F5.1.128 ORR, ORRS (register-shifted register)

Bitwise OR (register-shifted register) performs a bitwise (inclusive) OR of a register value and a register-shifted register value, and writes the result to the destination register. It can optionally update the condition flags based on the result.

A1

31 28	27	26	25	24	23	22	21	20	19 16	15 12	11 8	7	6 5	4	3 0
!=1111	0	0	0	1	1	0	0	S	Rn	Rd	Rs	0	type	1	Rm
cond															

Flag setting variant

Applies when `S == 1`.

```
ORRS{<c>}{<q>} {<Rd>,} <Rn>, <Rm>, <type> <Rs>
```

Not flag setting variant

Applies when `S == 0`.

```
ORR{<c>}{<q>} {<Rd>,} <Rn>, <Rm>, <type> <Rs>
```

Decode for all variants of this encoding

```
d = UInt(Rd);  n = UInt(Rn);  m = UInt(Rm);  s = UInt(Rs);
setflags = (S == '1');  shift_t = DecodeRegShift(type);
if d == 15 || n == 15 || m == 15 || s == 15 then UNPREDICTABLE;
```

Notes for all encodings

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Appendix K1 *Architectural Constraints on UNPREDICTABLE behaviors*.

Assembler symbols

`<c>`	See *Standard assembler syntax fields* on page F2-2848.
`<q>`	See *Standard assembler syntax fields* on page F2-2848.
`<Rd>`	Is the general-purpose destination register, encoded in the "Rd" field.
`<Rn>`	Is the first general-purpose source register, encoded in the "Rn" field.
`<Rm>`	Is the second general-purpose source register, encoded in the "Rm" field.
`<type>`	Is the type of shift to be applied to the second source register, encoded in the "type" field. It can have the following values:

`LSL`	when `type = 00`
`LSR`	when `type = 01`
`ASR`	when `type = 10`
`ROR`	when `type = 11`

`<Rs>`	Is the general-purpose source register holding a shift amount in its bottom 8 bits, encoded in the "Rs" field.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    shift_n = UInt(R[s]<7:0>);
    (shifted, carry) = Shift_C(R[m], shift_t, shift_n, PSTATE.C);
    result = R[n] OR shifted;
    R[d] = result;
    if setflags then
        PSTATE.N = result<31>;
        PSTATE.Z = IsZeroBit(result);
        PSTATE.C = carry;
        // PSTATE.V unchanged
```