LDSETP, LDSETPA, LDSETPAL, LDSETPL

Atomic bit set on quadword in memory atomically loads a 128-bit quadword from memory, performs a bitwise OR with the value held in a pair of registers on it, and stores the result back to memory. The value initially loaded from memory is returned in the same pair of registers.

- LDSETPA and LDSETPAL load from memory with acquire semantics.
- LDSETPL and LDSETPAL store to memory with release semantics.
- LDSETP has neither acquire nor release semantics.

Integer
(FEAT_LSE128)

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	0	0	1	1	0	0	1	A	R	1	Rt2	0	0	1	1	0	0	Rn	Rt

S   o3   opc

LDSETP (A == 0 && R == 0)

       LDSETP <Xt1>, <Xt2>, [<Xn|SP>]

LDSETPA (A == 1 && R == 0)

       LDSETPA <Xt1>, <Xt2>, [<Xn|SP>]

LDSETPAL (A == 1 && R == 1)

       LDSETPAL <Xt1>, <Xt2>, [<Xn|SP>]

LDSETPL (A == 0 && R == 1)

       LDSETPL <Xt1>, <Xt2>, [<Xn|SP>]

```
if !IsFeatureImplemented(FEAT_LSE128) then UNDEFINED;
if Rt == '11111' then UNDEFINED;
if Rt2 == '11111' then UNDEFINED;
integer t = UInt(Rt);
integer t2 = UInt(Rt2);
integer n = UInt(Rn);

boolean acquire = A == '1';
boolean release = R == '1';
boolean tagchecked = n != 31;

boolean rt_unknown = FALSE;

if t == t2 then
    Constraint c = ConstrainUnpredictable(Unpredictable_LSE128OVERLAP);
```

```
        assert c IN {Constraint_UNKNOWN, Constraint_UNDEF, Constraint_NOP};
        case c of
            when Constraint_UNKNOWN rt_unknown = TRUE;      // result is UNKN
            when Constraint_UNDEF   UNDEFINED;
            when Constraint_NOP     EndOfInstruction();
```

Assembler Symbols

<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.
<Xn|SP>	Is the 64-bit name of the general-purpose base register or stack pointer, encoded in the "Rn" field.

Operation

```
bits(64) address;
bits(64) value1 = X[t, 64];
bits(64) value2 = X[t2, 64];
bits(128) data;
bits(128) store_value;

AccessDescriptor accdesc = CreateAccDescAtomicOp(MemAtomicOp_ORR, acqui

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

store_value = if BigEndian(accdesc.acctype) then value1:value2 else val

bits(128) comparevalue = bits(128) UNKNOWN;      // Irrelevant when not e
data = MemAtomic(address, comparevalue, store_value, accdesc);

if rt_unknown then
    data = bits(128) UNKNOWN;

if BigEndian(accdesc.acctype) then
    X[t, 64] = data<127:64>;
    X[t2, 64] = data<63:0>;
else
    X[t, 64] = data<63:0>;
    X[t2, 64] = data<127:64>;
```

Operational information

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