engler, cs140e: annotated copy for vm labs.

this is the more specific version of the vm documentation as compared to b4, which applies to the armv6 family.

Chapter 6
Memory Management Unit

This chapter describes the *Memory Management Unit* (*MMU*) and how it is used. It contains the following sections:

6.1 About the MMU

The processor MMU works with the cache memory system to control accesses to and from external memory. The MMU also controls the translation of virtual addresses to physical addresses.

The processor implements an ARMv6 MMU enhanced with TrustZone features to provide address translation and access permission checks for all ports of the processor. The MMU controls table-walking hardware that accesses translation tables in main memory. In each world, Secure and Non-secure, a single set of two-level page tables stored in main memory controls the contents of the instruction and data side *Translation Lookaside Buffers* (TLBs). The finished virtual address to physical address translation is put into the TLB, associated with a *Non-secure Table IDentifier* (NSTID) that permits Secure and Non-secure entries to co-exist. The TLBs are enabled in each world from a single bit in CP15 Control Register c1, providing a single address translation and protection scheme from software.

The MMU features are:

conceivably can do a single 16mb mapping for most of our processes. simple PT. or, if we "lock" it, possibly don't need a page table at all.

- standard ARMv6 MMU mapping sizes, domains, and access protection scheme
- mapping sizes are 4KB, 64KB, 1MB, and 16MB
- the access permissions for 1MB sections and 16MB supersections are specified for the entire section
- you can specify access permissions for 64KB large pages and 4KB small pages separately for each quarter of the page, these quarters are called subpages
- 16 domains
- one 64-entry unified TLB and a lockdown region of eight entries
- you can mark entries as a global mapping, or associated with a specific application space identifier to eliminate the requirement for TLB flushes on most context switches
- access permissions extended to enable Privileged read-only and Privileged or User read-only modes to be simultaneously supported
- memory region attributes to mark pages shared by multiple processors
- hardware page table walks
- separate Secure and Non-secure entries and page tables
- Non-secure memory attribute
- possibility to restrict the eight lockdown entries to the Secure world.

The MMU memory system architecture enables fine-grained control of a memory system. This is controlled by a set of virtual to physical address mappings and associated memory properties held within one or more structures known as TLBs within the MMU. The contents of the TLBs are managed through hardware translation lookups from a set of translation tables in memory.

To prevent requiring a TLB invalidation on a context switch, you can mark each virtual to physical address mapping as being associated with a particular application space, or as global for all application spaces. Only global mappings and those for the current application space are enabled at any time. By changing the *Application Space IDentifier* (ASID) you can alter the enabled set of virtual to physical address mappings.

TrustZone extensions enable the system to mark each entry in the TLB as Secure or Non-secure with the NSTID. At any time the processor only enables entries with an NSTID that matches the Security state of the current application.

The set of memory properties associated with each TLB entry include:

Memory access permission control

This controls if a program has no-access, read-only access, or read/write access to the memory area. When an access is attempted without the required permission, a memory abort is signaled to the processor. The level of access possible can also be affected by whether the program is running in User mode, or a privileged mode, and by the use of domains. See *Memory access control* on page 6-11 for more details.

will use for gpio

Memory region attributes

These describe properties of a memory region. Examples include Strongly Ordered, Device, cacheable Write-Through, and cacheable Write-Back. If an entry for a virtual address is not found in a TLB then a set of translation tables in memory are automatically searched by hardware to create a TLB entry. This process is known as a translation table walk. If the processor is in ARMv5 backwards-compatible mode some new features, such as ASIDs, are not available. The MMU architecture also enables specific TLB entries to be locked down in a TLB. This ensures that accesses to the associated memory areas never require looking up by a translation table walk. This minimizes the worst-case access time to code and data for real-time routines.

Non-secure memory region attribute

This attribute is a TrustZone security extension to the existing ARMv6 MMU. It defines when the target memory is Secure or Non-secure. See *NS attribute* on page 6-19 for a detailed explanation of this bit.

6.2 TLB organization

The following sections describe the TLB organization:

- *MicroTLB*
- *Main TLB* on page 6-5
- *TLB control operations* on page 6-5
- *Page-based attributes* on page 6-5
- *Supersections* on page 6-6.

6.2.1 MicroTLB

The first level of caching for the page table information is a small MicroTLB of ten entries that is implemented on each of the instruction and data sides. These entities are implemented in logic, providing a fully associative lookup of the virtual addresses in a cycle. This means that a MicroTLB miss signal is returned at the end of the DC1 cycle. In addition to the virtual address, an *Address Space IDentifier* (ASID) and a NSTID are used to distinguish different address mappings that might be in use.

The current ASID is a small identifier, eight bits in size, that is programmed using CP15 when different address mappings are required. A memory mapping for a page or section can be marked as being global or referring to a specific ASID. The MicroTLB uses the current ASID in the comparisons of the lookup for all pages for which the global bit is not set.

The NSTID consists of one bit, and is automatically set when a new entry is written. The entry is marked as Secure when the MicroTLB request is Secure, that is when it is performed when the core is in Secure Monitor mode, whatever the value of the NS bit in the CP15 SCR register, or in any Secure mode, NS bit in CP15 SCR = 0.

The MicroTLB returns the physical address to the cache for the address comparison, and also checks the protection attributes in sufficient time to signal a Data Abort in the DC2 cycle. An additional set of attributes, to be used by the cache line miss handler, are provided by the MicroTLB. The timing requirements for these are less critical than for the physical address and the abort checking.

You can configure MicroTLB replacement to be round-robin or random. By default the round-robin replacement algorithm is used. The random replacement algorithm is designed to be selected for rare pathological code that causes extreme use of the MicroTLB. With such code, you can often improve the situation by using a random replacement algorithm for the MicroTLB. You can only select random replacement of the MicroTLB if random cache selection is in force, as set by the Control Register RR bit. If the RR bit is 0, then you can select random replacement of the MicroTLB by setting the Auxiliary Control Register bit 3. This register is only accessible in Secure Privileged modes.

——— **Note** ———

The RR bit is common to the Secure and Non-secure worlds.

———————————

All main TLB maintenance operations affect both the instruction and data MicroTLBs, causing them to be flushed.

The virtual addresses held in the MicroTLB include the FCSE translation from *Virtual Address* (VA) to *Modified Virtual Address* (MVA). For more information see the *ARM Architecture Reference Manual*. The process of loading the MicroTLB from the main TLB includes the FCSE translation if appropriate.

6.2.2 Main TLB

The main TLB is the second layer in the TLB structure that catches the cache misses from the MicroTLBs. It provides a centralized source for translation entries.

Misses from the instruction and data MicroTLBs are handled by a unified main TLB, that is accessed only on MicroTLB misses. Accesses to the main TLB take a variable number of cycles, according to competing requests between each of the MicroTLBs and other implementation-dependent factors. Entries in the lockable region of the main TLB are lockable at the granularity of a single entry, as *c10, TLB Lockdown Register* on page 3-100 describes.

Main TLB implementation

The main TLB is implemented as a combination of two elements:

- A fully-associative array of eight elements, that is lockable.
  You can restrict this region to store Secure entries only, that is entries with NSTID=0, when the TL bit is clear in the NSAC register, see *c1, Non-Secure Access Control Register* on page 3-55

  Note

  — If you clear the TL bit, after creating some NS entries in the Lockdown region, this does not invalidate these entries. The TL bit prevents the creation of new NS entries in the Lockdown region.

  — The TL bit has no influence on the Read/Write Lockdown entry operations, VA PA or Attributes, in the system control coprocessor, see *c15, TLB lockdown access registers* on page 3-149. When the TL bit is set, the processor can write an NS entry in the Lockdown region with the Write Lockdown operation of the system control coprocessor.

- A low-associativity Tag RAM and DataRAM structure similar to that used in the Cache.

The implementation of the low-associativity region is a 64-entry 2-way associative structure. Depending on the RAMs available, you can implement this as either:

"you" = hardware manufacturer

- four 32-bit wide RAMs
- two 64-bit wide RAMs
- a single 128-bit wide RAM.

Main TLB misses

Main TLB misses are handled in hardware by the two level page table walk mechanism, as used on previous ARM processors. See *c8, TLB Operations Register* on page 3-86.

Note

Automatic page table walks might be disabled by PD0 and PD1 bits in the TTB Control register.

i think this means we don't need page table at all, but can compute entries as needed. however, there does not seem to be an "insert tlb entry" for non-lockable entries, which places some severe restrictions on the number of entries.

6.2.3 TLB control operations

c8, TLB Operations Register on page 3-86 and *c10, TLB Lockdown Register* on page 3-100 describe the TLB control operations.

6.2.4 Page-based attributes

Memory access control on page 6-11 describe the page-based attributes for access protection. *Memory region attributes* on page 6-14 and *Memory attributes and types* on page 6-20 describe the memory types and page-based cache control attributes. The processor interprets the Shared bit in the MMU for regions that are Cacheable as making the accesses Noncacheable. This ensures memory coherency without incurring the cost of dedicated cache coherency hardware. *Behavior with MMU disabled* on page 6-9 describes the behavior of the memory system when the MMU is disabled.

6.2.5 Supersections

Supersections are defined using a first level descriptor in the page tables, similar to the way a Section is defined. Because each first level page table entry covers a 1MB region of virtual memory, the 16MB supersections require that 16 identical copies of the first level descriptor of the supersection exist in the first level page table.

means is only useful for OS pages? →

Every supersection is defined to have its Domain as 0.

Supersections can be specified regardless of whether subpages are enabled or not, as controlled by the CP15 Control Register XP bit, bit [23]. This bit is duplicated as Secure and Non-secure, so that supersections can be enabled or disabled separately in each world. Figure 6-6 on page 6-38 and Figure 6-9 on page 6-41 show the page table formats of supersections.

6.3 Memory access sequence

When the processor generates a memory access, the MMU:

1. Performs a lookup for a mapping for the requested virtual address and current ASID and current world, Secure or Non-secure, in the relevant Instruction or Data MicroTLB.
2. If step 1 misses then a lookup for a mapping for the requested virtual address and current ASID and current world, Secure or Non-secure, in the main TLB is performed.

If no global mapping, or mapping for the currently selected ASID, or no matching NSTID, for the virtual address can be found in the TLBs then a translation table walk is automatically performed by hardware, unless Page Table Walks are disabled by the PD0 or PD1 bits in the TTB Control register, that cause the processor to return a Section Translation fault. See *Hardware page table translation* on page 6-36.

If a matching TLB entry is found then the information it contains is used as follows:

1. The access permission bits and the domain are used to determine if the access is permitted. If the access is not permitted the MMU signals a memory abort, otherwise the access is enabled to proceed. *Memory access control* on page 6-11 describes how this is done.
2. The memory region attributes control the cache and write buffer, and determine if the access is Secure or Non-secure cached, uncached, or device, and if it is shared, as *Memory region attributes* on page 6-14 describes.
3. The physical address is used for any access to external or tightly coupled memory to perform Tag matching for cache entries.

6.3.1 TLB match process

Each TLB entry contains a virtual address, a page size, a physical address, and a set of memory properties. Each is marked as being associated with a particular application space, or as global for all application spaces. Register c13 in CP15 determines the currently selected application space. This register is duplicated as Secure and Non-secure to enable fast switching between Secure and Non-secure applications. Each entry is also associated with the Secure or Non-secure world by the NSTID.

A TLB entry matches if the NSTID matches the Secure or Non-secure request state of the MMU request, and if bits [31:N] of the Virtual Address match, where N is log$_2$ of the page size for the TLB entry. It is either marked as global, or the *Application Space IDentifier* (ASID) matches the current ASID. The behavior of a TLB if two or more entries match at any time, including global and ASID-specific entries, is Unpredictable. The operating system must ensure that, at most, one TLB entry matches at any time. With respect to operation in the Secure and Non-secure worlds, multiple matching can only occur on entries with the same NSTID, that is a Non-secure entry and a Secure entry can never be hit simultaneously.

A TLB can store entries based on the following four block sizes:

Supersections	Consist of 16MB blocks of memory.
Sections	Consist of 1MB blocks of memory.
Large pages	Consist of 64KB blocks of memory.
Small pages	Consist of 4KB blocks of memory.

Supersections, sections, and large pages are supported to permit mapping of a large region of memory while using only a single entry in a TLB. If no mapping for an address is found within the TLB, then the translation table is automatically read by hardware, if not disabled with PD0 and PD1 bits in the TTB Control register, and a mapping is placed in the TLB. See *Hardware page table translation* on page 6-36 for more details.

6.3.2 Virtual to physical translation mapping restrictions

You can use the processor MMU architecture in conjunction with virtually indexed physically tagged caches. For details of any mapping page table restrictions for virtual to physical addresses see *Restrictions on page table mappings page coloring* on page 6-41.

6.3.3 Tightly-Coupled Memory

There are no page table restrictions for mappings to the *Tightly-Coupled Memory* (TCM). For details of the TCM see *Tightly-coupled memory* on page 7-7.

6.4 Enabling and disabling the MMU

You can enable and disable the MMU by writing the M bit, bit 0, of the CP15 Control Register c1. On reset, this bit is cleared to 0, disabling the MMU. This bit, in addition to most of the MMU control parameters, is duplicated as Secure and Non-secure, to ensure a clear and distinct memory management policy in each world.

6.4.1 Enabling the MMU

To enable the MMU in one world you must:

1. Program all relevant CP15 registers of the corresponding world.
2. Program first-level and second-level descriptor page tables as required.
3. Disable and invalidate the Instruction Cache for the corresponding world. You can then re-enable the Instruction Cache when you enable the MMU.
4. Enable the MMU by setting bit 0 in the CP15 Control Register in the corresponding world.

6.4.2 Disabling the MMU

To disable the MMU in one world proceed as follows:

1. Clear bit 2 to 0 in the CP15 Control Register c1 of the corresponding world, to disable the Data Cache. You must disable the Data Cache in the corresponding world before, or at the same time as, disabling the MMU.

   Note

   If the MMU is enabled, then disabled, and subsequently re-enabled in the same world, the contents of the TLBs for this world are preserved. If these are now invalid, you must invalidate the TLBs in the corresponding world before you re-enable the MMU, see *c8, TLB Operations Register* on page 3-86.

2. Clear bit 0 to 0 in the CP15 Control Register c1 of the corresponding world.

6.4.3 Behavior with MMU disabled

When the MMU is disabled, the Data Cache is disabled and memory accesses are treated as follows for the corresponding world:

- When the TEX remap bit, bit [28] in the CP15 Control Register, is reset to 0, behavior is backward compatible:
  - All data accesses are treated as Strongly Ordered. The value of the C bit, bit [2] in the CP15 Control Register of the corresponding world, Should Be Zero.
  - All instruction accesses are treated as Cacheable if the I bit, bit [12] of the CP15 Control Register of the corresponding world, is set to 1, and Strongly Ordered if the I bit is reset to 0.
- When the TEX remap bit, bit [28] in the CP15 Control Register, is set to 1:
  - all accesses are treated with the same parameters, independently of the C and I bit values
  - those parameters depend on the programming of the PRRR and NMRR registers, see *TexRemap=1 configuration* on page 6-16 for more information on this behavior.

Note

By default, the PRRR and NMRR registers are reset to that all accesses are treated as Strongly Ordered.

The other parameters of the MMU behavior when disabled, independent of the TEX remap configuration, are:

- No memory access permission or Access bit checks are performed, and no aborts are generated by the MMU.
- The physical address for every access is equal to its virtual address. This is known as a flat address mapping.
- The NS attribute for the target memory region is equal to the state, Secure or Non-secure, of the request, that is Secure requests are considered to target Secure memory.
- The FCSE PID Should Be Zero when the MMU is disabled. This is the reset value of the FCSE PID. If the MMU is to be disabled the FCSE PID must be cleared.
- All CP15 MMU and cache operations can be executed even when the MMU is disabled.
- Accesses to the TCMs work as normal if the TCMs are enabled.

6.5 Memory access control

Access to a memory region is controlled by:

- *Domains*
- *Access permissions*
- *Execute never bits in the TLB entry* on page 6-12.

6.5.1 Domains

A domain is a collection of memory regions. In compliance with the ARM Architecture and the TrustZone Security Extensions, the ARM1176JZF-S supports 16 Domains in the Secure world and 16 Domains in the Non-secure world. Domains provide support for multi-user operating systems. All regions of memory have an associated domain.

domains = a way you can switch the page protection for a large set of pages with a single register write. it's an unusual feature of armv6. afaik it was dropped in later versions. neat final project is using this to isolate drivers, etc but keep in the same address space (which makes communication easier since addresses are same)

A domain is the primary access control mechanism for a region of memory and defines the conditions when an access can proceed. The domain determines whether:

- access permissions are used to qualify the access
- access is unconditionally permitted to proceed
- access is unconditionally aborted.

In the latter two cases, the access permission attributes are ignored.

Each page table entry and TLB entry contains a field that specifies the domain that the entry is in. Access to each domain is controlled by a 2-bit field in the Domain Access Control Register, CP15 c3. Each field enables very quick access to be achieved to an entire domain, so that whole memory areas can be efficiently swapped in and out of virtual memory. Two kinds of domain access are supported:

Clients	Clients are users of domains in that they execute programs and access data. They are guarded by the access permissions of the TLB entries for that domain. A client is a domain user, and each access has to be checked against the access permission settings for each memory block and the system protection bit, the S bit, and the ROM protection bit, the R bit, in CP15 Control Register c1. Table 6-1 on page 6-12 lists the access permissions.
Managers	Managers control the behavior of the domain, the current sections and pages in the domain, and the domain access. They are not guarded by the access permissions for TLB entries in that domain. Because a manager controls the domain behavior, each access has only to be checked to be a manager of the domain.

One program can be a client of some domains, and a manager of some other domains, and have no access to the remaining domains. This enables flexible memory protection for programs that access different memory resources.

6.5.2 Access permissions

The access permission bits control access to the corresponding memory region. If an access is made to an area of memory without the required permissions, then a permission fault is raised.

The access permissions are determined by a combination of the AP and APX bits in the page table, and the S and R bits in CP15 Control Register c1. For page tables not supporting the APX bit, the value 0 is used.

? You do not have to flush the TLB to enable the new S and R bit to take effect. Access permissions of entries in the TLB are automatically affected by the new S and R values.

Note

The use of the S and R bits is deprecated.

Table 6-1 lists the encoding of the access permission bits.

Table 6-1 Access permission bit encoding

APX	AP[1:0]	Privileged permissions	User permissions
0	b00	No access, recommended use. Read-only when S=1 and R=0 or when S=0 and R=1, deprecated.	No access, recommended use. Read-only when S=0 and R=1, deprecated.
0	b01	Read/write.	No access.
0	b10	Read/write.	Read-only.
0	b11	Read/write.	Read/write.
1	b00	Reserved.	Reserved.
1	b01	Read-only.	No access.
1	b10	Read-only.	Read-only.
1	b11	Read-only.	Read-only.

Restricted access permissions and the access bit

The Access bit is an ARMv6 enhancement, for full details see *Access bit fault* on page 6-32. Some OSs only use a restricted set of the access permissions:

- APX and AP[1:0] = b111, Read-Only for both Privileged and Unprivileged code
- APX and AP[1:0] = b011, Read-Write for both Privileged and Unprivileged code
- APX and AP[1:0] = b101, Read-Only for Privileged code, No Access for Unprivileged
- APX and AP[1:0] = b001, Read-Write for Privileged code, No Access for Unprivileged.

For such OSs the encoding of the Read-Only or Read-Write and the User or Kernel access permissions are orthogonal:

- APX selects the Read-Only or Read-Write permission
- AP[1] selects the User or Kernel access.

In this case, the AP[0] bit provides Access bit information so that software can optimize the memory management algorithm.

The Access bit behaves in this way except in the deprecated case that uses the S and R bits, that is when the S and R bits have opposite values, and when APX and AP[1:0] = b000.

6.5.3 Execute never bits in the TLB entry

Each memory region can be tagged as not containing executable code. If the Execute Never, XN, bit of the TLB entry is set to 1, then any attempt to execute an instruction in that region results in a permission fault. If the XN bit is cleared, then code can execute from that memory region. When the MMU is in ARMv5 mode, see the XP bit in *c1, Control Register* on page 3-44, the descriptors do not contain the XN bit, and all pages are executable. In ARMv6 mode, XP bit =1, the descriptors specify the XN attribute, see Figure 6-7 on page 6-39 and Figure 6-8 on page 6-40.

6.6 Memory region attributes

Each TLB entry has an associated set of memory region attributes. These control:

- accesses to the caches
- how the write buffer is used
- if the memory region is shareable
- if the targeted memory is Secure or not.

6.6.1 C and B bit, and type extension field encodings

The ARMv6 MMU architecture originally defined five bits to describe all of the options for inner and outer cachability. These five bits, the Type Extension Field, TEX[2:0], Cacheable, C, and Bufferable, B bits, are set in the descriptors.

Few application make use of all these options simultaneously. For this reason, a new configuration bit, TEX remap, bit [28] in the CP15 Control Register, permits the core to support a smaller number of options by using only the TEX[0], C and B bits.

The OS can configure this subset of options through a remap mechanism for these TEX[0], C, and B bits. The TEX[2:1] bits in the descriptor then become 2 OS managed page table bits.

Additionally, certain page tables contain the Shared bit, S, used to determine if the memory region is Shared or not. If not present in the descriptor, the Shared bit is assumed to be 0, Non-Shared. In the TexRemap=1 configuration, the Shared bit can be remapped too.

For TrustZone support, the TEX remap bit is duplicated as Secure and Non-secure versions, so it is possible to configure in each world the options that are available to the core.

The TLB does not cache the effect of the TEX remap bit on page tables. As a result, there is no requirement for the processor to invalidate the TLB on a change of the TEX remap bit to rely on the effect of those changes taking place.

Note

The terms Inner and Outer in this document represent the levels of caches that can be built in a system. Inner refers to the innermost caches, including level one. Outer refers to the outermost caches. The boundary between Inner and Outer caches is defined in the implementation of a cached system. Inner must always include level one. In a system with three levels of caches, an example is for the Inner attributes to apply to level one and level two, while the Outer attributes apply to level three. In a two-level system, it is envisaged that Inner always applies to level one and Outer to level two.

In the processor, Inner refers to level one and the **ARSIDEBAND[4:1]**, for read, and **AWSIDEBAND[4:1]**, for writes, signals show the Inner Cacheable values.

ARCACHE, for reads, and **AWCACHE**, for writes, show the Outer Cacheable properties.

TexRemap=0 configuration

This is the standard ARMv6 configuration. The five TEX[2:0], C, and B bits are used to encode the memory region type. For page tables formats with no TEX field, you must use the value 3'b000.

The S bit in the descriptors only applies to Normal, that is not Device and not Strongly Ordered memory. Table 6-2 summarizes the TEX[2:0], C, and B encodings used in the page table formats, and the value of the shareable attribute of the concerned page:

Table 6-2 TEX field, and C and B bit encodings used in page table formats

Page table encodings			Description	Memory type	Page shareable?
TEX	**C**	**B**			
b000	0	0	Strongly Ordered	Strongly Ordered	Shared[a]
b000	0	1	Shared Device	Device	Shared[a]
b000	1	0	Outer and Inner Write-Through, No Allocate on Write	Normal	s[b]
b000	1	1	Outer and Inner Write-Back, No Allocate on Write	Normal	s[b]
b001	0	0	Outer and Inner Noncacheable	Normal	s[b]
b001	0	1	Reserved	-	-
b001	1	0	Reserved	-	-
b001	1	1	Outer and Inner Write-Back, Allocate on Write[c]	Normal	s[b]
b010	0	0	Non-Shared Device	Device	Non-shared
b010	0	1	Reserved	-	-
010	1	X	Reserved	-	-
011	X	X	Reserved	-	-
1BB	A	A	Cached memory. BB = Outer policy, AA = Inner policy. See Table 6-3 on page 6-16.	Normal	s[b]

a. Shared, regardless of the value of the S bit in the page table.
b. s is Shared if the value of the S bit in the page table is 1, or Non-shared if the value of the S bit is 0 or not present.
c. The cache does not implement allocate on write.

The Inner and Outer cache policy bits control the operation of memory accesses to the external memory:

- The C and B bits are described as the AA bits and define the Inner cache policy
- The TEX[1:0] bits are described as the BB bits and define the Outer cache policy.

Table 6-3 shows how the MMU and cache interpret the cache policy bits.

Table 6-3 Cache policy bits

BB or AA bits	Cache policy
b00	Noncacheable
b01	Write-Back cached, Write Allocate
b10	Write-Through cached, No Allocate on Write
b11	Write-Back cached, No Allocate on Write

You can choose the write allocation policy that an implementation supports. The Allocate On Write and No Allocate On Write cache policies indicate the preferred allocation policy for a memory region, but you must not rely on the memory system implementing that policy. The processor does not support Inner Allocate on Write.

Not all Inner and Outer cache policies are mandatory. Table 6-4 lists possible implementation options.

Table 6-4 Inner and Outer cache policy implementation options

GPIO

Cache policy	Implementation options	Supported by the processor
Inner Noncacheable	Mandatory.	Yes
Inner Write-Through	Mandatory.	Yes
Inner Write-Back	Optional. If not supported, the memory system must implement this as Inner Write-Through.	Yes
Outer Noncacheable	Mandatory.	System-dependent
Outer Write-Through	Optional. If not supported, the memory system must implement this as Outer Non-cacheable.	System-dependent
Outer Write-Back	Optional. If not supported, the memory system must implement this as Outer Write-Through.	System-dependent

When the MMU is off and TexRemap=0:

- All data accesses are treated as Shared, Inner Strongly Ordered, Outer Non-cacheable.
- Instruction accesses are treated as Non-Shared, Inner and Outer Write-Through, No Allocate on Write, when the Instruction Cache is on, I=1, bit [12], see *c1, Control Register* on page 3-44.

  Instruction accesses are treated as Shared, Inner Strongly Ordered, Outer Non-Cacheable, when the Instruction Cache is off, see *Behavior with MMU disabled* on page 6-9.

TexRemap=1 configuration

Only three bits, TEX[0], C, and B, are relevant in this configuration. The OS can use the TEX[2:1] bits to manage the page tables.

In this configuration the processor provides the OS with a remap capability for the memory attribute. Two CP15 registers, the *Primary Region Remap Register* (PRRR) and the *Normal Memory Region Register* (NMRR) come into effect.

You can access the memory region remap registers of the MMU with:

`MCR/MRC {cond} p15, 0, Rd, c10, c2, 0` for the Primary Region Remap register and `MCR/MRC {cond} p15, 0, Rd, c10, c2, 1` for the Normal Memory Region Remap register, see *c10, Memory region remap registers* on page 3-101.

The remapping applies to all sources of MMU requests, that is the two registers are applicable to Data, Instruction and DMA requests.

For TrustZone support, the PRRR and NMRR registers are duplicated as Secure and Non-secure versions, and the processor uses the appropriate one for the remapping depending on whether the MMU request is Secure or not.

The PRRR and NMRR registers are expected to be static throughout operation.

However, if the PRRR or NMRR registers are modified in one world, the changes take effect immediately and enable each of the entries contained in the main TLB to be remapped, without the requirement to invalidate the TLB.

The remap capability has two levels:

1. The first level, the Primary Region Remap, enables remap of the primary memory type, Normal, Device or Strongly Ordered. See Table 6-5.
2. After primary remapping, any region remapped as Normal memory has the Inner and Outer cacheable attributes remapped by the Normal Memory Region Remap register. See Table 6-5. To provide maximum flexibility, this level of remapping permits regions that were originally not Normal memory to be remapped independently.

Similarly, if the obtained, remapped, memory type is Device or Normal memory, the S bit in the descriptor is independently remapped according to one of the PRRR[19:16] bit. See Table 6-6 on page 6-18.

Table 6-5 summarizes the parts of the PRRR and NMRR that are used to remap the different memory region attributes.

Table 6-5 Effect of remapping memory with TEX remap = 1

Page Table encodings			**Memory type**	**Inner Cache attributes when mapped as Normal**	**Outer Cache attributes when mapped as Normal**
TEX	**C**	**B**			
XX0	0	0	PRRR[1:0]	NMRR[1:0]	NMRR[17:16]
XX0	0	1	PRRR[3:2]	NMRR[3:2]	NMRR[19:18]
XX0	1	0	PRRR[5:4]	NMRR[5:4]	NMRR[21:20]
XX0	1	1	PRRR[7:6]	NMRR[7:6]	NMRR[23:22]
XX1	0	0	PRRR[9:8]	NMRR[9:8]	NMRR[25:24]
XX1	0	1	PRRR[11:10]	NMRR[11:10]	NMRR[27:26]
XX1	1	0	PRRR[13:12]	NMRR[13:12]	NMRR[29:28[
XX1	1	1	PRRR[15:14]	NMRR[15:14]	NMRR[31:30]

Table 6-6 lists how the memory type, the value of the S bit in the page table attributes, and the primary remap region register determine how the pages can be shared.

Table 6-6 Values that remap the shareable attribute

Memory Type	Shareable attribute when: S=0	S=1
Strongly Ordered	Shareable	Shareable
Device	PRRR[16]	PRRR[17]
Normal	PRRR[18]	PRRR[19]

Table 6-7 lists the encoding used for each region in the PRRR register, bits [15:0].

Table 6-7 Primary region type encoding

Region	Encoding
Strongly Ordered	b00
Device	b01
Normal Memory	b10
Unpredictable, normal memory for ARM1176JZF-S	b11

GPIO

Table 6-8 lists the encoding used for each Inner or Outer Cacheable attribute in the NMRR register, bits [31:0].

Table 6-8 Inner and outer region remap encoding

Inner or Outer Region	Encoding
Non-Cacheable	b00
WriteBack, WriteAllocate	b01
WriteThrough, Non-Write Allocate	b10
WriteBack, Non-WriteAllocate	b11

When the MMU is off the remapping takes place according to the settings in PRRR[1:0], and PRRR[19],PRRR[17], NMRR[1:0], and NMRR[17:16] as appropriate.

In this case, the S bit is treated as if it is 1 prior to remapping. This behavior takes place regardless of whether or not the instruction cache is enabled.

Note

- The reset value for each field of the PRRR and NMRR makes the MMU behave as if no remapping occurs, that is Strongly Ordered regions are remapped as Strongly Ordered and so on.
- For security reasons, the NS Attribute bit has no remap capability.

6.6.2 Shared

n/a since we are single processor

This bit indicates that the memory region can be shared by multiple processors. For a full explanation of the Shared attribute see *Memory attributes and types* on page 6-20.

6.6.3 NS attribute

The NS attribute is a TrustZone extension to the V6 MMU. It is specified in the L1 descriptors, in position 19 for sections and supersections, and in position 3 for coarse pages. It defines if the targeted memory region corresponding to the page is Secure or Non-secure, that is if this memory region is accessed with Secure or with Non-secure rights. This bit is ignored in the Non-secure world.

When the MMU is off, the NS Attribute is equal to the state, Secure or Non-secure, of the MMU request.

When the NS Attribute is set to 1, the access is performed with Non-secure rights:

- If the access is cacheable, it can only hit a cache line whose NS-Tag is Non-secure. If this access causes a linefill, then the created line in the cache has its NS Tag set to 1, Non-secure.
- The access can only hit TCM configured as Non-secure.
- If the access goes external to the core, then it is marked as Non-secure with **AxPROT[1]** = Non-secure.

The NS Attribute is specified in the L1 descriptors, in position 19 for sections and supersections, and in position 3 for coarse pages. The bit contained in the NS descriptors is always ignored, so that all NS entries in the TLB, that is entries with NSTID=1=Non-secure, have the NS Attribute=1=Non-secure. This ensures that the NS world always perform accesses with NS rights.

Note

This rule is also true when a new entry is created in the Lockdown region with the CP15 Read/Write PA in TLB Lockdown region operation. For this operation, when an entry is written with NSTID=1, then the corresponding NS Attribute of the entry is forced to 1. See *c15, TLB lockdown access registers* on page 3-149.

With this mechanism, only the Secure world can perform Secure accesses, and consequently is the only one permitted to access Secure memory. The Secure world can also access Non-secure memory, by setting the NS Attribute appropriately in the corresponding descriptor. The Non-secure world can only access Non-secure memory.

There is no check of the NS Attribute internally, and therefore the system can not generate an error because of a wrong NS Attribute. Only external aborts can be generated, if the system has implemented this feature.

6.7 Memory attributes and types

The processor provides a set of memory attributes that have characteristics that are suited to particular devices, including memory devices, that can be contained in the memory map. The ordering of accesses for regions of memory is also defined by the memory attributes. There are three mutually exclusive main memory type attributes:

- Strongly Ordered
- Device
- Normal.

These are used to describe the memory regions. The marking of the same memory locations as having two different attributes in the MMU, for example using synonyms in a virtual to physical address mapping, results in Unpredictable behavior but this does not break security. Table 6-9 lists a summary of the memory attributes.

Table 6-9 Memory attributes

Memory type attribute	Shared or Non-shared	Other attributes	Description
Strongly Ordered	-	-	All memory accesses to Strongly Ordered memory occur in program order. Some backwards compatibility constraints exist with ARMv5 instructions that change the CPSR interrupt masks. See *Strongly Ordered memory attribute* on page 6-23. All Strongly Ordered accesses are assumed to be shared.
Device	Shared	- n/a	Designed to handle memory-mapped peripherals that are shared by several processors.
	Non-shared	-	Designed to handle memory-mapped peripherals that are used only by a single processor.
Normal	Shared	Noncacheable/ Write-Through Cacheable/ Write-Back Cacheable	Designed to handle normal memory that is shared between several processors.
	Non-shared	Noncacheable/ Write-Through Cacheable/ Write-Back Cacheable	Designed to handle normal memory that is used only by a single processor.

6.7.1 Normal memory attribute

The Normal memory attribute is defined on a per-page basis in the MMU and provides memory access orderings that are suitable for normal memory. This type of memory stores information without side effects. Normal memory can be writable or read-only. For writable normal memory, unless there is a change to the physical address mapping:

- a load from a specific location returns the most recently stored data at that location for the same processor

bad for device mem since can spontaneously change without a store! $\rightarrow$

- two loads from a specific location, without a store in between, return the same data for each load.

For read-only normal memory:

- two loads from a specific location return the same data for each load.

This behavior describes most memory used in a system, and the term memory-like is used to describe this sort of memory. In this section, writable normal memory and read-only normal memory are not distinguished. Regions of memory with the Normal attribute can be Shared or Non-Shared, on a per-page basis in the MMU. The marking of the same memory locations as being Shared Normal and Non-Shared Normal in the MMU, for example by the use of synonyms in a virtual to physical address mapping, results in Unpredictable behavior but this does not break security. All explicit accesses to memory marked as Normal must correspond to the ordering requirements of accesses that *Ordering requirements for memory accesses* on page 6-23 describes. Accesses to Normal memory conform to the Weakly Ordered model of memory ordering. A description of this model is in standard texts describing memory ordering issues.

Shared Normal memory

The Shared Normal memory attribute is designed to describe normal memory that can be accessed by multiple processors or other system masters. A region of memory marked as Shared Normal is one where the effect of interposing a cache, or caches, on the memory system is entirely transparent. Implementations can use a variety of mechanisms to support this, from not caching accesses in shared regions to more complex hardware schemes for cache coherency for those regions. The processor does not cache shareable locations at level one. In systems that implement a TCM, the regions of memory covered by the TCM must not be marked as Shared. The attributes for these regions are remapped to Inner and Outer Write-Back Non-Shared. Writes to Shared Normal memory might not be atomic. That is, all observers might not see the writes occurring at the same time. To preserve coherence where two writes are made to the same location, the order of those writes must be seen to be the same by all observers. Reads to Shared Normal memory that are aligned in memory to the size of the access are atomic.

Non-Shared Normal memory

The Non-Shared Normal memory attribute describes normal memory that can be accessed only by a single processor. A region of memory marked as Non-Shared Normal does not have any requirement to make the effect of a cache transparent.

Cacheable Write-Through, Cacheable Write-Back, and Noncacheable

In addition to marking a region of Normal memory as being Shared or Non-Shared, a region of memory marked as Normal can also be marked on a per-page basis in an MMU as being one of:

- Cacheable Write-Through
- Cacheable Write-Back
- Noncacheable.

This marking is independent of the marking of a region of memory as being Shared or Non-Shared, and indicates the required handling of the data region for reasons other than those to handle the requirements of shared data. As a result, a region of memory that is marked as being Cacheable and Shared is not cached by the processor at level one. Marking the same memory locations as having different Cacheable attributes, for example by the use of synonyms in a virtual to physical address mapping, results in Unpredictable behavior but does not break security.

6.7.2 Device memory attribute

The Device memory attribute is defined for memory locations where an access to the location can cause side effects, or where the value returned for a load can vary depending on the number of loads performed. Memory-mapped peripherals and I/O locations are typical examples of areas of memory that you must mark as Device. The marking of a region of memory as Device is performed on a per-page basis in the MMU.

Accesses to memory-mapped locations that have side effects that apply to memory locations that are Normal memory might require Memory Barriers to ensure correct execution. An example where this might be an issue is the programming of the control registers of a memory controller while accesses are being made to the memories controlled by the controller. Instruction fetches must not be performed to areas of memory containing read-sensitive devices, because there is no ordering requirement between instruction fetches and explicit accesses.

n/a (i think)

As a result, instruction fetches from such devices can result in Unpredictable behavior. Up to 64 bytes can be prefetched sequentially ahead of the current instruction being executed. To enable this, read-sensitive devices must be located in memory in such a way to enable this prefetching.

Explicit accesses from the processor to regions of memory marked as Device occur at the size and order defined by the instruction. The number of location accesses is specified by the program. Repeat accesses to such locations when there is only one access in the program, that is the accesses are not restartable, are not possible in the processor.

An example of where a repeat access might be required is before and after an interrupt to enable the interrupt to abandon a slow access. You must ensure these optimizations are not performed on regions of memory marked as Device. If a memory operation that causes multiple transactions, such as an LDM or an unaligned memory access, crosses a 4KB address boundary, then it can perform more accesses than are specified by the program, regardless of one or both of the areas being marked as Device.

afaik, the r/pi must have already handled this for us in how it laid out dev memory addresses

For this reason, accesses to volatile memory devices must not be made using single instructions that cross a 4KB address boundary. This restriction is expected to cause restrictions to the placing of such devices in the memory map of a system, rather than to cause a compiler to be aware of the alignment of memory accesses. In addition, address locations marked as Device are not held in a cache.

Shared memory attribute

Regions of Memory marked as Device are also distinguished by the Shared attribute in the MMU. These memory regions can be marked as:

- Shared Device
- Non-Shared Device.

Explicit accesses to memory with each of the sets of attributes occur in program order relative to other explicit accesses to the same set of attributes. All explicit accesses to memory marked as Device must correspond to the ordering requirements of accesses that *Ordering requirements for memory accesses* on page 6-23 describes. The marking of the same memory location as being Shared Device and Non-Shared Device in an MMU, for example by the use of synonyms in a virtual to physical address mapping, results in Unpredictable behavior but this does not break security.

An example of an implementation where the Shared attribute is used to distinguish memory accesses is an implementation that supports a local bus for its private peripherals, while system peripherals are situated on the main system bus. Such a system can have more predictable access times for local peripherals such as watchdog timers or interrupt controllers. For shared device memory, the data of a write is visible to all observers before the end of a Data Synchronization

Barrier memory barrier. For non-shared device memory, the data of a write is visible to the processor before the end of a Data Synchronization Barrier memory barrier. See *Explicit Memory Barriers* on page 6-25.

6.7.3 Strongly Ordered memory attribute

Another memory attribute, Strongly Ordered, is defined on a per-page basis in the MMU. Accesses to memory marked as Strongly Ordered have a strong memory-ordering model with respect to all explicit memory accesses from that processor. An access to memory marked as Strongly Ordered acts as a memory barrier to all other explicit accesses from that processor, until the point at which the access is complete.

That is, has changed the state of the target location or data has been returned. In addition, an access to memory marked as Strongly Ordered must complete before the end of a Memory Barrier. See *Explicit Memory Barriers* on page 6-25. To maintain backwards compatibility with ARMv5 architecture, any ARMv5 instructions that implicitly or explicitly change the interrupt masks in the CSPR that appear in program order after a Strongly Ordered access must wait for the Strongly Ordered memory access to complete.

These instructions are MSR with the control field mask bit set, and the flag setting variants of arithmetic and logical instructions whose destination register is R15, that copies the SPSR to CSPR. This requirement exists only for backwards compatibility with previous versions of the ARM architecture, and the behavior is deprecated in ARMv6. Programs must not rely on this behavior, but instead include an explicit Memory Barrier between the memory access and the following instruction. See *Explicit Memory Barriers* on page 6-25.

The processor does not require an explicit memory barrier in this situation, but for future compatibility it is recommended that programmers insert a memory barrier.

Explicit accesses from the processor to memory marked as Strongly Ordered occur at their program size, and the number of accesses that occur to such locations is the number that are specified by the program. Implementations must not repeat accesses to such locations when there is only one access in the program. That is, the accesses are not restartable.

If a memory operation that causes multiple transactions, such as LDM or an unaligned memory access, crosses a 4KB address boundary, then it might perform more accesses than are specified by the program regardless of one or both of the areas being marked as Strongly Ordered.

For this reason, it is important that accesses to volatile memory devices are not made using single instructions that cross a 4KB address boundary. Address locations marked as Strongly Ordered are not held in a cache, and are treated as Shared memory locations. For Strongly Ordered memory, the data and side effects of a write are visible to all observers before the end of a Data Synchronization Barrier memory barrier. See *Explicit Memory Barriers* on page 6-25.

6.7.4 Ordering requirements for memory accesses

The various memory types defined in this section have restrictions in the memory orderings that are permitted.

Ordering requirements for two accesses

The order of any two explicit architectural memory accesses where one or more are to memory marked as Non-Shared must obey the ordering requirements that Figure 6-1 on page 6-24 lists.

Figure 6-1 shows the memory ordering between two explicit accesses A1 and A2, where A1 occurs before A2 in program order. The symbols used in the figure are as follows:

<	Accesses must occur strictly in program order. That is, A1 must occur strictly before A2. It must be impossible to tell otherwise from observation of the read/write values and side effects caused by the memory accesses.
?	Accesses can occur in any order, provided that the requirements of uniprocessor semantics are met, for example respecting dependencies between instructions within a single processor.

A1 \ A2	Normal read	Device read		Strongly Ordered read	Normal write	Device write		Strongly Ordered write
		Non-Shared	Shared			Non-Shared	Shared	
Normal read	?	?	?	<	?[a]	?	?	<
Device read, Non-Shared	?	<	?	<	?	<	?	<
Device read, Shared	?	?	<	<	?	?	<	<
Strongly Ordered read	<	<	<	<	<	<	<	<
Normal write	?	?	?	<	?	?	?	<
Device write, Non-Shared	?	<	?	<	?	<	?	<
Device write, Shared	?	?	<	<	?	?	<	<
Strongly Ordered write	<	<	<	<	<	<	<	<

a. The processor orders the normal read ahead of normal write.

Figure 6-1 Memory ordering restrictions

There are no ordering requirements for implicit accesses to any type of memory.

Definition of program order of memory accesses

The program order of instruction execution is defined as the order of the instructions in the control flow trace. Two explicit memory accesses in an execution can either be:

Ordered Denoted by <. If the accesses are Ordered, then they must occur strictly in order.

Weakly Ordered Denoted by <=. If the accesses are Weakly Ordered, then they must occur in order or simultaneously.

The rules for determining this for two accesses A1 and A2 are:

1. If A1 and A2 are generated by two different instructions, then:
   - A1 < A2 if the instruction that generates A1 occurs before the instruction that generates A2 in program order.
   - A2 < A1 if the instruction that generates A2 occurs before the instruction that generates A1 in program order.

2. If A1 and A2 are generated by the same instruction, then:
   - If A1 and A2 are the load and store generated by a SWP or SWPB instruction, then:
     - A1 < A2 if A1 is the load and A2 is the store
     - A2 < A1 if A2 is the load and A1 is the store.
   - If A1 and A2 are two word loads generated by an LDC, LDRD, or LDM instruction, or two word stores generated by an STC, STRD, or STM instruction, but excluding LDM or STM instructions whose register list includes the PC, then:
     - A1 <= A2 if the address of A1 is less than the address of A2
     - A2 <= A1 if the address of A2 is less than the address of A1.
   - If A1 and A2 are two word loads generated by an LDM instruction whose register list includes the PC or two word stores generated by an STM instruction whose register list includes the PC, then the program order of the memory operations is not defined.

Multiple load and store instructions, such as LDM, LDRD, STM, and STRD, generate multiple word accesses, each being a separate access to determine ordering.

6.7.5 Explicit Memory Barriers

✳

This section describes two explicit Memory Barrier operations:
- Data Memory Barrier
- Data Synchronization Barrier.

In addition, to ensure correct operation where the processor writes code, an explicit Flush Prefetch Buffer operation is provided.

These operations are implemented by writing to the CP15 Cache operation register c7. For details on how to use this register see *c7, Cache operations* on page 3-69. For more information on explicit memory barriers, see the *ARM Architecture Reference Manual*.

just stops memory load/store reordering

DSB >> DMS

stops reordering of mem w.r.t. coprocessor 15 instructions (and all overs)

Data Memory Barrier

This memory barrier ensures that all explicit memory transactions occurring in program order before this instruction are completed. No explicit memory transactions occurring in program order after this instruction are started until this instruction completes. Other instructions can complete out of order with the Data Memory Barrier instruction.

Data Synchronization Barrier

This memory barrier completes when all explicit memory transactions occurring in program order before this instruction are completed. No explicit memory transactions occurring in program order after this instruction are started until this instruction completes. In fact, no instructions occurring in program order after the Data Synchronization Barrier complete, or change the interrupt masks, until this instruction completes.

Flush Prefetch Buffer

The Flush Prefetch Buffer operation flushes the pipeline in the processor, so that all instructions following the pipeline flush are fetched from memory, including the cache, after the instruction has been completed. Combined with Data Synchronization Barrier, and potentially invalidating the Instruction Cache, this ensures that any instructions written by the processor are executed. This guarantee is required as part of the mechanism for handling self-modifying code. Performing a Data Synchronization Barrier operation and invalidating the Instruction Cache and Branch Target Cache are also required for the handling of self-modifying code. The Flush

Prefetch Buffer is guaranteed to perform this function, while alternative methods of performing the same task, such as a branch instruction, can be optimized in the hardware to avoid the pipeline flush, for example, by using a branch predictor.

6.7.6 Backwards compatibility

The ARMv6 memory attributes are significantly different from those in previous versions of the architecture. Table 6-10 lists the interpretation of the earlier memory types in the light of this definition.

Table 6-10 Memory region backwards compatibility

Previous architectures	ARMv6 attribute
NCNB, Noncacheable, Non Bufferable	Strongly Ordered
NCB, Noncacheable, Bufferable	Shared Device
Write-Through, Cacheable, Bufferable	Non-Shared Normal, Write-Through Cacheable
Write-Back, Cacheable, Bufferable	Non-Shared Normal, Write-Back Cacheable

6.8 MMU aborts

Mechanisms that can cause the processor to take an exception because of a memory access are:

MMU fault	The MMU detects a restriction and signals the processor.
Debug abort	Monitor debug-mode debug is enabled and a breakpoint or a watchpoint has been detected.
External abort	The external memory system signals an illegal or faulting memory access.

Collectively these are called *aborts*. Accesses that cause aborts are said to be aborted. If the memory request that aborts is an instruction fetch, then a Prefetch Abort exception is raised if and when the processor attempts to execute the instruction corresponding to the aborted access.

If the aborted access is a data access or a cache maintenance operation, a Data Abort exception is raised.

All Data Aborts, and aborts caused by cache maintenance operations, cause the *Data Fault Status Register* (DFSR) to be updated so that you can determine the cause of the abort.

For all Data Aborts, excluding external aborts, other than on translation, the *Fault Address Register* (FAR) is updated with the address that caused the abort. External Data Aborts, other than on translation, can all be imprecise and therefore the FAR does not contain the address of the abort. See *Imprecise Data Abort mask in the CPSR/SPSR* on page 2-47 for more details on imprecise Data Aborts.

For all prefetch aborts the processor updates the *Instruction Fault Address Register* (IFAR) with the address of the instruction that causes the abort.

When the EA bit is set, see *c1, Secure Configuration Register* on page 3-52, all external aborts are trapped to the Secure Monitor mode, and only the Secure versions of the FSR and FAR registers are updated. In all other cases, the FAR or FSR registers are updated in the world corresponding to the state of the core that caused the aborted access. For example if the core is in Secure state, the Secure version of the FAR and FSR are updated, even in the case when the aborted access has been performed with NS rights because of the NS Attribute being Non-secure in the MMU.

6.8.1 External aborts

External memory errors are defined as those that occur in the memory system other than those that are detected by an MMU. External memory errors are expected to be extremely rare and are likely to be fatal to the running process. Examples of events that can cause an external memory error are:

- an uncorrectable parity or ECC error on a level two memory structure
- a Non- Secure access to Secure memory.

External abort on instruction fetch

Externally generated errors during an instruction prefetch are precise in nature, and are only recognized by the processor if it attempts to execute the instruction fetched from the location that caused the error. The resulting failure is reported in the Instruction Fault Status Register if no higher priority abort, including a Data Abort, has taken place.

The IFAR is updated with the address of the instruction that causes the abort.

External abort on data read/write

b/c are catastrophic

Externally generated errors during a data read or write can be imprecise. This means that R14_abt on entry into the abort handler on such an abort might not hold an address that is related to the instruction that caused the exception. Correspondingly, external aborts can be unrecoverable. See *Aborts* on page 2-45 for more details.

The Fault Address Register is updated with an invalid value, all zeros, on an imprecise external abort on a data access.

In case a precise external abort occurs during a multiple load or store operation, the FAR in the appropriate world is always updated with the base address of an AXI burst.

External abort on VA to PA translation operation

For VA to PA translation operations, the only case when an external abort can be asserted is during the page table walk.

In this case, the external abort is precise, and both the DFSR and the FAR are updated in the world, Secure or Non-secure, that generated the VA to PA translation operation. This is in addition to the standard abort mechanism occurring during VA to PA translation operations, that update the PA register of the corresponding world with the appropriate FSR encoding.

External abort on a hardware page table walk

An external abort occurring on a hardware page table access must be returned with the page table data. Such aborts are precise. The FAR is updated on an external abort on a hardware page table walk on a data access, and the IFAR is updated on an external abort on a hardware page table walk on an instruction access. The appropriate Fault Status Register indicates that this has occurred.

6.9 MMU fault checking

During the processing of a section or page, the MMU behaves differently because it is checking for faults. The MMU can generate these faults:

- *Alignment fault* on page 6-32
- *Translation fault* on page 6-32
- *Access bit fault* on page 6-32
- *Domain fault* on page 6-33
- *Permission fault* on page 6-33.

Aborts that are detected by the MMU are taken before any external memory access takes place.

Alignment fault checking is enabled by the A bit in the Control Register CP15, This bit is duplicated in the Secure and Non-secure worlds for the support of TrustZone. Alignment fault checking is independent of the MMU being enabled. Translation, Access bit, domain, and permission faults are only generated when the MMU is enabled.

The access control mechanisms of the MMU detect the conditions that produce these faults. If a fault is detected as the result of a memory access, the MMU aborts the access and signals the fault condition to the processor. The MMU retains status and address information about faults generated by data accesses in DFSR and FAR, see *Fault status and address* on page 6-34. The MMU does not retain status about faults generated by instruction fetches.

An access violation for a given memory access inhibits any corresponding external access, and an abort is returned to the processor.

6.9.1 Fault checking sequence

Figure 6-2 and Figure 6-3 on page 6-31 show the fault checking sequence for translation table managed TLB modes.

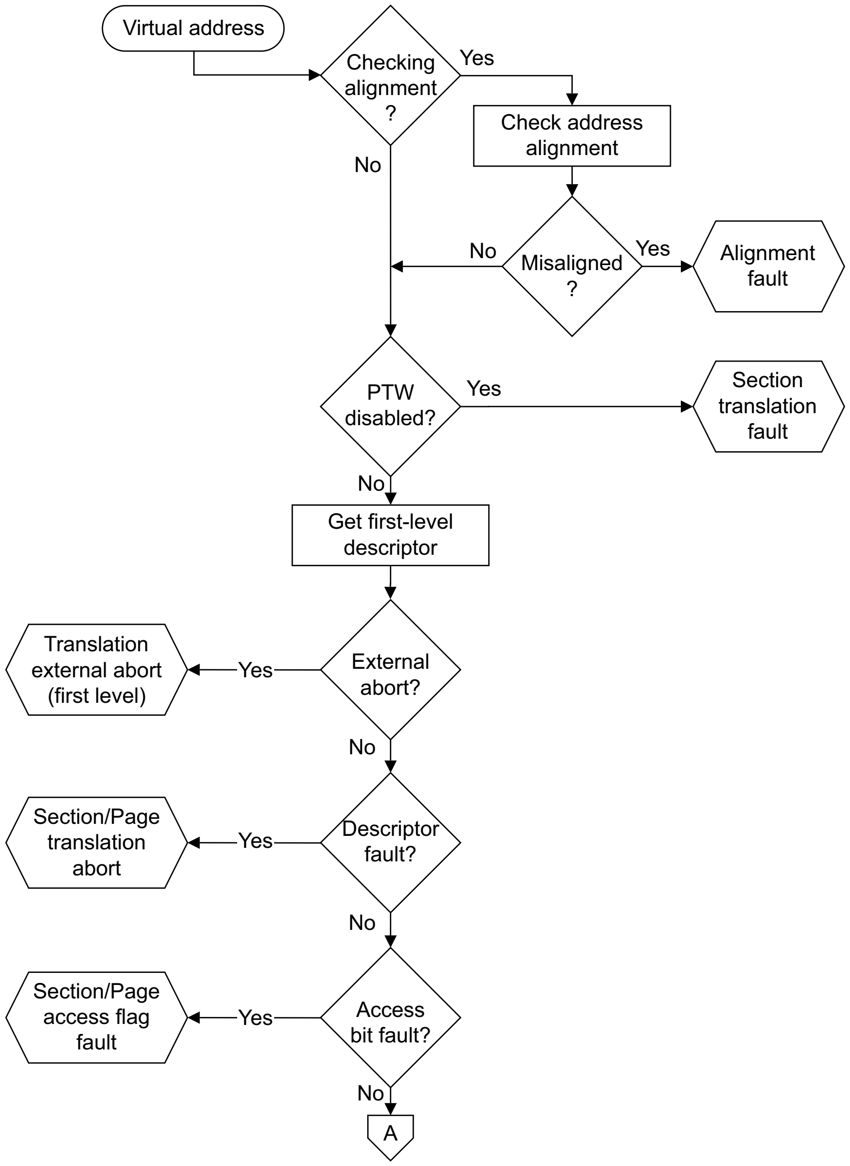

Figure 6-2 Translation table managed TLB fault checking sequence part 1

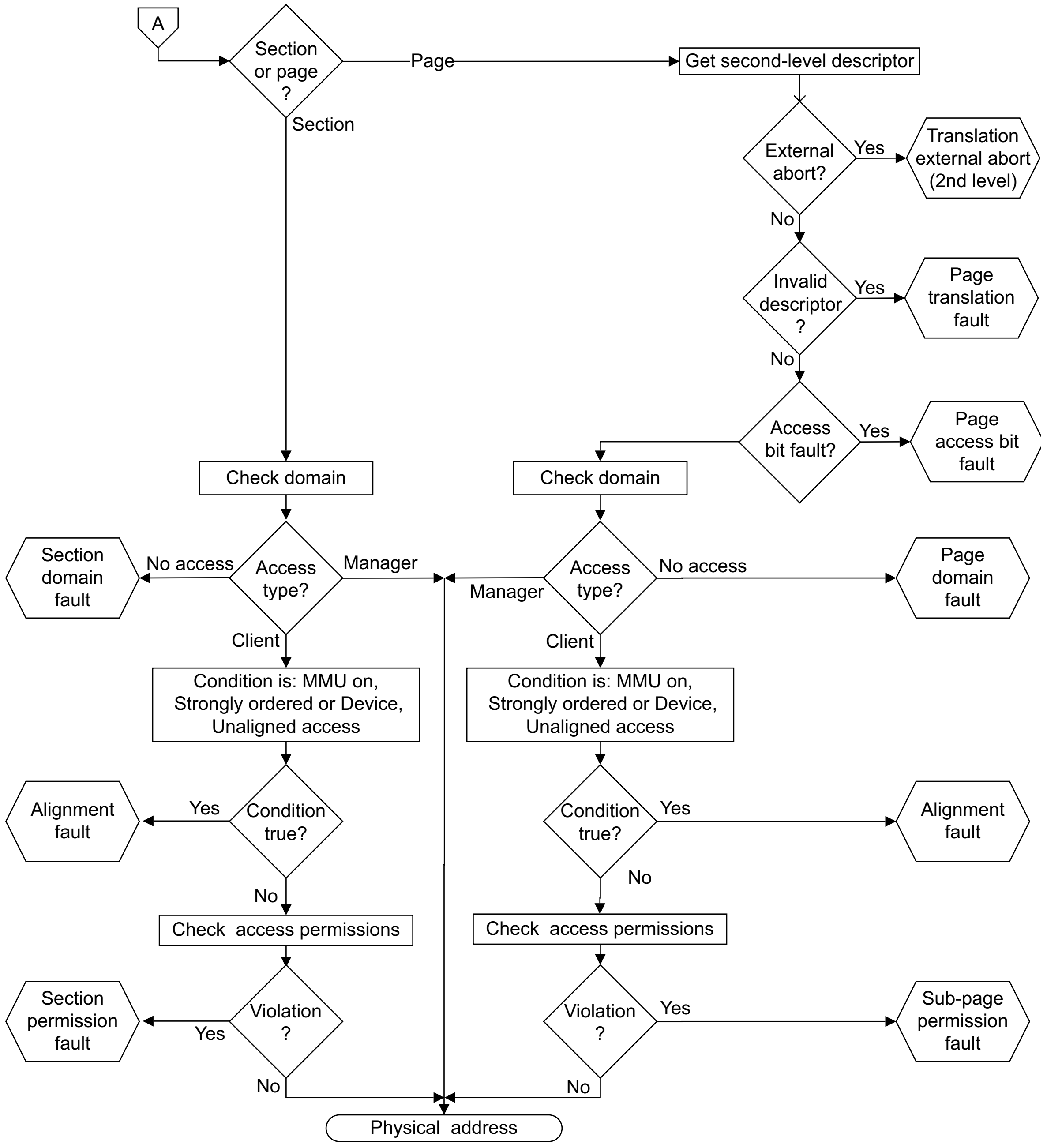

Figure 6-3 Translation table managed TLB fault checking sequence part 2

6.9.2 Alignment fault

An alignment fault occurs if the processor has attempted to access a particular data memory size at an address location that is not aligned with that size.

Operation of unaligned accesses on page 4-13 describes the conditions for generating Alignment faults.

Alignment checks are performed with the MMU both enabled and disabled.

6.9.3 Translation fault

we are going to do 1mb sections, so only get section faults.

There are two types of translation fault:

Section A section translation fault occurs if:

- The TLB tries to perform a page table walk but the page table walk is disabled by one of the PD0 or PD1 bits. For more details, see *Hardware page table translation* on page 6-36.
- The TLB fetches a first level translation table descriptor, and this first level descriptor is invalid. This is the case when bits[1:0] of this descriptor are b00 or b11.

Page A page translation fault occurs if the TLB fetches a second-level translation table descriptor and this descriptor is marked as invalid, bits [1:0] = b00.

6.9.4 Access bit fault

When the Force AP bit, see *c1, Control Register* on page 3-44 bit [29], is set then AP[0] indicates if there is an Access Bit Fault.

This bit is only taken into account when the MMU is in ARMv6 mode, that is XP=1, bit [23] in the CP15 Control register.

In the configuration XP=1 and ForceAP=1, the OS uses only bits APX and AP[1] as Access Permission bits, and AP[0] becomes an Access Bit, see *Access permissions* on page 6-11. The Access Bit records recent TLB access to a page, or section, and the OS can use this to optimize memory managements algorithms.

In the ARM1176JZF-S processor the Access Bit must be managed by the software.

Reading a page table entry into the TLB when the Access Bit is 0 causes an Access Bit fault. This fault is readily distinguished from other faults that the TLB generates and this permits fast setting of the Access Bit in software.

The processor can generate two kind of Access Bit faults:

- Section Access Bit fault, when the Access Bit, AP[0], is contained in a first level translation table descriptor
- Page Access Bit fault, when the Access Bit, AP[0], is contained in a second level translation table descriptor

The Force AP bit is banked in the Secure and Non-secure copies of the CP15 Control Register for TrustZone support.

The Force AP and XP bits are expected to be static throughout operations.

Any change in the Force AP or XP bit configuration to enable or disable the generation of Access Bit faults takes effect immediately. In the case where the TLB lookup hits an entry that was created before Access Bit faults generation was enabled, and that this entry contains AP[0]=0, then the TLB generates an Access Bit fault.

6.9.5 Domain fault

There are two types of domain fault:

Section	For a section the domain is checked when the first-level descriptor is returned.
Page	For a page the domain is checked when the second-level descriptor is returned.

For each type, the first-level descriptor indicates the domain in CP15 c3, the Domain Access Control Register, to select. If the selected domain has bit 0 set to 0 indicating either no access or reserved, then a domain fault occurs.

6.9.6 Permission fault

If the two-bit domain field returns Client, the access permission check is performed on the access permission field in the TLB entry. A permission fault occurs if the access permission check fails.

6.9.7 Debug event

When Monitor debug-mode debug is enabled an abort can be taken caused by a breakpoint on an instruction access or a watchpoint on a data access. In both cases the memory system completes the access before the abort is taken. If an abort is taken when in Monitor debug-mode debug then the appropriate FSR, IFSR or DFSR, is updated to indicate a debug abort.

If a watchpoint is taken the WFAR is set to the address that caused the watchpoint. Watchpoints are not taken precisely because following instructions can run underneath load and store multiples.

6.10 Fault status and address

Table 6-11 lists the encodings for the Fault Status Register.

Table 6-11 Fault Status Register encoding

Priority	Sources		FSR[10,3:0]	Domain	FSR[12]
Highest	Alignment		b00001	Invalid	SBZ
	TLB miss		b00000	Invalid	SBZ
	Instruction cache maintenance[a] operation fault		b00100	Invalid	SBZ
	External abort on translation	first-level	b01100	Invalid	SLVERR !DECERR
		second-level	b01110	Valid	SLVERR !DECERR
	Translation	Section	b00101	Invalid	SBZ
		Page	b00111	Valid	SBZ
	Access Bit Fault, Force AP only	Section	b00011	Valid	SBZ
		Page	b00110	Valid	SBZ
	Domain	Section	b01001	Valid	SBZ
		Page	b01011	Valid	SBZ
	Permission	Section	b01101	Valid	SBZ
		Page	b01111	Valid	SBZ
	Precise external abort		b01000	Valid	SLVERR !DECERR
	Imprecise external abort		b10110	Invalid	SLVERR !DECERR
	Parity error exception, not supported		b11000	Invalid	SBZ
Lowest	Instruction debug event		b00010	Valid	SBZ

a. These aborts cannot be signaled with the IFSR because they do not occur on the instruction side.

Note

All other Fault Status encodings are reserved.

If a translation abort occurs during a Data Cache maintenance operation by virtual address, then a Data Abort is taken and the DFSR indicates the reason. The FAR indicates the faulting address, and the IFAR indicates the address of the instruction causing the abort.

If a translation abort occurs during an Instruction Cache maintenance operation by virtual address, then a Data Abort is taken, and an Instruction Cache Maintenance Operation Fault is indicated in the DFSR. The IFSR indicates the reason. The FAR indicates the faulting address, and the IFAR indicates the address of the instruction causing the abort.

Domain and fault address information is only available for data accesses. For instruction aborts R14 must be used to determine the faulting address. You can determine the domain information by performing a TLB lookup for the faulting address and extracting the domain field.

Table 6-12 on page 6-35 lists a summary of the abort vector that is taken, and the Fault Status and Fault Address Registers that are updated for each abort type.

Table 6-12 Summary of aborts

Abort type	Abort taken	Precise?	Register updated?				
			IFSR	IFAR	DFSR	FAR	WFAR
Instruction MMU fault	Prefetch Abort	Yes	Yes	Yes	No	No	No
Instruction debug abort	Prefetch Abort	Yes	Yes	No	No	No	No
Instruction external abort on translation	Prefetch Abort	Yes[a]	Yes[a]	Yes	No	No	No
Instruction external abort	Prefetch Abort	Yes[a]	Yes[a]	Yes	No	No	No
Instruction cache maintenance operation	Data Abort	Yes	Yes	No	Yes	Yes	No
Data MMU fault	Data Abort	Yes	No	No	Yes	Yes	No
Data debug abort	Data Abort	No	No	No	Yes	Yes	Yes
Data external abort on translation	Data Abort	Yes[a]	No	No	Yes[a]	Yes[a]	No[a]
Data external abort	Data Abort	No[b]	No	No	Yes[a]	Yes	No
Data cache maintenance operation	Data Abort	Yes	No	No	Yes	Yes	No

a. When the EA bit is set, the updated FSR or FAR is always Secure.
b. Data Aborts can be precise, see *External aborts* on page 6-27 for more details.

6.11 Hardware page table translation

The processor MMU implements the hardware page table walking mechanism from ARMv4 and ARMv5 cached processors with the exception of the removal of the fine page table descriptor and the addition the page table walk disable bits in the TTB Control register.

The processor implements the page table walk disable feature. Two bits, PD0 and PD1, are implemented in the TTB Control register. These bits are banked for the Secure and Non-secure worlds for the support of TrustZone.

Each time a TLB miss occurs, the TLB computes the parameters for an automatic hardware page table walk. The address of the page table walk is computed from TTB0 or TTB1, see *First-level descriptor address* on page 6-43. If the address is computed with TTB0, and the PD0 bit is set in the TTB Control register of the corresponding world, or if the address is computed using TTB1 and the PD1 bit is set, then the processor does not perform the automatic hardware page table walk, and it generates a Section translation fault instead.

With this feature, only a small portion of the memory can be mapped in one world, for example the Secure world, if the code that runs in this world is expected to be small. This gives the system a simple way to avoid using a lot of memory to store full page tables.

When hardware page table walks are not disabled, the processor performs the page table walk in the usual way. A hardware page table walk occurs whenever there is a TLB miss. Processor hardware page table walks do not cause a read from the level one Unified/Data Cache. or the TCM. The P, RGN, S, and C bits in the Translation Table Base Registers determine the memory region attributes for the page table walk.

Two formats of page tables are supported:

- A backwards-compatible format supporting subpage access permissions. These have been extended so that certain page table entries support extended region types and with the NS Attribute bit for TrustZone.
- ARMv6 format, not supporting sub-page access permissions, but with support for ARMv6 MMU features. The NS Attribute bit for TrustZone has also been added. These features are:
  - extended region types
  - global and process specific pages
  - more access permissions
  - marking of Shared and Non-Shared regions
  - marking of Execute-Never regions.

we aren't doing worlds

Additionally, two translation table base registers are provided in each world. On a TLB miss, the Translation Table Base Control Register, CP15 c2 that is also duplicated in each world, and the top bits of the virtual address determine if the first or second translation table base is used. See *c2, Translation Table Base Control Register* on page 3-60 for details. The first-level descriptor indicates whether the access is to a section or to a page table. If the access is to a page table, the processor MMU fetches a second-level descriptor.

A page table holds 256 32-bit entries 4KB in size. You can determine the page type by examining bits [1:0] of the second-level descriptor. For both first and second level descriptors if bits [1:0] are b00, the associated virtual addresses are unmapped, and attempts to access them generate a translation fault. Software can use bits [31:2] for its own purposes in such a descriptor, because they are ignored by the hardware. Where appropriate, ARM Limited recommends that bits [31:2] continue to hold valid access permissions for the descriptor.

For both level 1 and level 2 page table walks, the processor performs external accesses with Secure or Non-secure rights depending on the Secure or Non-secure state of the MMU request that causes the page table walk. This ensures that Secure translation table descriptors are always fetched from a Secure memory, and that Non-secure translation table descriptors are always fetched from Non-secure memory.

6.11.1 Backwards-compatible page table translation subpage AP bits enabled

When the CP15 Control Register c1 bit 23 is set to 0, the subpage AP bits are enabled and the page table formats are backwards-compatible with ARMv4 and ARMv5 MMU architectures. This bit is duplicated as Secure and Non-secure versions so that the system can enable or disable subpages independently in each world.

ignore

All mappings are treated as global, and executable, XN = 0. All Normal memory is Non-Shared. Device memory can be Shared or Non-Shared as determined by the TEX bits and the C and B bits. For large and small pages, there can be four subpages defined with different access permissions. For a large page, the subpage size is 16KB and is accessed using bits [15:14] of the page index of the virtual address. For a small page, the subpage size is 1KB and is accessed using bits [11:10] of the page index of the virtual address.

The use of subpage AP bits where AP3, AP2, AP1, and AP0 contain different values is deprecated.

Backwards-compatible page table format

Figure 6-4 shows a backwards-compatible format first-level descriptor.

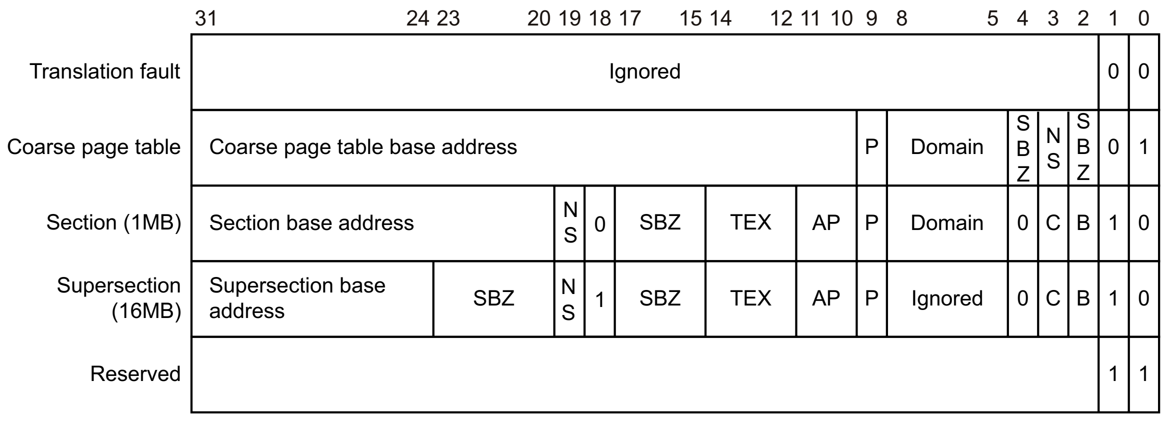

Figure 6-4 Backwards-compatible first-level descriptor format

If the P bit is supported and set for the memory region, it indicates to the system memory controller that this memory region has ECC enabled. ARM1176JZF-S processors do not support the P bit.

When bits [1:0] of the first-level descriptor are b01, the descriptor points to a second-level page table, called a *Coarse page table*. Figure 6-5 on page 6-38 shows a backwards-compatible format second-level descriptors.

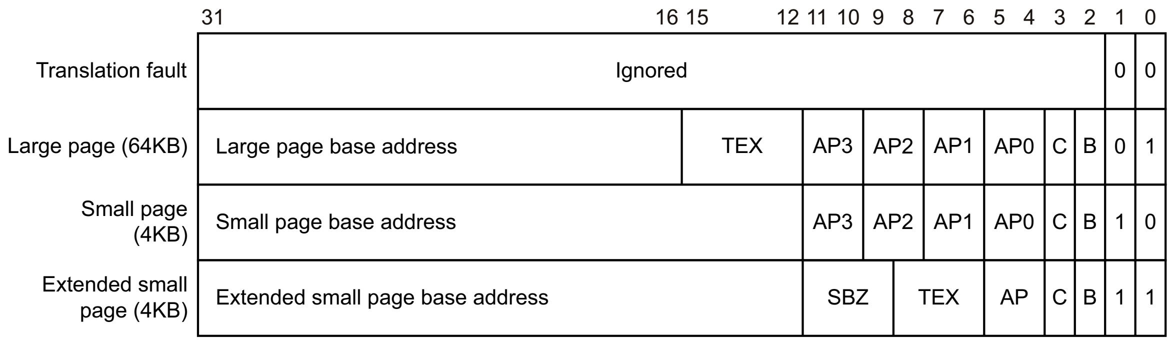

Figure 6-5 Backwards-compatible second-level descriptor format

For extended small page table entries without a TEX field you must use the value b000. For details of TEX encodings see *C and B bit, and type extension field encodings* on page 6-14.

Note

For any Supersection description in a first-level page table, and any Large page description in a second-level page table:

- you must repeat the description in 16 consecutive page table locations
- the first description must occur on a 16-word boundary

For more information see the *ARM Architecture Reference Manual*.

Figure 6-6 shows an overview of the section, supersection, and page translation process using backwards-compatible descriptors.

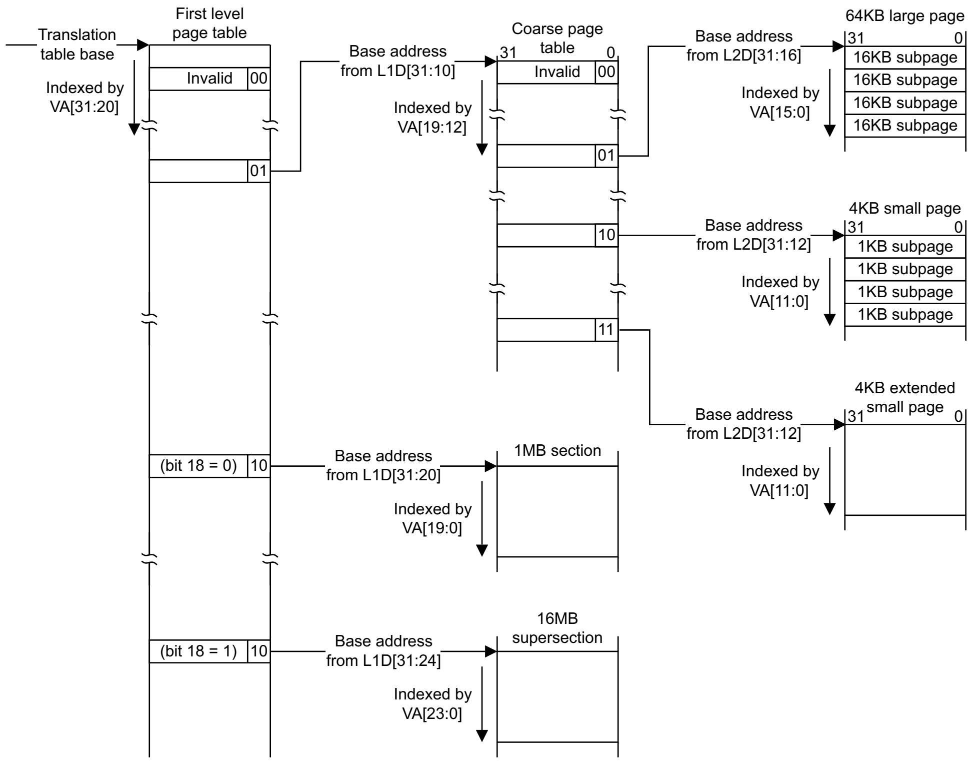

Figure 6-6 Backwards-compatible section, supersection, and page translation

6.11.2 ARMv6 page table translation subpage AP bits disabled

When the CP15 Control Register c1 Bit 23 is set to 1 in the corresponding world, the subpage AP bits are disabled and the page tables have support for ARMv6 MMU features. Four new page table bits are added to support these features:

- The Not-Global (nG) bit, determines if the translation is marked as global (0), or process-specific (1) in the TLB. For process-specific translations the translation is inserted into the TLB using the current ASID, from the ContextID Register, CP15 c13.
- The Shared (S) bit, determines if the translation is for Non-Shared (0), or Shared (1) memory. This only applies to Normal memory regions. Device memory can be Shared or Non-Shared as determined by the TEX bits and the C and B bits.
- The Execute-Never (XN) bit, determines if the region is Executable (0) or Not-executable (1).
- Three access permission bits. The access permissions extension (APX) bit, provides an extra access permission bit.

All ARMv6 page table mappings support the TEX field.

ARMv6 page table format

With the sub-pages enabled or not, all first level descriptors have been enhanced with the addition of the NS Attribute bit to enable the support of TrustZone.

Figure 6-7 shows the format of an ARMv6 first-level descriptor when subpages are disabled.

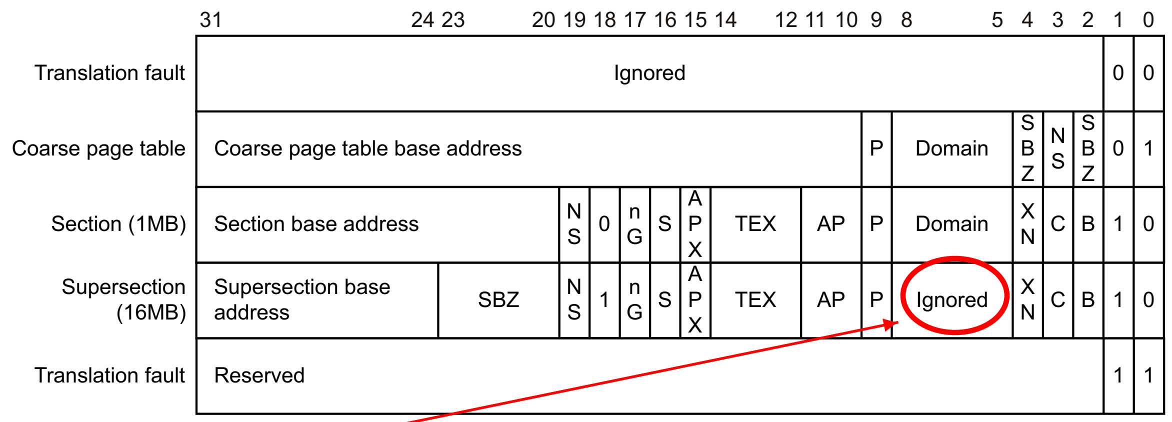

Figure 6-7 ARMv6 first-level descriptor formats with subpages disabled

think can only
do for OS pages

If the P bit is supported and set for the memory region, it indicates to the system memory controller that this memory region has ECC enabled. ARM1176JZF-S processors do not support the P bit. In addition to the invalid translation, bits [1:0] = b00, translations for the reserved entry, bits [1:0] = b11, result in a translation fault.

As shown in Figure 6-7, bits [1:0] of a level 1 page table entry determine the type of the entry:

Bits [1:0] == b00

Translation fault.

Bits [1:0] == b01

The entry points to a second-level page table, called a *Coarse page table*. Figure 6-8 on page 6-40 shows the formats of the possible entries in the Coarse page table.

Bits [1:0] == b10

The entry points to a either a 1MB *Section* of memory or a 16MB *Supersection* of memory. Bit [18] of the descriptor selects between a Section and a Supersection. For details of supersections see *Supersections* on page 6-6.

Note

You must repeat any Supersection description in 16 consecutive page table locations, with the first description occurring on a 16-word boundary. For more information see the *ARM Architecture Reference Manual*.

Bits [1:0] == b11

Reserved.

Figure 6-8 shows the format of an ARMv6 second-level descriptors.

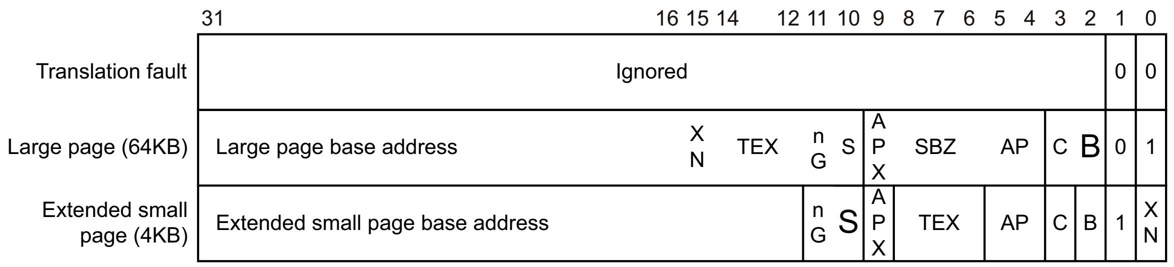

Figure 6-8 ARMv6 second-level descriptor format

As shown in Figure 6-8, bits [1:0] of a second-level descriptor determine the type of the descriptor:

Bits [1:0] == b00

Translation fault.

Bits [1:0] == b01

The entry points to a 64KB *Large page* in memory.

Note

You must repeat any Large page description in 16 consecutive page table locations, with the first description occurring on a 16-word boundary. For more information see the *ARM Architecture Reference Manual*.

Bits [1:0] == b1x

The entry points to a 4KB *Extended small page* in memory.

Bit [0] of the entry is the XN bit for the entry.

Figure 6-9 on page 6-41 shows an overview of the section, supersection, and page translation process using ARMv6 descriptors.

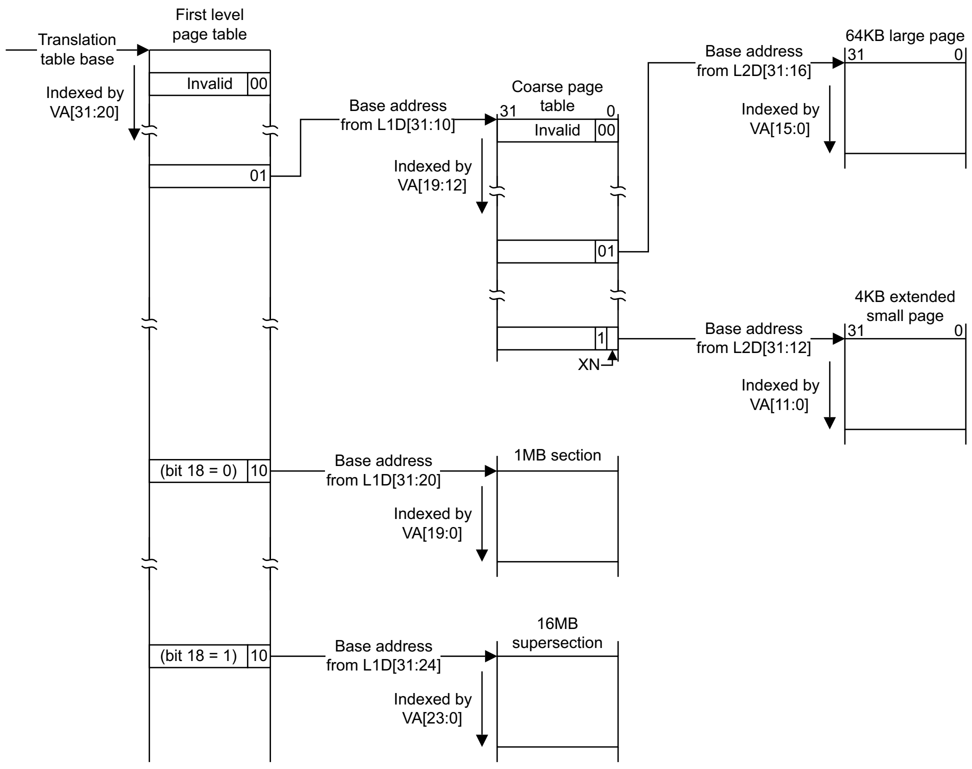

Figure 6-9 ARMv6 section, supersection, and page translation

6.11.3 Restrictions on page table mappings page coloring

i believe n/a to 1mb section pages

The processor uses virtually indexed, physically addressed caches. To prevent alias problems where cache sizes greater than 16KB have been implemented, you must restrict the mapping of pages that remap virtual address bits [13:12].

- for the Instruction Cache, the Isize P bit, bit[11], of the Cache Type Register CP15 c0, indicates if this is necessary
- for the Data Cache, the Dsize P bit, bit[23], of the Cache Type Register CP15 c0, indicates if this is necessary.

See *c0, Cache Type Register* on page 3-21 for more information.

This restriction, referred to as *page coloring*, enables the virtual address bits[13:12] to be used to index into the cache without requiring hardware support to avoid alias problems.

For pages marked as Non-Shared, if bit 11 or bit 23 of the Cache Type Register is set, the restriction applies to pages that remap virtual address bits [13:12] and might cause aliasing problems when 4KB pages are used. To prevent this you must ensure the following restrictions are applied:

1. If multiple virtual addresses are mapped onto the same physical address then for all mappings of bits [13:12] the virtual addresses must be equal and the same as bits [13:12] of the physical address. The same physical address can be mapped by TLB entries of different page sizes, including page sizes over 4KB. Imposing this requirement on the virtual address is called page coloring.

2. Alternatively, if all mappings to a physical address are of a page size equal to 4KB, then the restriction that bits [13:12] of the virtual address must equal bits [13:12] of the physical address is not necessary. Bits [13:12] of all virtual address aliases must still be equal.

There is no restriction on the more significant bits in the virtual address equalling those in the physical address.

Avoiding the page coloring restriction

The processor provides the ability to restrict the cache size to 16KB so that software does not have to support the page coloring restriction on mapping, see CZ bit in *c1, Auxiliary Control Register* on page 3-48.

——— **Note** ———

Setting the CZ flag in the CP15 Auxiliary Control Register does not affect the contents of the CP15 Cache Type Register. However, when the CZ flag is set, all caches are limited to 16KB, even if a larger cache size is specified in the CP15 Cache Type Register.

6.12 MMU descriptors

To support sections and pages, the processor MMU uses a two-level descriptor definition. The first-level descriptor indicates whether the access is to a section or to a page table. If the access is to a page table, the processor MMU determines the page table type and fetches a second-level descriptor.

6.12.1 First-level descriptor address

The ARM1176 contains:

- two Translation Table Base Registers, TTBR0 and TTBR1
- one Translation Table Base Control Register (TTBCR).

On a TLB miss, the top bits of the modified virtual address determine whether the first or second Translation Table Base is used. Figure 6-10 on page 6-44 shows the creation of a first-level descriptor address.

The expected use of two translation tables is to reduce the cost of OS context switches by enabling the OS, and each individual task or process, to have its own pagetable without consuming much memory.

In this model, the virtual address space is divided into two regions:

- `0x0` -> 1<<(32-N) that TTBR0 controls
- 1<<(32-N) -> 4GB that TTBR1 controls.

The value of N is set in the TTBCR. If N is zero, then TTBR0 is used for all addresses, and that gives legacy v5 behavior. If N is not zero, the OS and memory mapped IO are located in the upper part of the memory map, TTBR1, and the tasks or processes all occupy the same virtual address space in the lower part of the memory, TTBR0.

The TTBCR, TTBR0, and TTBR1 registers used for this process are banked. Depending on the state of the MMU requests that cause a page table walk, either Secure or Non-secure registers are used.

The translation table that TTBR0 points to can be truncated because it must only cover the first 1<<(32-N) bytes of memory. The first entry always corresponds to address `0x0`, so this mechanism is more efficient if processes start at a low virtual address such as `0x0` or `0x8000`. Table 6-13 lists the translation table size.

Table 6-13 Translation table size

N	Upper boundary	Translation table 0 size
0	4GB	16KB, 4096 entries, v5 behavior, TTBR1 not used.
1	2GB	8KB, 2048 entries
2	1GB	4KB, 1024 entries
3	512MB	2KB, 512 entries
4	256MB	1KB, 256 entries
5	128MB	512B, 128 entries
6	64MB	256B, 64 entries
7	32MB	128B, 32 entries we should do this one

The OS can maintain a different pagetable for each process, and update TTRB0 on a context switch. Using a truncated pagetable means that much less space is required to store the individual process page tables. Different processes can have different size pagetables, that is, different values of N, by updating the TTBCR during the context switch.

It is not required that the OS pagetables that TTBR1 points to are updated on a context switch. Figure 6-10 shows how to create a first level descriptor address.

The PD0 and PD1 bits in TTBCR can be used to prevent pagetable walks from either TTBR. In particular, disabling walks from TTBR1 and setting TTBR0 to the address of a truncated translation table can minimize the overhead otherwise incurred in unused translation table entries.

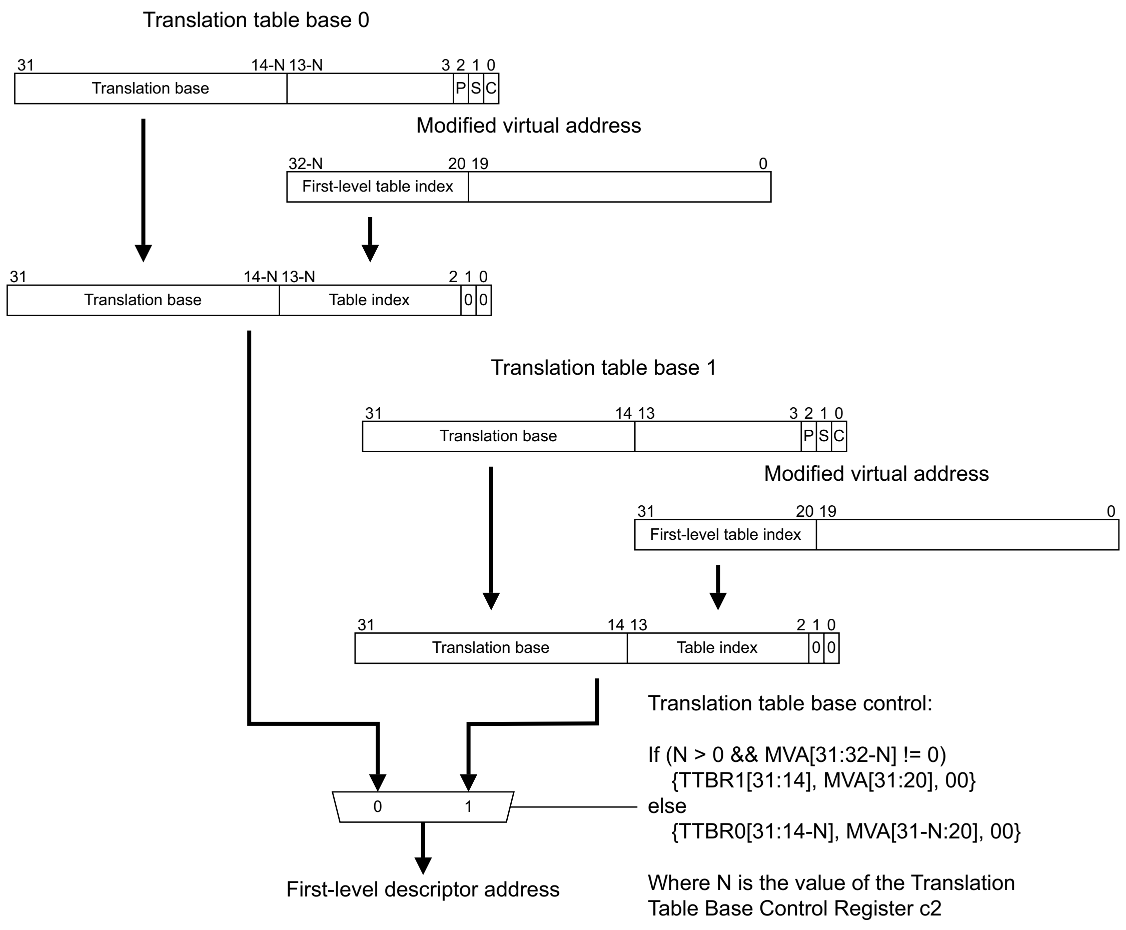

Figure 6-10 Creating a first-level descriptor address

6.12.2 First-level descriptor

Using the first-level descriptor address, a request is made to external memory. This returns the first-level descriptor. By examining bits [1:0] of the first-level descriptor, the access type is indicated as Table 6-14 lists.

Table 6-14 Access types from first-level descriptor bit values

Bit values	Access type
b00	Translation fault
b01	Page table base address
b10	Section base address
b11	Reserved, results in translation fault

First-level translation fault

If bits [1:0] of the first-level descriptor are b00 or b11, a translation fault is generated. This generates an abort to the processor, either a Prefetch Abort for the instruction side or a Data Abort for the data side, see *MMU fault checking* on page 6-29.

If the first level descriptor describes a section or supersection when the Force AP bit is set and the MMU is in ARMv6 mode, Access bit faults might be generated if AP[0]=0.

First-level page table address

If bits [1:0] of the first-level descriptor are b01, then a page table walk is required. *Second-level page table walk* on page 6-47 describes this process.

First-level section base address

If bits [1:0] of the first-level descriptor are b10, a request to a section memory block has occurred. Figure 6-11 on page 6-46 shows the translation process for a 1MB section using ARMv6 format, AP bits disabled.

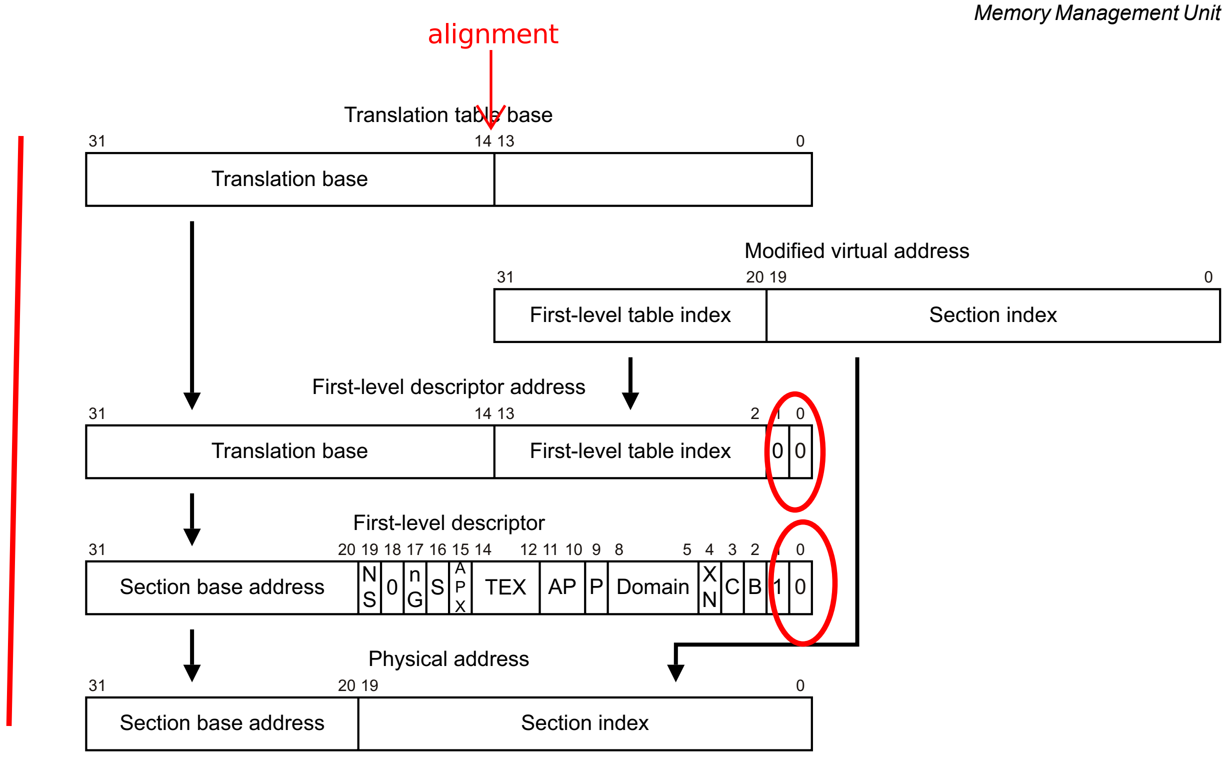

Figure 6-11 Translation for a 1MB section, ARMv6 format

Following the first-level descriptor translation, the physical address is used to transfer to and from external memory the data requested from and to the processor. This is done only after the domain and access permission checks are performed on the first-level descriptor for the section. *Memory access control* on page 6-11 describes these checks.

Figure 6-12 shows the translation process for a 1MB section using backwards-compatible format, AP bits enabled.

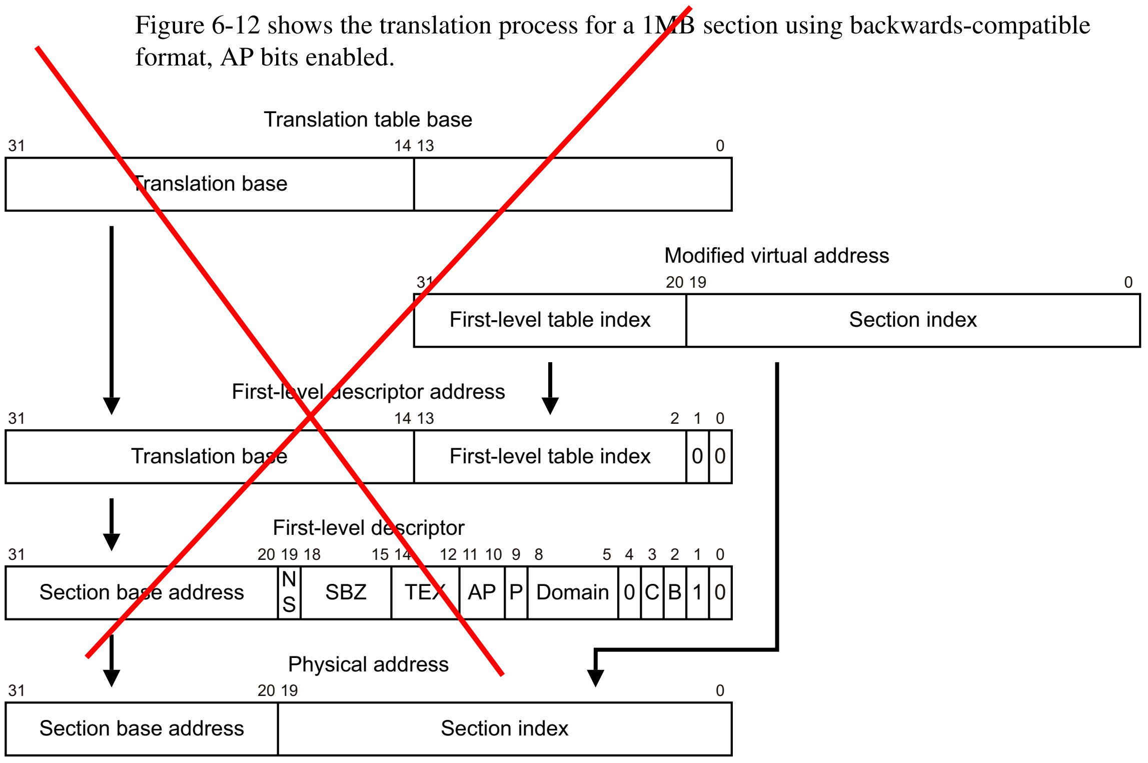

Figure 6-12 Translation for a 1MB section, backwards-compatible format

6.12.3 Second-level page table walk

If bits [1:0] of the first-level descriptor bits are b01, then a page table walk is required. The MMU requests the second-level page table descriptor from external memory. Figure 6-13 shows how the second-level page table address is generated.

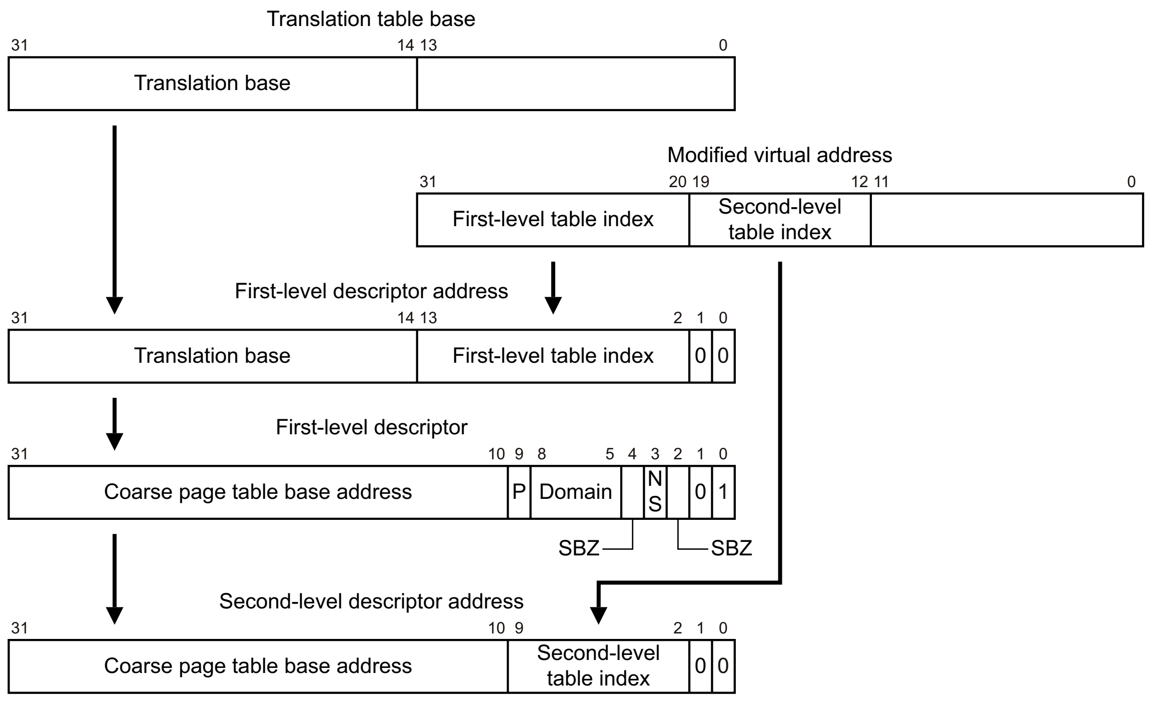

Figure 6-13 Generating a second-level page table address

When the page table address is generated, a request is made to external memory for the second-level descriptor.

By examining bits [1:0] of the second-level descriptor, the access type is indicated as Table 6-15 lists.

Table 6-15 Access types from second-level descriptor bit values

Descriptor format	Bit values	Access type
Both	b00	Translation fault
Backwards-compatible	b01	64KB large page
ARMv6	b01	64KB large page
Backwards- compatible	b10	4KB small page
ARMv6	b1XN	4KB extended small page
Backwards- compatible	b11	4KB extended small page

Second-level translation fault

If bits [1:0] of the second-level descriptor are b00, then a translation fault is generated. This generates an abort to the processor, either a Prefetch Abort for the instruction side or a Data Abort for the data side, see *MMU fault checking* on page 6-29.

If the second level descriptor describes a large page, a small page, or an extended small page when the Force AP bit is set and the MMU is in ARMv6 mode, Access bit faults might be generated if AP[0]=0.

Second-level large page base address

If bits [1:0] of the second-level descriptor are b01, then a large page table walk is required. Figure 6-14 shows the translation process for a 64KB large page using ARMv6 format, AP bits disabled.

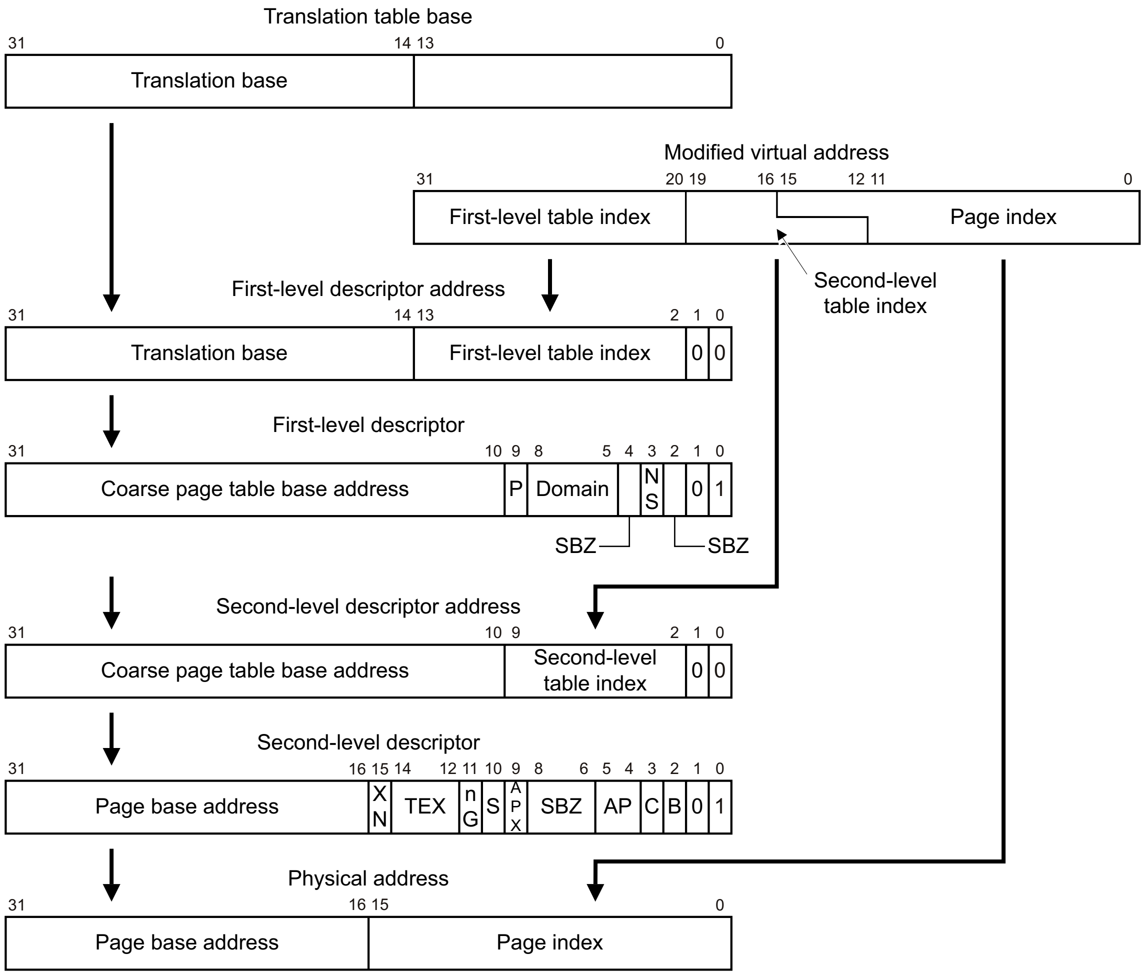

Figure 6-14 Large page table walk, ARMv6 format

Figure 6-15 on page 6-49 shows the translation process for a 64KB large page, or a 16KB large page subpage, using backwards-compatible format, AP bits enabled.

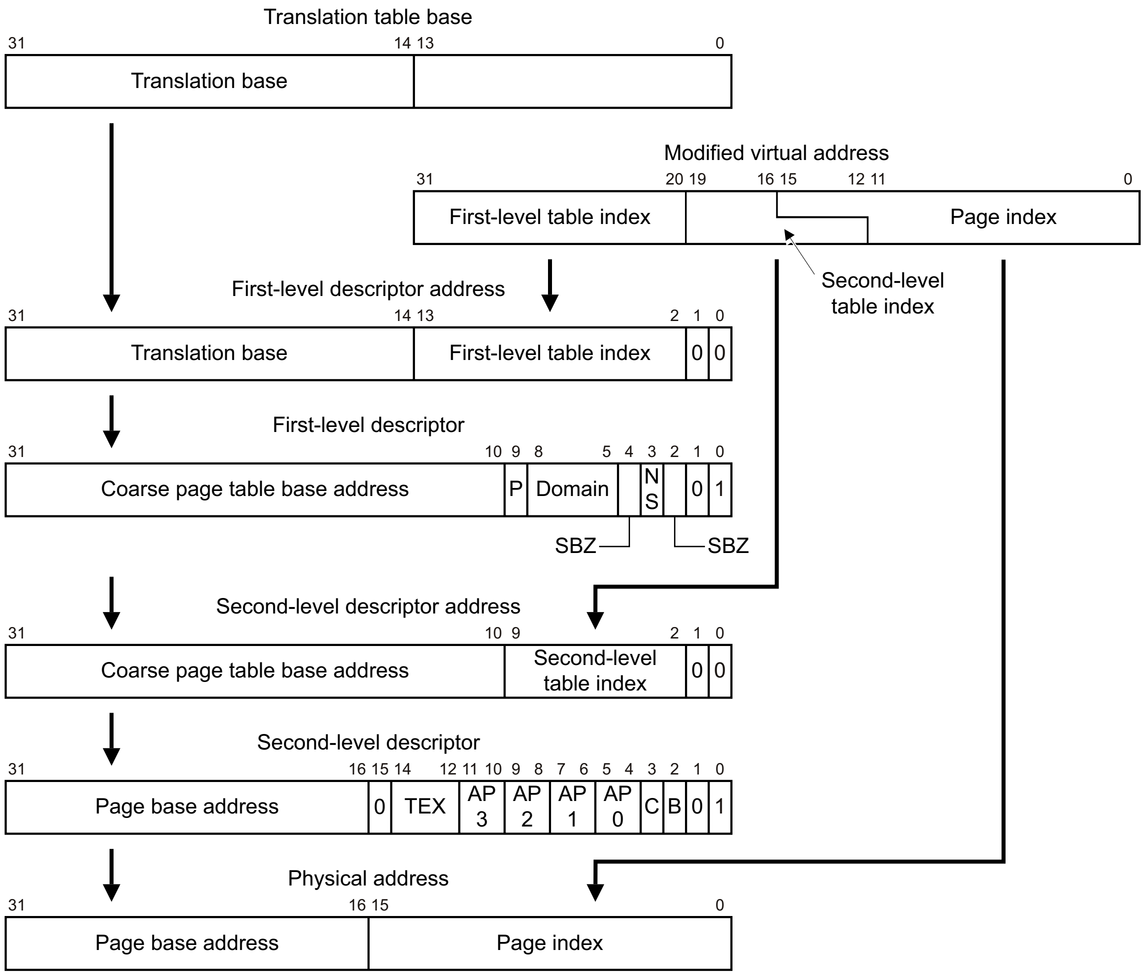

Figure 6-15 Large page table walk, backwards-compatible format

Using backwards-compatible format descriptors, the 64KB large page is generated by setting all of the AP bit pairs to the same values, AP3=AP2=AP1=AP0. If any one of the pairs are different, then the 64KB large page is converted into four 16KB large page subpages. The subpage access permission bits are chosen using the virtual address bits [15:14].

Second-level small page table walk

If bits [1:0] of the second-level descriptor are b10 for backwards-compatible format, then a small page table walk is required.

Figure 6-16 on page 6-50 shows the translation process for a 4KB small page or a 1KB small page subpage using backwards-compatible format descriptors, AP bits enabled.

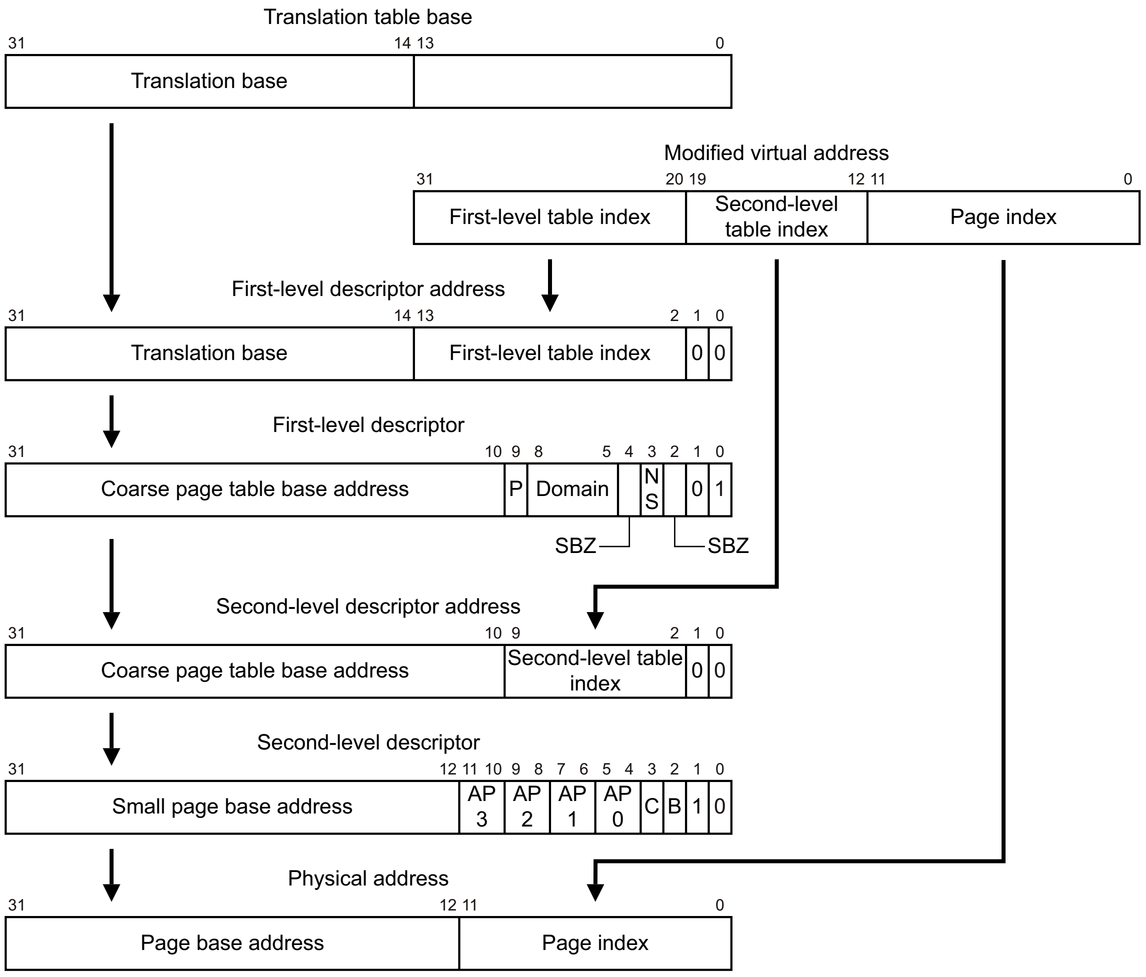

Figure 6-16 4KB small page or 1KB small subpage translations, backwards-compatible format

Using backwards-compatible descriptors, the 4KB small page is generated by setting all of the AP bit pairs to the same values, AP3=AP2=AP1=AP0. If any one of the pairs are different, then the 4KB small page is converted into four 1KB small page subpages. The subpage access permission bits are chosen using the virtual address bits [11:10].

Second-level extended small page table walk

If bits [1:0] of the second-level descriptor are b1XN for ARMv6 format descriptors, or b11 for backwards-compatible descriptors, then an extended small page table walk is required. Figure 6-17 on page 6-51 shows the translation process for a 4KB extended small page using ARMv6 format descriptors, AP bits disabled.

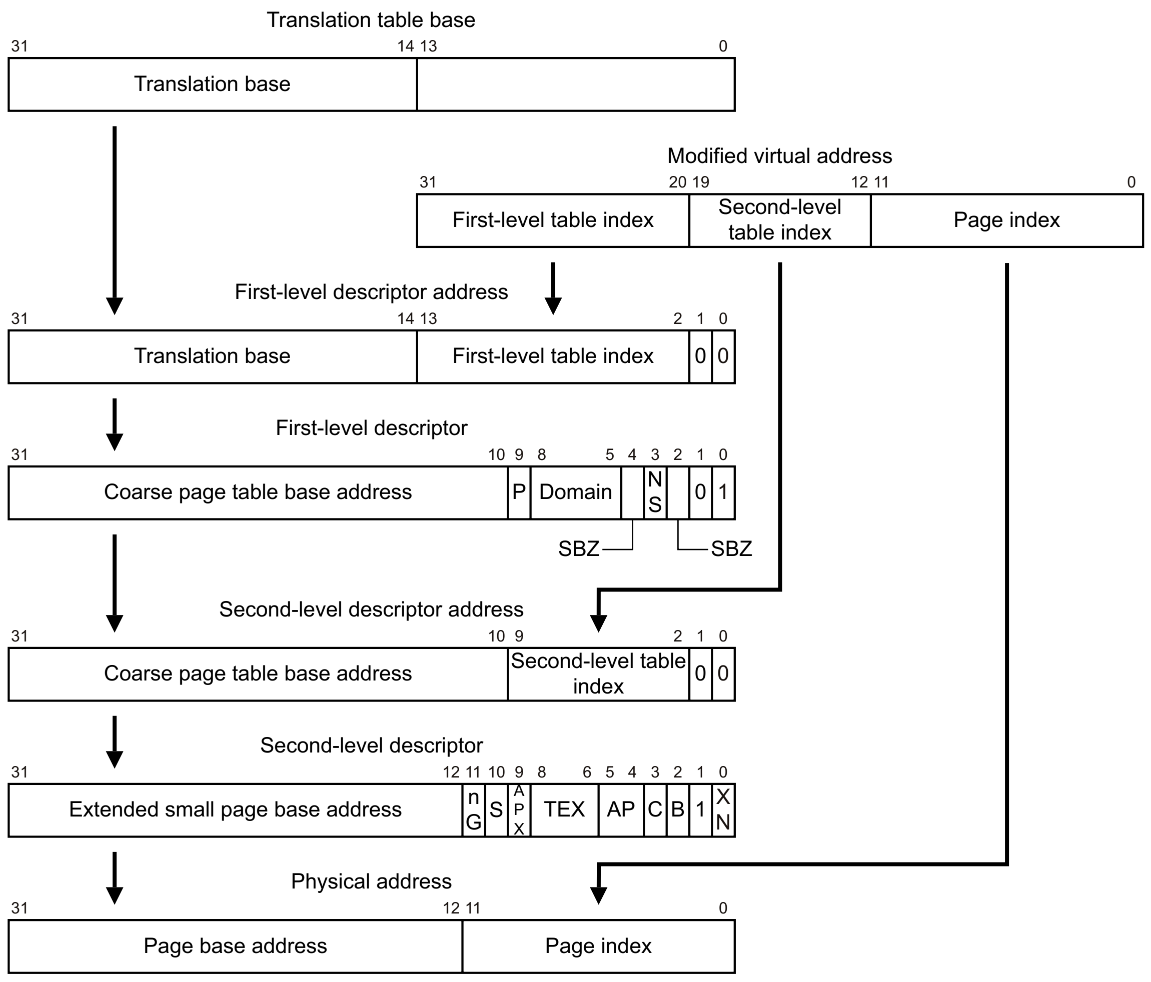

Figure 6-17 4KB extended small page translations, ARMv6 format

Figure 6-18 on page 6-52 shows the translation process for a 4KB extended small page or a 1KB extended small page subpage using backwards-compatible format descriptors, AP bits enabled.

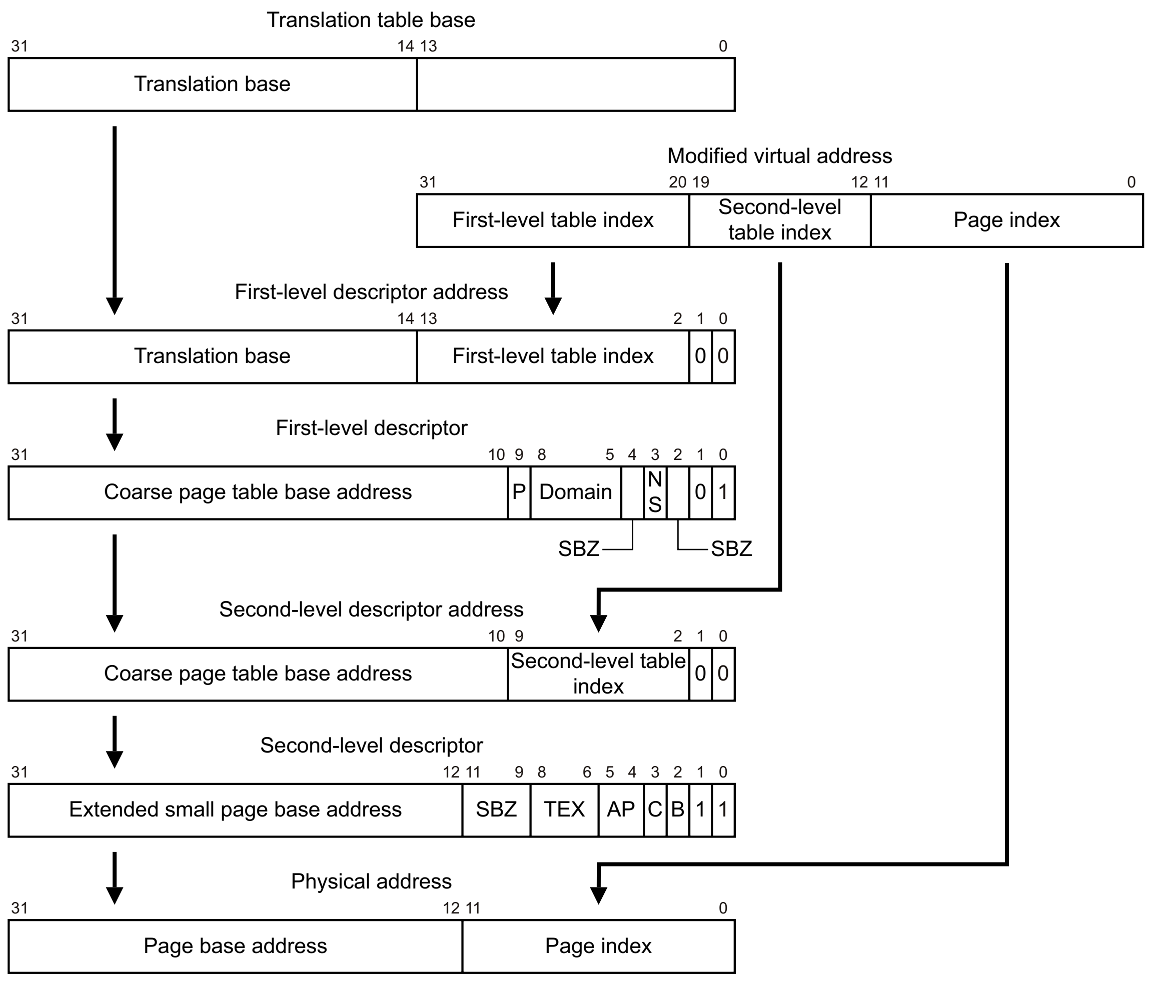

Figure 6-18 4KB extended small page or 1KB extended small subpage translations, backwards-compatible format

Using backwards-compatible descriptors, the 4KB extended small page is generated by setting all of the AP bit pairs to the same values, AP3=AP2=AP1=AP0. If any one of the pairs are different, then the 4KB extended small page is converted into four 1KB extended small page subpages. The subpage access permission bits are chosen using the virtual address bits [11:10].

6.13 MMU software-accessible registers

The MMU is controlled by the system control coprocessor, CP15 registers. Table 6-16, lists the system control processor registers and references to their detailed descriptions. For more information on the system control coprocessor, see Chapter 3 *System Control Coprocessor*.

Table 6-16 CP15 register functions

Register	Cross reference
TLB Type Register	*c0, TLB Type Register* on page 3-25
Control Register	*c1, Control Register* on page 3-44
Non-Secure Access Control Register	*c1, Non-Secure Access Control Register* on page 3-55
Translation Table Base Register 0	*c2, Translation Table Base Register 0* on page 3-57
Translation Table Base Register 1	*c2, Translation Table Base Register 1* on page 3-59
Translation Table Base Control Register	*c2, Translation Table Base Control Register* on page 3-60
Domain Access Control Register	*c3, Domain Access Control Register* on page 3-63
Data Fault Status Register (DFSR)	*c5, Data Fault Status Register* on page 3-64
Instruction Fault Status Register (IFSR)	*c5, Instruction Fault Status Register* on page 3-66
Fault Address Register (FAR)	*c6, Fault Address Register* on page 3-68 and *MMU fault checking* on page 6-29
Instruction Fault Address Register (IFAR)	*c6, Instruction Fault Address Register* on page 3-69 and *MMU fault checking* on page 6-29
TLB Operations Register	*c8, TLB Operations Register* on page 3-86
TLB Lockdown Register	*c10, TLB Lockdown Register* on page 3-100
Primary Region Remap Register	*c10, Memory region remap registers* on page 3-101
Normal Memory Remap Register	*c10, Memory region remap registers* on page 3-101
FCSE PID Register	*c13, FCSE PID Register* on page 3-126
ContextID Register	*c13, Context ID Register* on page 3-128.
Peripheral Port Remap Register	*c15, Peripheral Port Memory Remap Register* on page 3-130
TLB Lockdown Index Register	*c15, TLB lockdown access registers* on page 3-149
TLB Lockdown VA Register	*c15, TLB lockdown access registers* on page 3-149
TLB Lockdown PA Register	*c15, TLB lockdown access registers* on page 3-149
TLB Lockdown Attributes Register	*c15, TLB lockdown access registers* on page 3-149

Note

All the CP15 MMU registers, except CP15 c8, contain state that you read from using MRC instructions and write to using MCR instructions. Registers c5 and c6 are also written by the MMU. Reading CP15 c8 results in an Undefined exception.

The debug control coprocessor CP14 also influences the MMU when in Debug state. Table 6-17 lists the registers that affect the MMU.

Table 6-17 CP14 register functions

Register	Cross reference
Debug State MMU Control Register	*CP14 c11, Debug State MMU Control Register* on page 13-23
Debug State Cache Control Register	*CP14 c10, Debug State Cache Control Register* on page 13-23