

# IEEE 754-2008 Instruction Set Architecture ("ISA")

Designed for implementation in Xilinx Kintex UltraScale and UltraScale+ brand FPGAs

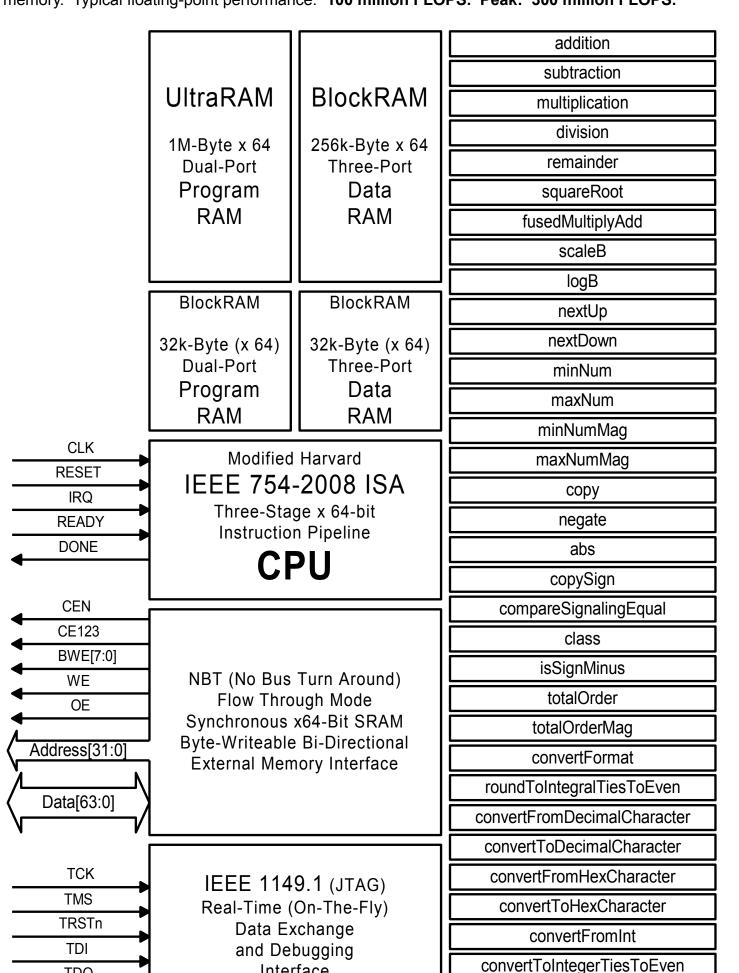
per operation. Much more efficient than conventional "load-store" models, it can push two 64-bit operands into an operator, whether it be floating point, integer or logical, every clock cycle, with results automatically spilling into one of sixteen memory-mapped result buffers dedicated to that operator.

The SYMPL 64-bit, IEEE 754-2008 ISA CPU is a novel "mover" architecture that efficiently implements in hardware "all" operations mandated by IEEE 754-2008 using one instruction

This instruction-set comprises features not available on conventional "load-store" models. For instance, it includes both direct and indirect addressing modes, the later with either autopost-increment/decrement (by up to 2047 bytes, or fixed displacement/offset by up to 1023 bytes). In addition, it features at least two very efficient hardware loop counters and repeat counters. When used in combination with auto-post increment/decrement indirect addressing mode, the REPEAT instruction is a powerful and efficient means for moving or pushing large chunks of data into any of the core's pipelined operators or block of memory much more efficiently than means available for doing the same thing in conventional load-store models, in that no extra cycles are needed to modify source or destination pointers after each read or write operation.

This ISA CPU supports binary16, binary32 and binary64 floating-point formats to the base range and precision of the installed floating-point operators. Meaning that, since the instruction format includes two "size" bits for each of the SourceA, SourceB and Destination address fields of the instruction, no explicit conversion is necessary for computations involving mixed formats, as results automatically inherit conversion exceptions. For stricter handling, numbers can of course be converted explicitly before submission to the operator.

Single 64-bit CPU implemented in Kintex UltraScale+ XCKU3p Half-precision floating-point with all IEEE 754-2008 operators implemented: 1.16ns WNS LUT 21% BRAM 27% URAM 17%, .86w@100MHz, 256k UltraRAM program memory, 128k blockRam data memory. Typical floating-point performance: 100 million FLOPS. Peak: 300 million FLOPS.



Interface

point performance: 200 million FLOPS. Peak: 600 million FLOPS @100 MHz..

**BlockRAM** 

256k-Byte x 64

Three-Port

Data

RAM

BlockRAM

32k-Byte (x 64)

Three-Port

Data

RAM

Modified Harvard

IEEE 754-2008 ISA

Three-Stage x 64-bit

Instruction Pipeline

CPU

NBT (No Bus Turn Around)

Flow Through Mode

Synchronous x64-Bit SRAM

Byte-Writeable Bi-Directional

External Memory Interface

IEEE 1149.1 (JTAG)

Real-Time (On-The-Fly)

Data Exchange

and Debugging

Interface

Two 64-bit CPUs implemented in Kintex UltraScale+ XCKU3p

addition

subtraction

multiplication

division

remainder

squareRoot

fusedMultiplyAdd

scaleB

logB

nextUp

nextDown

minNum

maxNum

minNumMag

maxNumMag

copy

negate

abs

copySign

compareSignalingEqual

isSianMinus

totalOrderMag

convertFormat

roundToIntegralTiesToEven

convertFromDecimalCharacter

convertToDecimalCharacter

convertFromHexCharacter

convertToHexCharacter

convertToIntegerTiesToEven

convertToIntegerExactTiesToEven

Half-precision floating-point with all IEEE 754-2008 operators implemented: .085ns WNS, LUT 34%; BRAM 58%; URAM

71%; CPU 1M-byte prog mem, 256k data mem; XCUs 64k data mem 64k prog mem. 1.24W @ 100MHz. Typical floating-

TDO

**UltraRAM** 

1M-Byte x 64

Dual-Port

Program

BlockRAM

32k-Byte (x 64)

Program

RAM

CLK

RESET

IRQ

READY

DONE

CEN

CE123

WE

OE

Address[31:0]

Data[63:0]

TCK

TMS

TRSTn

TDI

TDO

TDO

BWE[7:0]

# **All IEEE 754-2008 Floating-Point Operations** Implemented in Hardware

Because all operations mandated by IEEE 754-2008 for conformance are implemented in hardware (rather than software bit-banging algorithms), the SYMPL 64-bit IEEE 754-2008 ISA CPU can outperform other CPUs clocking several times faster, thereby resulting in much lower power consumption for the same amount of work.

# Parent CPU and Child XCU(s) Execute the **Same Instruction Set**

With exception to program memory and data memory size, parent and child CPU cores are functionally identical. At the top level module source code, implementers can specify memory size, number of XCUs, and which floating-point operators to include in the implementation prior to compile

# **On-Chip Real-Time Data Exchange and Debug**

The SYMPL, 64-bit, IEEE-2008 CPU features an extensible IEEE 1149-1 JTAG real-time data exchange and debug interface that provides the ability to not only exchange data with any or all of the specified CPU/XCU(s), but also provide real-time debug capability. For instance, any register or memory location in the CPU or a given XCU can be examined and/or modified in real-time. Hardware breakpoints can be set in the CPU and single-stepped when encountered. Any or all XCUs can be reset, breakpointed and single-stepped individually or simultaneously.

## **Fused Instruction Registers**

The instruction registers of the CPU and any eXtension Compute Units (XCUs) attached to it are fused. This means that the CPU, using special instructions, can command any one, any set, or all "child" XCUs to perform a given operation simultaneously. Such operations are synchronized and instantaneous This means that whenever the parent CPU issues an instruction to one or more child XCUs, the execution of such is immediate,

as if the CPU had executed it itself. **Dual-Port Program Memory** This IEEE 754-2008 ISA takes full advantage of Kintex UltraScale+

UltraRAMS for use as dual-port program memory. By configuring the

UltraRAMs as True-Dual-Port RAMs, the first port is used for accessing

instructions to be executed and the second port can be mapped into the

upper portion of the cores data memory space for accessing large tables.

Because Kintex UltraRAMs cannot be initialized as a ROM, the first 32k bytes of program memory space are implemented using standard blockRAMs so that a micro-kernel can reside and be available at power-up for loading UltraRAM program memory.

# **Three-Port Data Memory**

addition

subtraction

multiplication

division

remainder

squareRoot

fusedMultiplyAdd

convertFormat

roundToIntegralTiesToEven

convertFromDecimalCharacter

convertToDecimalCharacter

convertFromHexCharacter

convertToHexCharacter

scaleB

nextUp

nextDown

convertFromInt

convertToIntegerTiesToEven

convertToIntegerExactTiesToEven

minNum

maxNum

minNumMag

maxNumMag

copy

negate

copySign

compareSignalingEqual

class

isSignMinus

totalOrder

totalOrderMag

totalOrder

This enables movement of dual operands simultaneously into integer, logical and floating-point operators every clock cycle.

**Memory-Mapped** 

Standard blockRAMs are configured for use as three-port data memory.

All internal registers, including PC, Status, Auxilliary Registers, Repeat Counter, hardware Loop Counters, Stack Pointer, etc., are memory-mapped.

All Registers and Operators are

Able to accept two operands every clock cycle, all operators, including integer arithmetic, logical, floating-point arithmetic and conversion are memorymapped and fully pipelined, typically occupying sixteen consecutive double-word locations per operator.

# No Op-Codes

Since all registers and operators are memorymapped, this instruction set does not employ "opcodes", as that term is traditionally understood.

### All Rounding Modes Fully **Supported**

This IEEE 754-2008 floating-point ISA fully supports all directed rounding modes, including nearest, positive, negative, zero and away. Default is nearest, but this can be changed by modifying and enabling the rounding mode attribute bits in the Status Register.

Alternatively, if no attribute is specified and enabled in the Status Register, rounding direction can be specified, on-the-fly, using the two rounding direction bits in the instruction itself.

Three 64-bit CPUs implemented in Kintex UltraScale+ xcku3p Half-precision floating-point with all IEEE 754-2008 operators implemented: .417ns WNS, LUT 47%; BRAM 45%; URAM 21%; CPU 256k prog mem, 128k data mem; XCUs 64k data mem 32k prog mem; 1.488W@100MHz. Typical floating-point performance: 300 million FLOPS. Peak: 900 million FLOPS.

**Modified Harvard** 

**IEEE 754-2008 ISA** 

Three-Stage x 64-bit

Instruction Pipeline

CPU

convertToIntegerExactTiesToEven

**UltraRAM** 

64k-Byte (x 64)

Dual-Port

**Program** 

**BlockRAM** 

64k-Byte (x 64)

Three-Port

Data

**RAM** 

			addition			addition			addition
			subtraction			subtraction			subtraction
	UltraRAM	BlockRAM	multiplication			multiplication			multiplication
	1M-Byte x 64	256k-Byte x 64	division			division			division
	Dual-Port	Three-Port	remainder			remainder			remainder
	Program	Data	squareRoot	UltraRAM	BlockRAM	squareRoot	UltraRAM	BlockRAM	squareRoot
	RAM	RAM	fusedMultiplyAdd	64k-Byte (x 64) Dual-Port	64k-Byte (x 64) Three-Port	fusedMultiplyAdd	64k-Byte (x 64) Dual-Port	64k-Byte (x 64) Three-Port	fusedMultiplyAdd
			scaleB	Program RAM	Data RAM	convertFormat	Program RAM	Data RAM	convertFormat
			logB			roundToIntegralTiesToEven		1.0.1	roundToIntegralTiesToEven
	BlockRAM	BlockRAM	nextUp			convertFromDecimalCharacter			convertFromDecimalCharacter
	32k-Byte (x 64)	32k-Byte (x 64)	nextDown			convertToDecimalCharacter			convertToDecimalCharacter
	Dual-Port	Three-Port	minNum			convertFromHexCharacter			convertFromHexCharacter
	Program	Data	maxNum			convertToHexCharacter			convertToHexCharacter
	RAM	RAM	minNumMag			scaleB			scaleB
CLK RESET	Modified Harvard		maxNumMag			logB			logB
IRQ		-2008 ISA	сору			nextUp			nextUp
READY	Three-Sta		negate		XCU	nextDown		XCI	nextDown
DONE			abs			convertFromInt			convertFromInt
	† CF	70	copySign			convertToIntegerTiesToEven			convertToIntegerTiesToEven
CEN			compareSignalingEqual			convertToIntegerExactTiesToEven			convertToIntegerExactTiesToEven
CE123	4		class			minNum			minNum
BWE[7:0] WE	NBT (No Bus	Turn Around)	isSignMinus	Madifia	i Harvard	maxNum	Madifia	d Harvard	maxNum
OE OE	Flow Thro		totalOrder	IEEE 754	-2008 ISA	minNumMag	IEEE 754	-2008 ISA	minNumMag
	Synchronous		totalOrderMag	Instruction	ge x 64-bit n Pipeline	maxNumMag	Instructio	ge x 64-bit on Pipeline	maxNumMag
Address[31:0]	Byte-Writeable Bi-Directional External Memory Interface		convertFormat	С	PU	сору	C	PU	сору
Data[02:0]		,	roundToIntegralTiesToEven			negate			negate
Data[63:0]	1		convertFromDecimalCharacter			abs			abs
•		]	convertToDecimalCharacter			copySign			copySign
	.===	0.1 (ITAC)	convertFromHexCharacter			compareSignalingEqual			compareSignalingEqual
TCK	<b>Ы</b>   <u> </u>	J.I (JIAG) I							
TCK TMS TRSTn	IEEE 114 Real-Time ( Data Ex	On-The-Fly)	convertToHexCharacter			class			class

## Five 64-bit CPUs implemented in Kintex UltraScale+ xcku3p Half-precision floating-point with all IEEE 754-2008 operators implemented (in CPU): WNS .003ns, LUT 73%, BRAM 63%,

convertToIntegerTiesToEven

convertToIntegerExactTiesToEven

URAM 25%, CPU 256k prog mem, 128k data mem, XCUs 64k data mem, 32k prog mem, 1.87W@100MHz. Typical floatingpoint performance: 500 million FLOPS. Peak: 1.5 billion FLOPS.

			addition		
			subtraction		
	UltraRAM	BlockRAM	multiplication		
	1M-Byte x 64	256k-Byte x 64	division		
	Dual-Port	Three-Port	remainder		
	Program	Data	squareRoot		
	RAM	RAM	fusedMultiplyAdd		
			scaleB	VOLLA	NO.
			logB	XCU 0	XCU1
	BlockRAM	BlockRAM	nextUp		
	32k-Byte (x 64)	32k-Byte (x 64)	nextDown		
	Dual-Port	Three-Port	minNum		
	Program	Data	maxNum		
	RAM	RAM	minNumMag		
CLK RESET		l Harvard	maxNumMag		
IRQ		-2008 ISA	сору		
READY		ge x 64-bit n Pipeline	negate		
DONE			abs		
•	Ci	PU	copySign		
CEN	_		compareSignalingEqual		
CE123	4		class		
BWE[7:0] WE	NRT (No Rus	Turn Around)	isSignMinus		
◆ VE OE	Flow Thro	ough Mode	totalOrder		
		x64-Bit SRAM	totalOrderMag		
Address[31:0]		e Bi-Directional nory Interface	convertFormat	XCU 2	XCU-3
Data[63:0]	J	·	roundToIntegralTiesToEven		
Data[65.0]	1		convertFromDecimalCharacter		
,			convertToDecimalCharacter		
TCK	IFFE 114	·9.1 (JTAG)	convertFromHexCharacter		
TMS	Real-Time (	(On-The-Fly)	convertToHexCharacter		
TRSTn TDI		xchange bugging	convertFromInt		
TDO		rface	convertToIntegerTiesToEven		
<b>←</b>			convertToIntegerExactTiesToEven		
	·				

# FloPoCo-Generated Operators

totalOrder

The base floating-point RTL for multiplication, addition, subtraction, division, squareRoot, fusedMultiplyAdd, remainder, log, exp, pow, powr, pown in the instant design are FloPoCo-generated.

However, to make them conforming to IEEE 754-2008, substantial extra logic was added in the individual wrappers as well as slight modification to the generated RTL to disable rounding internally and adding module ports to them to make the round, guard and sticky bits visible to facilitate directed rounding.

For licensing and other information on the FPL 2017 Community Award-winning FloPoCo generator, visit:

http://flopoco.gforge.inria.fr

#### Nine 64-bit Floating-Point CPUs implemented in Kintex UltraScale+ XCKU5p Half-precision floating-point with all IEEE 754-2008 operators implemented (in CPU): WNS .331ns, LUT 93%, BRAM 87%, URAM 63%, CPU 1M prog mem, 256k data mem, XCUs 64k data mem, 32k prog mem 2.5W@80MHz. Typical floating-point performance: 810 million FLOPS. Peak: 2.43 billion FLOPS.

CLK RESET IRQ READY	IEEE 754 Three-Sta	BlockRAM  256k-Byte x 64 Three-Port Data RAM  BlockRAM  32k-Byte (x 64) Three-Port Data RAM  Harvard -2008 ISA ge x 64-bit	subtraction  multiplication  division  remainder  squareRoot  fusedMultiplyAdd  scaleB  logB  nextUp  nextDown  minNum  maxNum  minNum  maxNum  minNumMag  maxNumMag  copy  negate	XC		XCJ1	<b>)</b>	(CU-2	XCU3	
DONE  CEN CE123 BWE[7:0] WE OE  Address[31:0]  TCK TMS TRSTn TDI TDO	NBT (No Bus Flow Thro Synchronous Byte-Writeable External Men  IEEE 114 Real-Time ( Data Exand De	nory Interface 9.1 (JTAG)	abs copySign compareSignalingEqual class isSignMinus totalOrder totalOrderMag convertFormat roundToIntegralTiesToEven convertFromDecimalCharacter convertFromHexCharacter convertToHexCharacter convertToHexCharacter convertToIntegerTiesToEven convertToIntegerTiesToEven	XC		XCU 5	)	(CU 6	XCU 7	

# Seventeen Floating-Point 64-bit CPUs implemented in Kintex UltraScale+ xcku15p Half-precision floating-point with all IEEE 754-2008 operators implemented (in CPU): Typical floating-point performance: 1.36 billion FLOPS. Peak: 4.1 billion FLOPS @80MHz.

•			•	, , ,			_
				XCU 0	XCU1	XCU 2	XCU 3
CLK RESET IRQ	▶ IEEE 754	BlockRAM  256k-Byte x 64 Three-Port Data RAM  BlockRAM  32k-Byte (x 64) Three-Port Data RAM  d Harvard -2008 ISA	addition subtraction multiplication division remainder squareRoot fusedMultiplyAdd scaleB logB nextUp nextDown minNum maxNum minNum minNumMag maxNumMag copy	XCU 4	XCU 5	XCU 6	XCU 7
READY DONE  CEN CE123 BWE[7:0] WE OE  Address[31:0]  TCK TMS TRSTn TDI TDO	NBT (No Bus Flow Thromagness Flow Throma	age x 64-bit on Pipeline  PU  S Turn Around) ough Mode x64-Bit SRAM e Bi-Directional mory Interface  49.1 (JTAG) (On-The-Fly) xchange ebugging	negate abs copySign compareSignalingEqual class isSignMinus totalOrder totalOrderMag convertFormat roundToIntegralTiesToEven convertFromDecimalCharacter convertFromHexCharacter convertToHexCharacter convertToHexCharacter convertToIntegerTiesToEven convertToIntegerTiesToEven	XCU 8	XCU 9	XCU 10	XCU 11
				XCU 12	XCU-13	XCU 14	XCU 15

#### **Operations Implemented in Hardware** (actual SYMPL IL mnemonics shown):

roundToIntegralTiesToEven

roundToIntegralTiesToAway

;3 clocks ;3 clocks

;3 clocks

:1 clock

roundiointegralliesloAway	;3 CIOCKS
roundToIntegralTowardZero	;3 clocks
roundToIntegralTowardPositive)	;́3 clocks
roundToIntegralTowardNegative	:3 clocks
roundToIntegralExact	;3 clocks
nextUp	;3 clocks
nextDown	
remainder	;14 clock
minNum	;3 clocks
maxNum	;3 clocks
minNumMag	;3 clocks
maxNumMag	;3 clocks
scaleB	;4 clocks
logB	;4 clocks
adďition	:5 clocks
subtraction	;5 clocks
multiplication	;4 clocks
division	;8 clocks
squareRoot	;6 clocks
fusedMultiplyAdd	;6 clocks
	;2 clocks
convertFromInt	,2 CTOCKS
convertFromInt	;2 clocks
convertFromInt	;2 clocks
convertFromInt .	;¿ clocks
convertToIntegerTiesToEven	;3 clocks
convertToIntegerTowardZero	;3 clocks
convertToIntegerTowardPositive	;3 clocks
convertToIntegerTowardNegative	:3 clocks
convertToIntegerTiesToAway	:3 clocks
convertToIntegerExactTiesToEven	:3 clocks
convertToIntegerExactTowardZero	:3 clocks
convertToIntegerExactTowardPositi	;2 clocks ;2 clocks ;3 clocks
convertToIntegerExactTowardNegati	ve ;3 clocks
convertToIntegerExactTiesToAway	;3 clocks
convertFormat	;3 clocks
	,3 CIUCKS
convertFromDecimalCharacter	;7 clocks
convertToDecimalCharacter	; <u>8</u> clocks
convertFromHexCharacter	;7 clocks
convertToHexCharacter	;5 clocks
copy	;3 clocks
negate	;3 clocks
abs	;3 clocks
	. 2 - a la alca

### **Computational Signaling Operations**

copySign

compareSignalingEqual compareQuietEqual compareSignalingNotEqual compareQuietNotEqual compareSignalingGreater compareQuietGreater compareQuietGreaterEqual compareQuietGreaterEqual compareSignalingLess compareQuietLess compareQuietLess compareQuietLessEqual compareSignalingNotGreater compareQuietNotGreater compareQuietNotGreater compareQuietLessUnordered compareQuietLessUnordered compareQuietNotLess compareQuietNotLess compareQuietNotLess compareQuietTotLess compareQuietOrdered compareQuietGreaterUnordered compareQuietUnordered compareQuietUnordered	;1 clock
<pre>IF (compareTrue) GOTO: <label> IF NOT(compareTrue) GOTO: <label> IF (compareTrue) GOSUB: <label></label></label></label></pre>	;1 clock ;1 clock ;1 clock

### **Non-Computational Operations**

IF NOT(compareTrue) GOSUB: <label>

non compatational operations		
is754version1985() IF (754version1985) GOTO: <label> IF NOT(754version1985) GOTO: <label></label></label>	;1	clock clock clock
is754version2008() GOTO: <label> IF (754version2008) GOTO: <label> IF NOT(754version2008) GOTO: <label></label></label></label>	;1	clock clock clock
class	;1	clock
<pre>IF (signalingNaN) GOTO: <label> ;1 clock IF (quietNaN) GOTO: <label> IF (negativeInfinity) GOTO: <label> IF (negativeNormal) GOTO: <label> IF (negativeSubnormal) GOTO: <label> IF (negativeZero) GOTO: <label> IF (positiveZero) GOTO: <label> IF (positiveSubnormal) GOTO: <label> IF (positiveSubnormal) GOTO: <label> IF (positiveNormal) GOTO: <label> IF (positiveInfinity) GOTO: <label></label></label></label></label></label></label></label></label></label></label></label></pre>	;1 ;1 ;1 ;1 ;1 ;1	clock clock clock clock clock clock clock clock
<pre>IF NOT(signalingNaN) GOTO: <label> IF NOT(quietNaN) GOTO: <label> IF NOT(negativeInfinity) GOTO: <label> IF NOT(negativeNormal) GOTO: <label> IF NOT(negativeSubnormal) GOTO: <label> IF NOT(negativeZero) GOTO: <label> IF NOT(positiveZero) GOTO: <label> IF NOT(positiveSubnormal) GOTO: <label> IF NOT(positiveNormal) GOTO: <label> IF NOT(positiveInfinity) GOTO: <label></label></label></label></label></label></label></label></label></label></label></pre>	;1 ;1 ;1 ;1 ;1 ;1 ;1	clock clock clock clock clock clock clock clock clock
<pre>IF (signalingNaN) GOSUB: <label> IF (quietNaN) GOSUB: <label> IF (negativeInfinity) GOSUB: <label> IF (negativeNormal) GOSUB: <label> IF (negativeSubnormal) GOSUB: <label> IF (negativeZero) GOSUB: <label> IF (positiveZero) GOSUB: <label> IF (positiveSubnormal) GOSUB: <label> IF (positiveNormal) GOSUB: <label> IF (positiveInfinity) GOSUB: <label></label></label></label></label></label></label></label></label></label></label></pre>	;1 ;1 ;1 ;1 ;1 ;1 ;1	clock clock clock clock clock clock clock clock clock
<pre>IF NOT(signalingNaN) GOSUB: <label> IF NOT(quietNaN) GOSUB: <label> IF NOT(negativeInfinity) GOSUB: <label> IF NOT(negativeNormal) GOSUB: <label> IF NOT(negativeSubnormal) GOSUB: <label> IF NOT(negativeZero) GOSUB: <label> IF NOT(positiveZero) GOSUB: <label> IF NOT(positiveSubnormal) GOSUB: <label> IF NOT(positiveNormal) GOSUB: <label> IF NOT(positiveNormal) GOSUB: <label> IF NOT(positiveInfinity) GOSUB: <label></label></label></label></label></label></label></label></label></label></label></label></pre>	;1 ;1 ;1 ;1 ;1 ;1	clock clock clock

IF NOT(positiveInfinity) GOSUB: <label> ;1 clock

Non-exceptional predicates		
isInfinite(fh:sqrt.15)	;1 ;1 ;1 ;1 ;1 ;1	clock clock clock clock clock clock clock
<pre>IF (Subnormal) GOTO: <label> IF (Infinite) GOTO: <label></label></label></pre>	;1 ;1 ;1 ;1 ;1 ;1	clock clock clock clock clock clock clock
<pre>IF NOT(Zero) GOTO: <label> IF NOT(Subnormal) GOTO: <label> IF NOT(Infinite) GOTO: <label></label></label></label></pre>	;1 ;1 ;1 ;1 ;1 ;1	clock clock clock clock clock clock clock
<pre>IF (SignMinus) GOSUB: <label> IF (Normal) GOSUB: <label> IF (Finite) GOSUB: <label> IF (Zero) GOSUB: <label> IF (Subnormal) GOSUB: <label> IF (Infinite) GOSUB: <label> IF (NaN) GOSUB: <label> IF (Signaling) GOSUB: <label> IF (Canonical) GOSUB: <label></label></label></label></label></label></label></label></label></label></pre>	;1 ;1 ;1 ;1 ;1 ;1	clock clock clock clock clock clock clock
<pre>IF NOT(SignMinus) GOSUB: <label> IF NOT(Normal) GOSUB: <label> IF NOT(Finite) GOSUB: <label> IF NOT(Zero) GOSUB: <label> IF NOT(Subnormal) GOSUB: <label> IF NOT(Infinite) GOSUB: <label> IF NOT(NaN) GOSUB: <label> IF NOT(Signaling) GOSUB: <label> IF NOT(Canonical) GOSUB: <label></label></label></label></label></label></label></label></label></label></pre>	;1 ;1 ;1 ;1 ;1 ;1	clock clock clock clock clock clock clock
radix	;1	clock
totalOrder IF (totalOrder) GOTO: <label> IF NOT(totalOrder) GOTO: <label> IF (totalOrder) GOSUB: <label> IF NOT(totalOrder) GOSUB: <label></label></label></label></label>	;1 ;1 ;1	clock clock clock clock clock
<pre>totalOrderMag(fs:work_1, fs:work_2) IF (totalOrderMag) GOTO: <label> IF NOT(totalOrderMag) GOTO: <label> IF (totalOrderMag) GOSUB: <label> IF NOT(totalOrderMag) GOSUB: <label></label></label></label></label></pre>	;1 ;1 ;1	clock clock clock clock clock

# **Operations on Flag Subsets** lowerFlags

raiseFlags testFlags

exp

or

integer input in degrees)

default RaiseNoFlag	;1 clock ;1 clock
Resuming Alternate Exception Handling At	tributes
<pre>IF (aFlagRaised) GOTO: <label> IF NOT(aFlagRaised) GOTO: <label> IF (aFlagRaised) GOSUB: <label> IF NOT(aFlagRaised) GOSUB: <label></label></label></label></label></pre>	;1 clock ;1 clock ;1 clock ;1 clock
testSavedFlags restoreFlags saveAllFlags()	;1 clock ;1 clock ;1 clock
	,

;1 clock ;1 clock

;1 clock

;5 clocks

;2 clocks

;1 clock ;1 clock

;1 clock

;1 clock

;1 clock

;1 clock

;1 clock

# in Status

raiseSignals lowerSignals raiseSignals lowerSignals	;1 clock ;1 clock ;1 clock ;1 clock
<pre>enableAltImmediateHandlers disableAltImmediateHandlers</pre>	;1 clock ;1 clock
Set and Clear Alternate Exception S Register	ubstitution Bits in
setSubsInexact	;1 clock

Implemented (but not required) Correctly Rounded Functions				
setSubsInvalid clearSubsInvalid	;1 clock ;1 clock ;1 clock			
setSubsDivByZero clearSubsDivByZero	;1 clock ;1 clock			
clearSubsOverflow	;1 clock			
setSubsOverflow	;1 clock			
clearSubssubsUnderflow	;1 clock			
setSubssubsUnderflow	:1 clock			
clearSubsInexact	;I clock			

#### log ;9 clocks ;13 clocks pown ;13 clocks pow ;13 clocks

Implemented Correctly Rounded Trig Functions (these accept

sind cosd tand cotd	;3 clocks ;3 clocks ;3 clocks ;3 clocks
Operations on Dynamic Modes	
setBinaryRoundingDirection(NEAREST) setBinaryRoundingDirection(POSITIVE) setBinaryRoundingDirection(NEGATIVE) setBinaryRoundingDirection(AWAY) setBinaryRoundingDirection(ZERO) saveModes() restoreModes(ub:savedModes) defaultModes()	;1 clock

#### **Native Integer, Logical and Bit Test and Branch Operators** and ;2 clocks

<pre>vor add Setc addc SetC subb mul div min max bset bclr compare shift.0 = shift:(uh:work_3, LEFT, 1) shift.1 = shift:(uh:work_3, RIGHT, 3) shift.2 = shift:(uh:work_3, ASL, 5) shift.3 = shift:(uh:work_3, ASL, 5) shift.4 = shift:(uh:work_3, ROL, 8) shift.5 = shift:(uh:work_3, ROL, 8) shift.6 = shift:(uh:work_3, ASR, 6) shift.7 = shift:(uh:work_3, ROR, 20) endi.0 = endi convertFromBinaryToASCII convertTromBinaryFromASCII enableInt disableInt setV clearV setN clearV setN clearV setN clearC setZ clearZ IF (Z==1) GOTO: <label> IF (A==B) GOTO: <label> IF (C==0) GOTO: <label> IF (A==B) GOTO: <label> IF (A==B) GOTO: <label> IF (N==1) GOTO: <label> IF (A&lt;=B) GOTO: <label> IF (</label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></label></pre>	;2 clocks ;2 clocks ;1 clocks ;2 clock ;1 clock
<pre>IF (Z==1) GOSUB: <label> IF (Z==0) GOSUB: <label> IF (A==B) GOSUB: <label> IF (A!=B) GOSUB: <label> IF (C==1) GOSUB: <label> IF (C==0) GOSUB: <label> IF (N==1) GOSUB: <label> IF (N==0) GOSUB: <label> IF (V==1) GOSUB: <label> IF (V==0) GOSUB: <label> IF (V==0) GOSUB: <label> IF (A<b) <label="" gosub:=""> IF (A<b) <label="" gosub:=""> IF (A&gt;=B) GOSUB: <label> IF (A&lt;=B) GOSUB: <label> IF (A&lt;=B) GOSUB: <label></label></label></label></b)></b)></label></label></label></label></label></label></label></label></label></label></label></pre>	;1 clock
<pre>IF (uw:work_3:[bit8]==0) GOTO: <label> IF (uw:work_3:[bit7]==1) GOTO: <label> IF (uw:work_3:[bit6]==0) GOSUB: <label (uw:work_3:[bit5]="=1)" <label<="" gosub:="" if="" pre=""></label></label></label></pre>	;1 clock > ;1 clock

\*AR1++[8] = convertToHexCharacter:(uh:\*AR0++[8], ub:#0)

FOR (LPCNT0 = uw:#3) (

NEXT LPCNTO GOTO: loop\_0 )

nop

GOTO <label>

GOSUB < label>

REPEAT uh:#15

RETURN