **`FLUSH`**.

15 14 13 12	11 10 9	8 7 6	5 4 3	2 1 0		
1 1 1 1	Rb	Ra	1 1 0	1 0 0	**WR**ite **P**hisical address into **I**nstrucction TLB	`WRPI`
1 1 1 1	Rb	Ra	1 1 0	1 0 1	**WR**ite **V**irtual address into **I**nstruction TLB	`WRVI`
1 1 1 1	Rb	Ra	1 1 0	1 1 0	**WR**ite **P**hisical address into **D**ata TLB	`WRPD`
1 1 1 1	Rb	Ra	1 1 0	1 1 1	**WR**ite **V**irtual address into **D**ata TLB	`WRVD`
1 1 1 1	0 0 0	Ra	1 1 1	0 0 0	**FLUSH** caches and/or TLBs	`FLUSH`

Todas estas nuevas instrucciones son privilegiadas (o de sistema). Fijaos que todas las instrucciones privilegiadas (excepto **`IN`** y **`OUT`**) se encuentran englobadas dentro del código de operación 1111 y su ejecución debe provocar una excepción cuando son ejecutadas en modo usuario. Recordad también, que los 6 bits de menor peso del formato de la instrucción indican que instrucción concreta se debe ejecutando. Los códigos no usados quedan libres para futuras ampliaciones del lenguaje; la ejecución de una instrucción con estos códigos genera una excepción de "instrucción ilegal".

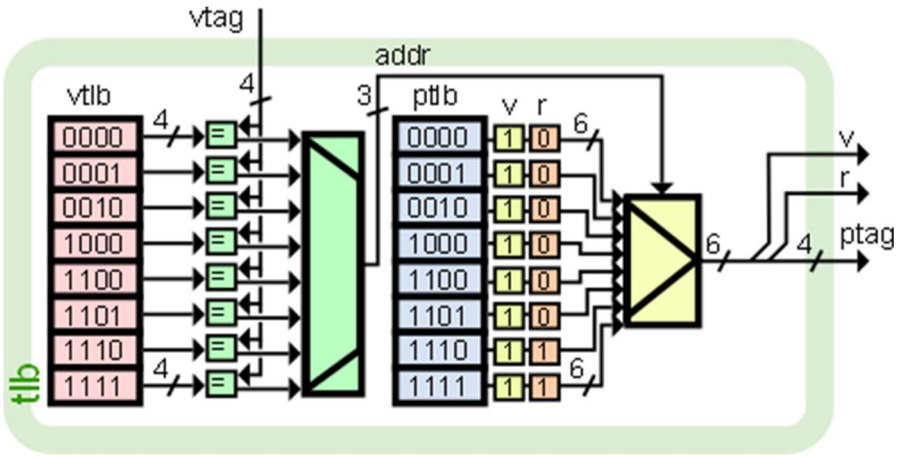
Para desarrollar un sistema operativo que soporte memoria virtual y paginación necesitamos apoyo por parte del hardware. Este debe tener una MMU capaz de traducir las direcciones lógicas a físicas de forma transparente. En este caso se podría colocar directamente la tabla de páginas directamente al procesador. Si hacemos un formato de página de 4 KB por ejemplo, al ser el espacio direccionable de 64 KB, entonces sólo necesitaríamos una tabla de páginas de 16 posiciones para acceder a todo el espacio direccionable y podríamos tenerla completa en el procesador. O podríamos organizar la memoria en páginas de 1024 bytes (1 KByte) empezando en la dirección 0 (como dice el manual del SISA-S), y tener un TLB de 64 entradas.

En este caso perderíamos la función que tiene el TLB. Como en un procesador comercial del espacio direccionable es mucho mayor, normalmente la tabla de páginas sería muy grande y sólo se puede tener una caché de la tabla de páginas en el procesador. Esto es el TLB.

Para hacerlo similar a un procesador comercial, también haremos que el TLB sea una caché de la tabla de páginas. Para hacer esto limitaremos el número de entradas en el TLB. Así se podrán provocar situaciones de miss en el TLB y esto generará una excepción que el sistema operativo deberá gestionar.

Por lo tanto, haremos un TLB de 8 entradas. Un TLB se implementa normalmente con una memoria CAM (*Content Addressable Memory*). Esta memoria compara el *tag* de la dirección lógica con todas las posiciones de los *tags* lógicos. Si se produce una coincidencia, entonces sale el correspondiente *tag* físico y la dirección completa se compone con el *tag* físico más los bits de menos peso de la dirección.

En la siguiente figura podemos ver un posible esquema funcional del TLB.



En este esquema se pueden observar de color rojo los *tags* virtuales, mientras que de color azul están representados los *tags* físicos. Estos es el esquema fundamental de una memoria direccionable por contenido (CAM). Se compara la entrada con todos los *tags* virtuales y la salida se codifica para sacar el *tag* físico correspondiente.

Además del *tag* físico (también llamado marco de página) tenemos dos bits adicionales **v** y **r** que nos indican si la página que estamos accediendo es válida o es de sólo lectura. Para una página normal **v** vale 1 mientras que **r** vale 0. En total salen 6 bits (4 por el *tag* físico y 2 por los bits adicionales).

Durante la lectura, se compara el *tag* virtual con todos los almacenados en el TLB para determinar que *tag* físico hay que devolver. Si durante esta comparación no hay ninguna entrada que coincida, entonces se produce un *miss* (fallo de TLB). Una vez tenemos el número de entrada, podemos indexar la tabla con los *tags* físicos, el bit de sólo lectura (r) y el de válido (v). Si durante la comparación aparecen más de una coincidencia escogeremos la de la entrada de menor peso (podríamos implementarlo con un codificador de prioridad).

Para modificar el contenido del TLB utilizaremos las nuevas instrucciones que hay que implementar. Cada instrucción escribe en el TLB que le corresponde (instrucciones o datos) y en la tabla adecuada (*tags* virtuales o físicos). Los 4 bits de menor peso del registro Ra indicarán que entrada de la tabla hay que modificar y los 4 bits de menor peso de Rb indican en nuevo valor.

En el caso de que la escritura sea en los *tags* físicos (**`WRPI`** o **`WRPD`**), además hay que escribir los bits de valido y de solo lectura. El valor de estos bits está localizado en los bits Rb<5> y Rb<4> respectivamente.

Esto obliga al sistema operativo a tener un control en todo momento de lo que está escrito en el TLB, ya que si se produce un fallo de TLB (miss), se lanza una excepción y el sistema operativo debe ser capaz de reemplazar la entrada que menos se use.

La inicialización del TLB es bastante simple pero muy importante. Los *tags* físicos y los *tags* virtuales deben coincidir al iniciar el procesador. Además, se reservan 3 páginas para usuarios y 5 más para sistema. Las páginas de usuario son las 3 primeras, direcciones de **`0x0000`** en la **`0x2FFF`**. Las 5 páginas restantes son de modo sistema. De éstas, ponemos una en la parte baja de RAM y las otras 4 en la parte del área de memoria no escribible (supuestamente implementada con una Flash). Así tendremos accesibles las direcciones **`0x8000`** hasta **`0x8FFF`** y también el área de sólo lectura **`0xC000`** a **`0xFFFF`**.

Esto lo hacemos así por la forma en que se inicializará el sistema operativo. Necesita tener accesible toda el área de flash para hacer la copia de la imagen de usuario en la parte baja de la RAM. Por tanto, no tendrá que tocar el TLB para acceder al área de flash.

Listado de las nuevas excepciones

Al añadir los TLB se pueden generar nuevas excepciones. Las nuevas excepciones que deberemos tratar serán las siguientes:

- Excepción 6: Miss en TLB de instrucciones. Se lanza esta excepción cuando se hace un fetch de una dirección, la página de la que no está al TLB y por tanto no puede traducirse en dirección física.
- Excepción 7: Miss en TLB de datos. Esta excepción se produce cuando se intenta hacer un load o store en una dirección, la página de la que no está al TLB de datos y no se puede traducir en dirección física.
- Excepción 8: Página inválida al TLB de instrucciones. Se produce cuando se intenta hacer un fetch de una dirección que está en una página con el bit de inválida al TLB.
- Excepción 9: Página inválida al TLB de datos. Igual que el anterior pero cuando se realiza en una página referenciada con un load o store.
- Excepción 10: Página protegida al TLB de instrucciones. Se hace un fetch en modo usuario de una página que pertenece al sistema operativo (tiene el bit de protección activado) y, por tanto, sólo él puede ejecutar código de esta página.
- Excepción 11: Página protegida al TLB de datos. De manera similar a la anterior, el programa de usuario está ejecutándose en modo no privilegiado e intenta leer (load) o escribir (store) en una página que pertenece al sistema operativo (tiene el bit de protección activado).
- Excepción 12: Página de sólo lectura. Esta excepción se lanza cuando una instrucción de store hace un acceso a una página que tiene el bit de sólo lectura activado el TLB de datos.

Cambios en el uso de los registros de sistema

Los registros de sistema que hay que modificar su uso añadiendo algo son los siguientes:

S1. Contiene la dirección de retorno después de una interrupción (externa), una excepción (interna) o la ejecución de la instrucción de llamada al sistema operativo (**`CALLS`**). Cuando se produce uno de estos eventos, se salva en S1 el PC actualizado, ya que acto seguido se copia en el PC el registro S5, que contiene la dirección del código de entrada al sistema operativo. En caso de excepción por fallo de TLB, el sistema operativo es responsable de decrementar S1 en 2 unidades, ya que se debe retornar a la misma instrucción que provoco el fallo.

S2. Contiene, en sus cuatro bits de menor peso, un código binario que puede tomar valores de 0 a 15 que identifica el tipo de evento que se ha producido: tipo concreto de excepción (interna), llamada a sistema (**`CALLS`**) o interrupción (externa). El resto de bits de S1 están siempre a 0. Por ahora los códigos usados son:

- 0: Excepción de tipo "Instrucción ilegal".
- 1: Excepción de tipo "Acceso a memoria mal alineado".
- 2: Excepción de tipo "Overflow en operación de coma flotante".
- 3: Excepción de tipo "División por cero en coma flotante".
- 4: Excepción de tipo "División por cero".
- 6: Excepción de tipo "Miss en TLB de instrucciones".
- 7: Excepción de tipo "Miss en TLB de datos".
- 8: Excepción de tipo "Página inválida al TLB de instrucciones".
- 9: Excepción de tipo "Página inválida al TLB de datos".
- 10: Excepción de tipo "Página protegida al TLB de instrucciones".
- 11: Excepción de tipo "Página protegida al TLB de datos".
- 12: Excepción de tipo "Página de sólo lectura".
- 13: Excepción de tipo "Instrucción protegida".

- 14: Llamada a sistema (**CALLS**)
- 15: Interrupción (externa)

S3. Contendrá una copia del registro que contiene el número de servicio de sistema operativo que se quiere ejecutar con la instrucción **CALLS**, o contiene la dirección de memoria que produjo el fallo de TLB, en caso de excepción por fallo de TLB.

S6. Contendrá el puntero a la pila de sistema, SSP, System Stack Pointer (cuando implementemos el S.O).

Tratamiento de las excepciones de TLB

Se trataran casi igual que el resto de excepciones.. Se usará el mismo estado extra (SYSTEM) que ya se ha implementado. Simplemente la secuencia acciones que debe realizar este estado es un poco distinta. Debe realizar alguna tarea nueva más.

Las principales novedades son:

- En se guarda en S1 la dirección el PC actualizado que es la dirección de retorno en todos los casos excepto en el de fallo de TLB en el que el sistema operativo es responsable de decrementar si en 2 unidades, ya que se debe retornar a la misma instrucción que provoco el fallo.
- Si se ha producido un fallo de TLB, se carga en S3 la dirección lógica (o virtual) de memoria que produjo el fallo.

Para que no se produzcan excepciones durante la ejecución de las primeras y últimas instrucciones de la rutina de entrada al sistema operativo, la página en la que se encuentra este código debe estar fija siempre en memoria y ocupar una entrada del TLB y lo mismo debe pasar con la pila de sistema y con todas las estructuras de datos que se acceden en este fragmento de código, como por ejemplo el vector de interrupciones.

Recordad que las direcciones lógicas que empiezan por 1 son de sistema y si no son de usuario. Así es como normalmente Linux también trabaja y para simplificar las cosas lo pondremos fijo.

Cambios en el módulo de ejecución (datapath)

Una vez visto el TLB, debemos explicar los cambios que hay que hacer en la parte de ejecución. Debemos añadir 2 TLB, uno para el acceso a memoria por parte del fetch y el otro como resultado de los loads y stores. Habitualmente el primero se llama TLB de instrucciones y el segundo TLB de datos.

Debemos tener en cuenta que antes la dirección que salía hacia la memoria simplemente era el PC o la salida de la ALU, dependiendo de si nos encontramos en el ciclo de fetch o bien en el ciclo de memoria. Ahora el bus de direcciones de memoria estará compuesto por los 12 bits de menor peso del PC y de la ALU, de la misma forma. Ahora, los 4 bits de más peso (los que forman el *tag*) pasarán por los TLB: los 4 bits de más peso del PC pasarán por TLB de instrucciones y los 4 bits de más peso de la ALU pasarán por TLB de datos.

Señales de control

Sólo tenemos que controlar el acceso de escritura al TLB ya que la lectura y traducción de *tags* se hace de manera automática por el TLB. Hay que generar nuevas señales que permitan realizar las 4 escrituras posibles que vienen dadas por las instrucciones **WRPI**, **WRVI**, **WRPD** y **WRVD** que corresponden respectivamente a escribir el *tag* físico de instrucciones, *tag* virtual de instrucciones, *tag* físico de datos y *tag* virtual de datos. Debemos tener presente que las señales que afectan al estado del procesador estarán sometidas al control del multiciclo.

Prueba de los TLB

Ahora ya tenemos el procesador listo para que se pueda implementar una tabla de páginas con memoria virtual. Haremos un pequeño programa que pruebe estas 4 instrucciones y como el procesador inicializa y hace correctamente la traducción de direcciones. El siguiente programa puede ayudarnos en esto.

```
 1   ; 0x3d = 11_1101 (v=1,r=1,marco=13)
 2   movi r1, 0x3d
 3   movi r2, 4
 4   wrpi r2, r1
 5   halt

 6  .org 0x1000
 7   and  r0, r0, r0
 8   and  r0, r0, r0
 9   and  r0, r0, r0

10   movi r3, 7
11   wrpi r3, r1   ; dejamos la pos 7 igual
12   movi r4, 12
13   movi r5, 15
14   wrvi r3, r4   ; itlb.v[7] = 12
15   wrvi r2, r5   ; itlb.v[6] = 15

16   movi r1, 3
17   addi r1, r1, -5
18   out 6, r1     ; -2 a los LEDs rojos

19   movi r0, 0
20   movi r4, 0x20 ; (v=1, r=0, marco=0)
21   movi r5, 0x3c ; (v=1, r=1, marco=12)
22   movi r1, 1
23   movi r2, 12

24   wrpd r1, r5
25   wrpd r0, r4
26   wrvd r1, r0
27   wrvd r0, r1

28   movi r3, 12
29   shl  r3, r1, r3
30   ld r2, 0(r0)  ; @log: 0000, @fis: c000
31   st 0(r3), r2  ; @log: 1000, @fis: 0000

32   halt
```

Las tres primeras instrucciones cambian la traducción de las direcciones lógicas **0xCXXX** de instrucciones. Las direcciones se traducen en la dirección física **0xDXXX** (marco 13, 0xd, válido y sólo lectura). En este momento, justo después de ejecutar el **WRPI** de la línea 4 ya estamos al marco siguiente. Por tanto, la instrucción **HALT** de la línea 5 no se ejecuta nunca.

Debido a que hemos puesto una directiva del ensamblador `.org 0x1000`, fuerza a que cuando se haga la copia de este código en la flash comience a copiar a partir de aquí en el origen absoluto + `0x1000`. Es decir, que si el programa siempre se copia a **0xC000**, el código que viene a continuación del `.org 0x1000` estará en la dirección **0xD000**. Por tanto, la instrucción **WRPI** de la línea 4 es como si hiciera saltar el código en el siguiente marco y ejecutar a continuación la instrucción movimiento de la línea 10.

Debemos tener en cuenta que las instrucciones de las líneas 7 a 9 no se ejecutan ya que el contador de programa al hacer el **WRPI** es de **0xC004** y después de ejecutar la instrucción, el PC traducido es de **0xD006**, que es justo la instrucción de la línea 10.

Ahora lo que haremos será cambiar las filas 7 y 8 del TLB de instrucciones. Para hacer esto, primero duplicaremos una entrada en el TLB. La línea 11 del programa deja igual la posición 7 del TLB porque en la inicialización ya toma este valor. A continuación, las líneas de la 12 a la 15 cambian el *tag* virtual de forma que ahora en la última posición vale 12 y la posición 6 vale 15.

Después de ejecutar la línea 14, hemos duplicado el *tag* virtual 12 y ahora está en la posición 6 y 7 del TLB. Como hemos puesto un codificador con prioridad, sólo la entrada 6 será la que se utilice. Pero después de ejecutar la instrucción de la

línea 15 desaparece el *tag* 12 de la posición 6. Ahora sólo se utiliza la posición 7, que apunta al mismo marco físico. Por tanto, el programa debe seguir ejecutándose sin problemas. Si esto es cierto al pasar por la línea 18 se encenderán todos los LED rojos menos el último.

Las siguientes pruebas se hacen sobre el TLB de datos, afectando a las instrucciones **`LD`** y **`ST`**. Las líneas 24 y 25 colocan un 0 en el *tag* físico de la posición 0 y un 12 en el *tag* físico de la posición 1 y las líneas 26 y 27 intercambian *tags* virtuales correspondientes a las posiciones 0 y 1.

Por lo tanto, hacer un acceso al contenido de las direcciones lógicas **`0x0XXX`** se traducirá en un acceso a la dirección física **`0xCXXX`** y el acceso a la dirección **`0x1XXX`** se traducirá en **`0x0XXX`**. Esto es lo que precisamente hacemos a las instrucciones de las líneas 30 y 31. Si todo va bien el **`LD`** leerá el word de código correspondiente al primer movimiento del programa (**`0x523D`**). A continuación, este dato se escribirá en la dirección física 0. ~~*(ojo con las excepciones, quizás haya que desactivar alguna para este juego de pruebas)*~~.

Juegos de pruebas

Realizad vosotros un juego de pruebas nuevo (uno o varios) que compruebe el correcto funcionamiento de las TLB y de todas las nuevas las excepciones que se acaban de añadir.

¿Qué pasa con la memoria de la VGA?

Si escribimos programas que usen el controlador de la VGA deberemos solucionar el "problema" de que la memoria de la VGA no ha sido inicializada en la TLB. Una solución sencilla seria alterar la inicialización que hace el procesador del TLB reservando un par de entradas para la memoria de la VGA. Otra solución, que es la que escogeremos, es que el procesador actualice el contenido de la TLB en las primeras fases de ejecución de un programa. Si escribís un programa en ensamblador, podéis usar al principio del programa las instrucciones de sistema para escribir los *tags* adecuados para la VGA en las entradas de la TLB que deseéis (sin eliminar ninguna de las estrictamente necesarias). Si el programa escrito es en C, podéis poner estas instrucciones en el fichero "entrada.s"