

我在这里翻译了 ARM DDI 0100E 《ARM Architecture Reference Manual》 2<sup>nd</sup> edition 的 Chapter B3 Memory Management Unit 供参考。文中有些地方翻译的不是很妥当，希望大家来函指正。 [gavinux@yahoo.com](mailto:gavinux@yahoo.com)

Copyright (c) ARM Limited. All rights reserved.

## 第三章 存储器管理单元

本文描述基于存储器管理单元的系统结构，包含以下内容：

- 关于存储器管理单元的结构
- 存储器访问的顺序
- 转换过程
- 访问权限
- 域
- 异常
- CP15 寄存器

### 3.1 关于存储器管理单元的结构

MMU 存储器系统的结构允许对存储器系统的精细控制。大部分的控制细节由存在存储器中的转换表提供。这些表的入口定义了从 1KB 到 1MB 的各种存储器区域的属性。这些属性包括：

#### 虚拟地址到物理地址映射

ARM 处理器产生的地址叫虚拟地址，MMU 允许把这个虚拟地址映射到一个不同的物理地址去。这个物理地址表示了被访问的主存储器的位置。

它允许用很多方式管理物理存储器的位置，例如：它可以用具有潜在冲突的地址映射为不同的进程分配存储器，或允许具有不连续地址的应用把它映射到连续的地址空间。

-----注-----

如果使用了快速上下文切换扩展(Fast Context Switch Extension)，则在本文中的虚拟地址的意思应该是修改过的虚拟地址 (Modified virtual address)

-----

#### 存储器访问权限(permissions)

这些控制对存储器区域的不可访问权限、只读权限、读写权限。当访问不可访问权限的存储器时，会有一个存储器异常通知 ARM 处理器。

允许权限的级别也受程序运行在用户状态还是特权状态影响，还受是否使用了域有关。

#### 高速缓存和缓冲位 (Cachability and bufferability bits [C and B])

这些在高速缓存和缓冲一节讲

系统控制协处理器的寄存器允许对系统的高级控制，如转换表的位置。他们也用来为 ARM 提供内存异常的状态信息。

查找整个转换表的过程叫转换表遍历。它由硬件制动进行，并需要大量的执行时间（至少一个存储器访问，通常是两个）。为了减少存储器访问的平均消耗，转换表

遍历结果被高速缓存在一个或多个叫作 Translation Lookaside Buffers(TLBs)的结构中。通常在 ARM 的实现中每个内存接口有一个 TLB。

- 有一个存储器接口的系统通常有一个唯一的 TLB
- 指令和数据的内存接口分开的系统通常有分开的指令 TLB 和数据 TLB

如果系统有高速缓存，高速缓存的数量也通常是由同样的方法确定的。所以在高速缓存的系统中，每个高速缓存一个 TLB。

当存储器中的转换表被改变或选中了不同的转换表(通过写 CP15 的寄存器 2)，先前高速缓存的转换表遍历结果将不再有效。MMU 结构提供了刷新 TLB 的操作。

MMU 结构也允许特定的转换表遍历结果被锁定在一个 TLB 中，这就保证了对相关的存储器区域的访问绝不会导致转换表遍历，这也对那些把指令和数据锁定在高速缓存中的实时代码有相同的好处。

### 3.2 存储器访问的顺序

当 ARM 要访问存储器时，MMU 先查找 TLB 中的虚拟地址表，如果 ARM 的结构支持分开的地址 TLB 和指令 TLB，那么它用：

- 取指令使用指令 TLB
- 其它的所有访问类别用数据 TLB

如果 TLB 中没有虚拟地址的入口，则转换表遍历硬件从存在主存储器中的转换表中获取转换和访问权限，一旦取到，这些信息将被放在 TLB 中，它会放在一个没有使用的入口处或覆盖一个已有的入口。关于转换表的信息和转换表遍历的实现参见转换过程一节。

一旦为存储器访问的 TLB 的入口被拿到，这些信息将被用于：

1. C (高速缓存) 和 B (缓冲) 位被用来控制高速缓存和写缓冲，并决定是否高速缓存。(如果系统中没有高速缓存和写缓冲，则对应的位将被忽略)
2. 访问权限和域位用来控制访问是否被允许。如果不允许，则 MMU 将向 ARM 处理器发送一个存储器异常；否则访问将被允许进行。  
访问权限、域和异常几节有详细描述。

3. 对没有高速缓存的系统（包括在没有高速缓存系统中的所有存储器访问），物理地址将被用作主存储器访问的地址。

对有高速缓存的系统，在高速缓存没有选中的情况下，物理地址将被用于行取 (line fetch) 的地址。如果选中了高速缓存，则物理地址将被忽略。

图 3-1 说明了这种高速缓存系统

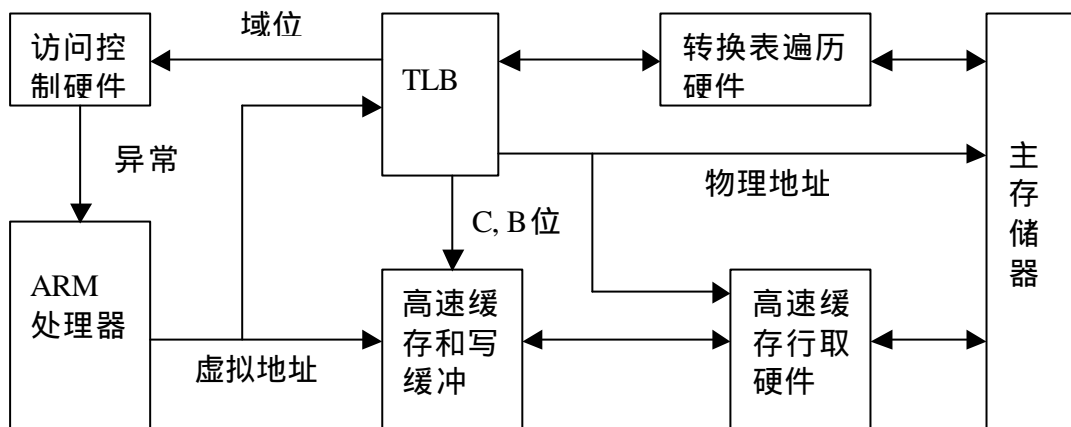


图 3-1 高速缓存的 MMU 存储器系统

### 3.2.1 允许和禁止 MMU

通过写系统控制协处理器的寄存器 1 的第 0 位可以允许和禁止 MMU。在复位后这位是 0，MMU 被禁止。

当 MMU 被禁止时，存储器访问将被按如下处理：

1. 由具体的实现确定当 MMU 被禁止时是否能够允许高速缓存和写缓冲。
  - 当 MMU 被禁止时不能允许高速缓存和写缓冲时，C 和 B 位不起作用。
  - 当 MMU 被禁止时能允许高速缓存和写缓冲时：
    - i. 访问数据时被认为没有高速缓存和写缓冲（C==0，B==0）
    - ii. 取指令时：
      - a) 当系统只有一个唯一的 TLB 时，认为没有高速缓存。（C==0）
      - b) 当系统只有独立的指令 TLB 时，认为有高速缓存。（C==1）
2. 没有存储器访问权限的检查，MMU 也不产生异常信号。
3. 物理地址与虚拟地址相同（即所谓的平坦地址映射模式）。

在允许 MMU 之前，必须在内存中建立适当的转换表，并且所有相关的 CP15 寄存器要被初始化正确。

注：-----

允许和禁止 MMU 直接改变了虚拟地址到物理地址的映射（除非转换表被设定为平坦地址映射模式）。所以很可能在允许 MMU 时所有的高速缓存需要被刷新。

另外，如果允许 MMU 的指令的物理地址和虚拟地址不同，取指令将变得复杂化。所以，强烈建议允许 MMU 的指令具有相同的物理地址和虚拟地址。

-----

### 3.3 转换过程

MMU 支持基于节或页的存储器访问：

节 (Section)      构成 1MB 的存储器块

支持 3 中不同的页尺寸：

微页 (Tiny page) 构成 1KB 的存储器块

小页 (Small page)      构成 4KB 的存储器块

大页 (Large page)      构成 64KB 的存储器块

节和大页是支持允许只用一个 TLB 入口去映射大的存储器区间。小页和大页有附加的访问控制：小页分成 1KB 的子页，和大页分成 16KB 的子页。微页没有子页，对微页的访问控制是对整个页。

存在主存储器内的转换表有两个级别：

第一级表      存储节转换表和指向第二级表的指针。

第二级表      存储大页和小页的转换表。一种类型的第二级表存储微页转换表。

MMU 把 CPU 产生的虚拟地址转换成物理地址去访问外部存储器，同时继承并检查访问权限。地址转换有四条路径。路径的选取由这个地址是被标记成节映射访问还是页映射访问确定。页映射访问可以是、小和微页的访问。

然而，转换过程总是由下面所描述的那样由第一级表的获取开始。节映射的访问只需要读取第一级表，页映射的访问还需要读取第二级表。

#### 3.3.1 转换表基址

当片上 (on-chip) 的 TLB 中不包含被要求的虚拟地址的入口时，转换过程被启动。转换表基址寄存器 (CP15 的寄存器 2) 保存着第一级转换表基址的物理地址。只有 bits[31:14] 有效，bits[13:0] 应该是零 (SBZ)。所以第一级表必须在 16KB 的边界。

#### 3.3.2 取第一级表

转换表基址寄存器的 bits[31:14] 与虚拟地址的 bits[31:20] 和两个 0 位连接形成 32 为物理地址，如图 3-2。这个地址选择了一个四字节转换表入口，它是第一级描述符或是指向第二级页表的指针。

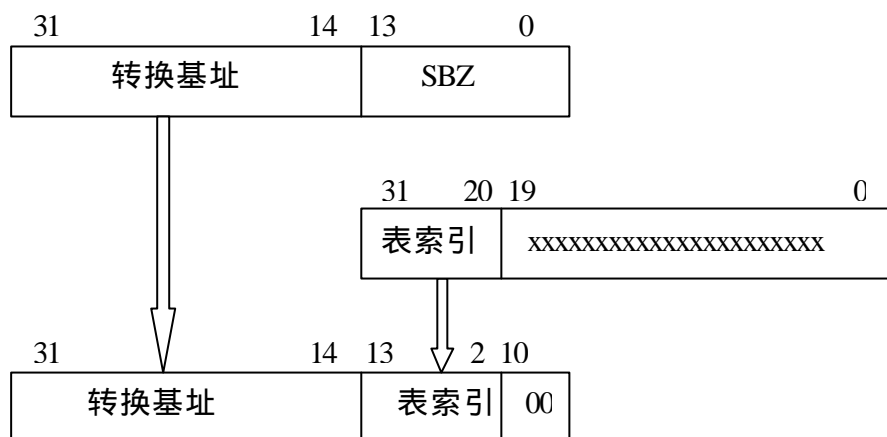


图 3-2 访问转换表的第一级描述符



### 3.3.3 第一级描述符

第一级表的每个入口是一个描述它所关联的 1MB 虚拟地址是如何映射的描述符。见表 3-1，根据 bits[1:0] 的组合，有四种可能：

- 如果 bits[1:0]==0b00，所关联的地址没有被映射，试图访问他们将产生一个转换错（fault）。因为他们被硬件忽略，所以软件可以利用这样的描述符的 bits[31:2] 做自己的用途。推荐为描述符继续保持正确的访问权限。
- 如果 bits[1:0]==0b10，这个入口是它所关联地址的节描述符。见节描述符和转换节参考中的细节。
- 如果 bits[0]==1，这个入口给出粗糙第二级表（bit[1]==0），或精细第二级表（bit[1]==1）。每一种类型的表描述了它所关联的 1MB 存储区域的映射。粗糙第二级表较小，每个表 1KB，每个精细第二级表 4KB。然而粗糙第二级表只能映射大页和小页，精细第二级表可以映射大页、小页和微页。

表 3-1 第一级描述符格式

	31	20	19	12	11	10	9	8	5	4	3	2	10
错	忽略												00
粗糙页表	粗糙页表基址							sbz	域	imp			00
节	节基址			SBZ		AP	sbz	域	imp	C	B	10	
精细页表	精细页表基址					SBZ		域	imp			11	

### 3.3.4 节描述符和转换节参考

如果第一级描述符是节描述符，那么各个字段有如下的意义：

- Bits[1:0] 描述符类型标识（0b10 表示节描述符）  
Bits[3:2] 高速缓存和缓冲位  
Bits[4] 由具体实现定义  
Bits[8:5] 这个描述符控制的节的 16 种域之一  
Bits[9] 现在没有使用，应该为零  
Bits[11:10] 访问控制，见表 3-3  
Bits[19:12] 现在没有使用，应该为零  
Bits[31:20] 节基址，形成物理地址的高 12 位

图 3-3 表示了节转换的完整过程。

注：-----

访问权限必须在物理地址产生之前去检查，检查访问权限的顺序见访问权限一节。

-----

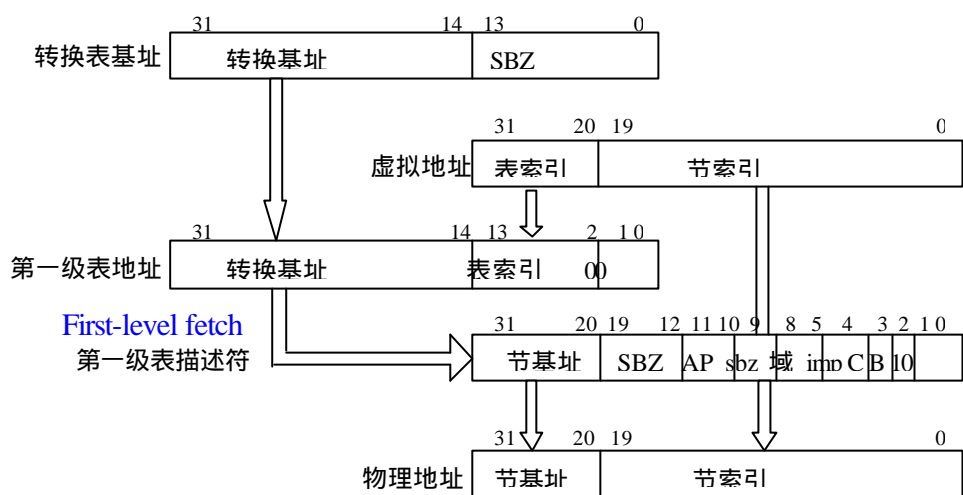


图 3-3 节转换

### 3.3.5 粗糙页表描述符

如果第一级描述符是粗糙页表描述符，那么各个字段有如下的意义：

- Bits[1:0] 描述符类型标识（0b01 表示粗糙页表描述符）
- Bits[4:2] 由具体实现定义
- Bits[8:5] 这个描述符控制的页的 16 种域之一
- Bits[9] 现在没有使用，应该为零
- Bits[31:10] 页表基地址是一个指向第二级粗糙页表的指针，它给出第二级表访问的基地址。而第二级粗糙页表必须在 1KB 边界对齐。

如果从第一级读取到的是二级粗糙页表描述符，那么会象图 3-4 所示执行第二级描述符读取。

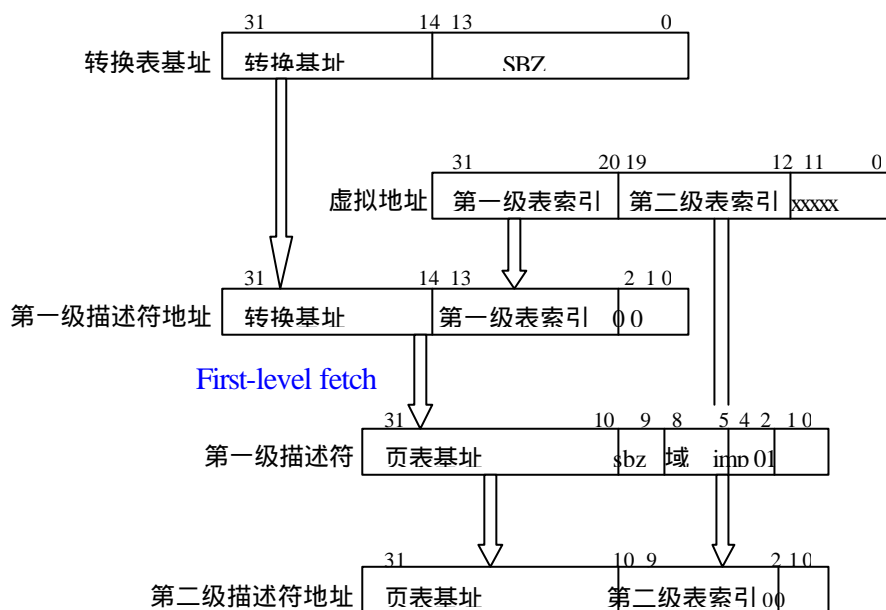


图 3-4 访问粗糙页表第二级描述符

### 3.3.6 精细页表描述符

如果第一级描述符是精细页表描述符，那么各个字段有如下的意义：

- Bits[1:0] 描述符类型标识（0b11 表示精细页表描述符）
- Bits[4:2] 由具体实现定义
- Bits[8:5] 这个描述符控制的页的 16 种域之一
- Bits[11:9] 现在没有使用，应该为零
- Bits[31:10] 页表基地址是一个指向第二级精细页表的指针，它给出第二级表访问的基地址。而第二级精细页表必须在 4KB 边界对齐。

如果从第一级读取到的是二级精细页表描述符，那么会象图 3-5 所示执行第二级描述符读取。

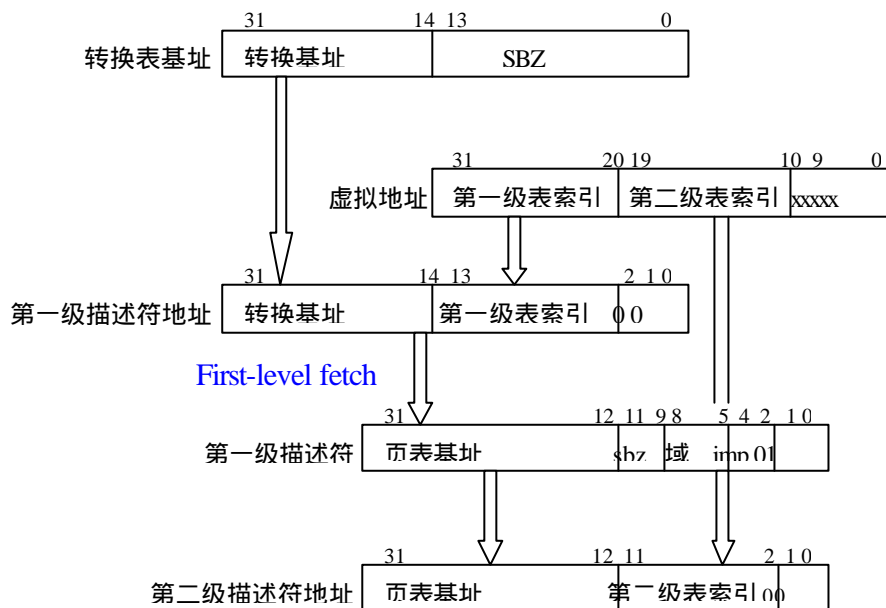


图 3-5 访问精细页表第二级描述符

### 3.3.7 第二级描述符

每个粗糙第二级表对映着以 4KB 为单位的虚拟地址范围市怎么映射的，每个精细第二级表对映着以 1KB 为单位的虚拟地址范围市怎么映射的。那些入口是页描述符，他们能够分别描述大于 4KB 或 1KB 的页。在这种情况下，这个描述符必须被重复足够次，以保证这个页始终使用相同的描述符，不论访问这个页中的哪个虚拟地址。

对于一个第二级描述符，有四种可能，由描述符的 bits[1:0]选择。见表 3-2：

- 如果 bits[1:0]==0b00，说关联的虚拟地址没有被映射，任何对这些虚拟地址的访问将会导致转换错(fault)。软件可以利用这样的描述符的 bits[31:2]做自己的用途，因为他们被硬件忽略。推荐为描述符继续保持正确的访问权限。
- 如果 bits[1:0]==0b01，这个入口是大页描述符，描述 64KB 的虚拟地址。见转换大页参考。

一个大页描述符在精细第二级表中必须被重复 64 次，在粗糙第二级表中必须被重复 16 次以保证所有的虚拟地址都被描述。

- 如果 bits[1:0]== 0b10，这个入口是小页描述符，描述 4KB 的虚拟地址。见转换小页参考。

一个小页描述符在精细第二级表中必须被重复 4 次，以保证所有的虚拟地址都被描述。在粗糙第二级表中只有一个实例。

- 如果 bits[1:0]== 0b11，这个入口是微页描述符，描述 1KB 的虚拟地址。见转换微页参考。

在精细第二级表中只需要一个微页描述符的实例。微页描述符不能在粗糙第二级表中出现，如果出现了，结果不可预测。

表 3-2 第二级描述符格式

	31	16	15	12	11	10	9	8	7	6	5	4	3	2	1	0	
错	忽略															00	
大页	大页基地址										SBZ	AP3	AP2	AP1	AP0	C	B 01
小页	小页基地址											AP3	AP2	AP1	AP0	C	B 01
微页	微页基地址										SBZ	AP				C	B 11

## 大页描述符字段

大页描述符的字段有如下意义：

bits[1:0] 表示描述符的类型  
bits[3:2] 高速缓促和缓冲位  
bits[11:4] 访问权限位。这些为控制对页的访问。关于这些位的解释见表 3-3。  
大页被分成 4 各子页。  
AP0 编码对第一个子页的访问权限。  
AP1 编码对第二个子页的访问权限。  
AP2 编码对第三个子页的访问权限。  
AP3 编码对第四个子页的访问权限。

bits[15:12] 现在没有使用，应该为零。  
bits[31:16] 用来形成物理地址的对应位。

## 小页描述符字段

小页描述符的字段有如下意义：

bits[1:0] 表示描述符的类型  
bits[3:2] 高速缓促和缓冲位  
bits[11:4] 访问权限位。这些为控制对页的访问。关于这些位的解释见表 3-3。  
小页被分成 4 各子页。  
AP0 编码对第一个子页的访问权限。  
AP1 编码对第二个子页的访问权限。  
AP2 编码对第三个子页的访问权限。  
AP3 编码对第四个子页的访问权限。

bits[31:12] 用来形成物理地址的对应位。

## 微页描述符字段

微页描述符的字段有如下意义：

bits[1:0] 表示描述符的类型  
bits[3:2] 高速缓促和缓冲位  
bits[5:4] 访问权限位。这些为控制对页的访问。关于这些位的解释见表 3-3 关于微页的解释。  
bits[9:6] 现在没有使用，应该为零。  
bits[31:10] 用来形成物理地址的对应位。

### 3.3.8 转换大页参考

图 3-6 显示了在粗糙第二级表中转换一个 64KB 的大页的完整顺序。在精细第二级表中的转换顺序页相似，只是第二级描述符的地址如精细页表描述符一节所决定。

注：-----

页索引的高 4 位和第二级表的低阶 4 位重叠，在粗糙页表中大页的每个页表入口必须被重复 16 次。在精细页表中大页的每个页表入口必须被重复 64 次。

-----

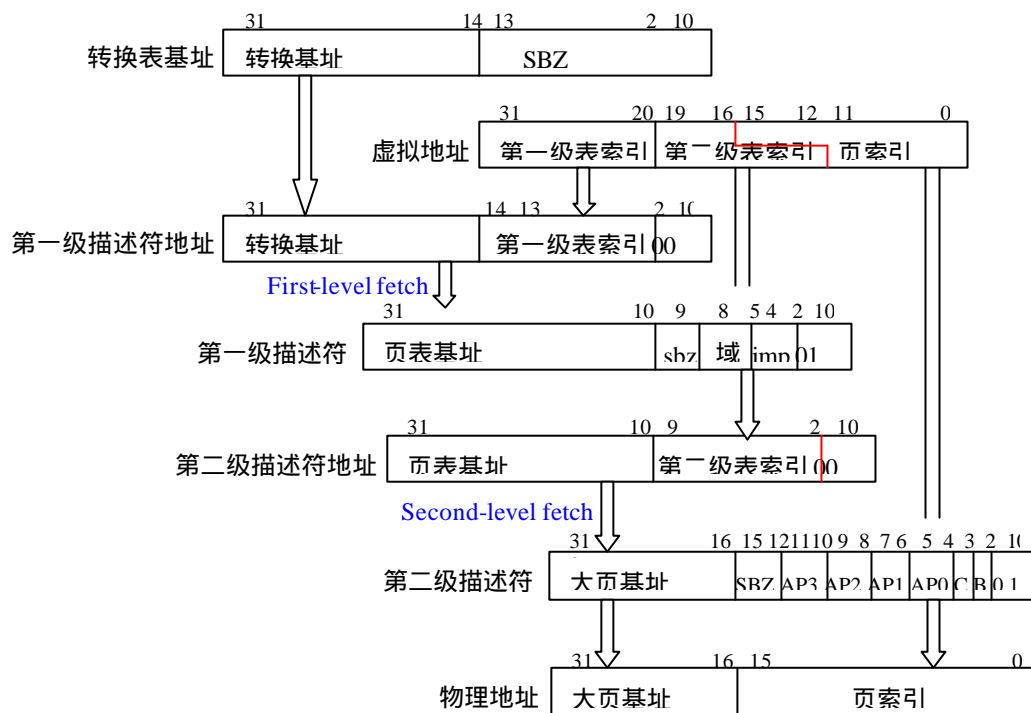


图 3-6 粗糙第二级表中的大页转换

### 3.3.9 转换小页参考

图 3-7 显示了在粗糙第二级表中转换一个 4KB 的小页的完整顺序。在精细第二级表中的转换顺序页相似，只是第二级描述符的地址如精细页表描述符一节所决定。

注：-----

当小页出现在精细第二级表中时，页索引的高 2 位和第二级表的低阶 2 位重叠，在精细页表中小页的每个页表入口必须被重复 4 次。

-----

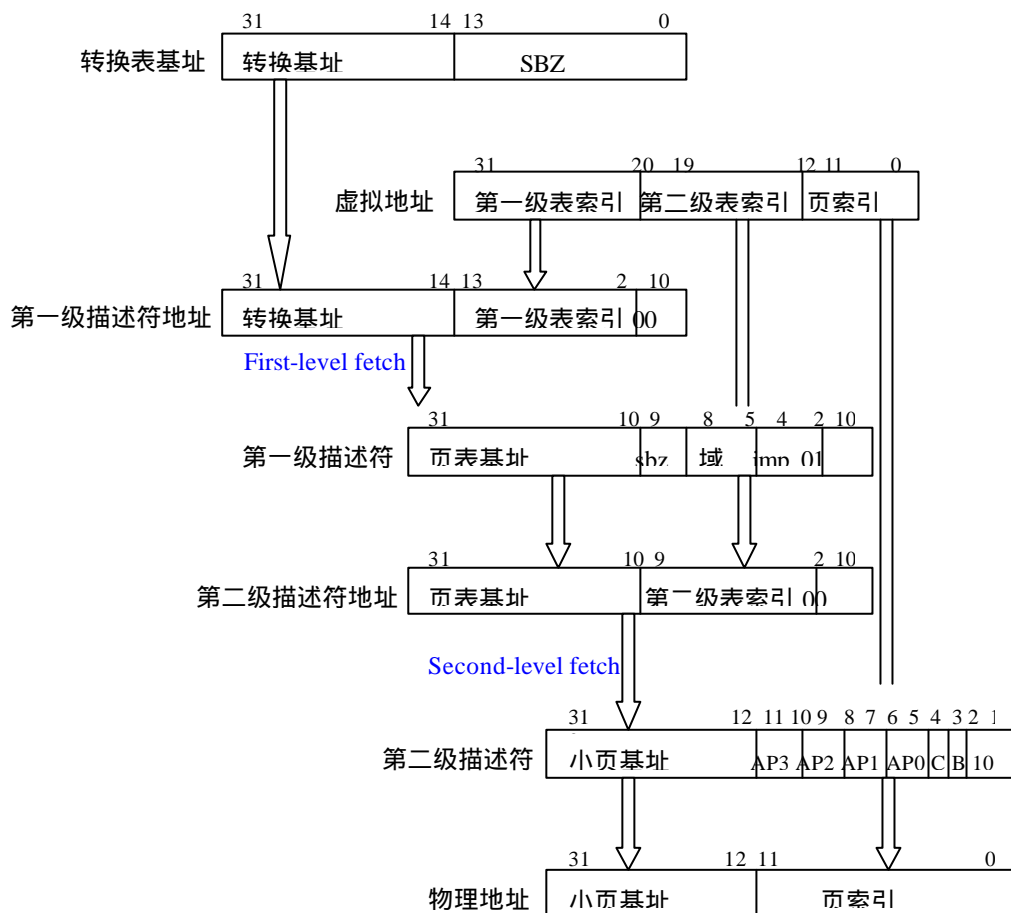


图 3-7 粗糙第二级表中的小页转换



### 3.3.10 转换微页索引

图 3-8 显示了在精细第二级表中转换 1KB 微页的完整过程。

注：-----  
微页不能出现在粗糙第二级表中。  
-----

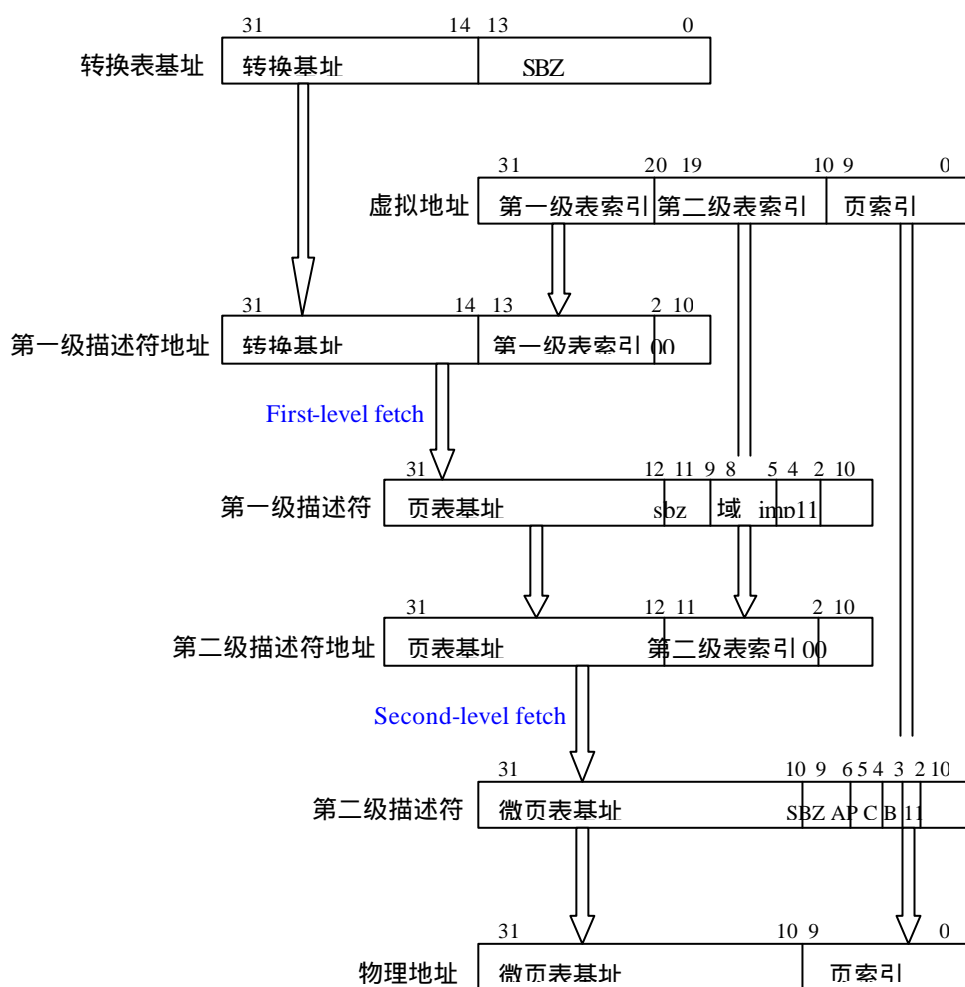


图 3-8 精细第二级表中的微页转换

### 3.4 访问权限

在节和页描述符中的访问权限位控制对相应的节和页的访问。访问权限由 CP15 的寄存器 1 的 System(S)和 ROM(R)位修改。表 3-3 描述了访问权限位和 S、R 位相互作用时的意义。如果访问了没有访问权限的存储器空间，将会产生权限错（见异常一节）。

表 3-3 MMU 访问权限

AP	S	R	Privileged permissions	User permissions
0b00	0	0	不能访问	不能访问
0b00	1	0	只读	不能访问
0b00	0	1	只读	只读
0b00	1	1	不可预测	不可预测
0b01	X	X	读 / 写	不能访问
0b10	X	X	读 / 写	只读
0b11	X	X	读 / 写	读 / 写

### 3.5 域

域是节、大页和小页的集合。ARM 结构支持 16 个域。对域的访问由域访问控制寄存器的两个位字段控制。因为每个字段对访问对应的域的使能非常迅速，所以整个存储器区间能很快地交换进出虚拟存储器。这里支持 2 种域访问方式：

**客户**                域的用户（执行程序，访问数据），被形成这个域的节或页来监督访问权限。

**管理者**            控制域的行为（域中的当前节和页，对域的访问），不被形成这个域的节或页来监督访问权限。

一个程序可以是一些域的客户，也是另外一些域的管理者，同时没有对其它域的访问权限。这允许对程序访问不同存储器资源的非常灵活的存储器保护。表 3-4 说明了域访问控制寄存器的位编码方式。

表 3-4 域访问的值

值	访问方式	描述
0b00	不能访问	任何访问都将导致一个域错(domain fault)
0b01	客户	能否访问将根据节或页描述符中的访问权限位确定
0b10	保留	使用这个值将导致不可预料的结果
0b11	管理者	不根据节或页描述符中的访问权限位确定能否访问，所以不会产生权限错。(permission fault)

## 3.6 异常

由于存储器访问的约束而导致 ARM 处理器停止执行的机制有：

MMU fault   MMU 检测到约束并统治处理器  
外部 Abort   外部存储器系统发出一个非法存储器访问信号

MMU fault 和 外部 Abort 都叫异常(ABORT)。

如果导致异常的存储器操作是取指令，那么当处理器去执行这个非法访问得到的指令时产生一个预取指令异常。如果导致异常的存储器操作是存取数据，那么产生一个数据异常。参见 2.6 节。

### 3.6.1 MMU fault

MMU 产生 4 种类型的 fault

- 对齐错
- 转换错
- 域错
- 权限错

存储器系统能终止 3 种类型的访问

- line fetch
- 存储器访问（没有高速缓存和没有缓冲的访问）
- 转换表访问

MMU 检测到的异常在任何外部存储器访问发生之前被停止。终止导致外部异常的外部访问是外部系统的职责。

系统控制协处理器有 2 个寄存器，当外部访问被终止时更新。指令预取异常不更新这些寄存器，由于程序流程的改变，产生遗产的指令没有被执行。

**错误地址寄存器(FAR)和错误状态寄存器(FSR)**

数据异常会被 CPU 立即响应。FSR 被一个 4 位的错误状态(FS[3:0])和域序号更新。虚拟地址被写入 FAR。如果数据访问连续地产生多个数据异常，则它们的优先级由表 3-5 决定。

由取指令产生的异常在指令进入指令流水线时被标记。只有当指令执行时产生预取异常。一个异常导致取指令没有被执行如果那条指令不被使用（例如跳转到别处）

通常当进入预取指令异常服务程序时，fault 所关联的地址被放在 R14\_abt 寄存器中。由具体实现决定当预取指令异常产生时是否更新 FSR 和 FAR。但是预取指令异常在数据异常产生时（更新 FAR 和 FSR）和进入数据异常项量入口时绝不更新 FSR 和 FAR。也就是说数据异常可以使用 FAR 和 FSR 的值，它们不会被预取指令异常给破坏。

表 3-5 fault 状态的优先级编码

优先级	源	FSR[3:0]	Domain[3:0]	FAR
最高	Terminal Exception	0b0010	Invalid	Implementation Defined
	Vector Exception	0b0000	Invalid	Valid
	Alignment	0b00x1	Invalid	Valid
	External Abort Translation: 1 <sup>st</sup> level 2 <sup>nd</sup> level	0b1100 0b1110	Invalid Valid	Valid Valid
	Translation: Section Page	0b0101 0b0111	Invalid Valid	Valid Valid
	Domain: Section Page	0b1001 0b1011	Valid Valid	Valid Valid
	Permission: Section Page External Abort on Linefetch: Section Page	0b1101 0b1111  0b0100 0b0110	Valid Valid  Valid Valid	Valid Valid  Valid Valid
最低	External Abort on Non_lineFetch: Section Page	 0b1000 0b1010	 Valid Valid	 Valid Valid

---注-----

Alignment 能写 0b0001 或 0b0011 到 FSR[3:0]。

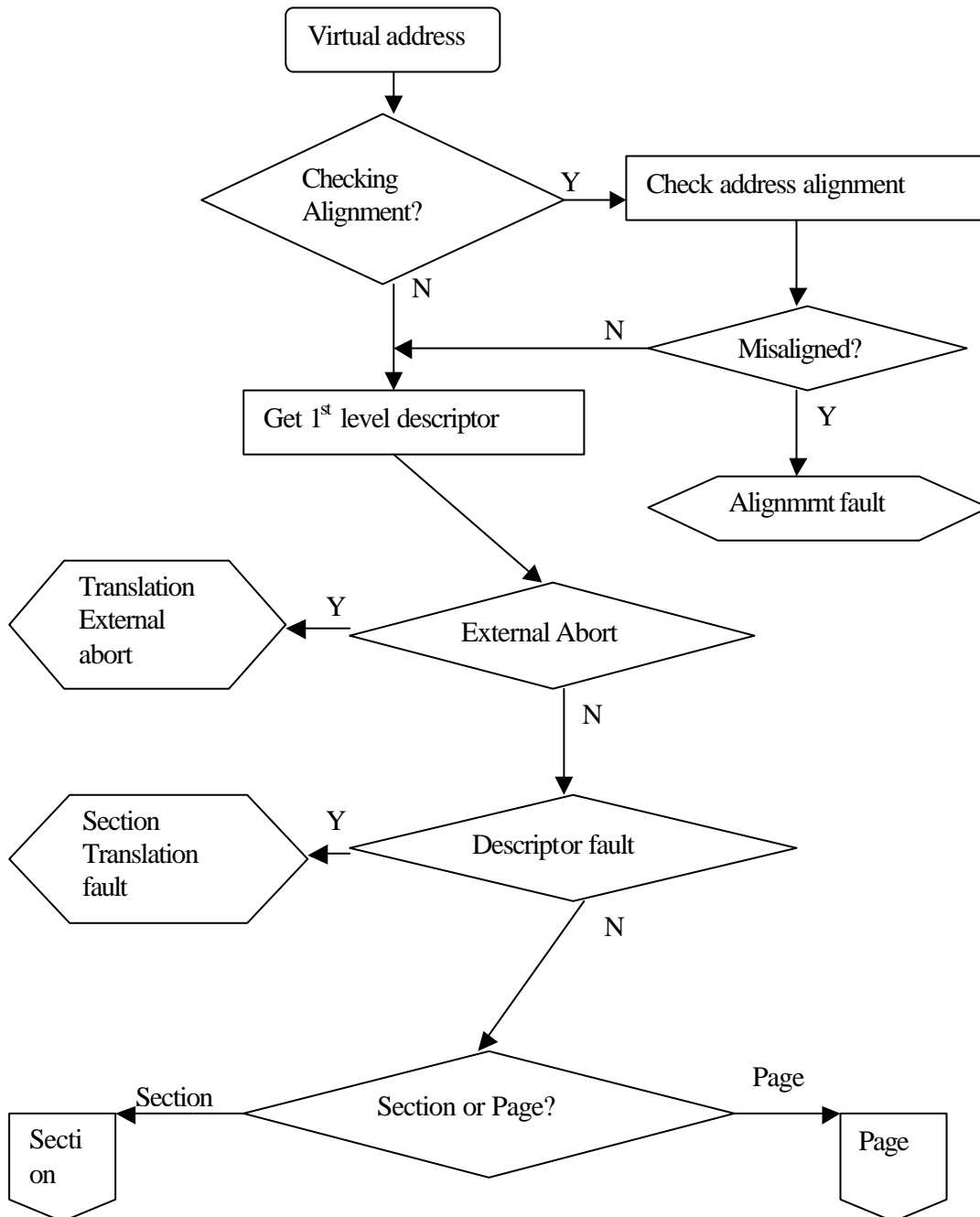
在域(Domain[3:0])中的非法值是因为 fault 发生在有效的域被装入之前。

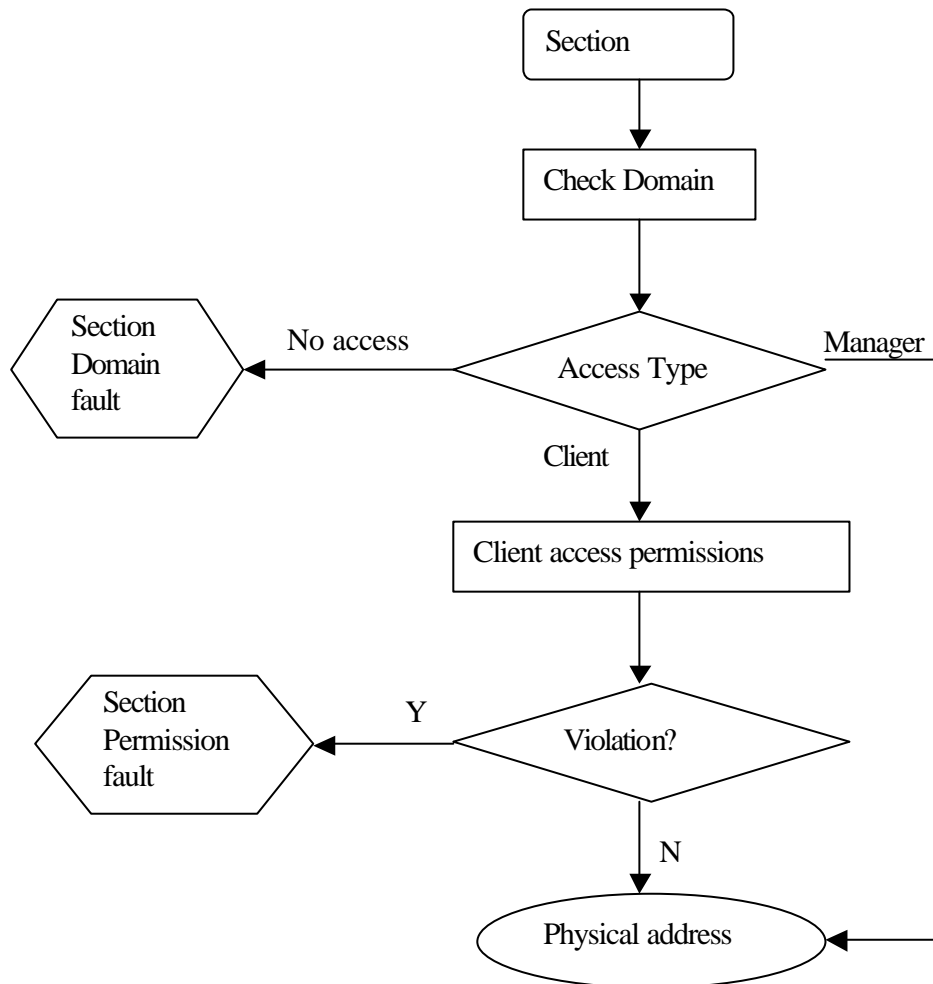
任何被优先级编码屏蔽的异常能够由修改原来的异常 (by fixing the primary abort) 并重新执行那条指令产生。

Vector Exception 打破了 FS[0]=0 标致着外部异常的模式。

错误检测顺序 (Fault-checking sequence)

MMU 检测访问错对节 (Section) 和页 (Page) 有些不同, 见图 3-9





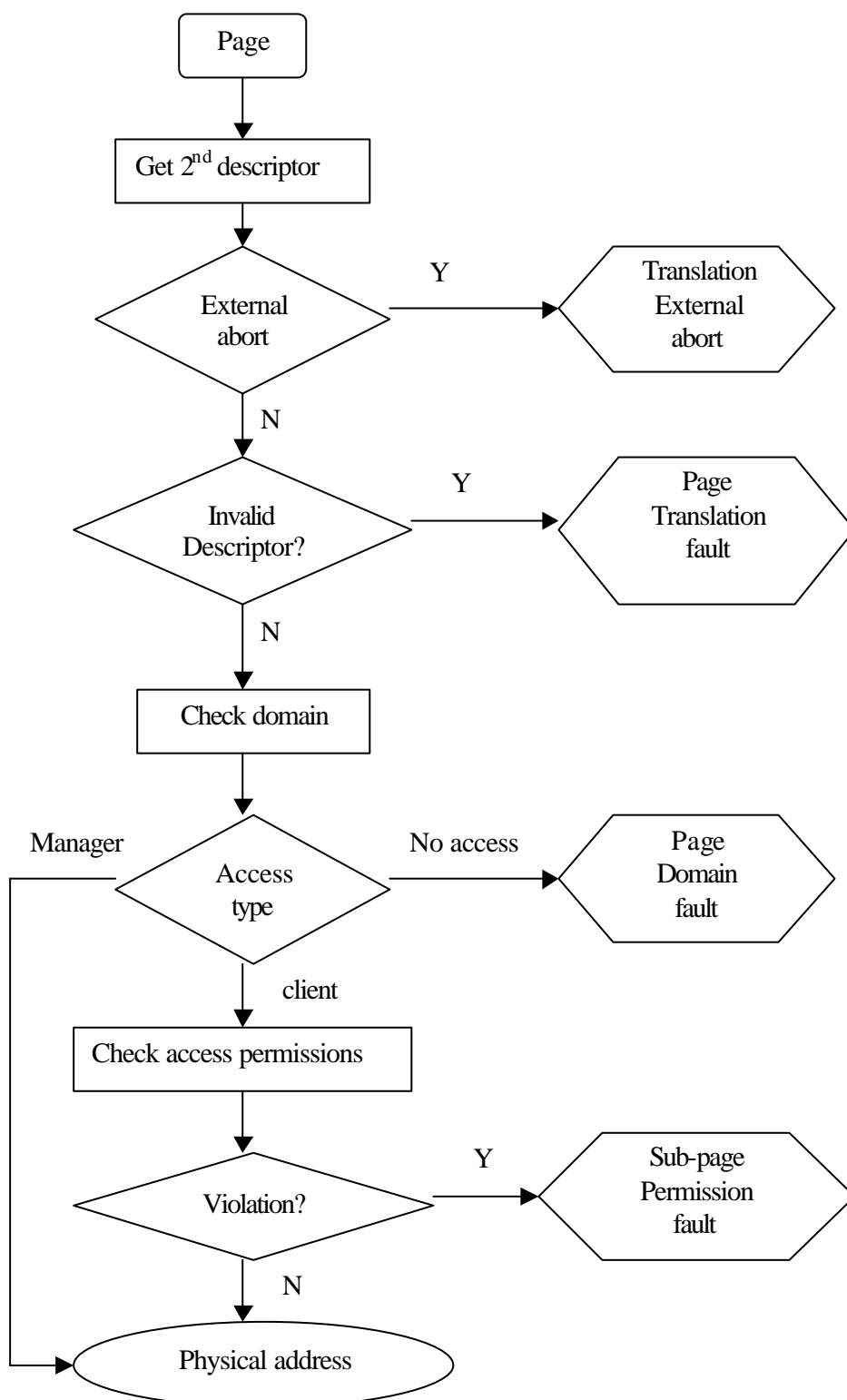


图 3-9 错位检查顺序



## Terminal exception 终端错

这标志着发生一个不可挽回的错误。这在（如果至少）是具体实现决定的情况下发生。

## Vector exception 向量错

当处理器是 32 位配置（PROG32 被激活），在 26 位模式（CPSR[4]=0），数据访问（不是取指令）通常的向量表（地址 0x00-0x1F）导致一个数据异常，这叫向量错。如果向量错发生在 MMU 被禁止时，由具体实现决定如何实现。详细信息见 A8-11 页。

## Alignment fault 对齐错

如果对齐错被允许，则当访问一个不在字对齐地址的数据字（虚拟地址位 bits[1:0] != 0b00）或访问一个不在半字对齐地址的数据半字（虚拟地址位 bits[0] != 0）时产生对齐错。取指令和字节访问不产生对齐错。

---注-----

如果产生对齐错，访问将被终止，而不去进行权限检查。如果对齐错发生在 MMU 被禁止时，由具体实现决定如何实现。

-----

## Translation fault 转换错

有两类转换错：

- 节 当第一级描述符被标志为无效时产生。这时候描述符的 bits[1:0] = 0b00。
- 页 当第二级描述符被标志为无效时产生。这时候描述符的 bits[1:0] = 0b00。

## Domain fault 域错

有两类域错：

- 节
- 页

在这两种情况下，第一级描述符中有 4 位域字段用来选择 16 个域中的一个。域由域访问控制寄存器中的 2 个位控制，这 2 个位用来检查访问权限。见表 3-4。

如果是节，域访问的检查在第一级描述符返回时进行。

如果是页，域访问的检查在第二级描述符返回时进行。

如果特定的访问在域访问控制寄存器中被标识为不可访问，则回产生一个节域错或页域错。

## Permission fault 权限错

有节权限错和子页权限错。

权限错与域错的检查同时进行。如果那 2 位的域字段返回的是客户(client)(01)，权限检查如下进行：

节 如果第一级描述符定义了一个节访问，那么描述符的 AP 位定义了访问是否允许，见表 3-3。如果访问不允许，那将产生节权限错。

子页 如果第一级描述符定义了一个页映射的访问，那么第二级描述符可以定义一个微页，小页或大页。如果第二级描述符定义了一个微页，那它只有一个 AP 位，这个位适用于整个页。否则第二级描述符有 4 个 AP 位（AP3、AP2、AP1 和 AP0），每一个对应 1/4 页。

小页：AP3 选择顶上的 1KB，AP0 选择底下的 1KB。大页：AP3 选择顶上的 16KB，AP0 选择底下的 16KB。被选择的 AP 位与节完全一样地解释，（见表 3-3），只是产生的是子页错。

### 3.6.2 外部异常

除了 MMU 错，ARM 结构还定义了一个外部异常管脚，可以用来标识访问外部存储器错误。然而不是所有的访问都能用这种方式终止，所以这个脚必须被仔细使用。以下的访问可以被外部终止并安全地重启动：

- 读
- 无缓冲的写
- 取第一级描述符
- 取第二级描述符
- 在没有高速缓存和没有缓冲的存储器区域的信号(Semaphore)

行取能够在行传送的任何一个字上安全地终止。如果异常发生在处理器请求的数据上（而不是在 cache line 上取的其它数据），访问被终止。任何不是立刻访问的数据传送（在 cache line 上取的其它数据），只在访问它们时产生异常。

由具体实现确定 FAR 寄存器是指向 cache line 的起始地址还是包含产生异常的地址。

缓冲的写不能被外部终止。所以系统必须配置成不向能够标识外部异常的存储器区域进行缓冲的写，或者用不同的机制去标志异常（例如中断）。

产生异常后，那个存储器空间的值是不确定的。

### 3.7 CP15 寄存器

MMU 由系统控制寄存器的 2、3、4、5、6、8、10 号寄存器和 1 号寄存器的一些位控制。

#### 3.7.1 寄存器 1 的 MMU 控制位

寄存器 1 的如下这些位用来控制 MMU：

M(bit[0]) 使能 MMU

0 = 禁止 MMU

1 = 允许 MMU

在没有 MMU 和保护单元的系统上，这个位应该读出为 0，并忽略写。

A(bit[1]) 使能对齐错检查

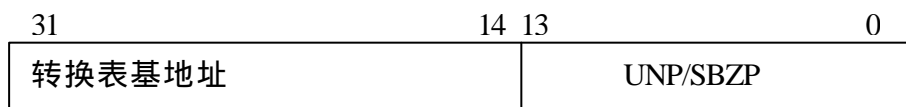
0 = 禁止

1 = 允许

S(bit[8]) 这是系统保护位，见 3-4 节。

R(bit[9]) 这是 ROM 保护位，见 3-4 节。

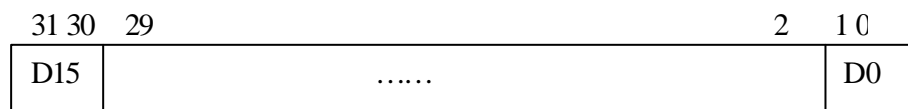
#### 3.7.2 寄存器 2: 转换表基地址



读 CP15 寄存器 2 时，在 bits[31:14] 返回当前活动的第一级转换表的物理地址，bits[13:0] 不确定。读 CP15 寄存器 2 时，CRm 和操作数 2 被忽略，并应该是 0。

写 CP15 寄存器 2 时，在 bits[31:14] 更新当前活动的第一级转换表的物理地址，bits[13:0] 应该写 0 或先前读回的值。写 CP15 寄存器 2 时，CRm 和操作数 2 被忽略，并应该是 0。

#### 3.7.3 寄存器 3: 域访问控制



读 CP15 寄存器 3 时，返回域访问控制寄存器的值。CRm 和操作数 2 被忽略，并应该是 0。

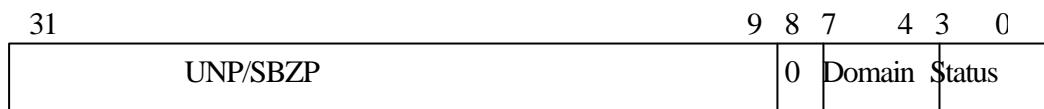
写 CP15 寄存器 3 时，更新域访问控制寄存器的值。CRm 和操作数 2 被忽略，并应该是 0。

域访问控制寄存器包含 16 个 2 位的字段，它定义了对应域的访问权限。见 3-5 节。

#### 3.7.4 寄存器 4: 保留

读写 CP15 寄存器 4 不可预料结果。

#### 3.7.5 寄存器 5: 错误状态 FSR



读 CP15 寄存器 5 时，返回 FSR 寄存器的值。FSR 包含最近一次数据错的信息。只有低 9 位有效，高 23 位不确定。FSR 指出异常发生时的域和试图访问的类型。

bit[8]        返回 0

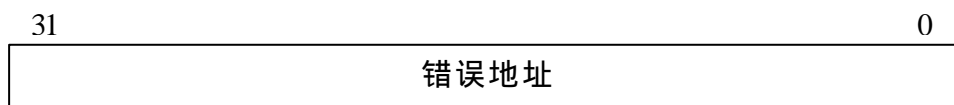
bit[7:4]     指出错位发生时访问的域

bit[3:0]     试图访问的类型，这些位的编码见表 3-5

FSR 在数据错时更新。由具体实现确定取指令异常是否更新 FSR。见 3.6.1 节的“错误地址寄存器 (FAR) 和错误状态寄存器 (FSR)”。CRm 和操作数 2 被忽略，并应该是 0。

写 FSR 将把 FSR 设定成写的值。这对于程序调试器非常有用，可以用来恢复 FSR 的值。高 24 位应该写 0 或上次读到的值。CRm 和操作数 2 被忽略，并应该是 0。

#### 3.7.6 寄存器 6: 错误地址 FAR



读 CP15 寄存器 6 返回 FAR 的值。FAR 保存着错误产生时访问的虚拟地址。在数据错时更新 FAR。由具体实现确定取指令异常是否更新 FSR。见 3.6.1 节的“错误地址寄存器 (FAR) 和错误状态寄存器 (FSR)”。CRm 和操作数 2 被忽略，并应该是 0。

写 FSR 将把 FAR 设定成写的值。这对于程序调试器非常有用，可以用来恢复 FAR 的值。高 24 位应该写 0 或上次读到的值。CRm 和操作数 2 被忽略，并应该是 0。

---注-----

如果使用了第六章描述的快速上下文切换扩展 (FCSE)，那么：

- 当存储器错更新 FAR 时，写入 FAR 的是修改的虚拟地址。
- 当用 MRC 指令读或用 MCR 指令写 FAR 时，它的值被当作数据对待，所以没有由 FCSE 产生的地址修改。

-----

### 3.7.7 寄存器 8:TLB 功能

当 CP15 的寄存器 8 用来控制 TLB 时是只读寄存器。表 3-6 显示了定义的 TLB 功能和在 MCR 指令中用的 CRm 和第二个朝操作数<opcode2>的值。使用没有在表中的 CRm 和 opcode2 的组合将导致不可预料的结果。

如果下面的任何操作被用在单一 TLB 的实现中，则在单一 TLB 中实现相同的功能：

- 无效的指令 TLB (Invalidate instruction TLB)
- 无效的指令单一入口 (Invalidate instruction single entry)
- 无效的整个数据 TLB (Invalidate entire data TLB)
- 无效的数据单一入口 (Invalidate data single entry)

否则，如果执行一个与特定实现不相关的功能，会导致不确定的结果。

试图用 MRC 指令读 CP15 寄存器 8 的结果不确定。

当只有很少量的存储器被重新映射时，无效的单一入口操作能被用来在一些实现中改善性能。对每个被重新映射的存储器区域（节、小页或大页），无效的单一入口需要在存储器区域的虚拟地址上执行。性能的改善来源于不用重新装载与没有被重新映射的存储器区域相关的 TLB 入口。

---注-----

无效的单一入口操作的性能改善并不被保证。具体实现可以是单一请求入口无效，直到使整个 TLB 无效。

-----

---小心-----

当存储器被重新映射时必须使与旧的映射相关的 TLB 入口无效。如果不这样，可能会进入两个 TLB 入口覆盖虚拟地址范围的状态。**在最好的情况下访问这样的覆盖虚拟地址范围会有不可预料的结果；在某些实现中甚至会物理损坏 MMU。**

强烈建议在重新映射存储器时要加倍小心使 TLB 适当地失效。

-----

表 3-6 TLB 功能

功能	Opcode2	CRm	Data	指令
无效整个唯一的 TLB 或指令和数据 TLB	0b000	0b0111	SBZ	MCR p15,0,Rd,c8,c7,0
无效唯一的单一入口	0b001	0b0111	Virtual address	MCR p15,0,Rd,c8,c7,1
无效整个指令 TLB	0b000	0b0101	SBZ	MCR p15,0,Rd,c8,c5,0
无效指令单一入口	0b001	0b0101	Virtual address	MCR p15,0,Rd,c8,c5,1
无效整个数据 TLB	0b000	0b0110	SBZ	MCR p15,0,Rd,c8,c6,0
无效数据单一入口	0b001	0b0110	Virtual address	MCR p15,0,Rd,c8,c6,1

---注-----

如果使用了第六章描述的快速上下文切换扩展 (FCSE)，那么表 3-6 中的一些功能传递给 CP15 的虚拟地址被当作数据。这意味着对它们来说没有由 FCSE 产生的地址修改。

-----

### 3.7.8 寄存器 10: TLB 锁定

转换表遍历的执行需要一定的时间，特别当访问慢速的主存储器时。在实时中断处理程序中，当 TLB 不包含中断处理程序的转换和/或要访问的数据时，中断延迟回大量加长。

TLB 锁定是一些 ARM 存储器系统的特性，它允许把特定的转换表遍历的结果装载到 TLB 中。这种方式不会被后来的转换表遍历的结果覆盖。由 CP15 寄存器 10 设定。

设  $W = \text{LOG}_2(\text{TLB 入口数})$ ，如果需要的话取整(round-up)，则 CP15 寄存器 10 的格式为：

31	32-W	31-W	32-2W	31-2W	1	0
base		victim		UNP/SBZP		P

如果具体的实现有分开的指令和数据 TLB，那么有 2 个不同的寄存器，由访问寄存器 10 的 MCR 或 MRC 指令中的 opcode2 字段选择：

opcode2 == 0          选择数据 TLB 锁定寄存器

opcode2 == 1          选择指令 TLB 锁定寄存器

如果具体的实现只有唯一的 TLB，那么只有 1 个寄存器，opcode2 字段应该为 0。

访问寄存器 10 的 MCR 或 MRC 指令中的 CRm 总应该为 0。

写寄存器 10 有如下结果：

victim 字段表示下次 TLB 失败(miss)时，转换表遍历的结果替代哪个 TLB 入口。

Base 字段包含 TLB 替换的策略，只使用从(base)到(TLB 入口-1)的 TLB 入口，victim 应该在这个区间。

转换表遍历的结果在写到 TLB 入口时，若 P==1 则它被保护起来，不能被寄存器 8 的使整个 TLB 失效操作影响；若 P==0 则会被那些操作给失效掉。

---注-----

如果 TLB 的入口不是 2 的 N 次方，那么写到大于或等于 TLB 入口数的 TLB 入口的 base 或 victim 的值将不确定。

-----

读寄存器 10 将返回它的值。

## TLB 锁定过程

通常锁定 N 个 TLB 入口的过程如下：

1. 禁止中断等，来保证当这个过程执行时不会产生异常
2. 如果一个指令 TLB 或唯一 TLB 被锁定，用 base==N、index==N 和 P==0 写到适当版本的寄存器 10。如果可能，把另指令预取很难理解的分枝预测功能关掉。
3. 使要被锁定的整个 TLB 失效。
4. 如果是指令 TLB 锁定，要确保剩下的锁定过程所要预取的指令相关的 TLB 入口都被装载。（要注意锁定是从哪里开始的，通常可能一个 TLB 入口包含所有这些。这时 TLB 被失效后的第一条指令能完成这个功能）

如果是数据 TLB 锁定，要确保剩下的锁定过程所要访问数据的相关的 TLB 入口都被装载。这包含被代码用到的嵌入文字 (inline literals)（通常最好避免在锁定过程中使用嵌入文字，并把所有的数据放在由一个 TLB 入口所包含的区域，然后从那里加载一个数据）

如果一个唯一 TLB 被锁定，执行以上所有的过程。

5. i 从 0 到 N-1 循环

- a. 用 base==i、index==i 和 P==1 写到寄存器 10。
- b. 强迫被锁定到 TLB 入口 i 处的转换表遍历结果的存储器区域发生转换表遍历：
  - \* 如果是数据 TLB 或唯一 TLB 被锁定，从那个区域加载一个数据
  - \* 如果是指令 TLB 被锁定，用 B5-15 页所描述的指令预取高速缓冲寄存器 7 来在那个区域产生指令预取。

6. 用 base==N、index==N 和 P==0 写到寄存器 10。

---注-----

如果你使用 FCSE，要注意第 5b 步，因为：

- 如果是数据 TLB 锁定或唯一 TLB 锁定，加载数据指令的地址是会被 FCSE 修改的。
- 如果是指令 TLB 锁定，用在寄存器 7 的地址被当作数据，所以不会被 FCSE 修改。

为了减少混淆，建议锁定过程应该是：

- 从禁止 FCSE 开始（设置 PID 为 0）
- 把适当的 PID 值 OR 到使用的虚拟地址的高 7 位来自己产生修改的虚拟地址。

-----

## TLB 解锁过程

用上面的过程解锁被锁定的 TLB 部分：

1. 用寄存器 8 的操作使每个被锁定的单一入口失效
2. 用 base==0、index==0 和 P==0 写到寄存器 10。

---注-----

第一步是为了保证 P==1 的入口在 TLB 中不在被剩下。如故它们被剩在 TLB 中，后续的 TLB 锁定过程中使整个 TLB 失效（第三步）将不会有预期的结果。

-----

一些术语可能翻译的不准，欢迎来函指教。

[gavinux@yahoo.com](mailto:gavinux@yahoo.com)