Intel® AVX-512 Instructions and Their Use in the Implementation of Math Functions

Marius Cornea, Intel Corporation

Intel SIMD ISA Evolution

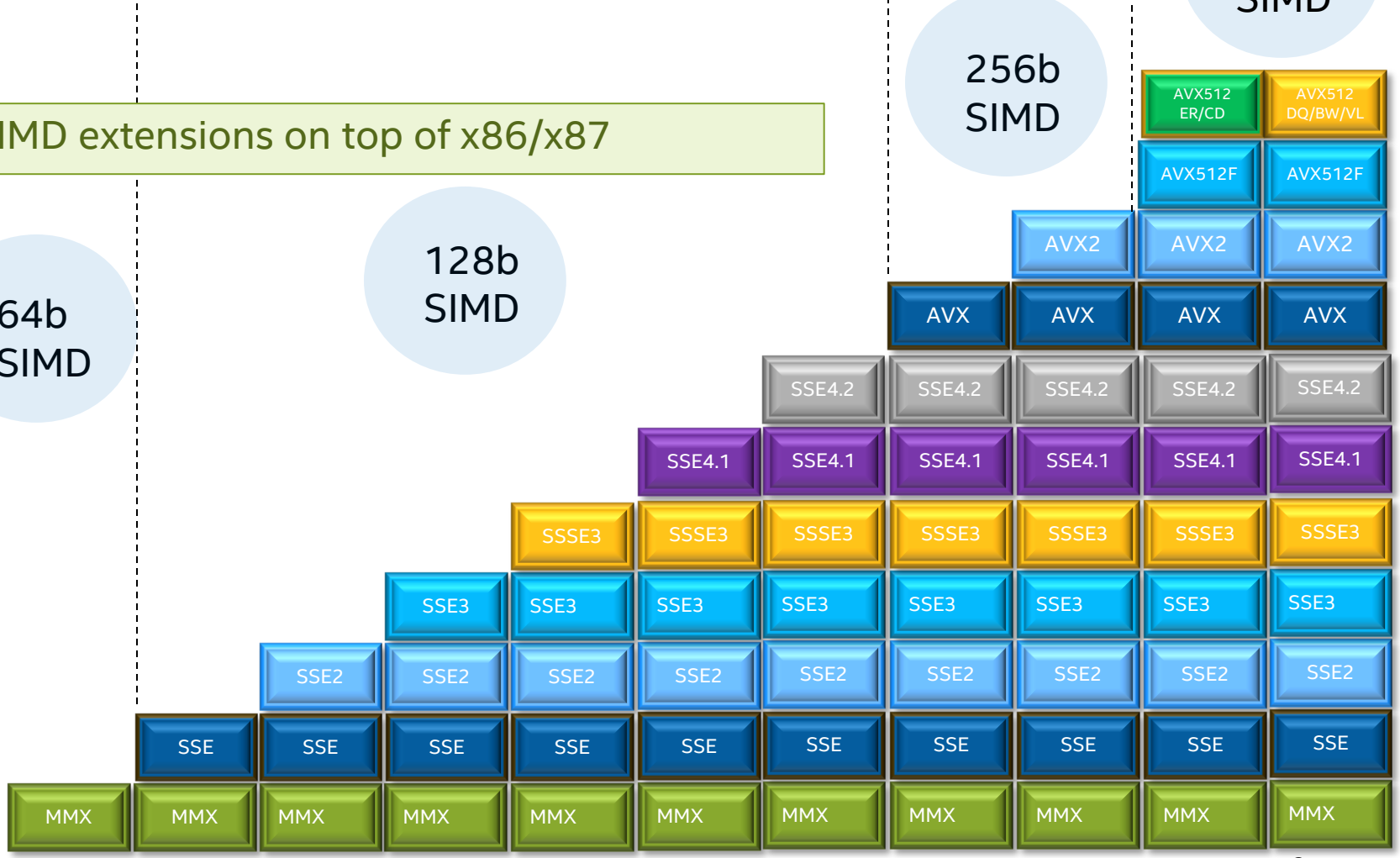
SIMD extensions on top of x86/x87

64b SIMD

128b SIMD

256b SIMD

512b SIMD



New Features Supporting Math Functions in AVX512F, AVX512DQ, and AVX512ER

Static Rounding and Exception Suppression Controls

Reciprocal/Reciprocal Square Root Approximations, 14-Bit: VRCP14, VRSQRT14

Reciprocal/Reciprocal Square Root Approximations, 28-Bit: VRCP28, VRSQRT28

Exponential Approximations: VEXP2

New Permute Instructions: VPERM, VPERMI2

Instructions to Extract Exponent, Mantissa Fields: VGETEXP, VGETMANT

Instruction to Scale FP values by Power of 2: VSCALEF

Instruction to Round to Given Number of Fraction Bits: VRNDSCALE

Instruction to Extract Reduced Argument: VREDUCE

Instruction to Test FP Input Type: VFPCLASS

Instruction to Fix Up Special FP Values: VFIXUPIMM

Range Restriction Calculation Instruction: VRANGE

Where To Use These New Instructions?

Newton-Raphson Iterations for inverse, division, and square root (they are self-correcting , and have a fast convergence rate)

- Start with an initial approximation of the inverse r (or inverse of the square root)
- Iterate using FMA (might need extended precision or extra exponent bits or range reduction to avoid potential underflow/overflow/loss of precision):

  $e = (1 - x \times r)_{rn}$

  $r' = (r + e \times r)_{rn}$
- Stop when the error requirement is met

Table Driven Algorithms

- From the input argument x, deduce variable y belonging to a much smaller domain, such that f(x) can easily be calculated from f(y)
- For example, to compute exponential functions, decompose

  $x = x' + (x-x') = round(x') + (x' - round(x')) + (x-x')$

  where x' is x rounded to a given number of fraction bits and

  $y = (x-x')$
- Calculate f(y) with a low-degree polynomial
- Reconstruct f(x) from f(y); this may involve a table look-up

Static Rounding and Exception Suppression Controls

Static (per instruction) rounding: a rounding attribute in the EVEX instruction prefix can override the MXCSR rounding mode

Static rounding also implies exception suppression (SAE) – they are linked together in the instruction encoding; behavior: as if all FP exceptions are disabled, and no status flags are set

Static rounding enables better accuracy control in intermediate steps

- E.g. the RN mode is used in intermediate steps for SW division and SW square root for extra precision; the default MXCSR rounding mode is used in the last step
- The LSB matters e.g. in range reduction for trigonometric functions and other cases

The SAE mode is useful when correct IEEE FP status flag settings are needed

- Special cases can execute silently in the main path
- The precision flag can be set correctly in SW sequences such as division, square root

New Reciprocal/Reciprocal Square Root Approximation Instructions

VRCP14PS/PD, VRSQRT14PS/PD

- 14-bit accuracy – e.g. eliminates one NR iteration for double precision (DP) calculations
- DP instruction is available - no more conversions needed between SP and DP

512-bit SW division and square root implementations can have up to 2X better throughput than the HW instructions (but HW instructions have better latency)

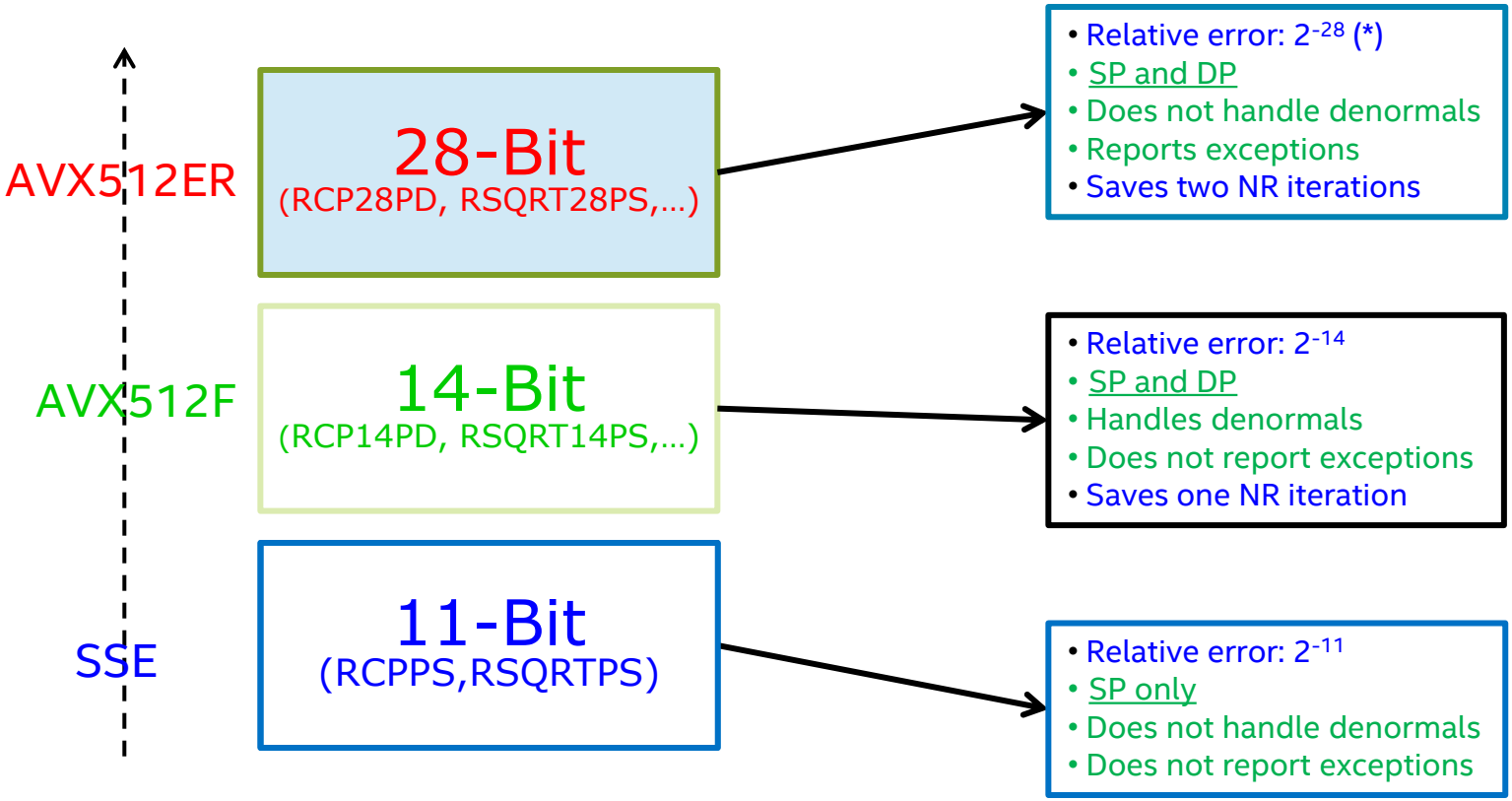
Short RCP/RSQRT approximations can be used also in many transcendental function implementations

- E.g. in logarithm and inverse trigonometric function implementations

VRCP28PS/PD, VRSQRT28PS/PD

VEXP2PS/PD

Evolution of Reciprocal Approximations



(*) The Binary32 format precision is of only 24 bits (less than 28). After rounding, the final error ends up in the range of ~ $2^{-23.4xxxx}$, very close to being IEEE correctly-rounded

New Permute Instructions

Permute instructions can be used in place of vector gather, e.g. for small lookup tables

VPERMI2PS/PD, VPERMT2PS/PD

- VPERMI2PS zmm1, zmm2, zmm3
  - Permute single-precision FP values from two tables in zmm3 and zmm2 using the indices in zmm1 and store the result in zmm1
- VPERMT2PS zmm1, zmm2, zmm3 (overwrites one table)
  - Permute single-precision FP values from two tables in zmm3 and zmm1 using indices in zmm2 and store the result in zmm1
- Useful for 32-element lookup (SP), or 16-element lookup (DP)

VPERMPS/PD

- VPERMPS zmm1, zmm2, zmm3
  - Permute single-precision floating-point values in zmm3 using indices in zmm2 and store the result in zmm1
- Useful for16-element lookup (SP), 8-element lookup (DP)

Compared to designs using vector gather for table lookup, implementations using smaller lookup tables (permutes) and somewhat longer polynomials can achieve ~2X better throughput

Extract Exponent, Mantissa Fields

VGETEXPPS/PD dst, src

- VGETEXPPD zmm1, zmm2{sae}
  - Convert the exponent of the packed double-precision floating-point values in the source operand to DP FP results representing the unbiased integer exponents, and store the results in the destination
- dst = floor($\log_2$(|src|)

VGETMANTPS/PD dst, src, imm8

- src mantissa is normalized to one of 4 possible ranges: [1,2), [1/2,2), [1/2,1), or [3/4,3/2); the range is specified by imm8[1:0]
- imm8[3:2] is used to control the output sign (same sign, or positive), and optionally to trigger the Invalid exception if src<0 (useful e.g. for log, sqrt)

Facilitates efficient, branch-free implementations, because denormals and other special arguments are processed on the main path, at no extra cost

Example: Logarithm Function

If $x = 2^N \times mx$, where mx is the mantissa (significand), then

$\log_2(x) = N + \log_2(mx) = N + \log_2(1+r)$

$\log(x) = N \times \log(2.0) + \log(mx) = N \times \log(2.0) + \log(1+r)$

N and mx are computed using VGETEXP and VGETMANT, with

$r = mx - 1$

$\log_2(1+r)$ can be approximated with a piecewise polynomial (without a constant term)

Scale by a Power of 2

VSCALEFPS/PD dst, src1, src2

- VSCALEFPS zmm1, zmm2, zmm3
  - Scale the packed single/double precision floating-point values in zmm2 (src1) using floating-point values from zmm3 (src2)
- dst = src1 × $2^{floor(src2)}$
- Accepts all FP operands and covers special cases (Inf, NaN), eliminating the need for other fix-up steps

Can be used e.g. to improve the exp function throughput: no need to extract the exponent k separately and to form 2^K

- Replace the final scaling in the exp implementation with VSCALEF
- Facilitates branch-free implementations (for up to 1 ulp accuracy level, including gradual underflow results)
- No effect on exp latency

VSCALEF can also be used e.g. in a branch-free implementation of cbrt, or in SW division and square root

Example: SW Division

Branch-free implementation of division:

$x = 2^{VGETEXP(x)} * VGETMANT(x, 0)$, where VGETMANT(x,0) normalizes x to [1,2), and transfers the sign of src to dst

a/b = VSCALEF(VGETMANT(a,0) / VGETMANT(b,0), VGETEXP(a) - VGETEXP(b))

where VGETMANT(a,0) / VGETMANT(b,0) can be calculated with a NR-iteration, without possible overflow, underflow, or loss of precision in the intermediate steps

Special cases (0, Inf, NAN) are also handled by VSCALEF

Round to a Given Number of Fraction Bits

VRNDSCALEPS/PD dst, src, imm8

- VRNDSCALEPD zmm1, zmm2{sae}, imm8
  - Rounds the packed double-precision FP values in zmm2 to a number of fraction bits specified by the imm8 field. Stores the result in zmm1
- ROUNDSCALE(x) = 2^{-M}*Round_to_INT(x*2^{M}, round_ctrl),

  where round_ctrl = imm8[3:0] & M=imm8[7:4] - an integer M in 0 …15
- round_ctrl = imm8[3:0] are the same as for VROUND:
  - bit 3 – Suppress Precision Exception
  - bit 2 chooses the rounding control from either MXCSR or imm8
  - bits 1,0 are the rounding control override (RN, RD, RU, or RZ)
- dst = 2^{-M}* VROUND(src × 2^{M}, imm8[3:0])
- There is no overflow/underflow

Can help in reducing the latency of some transcendental function implementations, e.g. in the exp2 argument reduction step

Can also help eliminate a MUL from some sequences

Extract the Reduced Argument

VREDUCEPS/PD dst, src, imm8

- VREDUCEPS zmm1, zmm2{sae}, imm8
  - Subtracts the integer part and the leading M fractional bits from the binary FP source value, where M is an unsigned integer specified by imm8[7:4]; i.e. it performs a reduction transformation on the packed single precision floating-point values in zmm2. Stores the result in zmm1 register
- dst = src – VRNDSCALE(src, imm8) = src – (ROUND(2^M*src))*2^{-M}
  - ROUND() treats src, 2^M, and their product as binary FP numbers with normalized significand and biased exponents
  - If src= 2^p*m, where m is the normalized significand and p is the unbiased exponent, then for RN 0<=|dst|<=2^{p-M-1}, and for other rounding modes 0<=|dst|<=2^{p-M}

Helps reduce computation latency

Usage examples: exp2, atan

Test Floating-Point Input Type

VFPCLASSPS/PD mask_dst, src, imm8

- Set mask if input is in the specified combination of these classes: QNaN, Neg, Denorm, -Inf, +Inf, -0, +0, SNaN
- One imm8 bit is reserved for each input class listed
- Set the imm8 bits for the input classes to be tested

Helps filter out special cases for branching, or for setting special results (under mask)

- vector masks make it easier to treat special inputs in a branch-free manner, even without a FIXUP instruction

Fix Up Special Values

VFIXUPIMMPD/PS dst, src, tbl32, imm8

- tbl32 is a 32-bit table storing the desired response for each of 8 possible input types: QNaN, SNaN, Zero, One, -Inf, +Inf, Negative, Positive (not 0 or 1)
- There are 16 possible responses (including that of leaving the destination register unmodified)
- The type of the input is checked, and then tbl32 is accessed to determine the output, based on the input type
- Invalid or Divide-by-Zero exceptions are raised for certain input types, if specified by the imm8 value

Can be used at the end of a computation, to set special results according to standard specification (e.g. IEEE, DX10)

Range Calculation

VRANGEPS/PD dst, src1, src2, imm8

- Computes min, max, minabs, or maxabs values according to the IEEE standard 754-2008
- imm8[1:0] selects one of four functions
- imm8[3:2] used to select desired sign of the result

Can be used e.g. at the beginning of a Fast2Sum algorithm

(need a > b)

$s = (a + b)_{rn}$

$z = (s - a)_{rn}$

$t = (b - z)_{rn}$

References

- https://software.intel.com/sites/default/files/managed/0d/53/319433-022.pdf
- http://www.intel.com/products/processor/manuals/

Legal Disclaimer & Optimization Notice

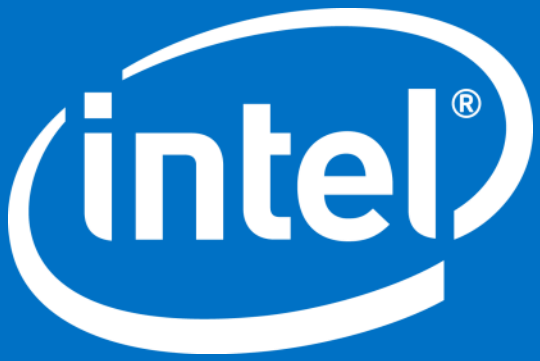
INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804


intel®