STRH (register)

Store Register Halfword (register) calculates an address from a base register value and an offset register value, and stores a halfword from a 32-bit register to the calculated address. For information about memory accesses, see *Load/Store addressing modes*.

The instruction uses an offset addressing mode, that calculates the address used for the memory access from a base register value and an offset register value. The offset can be optionally shifted and extended.

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 1	1 1 1	0	0 0	0 0	1	Rm	option	S	1 0	Rn	Rt
size				opc							

```
STRH <Wt>, [<Xn|SP>, (<Wm>|<Xm>){, <extend> {<amount>}}]
```

```
if option<1> == '0' then UNDEFINED;     // sub-word index
ExtendType extend_type = DecodeRegExtend(option);
integer shift = if S == '1' then 1 else 0;
```

Assembler Symbols

<Wt>	Is the 32-bit name of the general-purpose register to be transferred, 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.
<Wm>	When option<0> is set to 0, is the 32-bit name of the general-purpose index register, encoded in the "Rm" field.
<Xm>	When option<0> is set to 1, is the 64-bit name of the general-purpose index register, encoded in the "Rm" field.
<extend>	Is the index extend/shift specifier, defaulting to LSL, and which must be omitted for the LSL option when <amount> is omitted. encoded in “option”:

option	<extend>
010	UXTW
011	LSL
110	SXTW
111	SXTX

<amount>

Is the index shift amount, optional only when <extend> is not LSL. Where it is permitted to be optional, it defaults to #0. It is encoded in “S”:

S	<amount>
0	#0
1	#1

Shared Decode

```
integer n = UInt(Rn);
integer t = UInt(Rt);
integer m = UInt(Rm);
```

Operation

```
bits(64) offset = ExtendReg(m, extend_type, shift, 64);
bits(64) address;
bits(16) data;

boolean privileged = PSTATE.EL != EL0;
AccessDescriptor accdesc = CreateAccDescGPR(MemOp_STORE, FALSE, privile

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

address = address + offset;

data = X[t, 16];
Mem[address, 2, accdesc] = data;
```

Operational information

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