

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

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

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. 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 auto-

post-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.

nemory. Typicai fioa	ating-point performan	ce: 100 million FLOI	PS. Peak: 300 million 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
			logB
	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	Modified	Harvard	maxNumMag
IRQ	IEEE 754.	-2008 ISA	сору
READY	·	ge x 64-bit n Pipeline	negate
DONE	instruction C	T Pipellile	abs
	CF	<b>'</b> 0	copySign
CEN			compareSignalingEqual
CE123			class
BWE[7:0] WE	NBT (No Bus	Turn Around)	isSignMinus
● VE OE	Flow Thro	ugh Mode	totalOrder
	Synchronous : Byte-Writeable		totalOrderMag
Address[31:0]	External Mem		convertFormat
Data[63:0]			roundToIntegralTiesToEven
Data[63:0]	1		convertFromDecimalCharacter
			convertToDecimalCharacter
TCK	IEEE 114	9.1 (JTAG)	convertFromHexCharacter
TMS	Real-Time (	On-The-Fly)	convertToHexCharacter
TRSTn TDI	Data Ex and Del	•	convertFromInt
TDO	Inter		convertToIntegerTiesToEven

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

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

convertToIntegerExactTiesToEver

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-

**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

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, break-

pointed 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 instantaneous This means that whenever the parent CPU issues an

instructions, can command any one, any set, or all "child" XCUs to perform a given operation simultaneously. Such operations are synchronized and 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

## **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

totalOrder

totalOrderMag

for loading UltraRAM program memory.

Standard blockRAMs are configured for use as three-port data memory. This enables movement of dual operands simultaneously into integer, logical and floating-point operators every clock cycle.

> **Memory-Mapped** All internal registers, including PC, Status, Auxilliary Registers, Repeat Counter, hardware Loop

All Registers and Operators are

Counters, Stack Pointer, etc., are memory-mapped.

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	NAM	NAW	roundToIntegralTiesToEven	KAWI	NAM	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		XCIL	<b>O</b> nextDown		XCI	nextDown
DONE	Instructio	· · · · · · · · · · · · · · · · · · ·	abs		T	convertFromInt			convertFromInt
	] CH	PU	copySign			convertToIntegerTiesToEven			convertToIntegerTiesToEven
CEN			compareSignalingEqual			convertToIntegerExactTiesToEven			convertToIntegerExactTiesToEven
CE123	]		class			minNum			minNum
BWE[7:0]	NBT (No Bus	Turn Around)	isSignMinus	Modified Harvard IEEE 754-2008 ISA Three-Stage x 64-bit Instruction Pipeline		maxNum	Modified Harvard IEEE 754-2008 ISA Three-Stage x 64-bit Instruction Pipeline		maxNum
WE OE		ugh Mode	totalOrder			minNumMag			minNumMag
<b>4</b>	Synchronous	x64-Bit SRAM	totalOrderMag			maxNumMag			maxNumMag
Address[31:0]	Byte-Writeable External Men	Bi-Directional	convertFormat	CI	PU	сору	CF	יי	сору
\\	Zxtorriar mon	iory interruce	roundToIntegralTiesToEven			negate			negate
Data[63:0]	}		convertFromDecimalCharacter			abs			abs
,			convertToDecimalCharacter			copySign			copySign
TCK	IEEE 114	9.1 (JTAG)	convertFromHexCharacter			compareSignalingEqual			compareSignalingEqual
TMS	Real-Time (		convertToHexCharacter			class			class
TRSTn	Data Ex	change	convertFromInt			isSignMinus			isSignMinus
TDI TDO	and De Inter	bugging face	convertToIntegerTiesToEven			totalOrder			totalOrder
100									

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%, URAM 25%, CPU 256k prog mem, 128k data mem, XCUs 64k data mem, 32k prog mem, 1.87W@100MHz. Typical floating-

convertToIntegerExactTiesToEven

point 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	VOL
			logB	XCU 0	XCU1
	BlockRAM	BlockRAM	nextUp		
	32k-Byte (x 64)	32k-Byte (x 64)	nextDown		
	Dual-Port	Three-Port	minNum		
	Program RAM	Data RAM	maxNum		
CLK	KAIVI	KAW	minNumMag		
RESET		l Harvard	maxNumMag		
IRQ		-2008 ISA	сору		
READY	I hree-Sta	ge x 64-bit n Pipeline	negate		
DONE		<b>2</b> U	abs		
	Ci	-0	copySign		
CEN			compareSignalingEqual		
CE123 BWE[7:0]	4		class		
₩E WE	NBT (No Bus	Turn Around)	isSignMinus		
OE		ough Mode	totalOrder		
Address[31:0]		x64-Bit SRAM Bi-Directional	totalOrderMag		
Address[31:0]		nory Interface	convertFormat	XCU 2	XCU 3
Data[63:0]	y .		roundToIntegralTiesToEven		
\/	<b></b>		convertFromDecimalCharacter		
			convertToDecimalCharacter		
TCK TMS		9.1 (JTAG)	convertFromHexCharacter		
TRSTn		On-The-Fly)	convertToHexCharacter		
TDI		xchange bugging	convertFromInt		
TDO		rface	convertToIntegerTiesToEven		

convertToIntegerExactTiesToEven

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 copy negate	XC	X	C <b>J</b> 1	X	CJ-2	XCU 3	
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	Turn Around) bugh Mode x64-Bit SRAM e Bi-Directional hory Interface	abs  copySign  compareSignalingEqual  class  isSignMinus  totalOrder  totalOrderMag  convertFormat  roundToIntegralTiesToEven  convertFromDecimalCharacter  convertFromHexCharacter  convertToHexCharacter  convertToHexCharacter  convertToIntegerTiesToEven  convertToIntegerExactTiesToEven	XC	X	CU 5	X	CJ 6	XCU7	

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.

naii-precision iloating-point with all IEEE 754-2006 operato	XCU 0	XCU1	XCU2	XCU-3
addition subtraction				
UltraRAM  1M-Byte x 64 Dual-Port Program RAM  RAM  BlockRAM  256k-Byte x 64 Three-Port Data RAM  RAM  multiplicatio division remainder remainder RAM fusedMultiply/ scaleB		XCU 5	XCU 6	XCU 7
BlockRAM  32k-Byte (x 64) Dual-Port Program RAM  CLK  RESET IRQ  BlockRAM  32k-Byte (x 64) Three-Port Data RAM  Modified Harvard IEEE 754-2008 ISA Three-Stage x 64-bit  BlockRAM  32k-Byte (x 64) Three-Port Data RAM  minNum  maxNum  maxNumMa  copy				
Three-Stage x 64-bit Instruction Pipeline  CPU  CEN  CE123  BWE[7:0]  WE  NBT (No Bus Turn Around) Flow Through Mode Synchronous x64-Bit SRAM  Three-Stage x 64-bit Instruction Pipeline  CPU  CompareSignaling  class  isSignMinus  isSignMinus  totalOrderMa				
Address[31:0]  Byte-Writeable Bi-Directional External Memory Interface  ConvertForm  roundToIntegralTie convertFromDecimal  ConvertFromDecimal  ConvertFromDecimal  ConvertFromHexCit  TMS  Real-Time (On-The-Fly)  Data Exchange  ConvertFrom  ConvertFromHexCit  ConvertFromHexCit  ConvertFromHexCit  ConvertFromHexCit  ConvertFrom  C	t XCUB ToEven Character laracter aracter aracter	XCU 9	XCU 10	XCU 11
TDI and Debugging Interface ConvertToIntegerTie convertToIntegerExact				
	XCU 12	XCU 13	XCU 14	XCU 15

#### **Operations Implemented in Hardware** (actual SYMPL IL mnemonics shown): roundToIntegralTiesToEven ;3 clocks

roundToIntegralTiesToEven	;3 Clocks
roundToIntegralTiesToAway	;3 clocks
roundToIntegralTowardZero	;́3 clocks
roundToIntegralTowardPositive)	;3 clocks
roundToIntegralTowardNegative	;́3 clocks
roundToIntegralExact	;3 clocks
nextUp	;3 clocks
nextDown	;3 clocks
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
füsedMultiplyAdd	;6 clocks
convertFromInt	;2 clocks
convertFromInt	;́2 clocks
convertFromInt	;2 clocks
convertFromInt	;́2 clocks
convertToIntegerTiesToEven	;3 clocks
convertToIntegerTowardZero	;3 clocks
convertToIntegerTowardPositive	;3 clocks ;3 clocks
convertToIntegerTowardNegative	:3 clocks
convertToIntegerTiesToAway	:3 clocks
convertToIntegerExactTiesToEven	;3 clocks
convertToIntegerExactTowardZero	;́3 clocks
convertToIntegerExactTowardPositive	;3 clocks
convertToIntegerExactTowardNegative	;́3 clocks
convertToIntegerExactTiesToAway	;3 clocks
convertFormat	;3 clocks
convertFromDecimalCharacter	;7 clocks
convertToDecimalCharacter	; <u>8</u> clocks
convertFromHexCharacter	;7 clocks
convertToHexCharacter	;5 clocks
copy	;3 clocks
negate	;3 clocks

# **Computational Signaling Operations**

copySign

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

;3 clocks

:3 clocks

;1 clock ;1 clock

### **Non-Computational Operations**

IF (compareTrue) GOSUB: <label>

IF NOT(compareTrue) GOSUB: <label>

is754version1985() IF (754version1985) GOTO: <label> IF NOT(754version1985) GOTO: <label></label></label>	;1	cloc cloc cloc
is754version2008() GOTO: <label> IF (754version2008) GOTO: <label> IF NOT(754version2008) GOTO: <label></label></label></label>	;1	cloc cloc cloc
class	;1	cloc
<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	cloc cloc cloc cloc cloc cloc cloc
<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	cloc cloc cloc cloc cloc cloc cloc
<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	cloc cloc cloc cloc cloc cloc cloc
	;1 ;1 ;1	cloc

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

IF NOT(negativeZero) GOSUB: <label>

IF NOT(positiveZero) GOSUB: <label>

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

		,	_	
Non-exceptional predicate	s			
<pre>isSignMinus(fh:negate.3) isNormal(fh:sqrt.15) isFinite(fh:sqrt.15) isZero(fh:sqrt.15) isSubnormal(fh:sqrt.15) isInfinite(fh:sqrt.15) isNaN(fh:sqrt.15) isSignaling(fh:sqrt.15) isCanonical(fh:sqrt.15)</pre>		;1 ;1 ;1 ;1 ;1 ;1	c] c] c] c] c]	0C 0C 0C 0C 0C
<pre>IF (SignMinus) GOTO: <la (finite)="" (normal)="" (zero)="" <label="" goto:="" if=""> IF (Subnormal) GOTO: <la (infinite)="" (nan)="" <lab="" <label="" goto:="" if=""> IF (Signaling) GOTO: <la (canonical)="" <="" <la="" goto:="" if="" pre=""></la></la></la></pre>	> > bel> bel>	;1 ;1 ;1 ;1 ;1 ;1 ;1		0C 0C 0C 0C 0C
<pre>IF NOT(SignMinus) GOTO: IF NOT(Normal) GOTO: <la <="" <la="" <labe="" goto:="" goto:<="" if="" not(canonical)="" not(finite)="" not(infinite)="" not(nan)="" not(signaling)="" not(subnormal)="" not(zero)="" pre=""></la></pre>	bel> bel> l> <label> label&gt; l&gt;</label>	;1 ;1 ;1 ;1 ;1 ;1	c] c] c] c] c] c]	0C 0C 0C 0C 0C 0C
<pre>IF (SignMinus) GOSUB: &lt;1 IF (Normal) GOSUB: &lt;1abe IF (Finite) GOSUB: &lt;1abe IF (Zero) GOSUB: &lt;1abe1&gt; IF (Subnormal) GOSUB: &lt;1 IF (Infinite) GOSUB: &lt;1a IF (NaN) GOSUB: &lt;1abe1&gt; IF (Signaling) GOSUB: &lt;1 IF (Canonical) GOSUB: &lt;1</pre>	l> l> abel> bel>	;1 ;1 ;1 ;1 ;1 ;1	c] c] c] c] c]	0C 0C 0C 0C 0C
<pre>IF NOT(SignMinus) GOSUB: IF NOT(Normal) GOSUB: <l <l="" <lab="" gosub:="" gosub:<="" if="" not(canonical)="" not(finite)="" not(infinite)="" not(nan)="" not(signaling)="" not(subnormal)="" not(zero)="" pre=""></l></pre>	abel> abel> el> <label> <label> el&gt; <label></label></label></label>	;1 ;1 ;1 ;1 ;1 ;1	c] c] c] c] c]	0C 0C 0C 0C 0C
radix		;1	c1	oc
totalOrder IF (totalOrder) GOTO: <l (totalorder)="" <="" gosub:="" gosub<="" goto:="" if="" not(totalorder)="" td=""><td><label> label&gt;</label></td><td>;1 ;1 ;1 ;1</td><td>c] c] c]</td><td>0C 0C 0C</td></l>	<label> label&gt;</label>	;1 ;1 ;1 ;1	c] c] c]	0C 0C 0C

### IF NOT(totalorderMag) GOSUB: <label> **Operations on Flag Subsets** lowerFlags

enableAltImmediateHandlers

disableAltImmediateHandlers

raiseFlags

testFlags

totalOrderMag(fs:work\_1, fs:work\_2)

IF NOT(totalOrderMag) GOTO:\_<label>

IF (totalOrderMag) GOSUB: <label>

IF (totalOrderMag) GOTO: <label>

;1 clock

;1 clock

;1 clock ;1 clock

;1 clock

;1 clock ;1 clock

;1 clock

;1 clock ;1 clock

;2 clocks ;2 clocks

testSavedFlags	;1 clock
restoreFlags	;1 clock
saveAllFlags()	;1 clock
<pre>IF (aFlagRaised) GOTO: <label> IF NOT(aFlagRaised) GOTO: <lab (aflagraised)="" <lab<="" <label="" gosub:="" if="" not(aflagraised)="" pre=""></lab></label></pre>	pel> ;1 clock  > ;1 clock
Resuming Alternate Exception H	landling Attributes
default	;1 clock
RaiseNoFlag	;1 clock
raiseSignals	;1 clock
lowerSignals	;1 clock
raiseSignals	;1 clock
lowerSignals	:1 clock

### **Set and Clear Alternate Exception Substitution Bits in Status** Register

setSubsInexact clearSubsInexact setSubssubsUnderflow clearSubssubsUnderflow setSubsOverflow clearSubsOverflow setSubsDivByZero clearSubsDivByZero setSubsInvalid clearSubsInvalid	;1 clock
---	--

## ;5 clocks ;9 clocks exp log

Implemented (but not required) Correctly Rounded Functions

pown	;13 clocks ;13 clocks
pow powr	;13 clocks
Implemented ( integer input in	Correctly Rounded Trig Functions (these accept degrees)
sind	;3 clocks :3 clocks
cosd	:3 clocks

### ;3 clocks tand cotd ;3 clocks **Operations on Dynamic Modes**

Operations on Dynamic Modes		
setBinaryRoundingDirection(NEAsetBinaryRoundingDirection(POSsetBinaryRoundingDirection(NEGsetBinaryRoundingDirection(AWAsetBinaryRoundingDirection(ZERsaveModes())restoreModes(ub:savedModes)defaultModes()	SITIVE) ;1 clock SATIVE) ;1 clock NY) ;1 clock	
Native Integer, Logical and Bit Test and Branch Operators		

and

or     xor     add     Setc     addc     sub     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, LSL, 10)     shift.3 = shift:(uh:work_3, ASL, 5)     shift.4 = shift:(uh:work_3, ASL, 5)     shift.5 = shift:(uh:work_3, ASR, 6)     shift.6 = shift:(uh:work_3, ASR, 14)     shift.7 = shift:(uh:work_3, ROR, 20)     endi.0 = endi     convertFromBinaryToASCII     convertToBinaryFromASCII     enableInt     setV     clearV     setN     clearV     setN     clearC     setZ     clearZ     IF (Z==1) GOTO: <label>     IF (A=B) GOTO: <label>     IF (A=B) GOTO: <label>     IF (C==0) GOTO: <label>     IF (N==1) GOTO: <label>     IF (N==1) GOTO: <label>     IF (N==1) GOTO: <label>     IF (N==0) GOTO: <label>     IF (A&gt;B) GOTO: <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>	;2 clocks ;2 clocks ;1 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 (N==0) GOSUB: <label> IF (V==1) GOSUB: <label> IF (V==0) GOSUB: <label> IF (A<b) <label="" gosub:=""> IF (A&gt;=B) GOSUB: <label> IF (A&gt;=B) GOSUB: <label> IF (A&gt;=B) GOSUB: <label> IF (A&lt;=B) GOSUB: <label> IF (A&lt;=B) GOSUB: <label> IF (A&lt;=B) GOSUB: <label></label></label></label></label></label></label></b)></label></label></label></label></label></label></label></label></label></label></label></pre>	;1 clock

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

IF (uw:work\_3:[bit8]==0) GOTO: <label> ;1 clock
IF (uw:work\_3:[bit7]==1) GOTO: <label> ;1 clock
IF (uw:work\_3:[bit6]==0) GOSUB: <label> ;1 clock
IF (uw:work\_3:[bit5]==1) GOSUB: <label> ;1 clock

;1 clock

;1 clock

REPEAT uh:#15

IF (A>B) GOSUB: <label>

FOR (LPCNT0 = uw:#3) (