ARM Cortex-A. Модул за организация на паметта (MMU)

Автор: доц. д-р инж. Любомир Богданов

Съдържание

Модул за организация на паметта - въведение

Модул за организация на паметта (Memory Management Unit, MMU) – преобразува физическите (абсолютните) адреси във виртуални (несъществуващи).

Целта е всеки един процес на операционната система да се изпълнява в негово си, частно адресно поле.

Модул за организация на паметта - въведение

Транслиране на адреси – операцията, която MMU извършва – съпоставянето на виртуалните адреси на един процес с физическите адреси в системата.

Транслирането на адреси е **невидимо за програмиста,** т.е. става автоматично в хардуера и не са нужни допълнителни инструкции за управление на транслацията. *Аналогично на кеш паметите*

Модул за организация на паметта - въведение

MMU отговаря за още три операции:
*__права за достъп__ до паметта
*__подредбата__ на паметта (memory ordering)
*__кеш схеми__ на опресняване (cache policy)

Модул за организация на паметта - въведение

Приложните програми, изпълнявани от процесори с MMU се **линкват с виртуални адреси** (за разлика от bare-metal фърмуера).

Операционната система е тази, която **конфигурира** MMU преди да започне транслацията.

Ако **MMU не е включено**, виртуалните адреси ще бъдат директно съпоставени на реалните адреси и **системата няма да функционира**.

Модул за организация на паметта - въведение

Транслационни таблици – част от основната памет (RAM), която съдържа всичката необходима информация, за да е успешно транслирането на виртуалните адреси към физически [1].

Транслационни буфери (Translation Lookaside Buffer, **TLB**) и модул за търсене в таблица (Table Walk Unit, **TWU**) – виж MCxT, презентация 13.

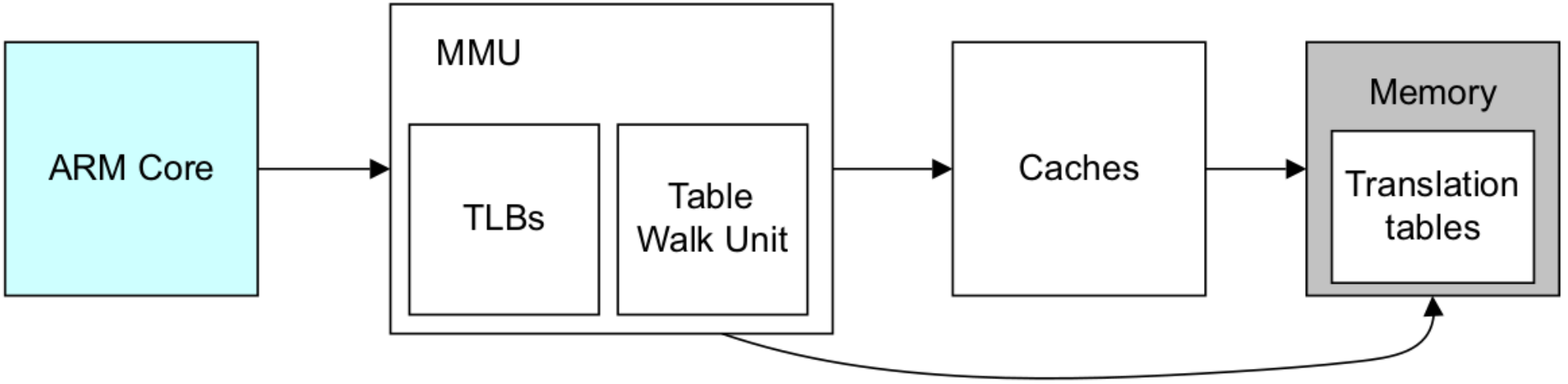


Figure 9-2 The Memory Management Unit

Инициализация на MMU

*Инициализацията на ARM Cortex MMU започва със **зареждане на транслационните таблици** в основната (RAM) памет. !!!!Пример!!!!

*Следва **пускане на MMU** чрез вдигане на бит #0 от контролния регистър на копроцесор #15 чрез копроцесорни инструкции:

```
1MRC p15, 0, R1, c1, C0, 0 ;Read control register
2ORR R1, #0x1              ;Set M bit
3MCR p15, 0,R1,C1, C0,0    ;Write control register and enable MMU
```

*Ако MMU промени адресното поле на кода, който пуска MMU-то, трябва да се използва **бариерна инструкция за паметта** (memory barrier).

Инициализация на MMU

*Ако OS промени транслационните таблици има риск TLB да съдържат стара транслационна информация. Затова OS трябва да невалидира (invalidate) TLB буферите чрез копроцесор #15.

В Линукс има API функции, които извършват тази операция [a]:

```
static inline void flush_tlb_all(void);

static inline void flush_tlb_range
(
        struct vm_area_struct *vma,
         unsigned long start,
        unsigned long end
);
```

Инициализация на MMU

```
static inline void flush_tlb_all(void)
{
    dsb(ishst);
    __tlbi(vmalle1is);
    dsb(ish);
    isb();
}
```

Където [a]:

dsb ishst – гарантира, че предишни опресняванията на транслационните таблици са завършили
tlbi – невалидиране на TLB
dsb ish - гарантира, че TLB невалидирането е завършило
isb – игнорира всички инструкции, които са били захванати (fetch) от старата (вече) карта на паметта

Инициализация на MMU

```
static inline void flush_tlb_range
(
    struct vm_area_struct *vma,
    unsigned long start,
    unsigned long end
)
{
    /*
     * We cannot use leaf-only invalidation here, since we may be invalidating
     * table entries as part of collapsing hugepages or moving page tables.
     * Set the tlb_level to TLBI_TTL_UNKNOWN because we can not get enough
     * information here.
     */
    __flush_tlb_range(vma, start, end, PAGE_SIZE, false,
TLBI_TTL_UNKNOWN);
}
```

Инициализация на MMU

***Страница в паметта (page memory)** – най-малкият регион от виртуална памет, който MMU може да съпостави (map-не) към физически адреси.

***Размерът на страницата (page size)** може да се конфигурира от операционната система (OS).

По-малки размери на страницата → по-прецизен контрол на блок памет + по-малко неизползвана памет.

Пример: ако един процес изисква 7 kB памет, то 2 x 4 kB ще оставят 1 kB неизползван.
Пример: ако един процес изисква 7 kB памет, то 1 x 64 kB ще оставят 57 kB неизползвани.

Инициализация на MMU

По-големи размери на страницата → по-големи размери на TLB буфера => шансът да настъпи TLB попадение (TLB hit), когато процесорът се обръща към паметта е по-голям.

Повече TLB hit означава по-малко TWU търсене във външната RAM, което пък означава по-малко време ще бъде изгубено за транслиране на адресите.

Поради тази причина, в практиката се използват до 16 MB на страница.

Транслация на ниво 1

* Първата стъпка при транслацията на адреса започва с намиране на позицията на виртуалния адрес в транслационната таблица [1].

*Транслационната таблица на първата стъпка, т.е. на първо ниво L1, се нарича **главна (master) транслационна таблица**.

*L1 таблицата разделя едно 4 GB адресно поле на 32-битово ядро на 4096 равни 1-мегабайтови секции.

*L1 съдържа 4096 позиции с 32-битови думи.

Транслация на ниво 1

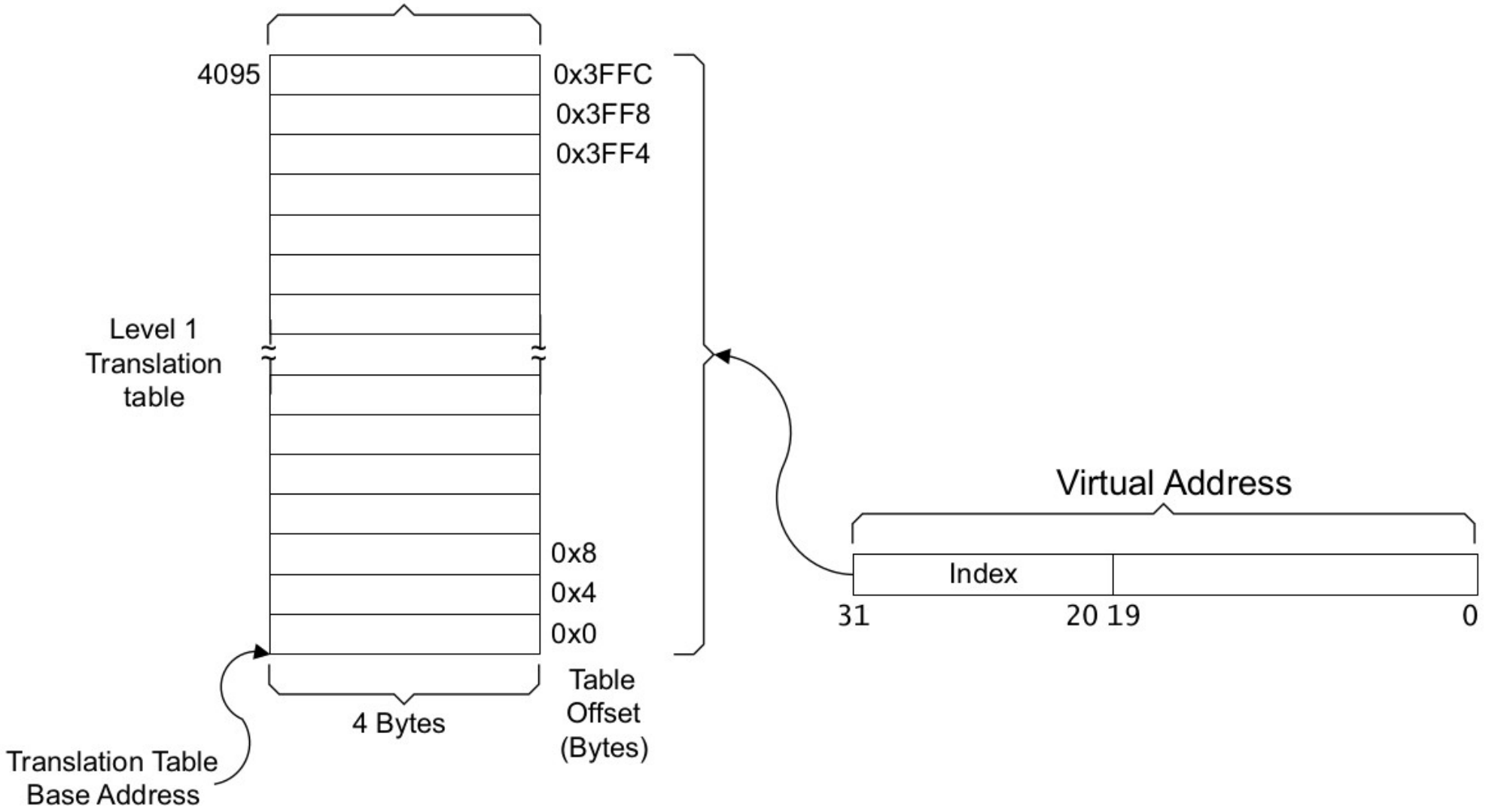


Figure 9-4 Finding the address of the level 1 translation table entry

Транслация на ниво 1

Всяка позиция от L1 може да съдържа:

***указател към базов адрес** от L2 таблица

или

***физически базов адрес** на 1-мегабайтова секция, съпоставена на виртуалния адрес

Младшите битове са едни и същи и в двата случая – указват отместване спрямо базовия адрес.

Транслация на ниво 1

Базов адрес на главна транслационна таблица (translation table base address, TTBA) – физически адрес на транслационната таблица.

TTBA се инициализира от копроцесор #15, регистър C2, който се нарича TTBR или още TTBR0_EL1.

TTBA трябва да е подравнен на 16-kB граница.

При 32-битови виртуални адреси, старшите 12 бита указват индекса в транслационната таблица, на който ще се търси физическият адрес / указателя към L2 таблица.

Транслация на ниво 1

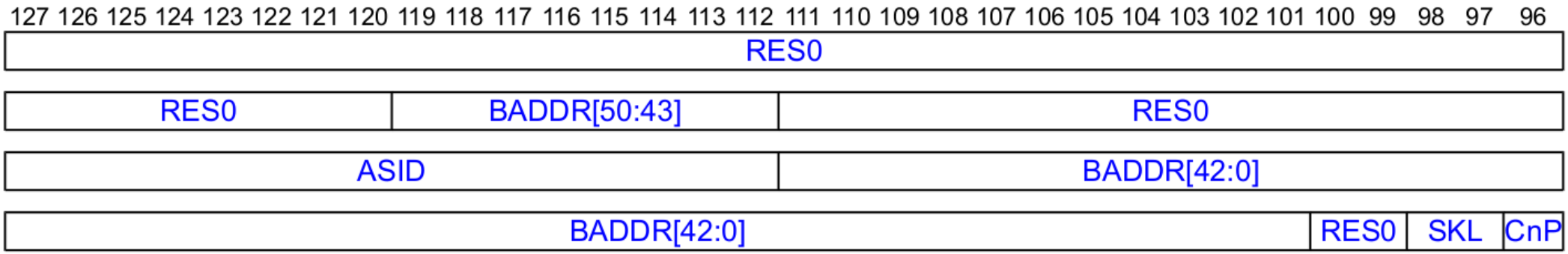
***TTBR** (translation table base register) [3] – това е 64- или 128-битов регистър за задаване на начален адрес на L1 транслационната таблица.

Пример за инициализация на TTBR [2]:

```
LDR r0,=TTB ; Set start of translation table base (on 16KB boundary)
MCR p15,0,r0,c2,c2,0    ; Write value to CP15 c2
```

Транслация на ниво 1

*Структурата на 128-битов TTBR е показана по-долу:



*Битове [127:88, 4:3] – не се използват.
***BADDR** [87:80, 47:5] – 51-битов базов адрес на транслационната таблица.
***ASID** [63-48] – 8- или 16-битово число, указващо номер на процес, използващ виртуалното адресно поле.
***SKL** [2:1] (Skip Level) – позволява да се прескачат нива по време на TWU търсене.

Транслация на ниво 1

Пример:
TTBA = 0x1230.0000 *(адресът е в RAM)*

VA = 0x0010.0000 = 0b**0000.0000.0001**.0000.0000.0000 0000.0000 *(адресът не съществува)*

Индексът от L1 транслационната таблица, на който се намира стойността на физическия адрес, се изчислява като:

TTBA + (table entry 0x001 * 4 bytes) = 0x1230.0000 + 0x004 = 0x1230.0004 *(адресът е в RAM и стойността му сочи към друг физически адрес)*

Транслация на ниво 1

*Процесът на транслация е показан по-долу:

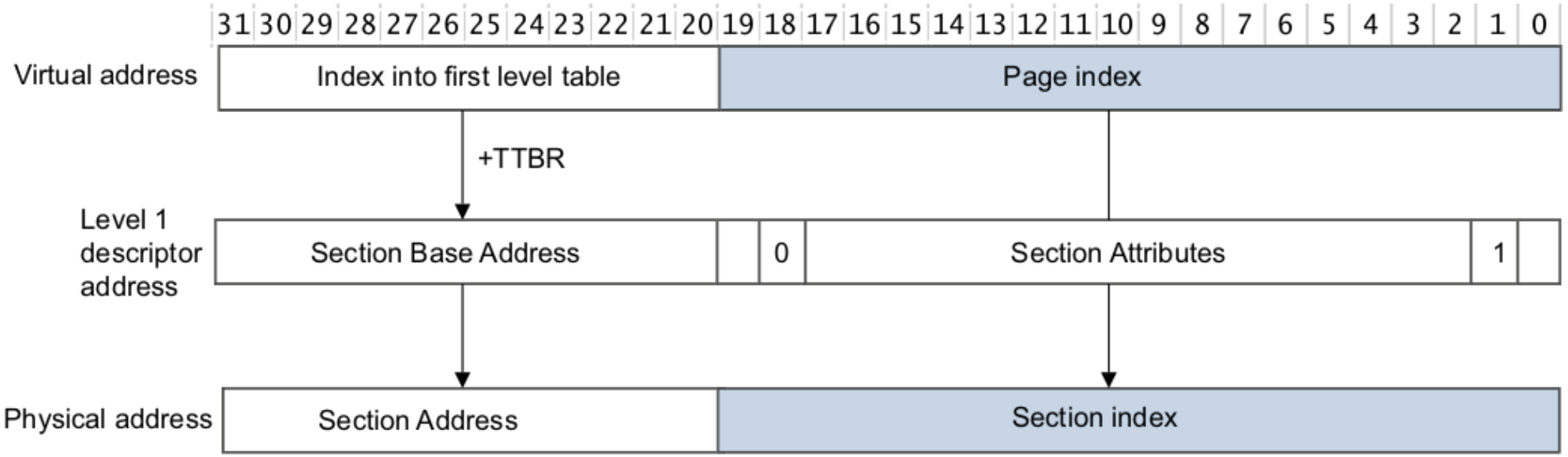


Figure 9-6 First level address translation

Транслация на ниво 1

Стойностите в транслационната таблица имат специален формат, показан по-долу:

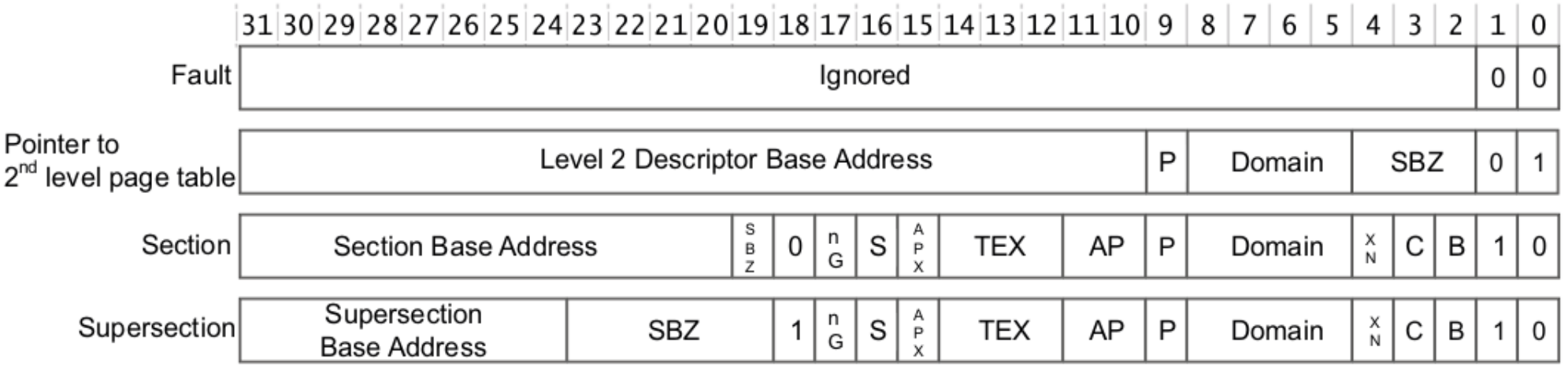


Figure 9-5 Level 1 translation table entry format

Транслация на ниво 1

Битове [1:0]* определят дали стойността е **грешка, **указател** към друга таблица, или базов адрес на **секция** с директно съпоставяне виртуален/физически адрес.

**Бит 18* указва, ако е базов адрес на секция, дали тя е 1- или 16-мегабайтова.

***Грешка** (fault) – тази стойност генерира изключение за прекратяване (Abort). Изключението може да бъде предизвикано от захващане на инструкция (fetch) или достъп до данни. Получава се, когато даден виртуален адрес не е съпоставен на нито един физически такъв.

Транслация на ниво 1

***Указател към L2** транслационна таблица – сочи към 1-мегабайтов регион от L2 таблица, който може да бъде разделен на страници.

***1-мегабайтова секция** – тази стойност съпоставя 1-мегабайтов регион с виртуални адреси към 1-мегабайтов регион със физически адреси.

***16-мегабайтова супер-секция** – специален 1-мегабайтов регион, който запълва 16 позиции от транслационната таблица, но от друга страна заема по-малко място в TLB буфера.

Транслация на ниво 1

*Както беше посочено, стойността от транслационната таблица, отговаряща за 1-мегабайтова секция, съдържа физическия базов адрес, както и още битове за:

→ права за достъп (access permissions, AP)
→ дали да бъде кеширан региона (cacheable, C)
→ дали да бъде буфериран региона (bufferable, B)

Транслация на ниво 1

*16-мегабайтовите супер-секции трябва да бъдат подравнени на 16-мегабайтови граници.

*Стойностите в L1 таблицата описват 1-мегабайтови региони, затова е необходимо да има 16 последователни и идентични стойности в таблицата, за да се формира една супер-секция.

Транслация на ниво 1

*Обобщение:

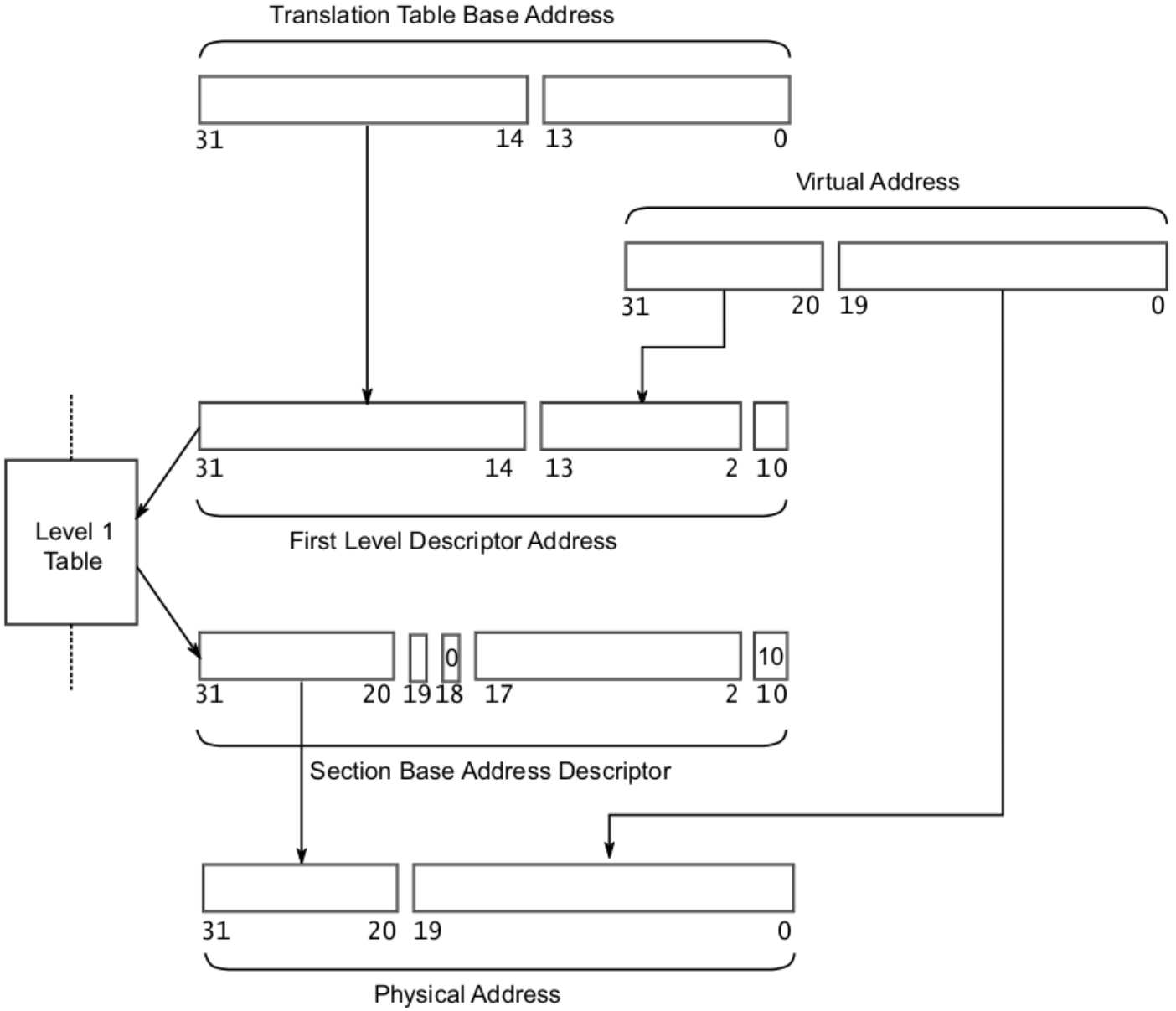


Figure 9-7 Generating a physical address from a level 1 translation table entry

Транслация на ниво 2

*Транслационните таблици на ниво 2 (L2) имат 256 32-битови стойности

*L2 таблиците заемат 1 kB от RAM-та

*Трябва да са подравнени на 1-килобайтови граници

*Всяка стойност транслира 4-килобайтов блок с виртуални адреси към 4-килобайтов блок с физически адреси

*Страниците са с размер 4 kB и 64 kB

Транслация на ниво 2

Аналогично на L1 таблиците, стойностите в L2 таблиците са три вида:

*стойност, сочеща към **малка страница** – размерът на страницата е 4 килобайта

*стойност, сочеща към **голяма страница** – размерът на страницата е 64 килобайта и затова тази стойност трябва да бъде повторена 16 пъти

*стойност за **грешка** (fault) – при опит за достъп до тази стойност се генерира Abort изключение (прекъсване).

Транслация на ниво 2

Структурата на L2 транслационна таблица е показана на фигурата по-долу.

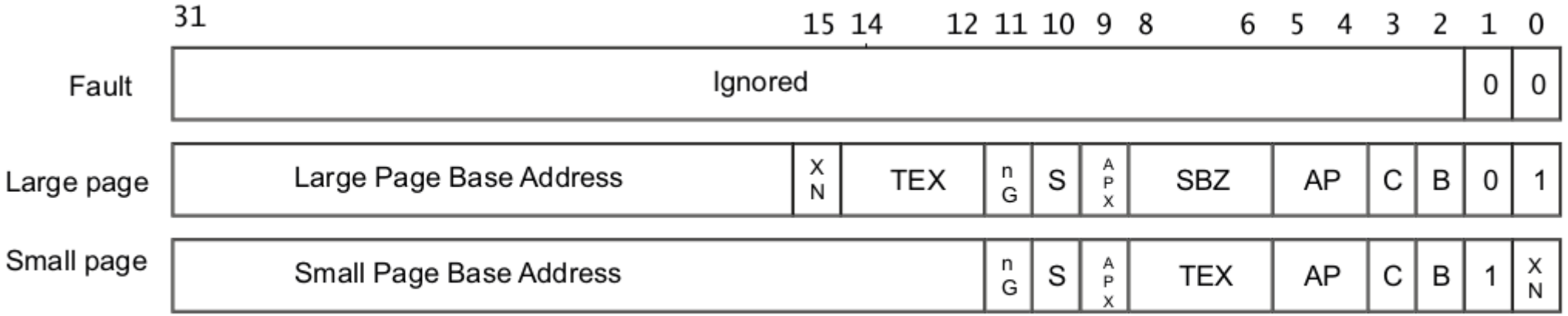


Figure 9-8 Format of a level 2 translation table entry

TEX (type extention), **S** (shearabe), **AP+APX** (Access Permission) – указват атрибутите за достъпа до паметта.

C и **B** – указват схема за зареждане на кеш.

nG (non-global) – бит, указващ дали в TLB да се използва полето ASID *(виж последната глава)*.

Транслация на ниво 2

XN (execute never) – забранява спекулативното захващане на инструкции. Ако бъде опитано изпълнение на инструкции от такъв регион, ще се генерира prefetch abort изключение.

Региони с периферни входно-изходни устройства се маркират като XN.

Транслация на ниво 2

Формирането на адрес от L2 таблица става по следния начин:

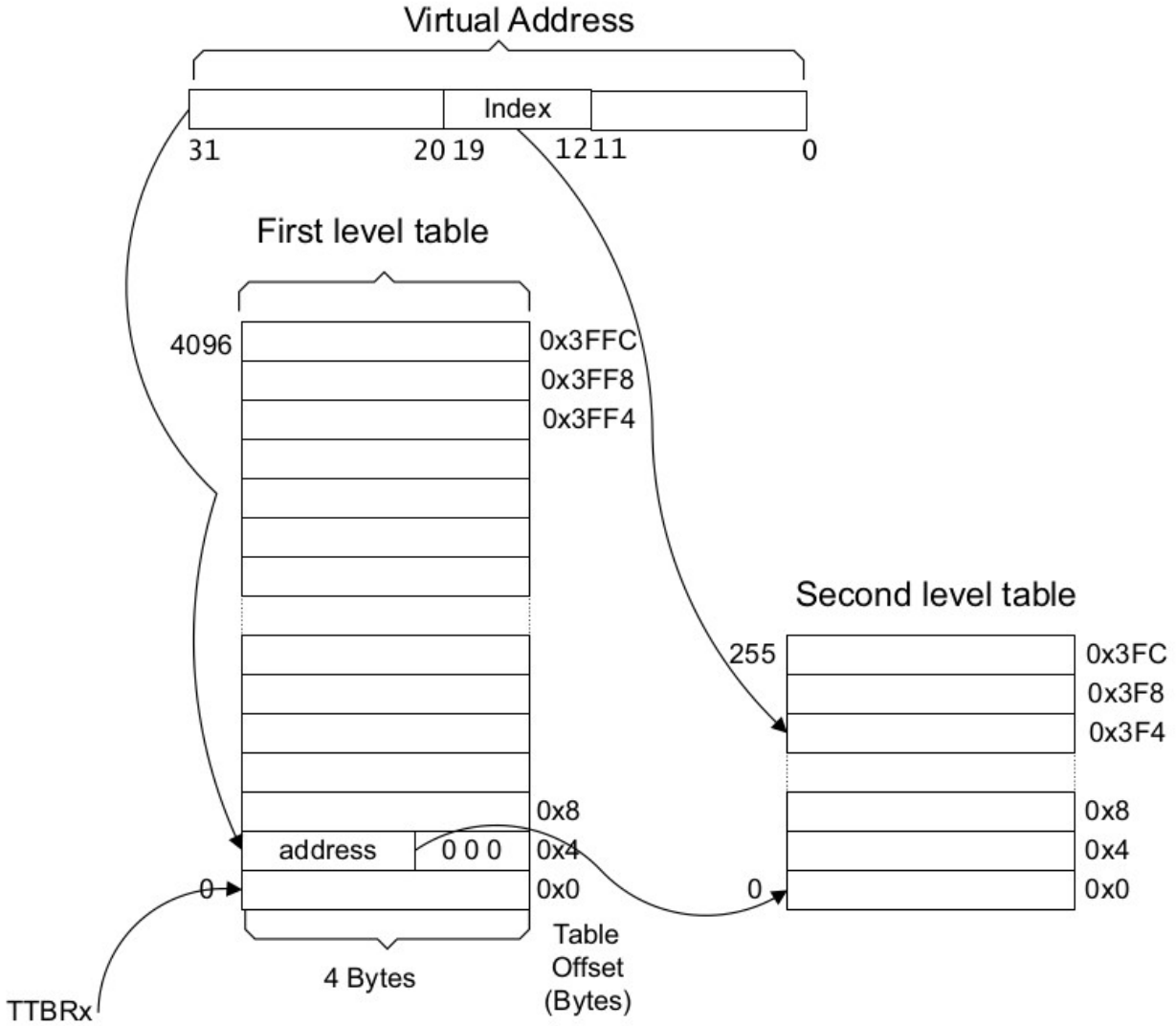


Figure 9-9 Generating the address of the level 2 translation table entry

Транслация на ниво 2

*Взима се базов адрес на L2 таблица от стойност на L1 таблица

*битове [19:12] – указват отместване спрямо базовия адрес. Така се намира 1 конкретна стойност от 256 в L2 таблицата.

Транслация на ниво 2

Обобщение:

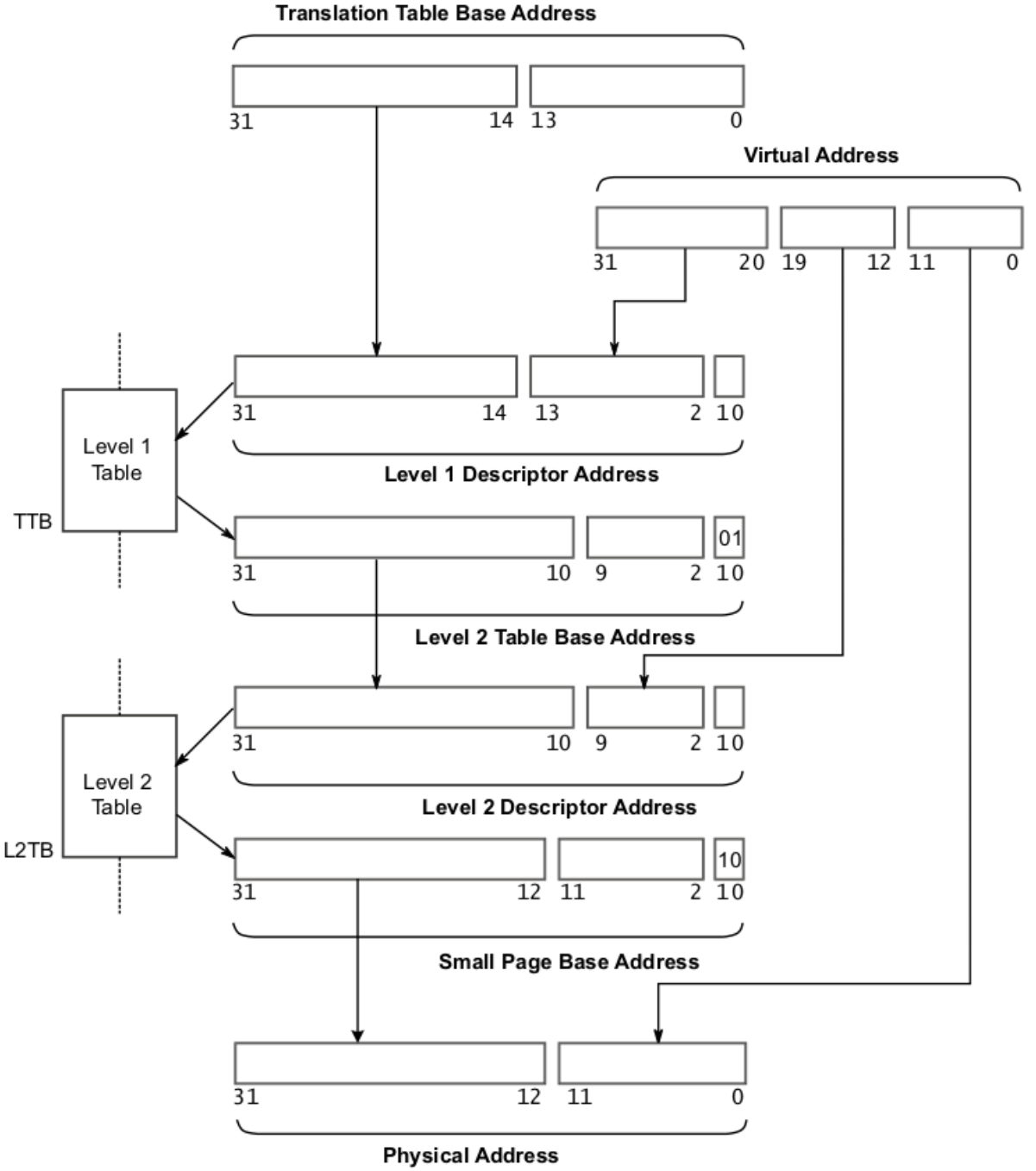


Figure 9-10 Summary of generation of physical address using the L2 translation table entry

Атрибути на паметта

Атрибутите на паметта, които се записват в транслационните таблици са:

*права за достъп (Access Permissions, AP+APX) – битове, указващи права за достъп.

→ Достъп до адреси в таблица, които сочат към данни и достъп, който е забранен ще бъде отказан **(precise abort)**.
→ Достъп до адреси, сочещи към инструкции ще бъдат отказани **(prefetch abort).**

Атрибути на паметта

Информация относно отказа за достъп е налична в копроцесор #15, в следните регистри:

*fault address – адресът, който е опитан да бъде достъпен с грешни права за достъп.

*fault status register – причината за генериране на abort изключение.

Кодът, който трябва да се справи с проблема може да **промени някоя от транслационните таблици**, след което да върне контрола обратно на приложението, което се опитва да достъпи адреса.
Друг вариант е просто **приложението да бъде терминирано**.

Атрибути на паметта

Таблица с правата за достъп, съответстващи на AP+APX битовете е дадена по-долу:

Table 9-1 Summary of Access Permission encodings

APX	AP	Privileged	Unprivileged	Description
0	00	No access	No access	Permission fault
0	01	Read/Write	No access	Privileged Access only
0	10	Read/Write	Read	No user-mode write
0	11	Read/Write	Read/Write	Full access
1	00	-	-	Reserved
1	01	Read	No access	Privileged Read only
1	10	Read	Read	Read only
1	11	-	-	Reserved

Атрибути на паметта

ARM v6 и v7 дефинират 3 вида памети:

***strongly ordered** – не може да бъдат споделяни и кеширани. Всички достъпи до тази памет стават в последователността, която е наредена от компилатора при компилирането на програмата.

***device** – не може да бъдат споделяни и кеширани. Създадена е за периферни устройства, които са поместени в картата на паметта. Достъпите до нея са както са наредени от компилатора.

Атрибути на паметта

***normal** - може да бъдат споделяни/несподеляни (shareable/non-shareable) и кеширани (cacheable).

S+C = използва се за инстр./данни, които се споделят между две или повече ядра.

NonS+C = използва се за инстр./данни, които се използват само от 1 ядро.

Атрибути на паметта

Видовете памет се избират от транслационната таблица посредством TEX, C и B битовете:

Table 9-3 Memory type and cacheable properties encoding in translation table entry

TEX	C	B	Description	Memory type
000	0	0	Strongly-ordered	Strongly-ordered
000	0	1	Shareable device	Device[a]
000	1	0	Outer and Inner write-through, no allocate on write	Normal
000	1	1	Outer and Inner write-back, no allocate on write	Normal
001	0	0	Outer and Inner non-cacheable	Normal
001	-	-	Reserved	-
010	0	0	Non-shareable device	Device[a]
010	-	-	Reserved	-
011	-	-	Reserved	-
1XX	Y	Y	Cached memory XX = Outer policy YY = Inner policy	Normal

a. LPAE treats all device accesses as Shareable

Атрибути на паметта

Двата най-младши бита на TEX битовото поле указват вида на схемата на зареждане на **външните кешове** (най-често L2, L3, L4).

Битовете C и B указват схемата на зареждане на **вътрешните кешове** (най-често L1, L2).

Пример: в ARM Cortex-A8 и Cortex-A15 за вътрешни кешове се приемат L1 и L2.

Атрибути на паметта

Домейни на паметта – 16 броя ID тагове, които може да се присвояват на региони от паметта. Предстои да бъдат премахнати (deprecated), но във v7 все още ги има.

Domain Access Control Register (DACR) – регистър C3 от копроцесор #15 съдържа двубитови стойности за всяко ID. Тези стойности указват права за достъп до съответния домейн:

***no-access** – предизвиква abort изключение при опит за достъп.

***manager mode** – игнорира всички права за достъп на страницата и позволява пълен достъп до нея.

***client mode** – използва правата за достъп на страницата от транслационната таблица.

Многозадачно изпълнение и MMU

*Съпоставянето на виртуални адреси към физически за **инструкциите и данните на една OS е фиксирано** – т.е. стойностите на транслационните таблици не се променят след като веднъж са били инициализирани.

*Това не е така за процесите – **при стартирането** им ядрото на **OS ще зареди** стойности на транслационна таблица, която ще отговаря за инструкциите и данните на този процес.

*Ако процесът използва malloc( ), с което ще **изиска допълнително памет**, OS ядрото ще **промени** първоначалната таблица на процеса.

Многозадачно изпълнение и MMU

*Когато един процес приключи, OS ядрото може да **изтрие транслационните му таблици** и да даде свободното пространство на друг процес.

*При превключване от един процес на друг (context switch), OS ядрото **превключва транслационните таблици**.

*Транслационните таблици на процес, който не се изпълнява в момента, са **защитени от MMU** и активният процес не може да достъпва инструкциите/данните му по никакъв начин.

Многозадачно изпълнение и MMU

***ASID** (Address Space ID) – когато nG = 1, това е 8-битово число, указващо номер на процес, което ще бъде използвано при транслирането на адресите.

ASID се записва в регистър C13 на копроцесор #15.

Така в TLB може да има валидни TLB адреси на една страница, но от различни процеси. Това значително **намалява времето на един context switch**, понеже се избягва TLB flush при такова превключване.

ASID помага за по-лесно дебъгване на ниво процес, когато има операционна система.

Многозадачно изпълнение и MMU

Пример: A, B, C са процеси, използващи виртуално адресно пространство от 0x0000.0000 нагоре. Забележете, че TLB съдържа информация и за трите процеса без това да предизвиква проблем.

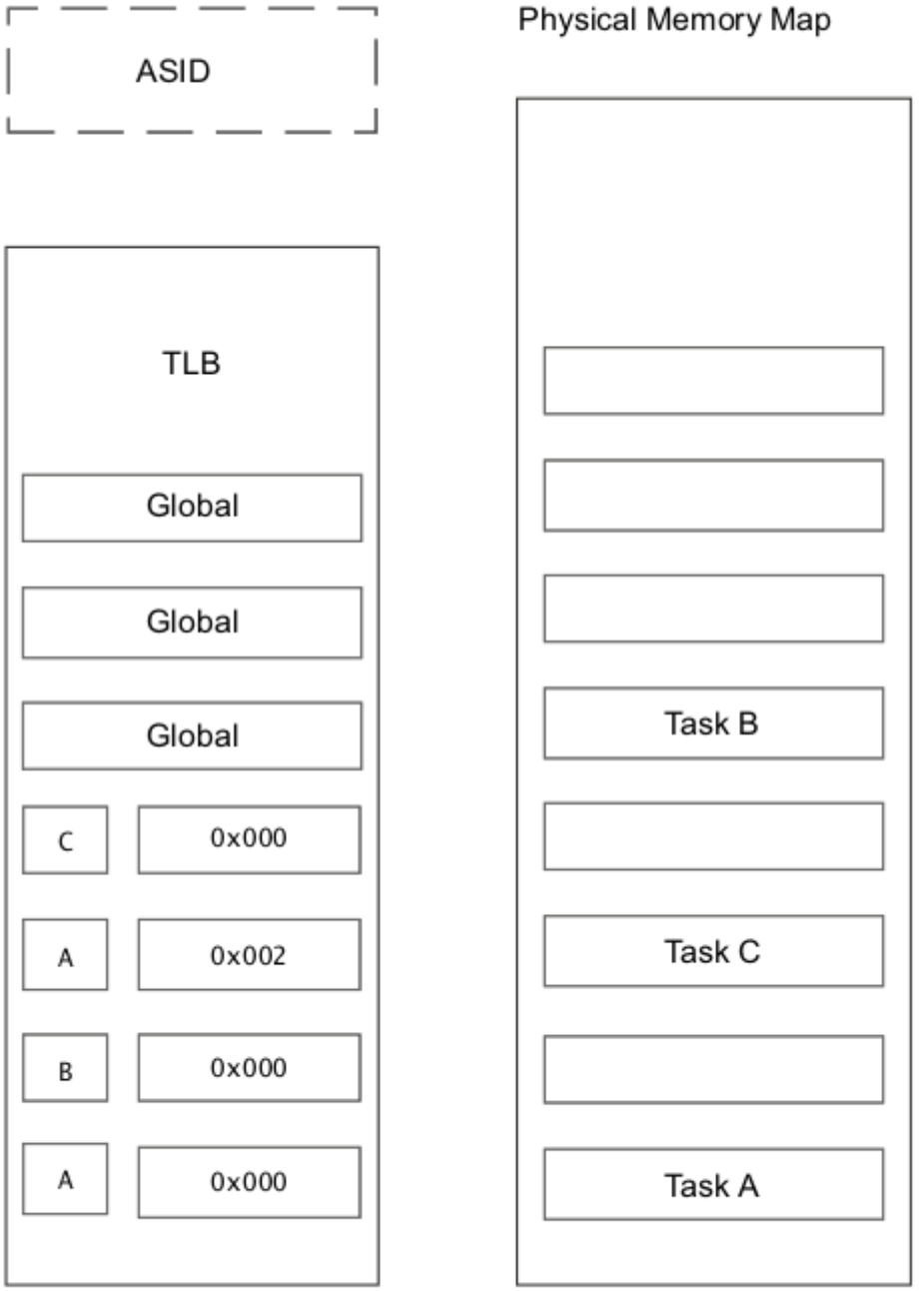


Figure 9-11 ASIDs in TLB mapping the same virtual address

Литература

[1] “ARM Cortex-A Series Programmer’s Guide”, Version 4.0, ARM DEN0013D (ID012214), 2013.
[2] “ARM Developer Suite”, Developer Guide, version 1.2, ARM DUI 0056D, 2001.
[3] “Arm Architecture Registers for A-profile architecture”, ARM DDI 0601, 2024.

Външни връзки

[a] www.kernel.org (linux-6.9-rc7/arch/arm64/include/asm/tlbflush.h