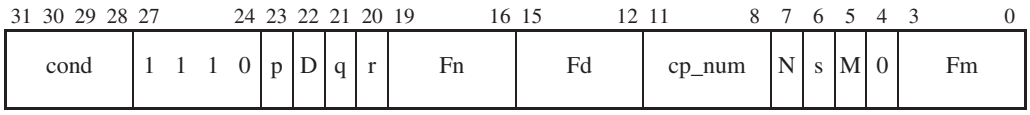
Chapter C3
VFP Instruction Set Overview

This chapter gives an overview of the VFP instruction set. It contains the following sections:

C3.1 Data-processing instructions

All VFP data-processing instructions are CDP instructions for coprocessors 10 or 11, with the following format:



p, q, r, s These bits collectively form the instruction's primary opcode. See Table C3-1 on page C3-3 for the assignment of these opcodes. When all of p, q, r and s are 1, the instruction is a two-operand *extension instruction*, with an extension opcode specified by the Fn and N bits.

Fd and D These bits normally specify the destination register of the instruction:

- For a single-precision instruction, Fd holds the top 4 bits of the register number and D holds the bottom bit.
- For a double-precision instruction, Fd holds the register number and D must be 0.

If D is 1 in a double-precision instruction, the instruction is UNDEFINED.

For multiply-accumulate instructions, this register is also the accumulate operand register. For comparison instructions, it is the first operand register rather than a destination register.

Fn and N These bits normally specify the first operand register of the instruction.

- For a single-precision instruction, Fn holds the top 4 bits of the register number and N holds the bottom bit.
- For a double-precision instruction, Fn holds the register number and N must be 0.

However, if p, q, r and s are all 1, the instruction is an extension instruction, and the Fn and N fields form an extension opcode instead of specifying a register. See Table C3-2 on page C3-4 for the assignment of these extension opcodes.

If N is 1 in a double-precision non-extension instruction, the instruction is UNDEFINED.

Fm and M These bits specify the second operand register of the instruction, or the only operand register for some extension instructions.

- For a single-precision instruction, Fm holds the top 4 bits of the register number and M holds the bottom bit.
- For a double-precision instruction, Fm holds the register number and M must be 0.

If M is 1 in a double-precision instruction, the instruction is UNDEFINED.

cp_num If cp_num is 0b1010 (coprocessor number 10), the instruction is a single-precision instruction. If cp_num is 0b1011 (coprocessor number 11), the instruction is a double-precision instruction.

For the instructions that convert between single-precision and double-precision (FCVTDS and FCVTSD), cp_num matches the source precision.

Table C3-1 and Table C3-2 on page C3-4 show the assignment of VFP data-processing opcodes. In these tables, Fd is used to mean a destination register of the appropriate precision, that is, Sd for single-precision instructions and Dd for double-precision instructions. Fn and Fm are used similarly.

Table C3-1 VFP data-processing primary opcodes

p	q	r	s	Instruction name cp_num=10	Instruction name cp_num=11	Instruction functionality
0	0	0	0	FMACS	FMACD	Fd = Fd + (Fn * Fm)
0	0	0	1	FNMACS	FNMACD	Fd = Fd - (Fn * Fm)
0	0	1	0	FMSCS	FMSCD	Fd = -Fd + (Fn * Fm)
0	0	1	1	FNMSCS	FNMSCD	Fd = -Fd - (Fn * Fm)
0	1	0	0	FMULS	FMULD	Fd = Fn * Fm
0	1	0	1	FNMULS	FNMULD	Fd = -(Fn * Fm)
0	1	1	0	FADDS	FADDD	Fd = Fn + Fm
0	1	1	1	FSUBS	FSUBD	Fd = Fn - Fm
1	0	0	0	FDIVS	FDIVD	Fd = Fn / Fm
1	0	0	1	-	-	UNDEFINED
1	0	1	0	-	-	UNDEFINED
1	0	1	1	-	-	UNDEFINED
1	1	0	0	-	-	UNDEFINED
1	1	0	1	-	-	UNDEFINED
1	1	1	0	-	-	UNDEFINED
1	1	1	1	See Table C3-2 on page C3-4	See Table C3-2 on page C3-4	Extension instructions

Table C3-2 VFP data-processing extension opcodes

Extension opcode		Instruction name		
Fn	**N**	**cp_num=10**	**cp_num=11**	**Instruction functionality**
0000	0	FCPYS	FCPYD	Fd = Fm
0000	1	FABSS	FABSD	Fd = abs(Fm)
0001	0	FNEGS	FNEGD	Fd = -Fm
0001	1	FSQRTS	FSQRTD	Fd = sqrt(Fm)
001x	x	-	-	UNDEFINED
0100	0	FCMPS	FCMPD	Compare Fd with Fm, no exceptions on quiet NaNs
0100	1	FCMPES	FCMPED	Compare Fd with Fm, with exceptions on quiet NaNs
0101	0	FCMPZS	FCMPZD	Compare Fd with 0, no exceptions on quiet NaNs
0101	1	FCMPEZS	FCMPEZD	Compare Fd with 0, with exceptions on quiet NaNs
0110	x	-	-	UNDEFINED
0111	0	-	-	UNDEFINED
0111	1	FCVTDS	FCVTSD	Single ↔ double-precision conversions
1000	0	FUITOS	FUITOD	Unsigned integer → floating-point conversions
1000	1	FSITOS	FSITOD	Signed integer → floating-point conversions
1001	x	-	-	UNDEFINED
101x	x	-	-	UNDEFINED
1100	0	FTOUIS	FTOUID	Floating-point → unsigned integer conversions
1100	1	FTOUIZS	FTOUIZD	Floating-point → unsigned integer conversions, RZ mode
1101	0	FTOSIS	FTOSID	Floating-point → signed integer conversions
1101	1	FTOSIZS	FTOSIZD	Floating-point → signed integer conversions, RZ mode
111x	x	-	-	UNDEFINED

C3.1.1 Basic arithmetic instructions and square root

The FADDS, FSUBS, FMULS, FDIVS, and FSQRTS instructions provide the four basic arithmetic operations and square root on single-precision values. Similarly, the FADDD, FSUBD, FMULD, FDIVD, and FSQRTD instructions supply these operations on double-precision values. In addition, the FNMULS and FNMULD instructions supply negated multiplications in single and double-precision respectively. Their results are precisely equivalent to those of performing an FMULS or FMULD instruction followed by an FNEGS or FNEGD instruction (which inverts the sign of the result).

All of these instructions can be made to operate on short vectors by setting the FPSCR LEN and STRIDE fields appropriately (see Chapter C5 *VFP Addressing Modes* for details).

The addition, subtraction, multiplication, division, and square root operations performed by all these instructions are always treated as floating-point operations, both for NaN handling and Flush-to-zero mode. In particular, signaling NaN operands cause Invalid Operation exceptions, and in Flush-to-zero mode, denormalized operands are treated as zero and sufficiently small results are forced to zero.

The negation operations performed by these instructions are not treated as floating-point operations. The sign bit is always inverted, even when the operand is a NaN.

C3.1.2 Multiply-accumulate instructions

FMACS, FMACD, FNMACS, FNMACD, FMSCS, FMSCD, FNMSCS, and FNMSCD are *multiply-accumulate* instructions. They multiply their two main operands, possibly invert the sign bit of the product, add or subtract the value in the destination register and write the result back to the destination register. They are in all respects equivalent to the following sequences of basic arithmetic and negation instructions:

```
FMACS  Sd,Sn,Sm:  FMULS   St,Sn,Sm
                  FADDS   Sd,Sd,St

FMACD  Dd,Dn,Dm:  FMULD   Dt,Dn,Dm
                  FADDD   Dd,Dd,Dt

FNMACS Sd,Sn,Sm:  FMULS   St,Sn,Sm
                  FNEGS   St,St
                  FADDS   Sd,Sd,St

FNMACD Dd,Dn,Dm:  FMULD   Dt,Dn,Dm
                  FNEGD   Dt,Dt
                  FADDD   Dd,Dd,Dt

FMSCS  Sd,Sn,Sm:  FMULS   St,Sn,Sm
                  FNEGS   Sd,Sd
                  FADDS   Sd,Sd,St

FMSCD  Dd,Dn,Dm:  FMULD   Dt,Dn,Dm
                  FNEGD   Dd,Dd
                  FADDD   Dd,Dd,Dt

FNMSCS Sd,Sn,Sm:  FMULS   St,Sn,Sm
                  FNEGS   St,St
                  FNEGS   Sd,Sd
```



```
                    FADDS   Sd,Sd,St

FNMSCD Dd,Dn,Dm:    FMULD   Dt,Dn,Dm
                    FNEGD   Dt,Dt
                    FNEGD   Dd,Dd
                    FADDD   Dd,Dd,Dt
```

where St or Dt describes a notional register used to hold intermediate results, treated as being a scalar if Sd or Dd is a scalar and a vector if Sd or Dd is a vector.

Note

This implies that each multiply-accumulate operation involves two roundings:

- one on the multiplication result
- one on the result of the final addition or subtraction.

Both of these roundings are performed fully and as defined by the IEEE 754 standard. In particular, these instructions do not specify *fused multiply-accumulates* as used in a number of other architectures.

All of these instructions can be made to operate on short vectors by setting the FPSCR LEN and STRIDE fields appropriately (see Chapter C5 *VFP Addressing Modes* for details). The multiply and add operations performed by all these instructions are always treated as floating-point operations, both for NaN handling and Flush-to-zero mode. In particular, signaling NaN operands cause Invalid Operation exceptions, and in Flush-to-zero mode, denormalized operands are treated as zero and sufficiently small results are forced to zero.

The negation operations performed by these instructions are not treated as floating-point operations. The sign bit is always inverted, even when the operand is a NaN.

C3.1.3 Comparison instructions

The `FCMPS`, `FCMPD`, `FCMPES`, and `FCMPED` instructions perform comparisons between two register values. The `FCMPZS`, `FCMPZD`, `FCMPEZS`, and `FCMPEZD` instructions perform comparisons between a register value and the constant +0.

The IEEE 754 standard specifies that precisely one of four relationships holds between any two values being compared. These are as follows:

- Two values are considered equal if any of the following conditions holds:
  - They are both numeric and have the same numerical value. This usually means that they have precisely the same representation, but also includes the case that one is +0 and the other is -0.
  - They are both $+\infty$ (plus infinity).
  - They are both $-\infty$ (minus infinity).
- The first value is considered less than the second value if any of the following conditions holds:
  - They are both numeric and the numeric value of the first is less than that of the second.
  - The first is $-\infty$ (minus infinity) and the second is numeric.

  - The first is numeric and the second is +∞ (plus infinity).
  - The first is −∞ (minus infinity) and the second is +∞ (plus infinity).
- The first value is considered greater than the second value if any of the following conditions holds:
  - They are both numeric and the numeric value of the first is greater than that of the second.
  - The first is +∞ (plus infinity) and the second is numeric.
  - The first is numeric and the second is −∞ (minus infinity).
  - The first is +∞ (plus infinity) and the second is −∞ (minus infinity).
- Two values are *unordered* if either or both of them are NaNs.

Note

If both values are the same NaN, the comparison result is *unordered*, not *equal*. If an exact bit-by-bit comparison is wanted, the ARM® comparison instructions must be used rather than VFP comparison instructions, both for this reason and because +0 and -0 compare as equal.

For all the comparison instructions, the result of the comparison is placed in the FPSCR flags, as shown in Table C3-3:

Table C3-3 VFP comparison flag values

Comparison result	N	Z	C	V
Equal	0	1	1	0
Less than	1	0	0	0
Greater than	0	0	1	0
Unordered	0	0	1	1

These FPSCR flag values need to be copied to the ARM CPSR flags before ARM conditional execution can be based on them. For this purpose, a special form of the `FMRX` instruction (called `FMSTAT`) is used. This is described in *System register transfer instructions* on page C3-21.

When the result of the comparison is *unordered*, it is possible that the comparison can also generate an Invalid Operation exception because of the NaN operand(s). These instructions supply two distinct forms of Invalid Operation exception generation:

- The `FCMPS`, `FCMPD`, `FCMPZS`, and `FCMPZD` instructions have the normal behavior of generating an Invalid Operation exception when either or both of their operands are signaling NaNs. If neither operand is a signaling NaN, but one or both are quiet NaNs, they generate an *unordered* result without an accompanying Invalid Operation exception.

- The FCMPES, FCMPED, FCMPEZS, and FCMPEZD instructions generate an Invalid Operation exception when either or both of their operands are NaNs, regardless of whether they are signaling or quiet NaNs. It is not possible to get an *unordered* result from these instructions without an accompanying Invalid Operation exception.

The VFP comparison instructions always treat their operands as scalars, regardless of the settings of the FPSCR LEN and STRIDE fields.

The operations performed by all these instructions are always treated as floating-point operations, both for NaN handling and Flush-to-zero mode. In particular, signaling NaN operands cause Invalid Operand exceptions, and in Flush-to-zero mode, denormalized operands are treated as zero.

Testing the IEEE 754 predicates

The IEEE 754 standard specifies two ways in which a floating-point comparison can deliver its results:

- As a *condition code* result, identifying one of the four relations:
  - equal
  - less than
  - greater than
  - unordered.
- As a true-or-false result to one of twenty-six *predicates*, each of which specifies a particular test on the values. Six of these are the standard ==, !=, <, <=, > and >= comparisons, used in common languages like C, C++ and related languages.

The VFP architecture uses the first approach. However, its condition code results have been carefully chosen to allow ARM conditional execution to test as many of the predicates as possible after a sequence of a VFP comparison instruction and an FMSTAT instruction. This includes all six of the commonly-used predicates.

Table C3-4 shows how each predicate must be tested to get the correct results according to the IEEE 754 standard:

Table C3-4 VFP predicate testing

Common language condition	IEEE predicate	Instruction type	ARM condition
==	=	FCMP	EQ
!=	?<>	FCMP	NE
>	>	FCMPE	GT
>=	>=	FCMPE	GE
<	<	FCMPE	MI or CC
<=	<=	FCMPE	LS

Table C3-4 VFP predicate testing (continued)

Common language condition	IEEE predicate	Instruction type	ARM condition
	?	FCMP	VS
	<>	FCMPE	Two conditions
	<=>	FCMPE	VC
	?>	FCMP	HI
	?>=	FCMP	PL or CS
	?<	FCMP	LT
	?<=	FCMP	LE
	?=	FCMP	Two conditions
	NOT(>)	FCMPE	LE
	NOT(>=)	FCMPE	LT
	NOT(<)	FCMPE	PL or CS
	NOT(<=)	FCMPE	HI
	NOT(?)	FCMP	VC
	NOT(<>)	FCMPE	Two conditions
	NOT(<=>)	FCMPE	VS
	NOT(?>)	FCMP	LS
	NOT(?>=)	FCMP	MI or CC
	NOT(?<)	FCMP	GE
	NOT(?<=)	FCMP	GT
	NOT(?=)	FCMP	Two conditions

In each case, the two main choices to be made are:

- Whether to use an FCMP-type instruction (that is, the appropriate one of `FCMPS`, `FCMPD`, `FCMPZS` or `FCMPZD`) or an FCMPE-type instruction (the appropriate one of `FCMPES`, `FCMPED`, `FCMPEZS` or `FCMPEZD`). This choice causes the predicate to have the correct behavior with regard to Invalid Operation exceptions.

- Which ARM condition is to be used. This is not always obvious. For example, a standard < comparison on floating-point numbers must use the ARM MI or LO/CC condition, not LT, despite the fact that floating-point comparisons are always signed.

  If this column contains *two conditions*, no single ARM condition can be used to test the predicate. Each of these predicates can be tested using a suitable combination of two ARM conditions, in several different ways. For example, the <> predicate can be tested by checking that NE and VC are both true, or that either of GT and MI is true.

C3.1.4 Conversion instructions

All of the VFP conversion instructions always treat their operands as scalars, regardless of the settings of the FPSCR LEN and STRIDE fields.

Conversions between single and double-precision

The FCVTDS and FCVTSD instructions perform conversions between single-precision and double-precision values. FCVTDS converts single-precision to double-precision and is a coprocessor 10 instruction, while FCVTSD converts double-precision to single-precision and is a coprocessor 11 instruction.

The FCVTDS and FCVTSD conversions are always treated as floating-point operations, both for NaN handling and Flush-to-zero mode. In particular, signaling NaN operands cause Invalid Operand exceptions, and in Flush-to-zero mode, denormalized operands are treated as zero.

The only exception possible for FCVTDS is an Invalid Operation exception caused by a signaling NaN operand, as single-precision numbers can always be represented exactly in double-precision. FCVTSD can additionally generate Overflow, Underflow and/or Inexact exceptions.

Conversions from floating-point to integers

The FTOSIS and FTOSID instructions convert floating-point values to signed integers, and the FTOUIS and FTOUID instructions convert floating-point values to unsigned integers, using the rounding mode specified by the FPSCR.

Variants of these instructions called FTOSIZS, FTOSIZD, FTOUIZS, and FTOUIZD perform similar conversions, but using Round towards Zero mode. These are useful because C and related languages specify that floating-point → integer conversions use this mode, whereas almost all other operations normally use Round to Nearest mode. Using these instructions avoids the need to change the FPSCR rounding mode every time a floating-point → integer conversion is wanted.

All of the floating-point → integer conversion instructions place their integer result in a single-precision register. This result can then be used in any of the following ways:

- store it to memory using FSTS or FSTMS
- transfer it to an ARM register using FMRS
- convert it to a floating-point number using any of FSITOS, FSITOD, FUITOS or FUITOD.

The operations performed by all these instructions are always treated as floating-point operations, both for NaN handling and Flush-to-zero mode. In particular, signaling NaN operands cause Invalid Operand exceptions, and in Flush-to-zero mode, denormalized operands are treated as zero.

Most exceptional conditions that can occur during these instructions are signaled as Invalid Operation exceptions. These cannot produce the normal quiet NaN value as their result, as the destination is an integer. Instead, the following list of values that generate Invalid Operation exceptions also specifies the integer default result in each case:

- If the operand is numeric, but converting it to an integer using the appropriate rounding mode would produce an integer that is greater than the maximum possible destination integer, the default result is the maximum possible destination integer.

- If the operand is numeric, but converting it to an integer using the appropriate rounding mode would produce an integer that is less than the minimum possible destination integer, the default result is the minimum possible destination integer.
- If the operand is $+\infty$ (plus infinity), the default result is the maximum possible destination integer.
- If the operand is $-\infty$ (minus infinity), the default result is the minimum possible destination integer.
- If the operand is a NaN (either signaling or quiet), the default result is 0.

Apart from these Invalid Operation exceptions, the only exceptions that can be produced by the floating-point $\rightarrow$ integer conversions are Inexact exceptions.

Conversions from integers to floating-point

The `FSITOS` and `FSITOD` instructions convert signed integers to floating-point values, and the `FUITOS` and `FUITOD` instructions convert unsigned integers to floating-point values. All of them take their integer operand from a single-precision register. This operand can have been placed in the register earlier in any of the following ways:

- loading it from memory using `FLDS` or `FLDMS`
- transferring it from an ARM register using `FMSR`
- converting a floating-point number to an integer using any of `FTOSIS`, `FTOSID`, `FTOSIZS`, `FTOSIZD`, `FTOUIS`, `FTOUID`, `FTOUIZS`, or `FTOUIZD`.

When an integer 0 is converted to floating-point, the result is +0. For the `FSITOS` and `FUITOS` instructions, some integer operands that exceed 2^{24} in magnitude cannot be converted exactly. Conversions of these operands are rounded according to the rounding mode specified in the FPSCR, with an Inexact exception being generated. Otherwise, no exceptions are possible with the integer $\rightarrow$ floating-point conversions.

C3.1.5 Copy, negation and absolute value instructions

The FCPYS and FCPYD instructions perform an exact copy of a floating-point value from one register to another.

The FNEGS and FNEGD instructions do the same as FCPYS and FCPYD, except that they invert the sign bit during the copy. This negates numerical values and infinities, in the way described in the Appendix to the IEEE 754 standard.

The FABSS and FABSD instructions do the same as FCPYS and FCPYD, except that they change the sign bit to 0 during the copy. This takes the absolute value of numerical values and infinities, in the way described in the Appendix to the IEEE 754 standard.

All of these instructions can be made to operate on short vectors by setting the FPSCR LEN and STRIDE fields appropriately (see Chapter C5 *VFP Addressing Modes*).

The IEEE 754 standard and its Appendix allow all these operations to be treated as non floating-point operations with regard to NaN handling. The VFP architecture requires this to be done. In particular, this implies the following:

- The VFP architecture requires these instructions *not* to generate Invalid Operation when their operands are signaling NaNs.
- The results of these instructions are generated by copying their operands (with appropriate sign bit adjustments), even when their operands are NaNs. This overrides the normal rules for generating the results of instructions with one or more NaN operands (described in *NaNs* on page C2-5).

In addition, the VFP architecture requires these instructions to be treated as non floating-point operations with regard to Flush-to-zero mode. In Flush-to-zero mode, they copy denormalized operands in the same way as they do in normal mode, and do not treat the operands as zero.

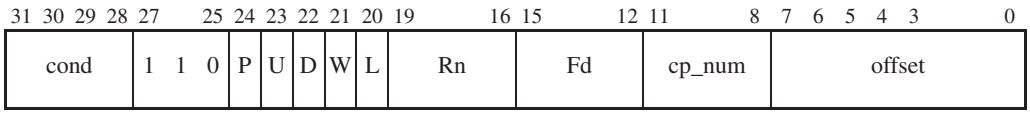
Note

Calculating the value of -x using FNEGS or FNEGD does not produce exactly the same results as calculating either (+0 - x) or (-0 - x) using FSUBS or FSUBD. The differences are:

- FSUBS or FSUBD produces an Invalid Operation exception if x is a signaling NaN, whereas FNEGS or FNEGD produces x with its sign bit inverted, without an exception.
- FSUBS or FSUBD produces an exact copy of x if x is a quiet NaN, whereas FNEGS or FNEGD produces x with its sign bit inverted.
- FNEGS or FNEGD applied to a zero always produces an oppositely signed zero. Calculating the value of (+0 - x) using FSUBS or FSUBD does this in RM rounding mode, but always produces +0 in RN, RP or RZ rounding mode. Calculating (-0 - x) always produces -0 in RM rounding mode, and produces an oppositely signed zero in RN, RP or RZ rounding mode.
- In Flush-to-zero mode, the calculation using FSUBS or FSUBD treats denormalized operands as zero, and therefore produce a zero result if x is denormalized. FNEGS or FNEGD ignore Flush-to-zero mode and produce a result of x with its sign bit inverted.

C3.2 Load and Store instructions

All VFP Load and Store instructions are LDC and STC instructions respectively for coprocessors 10 and 11, with the following format:



P, U, W These bits specify an addressing mode for the LDC or STC instruction, as described in *ARM Addressing Mode 5 - Load and Store Coprocessor* on page A5-49. In addition, a VFP implementation uses them to determine which load/store operation is required, as shown in Table C3-1 on page C3-15.

Fd and D These bits specify the destination floating-point register of a load instruction, or the source floating-point register of a store instruction.

- For a single-precision instruction, Fd holds the top 4 bits of the register number and D holds the bottom bit.
- For a double-precision instruction, Fd holds the register number and D must be 0.

If D is 1 in a double-precision instruction, the instruction is UNDEFINED.

For Load Multiple and Store Multiple instructions, the register specified by these fields is the lowest-numbered register to be transferred. Subsequent registers are transferred in order of register number, up to the number of registers determined by the offset field. If this would result in a register after S31 or D15 being transferred, the results are UNPREDICTABLE.

L bit This bit determines whether the instruction is a load (L == 1) or a store (L == 0).

Rn This specifies the ARM register used as the base register for the address calculation, as described in *ARM Addressing Mode 5 - Load and Store Coprocessor* on page A5-49.

cp_num If cp_num is 0b1010 (coprocessor number 10), the instruction is a single-precision instruction. If cp_num is 0b1011 (coprocessor number 11), the instruction is either a double-precision instruction or one of the instructions used to handle values of unknown precision (see *Storing and reloading values of unknown precision* on page C2-18).

offset These bits specify the word offset which is applied to the base register value to obtain the starting memory address for the transfer, as described in *ARM Addressing Mode 5 - Load and Store Coprocessor* on page A5-49.

The least significant bit of this offset also helps to determine which load/store operation is required, as shown in Table C3-1 on page C3-15. In addition, for Load Multiple and Store Multiple instructions, the offset determines how many registers are to be transferred.

Table C3-1 on page C3-15 shows how the name and other details of the instruction are determined from the P, U, W, and L bits and the cp_num and offset fields:

Table C3-1 VFP load and store instructions

PUW	cp_num	offset [0]	Instruction L==0	Instruction L==1	Addr mode	Registers transferred
0 0 0	x	x	TWO REG TRANSFER	-	-	See *Two-register transfer instructions* on page C3-22
0 0 1	x	x	UNDEFINED	-	-	-
0 1 0	0b1010	x	FSTMS	FLDMS	Unindexed	(offset) single-precision registers
0 1 0	0b1011	0	FSTMD	FLDMD	Unindexed	(offset)/2 double-precision registers
0 1 0	0b1011	1	FSTMX	FLDMX	Unindexed	(offset-1)/2 double-precision registers
0 1 1	0b1010	x	FSTMS	FLDMS	Increment	(offset) single-precision registers
0 1 1	0b1011	0	FSTMD	FLDMD	Increment	(offset)/2 double-precision registers
0 1 1	0b1011	1	FSTMX	FLDMX	Increment	(offset-1)/2 double-precision registers
1 0 0	0b1010	x	FSTS	FLDS	Negative offset	One single-precision register
1 0 0	0b1011	x	FSTD	FLDD	Negative offset	One double-precision register
1 0 1	0b1010	x	FSTMS	FLDMS	Decrement	(offset) single-precision registers
1 0 1	0b1011	0	FSTMD	FLDMD	Decrement	(offset)/2 double-precision registers
1 0 1	0b1011	1	FSTMX	FLDMX	Decrement	(offset-1)/2 double-precision registers
1 1 0	0b1010	x	FSTS	FLDS	Positive offset	One single-precision register
1 1 0	0b1011	x	FSTD	FLDD	Positive offset	One double-precision register
1 1 1	x	x	UNDEFINED	-	-	-

All load instructions perform a copy of the loaded value(s) from memory, and all store instructions perform a copy of the stored value(s) to memory. No exceptions are ever raised and the value(s) transferred are not changed, except possibly for a reversible conversion to the internal register format of an implementation. The copy is treated as a non floating-point operation for the purposes of NaN handling and Flush-to-zero mode. In particular, the VFP architecture requires:

- a load or store of a signaling NaN not to raise an Invalid Operation exception, nor to change the signaling NaN into a quiet NaN
- a load or store of a denormalized number in Flush-to-zero mode not to change it into zero.

C3.2.1 Load/store one value

The FLDS and FSTS instructions allow single-precision values and 32-bit integers to be loaded and stored, and the FLDD and FSTD instructions allow double-precision values to be loaded and stored. Each of these instructions transfers just one register of the type concerned.

Of the addressing modes described in *ARM Addressing Mode 5 - Load and Store Coprocessor* on page A5-49, only the Immediate offset mode (see *Load and Store Coprocessor - Immediate offset* on page A5-51) is allowed for these instructions. This addressing mode allows the address to be specified by the base register value Rn, plus or minus an immediate offset which lies in the range 0 to 1020 and is a multiple of 4. No base register write-back is available.

C3.2.2 Load/store multiple values

The FLDMS and FSTMS instructions allow multiple single-precision values and/or integers to be loaded and stored, and the FLDMD and FSTMD instructions allow multiple double-precision values to be loaded and stored.

Each of these instructions transfers a number of registers determined by the offset field of the instruction. The offset field is equal to the total number of words transferred for all of these instructions, that is, it is the number of registers for FLDMS and FSTMS, and twice the number of registers for FLDMD and FSTMD.

In addition, the FSTMX instruction can be used to store double-precision registers when it is not known whether they contain single-precision or double-precision values, in a format that allows a matching FLDMX instruction to reload them correctly (see *Storing and reloading values of unknown precision* on page C2-18). In these instructions, the offset field is twice the number of double-precision registers to be transferred, plus one. This is the maximum number of words these instructions can transfer. Some implementations transfer one fewer word than this maximum, leaving a memory word unused.

The FSTMX and FLDMX instructions are encoded as coprocessor 11 instructions, like FSTMD and FLDMD. They are distinguished from the latter by the fact that the offset field is odd in FSTMX and FLDMX instructions, and even in FSTMD and FLDMD instructions.

The FSTMX and FLDMX instructions are the only coprocessor 11 instructions which are present in single-precision-only variants (non-D variants) of the VFP architecture. To aid software portability, it is recommended that programs written for such variants must use them in the same situations as a program written for a D variant would, even though the registers are known to hold single-precision values in non-D variants. The main situations affected are when storing and reloading callee-save registers, and in process swap code.

Three addressing modes are available for these instructions:

- Unindexed mode is the same as the LDC/STC Unindexed addressing mode (see *Load and Store Coprocessor - Unindexed* on page A5-54). The base register Rn determines the starting address for the transfer and is left unchanged.

  The offset field determines the number of registers to transfer, but does not affect the address calculations.

- Increment mode is the same as the LDC/STC Immediate post-indexed addressing mode with a positive offset (see *Load and Store Coprocessor - Immediate post-indexed* on page A5-53). The base register Rn determines the starting address for the transfer. The offset field determines the number of registers to transfer, and is also multiplied by 4, added to the value of Rn and written back to Rn.

  After the transfer, Rn therefore points to the memory word immediately after the last word to be transferred (or the last word that *could* have been transferred in the case of FSTMX and FLDMX). This means that it is suitable for pushing values on to an Empty Ascending stack or for popping them from a Full Descending stack.

- Decrement mode is the same as the LDC/STC Immediate pre-indexed addressing mode with a negative offset (see *Load and Store Coprocessor - Immediate pre-indexed* on page A5-52). The offset is multiplied by 4 and added to the value of the base register Rn to determine the starting address for the transfer, and this starting address is written back to Rn. The offset field also determines the number of registers to transfer.

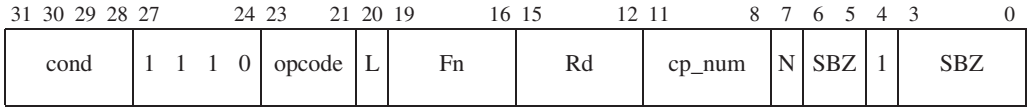
  Before the transfer, Rn therefore points to the memory word immediately after the last word to be transferred (or the last word that *could* have been transferred in the case of FSTMX and FLDMX). This means that it is suitable for pushing values on to a Full Descending stack or for popping them from an Empty Ascending stack.

Note

There are no short vector forms of the load and store instructions as such, but the FLDMS, FLDMD, FSTMS and FSTMD instructions can be used to load and store many of the possible short vectors. However, note that short vectors wrap around within banks as described in Chapter C5 *VFP Addressing Modes*, while the load multiple and store multiple instructions simply advance linearly through S0-S31 or D0-D15. If a short vector that wraps around is to be loaded or stored, two or more instructions are needed.

C3.3 Single register transfer instructions

All VFP single-register transfer instructions are MCR and MRC instructions for coprocessors 10 and 11, with the following format:



opcode This determines which register transfer operation is required, as shown in Table C3-2 on page C3-19.

L bit This bit determines the direction of the transfer:

L == 0 From an ARM register to a VFP register. (An MCR instruction.)

L == 1 From a VFP register to an ARM register. (An MRC instruction.)

Fn and N bit These bits specify the VFP register involved in the transfer:

- For a single-precision register, Fn holds the top 4 bits of the register number, and N holds the bottom bit.
- For a double-precision register, Fn holds the register number, and N must be 0.
- For a system register, Fn and N specify the register as shown in Table C3-3 on page C3-19.

If N is 1 in an instruction that transfers a double-precision register, the instruction is UNDEFINED.

Rd This specifies the ARM register involved in the transfer. If Rd is R15, the behavior is as specified for the generic ARM instruction:

- For an MCR instruction (L == 0), the instruction is UNPREDICTABLE.
- For an MRC instruction (L == 1), the top 4 bits of the value transferred are placed in the ARM condition code flags, and the remaining 28 bits are discarded. The FMSTAT instruction is the only VFP instruction that uses this behavior, enabling the transfer of comparison results to the ARM. All other MRC instructions where Rd is R15 are UNPREDICTABLE.

cp_num If cp_num is 0b1010 (coprocessor number 10), the instruction is a single-precision or system register transfer.

If cp_num is 0b1011 (coprocessor number 11), the instruction is a double-precision register transfer.

Table C3-2 shows the assignment of register transfer opcodes and other details of the instructions:

Table C3-2 VFP single register transfer instructions

opcode	cp_num	L	Instruction name	Instruction functionality
000	0b1010	0	FMSR	Sn = Rd
000	0b1010	1	FMRS	Rd = Sn
000	0b1011	0	FMDLR	Dn[31:0] = Rd
000	0b1011	1	FMRDL	Rd = Dn[31:0]
001	0b1010	x	-	UNDEFINED
001	0b1011	0	FMDHR	Dn[63:32] = Rd
001	0b1011	1	FMRDH	Rd = Dn[63:32]
01x	0b101x	x	-	UNDEFINED
10x	0b101x	x	-	UNDEFINED
110	0b101x	x	-	UNDEFINED
111	0b1010	0	FMXR	SystemReg(Fn,N) = Rd
111	0b1010	1	FMRX	Rd = SystemReg(Fn,N)
111	0b1011	x	-	UNDEFINED

Table C3-3 shows how system registers are encoded in FMXR and FMRX instructions:

Table C3-3 VFP system register encodings

Fn	N	System register
0b0000	0	FPSID
0b0001	0	FPSCR
0b1000	0	FPEXC

Encodings that are not shown in this table are:

- Reserved for future expansion if the top bit of Fn is 0. FMXR and FMRX instructions using these encodings are UNDEFINED.
- Reserved for additional SUB-ARCHITECTURE DEFINED system registers if the top bit of Fn is 1. FMXR and FMRX instructions using these encodings are SUB-ARCHITECTURE DEFINED.

C3.3.1 General-purpose single register transfer instructions

The FMRS instruction allows a single-precision value or a 32-bit integer in a single-precision register to be transferred to an ARM register, and the FMSR instruction allows a similar transfer from an ARM register to a single-precision register.

The FMRDH and FMRDL instructions allow a double-precision value in a double-precision register to be transferred to a pair of ARM registers. The FMRDH instruction transfers the most significant word of the double-precision value, which contains the sign, exponent and 20 most significant fraction bits. The FMRDL instruction transfers the least significant word, which contains the remaining fraction bits.

Similarly, the FMDHR and FMDLR instructions allow a double-precision value in a pair of ARM registers to be transferred to a double-precision register. FMDHR transfers the most significant word and FMDLR transfers the least significant word.

Note

The FMDHR and FMDLR instructions must be used in pairs, writing to the same double-precision register. These need not be executed consecutively, but while one of a pair has been executed and the other has not, the only valid uses of the destination double-precision register are:

- as the destination register of the second instruction of the pair
- storing it with FSTMX and reloading it with FLDMX, and using it for other purposes between the store and the reload.

All of these instructions always treat their floating-point operand as a scalar, regardless of the settings of the FPSCR LEN and STRIDE fields.

The register transfer performed is always a simple copy. No exceptions are ever raised and the value transferred is not changed, except possibly for a reversible conversion to or from the internal register format of an implementation.

The copy is treated as a non floating-point operation for the purposes of NaN handling and Flush-to-zero mode. In particular, the VFP architecture requires:

- a register transfer of a signaling NaN not to raise an Invalid Operation exception, nor to change the signaling NaN into a quiet NaN
- a register transfer of a denormalized number in Flush-to-zero mode not to change it into zero.

C3.3.2 System register transfer instructions

The FMRX instruction transfers a system register value to an ARM register, and the FMXR instruction transfers an ARM register value to a system register. Their exact effects depend on the definition of the system register concerned. For more details, see *System registers* on page C2-21 for the architecturally defined system registers, or sub-architecture documentation for SUB-ARCHITECTURE DEFINED system registers.

These instructions are *serializing* instructions.

When an FMXR or FMRX instruction is executed to access the FPEXC/FPSID, the register transfer is delayed until:

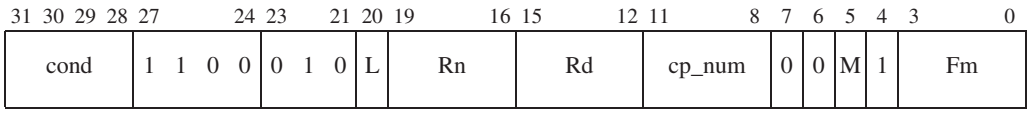
- all floating-point operations in progress have determined whether they are going to generate an exception
- all effects of floating-point operations in progress on sub-architectural register contents required to enable any software processing of these floating point operations have occurred
- all floating-point operations in progress are no longer affected by changes to system register contents (for example, by rounding mode or Flush-to-zero mode changes).

When an FMXR or FMRX instruction is executed to access the FPSCR, the register transfer is delayed until:

- all floating-point operations in progress have determined whether they are going to generate an exception
- any trapped exception handling or other software processing of floating-point operations in progress has completed
- all effects of floating-point operations in progress on system register contents (such as setting cumulative exception flags for untrapped exceptions) have occurred
- all floating-point operations in progress are no longer affected by changes to system register contents (for example, by rounding mode or Flush-to-zero mode changes).

C3.4 Two-register transfer instructions

All VFP two-register transfer instructions are `MCRR` and `MRRC` instructions for coprocessors 10 and 11, with the following format:



L bit This bit determines the direction of the transfer:

L == 0 From two ARM registers to a VFP register. (An `MCRR` instruction.)

L == 1 From a VFP register to two ARM registers. (An `MRRC` instruction.)

Fm and M bit

These bits specify the VFP register, or register pair, involved in the transfer:

- For a pair of single-precision registers, Fm holds the top four bits of the register number, and M holds the bottom bit.
- For a double-precision register, Fm holds the register number, and M must be 0.

If M is 1 in an instruction that transfers a double-precision register, the instruction is UNDEFINED.

Rn Specifies the ARM register for the upper half of a double-precision register, or for the Fm single-precision VFP register.

If Rn is R15, the behavior is UNPREDICTABLE.

Rd Specifies the ARM register for the lower half of a double-precision register, or for the (Fm+1) single-precision VFP register.

If Rd is R15, the behavior is UNPREDICTABLE.

cp_num If cp_num is 0b1010 (coprocessor number 10), the instruction is two single-precision register transfers.

If cp_num is 0b1011 (coprocessor number 11), the instruction is a double-precision register transfer.

Table C3-4 shows details of the instructions:

Table C3-4 VFP two register transfer instructions

cp_num	L	Instruction name	Instruction functionality
0b1010	0	FMSRR	Fm = Rn, (Fm+1) = Rd
0b1010	1	FMRRS	Rn = Fm, Rd = (Fm+1)
0b1011	0	FMDRR	Fm[31:0] = Rd, Fm[63:32] = Rn
0b1011	1	FMRRD	Rd = Fm[31:0], Rn = Fm[63:32]

The FMRRS instruction allows two single-precision values, or 32-bit integers, in two consecutively-numbered single-precision registers to be transferred to two ARM registers. The FMSRR instruction allows a similar transfer from two ARM registers to two consecutively-numbered single-precision registers. The ARM registers do not have to be contiguous.

The FMRRD instruction allows a double-precision value in a double-precision register to be transferred to two ARM registers. The ARM registers do not have to be contiguous

Similarly, the FMDRR instruction allows a double-precision value in two ARM registers to be transferred to a VFP double-precision register. The ARM registers do not have to be contiguous.

All of these instructions always treat their floating-point operand as a scalar, regardless of the settings of the FPSCR LEN and STRIDE fields.

The register transfer performed is always a simple copy. No exceptions are ever raised and the value transferred is not changed, except possibly for a reversible conversion to or from the internal register format of an implementation.

The copy is treated as a non floating-point operation for the purposes of NaN handling and Flush-to-zero mode. In particular, the VFP architecture requires:

- a register transfer of a signaling NaN not to raise an Invalid Operation exception, nor to change the signaling NaN into a quiet NaN
- a register transfer of a denormalized number in Flush-to-zero mode not to change it into zero.

