LDIAPP

Load-Acquire RCpc ordered Pair of registers calculates an address from a base register value and an optional offset, loads two 32-bit words or two 64-bit doublewords from memory, and writes them to two registers. For information on single-copy atomicity and alignment requirements, see *Requirements for single-copy atomicity* and *Alignment of data accesses*. The instruction also has memory ordering semantics, as described in *Load-Acquire, Load-AcquirePC, and Store-Release*, except that:

- The Memory effects associated with Xt1/Wt1 are Ordered-before the Memory effects associated with Xt2/Wt2.
- 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.

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

Integer
(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
1 x	0 1 1	0	0 1 0	1	0	Rt2	0 0 0 x	1 0	Rn	Rt
size				L			opc2			

32-bit (size == 10 && opc2 == 0001)

```
LDIAPP <Wt1>, <Wt2>, [<Xn|SP>]
```

32-bit post-index (size == 10 && opc2 == 0000)

```
LDIAPP <Wt1>, <Wt2>, [<Xn|SP>], #8
```

64-bit (size == 11 && opc2 == 0001)

```
LDIAPP <Xt1>, <Xt2>, [<Xn|SP>]
```

64-bit post-index (size == 11 && opc2 == 0000)

```
LDIAPP <Xt1>, <Xt2>, [<Xn|SP>], #16
```

```
boolean postindex;
boolean wback;
postindex = opc2<0> == '0';
wback = opc2<0> == '0';
```

`LDIAPP` has the same constrained unpredictable behavior as `LDP`. For information about this constrained unpredictable behavior, see *Architectural Constraints on UNPREDICTABLE behaviors*, and particularly *LDP*.

Assembler Symbols

<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.
<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.

Shared Decode

```
integer offset;
integer n = UInt(Rn);
integer t = UInt(Rt);
integer t2 = UInt(Rt2);
constant integer scale = 2 + UInt(size<0>);
constant integer datasize = 8 << scale;
offset = if opc2<0> == '0' then (2 << scale) else 0;
boolean tagchecked = wback || n != 31;

boolean rt_unknown = FALSE;
boolean wb_unknown = FALSE;

if wback && (t == n || t2 == n) && n != 31 then
    Constraint c = ConstrainUnpredictable(Unpredictable_WBOVERLAPLD);
    assert c IN {Constraint_WBSUPPRESS, Constraint_UNKNOWN, Constraint_
    case c of
        when Constraint_WBSUPPRESS wback = FALSE;     // writeback is su
        when Constraint_UNKNOWN    wb_unknown = TRUE;     // writeback i
        when Constraint_UNDEF      UNDEFINED;
        when Constraint_NOP        EndOfInstruction();

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

Operation

```
bits(64) address;
bits(datasize) data1;
bits(datasize) data2;
constant integer dbytes = datasize DIV 8;

AccessDescriptor accdesc = CreateAccDescAcqRel(MemOp_LOAD, tagchecked);

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

if !postindex then
    address = address + offset;

if IsFeatureImplemented(FEAT_LSE2) then
    bits(2*datasize) full_data;
    accdesc.ispair = TRUE;
    full_data = Mem[address, 2*dbytes, accdesc];
    if BigEndian(accdesc.acctype) then
        data2 = full_data<(datasize-1):0>;
        data1 = full_data<(2*datasize-1):datasize>;
    else
        data1 = full_data<(datasize-1):0>;
        data2 = full_data<(2*datasize-1):datasize>;
else
    data1 = Mem[address, dbytes, accdesc];
    data2 = Mem[address+dbytes, dbytes, accdesc];
if rt_unknown then
    data1 = bits(datasize) UNKNOWN;
    data2 = bits(datasize) UNKNOWN;

X[t, datasize] = data1;
X[t2, datasize] = data2;

if wback then
    if wb_unknown then
        address = bits(64) UNKNOWN;
    elsif postindex then
        address = address + offset;
    if n == 31 then
        SP[] = address;
    else
        X[n, 64] = address;
```

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