### **PhysicalTag**

****// AArch64.PhysicalTag()

// =====================

// Generate a Physical Tag from a Logical Tag in an address

bits(4) AArch64.PhysicalTag(bits(64) vaddr)

return vaddr<59:56>;

### **TranslateAddressForAtomicAccess**

****// AArch64.TranslateAddressForAtomicAccess()

// =========================================

// Performs an alignment check for atomic memory operations.

// Also translates 64-bit Virtual Address into Physical Address.

AddressDescriptor AArch64.TranslateAddressForAtomicAccess(bits(64) address, integer sizeinbits)

boolean iswrite = FALSE;

size = sizeinbits DIV 8;

assert size IN {1, 2, 4, 8, 16};

aligned = [AArch64.CheckAlignment](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Functions?lang=en#AArch64.CheckAlignment.4)(address, size, [AccType\_ATOMICRW](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#AccType_ATOMICRW), iswrite);

// MMU or MPU lookup

memaddrdesc = [AArch64.TranslateAddress](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Translation?lang=en#AArch64.TranslateAddress.5)(address, [AccType\_ATOMICRW](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#AccType_ATOMICRW), iswrite,

aligned, size);

// Check for aborts or debug exceptions

if [IsFault](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.IsFault.1)(memaddrdesc) then

[AArch64.Abort](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Exceptions?lang=en#AArch64.Abort.2)(address, memaddrdesc.fault);

// Effect on exclusives

if memaddrdesc.memattrs.shareability != [Shareability\_NSH](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#Shareability_NSH) then

[ClearExclusiveByAddress](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.ClearExclusiveByAddress.3)(memaddrdesc.paddress, [ProcessorID](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.ProcessorID.0)(), size);

if [HaveMTE2Ext](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.HaveMTE2Ext.0)() && [AArch64.AccessIsTagChecked](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Functions?lang=en#AArch64.AccessIsTagChecked.2)(address, [AccType\_ATOMICRW](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#AccType_ATOMICRW)) then

bits(4) ptag = [AArch64.PhysicalTag](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Functions?lang=en#AArch64.PhysicalTag.1)(address);

accdesc = [CreateAccessDescriptor](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.CreateAccessDescriptor.1)([AccType\_ATOMICRW](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#AccType_ATOMICRW));

if ![AArch64.CheckTag](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Functions?lang=en#AArch64.CheckTag.4)(memaddrdesc, accdesc, ptag, iswrite) then

[AArch64.TagCheckFault](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Exceptions?lang=en#AArch64.TagCheckFault.3)(address, [AccType\_ATOMICRW](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#AccType_ATOMICRW), iswrite);

return memaddrdesc;



### **AArch64.MemTag (Load/Store)**

****// AArch64.MemTag[] - non-assignment (read) form

// =============================================

// Load an Allocation Tag from memory.

bits(4) AArch64.MemTag[bits(64) address, [AccType](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#AccType) acctype]

[AddressDescriptor](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Translation?lang=en#AddressDescriptor) memaddrdesc;

bits(4) value;

iswrite = FALSE;

aligned = TRUE;

memaddrdesc = [AArch64.TranslateAddress](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Translation?lang=en#AArch64.TranslateAddress.5)(address, acctype, iswrite, aligned,

[TAG\_GRANULE](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#TAG_GRANULE));

accdesc = [CreateAccessDescriptor](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.CreateAccessDescriptor.1)(acctype);

// Check for aborts or debug exceptions

if [IsFault](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.IsFault.1)(memaddrdesc) then

[AArch64.Abort](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Exceptions?lang=en#AArch64.Abort.2)(address, memaddrdesc.fault);

// Return the granule tag if tagging is enabled...

if [AArch64.AllocationTagAccessIsEnabled](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Functions?lang=en#AArch64.AllocationTagAccessIsEnabled.1)(acctype) && memaddrdesc.memattrs.tagged then

if [HaveRME](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.HaveRME.0)() then

fault = [NoFault](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Translation?lang=en#impl-shared.NoFault.0)();

accdesc = [CreateAccessDescriptor](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.CreateAccessDescriptor.1)(acctype);

fault.gpcf = [GranuleProtectionCheck](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Translation?lang=en#impl-shared.GranuleProtectionCheck.2)(memaddrdesc, accdesc);

if fault.gpcf.gpf != [GPCF\_None](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#GPCF_None) then

fault.statuscode = [Fault\_GPCFOnOutput](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#Fault_GPCFOnOutput);

fault.paddress = memaddrdesc.paddress;

fault.acctype = acctype;

fault.write = FALSE;

[AArch64.Abort](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Exceptions?lang=en#AArch64.Abort.2)(address, fault);

(memstatus, tag) = [PhysMemTagRead](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Functions?lang=en#impl-aarch64.PhysMemTagRead.2)(memaddrdesc, accdesc);

if [IsFault](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.IsFault.1)(memstatus) then

[HandleExternalReadAbort](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.HandleExternalReadAbort.4)(memstatus, memaddrdesc, 1, accdesc);

return tag;

else

// ...otherwise read tag as zero.

return '0000';

// AArch64.MemTag[] - assignment (write) form

// ==========================================

// Store an Allocation Tag to memory.

AArch64.MemTag[bits(64) address, [AccType](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#AccType) acctype] = bits(4) value

[AddressDescriptor](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Translation?lang=en#AddressDescriptor) memaddrdesc;

iswrite = TRUE;

// Stores of allocation tags must be aligned

if address != [Align](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.Align.2)(address, [TAG\_GRANULE](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#TAG_GRANULE)) then

boolean secondstage = FALSE;

[AArch64.Abort](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Exceptions?lang=en#AArch64.Abort.2)(address, [AlignmentFault](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Translation?lang=en#impl-shared.AlignmentFault.3)(acctype, iswrite, secondstage));

aligned = TRUE;

memaddrdesc = [AArch64.TranslateAddress](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Translation?lang=en#AArch64.TranslateAddress.5)(address, acctype, iswrite, aligned,

[TAG\_GRANULE](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#TAG_GRANULE));

// It is CONSTRAINED UNPREDICTABLE if tags stored to memory locations marked as Device

// generate an Alignment Fault or store the data to locations.

if memaddrdesc.memattrs.memtype == [MemType\_Device](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#MemType_Device) then

c = [ConstrainUnpredictable](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.ConstrainUnpredictable.1)([Unpredictable\_DEVICETAGSTORE](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#Unpredictable_DEVICETAGSTORE));

assert c IN {[Constraint\_NONE](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#Constraint_NONE), [Constraint\_FAULT](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#Constraint_FAULT)};

if c == [Constraint\_FAULT](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#Constraint_FAULT) then

boolean secondstage = FALSE;

[AArch64.Abort](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Exceptions?lang=en#AArch64.Abort.2)(address, [AlignmentFault](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Translation?lang=en#impl-shared.AlignmentFault.3)(acctype, iswrite, secondstage));

// Check for aborts or debug exceptions

if [IsFault](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.IsFault.1)(memaddrdesc) then

[AArch64.Abort](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Exceptions?lang=en#AArch64.Abort.2)(address, memaddrdesc.fault);

accdesc = [CreateAccessDescriptor](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.CreateAccessDescriptor.1)(acctype);

// Memory array access

if [AArch64.AllocationTagAccessIsEnabled](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Functions?lang=en#AArch64.AllocationTagAccessIsEnabled.1)(acctype) && memaddrdesc.memattrs.tagged then

if [HaveRME](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.HaveRME.0)() then

fault = [NoFault](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Translation?lang=en#impl-shared.NoFault.0)();

fault.gpcf = [GranuleProtectionCheck](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Translation?lang=en#impl-shared.GranuleProtectionCheck.2)(memaddrdesc, accdesc);

if fault.gpcf.gpf != [GPCF\_None](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#GPCF_None) then

fault.statuscode = [Fault\_GPCFOnOutput](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#Fault_GPCFOnOutput);

fault.paddress = memaddrdesc.paddress;

fault.acctype = acctype;

fault.write = TRUE;

[AArch64.Abort](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Exceptions?lang=en#AArch64.Abort.2)(address, fault);

memstatus = [PhysMemTagWrite](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Functions?lang=en#impl-aarch64.PhysMemTagWrite.3)(memaddrdesc, accdesc, value);

if [IsFault](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.IsFault.1)(memstatus) then

[HandleExternalWriteAbort](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.HandleExternalWriteAbort.4)(memstatus, memaddrdesc, 1, accdesc);



### **AArch64.CheckTag**

****// AArch64.CheckTag()

// ==================

// Performs a Tag Check operation for a memory access and returns

// whether the check passed

boolean AArch64.CheckTag([AddressDescriptor](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Translation?lang=en#AddressDescriptor) memaddrdesc, [AccessDescriptor](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#AccessDescriptor) accdesc, bits(4) ptag, boolean write)

if memaddrdesc.memattrs.tagged then

(memstatus, readtag) = [PhysMemTagRead](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Functions?lang=en#impl-aarch64.PhysMemTagRead.2)(memaddrdesc, accdesc);

if [IsFault](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.IsFault.1)(memstatus) then

[HandleExternalReadAbort](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.HandleExternalReadAbort.4)(memstatus, memaddrdesc, 1, accdesc);

return ptag == readtag;

else

return TRUE;



### **PhysMemTagRead**

****// This is the hardware operation which perform a single-copy atomic,

// Allocation Tag granule aligned, memory access from the tag in PA space.

//

// The function address the array using desc.paddress which supplies:

// \* A 52-bit physical address

// \* A single NS bit to select between Secure and Non-secure parts of the array.

//

// The accdesc descriptor describes the access type: normal, exclusive, ordered, streaming,

// etc and other parameters required to access the physical memory or for setting syndrome

// register in the event of an External abort.

(PhysMemRetStatus, bits(4)) PhysMemTagRead([AddressDescriptor](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Translation?lang=en#AddressDescriptor) desc, [AccessDescriptor](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#AccessDescriptor) accdesc);



### **PhysMemTagWrite**

****// This is the hardware operation which perform a single-copy atomic,

// Allocation Tag granule aligned, memory access to the tag in PA space.

//

// The function address the array using desc.paddress which supplies:

// \* A 52-bit physical address

// \* A single NS bit to select between Secure and Non-secure parts of the array.

//

// The accdesc descriptor describes the access type: normal, exclusive, ordered, streaming,

// etc and other parameters required to access the physical memory or for setting syndrome

// register in the event of an External abort.

PhysMemRetStatus PhysMemTagWrite([AddressDescriptor](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Translation?lang=en#AddressDescriptor) desc, [AccessDescriptor](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#AccessDescriptor) accdesc, bits (4) value);

