Extended Instruction Set

The advanced instruction set provides instructions that perform more complex arithmetic or on larger, memory-based operands referenced by registers. New types introduced in this instruction set are vector tuples of 2 and 3 floating point elements that are always referenced by address.

Extended instructions are composed of at least two 16-bit words. The first combines the identifying prefix and the opcode. The second encodes up to four operands.

OPERATION

Summary description of the operation.

generalised syntactical form

Detailed description of the operation.

List of types or variants supported.

Type	**Prefix**	**Opcode**
First type	ADV	Enumerated Advanced Opcode
Second type	ADV	Enumerated Advanced Opcode
...	...	...

Type	**Operand Word**	
Applicable type		

Additional notes, where relevant.

ACOS

Arc cosine funcion.

acos.type rS, rD

The arc cosine of the value in the source register is computed and stored in the destination register.

32 and 64-bit floating point types are supported.

Type	Prefix	Opcode
f32	ADV	ACOS_F32
f64	ADV	ACOS_F64

Type	Operand Word			
all	0x0	0x0	S	D

Input values are assumed to be in radians.

Inline C Macros

    _acos_f32(src reg, dst reg)
    _acos_f64(src reg, dst reg)

ADD_V2

Add 2-component vector.

add_vec2.type rA, rB, rD
add_cmpl.type rA, rB, rD

The vector sum of the 2-component vectors referenced by source register A and source register B is computed and the result stored in the 2-component vector referenced by the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	ADD_V2F32
f64	ADV	ADD_V2F64

Type	**Operand Word**			
all	0x0	0xA	B	D

The same opcode also handles complex number addition.

Inline C Macros

```
_add_v2f32(src reg A, src reg B, dst reg)
_add_v2f64(src reg A, src reg B, dst reg)
_add_c2f32(src reg A, src reg B, dst reg)
_add_c2f64(src reg A, src reg B, dst reg)
```

ADD_V3

Add 3-component vector.

add_vec3..type rA, rB, rD

The vector sum of the 3-component vectors referenced by source register A and source register B is computed and the result stored in the 3-component vector referenced by the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	ADD_V3F32
f64	ADV	ADD_V3F64

Type	**Operand Word**			
all	0x0	0xA	B	D

Notes.

Inline C Macros

_add_v3f32(src reg A, src reg B, dst reg)
_add_v3f64(src reg A, src reg B, dst reg)

ASIN

Arc sine function.

asin.type rS, rD

The arc sine of the value in the source register is computed and stored in the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	ASIN_F32
f64	ADV	ASIN_F64

Type	**Operand Word**			
all	0x0	0x0	S	D

Input values are assumed to be in radians.

Inline C Macros

_asin_f32(src reg, dst reg)
_asin_f64(src reg, dst reg)

ATAN

Arc tangent function.

atan.type rS, rD

The arc tangent of the value in the source register is computed and stored in the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	ATAN_F32
f64	ADV	ATAN_F64

Type	**Operand Word**			
all	0x0	0x0	S	D

Input values are assumed to be in radians.

Inline C Macros

_atan_f32(src reg, dst reg)
_atan_f64(src reg, dst reg)

CEIL

Ceiling function.

ceil.type rS, rD

The ceiling of the value in the source register is computed and stored in the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	CEIL_F32
f64	ADV	CEIL_F64

Type	**Operand Word**			
all	0x0	0x0	S	D

The resulting integer is represented by the same floating point format implied by the type.

Inline C Macros

```
_ceil_f32(src reg, dst reg)
_ceil_f64(src reg, dst reg)
```

COPY_V2

Clone a 2-component vector.

copy_vec2.type rS, rD
copy_cmpl.type rS, rD

The contents of the 2-component vector referenced by the source register are copied to the 2-component vector referenced by the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	COPY_V2F32
f64	ADV	COPY_V2F64

Type	**Operand Word**			
all	0x0	0x0	S	D

The same opcode also handled the copying of complex number types.

Inline C Macros

_copy_v2f32(src reg, dst reg)
_copy_v2f64(src reg, dst reg)
_copy_c2f32(src reg, dst reg)
_copy_c2f64(src reg, dst reg)

COPY_V3

Clone a 3-component vector.

copy_vec3.type rS, rD

The contents of the 3-component vector referenced by the source register are copied to the 3-component vector referenced by the destination register.

32 and 64-bit floating point types are supported.

Type	Prefix	Opcode
f32	ADV	COPY_V3F32
f64	ADV	COPY_V3F64

Type	Operand Word			
all	0x0	0x0	S	D

Notes.

Inline C Macros

_copy_v3f32(src reg, dst reg)
_copy_v3f64(src reg, dst reg)

COS

Cosine function.

cos.type rS, rD

The cosine of the value in the source register is computed and stored in the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	COS_F32
f64	ADV	COS_F64

Type	**Operand Word**			
all	0x0	0x0	S	D

Input values are assumed to be in radians.

Inline C Macros

_cos_f32(src reg, dst reg)
_cos_f64(src reg, dst reg)

COSH

Hyperbolic cosine function.

cosh.type rS, rD

The hyperbolic cosine of the value in the source register is computed and stored in the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	COSH_F32
f64	ADV	COSH_F64

Type	**Operand Word**			
all	0x0	0x0	S	D

Input values are assumed to be in radians.

Inline C Macros

_cosh_f32(src reg, dst reg)
_cosh_f64(src reg, dst reg)

CROSS_V3

Cross product for 3-component vector.

cross_vec3.type rA, rB, rD

The vector cross product of the 3-component vectors referenced by source register A and source register B is computed and the result stored in the 3-component vector referenced by the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	CROSS_V3F32
f64	ADV	CROSS_V3F64

Type	**Operand Word**			
all	0x0	0xA	B	D

Notes.

Inline C Macros

_cross_v3f32(src reg A, src reg B, dst reg)
_cross_v3f64(src reg A, src reg B, dst reg)

DIV_C2

Complex division.

div_cmpl.type rA, rB, rD

The complex number referenced by source register rA is divided by the complex number referenced by register B and the result is stored in the complex number referenced by the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	ADD_V2F32
f64	ADV	ADD_V2F64

Type	**Operand Word**			
all	0x0	A	B	D

There is no 2-component vector division.

Inline C Macros

_div_c2f32(src reg A, src reg B, dst reg)
_div_c2f64(src reg A, src reg B, dst reg)

DOT_V2

Dot product for 2-component vector.

dot_vec2.type rA, rB, rD

The scalar dot product of the 2-component vectors referenced by source register A and source register B is computed and the result stored in the the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	DOT_V2F32
f64	ADV	DOT_V2F64

Type	**Operand Word**			
all	0x0	0xA	B	D

Notes.

Inline C Macros

_dot_v2f32(src reg A, src reg B, dst reg)
_dot_v2f64(src reg A, src reg B, dst reg)

DOT_V3

Dot product for 3-component vector.

dot_vec3.type rA, rB, rD

The scalar dot product of the 3-component vectors referenced by source register A and source register B is computed and the result stored in the the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	DOT_V3F32
f64	ADV	DOT_V3F64

Type	**Operand Word**			
all	0x0	0xA	B	D

Notes.

Inline C Macros

_dot_v3f32(src reg A, src reg B, dst reg)
_dot_v3f64(src reg A, src reg B, dst reg)

EXP

Exponentiation function.

ceil.type rS, rD

The natural exponent of the value in the source register is computed and stored in the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	EXP_F32
f64	ADV	EXP_F64

Type	**Operand Word**			
all	0x0	0x0	S	D

Notes

Inline C Macros

_exp_f32(src reg, dst reg)
_exp_f64(src reg, dst reg)

FLOOR

Floor function.

ceil.type rS, rD

The floor of the value in the source register is computed and stored in the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	CEIL_F32
f64	ADV	CEIL_F64

Type	**Operand Word**			
all	0x0	0x0	S	D

The resulting integer is represented by the same floating point format implied by the type.

Inline C Macros

_floor_f32(src reg, dst reg)
_floor_f64(src reg, dst reg)

ISQRT

Inverse square root function.

isqrt.type rS, rD

The inverse square root of the value in the source register is computed and stored in the destination register.

32 and 64-bit floating point types are supported.

Type	Prefix	Opcode
f32	ADV	ISQRT_F32
f64	ADV	ISQRT_F64

Type	Operand Word			
all	0x0	0x0	S	D

Notes.

Inline C Macros

_isqrt_f32(src reg, dst reg)
_isqrt_f64(src reg, dst reg)

LD_CONST

Load predefined constant.

ld_const.type enum, rD

The enumerated constant indicated is stored in the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	LD_CONST_F32
f64	ADV	LD_CONST_F64

Type	**Operand Word**			
all	0x0	0x0	C	D

Enum	**C**	**Description**	**ROM Value**
PI	0	π Ratio of circle circumference to diameter	3.141592653589793
2PI	1	2 * π	6.283185307179586
HALF_PI	2	π / 2	1.570796326794897
INV_PI	3	1 / π	0.318309886183791
PHI	4	ϕ Golden ratio	1.618033988749895
E	5	*e* Euler’s number	2.718281828459045
SQRT_2	6	√2 Pythagoras’ constant	1.414213562373095
INV_SQRT_2	7	1 / √2	0.707106781186548
SQRT_3	8	√3 Theodorus’ constant	1.732050807568877
SQRT_5	9	√5	2.236067977499790
LN_2	10	ln(2)	0.693147180559945
INV_LN_2	11	1 / ln (2)	1.442695040888963
UNIPB	12	P_2 Universal Parabolic Constant	2.295587149392638

Note that the ROM value for the constant is expected to be stored at at least the precision indicated above, even if the 32-bit version of the operation is used.

Inline C Macros

_ld_const_f32(enum, dst reg)
_ld_const_f64(enum, dst reg)

LD_RII

Load register indirect with index.

ld.type (rB, rI, #S), rD

The value at the effective address computed by taking the base register B, offset by the 32-bit signed index value in register I, scaled by shifting with integer value S, is loaded into the destination register.

8, 16, 32 and 64-bit sizes are supported.

Size	**Prefix**	**Opcode**
8	ADV	LD_RII_8
16	ADV	LD_RII_16
32	ADV	LD_RII_32
64	ADV	LD_RII_64

Type	**Operand Word**			
all	#S	I	B	D

Scale value is applied as a shift, meaning the largest scale factor possible is 32768. The operand size is not factored into the scaling, such that ld.8 (rB, rI, 0) and ld.64 (rB, rI, 0) will result in identical effective addresses.

Inline C Macros

_ld_rii_8(base reg B, index reg I, scale, dst reg)
_ld_rii_16(base reg B, index reg I, scale, dst reg)
_ld_rii_32(base reg B, index reg I, scale, dst reg)
_ld_rii_64(base reg B, index reg I, scale, dst reg)

LERP

Linearly interpolate between two values.

lerp.type rA, rB, rL, rD

The value calculated by taking the linear interpolation between the values stored in rA and rB using the normalised interpolation factor in rL is stored in the destination register.

32 and 64-bit floating point types are supported.

Type	Prefix	Opcode
f32	ADV	LERP_F32
f64	ADV	LERP_F64

Type	Operand Word			
all	A	B	L	D

The use of interpolation values outside the rage 0.0 to 1.0 are not trapped as erroneous.

Inline C Macros

_lerp_f32(src reg A, src reg B, src reg L, dst reg)
_lerp_f64(src reg A, src reg B, src reg L, dst reg)

LERP_V2

Linearly interpolate between two 2-component vectors.

lerp_vec2.type rA, rB, rL, rD

The value calculated by taking the linear interpolation between the 2-component vectors referenced by rA and rB using the normalised interpolation factor in rL is stored in the 2-component vector referenced by the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	LERP_V2F32
f64	ADV	LERP_V2F64

Type	**Operand Word**			
all	A	B	L	D

The use of interpolation values outside the rage 0.0 to 1.0 are not trapped as erroneous.

Inline C Macros

```
_lerp_v2f32(src reg A, src reg B, src reg L, dst reg)
_lerp_v2f64(src reg A, src reg B, src reg L, dst reg)
```

LERP_V3

Linearly interpolate between two 3-component vectors.

lerp_vec3.type rA, rB, rL, rD

The value calculated by taking the linear interpolation between the 3-component vectors referenced by rA and rB using the normalised interpolation factor in rL is stored in the 3-component vector referenced by the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	LERP_V3F32
f64	ADV	LERP_V3F64

Type	**Operand Word**			
all	A	B	L	D

The use of interpolation values outside the rage 0.0 to 1.0 are not trapped as erroneous.

Inline C Macros

_lerp_v3f32(src reg A, src reg B, src reg L, dst reg)
_lerp_v3f64(src reg A, src reg B, src reg L, dst reg)

LOG10

Base-10 logarithm function.

log10.type rS, rD

The base-10 logarithm of the value in the source register is computed and stored in the destination register.

32 and 64-bit floating point types are supported.

Type	Prefix	Opcode
f32	ADV	LOG10_F32
f64	ADV	LOG10_F64

Type	Operand Word			
all	0x0	0x0	S	D

Notes

Inline C Macros

    _log10_f32(src reg, dst reg)
    _log10_f64(src reg, dst reg)

LOG2

Binary logarithm.

log2.type rS, rD

The base-2 logarithm of the value in the source register is computed and stored in the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	LOG2_F32
f64	ADV	LOG2_F64

Type	**Operand Word**			
all	0x0	0x0	S	D

Notes

Inline C Macros

_log2_f32(src reg, dst reg)
_log2_f64(src reg, dst reg)

LOGN

Natural logarithm function.

logn.type rS, rD

The natural logarithm of the value in the source register is computed and stored in the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	LOGN_F32
f64	ADV	LOGN_F64

Type	**Operand Word**			
all	0x0	0x0	S	D

Notes

Inline C Macros

_logn_f32(src reg, dst reg)
_logn_f64(src reg, dst reg)

LOGX

Base-X logarithm function.

logn.type rS, rX, rD

The base-x logarithm of the value in the source register is computed and stored in the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	LOGX_F32
f64	ADV	LOGX_F64

Type	**Operand Word**			
all	0x0	X	S	D

This may be implemented internally as the natural logarithm of the source value divided by the natural logarithm of the base on systems that do not have a dedicated function.

Inline C Macros

_logx_f32(src reg, base x reg, dst reg)
_logx_f64(src reg, base x reg, dst reg)

LSLM

Logical shift left with mask.

lslm.size rS, rM, rC, rD

The bits within the masked region of the source register are shifted left by the value in the count register, recombined with the unmasked bits of the original value and the result stored in the destination register.

32 and 64-bit sizes are supported.

Size	**Prefix**	**Opcode**
32	ADV	LSLM_32
64	ADV	LSLM_64

Size	**Operand Word**			
all	S	M	C	D

Notes

Inline C Macros

_lslm_32(src reg, mask reg, count reg, dst reg)
_lslm_64(src reg, mask reg, count reg, dst reg)

LSLR

Logical shift right with mask.

lslr.size rS, rM, rC, rD

The bits within the masked region of the source register are shifted right by the value in the count register, recombined with the unmasked bits of the original value and the result stored in the destination register.

32 and 64-bit sizes are supported.

Size	**Prefix**	**Opcode**
32	ADV	LSLM_32
64	ADV	LSLM_64

Size	**Operand Word**			
all	S	M	C	D

Notes

Inline C Macros

_lslr_32(src reg, mask reg, count reg, dst reg)
_lslr_64(src reg, mask reg, count reg, dst reg)

MADD

Multiply add.

madd.type rA, rB, rC, rD

The value in the register rA is added to the product of the values stored in source registers rB and rC and the result stored in the destination register.

32 and 64-bit floating point types are supported.

Type	Prefix	Opcode
f32	ADV	MADD_F32
f64	ADV	MADD_F64

Type	Operand Word			
all	A	B	C	D

Note that the operation as described is actually D = (C * B) + A, reflecting the order in which the operand parameters are extracted from the operand word.

Inline C Macros

_madd_f32(add reg, mul reg 1, mul reg 2, dst reg)
_madd_f64(add reg, mul reg 1, mul reg 2, dst reg)

MAGN_V2

Scalar magnitude of 2-component vector.

magn_vec2.type rS, rD

The scalar magnitude of the 2-component vector referenced by the source register is calculated and stored in the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	MAGN_V2F32
f64	ADV	MAGN_V2F64

Type	**Operand Word**			
all	0x0	0x0	S	D

Notes.

Inline C Macros

_magn_v2f32(src reg, dst reg)
_magn_v2f64(src reg, dst reg)

MAGN_V3

Scalar magnitude of 3-component vector.

magn_vec3.type rS, rD

The scalar magnitude of the 3-component vector referenced by the source register is calculated and stored in the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	MAGN_V3F32
f64	ADV	MAGN_V3F64

Type	**Operand Word**			
all	0x0	0x0	S	D

Notes.

Inline C Macros

_magn_v3f32(src reg, dst reg)
_magn_v3f64(src reg, dst reg)

MUL_C2

Complex number multiplication.

mul_cmpl.type rA, rB, rD

The complex number referenced by source register rA is multiplied by the complex number referenced by register B and the result is stored in the complex number referenced by the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	MUL_C2F32
f64	ADV	MUL_C2F64

Type	**Operand Word**			
all	0x0	A	B	D

Notes.

Inline C Macros

_mul_c2f32(src reg A, src reg B,, dst reg)
_mul_c2f64(src reg A, src reg B, dst reg)

NORM_V2

Normalise a 2-component vector.

norm_vec2.type rS, rD

The scalar magnitude of the 2-component vector referenced by the source register is calculated, divided into the component elements and the resulting values written into the 2-component vector referenced by the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	NORM_V2F32
f64	ADV	NORM_V2F64

Type	**Operand Word**			
all	0x0	0x0	S	D

For performance reasons, implementations will generally compute the inverse of the magnitude and use member multiplication rather than actual division.

Inline C Macros

_norm_v2f32(src reg, dst reg)
_norm_v2f64(src reg, dst reg)

NORM_V3

Normalise a 3-component vector.

norm_vec3.type rS, rD

The scalar magnitude of the 3-component vector referenced by the source register is calculated, divided into the component elements and the resulting values written into the 3-component vector referenced by the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	NORM_V3F32
f64	ADV	NORM_V3F64

Type	**Operand Word**			
all	0x0	0x0	S	D

Notes.

Inline C Macros

_norm_v3f32(src reg, dst reg)
_norm_v3f64(src reg, dst reg)

POW

Power function.

pow.type rS, rP, rD

The value in the source register is raised to the power stored in the power register and the result stored in the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	LOGX_F32
f64	ADV	LOGX_F64

Type	**Operand Word**			
all	0x0	P	S	D

Notes.

Inline C Macros

_pow_f32(src reg, power reg, dst reg)
_pow_f64(src reg, power reg, dst reg)

SCALE_V2

Multiply a 2-component vector by a scalar.

scale_vec2.type rS, rF, rD

The members of the 2-component vector referenced by the source register are multiplied by the scale factor register and the result stored in the 2-component vector referenced by the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	SCALE_V2F32
f64	ADV	SCALE_V2F64

Type	**Operand Word**			
all	0x0	F	S	D

For performance reasons, implementations will generally compute the inverse of the magnitude and use member multiplication rather than actual division.

Inline C Macros

_scale_v2f32(src reg, factor reg, dst reg)
_scale_v2f64(src reg, factor reg, dst reg)

SCALE_V3

Multiply a 3-component vector by a scalar.

scale_vec3.type rS, rF, rD

The members of the 3-component vector referenced by the source register are multiplied by the scale factor register and the result stored in the 2-component vector referenced by the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	SCALE_V3F32
f64	ADV	SCALE_V3F64

Type	**Operand Word**			
all	0x0	F	S	D

For performance reasons, implementations will generally compute the inverse of the magnitude and use member multiplication rather than actual division.

Inline C Macros

_scale_v3f32(src reg, factor reg, dst reg)
_scale_v3f64(src reg, factor reg, dst reg)

SINCOS

Simultaneous sine and cosine.

sincos.type rX, rS, rC

The sine and cosine of the value in the source register X are computed and saved into the destination sine register S and destination cosine register C.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	LOGX_F32
f64	ADV	LOGX_F64

Type	**Operand Word**			
all	0x0	X	C	S

This instruction has two destination operands.

Inline C Macros

_sincos_f32(src reg X, dst reg C, dst reg S)
_sincos_f64(src reg X, dst reg C, dst reg S)

SIN

Sine function.

sin.type rS, rD

The sine of the value in the source register is computed and stored in the destination register.

32 and 64-bit floating point types are supported.

Type	Prefix	Opcode
f32	ADV	SIN_F32
f64	ADV	SIN_F64

Type	Operand Word			
all	0x0	0x0	S	D

Input values are assumed to be in radians.

Inline C Macros

_sin_f32(src reg, dst reg)
_sin_f64(src reg, dst reg)

SINH

Hyperbolic sine function.

sinh.type rS, rD

The hyperbolic sine of the value in the source register is computed and stored in the destination register.

32 and 64-bit floating point types are supported.

Type	Prefix	Opcode
f32	ADV	SINH_F32
f64	ADV	SINH_F64

Type	Operand Word			
all	0x0	0x0	S	D

Input values are assumed to be in radians.

Inline C Macros

_sinh_f32(src reg, dst reg)
_sinh_f64(src reg, dst reg)

SPLAT_V2

Fill a 2-component vector.

splat_vec2.type rS, rD

The scalar value in the source register is written to each member of the 2-component vector referenced by the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	SPLAT_V2F32
f64	ADV	SPLAT_V2F64

Type	**Operand Word**			
all	0x0	0x0	S	D

Notes.

Inline C Macros

_splat_v2f32(src reg, dst reg)
_splat_v2f64(src reg, dst reg)

SPLAT_V3

Fill a 3-component vector.

splat_vec3.type rS, rD

The scalar value in the source register is written to each member of the 3-component vector referenced by the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	SPLAT_V3F32
f64	ADV	SPLAT_V3F64

Type	**Operand Word**			
all	0x0	0x0	S	D

Notes.

Inline C Macros

_splat_v3f32(src reg, dst reg)
_splat_v3f64(src reg, dst reg)

SQRT

Square root function.

isqrt.type rS, rD

The square root of the value in the source register is computed and stored in the destination register.

32 and 64-bit floating point types are supported.

Type	Prefix	Opcode
f32	ADV	SQRT_F32
f64	ADV	SQRT_F64

Type	Operand Word			
all	0x0	0x0	S	D

Notes.

Inline C Macros

_sqrt_f32(src reg, dst reg)
_sqrt_f64(src reg, dst reg)

ST_RII

Store register indirect with index.

st.type rS, (rB, rI, #S)

The value in the source register is stored at the effective address computed by taking the base register B, offset by the 32-bit signed index value in register I, scaled by shifting with integer value S.

8, 16, 32 and 64-bit sizes are supported.

Size	**Prefix**	**Opcode**
8	ADV	ST_RII_8
16	ADV	ST_RII_16
32	ADV	ST_RII_32
64	ADV	ST_RII_64

Type	**Operand Word**			
all	#S	I	B	D

Scale value is applied as a shift, meaning the largest scale factor possible is 32768. The operand size is not factored into the scaling, such that st.8 (rB, rI, 0) and st.64 (rB, rI, 0) will result in identical effective addresses.

Inline C Macros

_st_rii_8(src reg, base reg B, index reg I, scale)
_st_rii_16(src reg, base reg B, index reg I, scale)
_st_rii_32(src reg, base reg B, index reg I, scale)
_st_rii_64(src reg, base reg B, index reg I, scale)

SUB_V2

Subtract 2-component vector.

sub_vec2.type rA, rB, rD
sub_cmpl.type rA, rB, rD

The vector difference of the 2-component vectors referenced by source register A and source register B is computed and the result stored in the 2-component vector referenced by the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	SUB_V2F32
f64	ADV	SUB_V2F64

Type	**Operand Word**			
all	0x0	0xA	B	D

The same opcode also handles complex number addition.

Inline C Macros

_sub_v2f32(src reg A, src reg B, dst reg)
_sub_v2f64(src reg A, src reg B, dst reg)
_sub_c2f32(src reg A, src reg B, dst reg)
_sub_c2f64(src reg A, src reg B, dst reg)

SUB_V3

Subtract 3-component vector.

sub_vec3.type rA, rB, rD

The vector difference of the 3-component vectors referenced by source register A and source register B is computed and the result stored in the 2-component vector referenced by the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	SUB_V3F32
f64	ADV	SUB_V3F64

Type	**Operand Word**			
all	0x0	0xA	B	D

Notes.

Inline C Macros

_sub_v2f32(src reg A, src reg B, dst reg)
_sub_v2f64(src reg A, src reg B, dst reg)

TAN

Tangent function.

tan.type rS, rD

The tangent of the value in the source register is computed and stored in the destination register.

32 and 64-bit floating point types are supported.

Type	**Prefix**	**Opcode**
f32	ADV	TAN_F32
f64	ADV	TAN_F64

Type	**Operand Word**			
all	0x0	0x0	S	D

Input values are assumed to be in radians.

Inline C Macros

_tan_f32(src reg, dst reg)
_tan_f64(src reg, dst reg)

TANH

Hyperbolic tangent function.

tanh.type rS, rD

The hyperbolic tangent of the value in the source register is computed and stored in the destination register.

32 and 64-bit floating point types are supported.

Type	Prefix	Opcode
f32	ADV	TANH_F32
f64	ADV	TANH_F64

Type	Operand Word			
all	0x0	0x0	S	D

Input values are assumed to be in radians.

Inline C Macros

_tanh_f32(src reg, dst reg)
_tanh_f64(src reg, dst reg)

TODO

_xform_v2f32()
_xform_v2f64()
_xform_v3f32()
_xform_v3f64()