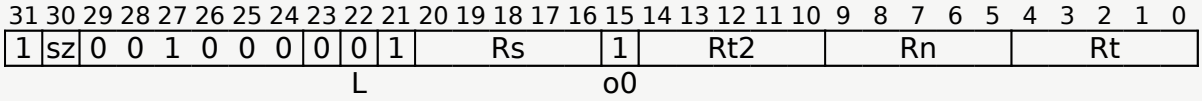
STLXP

Store-Release Exclusive Pair of registers stores two 32-bit words or two 64-bit doublewords to a memory location if the PE has exclusive access to the memory address, from two registers, and returns a status value of 0 if the store was successful, or of 1 if no store was performed. See *Synchronization and semaphores*. For information on single-copy atomicity and alignment requirements, see *Requirements for single-copy atomicity* and *Alignment of data accesses*. If a 64-bit pair Store-Exclusive succeeds, it causes a single-copy atomic update of the 128-bit memory location being updated. The instruction also has memory ordering semantics, as described in *Load-Acquire, Store-Release*. For information about memory accesses, see *Load/Store addressing modes*.



32-bit (sz == 0)

```
STLXP <Ws>, <Wt1>, <Wt2>, [<Xn|SP>{,#0}]
```

64-bit (sz == 1)

```
STLXP <Ws>, <Xt1>, <Xt2>, [<Xn|SP>{,#0}]
```

```
integer n = UInt(Rn);
integer t = UInt(Rt);
integer t2 = UInt(Rt2);    // ignored by load/store single register
integer s = UInt(Rs);    // ignored by all loads and store-release

constant integer elsize = 32 << UInt(sz);
constant integer datasize = elsize * 2;
boolean tagchecked = n != 31;

boolean rt_unknown = FALSE;
boolean rn_unknown = FALSE;
if s == t || (s == t2) then
    Constraint c = ConstrainUnpredictable(Unpredictable_DATAOVERLAP);
    assert c IN {Constraint_UNKNOWN, Constraint_UNDEF, Constraint_NOP};
    case c of
        when Constraint_UNKNOWN rt_unknown = TRUE;    // store UNKNOWN
        when Constraint_UNDEF   UNDEFINED;
        when Constraint_NOP     EndOfInstruction();
if s == n && n != 31 then
    Constraint c = ConstrainUnpredictable(Unpredictable_BASEOVERLAP);
    assert c IN {Constraint_UNKNOWN, Constraint_UNDEF, Constraint_NOP};
    case c of
        when Constraint_UNKNOWN rn_unknown = TRUE;    // address is UNK
        when Constraint_UNDEF   UNDEFINED;
        when Constraint_NOP     EndOfInstruction();
```

For information about the constrained unpredictable behavior of this instruction, see *Architectural Constraints on UNPREDICTABLE behaviors*, and particularly *STLXP*.

Assembler Symbols

<Ws>	Is the 32-bit name of the general-purpose register into which the status result of the store exclusive is written, encoded in the "Rs" field. The value returned is: 0 If the operation updates memory. 1 If the operation fails to update memory.
<Xt1>	Is the 64-bit name of the first general-purpose register to be transferred, encoded in the "Rt" field.
<Xt2>	Is the 64-bit name of the second general-purpose register to be transferred, encoded in the "Rt2" field.
<Wt1>	Is the 32-bit name of the first general-purpose register to be transferred, encoded in the "Rt" field.
<Wt2>	Is the 32-bit name of the second general-purpose register to be transferred, encoded in the "Rt2" field.
<Xn|SP>	Is the 64-bit name of the general-purpose base register or stack pointer, encoded in the "Rn" field.

Aborts and alignment

If a synchronous Data Abort exception is generated by the execution of this instruction:

- Memory is not updated.
- <Ws> is not updated.

Accessing an address that is not aligned to the size of the data being accessed causes an Alignment fault Data Abort exception to be generated, subject to the following rules:

- If AArch64.ExclusiveMonitorsPass() returns TRUE, the exception is generated.
- Otherwise, it is implementation defined whether the exception is generated.

If AArch64.ExclusiveMonitorsPass() returns FALSE and the memory address, if accessed, would generate a synchronous Data Abort exception, it is implementation defined whether the exception is generated.

Operation

```
bits(64) address;
bits(datasize) data;
```

```
constant integer dbytes = datasize DIV 8;

AccessDescriptor accdesc = CreateAccDescExLDST(MemOp_STORE, TRUE, tagch

if n == 31 then
    CheckSPAlignment();
    address = SP[];
elsif rn_unknown then
    address = bits(64) UNKNOWN;
else
    address = X[n, 64];

if rt_unknown then
    data = bits(datasize) UNKNOWN;
else
    bits(datasize DIV 2) el1 = X[t, datasize DIV 2];
    bits(datasize DIV 2) el2 = X[t2, datasize DIV 2];
    data = if BigEndian(accdesc.acctype) then el1:el2 else el2:el1;
bit status = '1';
// Check whether the Exclusives monitors are set to include the
// physical memory locations corresponding to virtual address
// range [address, address+dbytes-1].

// If AArch64.ExclusiveMonitorsPass() returns FALSE and the memory addr
// if accessed, would generate a synchronous Data Abort exception, it i
// IMPLEMENTATION DEFINED whether the exception is generated.
// It is a limitation of this model that synchronous Data Aborts are ne
// generated in this case, as Mem[] is not called.
// If FEAT_SPE is implemented, it is also IMPLEMENTATION DEFINED whethe
// physical address packet is output when permitted and when
// AArch64.ExclusiveMonitorPass() returns FALSE for a Store Exclusive i
// This behavior is not reflected here due to the previously stated limi
if AArch64.ExclusiveMonitorsPass(address, dbytes, accdesc) then
    // This atomic write will be rejected if it does not refer
    // to the same physical locations after address translation.
    Mem[address, dbytes, accdesc] = data;
    status = ExclusiveMonitorsStatus();
X[s, 32] = ZeroExtend(status, 32);
```

Operational information

If PSTATE.DIT is 1, the timing of this instruction is insensitive to the value of the data being loaded or stored.

Internal version only: isa v33.64, AdvSIMD v29.12, pseudocode no_diffs_2023_09_RC2, sve v2023-06_rel ; Build timestamp: 2023-09-18T17:56