# Document

# Armv8.5-A Memory Tagging Extension Whitepaper

file:///C:/Users/juanm/Downloads/Arm\_Memory\_Tagging\_Extension\_Whitepaper-2.pdf

## Design:

* The Arm Memory Tagging Extension implements lock and key access to memory. Locks can be set on memory and keys provided during memory access. If the key matches the lock, the access is permitted. If it does not match, an error is reported.
* Memory locations are tagged by adding four bits of metadata to each 16 bytes of physical memory. This is the Tag Granule. Tagging memory implements the lock.
  + Pointers, and therefore virtual addresses, are modified to contain the key.
  + In order to implement the key bits without requiring larger pointers MTE uses the Top Byte Ignore (TBI) feature of the Armv8-A Architecture
* MTE relies on the lock and the key being different to detect memory safety violations
* As there are a limited number of tag bits available, it cannot be guaranteed that two memory allocations will have different tags for any specific execution.

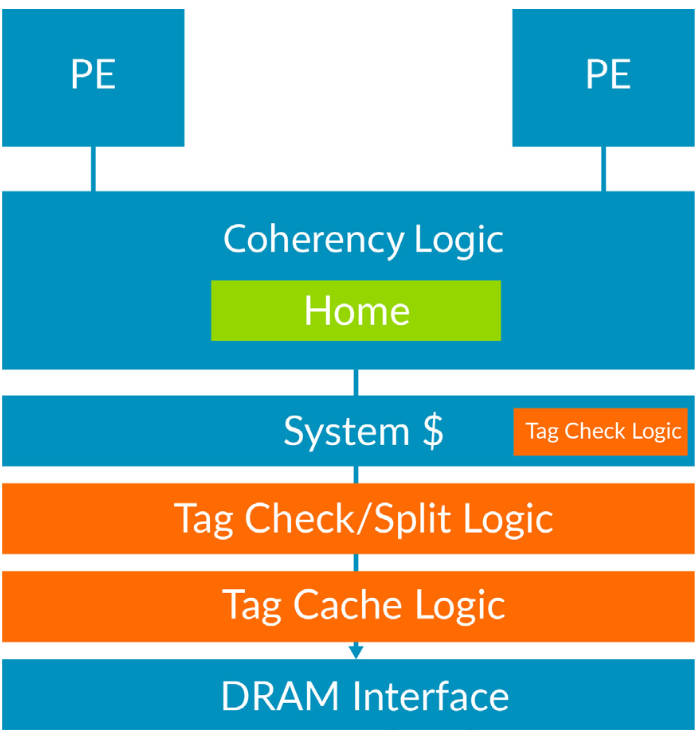
## Architecture Details:

* MTE adds a new memory type, Normal Tagged Memory, to the Arm Architecture.
* A mismatch between the tag in the address and the tag in memory can be configured to cause a synchronous exception or to be asynchronously reported.
  + SYNCHRONOUS: possible to exactly determine which load or store instruction caused the tag mismatch.
  + ASYNCHRONOUS: details are accumulated in a system register. A control is provided to ensure that this register is updated on entry to software running at a higher exception level. asynchronous reporting is imprecise as it is only possible to isolate the mismatch to a particular thread of execution.

## Instructions

| Instruction | What it does | TL;DR Summary |
| --- | --- | --- |
| Instructions for tag manipulation applicable to stack and heap tagging. | | |
| IRG | A source of random tags is required for the statistical basis of MTE to be valid. IRG is defined to provide this in hardware and insert such a tag into a register for use  by other instructions | Randomness for tags inserted into register |
| GMI | This instruction is for manipulating the excluded set of tags for use with the IRG instruction.  This is intended for cases where software uses specific tag values for special purposes  while retaining random tag behavior for normal allocations | Exclude specific tags for private use |
| LDG, STG, STZG | allow getting or setting tags in memory. They are intended for changing tags in memory either without modifying the data or zeroing the data. | |
| ST2G, STZ2G | denser alternatives to STG and STZG which operate on two granules of memory  when allocation size allows them to be used | 2 granules of memory instead of 1 |
| STGP | This instruction stores both tag and data to memory. | |
| Instructions Intended for pointer arithmetic and stack tagging: | | |
| ADDG and SUBG | Arithmetic on addresses. They allow both the tag and address to be separately modified by an immediate value.  These instructions are intended for creating the addresses of objects on the stack. |  |
| SUBP(S) | 56-bit subtract with optional flag setting which is required  for pointer arithmetic that ignores the tag in the top byte. |  |
| Instructions intended for system use: | | |
| LDGM, STGM, and STZGM | bulk tag manipulation instructions which are UNDEFINED at EL0.  Intended for system software to manipulate tags for the purposes of initialization and  serialization. |  |

## Deploying MTE in HW

* Modifications to AMBA 5 Coherent Hub Interface (CHI) allows implementation of MTE.
* 

Heap Tagging:

* In a dynamically linked system, it’s possible to deploy a tagged heap without changing existing binaries. Only OS kernel and C library code needs to be altered
* They modified malloc, free, calloc, and realloc

Stack Tagging

* Tagging memory allocated on a run-time stack requires compiler support and kernel support. Binaries must be recompiled.
* Our partners have prototyped a strategy of choosing a random tag, using the IRG instruction, during function entry when a new stack frame is allocated. The compiler then uses the ADDG and SUBG instructions to create tagged addresses for each stack slot within the function, where the tag is offset from the initial random tag. The stack allocation might be bulk-initialized using an appropriate tag store instruction but a compiler need not initialize any slot that will be provably initialized before use by the function’s body code.
* Protecting adjacent objects on the stack requires increasing the alignment of those objects to the Tag Granule, which is to 16 bytes. In some programs, MTE causes an increase stack usage because of this effect. Our benchmarking suggests that the increase is usually small.

# Arm Architecture Reference Manual for A-Profile Architecture

file:///C:/Users/juanm/Downloads/DDI0487J\_a\_a-profile\_architecture\_reference\_manual.pdf

## Versions of MTE:

* FEAT\_MTE
  + Implements memory tagging instructions accessible in EL0
  + A set of tag load and tag store instructions is provided.
  + Instructions to generate and insert Logical Tags in addresses are provided.
  + System instructions to Clean, and Clean and Invalidate Allocation Tags from caches are provided.
* FEAT\_MTE2
  + FEAT\_MTE2 supports all instructions and System registers defined by the extension, Allocation Tags in memory, and Tag Checking of accesses to tagged memory.
  + When FEAT\_MTE2 is implemented:
    - All FEAT\_MTE functionalities are available for use.
    - System register and page level control over access to Allocation Tags in memory is provided.
    - Allocation Tags are provided for each 16-byte granule of Conventional memory.
    - The tag PA space is separate to the data physical address (data PA) space accessed by data
    - load and store instructions to access data in normal memory and devices.
    - Any associated fields in System control registers are available for use.
    - All System registers defined by the extension become available for use.
    - All System instructions and instructions defined by the extension become available for use.
* FEAT\_MTE3
  + FEAT\_MTE3 adds support for asymmetric Tag Check Fault handling
  + All FEAT\_MTE and FEAT\_MTE2 functionalities are available for use.
  + Tag Check Faults can be configured to cause a synchronous exception on reads, and be asynchronously accumulated on writes.
  + Any Tag Check Fault on an access that performs both a read and a write can be configured to cause a synchronous exception

## Allocation Tags

* The tag PA space provides access to Allocation Tags stored in memory. The data PA space provides access to data held in memory.

## Cache activity and Allocation Tags

* When data is evicted from a cache entry at a cache level, the evicted data can overwrite data in memory that has been written by another observer if either, or any of the following are true:
  + The data has been written by an observer in the Shareability domain of that memory location, where the maximum size of the memory that can be overwritten is defined by the Cache Write-Back Granule in CTR\_EL0.
  + The associated Allocation Tags have been written to by an observer in the Shareability domain of that memory location, where the maximum size of the memory that can be overwritten is defined by the Cache Write-Back Granule in CTR\_EL0
* When Allocation Tags are evicted from a cache entry at a cache level, the evicted Allocation Tags can overwrite Allocation Tags in memory that has been written by another observer if either, or any of the following are true:
  + The Allocation Tags associated with memory within an address range of the size of the Cache Write-Back Granule, aligned to that size, have been written to by an observer in the Shareability domain of that memory location.
  + Data within an address range of the size of Cache Write-Back Granule in CTR\_EL0, aligned to that size, of the associated data has been written to by an observer in the Shareability domain of that memory location.
* If an implementation can overwrite Allocation Tags in memory that have been written by another observer, where the Allocation Tags have not been written by an observer in the Shareability domain of that memory location, then:
  + A cache maintenance operation which cleans data from a cache level must also clean the associated Allocation Tags
  + A cache maintenance operation which invalidates, or cleans and invalidates data from a cache level, must also clean and invalidate the associated Allocation Tags.

## Tag checking

* A memory access that is a read or write can be either Tag Checked or Tag Unchecked.
* An access to the data PA space can be either Tag Checked or Tag Unchecked.
* An access to the tag PA space is always Tag Unchecked.
* A data access which is performed as part of a prefetch operation is Tag Unchecked.
* A Tag Checked memory access includes a Physical Address Tag.
* A Tag Checked access causes a Tag Check operation to be performed

Multi-Tag: A Hardware-Software Co-Design for Memory Safety based on Multi-Granular Memory Tagging

<https://arxiv.org/pdf/2209.00307.pdf>

Titan memory tagging, memory efficient design

[AMBA CHI Architecture Specification](https://developer.arm.com/documentation/ihi0050/latest)

TODO:

* Weekly Schedule of per week what we’re going to do
* List of tasks, delegated, etc what I’m doing, what Grange is doing, etc