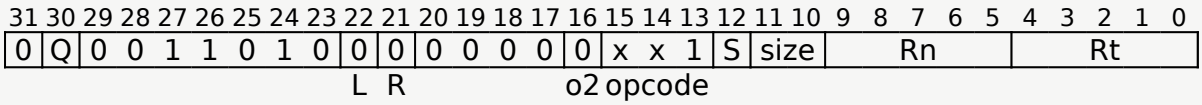
## ST3 (single structure)

Store single 3-element structure from one lane of three registers. This instruction stores a 3-element structure to memory from corresponding elements of three SIMD&FP registers.

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.

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

### No offset




#### 8-bit (opcode == 001)

```
ST3 { <Vt>.B, <Vt2>.B, <Vt3>.B }[<index>], [<Xn|SP>]
```

#### 16-bit (opcode == 011 && size == x0)

```
ST3 { <Vt>.H, <Vt2>.H, <Vt3>.H }[<index>], [<Xn|SP>]
```

#### 32-bit (opcode == 101 && size == 00)

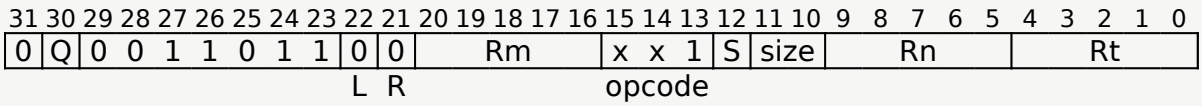
```
ST3 { <Vt>.S, <Vt2>.S, <Vt3>.S }[<index>], [<Xn|SP>]
```

#### 64-bit (opcode == 101 && S == 0 && size == 01)

```
ST3 { <Vt>.D, <Vt2>.D, <Vt3>.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;
```

### Post-index




#### 8-bit, immediate offset (Rm == 11111 && opcode == 001)

```
ST3 { <Vt>.B, <Vt2>.B, <Vt3>.B }[<index>], [<Xn|SP>], #3
```

**8-bit, register offset (Rm != 11111 && opcode == 001)**

```
ST3 { <Vt>.B, <Vt2>.B, <Vt3>.B }[<index>], [<Xn|SP>], <Xm>
```

**16-bit, immediate offset (Rm == 11111 && opcode == 011 && size == x0)**

```
ST3 { <Vt>.H, <Vt2>.H, <Vt3>.H }[<index>], [<Xn|SP>], #6
```

**16-bit, register offset (Rm != 11111 && opcode == 011 && size == x0)**

```
ST3 { <Vt>.H, <Vt2>.H, <Vt3>.H }[<index>], [<Xn|SP>], <Xm>
```

**32-bit, immediate offset (Rm == 11111 && opcode == 101 && size == 00)**

```
ST3 { <Vt>.S, <Vt2>.S, <Vt3>.S }[<index>], [<Xn|SP>], #12
```

**32-bit, register offset (Rm != 11111 && opcode == 101 && size == 00)**

```
ST3 { <Vt>.S, <Vt2>.S, <Vt3>.S }[<index>], [<Xn|SP>], <Xm>
```

**64-bit, immediate offset (Rm == 11111 && opcode == 101 && S == 0 && size == 01)**

```
ST3 { <Vt>.D, <Vt2>.D, <Vt3>.D }[<index>], [<Xn|SP>], #24
```

**64-bit, register offset (Rm != 11111 && opcode == 101 && S == 0 && size == 01)**

```
ST3 { <Vt>.D, <Vt2>.D, <Vt3>.D }[<index>], [<Xn|SP>], <Xm>
```

```
integer t = UInt(Rt);
integer n = UInt(Rn);
integer m = UInt(Rm);
boolean wback = TRUE;
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. |
| <Vt2> | Is the name of the second SIMD&FP register to be transferred, encoded as "Rt" plus 1 modulo 32. |
| <Vt3> | Is the name of the third SIMD&FP register to be transferred, encoded as "Rt" plus 2 modulo 32. |

| | |
|---|---|
| <index> | For the 8-bit variant: is the element index, encoded in "Q:S:size".<br><br>For the 16-bit variant: is the element index, encoded in "Q:S:size<1>".<br><br>For the 32-bit variant: is the element index, encoded in "Q:S".<br><br>For the 64-bit variant: 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. |
| <Xm> | Is the 64-bit name of the general-purpose post-index register, excluding XZR, encoded in the "Rm" 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 = CreateAccDescASIMD(memop, nontemporal, tagch

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