### **MemAtomicCompareAndSwap**

****// MemAtomicCompareAndSwap()

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

// Compares the value stored at the passed-in memory address against the passed-in expected

// value. If the comparison is successful, the value at the passed-in memory address is swapped

// with the passed-in new\_value.

bits(size) MemAtomicCompareAndSwap(bits(64) address, bits(size) expectedvalue,

bits(size) newvalue\_in, [AccType](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#AccType) ldacctype, [AccType](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#AccType) stacctype)

bits(size) newvalue = newvalue\_in;

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

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

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

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, ldaccdesc);

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 = ldacctype;

fault.write = boolean UNKNOWN;

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

// All observers in the shareability domain observe the

// following load and store atomically.

(memstatus, oldvalue) = [PhysMemRead](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.PhysMemRead.3)(memaddrdesc, size DIV 8, ldaccdesc);

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, size DIV 8, ldaccdesc);

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

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

if oldvalue == expectedvalue then

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

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

memstatus = [PhysMemWrite](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.PhysMemWrite.4)(memaddrdesc, size DIV 8, staccdesc, newvalue);

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, size DIV 8, staccdesc);

return oldvalue;



### **MemSwapTableDesc**

****// AArch64.MemSwapTableDesc()

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

// Perform HW update of table descriptor as an atomic operation

(FaultRecord, bits(64)) AArch64.MemSwapTableDesc([FaultRecord](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#FaultRecord) fault\_in, bits(64) prev\_desc,

bits(64) new\_desc, bit ee,

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

descupdateaccess = [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));

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

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

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

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

fault.paddress = descupdateaddress.paddress;

fault.gpcfs2walk = fault.secondstage;

return (fault, bits(64) UNKNOWN);

// All observers in the shareability domain observe the

// following memory read and write accesses atomically.

(memstatus, mem\_desc) = [PhysMemRead](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.PhysMemRead.3)(descupdateaddress, 8, descupdateaccess);

if ee == '1' then

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

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

iswrite = FALSE;

fault = [HandleExternalTTWAbort](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.HandleExternalTTWAbort.6)(memstatus, iswrite, descupdateaddress, descupdateaccess,

8, fault);

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

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

return (fault, bits(64) UNKNOWN);

if mem\_desc == prev\_desc then

ordered\_new\_desc = if ee == '1' then [BigEndianReverse](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.BigEndianReverse.1)(new\_desc) else new\_desc;

memstatus = [PhysMemWrite](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.PhysMemWrite.4)(descupdateaddress, 8, descupdateaccess, ordered\_new\_desc);

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

iswrite = TRUE;

fault = [HandleExternalTTWAbort](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.HandleExternalTTWAbort.6)(memstatus, iswrite, descupdateaddress, descupdateaccess,

8, fault);

fault.acctype = memstatus.acctype;

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

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

return (fault, bits(64) UNKNOWN);

// Reflect what is now in memory (in little endian format)

mem\_desc = new\_desc;

return (fault, mem\_desc);



### **MemLoad64B**

****// MemLoad64B()

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

// Performs an atomic 64-byte read from a given virtual address.

bits(512) MemLoad64B(bits(64) address, [AccType](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#AccType) acctype)

bits(512) data;

boolean iswrite = FALSE;

constant integer size = 64;

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

if ![AddressSupportsLS64](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Functions?lang=en#impl-aarch64.AddressSupportsLS64.1)(address) then

c = [ConstrainUnpredictable](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.ConstrainUnpredictable.1)([Unpredictable\_LS64UNSUPPORTED](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#Unpredictable_LS64UNSUPPORTED));

assert c IN {[Constraint\_LIMITED\_ATOMICITY](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#Constraint_LIMITED_ATOMICITY), [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

// Generate a stage 1 Data Abort reported using the DFSC code of 110101.

boolean secondstage = FALSE;

boolean s2fs1walk = FALSE;

fault = [AArch64.ExclusiveFault](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Translation?lang=en#AArch64.ExclusiveFault.4)(acctype, iswrite, secondstage, s2fs1walk);

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

else

// Accesses are not single-copy atomic above the byte level

for i = 0 to 63

data<7+8\*i : 8\*i> = [AArch64.MemSingle](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Functions?lang=en#AArch64.MemSingle.read.4)[address+8\*i, 1, acctype, aligned];

return data;

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

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

// Memory array access

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

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

accdesc.transactional = TSTATE.depth > 0;

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 = iswrite;

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

if [HaveMTE2Ext](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.HaveMTE2Ext.0)() then

if [AArch64.AccessIsTagChecked](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Functions?lang=en#AArch64.AccessIsTagChecked.2)([ZeroExtend](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.ZeroExtend.2)(address, 64), acctype) then

bits(4) ptag = [AArch64.PhysicalTag](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Functions?lang=en#AArch64.PhysicalTag.1)([ZeroExtend](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.ZeroExtend.2)(address, 64));

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, iswrite);

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

(memstatus, data) = [PhysMemRead](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.PhysMemRead.3)(memaddrdesc, size, 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, size, accdesc);

return data;



### **AArch64.MemSingle**

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

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

// Perform an atomic, little-endian read of 'size' bytes.

bits(size\*8) AArch64.MemSingle[bits(64) address, integer size, [AccType](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#AccType) acctype, boolean aligned]

boolean ispair = FALSE;

return [AArch64.MemSingle](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Functions?lang=en#AArch64.MemSingle.read.5)[address, size, acctype, aligned, ispair];

// AArch64.MemSingle[] - non-assignment (read) form

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

// Perform an atomic, little-endian read of 'size' bytes.

bits(size\*8) AArch64.MemSingle[bits(64) address, integer size, [AccType](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#AccType) acctype, boolean aligned, boolean ispair]

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

constant halfsize = size DIV 2;

if [HaveLSE2Ext](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.HaveLSE2Ext.0)() then

assert [CheckAllInAlignedQuantity](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Functions?lang=en#impl-aarch64.CheckAllInAlignedQuantity.3)(address, size, 16);

else

assert address == [Align](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.Align.2)(address, size);

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

bits(size\*8) value;

iswrite = FALSE;

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

// Memory array access

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

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

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

accdesc.transactional = TSTATE.depth > 0 && !(acctype IN {[AccType\_IFETCH](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#AccType_IFETCH),[AccType\_TTW](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#AccType_TTW)});

if accdesc.transactional && ![MemHasTransactionalAccess](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Functions?lang=en#impl-aarch64.MemHasTransactionalAccess.1)(memaddrdesc.memattrs) then

[FailTransaction](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Functions?lang=en#impl-aarch64.FailTransaction.2)([TMFailure\_IMP](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Functions?lang=en#TMFailure_IMP), FALSE);

else

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

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 = FALSE;

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

if [HaveMTE2Ext](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.HaveMTE2Ext.0)() then

if [AArch64.AccessIsTagChecked](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Functions?lang=en#AArch64.AccessIsTagChecked.2)([ZeroExtend](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.ZeroExtend.2)(address, 64), acctype) then

bits(4) ptag = [AArch64.PhysicalTag](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Functions?lang=en#AArch64.PhysicalTag.1)([ZeroExtend](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.ZeroExtend.2)(address, 64));

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)([ZeroExtend](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.ZeroExtend.2)(address, 64), acctype, iswrite);

(atomic, splitpair) = [CheckSingleAccessAttributes](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Functions?lang=en#impl-aarch64.CheckSingleAccessAttributes.7)(address, memaddrdesc.memattrs, size, acctype, iswrite, aligned, ispair);

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

if atomic then

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

elsif splitpair then

assert ispair;

bits(halfsize \* 8) lowhalf, highhalf;

(memstatus, lowhalf) = [PhysMemRead](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.PhysMemRead.3)(memaddrdesc, halfsize, 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, halfsize, accdesc);

memaddrdesc.paddress.address = memaddrdesc.paddress.address + halfsize;

(memstatus, highhalf) = [PhysMemRead](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.PhysMemRead.3)(memaddrdesc, halfsize, 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, halfsize, accdesc);

value = highhalf:lowhalf;

else

for i = 0 to size-1

(memstatus, value<8\*i+7:8\*i>) = [PhysMemRead](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.PhysMemRead.3)(memaddrdesc, 1, 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);

memaddrdesc.paddress.address = memaddrdesc.paddress.address + 1;

return value;

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

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

AArch64.MemSingle[bits(64) address, integer size, [AccType](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#AccType) acctype, boolean aligned] = bits(size\*8) value

boolean ispair = FALSE;

[AArch64.MemSingle](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Functions?lang=en#AArch64.MemSingle.write.5)[address, size, acctype, aligned, ispair] = value;

return;

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

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

// Perform an atomic, little-endian write of 'size' bytes.

AArch64.MemSingle[bits(64) address, integer size, [AccType](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#AccType) acctype, boolean aligned, boolean ispair] = bits(size\*8) value

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

constant halfsize = size DIV 2;

if [HaveLSE2Ext](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.HaveLSE2Ext.0)() then

assert [CheckAllInAlignedQuantity](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Functions?lang=en#impl-aarch64.CheckAllInAlignedQuantity.3)(address, size, 16);

else

assert address == [Align](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.Align.2)(address, size);

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

iswrite = 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, 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);

// Memory array access

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

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

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

accdesc.transactional = TSTATE.depth > 0;

if accdesc.transactional && ![MemHasTransactionalAccess](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Functions?lang=en#impl-aarch64.MemHasTransactionalAccess.1)(memaddrdesc.memattrs) then

[FailTransaction](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Functions?lang=en#impl-aarch64.FailTransaction.2)([TMFailure\_IMP](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Functions?lang=en#TMFailure_IMP), FALSE);

else

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

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);

if [HaveMTE2Ext](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.HaveMTE2Ext.0)() then

if [AArch64.AccessIsTagChecked](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Functions?lang=en#AArch64.AccessIsTagChecked.2)([ZeroExtend](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.ZeroExtend.2)(address, 64), acctype) then

bits(4) ptag = [AArch64.PhysicalTag](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Functions?lang=en#AArch64.PhysicalTag.1)([ZeroExtend](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.ZeroExtend.2)(address, 64));

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)([ZeroExtend](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.ZeroExtend.2)(address, 64), acctype, iswrite);

(atomic, splitpair) = [CheckSingleAccessAttributes](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/AArch64-Functions?lang=en#impl-aarch64.CheckSingleAccessAttributes.7)(address, memaddrdesc.memattrs, size, acctype, iswrite, aligned, ispair);

if atomic then

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

elsif splitpair then

assert ispair;

bits(halfsize\*8) lowhalf, highhalf;

<highhalf, lowhalf> = value;

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

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, halfsize, accdesc);

memaddrdesc.paddress.address = memaddrdesc.paddress.address + halfsize;

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

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, halfsize, accdesc);

else

for i = 0 to size-1

memstatus = [PhysMemWrite](https://developer.arm.com/documentation/ddi0602/2021-12/Shared-Pseudocode/Shared-Functions?lang=en#impl-shared.PhysMemWrite.4)(memaddrdesc, 1, accdesc, value<8\*i+7:8\*i>);

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);

memaddrdesc.paddress.address = memaddrdesc.paddress.address + 1;

return;



### **AArch64.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;

