Floating Point

Operations Supported

Only the most basic floating point operations are supported with hardware. Supported operations include addition, subtraction, multiplication, division, absolute value, integer to float and float to integer conversions. Also supported are comparison operations. There are also a number of control and status instructions.

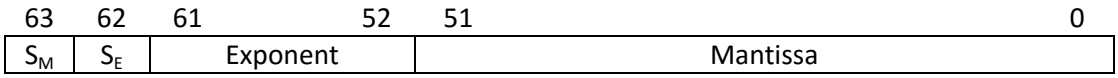
Supported Operations:

Mnemonic	Precision	Clocks	Operation
FADD	S,D	4	addition
FSUB	S,D	4	subtraction
FMUL	S,D	4	multiplication
FDIV	S,D	12,21	division
FABS	S,D	1	absolute value
FNEG	S,D	1	negation
FTOI	S,D	2	float to integer
ITOF	S,D	2	integer to float
FSIGN	S,D	1	sign of value
FMAN	S,D	1	mantissa of value
FSTAT	-	1	get status register
FRM	-	1	set rounding mode
FTX	-	1	trigger exception
TCX	-	1	clear exception
TDX	-	1	disable exception
FEX	-	1	enable exception
FCMP	S, D	1	comparison
FTST	S, D	1	test against zero

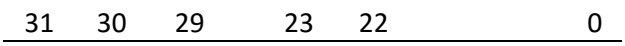
Representation

The floating point format is an IEEE-754 representation for both single and double precision. Briefly,

Double Precision Format:



Single Precision Format:



S_M	S_E	Exponent	Mantissa

S_M – sign of mantissa

S_E – sign of exponent

The exponent and mantissa are both represented as two's complement numbers, however the sign bit of the exponent is inverted.

S_eEEEEEEEEE	
11111111111	Maximum exponent
….	
01111111111	exponent of zero
….	
00000000000	Minimum exponent

The exponent ranges from -1024 to +1023 for double precision numbers

If the core is built with the 32 bit data-bus 64 bit double precision floating point is unavailable.

Floating point comparisons and tests are executed on the integer ALU. This allows a comparison operation to proceed in parallel with another floating point operation.

Performance

Generally, double precision operations are just as fast as single precision operations with the exception of the divide operation which takes multiple clock cycles.

The floating point divider uses a radix 8 division. (three bits are processed each clock cycle).

Floating Point Instruction Set

FABS – Absolute Value

Description:

This instruction takes the absolute value of a double precision floating point number contained in a general purpose register. The sign bit of the number is cleared. The precision of the number is not affected and number is not rounded.

Instruction Format:

31 28	27 22	21 16	15 8	7 0	
5_4	Rt_6	Ra_6	77_8	Pn_4	Pc_4

Clock Cycles: 1

Execution Units: All Floating Point

Operation:

Rt = Ra

FABSS – Single Precision Absolute Value

Description:

This instruction takes the absolute value of a single precision floating point number contained in a general purpose register. The sign bit of the number is cleared.

Instruction Format:

31 28	27 22	21 16	15 8	7 0
5_4	Rt_6	Ra_6	79_8	Pn_4	Pc_4

Clock Cycles: 1

Execution Units: All Floating Point

Operation:

Rt = Ra

FADD – Floating point addition

Description:

Add two double precision floating point numbers in registers Ra and Rb and place the result into target register Rt.

Instruction Format:

39 34	33 28	27 22	21 16	15 8	7 0
8_6	Rt_6	Rb_6	Ra_6	$78h_8$	Pn_4 Pc_4

Clock Cycles: 4

Execution Units: All Floating Point

FADDS – Floating Point Single Precision addition

Description:

Add two single precision floating point numbers in registers Ra and Rb and place the result into target register Rt.

Instruction Format:

39 34	33 28	27 22	21 16	15 8	7 0
18_6	Rt_6	Rb_6	Ra_6	$78h_8$	Pn_4	Pc_4

Clock Cycles: 4

Execution Units: All Floating Point

FCMP - Float Compare

Description:

The register compare instruction compares two registers as floating point doubles and sets the flags in the target predict register as a result. While this is a floating point operation it is executed on the integer ALU.

Instruction Format:

3128	27 22	21 16	15 12	11 8	7	0
2_4	Rb_6	Ra_6	1_4	Pt_4	Pn_4	Pc_4

Clock Cycles: 1

Execution Units: All ALU's

Operation:

```
if Ra < Rb
        P.lt = true
else
        P.lt = false
if mag Ra < mag Rb
        P.ltu = true
else
        P.ltu = false
if Ra = Rb
        P.eq = true
else
        P.eq = false
if unordered
        P.un = true
else
        P.un = false
```

FCMPS - Float Compare Single

Description:

The register compare instruction compares two registers as floating point singles and sets the flags in the target predict register as a result. While this is a floating point operation it is executed on the integer ALU.

Instruction Format:

3128	27 22	21 16	15 12	11 8	7	0
1_4	Rb_6	Ra_6	1_4	Pt_4	Pn_4	Pc_4

Clock Cycles: 1

Execution Units: All ALU's

Operation:

```
if Ra < Rb
        P.lt = true
else
        P.lt = false
if mag Ra < mag Rb
        P.ltu = true
else
        P.ltu = false
if Ra = Rb
        P.eq = true
else
        P.eq = false
if unordered
        P.un = true
else
        P.un = false
```

FDIV – Floating point division

Description:

Divide two double precision floating point numbers in registers Ra and Rb and place the result into target register Rt.

Instruction Format:

39 34	33 28	27 22	21 16	15 8	7 0
Bh_6	Rt_6	Rb_6	Ra_6	$78h_8$	Pn_4 Pc_4

Clock Cycles: 21

Execution Units: All Floating Point

FDIVS – Single Precision Floating point division

Description:

Divide two single precision floating point numbers in registers Ra and Rb and place the result into target register Rt.

Instruction Format:

39 34	33 28	27 22	21 16	15 8	7 0
$1Bh_6$	Rt_6	Rb_6	Ra_6	$78h_8$	Pn_4 Pc_4

Clock Cycles: 12

Execution Units: All Floating Point

FCX – Clear Floating Point Exceptions

Description:

This instruction clears floating point exceptions. The Exceptions to clear are identified as the bits set in the union of register Ra and an immediate field in the instruction. Either the immediate or Ra should be zero.

Instruction Format:

31 28	27 22	21 16	15 8	7 0
D_4	$Immed_6$	Ra_6	79_8	Pn_4	Pc_4

Execution Units: All Floating Point

Operation:

Exceptions:

Bit	Exception Enabled
0	global invalid operation clears the following: - division of infinities - zero divided by zero - subtraction of infinities - infinity times zero - NaN comparison - division by zero
1	overflow
2	underflow
3	divide by zero
4	inexact operation
5	summary exception

FDX – Disable Floating Point Exceptions

Description:

This instruction disables floating point exceptions. The Exceptions disabled are identified as the bits set in the union of register Ra and an immediate field in the instruction. Either the immediate or Ra should be zero.

Instruction Format:

31 28	27 22	21 16	15 8	7 0
F_4	$Immed_6$	Ra_6	79_8	Pn_4	Pc_4

Execution Units: All Floating Point

Operation:

Exceptions:

Bit	Exception Disabled
0	invalid operation
1	overflow
2	underflow
3	divide by zero
4	inexact operation
5	reserved

FEX – Enable Floating Point Exceptions

Description:

This instruction enables floating point exceptions. The Exceptions enabled are identified as the bits set in the union of register Ra and an immediate field in the instruction. Either the immediate or Ra should be zero.

Instruction Format:

31 28	27 22	21 16	15 8	7 0
E_4	$Immed_6$	Ra_6	79_8	Pn_4	Pc_4

Execution Units: All Floating Point

Operation:

Exceptions:

Bit	Exception Enabled
0	invalid operation
1	overflow
2	underflow
3	divide by zero
4	inexact operation
5	reserved

FTX – Trigger Floating Point Exceptions

Description:

This instruction triggers floating point exceptions. The Exceptions to trigger are identified as the bits set in the union of register Ra and an immediate field in the instruction. Either the immediate or Ra should be zero.

Instruction Format:

31 28	27 22	21 16	15 8	7 0
C_4	$Immed_6$	Ra_6	79_8	Pn_4 Pc_4

Execution Units: All Floating Point

Operation:

Exceptions:

Bit	Exception Enabled
0	global invalid operation
1	overflow
2	underflow
3	divide by zero
4	inexact operation
5	reserved

FMAC – Floating Point Multiply Accumulate (planned)

Description:

Multiply two floating point numbers in registers Ra and Rb add a third number from register Rc and place the result into target register Rt.

Instruction Format:

47 45	44 40	39 34	33 28	27 22	21 16	15 8	7 0	
~$_3$	O_5	Rt_6	Rc_6	Rb_6	Ra_6	$76h_8$	Pn_4	Pc_4

Clock Cycles: 8

Execution Units: All Floating Point

O_5	Precision	Mnemonic	Operation	
8	S	FMAC.S	Rt = (Ra * Rb) + Rc	multiply accumulate
9	S	FMAS.S	Rt = (Ra * Rb) - Rc	multiply subtract
10	S	FNMAC.S	Rt = -((Ra * Rb) + Rc)	negate multiply accumulate
11	S	FNMAS.S	Rt = -((Ra * Rb) - Rc)	negate multiply subtract
16	D	FMAC	Rt = (Ra * Rb) + Rc	multiply accumulate
17	D	FMAS	Rt = (Ra * Rb) - Rc	multiply subtract
18	D	FNMAC	Rt = -((Ra * Rb) + Rc)	negate multiply accumulate
19	D	FNMAS	Rt = -((Ra * Rb) - Rc)	negate multiply subtract

FMAN – Mantissa of Number

Description:

This instruction provides the mantissa of a double precision floating point number contained in a general purpose register as a 52 bit zero extended result. The hidden bit of the floating point number remains hidden.

Instruction Format:

31 28	27 22	21 16	15 8	7 0
7_4	Rt_6	Ra_6	77_8	Pn_4	Pc_4

Clock Cycles: 1

Execution Units: All Floating Point

Operation:

Rt = Ra

FMANS – Mantissa of Number

Description:

This instruction provides the mantissa of a single precision floating point number contained in a general purpose register as a 23 bit zero extended result. The hidden bit of the floating point number remains hidden.

Instruction Format:

31 28	27 22	21 16	15 8	7 0
7_4	Rt_6	Ra_6	79_8	Pn_4	Pc_4

Clock Cycles: 1

Execution Units: All Floating Point

Operation:

Rt = Ra

FMOV – Move Double Precision

Description:

This instruction moves one general purpose register to another. This instruction is shorter and uses one less register port than using the OR instruction to move between registers. See also the MOV instruction. This instruction currently performs the same operation as the MOV instruction.

Instruction Format:

31 28	27 22	21 16	15 8	7 0
0_4	Rt_6	Ra_6	77_8	Pn_4	Pc_4

Clock Cycles: 1

Execution Units: All Floating Point

Operation:

Rt = Ra

FMOVS – Move Single Precision

Description:

This instruction moves one general purpose register to another. This instruction is shorter and uses one less register port than using the OR instruction to move between registers. See also the MOV instruction. This instruction currently performs the same operation as the MOV instruction.

Instruction Format:

31 28	27 22	21 16	15 8	7 0
0_4	Rt_6	Ra_6	79_8	Pn_4	Pc_4

Clock Cycles: 1

Execution Units: All Floating Point

Operation:

Rt = Ra

FMUL – Floating point multiplication

Description:

Multiply two double precision floating point numbers in registers Ra and Rb and place the result into target register Rt.

Instruction Format:

39 34	33 28	27 22	21 16	15 8	7 0
Ah_6	Rt_6	Rb_6	Ra_6	$78h_8$	Pn_4 Pc_4

Clock Cycles: 4

Execution Units: All Floating Point

FMULS – Single Precision Floating point multiplication

Description:

Multiply two single precision floating point numbers in registers Ra and Rb and place the result into target register Rt.

Instruction Format:

39 34	33 28	27 22	21 16	15 8	7 0
$1Ah_6$	Rt_6	Rb_6	Ra_6	$78h_8$	Pn_4 Pc_4

Clock Cycles: 4

Execution Units: All Floating Point

FNEG – Negate Register

Description:

This instruction negates a double precision floating point number contained in a general purpose register. The sign bit of the number is inverted.

Instruction Format:

31 28	27 22	21 16	15 8	7 0
4_4	Rt_6	Ra_6	77_8	Pn_4	Pc_4

Clock Cycles: 1

Execution Units: All Floating Point

Operation:

Rt = Ra

FNEGS – Negate Single Precision

Description:

This instruction negates a single precision floating point number contained in a general purpose register. The sign bit of the number is inverted.

Instruction Format:

31 28	27 22	21 16	15 8	7 0
4_4	Rt_6	Ra_6	79_8	Pn_4	Pc_4

Clock Cycles: 1

Execution Units: All Floating Point

Operation:

Rt = Ra

FRM – Set Floating Point Rounding Mode

Description:

This instruction sets the rounding mode bits in the floating point control register (FPSCR). The rounding mode bits are set to the bitwise 'or' of an immediate field in the instruction and the contents of register Ra. Either Ra or the immediate field should be zero.

Instruction Format:

31 28	27 22	21 16	15 8	7 0
D_4	Imm_6	Ra_6	77_8	Pn_4 Pc_4

Execution Units: All Floating Point

Operation:

FPSCR.RM = Ra | Immediate

FSIGN – Sign of Number

Description:

This instruction provides the sign of a double precision floating point number contained in a general purpose register as a floating point double result. The result is +1.0 if the number is positive, 0.0 if the number is zero, and -1.0 if the number is negative.

Instruction Format:

31 28	27 22	21 16	15 8	7 0
6_4	Rt_6	Ra_6	77_8	Pn_4	Pc_4

Clock Cycles: 1

Execution Units: All Floating Point

Operation:

Rt = Ra

FSIGNS – Single Precision Sign of Number

Description:

This instruction provides the sign of a single precision floating point number contained in a general purpose register as a floating point single result. The result is +1.0 if the number is positive, 0.0 if the number is zero, and -1.0 if the number is negative.

Instruction Format:

31 28	27 22	21 16	15 8	7 0
6_4	Rt_6	Ra_6	79_8	Pn_4	Pc_4

Clock Cycles: 1

Execution Units: All Floating Point

Operation:

Rt = Ra

FSTAT – Get Floating Point Status and Control

Description:

The floating point status and control register may be read using the FSTAT instruction. The format of the FPSCR register is outlined on the next page.

Instruction Format:

31 28	27 22	21 16	15 8	7 0
C_4	Rt_6	$\sim_6$	77_8	Pn_4	Pc_4

Execution Units: All Floating Point

Operation:

Rt = FPSCR

Floating Point Status And Control Register Format:

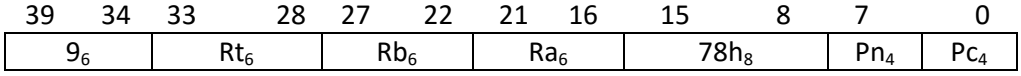
Bit		Symbol	Description
31:29	**RM**	rm	rounding mode (unimplemented)
28	**E5**	inexe	- inexact exception enable
27	**E4**	dbzxe	- divide by zero exception enable
26	**E3**	underxe	- underflow exception enable
25	**E2**	overxe	- overflow exception enable
24	**E1**	invopxe	- invalid operation exception enable
23	**NS**	ns	- non standard floating point indicator
Result Status			
22		fractie	- the last instruction (arithmetic or conversion) rounded intermediate result (or caused a disabled overflow exception)
21	**RA**	rawayz	rounded away from zero (fraction incremented)
20	**SC**	C	denormalized, negative zero, or quiet NaN
19	**SL**	neg <	the result is negative (and not zero)
18	**SG**	pos >	the result is positive (and not zero)
17	**SE**	zero =	the result is zero (negative or positive)
16	**SI**	inf ?	the result is infinite or quiet NaN
Exception Occurrence			
15	**X6**	swt	{reserved} - set this bit using software to trigger an invalid operation
14	**X5**	inerx	- inexact result exception occurred (sticky)
13	**X4**	dbzx	- divide by zero exception occurred
12	**X3**	underx	- underflow exception occurred
11	**X2**	overx	- overflow exception occurred
10	**X1**	giopx	- global invalid operation exception – set if any invalid operation exception has occurred
9	**GX**	gx	- global exception indicator – set if any enabled exception has happened
8	**SX**	sumx	- summary exception – set if any exception could occur if it was enabled - can only be cleared by software
Exception Type Resolution			
7	**X1T**	cvt	- attempt to convert NaN or too large to integer
6	**X1T**	sqrtx	- square root of non-zero negative
5	**X1T**	NaNCmp	- comparison of NaN not using unordered comparison instructions
4	**X1T**	infzero	- multiply infinity by zero
3	**X1T**	zerozero	- division of zero by zero
2	**X1T**	infdiv	- division of infinities
1	**X1T**	subinfx	- subtraction of infinities
0	**X1T**	snanx	- signaling NaN

Greyed out items are not implemented.

FSUB – Floating point subtraction

Description:

Instruction Format:



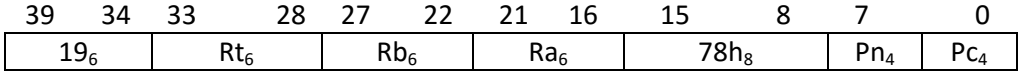
Clock Cycles: 4

Execution Units: All Floating Point

FSUBS – Single Precision Floating point subtraction

Description:

Instruction Format:



Clock Cycles: 4

Execution Units: All Floating Point

FTOI – Float to Integer

Description:

This instruction converts a floating point double value to an integer value.

Instruction Format:

31 28	27 22	21 16	15 8	7 0
2_4	Rt_6	Ra_6	$77h_8$	Pn_4 Pc_4

Clock Cycles: 2

Execution Units: All Floating Point

FTOIS – Single Precision Float to Integer

Description:

This instruction converts a floating point single value to an integer value.

Instruction Format:

31 28	27 22	21 16	15 8	7 0
2_4	Rt_6	Ra_6	$79h_8$	Pn_4 Pc_4

Clock Cycles: 2

Execution Units: All Floating Point

FTST – Float Register Test Compare

Description:

The register test compare compares floating point double in a register against the value zero and sets the predicate flags appropriately. This instruction is executed on the integer ALU.

Instruction Format:

2322	21 16	15 12	11 8	7	0
2_2	Ra_6	0_4	Pt_4	Pn_4	Pc_4

Clock Cycles: 1

Execution Units: All ALU's

Operation:

```
if Ra < 0
        Pt.lt = 1
else
        Pt.lt = 0
if Ra = 0
        Pt.eq = 1
else
        Pt.eq = 0
if unordered
        Pt.un = 1
else
        Pt.un = 0
Pt.ltu = 0
```

Exceptions: none

FTSTS – Float Single Test Compare

Description:

The register test compare compares floating point single in a register against the value zero and sets the predicate flags appropriately. This instruction is executed on the integer ALU.

Instruction Format:

2322	21 16	15 12	11 8	7	0
1_2	Ra_6	0_4	Pt_4	Pn_4	Pc_4

Clock Cycles: 1

Execution Units: All ALU's

Operation:

```
if Ra < 0
        Pt.lt = 1
else
        Pt.lt = 0
if Ra = 0
        Pt.eq = 1
else
        Pt.eq = 0
if unordered
        Pt.un = 1
else
        Pt.un = 0
Pt.ltu = 0
```

Exceptions: none

ITOF – Integer to Float

Description:

This instruction converts an integer value to a double precision floating point representation.

Instruction Format:

31 28	27 22	21 16	15 8	7 0
3_4	Rt_6	Ra_6	$77h_8$	Pn_4	Pc_4

Clock Cycles: 2

Execution Units: All Floating Point

ITOFS – Integer to Float Single

Description:

This instruction converts an integer value to a single precision floating point representation.

Instruction Format:

31 28	27 22	21 16	15 8	7 0
3_4	Rt_6	Ra_6	$79h_8$	Pn_4 Pc_4

Clock Cycles: 2

Execution Units: All Floating Point