## LDAP1 (SIMD&FP)

Load-Acquire RCpc one single-element structure to one lane of one register. This instruction loads a single-element structure from memory and writes the result to the specified lane of the SIMD&FP register without affecting the other bits of the register.

The instruction has memory ordering semantics, as described in *Load-Acquire, Load-AcquirePC, and Store-Release*, except that:

- There is no ordering requirement, separate from the requirements of a Load-AcquirePC or a Store-Release, created by having a Store-Release followed by a Load-AcquirePC instruction.
- The reading of a value written by a Store-Release by a Load-AcquirePC instruction by the same observer does not make the write of the Store-Release globally observed.

This difference in memory ordering is not described in the pseudocode.

For information about memory accesses, see *Load/Store addressing modes*.

Depending on the settings in the *CPACR_EL1*, *CPTR_EL2*, and *CPTR_EL3* registers, and the current Security state and Exception level, an attempt to execute the instruction might be trapped.

**64-bit**
**(FEAT_LRCPC3)**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 8 7 6 5 | 4 3 2 1 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Q | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | Rn | Rt |
| | | | | | | | | | L | R | | | | | | opcode | | | S | size | | | |

**LDAP1 { <Vt>.D }[<index>], [<Xn|SP>]**

```
integer t = UInt(Rt);
integer n = UInt(Rn);
integer m = integer UNKNOWN;
boolean wback = FALSE;
boolean nontemporal = FALSE;
boolean tagchecked = wback || n != 31;
```

### Assembler Symbols

| | |
|---|---|
| <Vt> | Is the name of the first or only SIMD&FP register to be transferred, encoded in the "Rt" field. |
| <index> | Is the element index, encoded in "Q". |
| <Xn\|SP> | Is the 64-bit name of the general-purpose base register or stack pointer, encoded in the "Rn" field. |

### Shared Decode

```
bits(2) scale = opcode<2:1>;
integer selem = UInt(opcode<0>:R) + 1;
boolean replicate = FALSE;
integer index;

case scale of
    when '11'
        // load and replicate
        if L == '0' || S == '1' then UNDEFINED;
        scale = size;
        replicate = TRUE;
    when '00'
        index = UInt(Q:S:size);     // B[0-15]
    when '01'
        if size<0> == '1' then UNDEFINED;
        index = UInt(Q:S:size<1>);     // H[0-7]
    when '10'
        if size<1> == '1' then UNDEFINED;
        if size<0> == '0' then
            index = UInt(Q:S);     // S[0-3]
        else
            if S == '1' then UNDEFINED;
            index = UInt(Q);     // D[0-1]
            scale = '11';

MemOp memop = if L == '1' then MemOp_LOAD else MemOp_STORE;
constant integer datasize = 64 << UInt(Q);
constant integer esize = 8 << UInt(scale);
```

**Operation**

```
CheckFPAdvSIMDEnabled64();

bits(64) address;
bits(64) offs;
bits(128) rval;
bits(esize) element;
constant integer ebytes = esize DIV 8;

AccessDescriptor accdesc = CreateAccDescASIMDAcqRel(memop, tagchecked);

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

offs = Zeros(64);
if replicate then
    // load and replicate to all elements
    for s = 0 to selem-1
        element = Mem[address+offs, ebytes, accdesc];
        // replicate to fill 128- or 64-bit register
        V[t, datasize] = Replicate(element, datasize DIV esize);
        offs = offs + ebytes;
        t = (t + 1) MOD 32;
else
    // load/store one element per register
    for s = 0 to selem-1
        rval = V[t, 128];
        if memop == MemOp_LOAD then
            // insert into one lane of 128-bit register
            Elem[rval, index, esize] = Mem[address+offs, ebytes, accdesc
            V[t, 128] = rval;
        else // memop == MemOp_STORE
            // extract from one lane of 128-bit register
            Mem[address+offs, ebytes, accdesc] = Elem[rval, index, esize
        offs = offs + ebytes;
        t = (t + 1) MOD 32;

if wback then
    if m != 31 then
        offs = X[m, 64];
    if n == 31 then
        SP[] = address + offs;
    else
        X[n, 64] = address + offs;
```

**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