FACGT

Floating-point Absolute Compare Greater than (vector). This instruction compares the absolute value of each vector element in the first source SIMD&FP register with the absolute value of the corresponding vector element in the second source SIMD&FP register and if the first value is greater than the second value sets every bit of the corresponding vector element in the destination SIMD&FP register to one, otherwise sets every bit of the corresponding vector element in the destination SIMD&FP register to zero.

This instruction can generate a floating-point exception. Depending on the settings in *FPCR*, the exception results in either a flag being set in *FPSR*, or a synchronous exception being generated. For more information, see *Floating-point exception traps*.

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 4 classes: Scalar half precision , Scalar single-precision and double-precision , Vector half precision and Vector single-precision and double-precision

Scalar half precision
(FEAT_FP16)

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	1	1	0	1	1	0	Rm					0	0	1	0	1	1	Rn					Rd				
		U						E												ac											

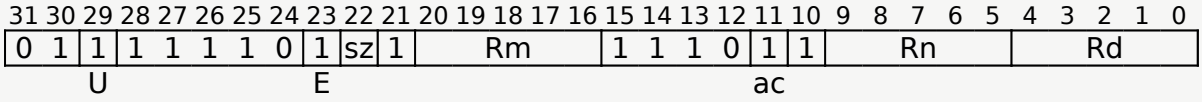
FACGT <Hd>, <Hn>, <Hm>

```
if !IsFeatureImplemented(FEAT_FP16) then UNDEFINED;

integer d = UInt(Rd);
integer n = UInt(Rn);
integer m = UInt(Rm);
constant integer esize = 16;
constant integer datasize = esize;
integer elements = 1;
CompareOp cmp;
boolean abs;

case E:U:ac of
    when '000' cmp = CompareOp_EQ; abs = FALSE;
    when '010' cmp = CompareOp_GE; abs = FALSE;
    when '011' cmp = CompareOp_GE; abs = TRUE;
    when '110' cmp = CompareOp_GT; abs = FALSE;
    when '111' cmp = CompareOp_GT; abs = TRUE;
    otherwise UNDEFINED;
```

Scalar single-precision and double-precision

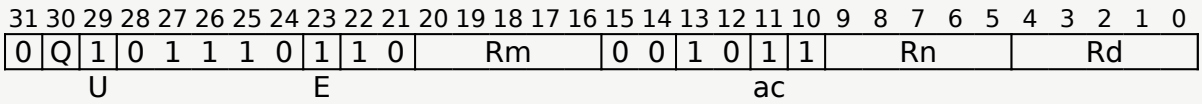


FACGT <V><d>, <V><n>, <V><m>

```
integer d = UInt(Rd);
integer n = UInt(Rn);
integer m = UInt(Rm);
constant integer esize = 32 << UInt(sz);
constant integer datasize = esize;
integer elements = 1;
CompareOp cmp;
boolean abs;

case E:U:ac of
    when '000' cmp = CompareOp_EQ; abs = FALSE;
    when '010' cmp = CompareOp_GE; abs = FALSE;
    when '011' cmp = CompareOp_GE; abs = TRUE;
    when '110' cmp = CompareOp_GT; abs = FALSE;
    when '111' cmp = CompareOp_GT; abs = TRUE;
    otherwise UNDEFINED;
```

Vector half precision
(FEAT_FP16)



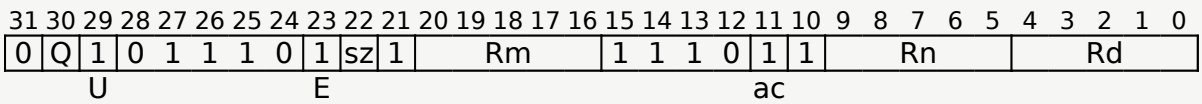
FACGT <Vd>.<T>, <Vn>.<T>, <Vm>.<T>

```
if !IsFeatureImplemented(FEAT_FP16) then UNDEFINED;

integer d = UInt(Rd);
integer n = UInt(Rn);
integer m = UInt(Rm);
constant integer esize = 16;
constant integer datasize = 64 << UInt(Q);
integer elements = datasize DIV esize;
CompareOp cmp;
boolean abs;

case E:U:ac of
    when '000' cmp = CompareOp_EQ; abs = FALSE;
    when '010' cmp = CompareOp_GE; abs = FALSE;
    when '011' cmp = CompareOp_GE; abs = TRUE;
    when '110' cmp = CompareOp_GT; abs = FALSE;
    when '111' cmp = CompareOp_GT; abs = TRUE;
    otherwise UNDEFINED;
```

Vector single-precision and double-precision



FACGT <Vd>.<T>, <Vn>.<T>, <Vm>.<T>

```
integer d = UInt(Rd);
integer n = UInt(Rn);
integer m = UInt(Rm);
if sz:Q == '10' then UNDEFINED;
constant integer esize = 32 << UInt(sz);
constant integer datasize = 64 << UInt(Q);
integer elements = datasize DIV esize;
CompareOp cmp;
boolean abs;

case E:U:ac of
    when '000' cmp = CompareOp_EQ; abs = FALSE;
    when '010' cmp = CompareOp_GE; abs = FALSE;
    when '011' cmp = CompareOp_GE; abs = TRUE;
    when '110' cmp = CompareOp_GT; abs = FALSE;
    when '111' cmp = CompareOp_GT; abs = TRUE;
    otherwise UNDEFINED;
```

Assembler Symbols

<Hd> Is the 16-bit name of the SIMD&FP destination register, encoded in the "Rd" field.

<Hn> Is the 16-bit name of the first SIMD&FP source register, encoded in the "Rn" field.

<Hm> Is the 16-bit name of the second SIMD&FP source register, encoded in the "Rm" field.

<V> Is a width specifier, encoded in “sz”:

sz	<V>
0	S
1	D

<d> Is the number of the SIMD&FP destination register, in the "Rd" field.

<n> Is the number of the first SIMD&FP source register, encoded in the "Rn" field.

<m> Is the number of the second SIMD&FP source register, encoded in the "Rm" field.

<Vd> Is the name of the SIMD&FP destination register, encoded in the "Rd" field.

<T>

For the half-precision variant: is an arrangement specifier, encoded in “Q”:

Q	<T>
0	4H
1	8H

For the single-precision and double-precision variant: is an arrangement specifier, encoded in “sz:Q”:

sz	Q	<T>
0	0	2S
0	1	4S
1	0	RESERVED
1	1	2D

<Vn> Is the name of the first SIMD&FP source register, encoded in the "Rn" field.

<Vm> Is the name of the second SIMD&FP source register, encoded in the "Rm" field.

Operation

```
CheckFPAdvSIMDEnabled64();
bits(datasize) operand1 = V[n, datasize];
bits(datasize) operand2 = V[m, datasize];

bits(esize) element1;
bits(esize) element2;
boolean test_passed;
FPCRType fpcr = FPCR[];
boolean merge = elements == 1 && IsMerging(fpcr);
bits(128) result = if merge then V[m, 128] else Zeros(128);

for e = 0 to elements-1
    element1 = Elem[operand1, e, esize];
    element2 = Elem[operand2, e, esize];
    if abs then
        element1 = FPAbs(element1);
        element2 = FPAbs(element2);
    case cmp of
        when CompareOp_EQ test_passed = FPCompareEQ(element1, element2,
        when CompareOp_GE test_passed = FPCompareGE(element1, element2,
        when CompareOp_GT test_passed = FPCompareGT(element1, element2,
    Elem[result, e, esize] = if test_passed then Ones(esize) else Zeros

V[d, 128] = result;
```

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