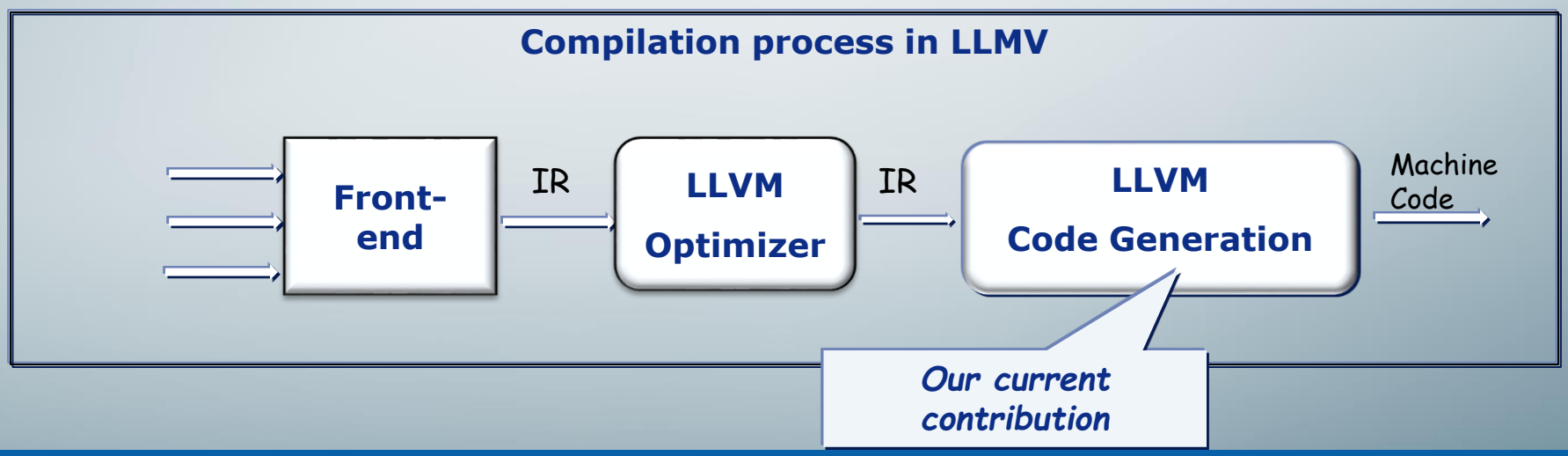
Intel® AVX-512 and its support in LLVM

Elena Demikhovsky
Intel® Software and Services Group
Israel

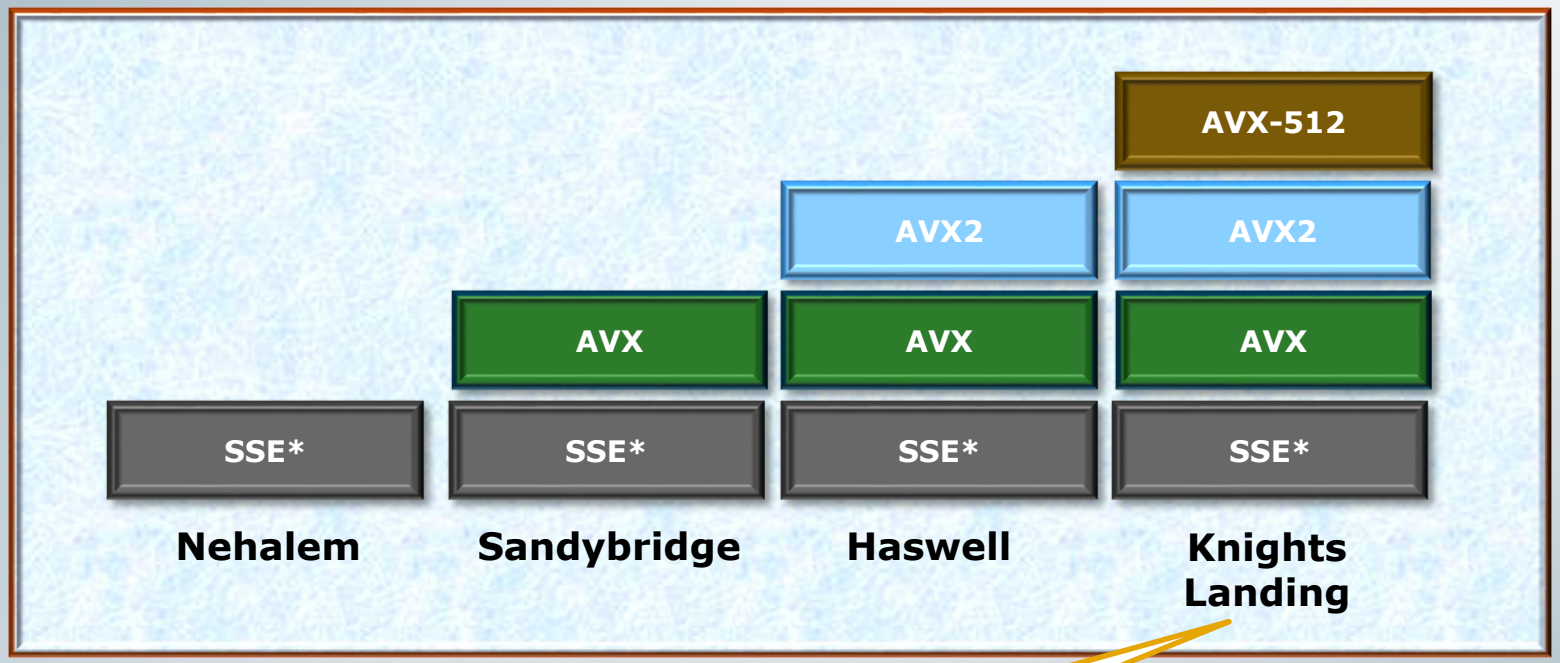
November, 2013

LLVM

- LLVM is a collection of modular and reusable compiler and toolchain technologies.
  - Open Source Project with many contributors
  - Supports many targets, including multiple generations of Intel® processors
- Our OpenCL CPU backend is based on LLVM
- We started adding AVX-512 ISA to LLVM since July 2013



Intel® AVX-512 – KNL processor

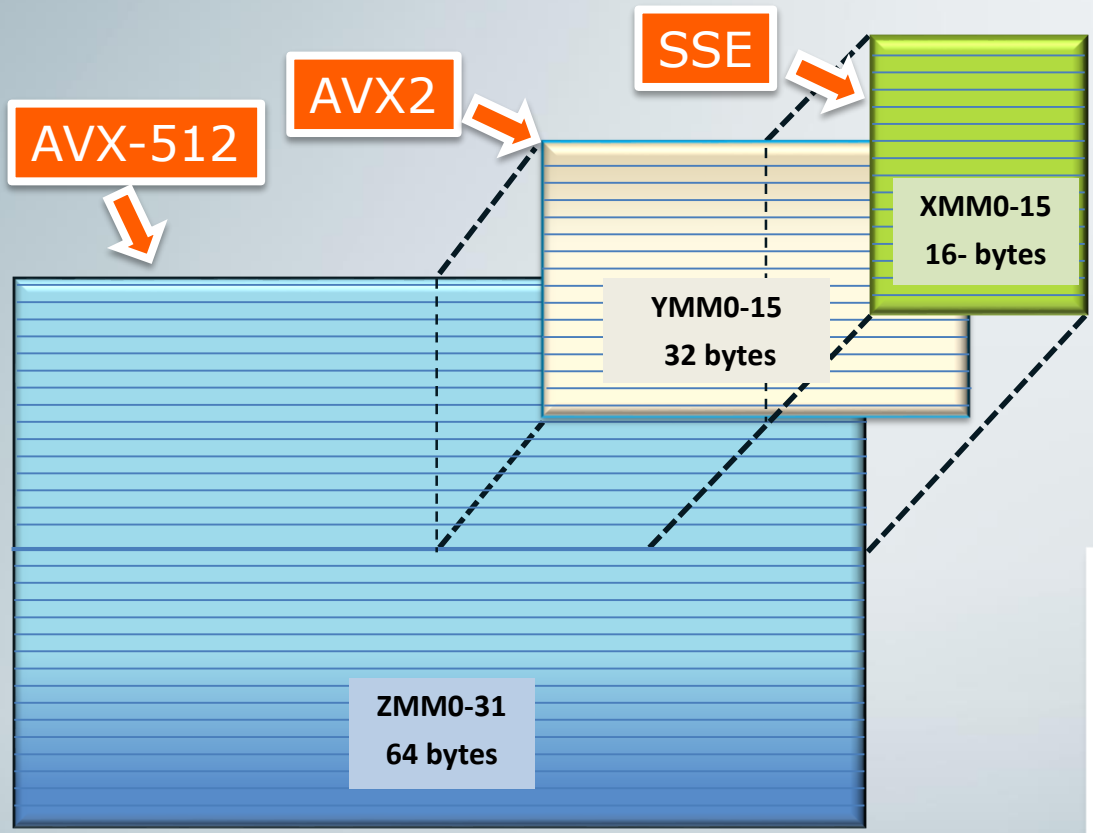


KNL – 2nd generation MIC architecture product

Intel® AVX-512 – Comprehensive vector extension for HPC

- ***Parallel computation***
  - ***More and wider SIMD registers***
  - ***Masking***
  - ***Gather and Scatter***
  - ***Compress and Expand***
  - ***Conflict Detection***
- ***Do more in one instruction***
  - ***FP rounding mode per instruction***
  - ***Embedded Broadcast***
  - ***2-source vector shuffles***
- ***Math support***
  - ***New math instructions***

Greatly increased register file



32 x 512 bit registers

Higher throughput

Greatly improved unrolling and inlining opportunities

Wider data vector

AVX2

```
float A[N], B[N], C[N]

for(i=0; i<8; i++)
{
      C[i] = A[i] + B[i];
}
```

```
VADDPS YMM0, YMM1, YMM2
```

16 x 256-bit registers
In each register:
8 float or 4 double
8 integer or 4 long

AVX-512

```
float A[N], B[N], C[

for(i=0; i<16; i++)
{
      C[i] = A[i] + B[i];
}
```

```
VADDPS ZMM0, ZMM1, ZMM2
```

32 x 512-bit registers
In each register:
16 float or 8 double
16 integer or 8 long

Supported in LLVM:

- AVX-512 instruction set
- New wide registers
- New data types
- Instruction selection algorithm

Masking

- 8 new 64-bits mask registers k0-k7

CMPPS k1, zmm21, zmm31

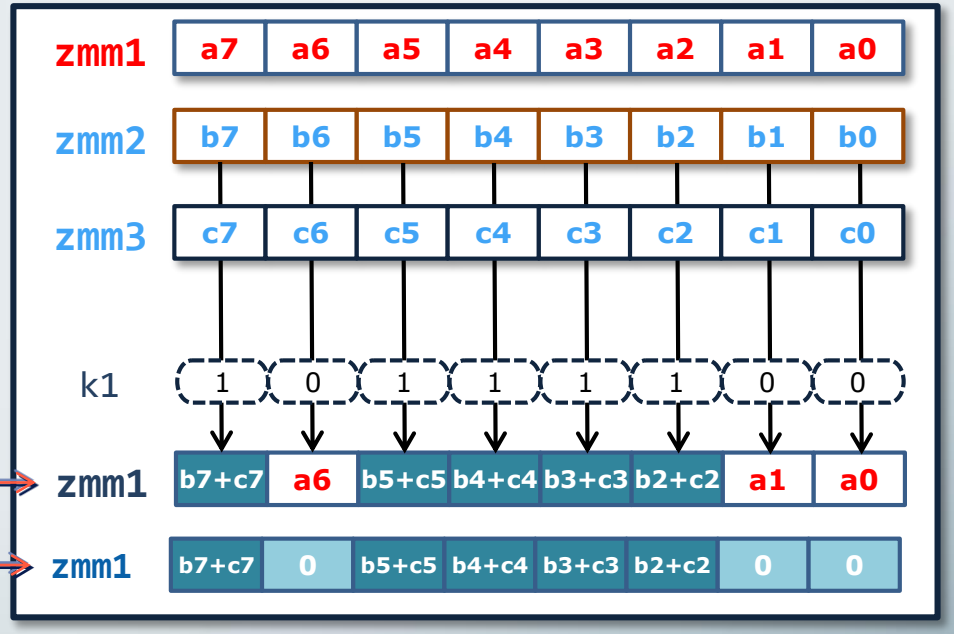
k1 = …0101100111

Unmasked elements remain unchanged:

VADDPD **zmm1** {k1}, zmm2, zmm3

Or zeroed:

VADDPD zmm1 {k1} {z}, zmm2, zmm3



Why masking?

- Memory fault suppression
  - if-conditional statements or loop remainders
- Avoid FP exceptions
- Zeroing/merging
  - Use zeroing to avoid false dependencies
  - Use merging to avoid extra blends

Code Example

```
float A[N], B[N], C[N];

for(i=0; i<16; i++)
{
  if (B[i] != 0)
    A[i] = A[i] / B[i];
  else
    A[i] = A[i] * C[i];
}
```

```
VMOVUPS zmm2, A[16]
VCMPPS k1, zmm0, B
VDIVPS  zmm1 {k1}{z}, zmm2, B
KNOT k2, k1
VMULPS  zmm1 {k2}, zmm2, C
VMOVUPS A[16], zmm1
```

Supported in LLVM:

- Mask registers and data types
- Generation and encoding of masked instructions

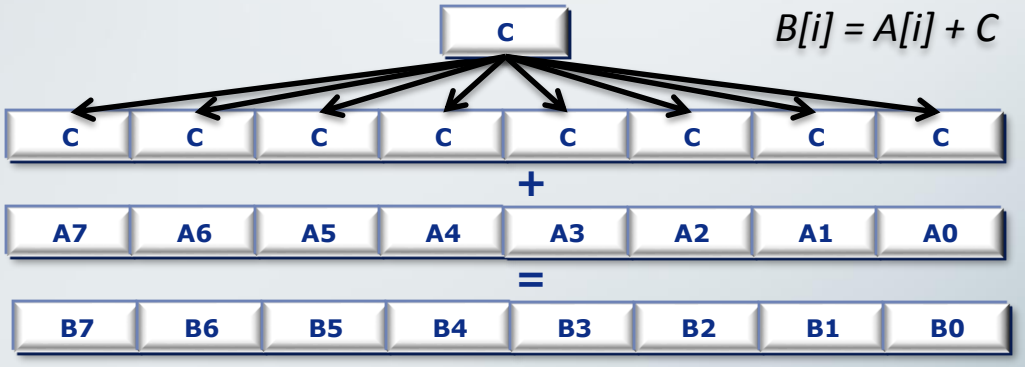
Code with masks is not generated by LLVM yet

Embedded Broadcast

Broadcast one scalar from memory into all vector elements

```
long A[N], B[N], C

for(i=0; i<8; i++)
{
  B[i] = A[i] + C;
}
```




```
vpbroadcastq zmm3, [rax]
vpaddq zmm1, zmm2, zmm3
```

```
vpaddq zmm1, zmm2, [rax] {1to8}
```

Supported in LLVM:

- Logic for folding broadcast into another operation

Compressed Store and Expanded Load

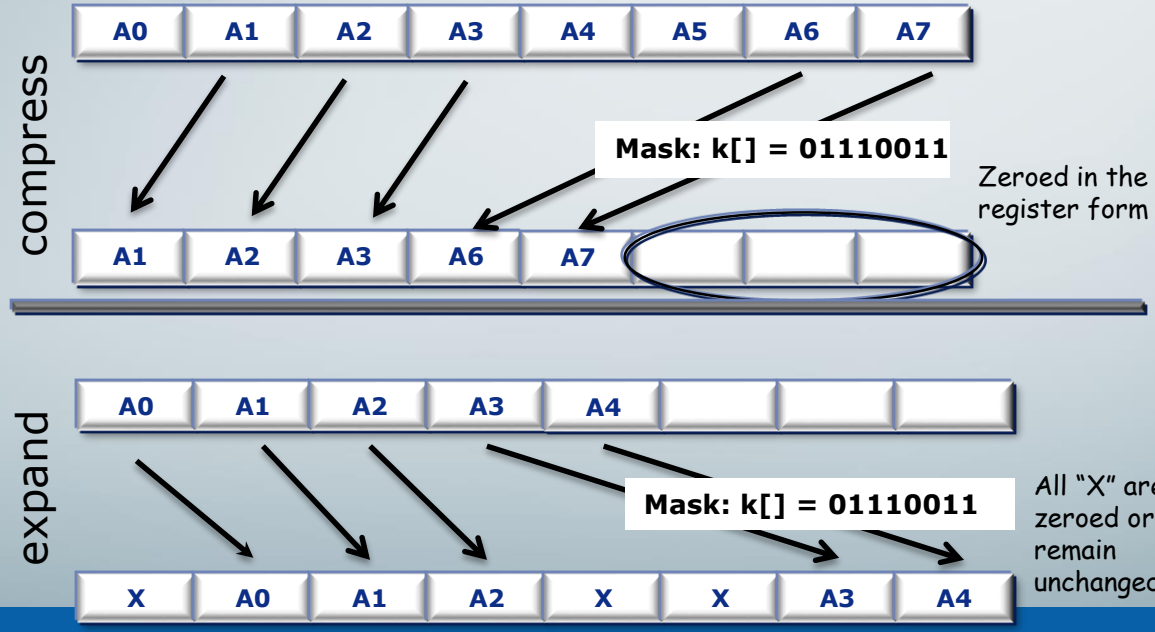
```
double A[N], B[N], C[N];

for(i=0; i<8; i++)
{
  if (B[i] != 0)
    *dst++ = A[i];
}
```

```
VMOVUPD zmm2, A[8]

VCMPPD k1, zmm0, B

VCOMPRESSPD [dst] {k1}, zmm2
```



LLVM:

- Compress/Expand algorithm is not supported yet

Conflict detection

Sparse computations are hard for vectorization

```
for(i=0; i<16; i++)
{
  j = B[i];
  A[j] = C[i] + D[i];
}
```

```
j = vload &B[i]          // Load 16 B[i]
val_A = C + D            // Compute new val
vscatter A, j, val_A     // Update A[j]
```

```
for(i=0; i<16; i++)
{
 j = B[i];
 A[j]++;
}
```

```
j = vload &B[i]          // Load 16 B[i]
val_A = vgather A, j     // Grab A[B[i]]
val_A++                  // Compute new val
vscatter A, j, val_A     // Update A[B[i]]
```

! Code above is wrong if any values within B[i] are duplicated

Conflict Detection – how does it work?

Conflict Free Code

```
for(i=0; i<16; i++)
{
  j = B[i];
  A[j]++;
}
```

```
j = vload &B[i]
pending_elts = 0xFFFF;
do {
  mask = conflict_free(j, pending_elts)
  val_A = vgather {mask} A, j            // Grab A[j]
  val_A++                                // Compute new values
  vscatter A {mask}, j, val_A            // Update A[j]
  pending_elts ^= mask                   // remove done idx
} while (pending_elts)
```

LLVM:

- Conflict detection is not supported yet - a task for a loop vectorizer

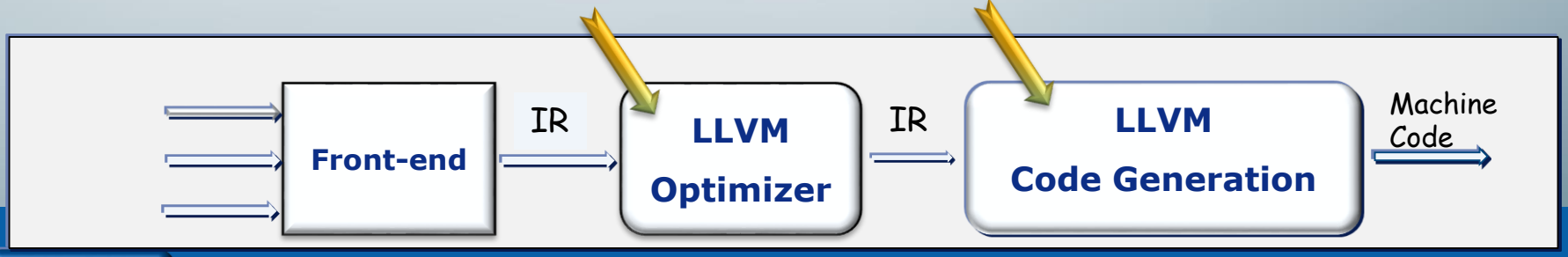
Summary

Currently available

- 99% of the new AVX-512 instructions
- New LLVM data types for wide vectors and masks
- EVEX encoding, instructions with masks and broadcast
- Instruction selection algorithm

Future opportunities – Performance

1. **In the LLVM optimizer**
   - Learn vectorizer to use the conflict detector
   - Use compress and expand capabilities
2. **In code generator**
   - Optimize instruction scheduling
   - Enhance instruction selection algorithm
   - Late machine code optimization phase - set masks to avoid extra blends.



Legal Disclaimer

- INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.
- A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.
- Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.
- The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.
- Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.
- Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm
- 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 SSE3 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.
-

Embedded Rounding Control & SAE (Suppress All Exceptions)

Static (per instruction) Rounding Mode .

Ignoring the value of RM bits in MXCSR.

```
VADDPS          zmm7 {k6}, zmm2, zmm4 {rd}
VCVTDQ2PS       zmm1, zmm2 {ru}
```

SAE is always implied by using embedded rounding mode

Why?

- Avoid MXCSR access – slow and cumbersome

- Set rounding mode per instruction
  - Simplifies development of high performance math software sequences