PAGINACION

Memoria virtual

La memoria virtual es una técnica de manejo de memoria que permite manejar múltiples procesos y que cada proceso pueda utilizar gran cantidad de memoria para sí. Esto requiere soporte tanto del sistema operativo como del procesador.

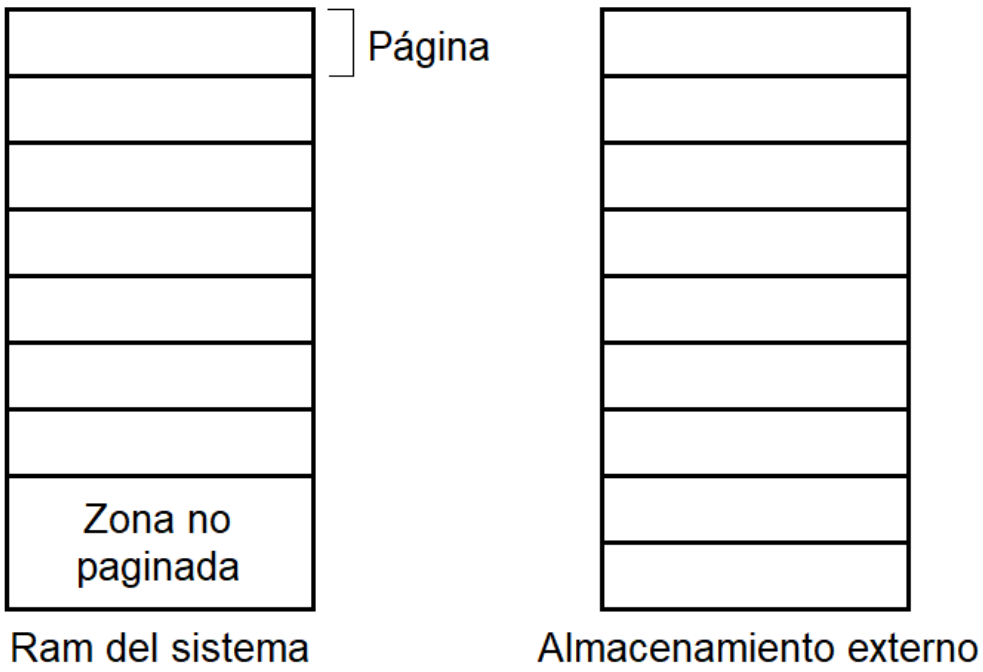
Para lograr esto, se subdivide la memoria direccionable en bloques del mismo tamaño, denominados “páginas”. En general estas páginas son de 4 KiB, pero es posible que el sistema soporte una cantidad limitada de tamaños de páginas (por ejemplo, 2 MiB, 4 MiB y 1 GiB para procesadores Intel y 64 KiB, 1 MiB y 16 MiB para procesadores ARM).

La ventaja de usar tamaños fijos de páginas es que, si el sistema se queda sin memoria RAM, es posible almacenar las páginas en un almacenamiento externo, sea una SSD o un disco rígido para recuperarlo más tarde. De esta manera se libera memoria RAM que puede servir para usarla para código o datos.

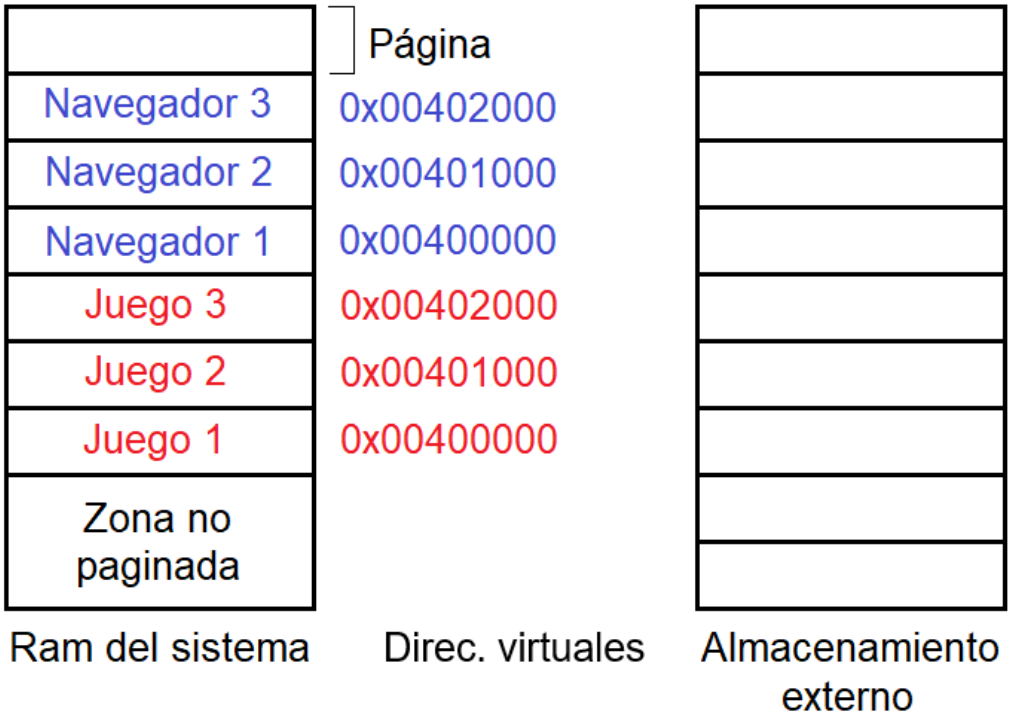
En todos los sistemas siempre hay un área no paginada, que incluye la zona de datos y código del kernel que realiza el

manejo de memoria, los manejadores de interrupciones y excepciones, etc.

Esquema de memoria virtual:



A medida que se van cargando el código y los datos de los diferentes procesos en memoria, se van llenando las páginas que se encuentran en RAM. Como la cantidad de memoria RAM de hoy en día se mide en gigabytes, podemos ver que hay millones de páginas en el sistema. En el diagrama solo se ven pocas páginas para poder mostrar lo que ocurre cuando se llena la memoria física.



Las direcciones lineales (terminología Intel) o virtuales (terminología ARM) son aquéllas a las que tiene acceso el programador. Todos los punteros en lenguaje C o los accesos a memoria en lenguaje ensamblador se refieren a estas direcciones virtuales.

Las direcciones físicas son aquéllas que salen por los pines del bus de direcciones del microprocesador. Como se puede apreciar en el diagrama anterior, es posible que una dirección virtual en dos procesos diferentes corresponda a distintas direcciones físicas.

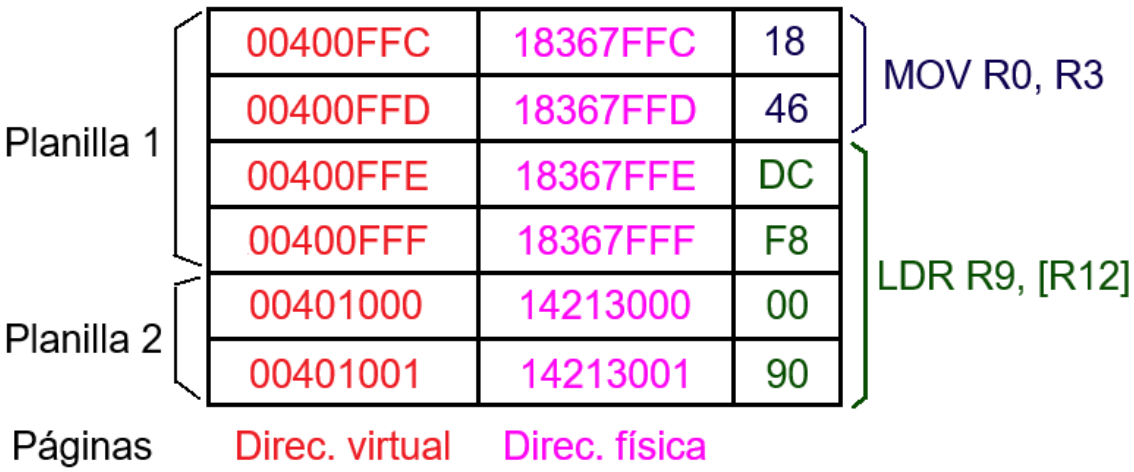
En versiones antiguas de los sistemas operativos más conocidos, los procesos siempre comenzaban en la misma dirección virtual. Por ejemplo, en Windows comenzaba en 0x00400000 (32 bits) o bien 0x140000000 (64 bits). En Linux también se usaba la dirección virtual 0x00400000.

Debido a ataques a los sistemas operativos, éstos hicieron cambios para que los procesos arranquen en direcciones diferentes cada vez que comienzan su ejecución. Este esquema se llama *Address Space Layout Randomization* (ASLR) y por eso hoy en día no se puede saber a priori la dirección virtual en donde arranca un programa. Los programas viejos se deben recompilar para que puedan usar ASLR.

Una vez que se llena la memoria RAM y se carga un nuevo programa o uno existente pide más memoria (usando malloc, por ejemplo), el sistema envía las páginas menos usadas al medio de almacenamiento externo y las elimina de la RAM, haciendo lugar para que se carguen páginas correspondientes al nuevo programa o la memoria adicional correspondiente a un programa en ejecución.

Como se observa en el diagrama, no es necesario que las páginas estén en orden ni que sean consecutivas. De hecho, es posible que parte de una instrucción a ejecutar esté en una página y otra parte en otra página diferente.

Por ejemplo, el código en binario de la instrucción LDR R9, [R12] en un procesador ARMv7 en el estado Thumb es DC F8 00 90. Como Thumb solo requiere que las instrucciones comiencen en una dirección par, podemos lograr que parte de la instrucción esté al final de una página y otra parte al principio de otra página, como muestra el siguiente esquema:



Es decir que el procesador debe ser capaz de poder ejecutar la instrucción, aunque no esté en direcciones consecutivas de memoria RAM.

Paginación

Para poder soportar la memoria virtual, el procesador debe implementar paginación. Este hardware adicional permite traducir eficientemente entre direcciones virtuales y físicas y también puede detectar rápidamente que la página no está presente porque se envió al almacenamiento externo.

En este último caso, el procesador ejecutará un manejador de excepción en el que el sistema operativo deberá recargar del almacenamiento externo la página que se desea acceder para su ejecución o para lectura o escritura de datos.

Leyendo determinados registros del procesador, el manejador puede saber cuál es la dirección virtual que produjo el fallo y la causa (por ejemplo, página no presente, intento de escritura en página de solo lectura, intento de acceso de código de nivel de usuario a página de supervisor, etc.)

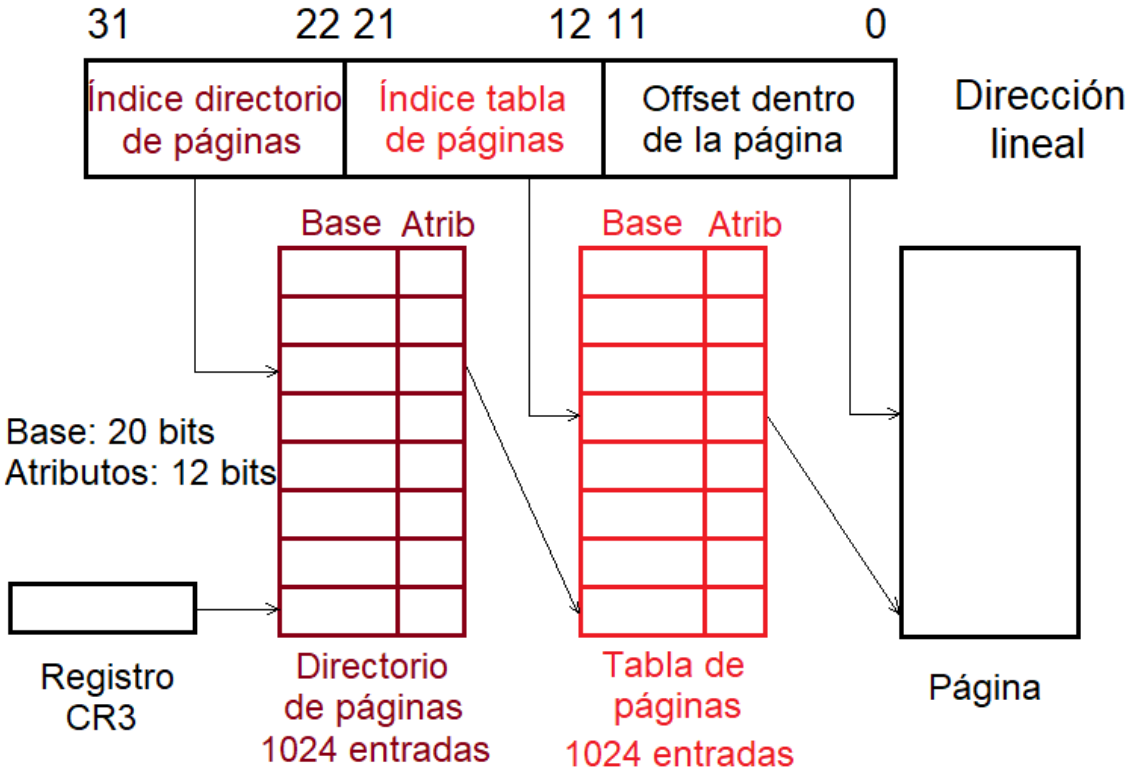
Para convertir las direcciones virtuales a direcciones físicas, el procesador utiliza tablas de paginación que el kernel puso en memoria antes de activar el sistema de paginación (en algunos procesadores también llamado MMU, por Memory Management Unit).

Las tablas se pueden modificar dinámicamente para ir agregando y quitando páginas de memoria virtual. Esto requiere que el código le avise al procesador qué se tocó en las tablas de paginación.

Cada proceso tiene su propia traducción de memoria virtual a memoria física, por lo que a cada proceso le corresponden tablas independientes ubicadas en espacio de kernel.

A continuación, se puede ver un diagrama básico de paginación que corresponde al procesador 80386 de Intel que solo soporta páginas de 4 KiB.

Se puede observar que la dirección virtual se subdivide en varias partes (en este caso 3, otros procesadores pueden usar otra cantidad). La parte baja (en este caso 12 bits) no participa en la traducción, por lo que los 12 bits menos significativos de la dirección virtual son iguales a los 12 bits menos significativos de la dirección física.



Siempre hay un registro interno del procesador (en este caso el registro CR3), que apunta a la primera tabla de la jerarquía de paginación, que también se denomina tabla de primer nivel de paginación.

Cada elemento de la tabla tiene dos partes: un puntero base y atributos de la página. Dicho puntero base puede apuntar a otra tabla de paginación (por ejemplo, los elementos del directorio de páginas apuntan a distintas tablas de páginas) o bien a la página que contiene el código o los datos que nos interesan.

La página siempre arranca en una dirección que es múltiplo del tamaño de dicha página. Es decir que, si el tamaño de la página es de 4 KiB, las páginas arrancan en una dirección física que termina en 000 hexadecimal. De esta manera, se puede ver que los 12 bits menos significativos del puntero siempre van a valer cero. En vez de guardar ceros en la tabla de paginación, se pueden utilizar esos bits para almacenar atributos.

Los atributos de la página indican entre otras cosas, si la página está presente o no (en este último caso significa que el dato o código está en almacenamiento externo o bien que la dirección virtual no le corresponde a ese proceso), si es de lectura/escritura o solo lectura, si le pertenece al usuario o al supervisor, etc. Está claro que diferentes procesadores usarán distintos atributos.

El procesador interpreta los bloques que componen la dirección virtual como índices a las tablas de paginación que le corresponden. En el caso que estamos viendo, como los dos bloques de la dirección virtual tienen 10 bits cada uno, significa

que ambas tablas de paginación tienen 2^{10} = 1024 entradas. Otros procesadores usan otra cantidad de bits para cada una de las subdivisiones de la memoria virtual.

De esta manera, se puede ver que en este esquema que vemos en el procesador 80386 siempre hay una tabla de directorio de páginas y hasta 1024 tablas de páginas por proceso. No hace falta reservar espacio para todas las tablas de páginas. Si el proceso no usa las direcciones virtuales 0x00XXXXXX (los primeros 16 MiB de direcciones virtuales), se pueden marcar los atributos de los primeros cuatro elementos de la tabla de directorio de páginas como no presente y por lo tanto, no hace falta reservar espacio para las primeras cuatro tablas de páginas.

Algunos procesadores soportan páginas más grandes: si el elemento del directorio de páginas tiene un bit encendido, entonces el puntero que se encuentra en dicho elemento no apunta a otra tabla de paginación, sino que apunta directamente a una página de 4 MiB. Esto ocurre porque la subdivisión del medio de la dirección virtual no entra en la traducción. Así, los bits 21 a 0 corresponden al offset dentro de la página y obtenemos páginas de tamaño 2^{22} = 4 MiB. Si ese bit de atributo está apagado, entonces el puntero apunta a una tabla de páginas.

Manejo de punteros nulos

Si bien cualquier puntero se puede cargar con NULL en lenguaje C, si luego se usa esa variable para acceder memoria (sea lectura o escritura), debe ocurrir una excepción de puntero nulo (*Null Pointer Exception*). En Linux esto genera una señal SIGSEGV (Violación de segmento).

Para lograr esto, se puede marcar la primera entrada del primer nivel de paginación como no presente. Esto genera la excepción si se quiere acceder a alguna de las direcciones virtuales 0x00000000 – 0x003FFFFF (los primeros 4 MiB). Esto es útil porque si la variable cargada con NULL es un puntero a estructura, podemos lograr la excepción al acceder a cualquier campo de la estructura (suponiendo que su longitud sea menor que 4 MiB).

Identity mapping

Se dice que una página tiene *identity mapping* cuando la MMU está habilitada y la dirección virtual coincide con la física. Este concepto es importante cuando se habilita la MMU porque antes de que esto ocurra, las direcciones virtuales y físicas son

las mismas. Por lo tanto, el código que habilita la paginación (o el que lo deshabilita) debe tener *identity mapping*.

Translation lookaside buffer **(TLB)**

Por lo visto hasta ahora, cuando la paginación está activada, si se desea acceder a memoria, el procesador primero tiene que determinar la dirección física y luego realizar la lectura o escritura. Por lo tanto, cada acceso a memoria requiere tres, ya que hay dos accesos a las tablas de paginación.

Para evitar este problema, el procesador retiene las últimas traducciones de memoria virtual a memoria física en un bloque interno denominado TLB. El procesador tiene tantos TLB como tamaños de página.

El TLB es una caché completamente asociativa que tiene el siguiente formato:

La cantidad de bits de los campos Dirección virtual y física es la cantidad de bits que participan de la traducción. En el caso de páginas de 4 KiB, sólo participan 20 bits en la traducción, ya que los 12 bits menos significativos no se modifican. Por lo tanto, los campos Dirección virtual y Dirección fiscal tienen 20 bits en este caso.

El procesador obtiene los atributos de la página de las tablas de paginación.

Los atributos del caché indican si la línea tiene una traducción válida y almacena información LRU (*Least Recently Used*) para poder eliminar las traducciones que no se usaron hace más

tiempo cuando se llena la TLB y hace falta guardar otra traducción.

Cuando el procesador tiene que hacer una traducción, primero verifica si ya se realizó dicha traducción previamente. Para eso compara los bits que se van a traducir de la dirección virtual (20 en nuestro caso) con los que figuran en el TLB utilizando tantos comparadores como entradas en la TLB (así funcionan los cachés completamente asociativos).

Si hay coincidencia, obtiene de la TLB tanto la dirección física como los atributos de la página.

En caso contrario, el procesador debe hacer una o dos lecturas en las tablas de paginación para obtener la traducción y los atributos de la página. Esta información ingresa en la TLB, liberando una entrada vieja si la TLB está llena.

Como la comparación se hace contra las direcciones virtuales guardadas en la TLB y los diferentes procesos del sistema pueden reusar las direcciones virtuales, es imperativo que la TLB solo tenga las traducciones del proceso actual y las páginas “globales”, que son aquéllas en las que no se modifica la traducción al cambiar de proceso.

Paginación en ARMv7-A

El bloque funcional que permite la paginación se denomina *Virtual Memory System Architecture* (VMSA). La VMSA es un desarrollo que tiene varias iteraciones desde la versión 4 de ARM.

La VMSA soporta tres niveles de privilegio:

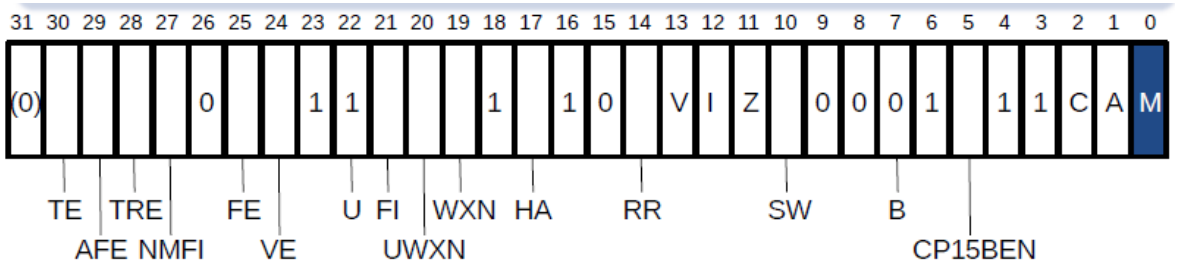
- PL0: Software corriendo en modo usuario. También se denomina de ejecución no privilegiada.
- PL1: Software corriendo en otro modo.
- PL2: En las implementaciones que incluyen extensiones de virtualización, el software que corre en el modo hipervisor ejecuta en el nivel 2.

En este documento solo se va a tratar los niveles de privilegio PL0 y PL1.

Para programar los registros del VMSA se utiliza el coprocesador 15, que se accede mediante las instrucciones MRC p15, … (lectura del coprocesador) y MCR p15, … (escritura al coprocesador).

Si bien ARMv7 es un procesador de 32 bits, algunas implementaciones soportan 40 bits de bus de direcciones. En este caso, las tablas de paginación tienen dos formatos: descriptor largo (para 40 bits) y corto (para 32 bits). Nosotros siempre vamos a usar el descriptor corto.

A continuación, se muestra el registro de control del sistema (SCRTL) que se accede solamente en PL1. Para habilitar la MMU hay que encender el bit 0, SCTRL.M.



Las instrucciones para habilitar la MMU son:

- MRC p15, 0, r0, c1, c0, 0
- ORR r0, #0x01
- MCR p15, 0, r0, c1, c0, 0

En esta secuencia se puede cambiar el registro r0 por cualquier otro registro de uso general.

Otros bits de interés del registro SCTRL:

- Bit 13 = 0: Base de vectores de excepción 0x00000000.
- Bit 13 = 1: Base de vectores de excepción 0xFFFF0000.
- Bit 12 = Habilitación de caché de instrucciones.
- Bit 2 = Habilitación de caché de datos o caché unificado.
- Bit 1 = Habilitación de fallo de alineación.

Antes de habilitar el bit 0, el programador del sistema debe generar las tablas de paginación y hacer que el registro TTBR0 (perteneciente a la MMU) apunte a la base de la tabla de primer nivel.

Existen cuatro tamaños de páginas en los sistemas ARMv7-A:

- **Súper secciones**: Bloques de memoria de 16 MiB. El soporte es opcional, aunque en el caso de que el procesador soporte más de 32 bits de direcciones físicas, su implementación es obligatoria.
- **Secciones**: Bloques de memoria de 1MiB.
- **Páginas grandes**: Bloques de memoria de 64 KiB.
- **Páginas pequeñas**: Bloques de memoria de 4 KiB.

Las súper secciones y las secciones operan con un solo nivel de tablas de paginación. En los otros dos casos, se usan dos niveles.

Tablas de paginación

La **tabla de primer nivel** tiene 4096 entradas de cuatro bytes cada uno, es decir, 16 KiB. Los elementos de esta tabla contienen las direcciones físicas base y los atributos de las súper secciones o secciones, o bien un puntero a una tabla de segundo nivel y atributos.

Las **tablas de segundo nivel** tienen 256 entradas de cuatro bytes cada uno, es decir, 1 KiB. Los elementos de esta tabla contienen las físicas base y los atributos de las páginas grandes y pequeñas. La tabla de segundo nivel también se llama tabla de páginas.

Las súper secciones ocupan 16 entradas consecutivas de la tabla de primer nivel. Dichas entradas deben tener el mismo contenido. El índice de la primera entrada debe ser múltiplo de 16. Lo mismo ocurre con las páginas grandes en las tablas de segundo nivel.

Las entradas en la tabla de primer nivel son las siguientes:

<table>
<tr><th>Bits</th><th>Tabla de páginas</th><th>Sección 1 MiB</th><th>Súper Sección 16 MiB</th><th>Entrada inválida</th></tr>
<tr><td>31-24</td><td rowspan="10">Dirección física base (31-10)</td><td rowspan="2">Base (31-20)</td><td>Base (31-24)</td><td rowspan="14">Ignorado</td></tr>
<tr><td>23-20</td><td>Base (35-32)</td></tr>
<tr><td>19</td><td>0</td><td>1</td></tr>
<tr><td>18</td><td colspan="2">nG</td></tr>
<tr><td>17</td><td></td><td></td></tr>
<tr><td>16</td><td></td><td></td></tr>
<tr><td>15</td><td colspan="2">AP[2]</td></tr>
<tr><td>14-12</td><td colspan="2">TEX[2:0]</td></tr>
<tr><td>11-10</td><td colspan="2">AP[1:0]</td></tr>
<tr><td>9</td><td colspan="3">Definido por la implementación</td></tr>
<tr><td>8-5</td><td colspan="2">Dominio</td><td>Base (39-36)</td></tr>
<tr><td>4</td><td>SBZ</td><td colspan="2">XN</td></tr>
<tr><td>3</td><td>NS</td><td colspan="2">C</td></tr>
<tr><td>2</td><td>PXN</td><td colspan="2">B</td></tr>
<tr><td>1</td><td>0</td><td colspan="2">1</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td colspan="2">PXN</td><td>0</td></tr>
<tr><th>Bits</th><th>Tabla de páginas</th><th>Sección 1 MiB</th><th>Súper Sección 16MiB</th><th>Entrada inválida</th></tr>
</table>

Se puede observar que las súper secciones tienen dos campos: dirección base extendida 35:32 y dirección base extendida 39:36. Esto permite acceder hasta un 1 TiB de memoria física.

Explicación de los campos:

SBZ = *Should be zero* (Debe ser cero).

XN = *Execute never*. Se utiliza en páginas de datos para prevenir a ejecución de instrucciones en PL0.

PXN = Privilege execute never. Este bit está implementado en algunos procesadores e indica si se puede ejecutar la instrucción en PL1.

Los bits TEX[2], TEX[1], TEX[0], C y B indican cómo se comporta la página, es decir si las celdas corresponden a memoria RAM o a dispositivo, el tipo de caché (write-back, write-through), etc.

Los bits AP[2], AP[1] y AP[0] (*Access Permission*) indican los accesos a la página:

AP[2]	AP[1]	AP[0]	Acceso PL1	Acceso PL0
0	0	0	Sin acceso	Sin acceso
0	0	1	Lectura/escritura	Sin acceso
0	1	0	Lectura/escritura	Solo lectura
0	1	1	Lectura/escritura	Lectura/escritura
1	0	0	Reservado	Reservado
1	0	1	Solo lectura	Sin acceso
1	1	0	Solo lectura	Solo lectura
1	1	1	Solo lectura	Solo lectura

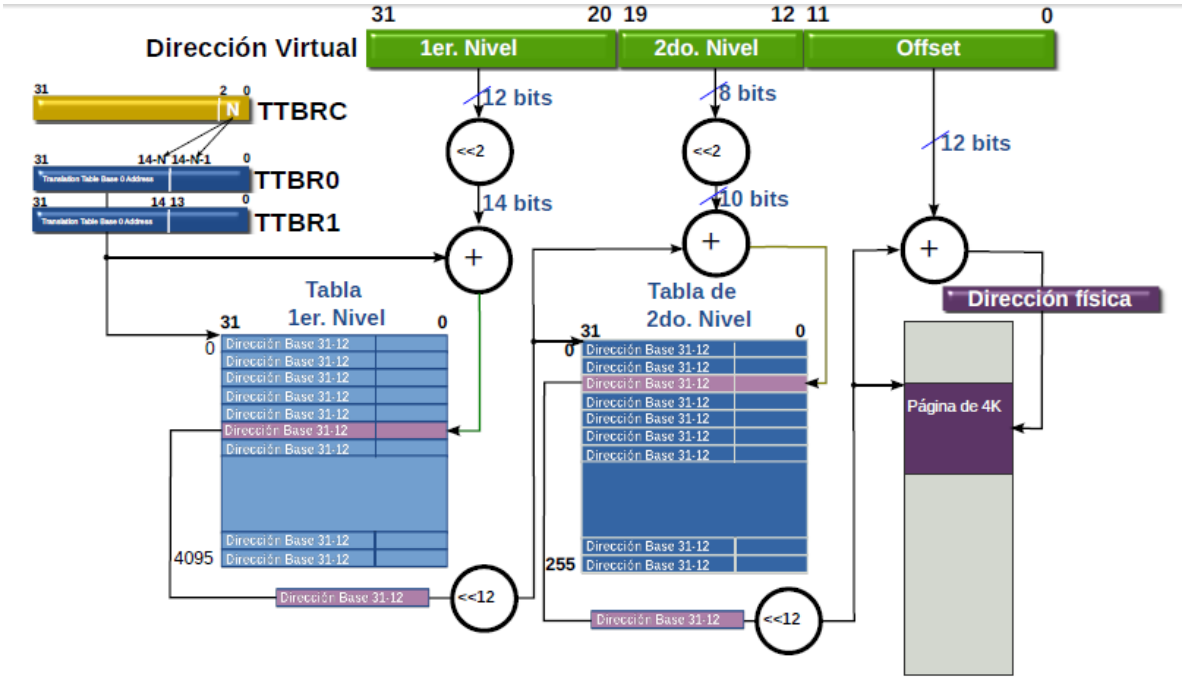
Esta tabla es válida cuando SCTRL.AFE (bit 29) vale 0.

El bit nG (*non-global*) se pone a ‘1’ si la página corresponde a una tarea específica del kernel o si es una página de aplicación. Se pone a ‘0’ si la entrada es global y corresponde a todas las tareas del sistema.

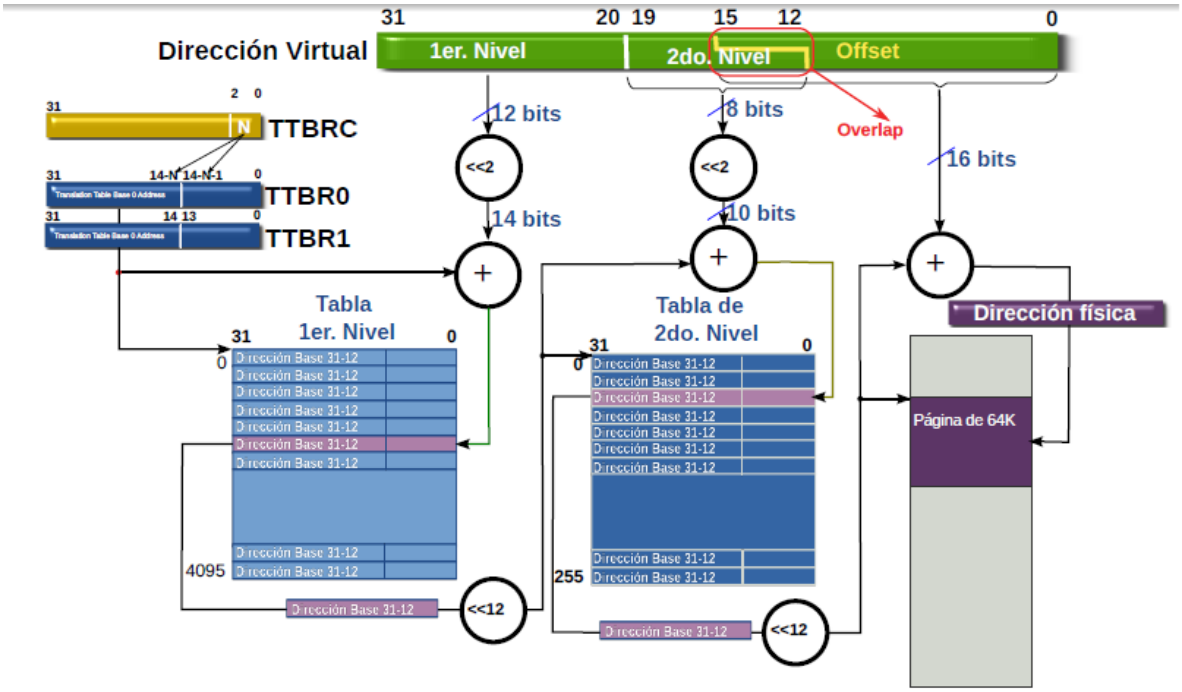
Las entradas en las tablas de segundo nivel son las siguientes:

<table>
<tr><th>Bits</th><th>Página pequeña 4 KiB</th><th>Página grande 64 KiB</th><th>Entrada Inválida</th></tr>
<tr><td>31-16</td><td rowspan="3">Dirección física base (31-12)</td><td>Base (31-16)</td><td rowspan="11">Ignorado</td></tr>
<tr><td>15</td><td>XN</td></tr>
<tr><td>14-12</td><td>TEX[2:0]</td></tr>
<tr><td>11</td><td colspan="2">nG</td></tr>
<tr><td>10</td><td colspan="2">S</td></tr>
<tr><td>9</td><td colspan="2">AP[2]</td></tr>
<tr><td>8-6</td><td>TEX[2:0]</td><td>SBZ</td></tr>
<tr><td>5-4</td><td colspan="2">AP[1:0]</td></tr>
<tr><td>3</td><td colspan="2">C</td></tr>
<tr><td>2</td><td colspan="2">B</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>XN</td><td>1</td><td>0</td></tr>
<tr><th>Bits</th><th>Página pequeña 4 KiB</th><th>Página grande 64 KiB</th><th>Entrada Inválida</th></tr>
</table>

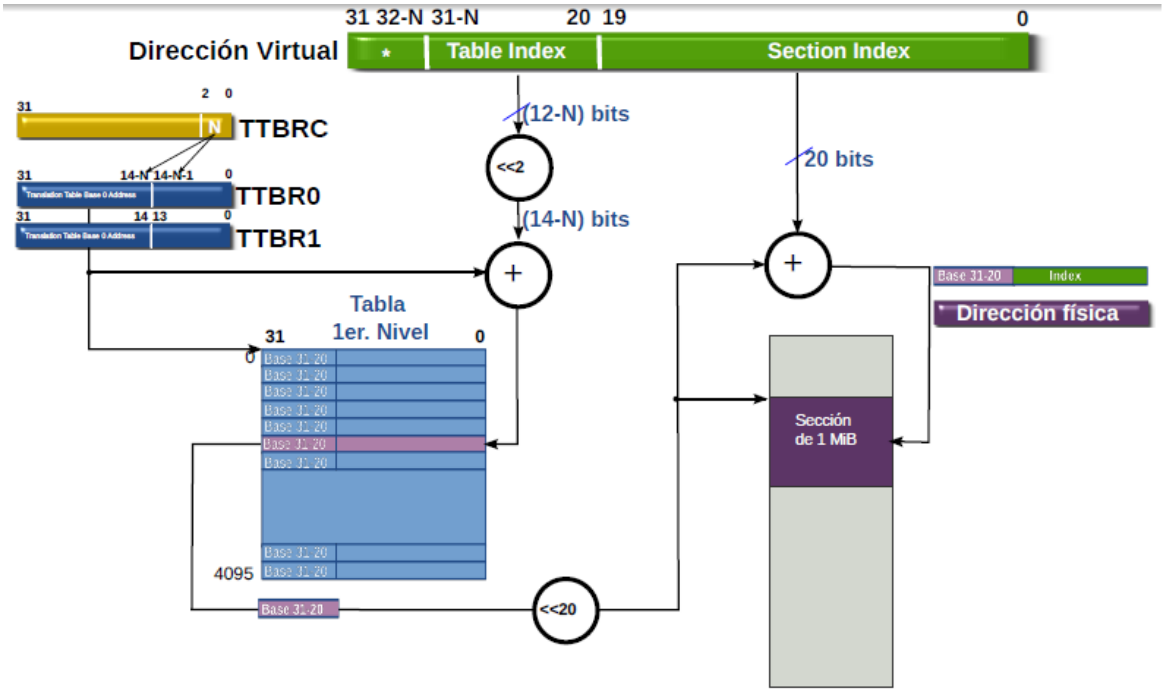
Traducción de páginas de 4 KiB:



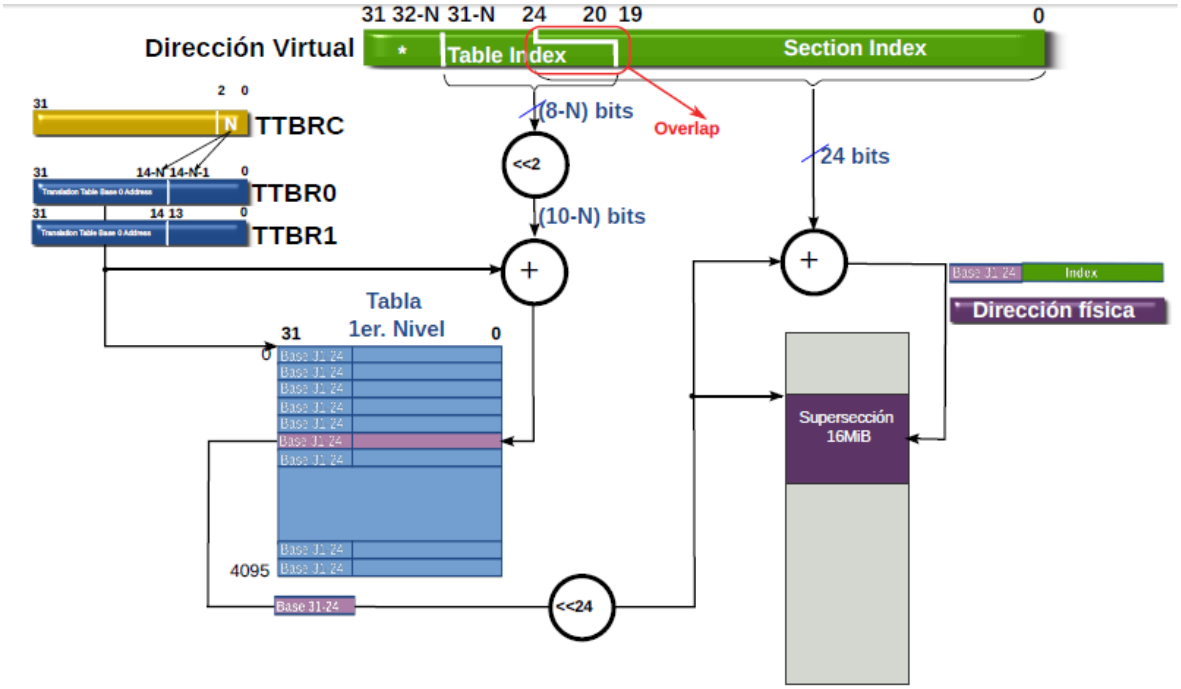
Traducción de páginas de 64 KiB:



Traducción de secciones de 1 MiB:



Traducción de super secciones de 16 MiB:



Registros del sistema

El registro TTBR0 (en versiones anteriores de ARM se llamaba TTBR = *Translation Table Base Register*) apunta a la base de la tabla de primer nivel. También existe el registro TTBR1 que no se va a analizar en este documento. El puntero a la tabla de primer nivel se encuentra en los bits 31 a “x”, donde ese valor de “x” (de 7 a 14) se puede seleccionar en el registro TTBCR. Para mayor compatibilidad, lo mejor es que la base de la tabla de primer nivel esté en una dirección múltiplo de 16 KB (x = 14).

- Lectura del registro TTBR0: MRC p15, 0, <Rt>, c2, c0, 0
- Escritura del registro TTBR0: MCR p15, 0, <Rt>, c2, c0, 0

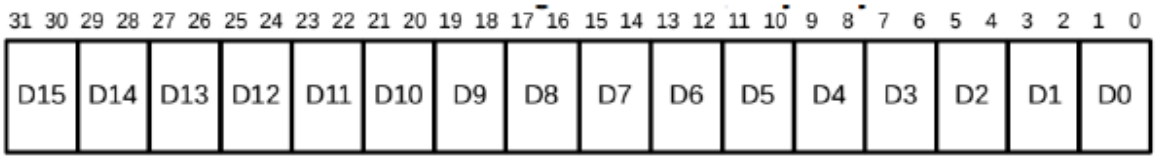
El registro Rt puede ser cualquier registro de uso general.

Dominios

Permite describir la memoria en hasta 16 formas distintas. Los descriptores de páginas grandes y pequeñas y de las secciones tienen un campo de cuatro bits que indican a qué dominio pertenece.

Los descriptores de las súper secciones no tienen campo de dominio. Siempre pertenecen al dominio cero.

El registro DACR define el acceso para cada dominio:



El número indicado es el número de dominio.

Los bits de permisos son:

- 00: Sin acceso: Los accesos a páginas o secciones que tengan este dominio generan fallo de dominio.
- 01: Cliente: En este caso se verifican los atributos de permisos de acceso y es posible que se genere un fallo de permisos.
- 10: Reservado
- 11: Manager: En este caso no se verifican los atributos de permiso de acceso.

El acceso al registro DACR opera de la siguiente forma:

- Lectura del registro DACR: MRC p15, 0, <Rt>, c3, c0, 0
- Escritura del registro DACR: MCR p15, 0, <Rt>, c3, c0, 0

El registro Rt puede ser cualquier registro de uso general.

Para aplicaciones sencillas que usan PL0 y PL1, siempre se define los dominios como cliente para que se verifiquen los atributos que se encuentran en las entradas de las tablas de primer y segundo nivel de paginación. De esta manera en la inicialización se debe cargar el registro DACR con el valor 0x55555555.

Ejemplo: Indicar las direcciones y el contenido de los descriptores de las tablas de primer y segundo nivel para lograr las siguientes traducciones:

- Dir. virtual 0x70010000 -> Dir física 0x70010000 (4KiB) sólo lectura en PL0 y lectura/escritura en PL1.
- Dir virtual 0x80000000 -> Dir física 0x70800000 (64 KiB) lectura y escritura en ambos niveles de privilegio.

La ubicación de las bases de las tablas de paginación deberá ser:

- Tabla de primer nivel: 0x70080000
- Primera tabla de segundo nivel: 0x70090000
- Segunda tabla de segundo nivel: 0x700A0000

Primera traducción:

- Dividimos la dirección virtual en tres partes para obtener los índices a las tablas:

700 10 000 → índice tabla primer nivel 0x700, segundo nivel 0x10.

- Dirección física de la tabla de primer nivel a escribir:
  Base tabla primer nivel + índice * 4 =
  0x70080000 + 0x700 * 4 = 0x70081C00
- Valor del descriptor a escribir en 0x70081C00 (ver entrada tabla de página en la tabla de primer nivel más arriba, el atributo vale 1):
  Base primera tabla de segundo nivel + atributos =
  = 0x70090000 + 1 = 0x70090001 (apuntamos a primera tabla)
- Dirección física de la tabla de segundo nivel a escribir:
  Base primera tabla de segundo nivel + índice * 4 =
  0x70090000 + 0x10 * 4 = 0x70090040
- Valor del descriptor a escribir en 0x70090040 (ver entrada página pequeña en la tabla de segundo nivel más arriba):
  Sólo lectura en PL0 y lectura/escritura en PL1 corresponde AP[2]: AP[1]: AP[0] = Bits 9, 5, 4 = 010. El atributo vale 0x22.
  Dirección base página 4 KB + atributos =
  = 0x70010000 + 0x22 = 0x70010022.

Segunda traducción:

- Dividimos la dirección virtual en tres partes para obtener los índices a las tablas:
  800 00 000 → índice tabla primer nivel 0x800, segundo nivel 0x00.

- Dirección física de la tabla de primer nivel a escribir:
  Base tabla primer nivel + índice * 4 =
  = 0x70080000 + 0x800 * 4 = 0x70082000
- Valor del descriptor a escribir en 0x70082000 (ver entrada tabla de página en la tabla de primer nivel más arriba, el atributo vale 1):
  Base segunda tabla de segundo nivel + atributos =
  = 0x700A0000 + 1 = 0x700A0001
- Dirección física de la tabla de segundo nivel a escribir:
  Base segunda tabla de segundo nivel + índice * 4 =
  = 0x700A0000 + 0x00 * 4 = 0x700A0000
- Valor del descriptor a escribir en 0x700A0000 (ver entrada página grande en la tabla de segundo nivel más arriba):
  Lectura y escritura en ambos niveles de privilegio corresponde AP[2]: AP[1]: AP[0] = Bits 9, 5, 4 = 011. El atributo vale 0x31.
  Dirección base página 64 KB + atributos =
  = 0x70800000 + 0x31 = 0x70800031.
- Como se indicó más arriba, se debe escribir el mismo valor 0x70800031 en las 16 entradas de la segunda tabla de paginación a partir de la dirección 0x700A0000.

El registro TTBR0 se debe cargar con el valor 0x70080000. Las entradas no usadas de las tablas se deben cargar con ceros.

Paginación en multitarea

Con respecto al manejo de traducciones que pertenecen a diferentes procesos, ARM usa el concepto de “Address Space Identifier” (ASID) en el que cada proceso usa un número diferente elegido por el sistema operativo.

De esta manera, una traducción incluye la dirección virtual y el ASID y la unidad de paginación convierte eso en dirección física.

El ASID actual (del proceso que está corriendo) reside en un registro del coprocesador 15 dentro del registro CONTEXTIDR.

Los bits 31 a 8 de CONTEXTIDR contienen PROCID (Process ID), mientras que los bits 7 a 0 contienen el ASID.

Para acceder al registro CONTEXTIDR se utilizan las siguientes instrucciones:

- MRC p15, 0, <Rt>, c13, c0, 1 /* Rt <- CONTEXTIDR */
- MCR p15, 0, <Rt>, c13, c0, 1 /* CONTEXTIDR <- Rt */

Las páginas que usen ASID deben poner el bit nG (non-global) a uno. Ese es el bit 11 de la entrada de la tabla del primer nivel de

paginación cuando se usan páginas de 1MB o del segundo nivel de paginación para páginas de otro tamaño.

Las páginas que se usan en todos los procesos (generalmente código y datos de kernel), deben poner el bit nG a cero.

En el cambio de tarea, se debe modificar el valor del registro CONTEXTIDR para que cada tarea use su propio valor de ASID.

Las operaciones necesarias para modificar TTBR0 cuando ocurre un cambio de contexto son las siguientes:

1) Cambiar ASID a cero.
2) Ejecutar la instrucción ISB
3) Cambiar TTBR0 para que apunte a la tabla de primer nivel que corresponde al nuevo proceso.
4) Ejecutar la instrucción ISB nuevamente.
5) Cambiar ASID al valor que corresponde al nuevo proceso.

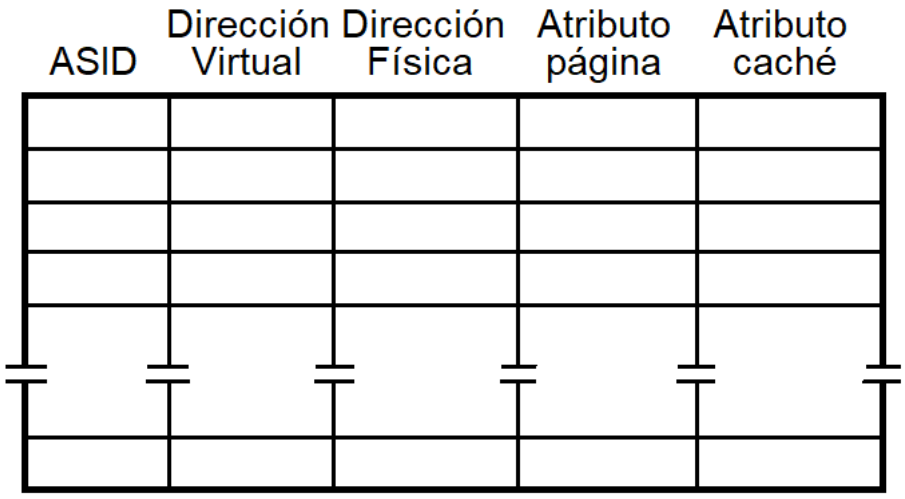
Esto supone que no se usa el valor cero para ASID en ninguna tarea.

Translation lookaside buffer en ARMv7

El “translation lookaside buffer” (TLB) es un caché completamente asociativo que tiene las últimas traducciones de direcciones virtuales a direcciones físicas.

Cuando el procesador necesita hacer una traducción, primero verifica si la traducción está presente en la TLB. En caso afirmativo, usa la dirección física y los atributos de página ya almacenados. En caso contrario, el procesador debe acceder a ambas tablas de paginación y luego almacenar la traducción en el TLB.

El formato del TLB es el siguiente:



El campo ASID almacena el valor de ASID cuando ocurrió la traducción, siempre y cuando nG = 1.

Los campos de dirección virtual y física incluyen solo los bits necesarios para la traducción. En el caso de páginas de 4 KiB, estos campos tienen 20 bits.

El campo de atributo de página contiene los atributos que se encontraban en las tablas de paginación en el momento que se hizo la traducción.

Por último, los atributos de caché indican si la entrada de la TLB es válida y también hay bits adicionales para el algoritmo de LRU (las entradas más viejas se descartan para hacer lugar a nuevas traducciones).

Al modificar el registro TTBR0, el procesador mantiene todas las entradas en el TLB.

Se debe tener en cuenta que el TLB soporta 256 valores de ASID diferentes. Si el sistema operativo está corriendo más de 256 procesos al mismo tiempo, deberá borrar manualmente el TLB cuando ocurra un cambio de tarea.

Existen instrucciones para borrar parcial o totalmente el TLB.

Fallos de paginación

En ARMv7-A hay dos tipos de excepciones que pueden ocurrir relacionados con la paginación:

- Prefetch abort: ocurre cuando el procesador lee una instrucción desde la memoria.
- Data abort: ocurre cuando la instrucción en curso accede a datos en memoria.

Una vez que el procesador entra a los manejadores de estas excepciones, el código debe saber cuál es la dirección virtual que produjo la falla y qué fue lo que ocurrió (la causa del fallo).

El registro DFSR tiene información sobre una excepción de data abort, mientras que el registro IFSR tiene información sobre una excepción de prefetch abort.

En ambos registros se puede consultar la causa del fallo en los bits 10, 3, 2, 1 y 0 que colectivamente forman el campo FS (*Fault Status*).

Adicionalmente, en el bit 11 del registro DFSR se puede consultar si el abort fue causado por una instrucción de lectura (bit a cero) o de escritura (bit a uno).

FS	Fuente
00001	Fallo de alineación (solo DFSR)
00010	Evento de debug
00011	Fallo de flag de acceso (solo ocurre si SCTRL.AFE=1) en tabla de primer nivel
00110	Fallo de flag de acceso (solo ocurre si SCTRL.AFE=1) en tabla de segundo nivel
00100	Fallo durante el manejo de caché de instrucciones (solo DFSR)
00101	Fallo de traducción en tabla de primer nivel
00111	Fallo de traducción en tabla de segundo nivel
01000	Abort externo sincrónico
01001	Fallo de dominio en tabla de primer nivel
01011	Fallo de dominio en tabla de segundo nivel
01100	Abort externo sincrónico leyendo tablas de paginación en tabla de primer nivel
01110	Abort externo sincrónico leyendo tablas de paginación en tabla de segundo nivel
01101	Fallo de permisos en tabla de primer nivel
01111	Fallo de permisos en tabla de segundo nivel
10000	Abort por conflicto en TLB
10100	Definido por la implementación (lockdown)
10110	Abort externo asincrónico (solo DFSR)
11000	Error de paridad asincrónico accediendo a memoria (solo DFSR)

11001	Error de paridad sincrónico accediendo a memoria
11010	Definido por la implementación (abort por coprocesador)
11100	Error de paridad sincrónica al leer las tablas de paginación en tabla de primer nivel
11110	Error de paridad sincrónica al leer las tablas de paginación en tabla de segundo nivel

Los registros IFAR y DFAR indican la dirección virtual que provocó el prefetch abort o data abort respectivamente.

Acceso a registros:

Para leer DFSR: MRC p15, 0, <Rt>, c5, c0, 0
Para escribir DFSR: MCR p15, 0, <Rt>, c5, c0, 0
Para leer DFAR: MRC p15, 0, <Rt>, c6, c0, 0
Para escribir DFAR: MCR p15, 0, <Rt>, c6, c0, 0
Para leer IFSR: MRC p15, 0, <Rt>, c5, c0, 1
Para escribir IFSR: MCR p15, 0, <Rt>, c5, c0, 1
Para leer IFAR: MRC p15, 0, <Rt>, c6, c0, 2
Para escribir IFAR: MCR p15, 0, <Rt>, c6, c0, 2