3. Memory.

NXSOL-OJT-2024

Exported on 02/27/2024

Table of Contents

1 Memory

1.1 About the VMSA

- **Virtual Memory System**: VMSA involves dynamically allocating memory and mapping it to physical addresses via a Memory Management Unit (MMU).
- **Translation Table Walk**: This is the process of converting virtual addresses to physical ones, searching page tables.
- **Translation Lookaside Buffers (TLBs)**: TLBs store the results of recent translation table lookups.(not the entire contents) If a virtual address translation is in the TLB, the MMU can use it directly, avoiding a full table walk.

1.1.1 Enhancements in ARMv6 (VMSA6)

VMSA has been improved in ARMv6 to avoid the need for TLB invalidation on a context switch, thus improving performance.
The key enhancements include:

- Global mappings for applications, avoiding the need for TLB flushes on most context switches.
- New memory types for efficient memory management.
- Enhanced memory properties in TLB entries for better access control.

1.2 Memory Access Sequence

When the ARM CPU generates a memory access, the MMU performs a lookup for a modified virtual address in a TLB. This includes checking the current ASID in ARMv6 implementations.

- **Modified virtual address**: refers to a combination of the 32-bit virtual address along with additional bits for the Application Space Identifier (ASID), which provides a way to separate the address spaces of different processes.
- **ASID (Address Space Identifier)**: Uniquely identifies a process's virtual memory space. It is used to distinguish between memory pages with the same virtual address but used by different tasks.
- Instruction fetches use the *instruction TLB* and data accesses use the *data TLB*.
- If no appropriate TLB entry is found, a translation table walk is performed by hardware.

1.2.1 *Note*

- Modified virtual address translations are globally mapped from ARMv6, considering the 32-bit modified virtual address plus ASID values when non-global address is accessed.
- The FCSE mechanism is deprecated in ARMv6. Concurrent use of FCSE and ASID leads to unpredictable behavior.

1.2.2 TLB Match Process

Each TLB entry contains a modified virtual address, a page size, a physical address, and memory properties. It's associated with an application space identifier (ASID) or marked as global.

- A match occurs if the higher order bits of the modified virtual address match and the ASID is the same.
- The behavior of a TLB is unpredictable if multiple entries match at any time.

1.3 Enabling and disabling the MMU

1.3.1 Controlling the MMU:

- The MMU is enabled and disabled by setting or clearing the M bit (bit[0]) in register 1 of the System Control coprocessor (CP15).
- It's disabled by default upon reset.

1.3.2 Memory Access Behavior When MMU is Disabled:

1.3.2.1 Prior knowledge

Strongly ordered memory: Memory accesses to Strongly Ordered memory regions are strictly ordered. This means that any access (read or write) to a Strongly Ordered memory location must be completed before any subsequent access can begin. This order is maintained across all processors in a multi-core system.

1.3.2.2 Data Accesses:

- Treated as uncacheable and strongly ordered.
- Unexpected data cache hit behavior varies depending on implementation.

1.3.2.3 Instruction Accesses:

Behavior depends on cache architecture:

- Harvard cache: Cacheable or non-cacheable based on I bit (bit[12]) of CP15 register 1.
- Unified cache: Always non-cacheable.

1.3.2.4 Explicit Accesses:

- Strongly ordered.
- W bit (bit[3], write buffer enable) of CP15 register 1 is ignored.

1.3.2.5 Memory Access Permissions and Aborts:

- No permission checks are performed.
- No aborts are generated by the MMU.

1.3.2.6 Address Mapping:

- Flat address mapping: Physical address = modified virtual address.

1.3.2.7 FCSE PID:

- Should be zero (SBZ) when MMU is disabled.
- Clearing FCSE PID is recommended before disabling MMU.

1.3.2.8 Cache and TLB Operations:

- Cache CP15 operations work regardless of MMU state, but use flat mapping when MMU is disabled.
- CP15 TLB invalidate operations work regardless of MMU state.

1.3.2.9 Other Operations:

- Instruction and data prefetch operations work normally.
- Accesses to TCMs (Tightly Coupled Memories) work normally if TCM is enabled.

1.3.3 Prerequisites for Enabling MMU:

- **CP15 Register Programming**: Set up relevant CP15 registers, including translation tables.
- **Instruction Cache Handling**:
    - Disable and invalidate instruction cache before enabling MMU.
    - Re-enable instruction cache along with MMU.

1.4 Memory access control

Access to a memory region is controlled by the access permission and domain bits in the TLB entry. APX and XN (execute never) bits have been added in VMSAv6.

1.4.1 Access permissions

- **Permission fault**: If an access is made to an area of memory without the required permissions, a Permission Fault is raised.

- **Access permissions**: The access permissions are determined by a combination of the AP and APX bits in the page table.

1.4.2 Domains(deprecated)

- A domain is a named collection of memory regions. The ARM architecture supports up to 16 domains.
- Each page table entry and TLB entry contains a field that specifies the domain the entry belongs to.
- Access to each domain is controlled by a two-bit field in the Domain Access Control Register (DACR).

1.4.2.1 DACR:

- The DACR is a register that holds 16 two-bit fields, one for each domain.
- These fields can be set to one of three values:
    - b'00: No access - Any attempt to access memory in that domain will generate a domain fault.
    - b'01: Client - Accesses are allowed, but they must follow the access permissions set in the TLB entries for that domain.
    - b'11: Manager - Accesses are allowed without checking the TLB entries. This gives manager programs full control over the domain, but also bypasses security checks.

1.5 Memory region attributes

1.5.1 Prior to VMSAv6(Problem)

- Only two bits were used for memory region attributes: C (cacheable) and B (bufferable).
- **Incompatible**: The exact usage of these bits was implementation defined, which meant different ARM processors could behave differently, even if they used the same memory region attributes.

1.5.2 VMSAv6 and Later

- VMSAv6 introduced a more formal memory model, with additional bit fields and definitions for memory region attributes.
- These attributes provide more consistent behaviors across different ARM processors.

1.5.3 Key Memory Region Attributes

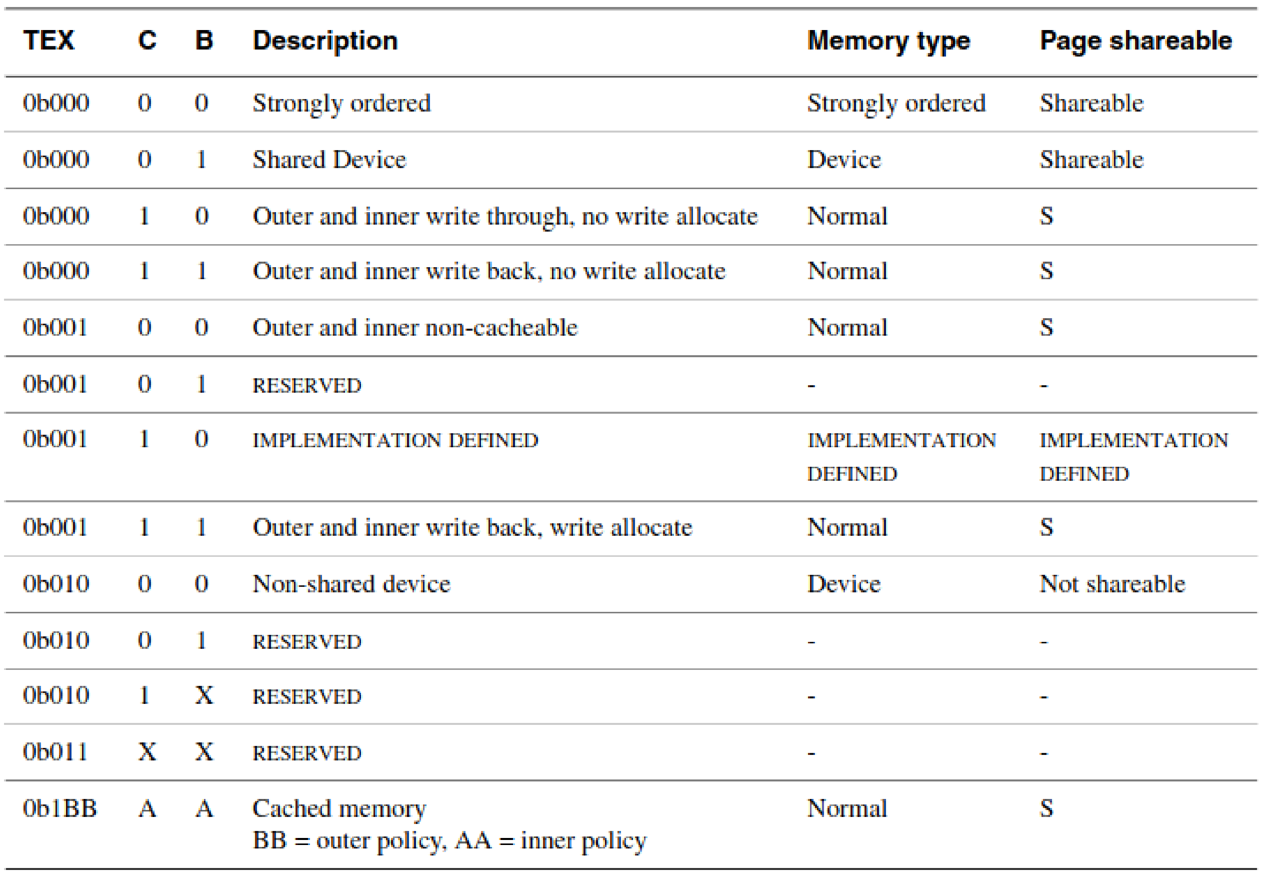
- TEX (Type Extension): This field provides additional information about the memory region type, such as whether it is normal memory, device memory, or strongly ordered memory.
- C (Cacheable): This bit controls whether the memory region can be cached.

- B (Bufferable): This bit controls whether writes to the memory region can be buffered in the write buffer.
- S (Shared): This bit (only present in certain page tables) controls whether the memory region is shared between multiple processors.

1.5.4 Page Table Formats

- The TEX, C, and B bits are encoded in the page table entries.
- Table B4-3 in the ARM Architecture Reference Manual shows the mapping of these bits to different memory region types.

Table B4-3 CB + TEX Encodings



TEX	C	B	Description	Memory type	Page shareable
0b000	0	0	Strongly ordered	Strongly ordered	Shareable
0b000	0	1	Shared Device	Device	Shareable
0b000	1	0	Outer and inner write through, no write allocate	Normal	S
0b000	1	1	Outer and inner write back, no write allocate	Normal	S
0b001	0	0	Outer and inner non-cacheable	Normal	S
0b001	0	1	RESERVED	-	-
0b001	1	0	IMPLEMENTATION DEFINED	IMPLEMENTATION DEFINED	IMPLEMENTATION DEFINED
0b001	1	1	Outer and inner write back, write allocate	Normal	S
0b010	0	0	Non-shared device	Device	Not shareable
0b010	0	1	RESERVED	-	-
0b010	1	X	RESERVED	-	-
0b011	X	X	RESERVED	-	-
0b1BB	A	A	Cached memory BB = outer policy, AA = inner policy	Normal	S

1.6 Aborts

Aborts are exceptions caused by memory access issues.

- **MMU fault**: Triggered by the MMU when it detects a memory access violation.
- **Debug abort**: Occurs when the processor's debug mode is active, and a breakpoint or watchpoint is hit.
- **External abort**: Caused by illegal or faulting memory accesses from the external memory system.

Faults(aborts) are recorded using the **Fault Address** and **Fault Status registers**.

1.6.1 MMU Faults

The MMU can generate four types of faults:

- **Alignment fault**: Occurs when memory access is not aligned correctly.
- **Translation fault**: Happens when there is an error translating a virtual memory address to a physical one.
- **Domain fault**: Triggered when access to a domain is not allowed or improperly configured.
- **Permission fault**: Arises when there is an attempt to access memory with improper permissions.

1.6.1.1 Fault-checking Sequence (Figure B4-2)

The MMU follows a sequence to check for faults to determine if a fault should be reported.

1.6.1.2 Translation and Domain Faults

- **Section translation fault**: Occurs when the first-level descriptor is invalid.
- **Section domain fault**: Occurs when the domain of the first-level descriptor is checked and found to be incorrect.
- **Page translation fault**: Occurs when the second-level descriptor is invalid.
- **Page domain fault**: Similar to section domain faults but at the second descriptor level.

1.6.2 Debug Events

- **Precise aborts**: When the exact address of the failing instruction is known.
- **Imprecise aborts**: When the exact address is not known due to subsequent instructions executing(pipeline) before the abort is processed.

1.6.3 External Aborts

External aborts are errors from the external memory system and can be precise or imprecise, affecting the Fault Address register and Fault Status register updates.

1.6.4 Parity Error Reporting

Parity errors can be precise or imprecise, with specific fault status codes assigned for reporting these errors. These are implementation-defined.

1.7 Fault Address and Fault Status registers

1.7.1 Registers:

Prior to VMSAv6, the architecture supported a single Fault Address Register (FAR) and Fault Status Register (FSR).
VMSAv6 requires four registers:

- **Instruction Fault Status Register (IFSR):** Updated on Prefetch Aborts.
- **Data Fault Status Register (DFSR):** Updated on Data Aborts.
- **Fault Address Register (FAR):** Contains the faulting address for precise exceptions.
- **Watchpoint Fault Address Register (WFAR):** Updated on a watchpoint access that triggers a Data Abort.

1.7.2 Notes:

- IFSR and DFSR are updated on Data Aborts due to instruction cache maintenance operations.
- A fifth fault status bit (bit[10]) for IFSR and DFSR is implementation-specific. DFSR also includes a write flag (bit[11]).
- Precise Data Aborts prompt immediate CPU action, updating DFSR with Fault Status and domain number. FAR records the address causing the abort.
- Instruction fetches that cause aborts update IFSR but not FAR unless the instruction is executed.
- For Precise Data Aborts, the Fault Address is determined from R14 at the time the Prefetch Abort exception is encountered.

1.7.3 Notes for Fault Status Register Encodings Table (B4.6.1)

- Prior to VMSAv6, FS[3:0] values were implementation-defined.
- In VMSAv6, Domain information for Data Accesses is obtained via TLB lookup. For Prefetch Aborts, faulting address and domain are determined from the TLB.
- All Data Aborts affect DFSR; all Instruction Aborts affect IFSR.
- Data Aborts not from external translation update FAR with the aborting address. If from external translation, FAR does not contain the aborting address.
- Translation aborts during data cache maintenance operations update DFSR with the reason and FAR with the faulting address.
- Precise Aborts during instruction cache maintenance are indicated in DFSR, with the modified virtual address reflected in IFSR.

1.8 Hardware page table translation

MMU Function: Translates memory accesses based on sections or pages.

1.8.1 Memory Sections:

- *Supersections* (optional, 16MB)
- *Sections* (1MB)

1.8.2 Supported page sizes:

- *Large pages* (64KB)
- *Small pages* (4KB)
- *Tiny pages* (1KB, not in VMSAv6)

1.8.3 Translation Tables: Two-level structure in main memory for translating virtual addresses to physical addresses.

- **First-level table**: Contains section and supersection translations, and pointers to second-level tables.
- **Second-level tables**: Handle translations for pages.

1.8.4 First-level Fetch

- **Address Translation**: Bits[31:14] of the Translation Table Base Register are used with modified virtual addresses to fetch descriptors or pointers to second-level tables.

1.8.5 Etc

- **On TLB Miss**: When a TLB miss occurs, the Translation Table Base Register (in CP15 register 2) holds the base address of the first-level table.
- **Endian Configuration**: The EE-bit in the System Control coprocessor determines endianness during table lookups.

1.9 Page Table Translation in VMSAv6

VMSAv6 supports two page table formats:

1. **Backward-Compatible Format**: This format supports sub-page access permissions and has been extended to support extended region types.
2. **New Format**: This format does not support sub-page access permissions but includes support for the following features:

- Extended region types
- Global and process-specific pages
- More access permissions
- Marking of shared and non-shared regions

- Marking of execute-never regions.

1.9.1 First-Level Descriptors(Entries in the first-level translation table)

- When a virtual address is translated to a physical address, the MMU first looks at the first-level translation table.
- The first-level descriptor can either directly point to a large section of physical memory (like a 1MB section), or it can point to a second-level translation table that handles smaller blocks of memory.

The first-level translation table contains entries that can be one of several types:

- **Supersection Descriptor**: Similar to a section descriptor, a supersection descriptor maps a larger block of memory (typically 16MB) and also does not point to a second-level table.
- **Section Descriptor**: This type of entry maps a full 1MB section of virtual address space directly to physical memory. It does not point to a second-level table but instead provides the physical base address of the section along with access permissions and other attributes.
- **Coarse Page Table Descriptor**: This entry does point to a second-level table, specifically a **coarse second-level page table**. Each entry in the coarse page table then maps a smaller portion of memory (typically 4KB pages).
- **Fine Page Table Descriptor(OBSOLETE)**: This is similar to the coarse page table descriptor but allows for an even finer granularity of mapping (typically **1KB** pages) and thus points to a **fine seond-level page table** with more entries than the coarse table.

1.9.2 Second-Level Descriptor(Entries in the second-level translation table)

- If the first-level descriptor points to a second-level translation table, the MMU then uses this table for further translation.
- Second-level descriptors provide a finer granularity of memory management, allowing for smaller sections of memory to be individually controlled, such as 64KB large pages or 4KB small pages.
- These descriptors are used to define more specific attributes of the memory region, such as access permissions, cache behavior, and whether the memory is buffered.

1.10 CP15 Registers (deprecated in ARMv8)

- **Memory management**: This includes the control of the Memory Management Unit (MMU), Translation Lookaside Buffers (TLB), and cache control for instruction and data caches.
- **Protection**: CP15 registers define and control the different protection regions in the memory to prevent unauthorized access.
- **System identification**: These registers can hold information about the processor, such as the manufacturer ID, CPU ID, and version numbers.
- **Performance monitoring**: Certain CP15 registers are used to monitor the performance of the CPU, including counting cache misses, instruction execution, and other performance metrics.
- **Configuration**: CP15 registers allow the configuration of other system settings, like the endianess of data processing, interrupt handling, and so forth.

1.11 Protected Memory System Architecture (PMSA)

1.11.1 Overview of PMSA

- **Functionality**: Controls access to memory regions.(hardware based)
- **Target Devices**: Primarily used in embedded systems and low-power devices.
- **Memory Protection Unit (MPU)**: Central to PMSA, the MPU controls access rights.

1.11.2 Memory protection

- **Protection Attributes**: Each memory segment is assigned specific attributes (read/write/execute permissions).
- **Fault Handling**: If a task accesses a memory region it is not permitted to without proper permissions, a fault is generated.
- **Isolation of Faults**: Limits the impact of faults to specific regions.

1.11.3 Operational Modes

- **Privileged Mode**: Full access to all resources, usually for the operating system.
- **Unprivileged Mode**: Restricted access, typically for application code.

1.11.4 Advantages and Limitations

- **Advantages**: Enhanced security, fine-grained control over memory, and fault isolation.
- **Limitations**: Complexity in configuration, overhead in terms of memory and processing.