LDAPR

Load-Acquire RCpc Register derives an address from a base register value, loads a 32-bit word or 64-bit doubleword from the derived address in memory, and writes it to a 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*.

It has encodings from 2 classes: No offset and Post-index

No offset

(FEAT_LRCPC)

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	1	1	1	0	0	0	1	0	1	(1)	(1)	(1)	(1)	(1)	1	1	0	0	0	0	Rn					Rt				
size											Rs																					

32-bit (size == 10)

```
LDAPR <Wt>, [<Xn|SP> {,#0}]
```

64-bit (size == 11)

```
LDAPR <Xt>, [<Xn|SP> {,#0}]
```

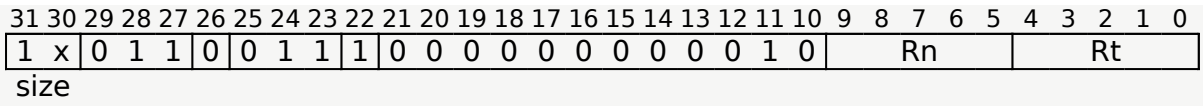
```
boolean wback = FALSE;
integer offset = 0;
boolean wb_unknown = FALSE;

integer n = UInt(Rn);
integer t = UInt(Rt);

constant integer elsize = 8 << UInt(size);
constant integer regsize = if elsize == 64 then 64 else 32;
constant integer datasize = elsize;
boolean tagchecked = n != 31;
```

Post-index

(FEAT_LRCPC3)



32-bit (size == 10)

```
LDAPR <Wt>, [<Xn|SP>], #4
```

64-bit (size == 11)

```
LDAPR <Xt>, [<Xn|SP>], #8
```

```
boolean wback = TRUE;

integer n = UInt(Rn);
integer t = UInt(Rt);

constant integer regsize = if size == '11' then 64 else 32;
constant integer datasize = 8 << UInt(size);
constant integer offset = 1 << UInt(size);

boolean tagchecked = TRUE;

boolean wb_unknown = FALSE;

if n == t && 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();
```

Assembler Symbols

<Wt>	Is the 32-bit name of the general-purpose register to be loaded, encoded in the "Rt" field.
<Xt>	Is the 64-bit name of the general-purpose register to be loaded, encoded in the "Rt" 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(datasize) data;
constant integer dbytes = datasize DIV 8;

AccessDescriptor accdesc = CreateAccDescLDAcqPC(tagchecked);

if n == 31 then
    CheckSPAlignment();
```

```
    address = SP[];
else
    address = X[n, 64];

data = Mem[address, dbytes, accdesc];
X[t, regsize] = ZeroExtend(data, regsize);

if wback then
    if wb_unknown then
        address = bits(64) UNKNOWN;
    else
        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