SYMPL

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

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

ADVANCE INFORMATION

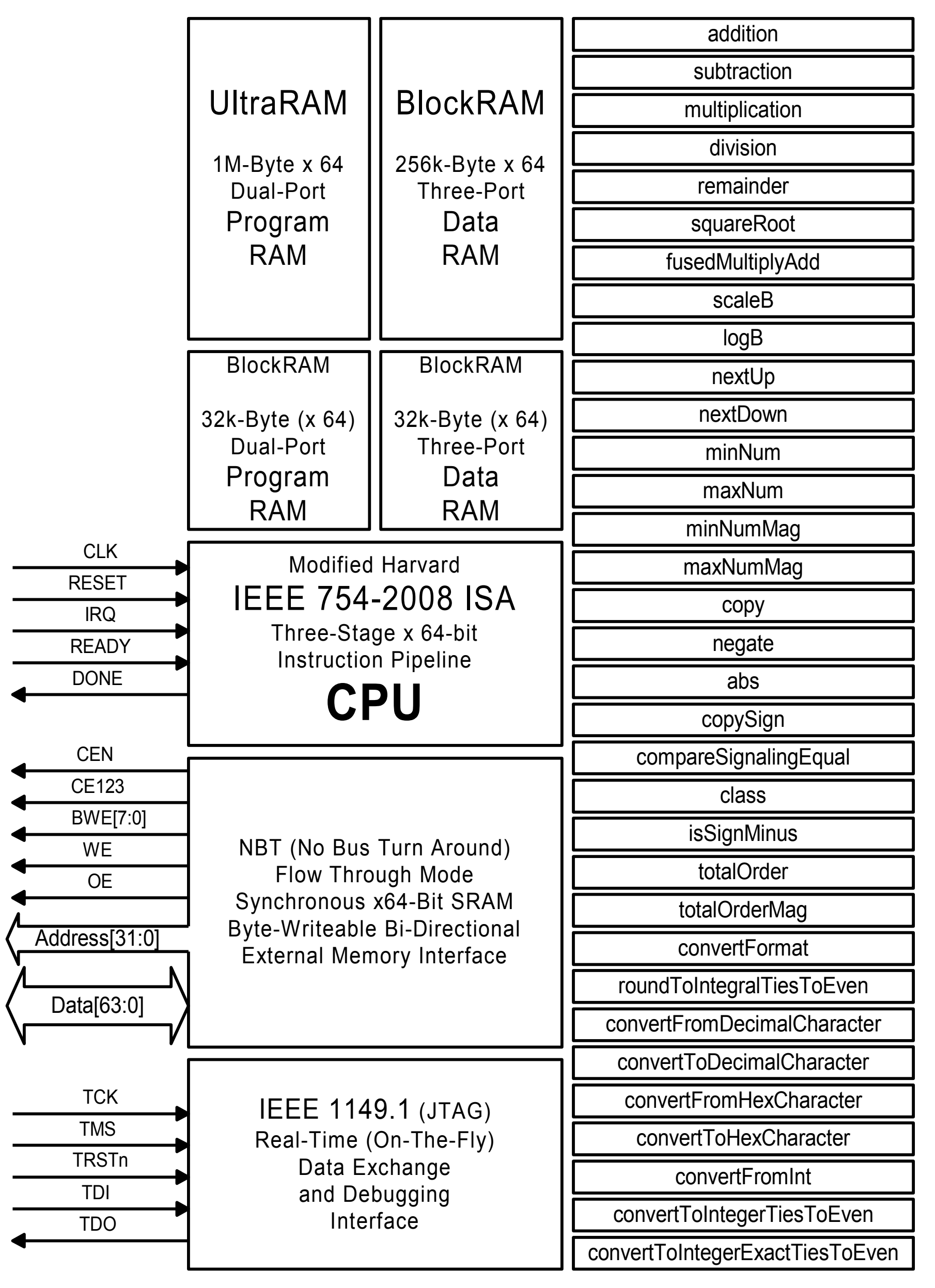
The SYMPL 64-bit, IEEE 754-2008 ISA CPU is a novel "mover" architecture that efficiently implements in hardware **"all"** operations mandated by IEEE 754-2008 using one instruction per operation. Much more efficient than conventional "load-store" models, it can push two 64-bit operands into an operator, 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.**



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 time.

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 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.

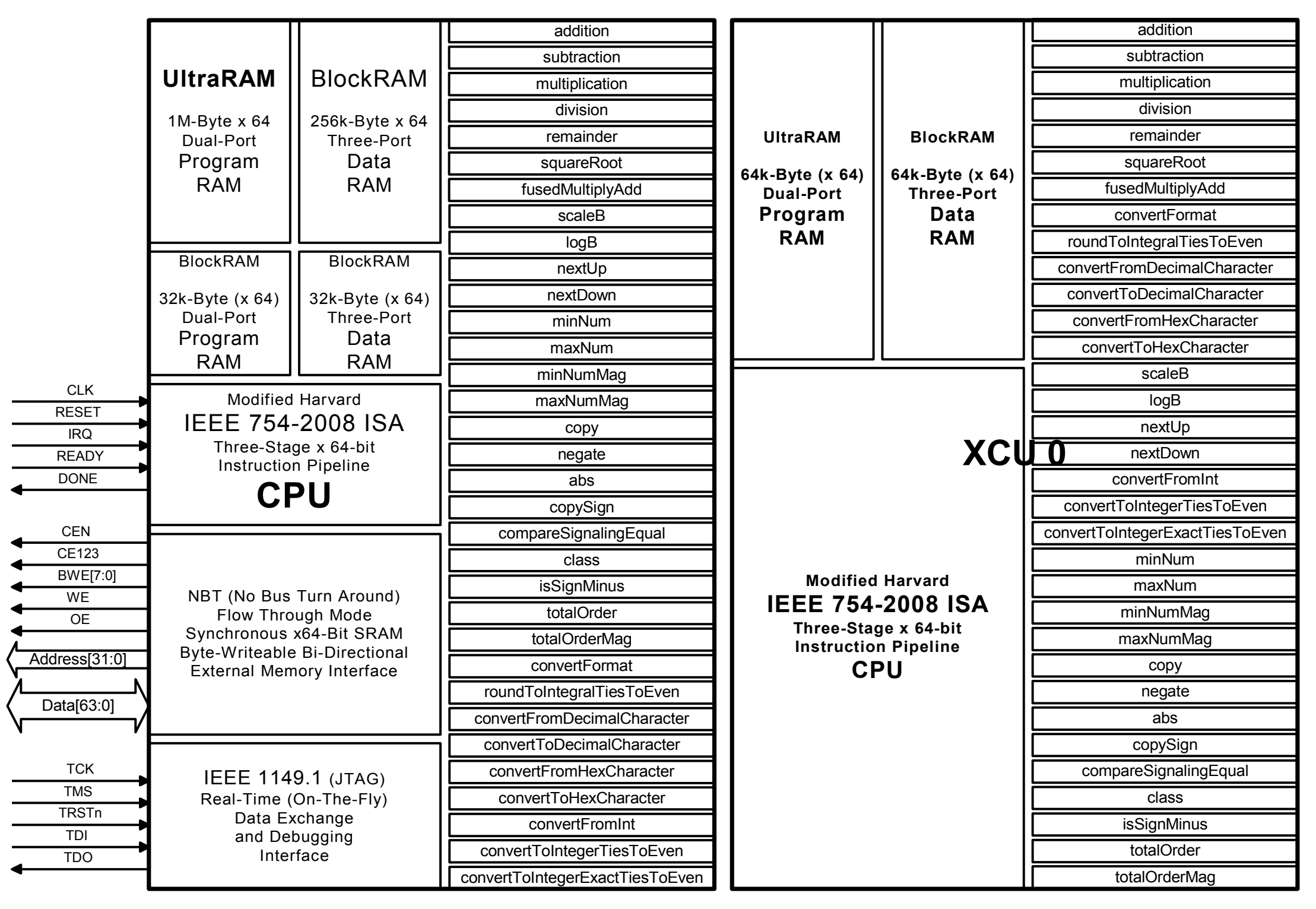
Because Kintex UltraRAMs cannot be initialized as a ROM, the first 32k bytes of program memory space are implemented using standard blockRAMs so that a micro-kernel can reside and be available at power-up for loading UltraRAM program memory.

Three-Port Data Memory

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.

Two 64-bit CPUs implemented in Kintex UltraScale+ xcku3p

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



All Registers and Operators are Memory-Mapped

All internal registers, including PC, Status, Auxilliary Registers, Repeat Counter, hardware Loop 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 memory-mapped and fully pipelined, typically occupying sixteen consecutive double-word locations per operator.

No Op-Codes

Since all registers and operators are memory-mapped, this instruction set does not employ "op-codes", as that term is traditionally understood.

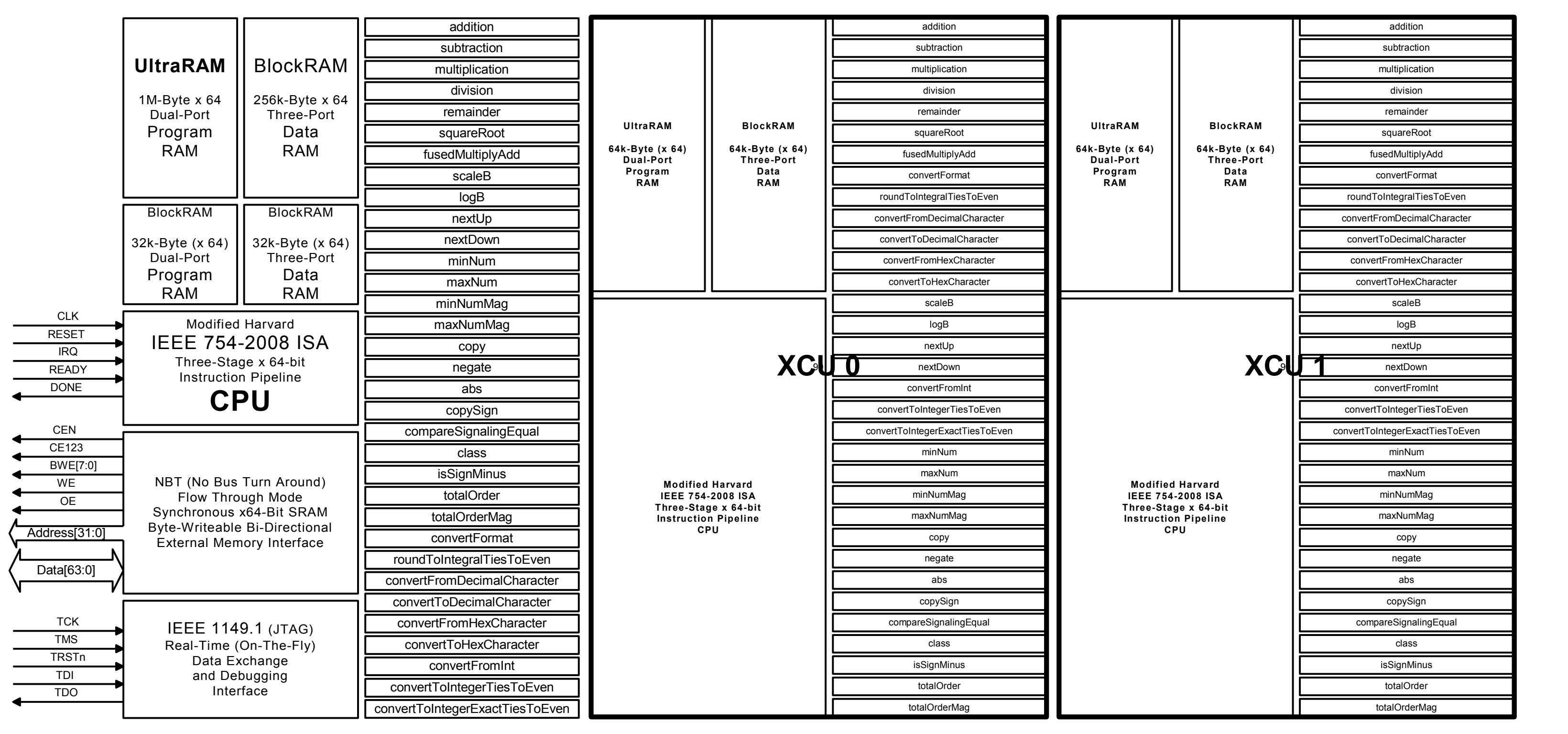
All Rounding Modes Fully Supported

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.

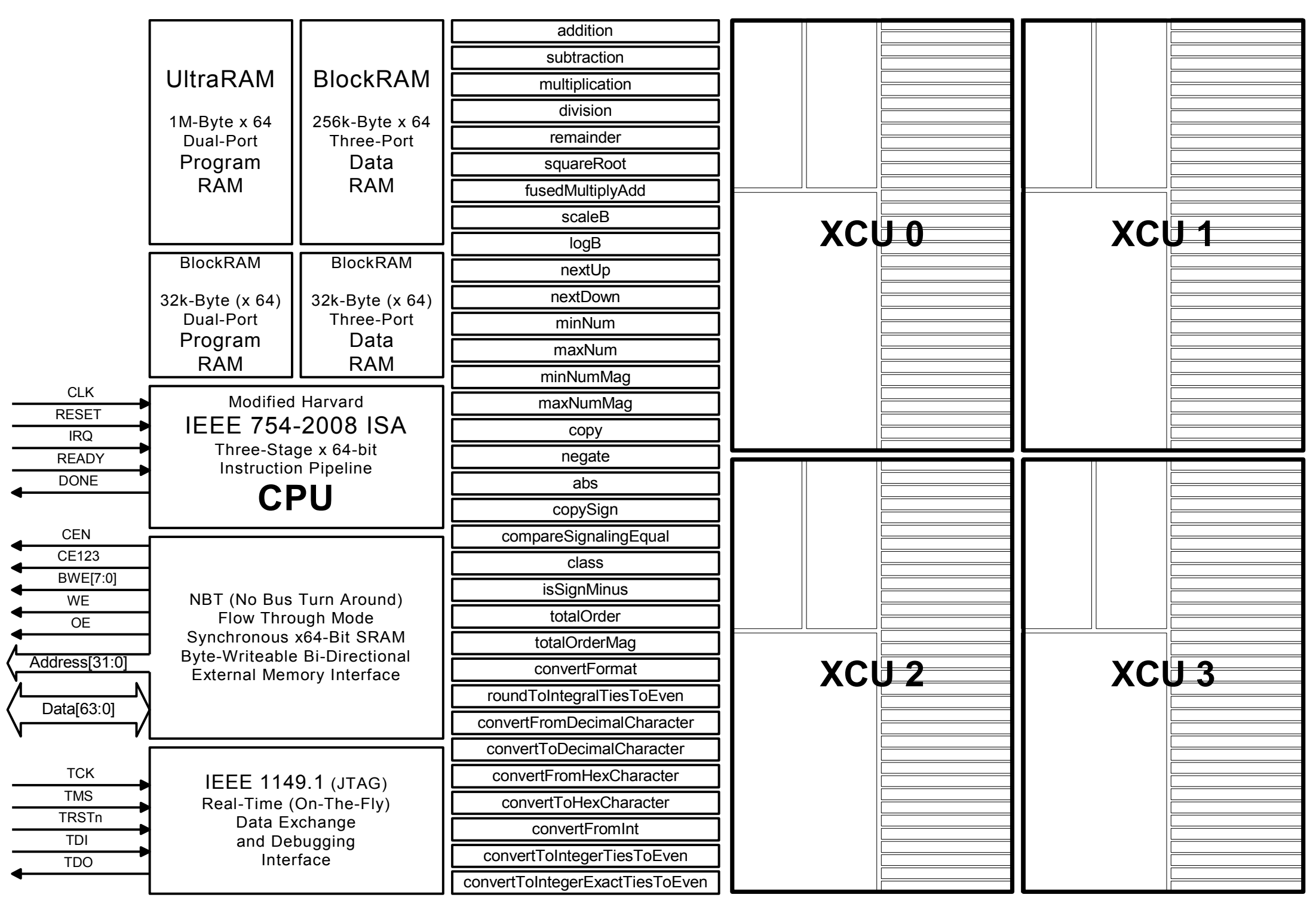
Three 64-bit CPUs implemented in Kintex UltraScale+ xcku3p

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.**



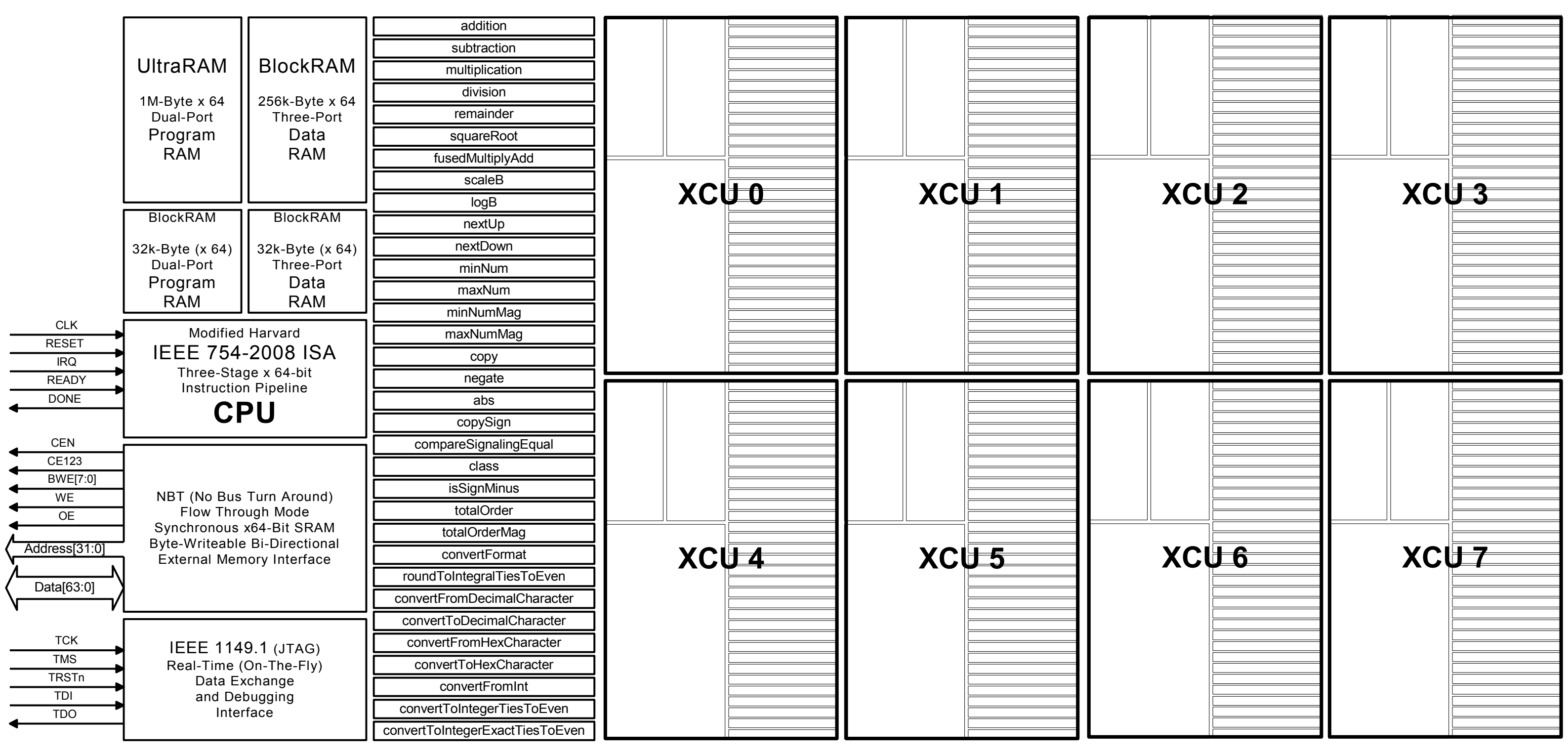
Five 64-bit CPUs implemented in Kintex UltraScale+ xcku3p

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-point performance: 500 million FLOPS. Peak: 1.5 billion FLOPS.**



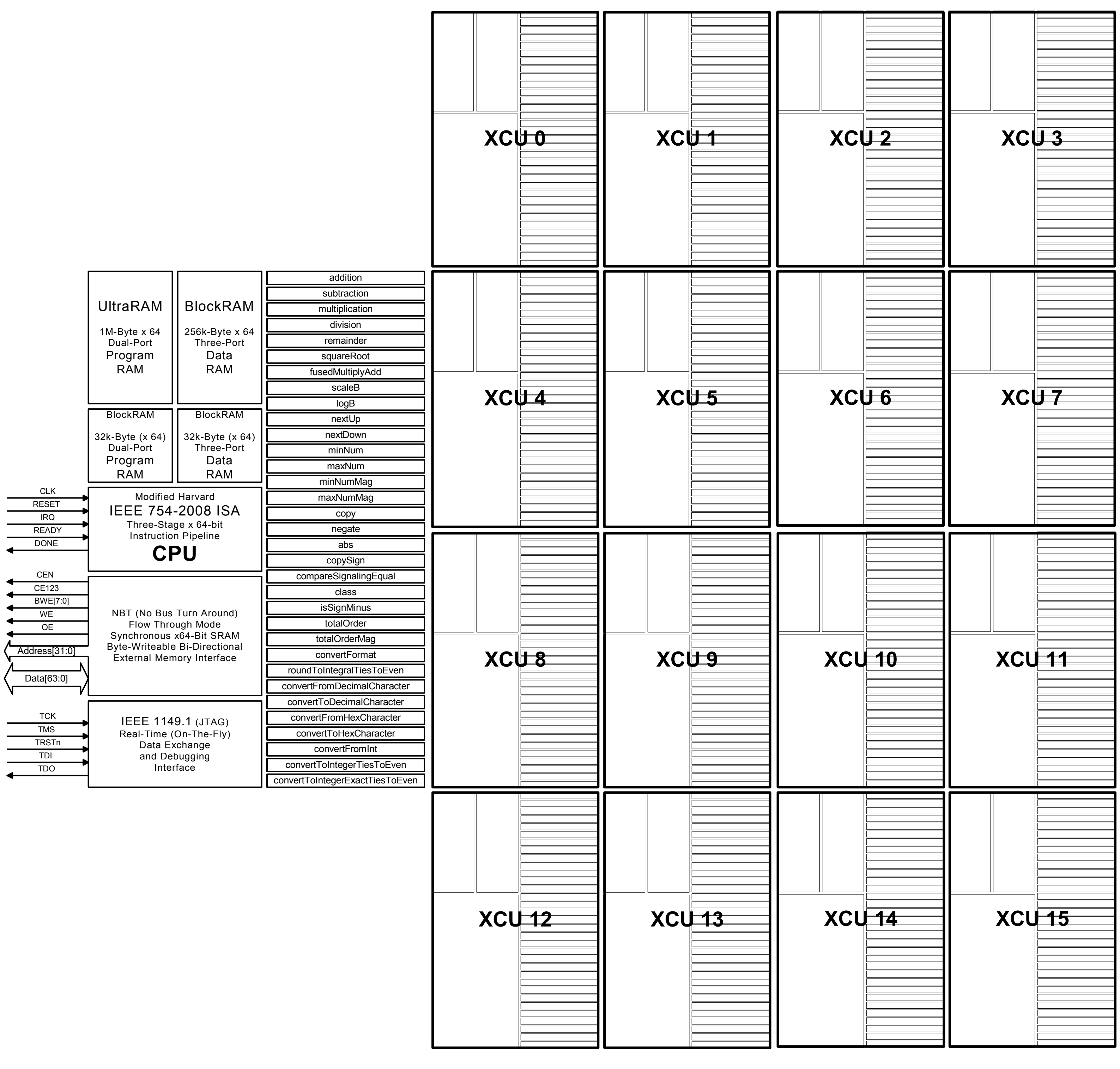
Nine 64-bit Floating-Point CPUs implemented in Kintex UltraScale+ xcku5p

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.**



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.**



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

```
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
addition                                  ;5 clocks
subtraction                               ;5 clocks
multiplication                            ;4 clocks
division                                  ;8 clocks
squareRoot                                ;6 clocks
fusedMultiplyAdd                          ;6 clocks
convertFromInt                            ;2 clocks
convertFromInt                            ;2 clocks
convertFromInt                            ;2 clocks
convertFromInt                            ;2 clocks
convertToIntegerTiesToEven                ;3 clocks
convertToIntegerTowardZero                ;3 clocks
convertToIntegerTowardPositive            ;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                 ;8 clocks
convertFromHexCharacter                   ;7 clocks
convertToHexCharacter                     ;5 clocks
copy                                      ;3 clocks
negate                                    ;3 clocks
abs                                       ;3 clocks
copySign                                  ;3 clocks
```

Computational Signaling Operations

```
compareSignalingEqual                     ;1 clock
compareQuietEqual                         ;1 clock
compareSignalingNotEqual                  ;1 clock
compareQuietNotEqual                      ;1 clock
compareSignalingGreater                   ;1 clock
compareQuietGreater                       ;1 clock
compareSignalingGreaterEqual              ;1 clock
compareQuietGreaterEqual                  ;1 clock
compareSignalingLess                      ;1 clock
compareQuietLess                          ;1 clock
compareSignalingLessEqual                 ;1 clock
compareQuietLessEqual                     ;1 clock
compareSignalingNotGreater                ;1 clock
compareQuietNotGreater                    ;1 clock
compareSignalingLessUnordered             ;1 clock
compareQuietLessUnordered                 ;1 clock
compareSignalingNotLess                   ;1 clock
compareQuietNotLess                       ;1 clock
compareSignalingGreaterUnordered          ;1 clock
compareQuietGreaterUnordered              ;1 clock
compareQuietUnordered                     ;1 clock
compareQuietOrdered                       ;1 clock

IF (compareTrue) GOTO: <label>            ;1 clock
IF NOT(compareTrue) GOTO: <label>         ;1 clock
IF (compareTrue) GOSUB: <label>           ;1 clock
IF NOT(compareTrue) GOSUB: <label>        ;1 clock
```

Non-Computational Operations

```
is754version1985()                        ;1 clock
IF (754version1985) GOTO: <label>         ;1 clock
IF NOT(754version1985) GOTO: <label>      ;1 clock

is754version2008() GOTO: <label>          ;1 clock
IF (754version2008) GOTO: <label>         ;1 clock
IF NOT(754version2008) GOTO: <label>      ;1 clock

class                                     ;1 clock

IF (signalingNaN) GOTO: <label>     ;1 clock
IF (quietNaN) GOTO: <label>               ;1 clock
IF (negativeInfinity) GOTO: <label>       ;1 clock
IF (negativeNormal) GOTO: <label>         ;1 clock
IF (negativeSubnormal) GOTO: <label>      ;1 clock
IF (negativeZero) GOTO: <label>           ;1 clock
IF (positiveZero) GOTO: <label>           ;1 clock
IF (positiveSubnormal) GOTO: <label>      ;1 clock
IF (positiveNormal) GOTO: <label>         ;1 clock
IF (positiveInfinity) GOTO: <label>       ;1 clock

IF NOT(signalingNaN) GOTO: <label>        ;1 clock
IF NOT(quietNaN) GOTO: <label>            ;1 clock
IF NOT(negativeInfinity) GOTO: <label>    ;1 clock
IF NOT(negativeNormal) GOTO: <label>      ;1 clock
IF NOT(negativeSubnormal) GOTO: <label>   ;1 clock
IF NOT(negativeZero) GOTO: <label>        ;1 clock
IF NOT(positiveZero) GOTO: <label>        ;1 clock
IF NOT(positiveSubnormal) GOTO: <label>   ;1 clock
IF NOT(positiveNormal) GOTO: <label>      ;1 clock
IF NOT(positiveInfinity) GOTO: <label>    ;1 clock

IF (signalingNaN) GOSUB: <label>          ;1 clock
IF (quietNaN) GOSUB: <label>              ;1 clock
IF (negativeInfinity) GOSUB: <label>      ;1 clock
IF (negativeNormal) GOSUB: <label>        ;1 clock
IF (negativeSubnormal) GOSUB: <label>     ;1 clock
IF (negativeZero) GOSUB: <label>          ;1 clock
IF (positiveZero) GOSUB: <label>          ;1 clock
IF (positiveSubnormal) GOSUB: <label>     ;1 clock
IF (positiveNormal) GOSUB: <label>        ;1 clock
IF (positiveInfinity) GOSUB: <label>      ;1 clock

IF NOT(signalingNaN) GOSUB: <label>       ;1 clock
IF NOT(quietNaN) GOSUB: <label>           ;1 clock
IF NOT(negativeInfinity) GOSUB: <label>   ;1 clock
IF NOT(negativeNormal) GOSUB: <label>     ;1 clock
IF NOT(negativeSubnormal) GOSUB: <label>;1 clock
IF NOT(negativeZero) GOSUB: <label>       ;1 clock
IF NOT(positiveZero) GOSUB: <label>       ;1 clock
IF NOT(positiveSubnormal) GOSUB: <label>;1 clock
IF NOT(positiveNormal) GOSUB: <label>     ;1 clock
IF NOT(positiveInfinity) GOSUB: <label>   ;1 clock
```

Non-exceptional predicates

```
isSignMinus(fh:negate.3)                  ;1 clock
isNormal(fh:sqrt.15)                      ;1 clock
isFinite(fh:sqrt.15)                      ;1 clock
isZero(fh:sqrt.15)                        ;1 clock
isSubnormal(fh:sqrt.15)                   ;1 clock
isInfinite(fh:sqrt.15)                    ;1 clock
isNaN(fh:sqrt.15)                         ;1 clock
isSignaling(fh:sqrt.15)                   ;1 clock
isCanonical(fh:sqrt.15)                   ;1 clock

IF (SignMinus) GOTO: <label>              ;1 clock
IF (Normal) GOTO: <label>                 ;1 clock
IF (Finite) GOTO: <label>                 ;1 clock
IF (Zero) GOTO: <label>                   ;1 clock
IF (Subnormal) GOTO: <label>              ;1 clock
IF (Infinite) GOTO: <label>               ;1 clock
IF (NaN) GOTO: <label>                    ;1 clock
IF (Signaling) GOTO: <label>              ;1 clock
IF (Canonical) GOTO: <label>              ;1 clock

IF NOT(SignMinus) GOTO: <label>           ;1 clock
IF NOT(Normal) GOTO: <label>              ;1 clock
IF NOT(Finite) GOTO: <label>              ;1 clock
IF NOT(Zero) GOTO: <label>                ;1 clock
IF NOT(Subnormal) GOTO: <label>           ;1 clock
IF NOT(Infinite) GOTO: <label>            ;1 clock
IF NOT(NaN)  GOTO: <label>                ;1 clock
IF NOT(Signaling) GOTO: <label>           ;1 clock
IF NOT(Canonical) GOTO: <label>           ;1 clock

IF (SignMinus) GOSUB: <label>             ;1 clock
IF (Normal) GOSUB: <label>                ;1 clock
IF (Finite) GOSUB: <label>                ;1 clock
IF (Zero) GOSUB: <label>                  ;1 clock
IF (Subnormal) GOSUB: <label>             ;1 clock
IF (Infinite) GOSUB: <label>              ;1 clock
IF (NaN) GOSUB: <label>                   ;1 clock
IF (Signaling) GOSUB: <label>             ;1 clock
IF (Canonical) GOSUB: <label>             ;1 clock

IF NOT(SignMinus) GOSUB: <label>          ;1 clock
IF NOT(Normal) GOSUB: <label>             ;1 clock
IF NOT(Finite) GOSUB: <label>             ;1 clock
IF NOT(Zero) GOSUB: <label>               ;1 clock
IF NOT(Subnormal) GOSUB: <label>          ;1 clock
IF NOT(Infinite) GOSUB: <label>           ;1 clock
IF NOT(NaN)  GOSUB: <label>               ;1 clock
IF NOT(Signaling) GOSUB: <label>          ;1 clock
IF NOT(Canonical) GOSUB: <label>          ;1 clock

radix                                     ;1 clock

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

totalOrderMag(fs:work_1, fs:work_2)       ;1 clock
IF (totalOrderMag) GOTO: <label>          ;1 clock
IF NOT(totalOrderMag) GOTO: <label>       ;1 clock
IF (totalOrderMag) GOSUB: <label>         ;1 clock
IF NOT(totalOrderMag) GOSUB: <label>      ;1 clock
```

Operations on Flag Subsets

```
lowerFlags                                ;1 clock
raiseFlags                                ;1 clock
testFlags                                 ;1 clock

testSavedFlags                            ;1 clock
restoreFlags                              ;1 clock
saveAllFlags()                            ;1 clock

IF (aFlagRaised) GOTO: <label>            ;1 clock
IF NOT(aFlagRaised) GOTO: <label>         ;1 clock
IF (aFlagRaised) GOSUB: <label>           ;1 clock
IF NOT(aFlagRaised) GOSUB: <label>        ;1 clock
```

Resuming Alternate Exception Handling Attributes

```
default                                   ;1 clock
RaiseNoFlag                               ;1 clock

raiseSignals                              ;1 clock
lowerSignals                              ;1 clock
raiseSignals                              ;1 clock
lowerSignals                              ;1 clock

enableAltImmediateHandlers                ;1 clock
disableAltImmediateHandlers               ;1 clock
```

Set and Clear Alternate Exception Substitution Bits in Status Register

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

Implemented (but not required) Correctly Rounded Functions

```
exp                                       ;5 clocks
log                                       ;9 clocks
pown                                      ;13 clocks
pow                                       ;13 clocks
powr                                      ;13 clocks
```

Implemented Correctly Rounded Trig Functions (these accept integer input in degrees)

```
sind                                      ;3 clocks
cosd                                      ;3 clocks
tand                                      ;3 clocks
cotd                                      ;3 clocks
```

Operations on Dynamic Modes

```
setBinaryRoundingDirection(NEAREST)       ;1 clock
setBinaryRoundingDirection(POSITIVE)      ;1 clock
setBinaryRoundingDirection(NEGATIVE)      ;1 clock
setBinaryRoundingDirection(AWAY)          ;1 clock
setBinaryRoundingDirection(ZERO)          ;1 clock
saveModes()                               ;1 clock
restoreModes(ub:savedModes)               ;1 clock
defaultModes()                            ;1 clock
```

Native Integer, Logical and Bit Test and Branch Operators

```
and                                       ;2 clocks
or                                        ;2 clocks
xor                                       ;2 clocks
add                                       ;2 clocks
Setc                                      ;1 clock
addc                                      ;2 clocks
sub                                       ;2 clocks
SetC                                      ;1 clock
subb                                      ;2 clocks
mul                                       ;2 clocks
div                                       ;11 clocks
min                                       ;2 clocks
max                                       ;2 clocks
bset                                      ;2 clocks
bclr                                      ;2 clock
compare                                   ;1 clock
shift.0 = shift:(uh:work_3, LEFT, 1)      ;2 clock
shift.1 = shift:(uh:work_3, RIGHT, 3)     ;2 clock
shift.2 = shift:(uh:work_3, LSL, 10)      ;2 clock
shift.3 = shift:(uh:work_3, ASL, 5)       ;2 clock
shift.4 = shift:(uh:work_3, ROL, 8)       ;2 clock
shift.5 = shift:(uh:work_3, LSR, 6)       ;2 clock
shift.6 = shift:(uh:work_3, ASR, 14)      ;2 clock
shift.7 = shift:(uh:work_3, ROR, 20)      ;2 clock
endi.0 = endi                             ;2 clock
convertFromBinaryToASCII                  ;2 clock
convertToBinaryFromASCII                  ;2 clock
enableInt                                 ;1 clock
disableInt                                ;1 clock
setV                                      ;1 clock
clearV                                    ;1 clock
setN                                      ;1 clock
clearN                                    ;1 clock
setC                                      ;1 clock
clearC                                    ;1 clock
setZ                                      ;1 clock
clearZ                                    ;1 clock
IF (Z==1) GOTO: <label>                   ;1 clock
IF (Z==0) GOTO: <label>                   ;1 clock
IF (A==B) GOTO: <label>                   ;1 clock
IF (A!=B) GOTO: <label>                   ;1 clock
IF (C==1) GOTO: <label>                   ;1 clock
IF (C==0) GOTO: <label>                   ;1 clock
IF (N==1) GOTO: <label>                   ;1 clock
IF (N==0) GOTO: <label>                   ;1 clock
IF (V==1) GOTO: <label>                   ;1 clock
IF (V==0) GOTO: <label>                   ;1 clock
IF (A<B)  GOTO: <label>                   ;1 clock
IF (A>=B) GOTO: <label>                   ;1 clock
IF (A<=B) GOTO: <label>                   ;1 clock
IF (A>B)  GOTO: <label>                   ;1 clock

IF (Z==1) GOSUB: <label>                  ;1 clock
IF (Z==0) GOSUB: <label>                  ;1 clock
IF (A==B) GOSUB: <label>                  ;1 clock
IF (A!=B) GOSUB: <label>                  ;1 clock
IF (C==1) GOSUB: <label>                  ;1 clock
IF (C==0) GOSUB: <label>                  ;1 clock
IF (N==1) GOSUB: <label>                  ;1 clock
IF (N==0) GOSUB: <label>                  ;1 clock
IF (V==1) GOSUB: <label>                  ;1 clock
IF (V==0) GOSUB: <label>                  ;1 clock
IF (A<B)  GOSUB: <label>                  ;1 clock
IF (A>=B) GOSUB: <label>                  ;1 clock
IF (A<=B) GOSUB: <label>                  ;1 clock
IF (A>B)  GOSUB: <label>                  ;1 clock

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

FOR (LPCNT0 = uw:#3) (                    ;1 clock
    nop                                   ;1 clock
    nop                                   ;1 clock
NEXT LPCNT0 GOTO: loop_0 )                ;1 clock

GOTO <label>                              ;1 clock
GOSUB <label>                             ;1 clock
RETURN                                    ;1 clock

REPEAT uh:#15
   *AR1++[8] = convertToHexCharacter:(uh:*AR0++[8], ub:#0)
```