Hardware-Aware Optimization: Using Intel® Streaming SIMD Extensions

Dr. Thomas Willhalm

Sr. Application Engineer

Software and Services Group

Legal Disclaimer

Intel may make changes to specifications and product descriptions at any time, without notice.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, visit Intel Performance Benchmark Limitations

Intel does not control or audit the design or implementation of third party benchmarks or Web sites referenced in this document. Intel encourages all of its customers to visit the referenced Web sites or others where similar performance benchmarks are reported and confirm whether the referenced benchmarks are accurate and reflect performance of systems available for purchase.

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. See www.intel.com/products/processor_number for details.

Intel, processors, chipsets, and desktop boards 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.

Intel Virtualization Technology requires a computer system with a processor, chipset, BIOS, virtual machine monitor (VMM) and applications enabled for virtualization technology. Functionality, performance or other virtualization technology benefits will vary depending on hardware and software configurations. Virtualization technology-enabled BIOS and VMM applications are currently in development.

64-bit computing on Intel architecture requires a computer system with a processor, chipset, BIOS, operating system, device drivers and applications enabled for Intel® 64 architecture. Performance will vary depending on your hardware and software configurations. Consult with your system vendor for more information.

Lead-free: 45nm product is manufactured on a lead-free process. Lead is below 1000 PPM per EU RoHS directive (2002/95/EC, Annex A). Some EU RoHS exemptions for lead may apply to other components used in the product package.

Halogen-free: Applies only to halogenated flame retardants and PVC in components. Halogens are below 900 PPM bromine and 900 PPM chlorine.

Agenda

- Introduction to SIMD
- SIMD arithmetic
- Conditional Code with SIMD
- Search and String Operations
- Data layout for SIMD
- Using SIMD for Full-Table Scans
- Summary

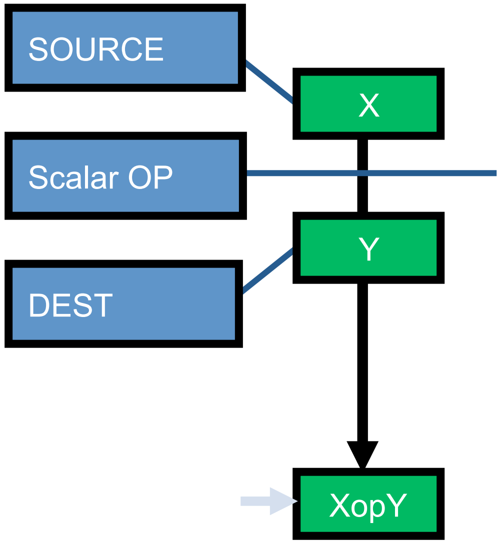
This presentation introduces only a subset of Intel® SSE with a strong focus on integer operations.

Single Instruction Multiple Data (SIMD)

Scalar processing

– traditional mode

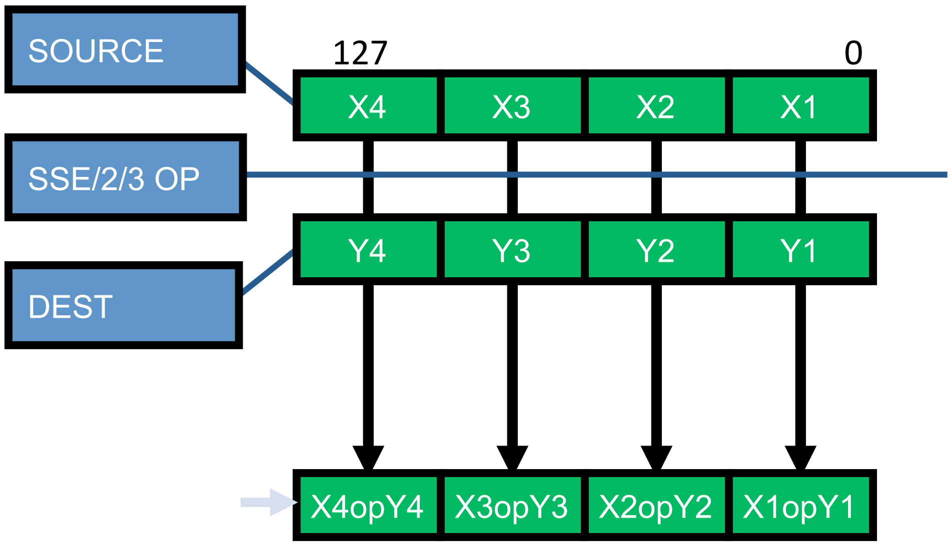
– one instruction produces one result



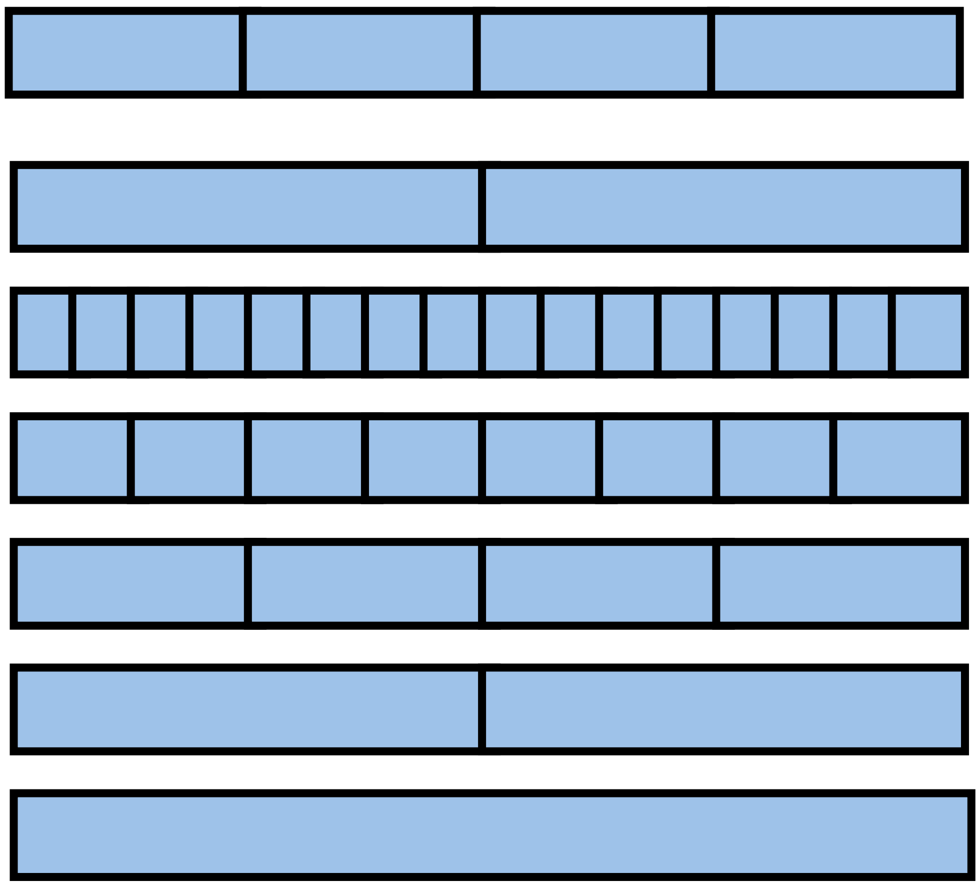
SIMD processing

– with Intel® SSE

– one instruction produces multiple results



SSE Data Types & Speedup Potential



4x floats

2x doubles

16x bytes

8x 16-bit shorts

4x 32-bit integers

2x 64-bit integers

1x 128-bit integer

Potential speed-up is roughly the amount of packing

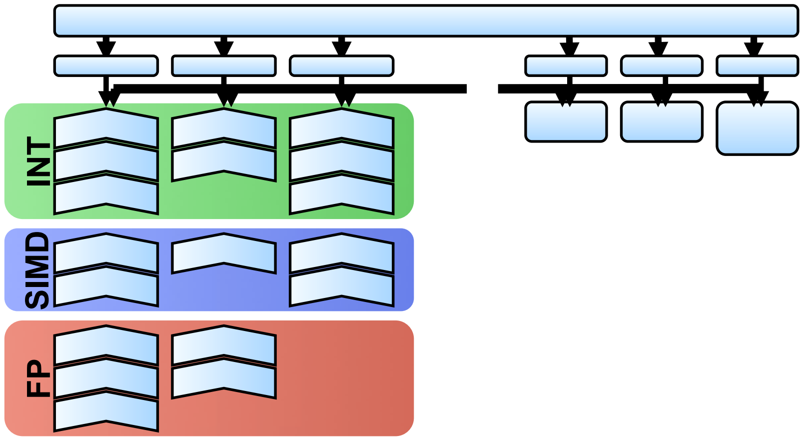
SIMD is orthogonal to Multi-Core and Instruction-Level Parallelism

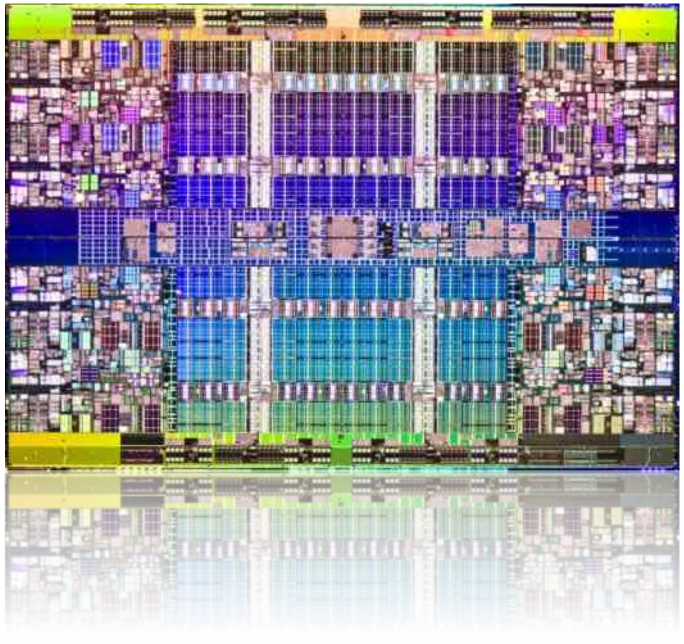
Available in all cores

Multiple Execution units

Allow SIMD parallelism

Up to 4 instructions can be retired in 1 clock cycle



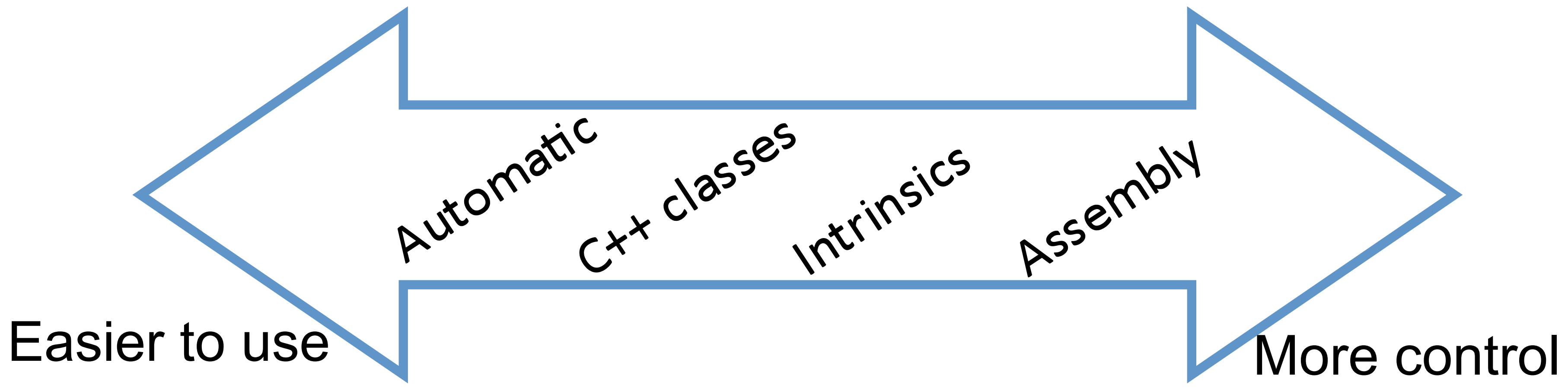


Evolution of SIMD on Intel® Architecture

- MMX™
  - 64 bit
- Intel® Streaming SIMD Extensions (Intel® SSE)
  - 128 bit
- Intel® Advanced Vector Extensions (Intel® AVX)
  - 256 bit
  - Announced for Sandy Bridge architecture
- Intel® Many Integrated Core Architecture
  - 512 bit

Four Vectorization Approaches

- Inline assembly language
- Intrinsics (see next slide)
- C++ class library
- Automatic vectorization by compiler



Intel® SSE instructions are accessed as C functions:

```
// Original version using standard C/C++
void half(int array[], int len) {
  for (int i = 0; i < len; i++) {
     array[i] = array[i] >> 1; // shift right by 1
  }
}
// Modified version using intrinsics
void halfIntrinsic(int array[], int len) {
  __m128i *array4 = (__m128i *) array;
  for (int i = 0; i < len/4; i++) {
     array4[i] = _mm_srai_epi32(array4[i], 1);
  }
}
```

Agenda

- Introduction to SIMD
- **SIMD arithmetic**
- Conditional Code with SIMD
- Search and String Operations
- Data layout for SIMD
- Using SIMD for Full-Table Scans
- Summary

Basic Operations

PADD{B,W,D,Q,SB,SW} xmm, xmm/m128
PSUB{B,W,D,Q,SB,SW} xmm, xmm/m128
PAVG{B,W} xmm, xmm/m128

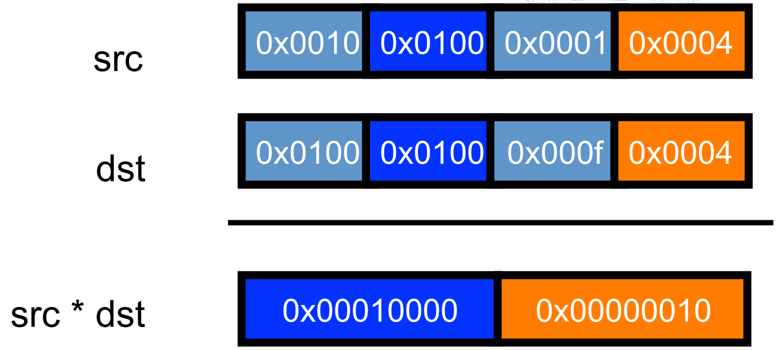
PAND xmm, xmm/m128
PANDNOT xmm, xmm/m128
POR xmm, xmm/m128
PXOR xmm, xmm/m128

P{MIN,MAX}{UB,SB,UW,SW,UD,SD} xmm, xmm/m128

SSE 2 SSE 4.1

Multiplication

PMULUDQ xmm, xmm/m128
PMULDQ xmm, xmm/m128

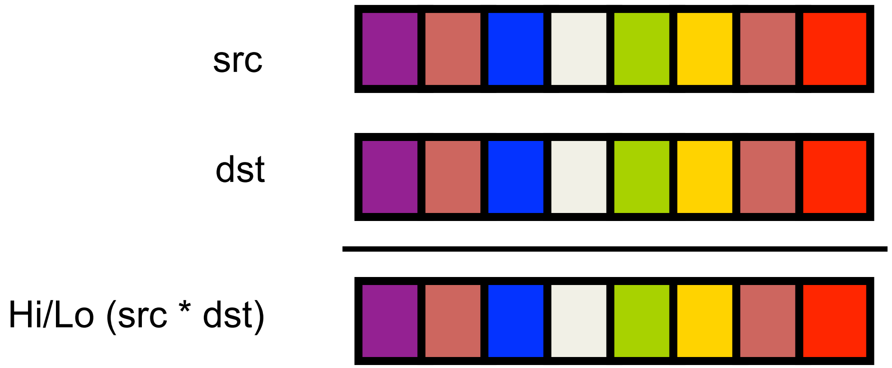


PMULH{W,SW} xmm, xmm/m128

PMULLW xmm, xmm/m128

PMULLD xmm, xmm/m128

Higher and lower 32bits of product



SSE 2 SSE 4.1

Shifting

PSLL{W,D,Q}   xmm, xmm/imm8
PSRA{W,D}   xmm, xmm/imm8
Shifting by bits

src	0x0110	0x0200	0x0010	0x0321
b	0x04			
src >> b	0x0011	0x0020	0x0001	0x0032

PSRL{W,D,Q,DQ}xmm, xmm/imm8
Shifting by bytes

src	0x0110	0x0200	0x0010	0x0321
b	0x01			
src >> b*8	0x0001	0x0002	0x0000	0x0003

SSE 2

Agenda

- Introduction to SIMD
- SIMD arithmetic
- **Conditional Code with SIMD**
- Search and String Operations
- Data layout for SIMD
- Using SIMD for Full-Table Scans
- Summary

Packed Comparison

PCMPEQ{B,W,D,Q} xmm, xmm/m128
PCMPGT{B,W,D} xmm, xmm/m128
PCMPGT{B,W,D}r xmm, xmm/m128

Ex: pcmpeqd xmm1, xmm2

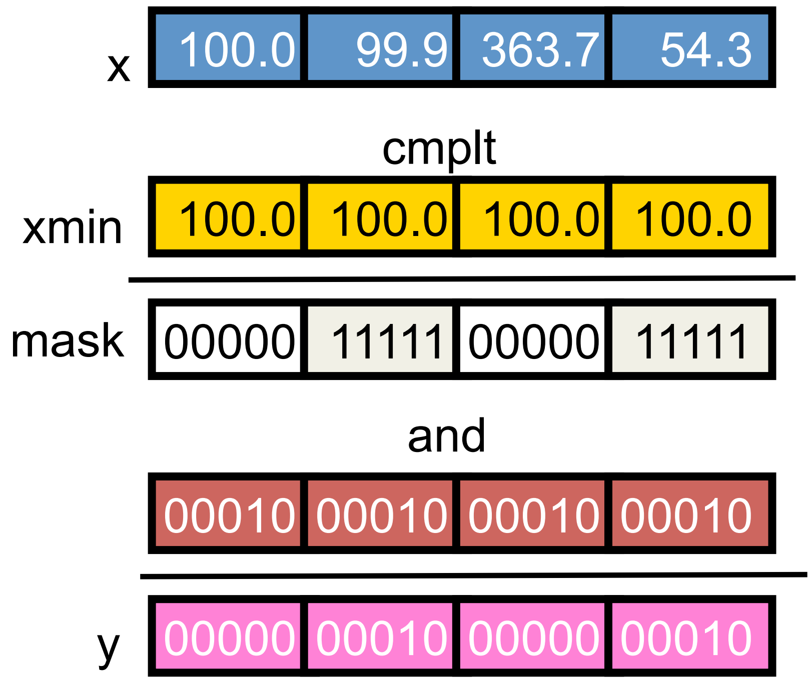
src	0x0010	0x0100	0x0001	0x0004
dst	0x0100	0x0100	0x0001	0x0321
src == dst	0x0000	0xffff	0xffff	0x0000

- Pair-wise compare equal, greater than, less than
- All the bits are set “1” if a pair is equal

SSE 2 SSE 4.1

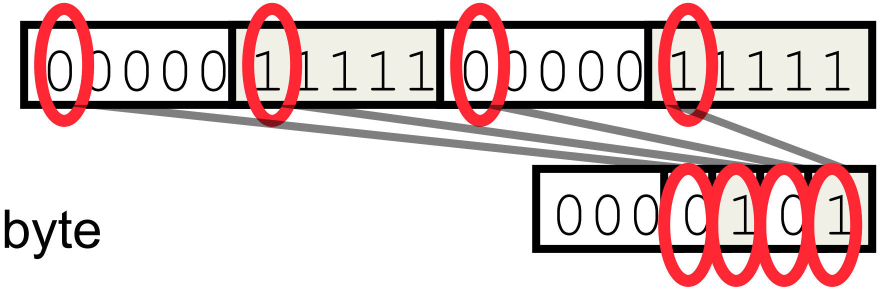
Use Masks for Conditional Expressions

```
// Original version using standard C/C++
if (x<xmin)
    y=10;
else
    y=0;
```



Convert Mask to Bit-Vectors

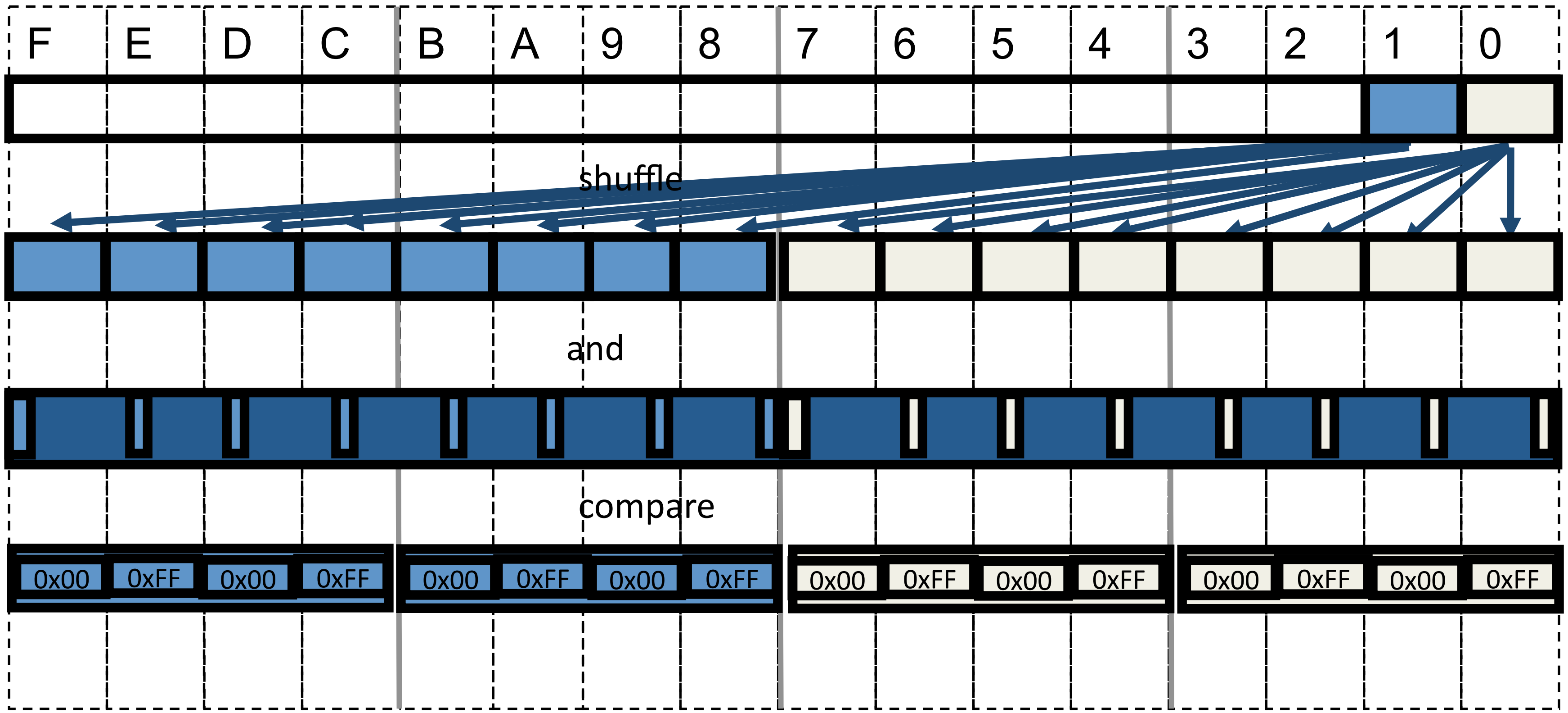
pmovmskb



- Extract highest bit of each byte
- Converts comparison result to bit-vector

Can be used to store the result of a search as bit-vector.

Convert Bit-Vector to Mask



Agenda

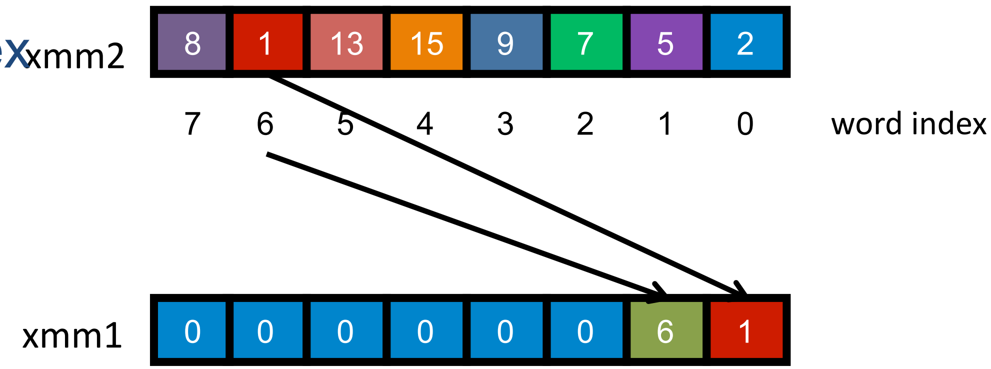
- Introduction to SIMD
- SIMD arithmetic
- Conditional Code with SIMD
- **Search and String Operations**
- Data layout for SIMD
- Using SIMD for Full-Table Scans
- Summary

Search Minimum

PHMINPOSUW xmm, xmm/m128

- Minimum of 8 words is put in the lowest word
- Index of the minimum word index is put in bit
- All the other bits are set 0

Ex: phminposuw xmm1, xmm2



SSE 4.1

String Operations

PCMPxSTRx: 8 and 16 bit, signed and unsigned, 0-terminated or fixed length

Equal *[i.e. strcmp()]*

- true for each character in Src2 if same position in Src1 is equal

```
Src1:   These_are_the_sa
Src2:   These are the sa
Mask:   1111101110111011
Index:0x00
```

Sub-string*[i.e. strspn()]*

- finds the start of a substring (Src1) within another string (Src2)

```
Src1:   sub
Src2:   busubusubusubusu
Mask:   0010001000100010
Index:0x02
```

Equal Any *[i.e. strcspn()]*

- true for each character in Src2 if any character in Src1 matches

```
Src1:   {}~|#\\@\b\t\0
Src2:   bad#passw@rd\0
Mask:   0001000001000000
Index:0x03
```

Ranges

- true if a character in Src2 is in at least one of up to 8 ranges in Src1

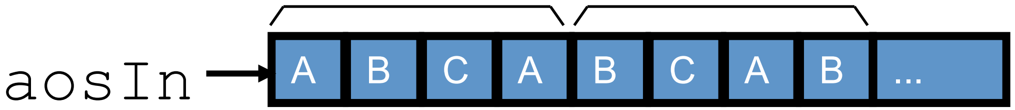
```
Src1:   AZaz09\0
Src2:   #AlphaNumeric#\0
Mask:   0111111111111000
Index:0x01
```

Agenda

- Introduction to SIMD
- SIMD arithmetic
- Conditional Code with SIMD
- Search and String Operations
- **Data layout for SIMD**
- Using SIMD for Full-Table Scans
- Summary

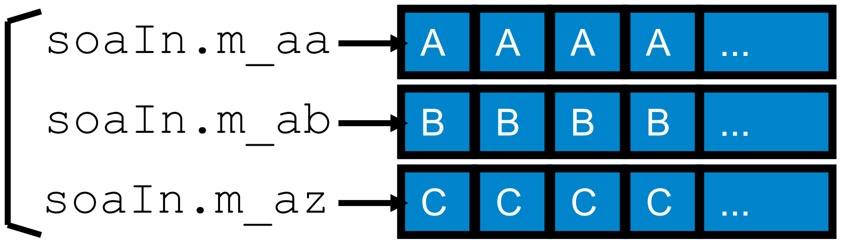
Row-Orientation is Unsuited for SIMD

AoS: Array of Structures "Row-oriented"



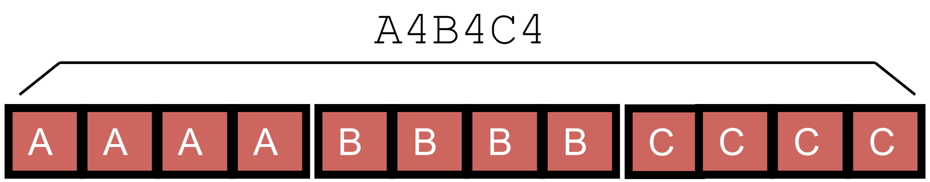
Row-otientation defeats SIMD

SoA: Structure of Arrays "Column-oriented"



Column-orientation is perfect for SIMD!

Hybrid Structure

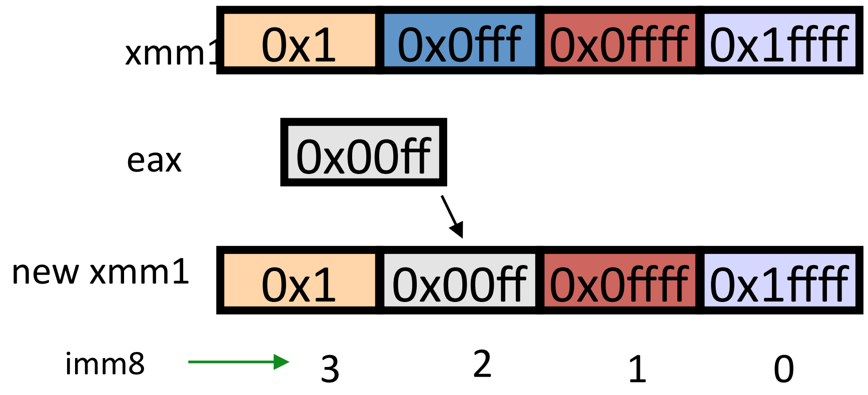


Insert and Extract Single Values

PINSRB xmm, r/m8, imm8
PINSRW xmm, r/m8, imm8
PINSRD xmm, r/m32, imm8
PINSRQ xmm, r/m64, imm8

- Insert a byte, a word, a dword, or a qword to the dst indicated by the offset in imm8

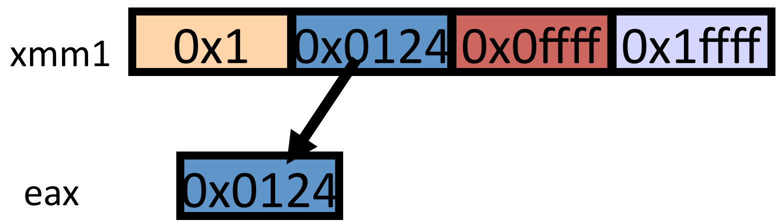
Ex: pinsrd xmm1, eax, 2



PEXTRB r32/r64/m8, xmm, imm8
PEXTRW r/m32/m64, xmm, imm8
PEXTRD r/m32, xmm, imm8
PEXTRQ r/m64, xmm, imm8

- Extract byte / word / dword / qword indicated by offset in imm8

Ex: pextrd eax, xmm, 3

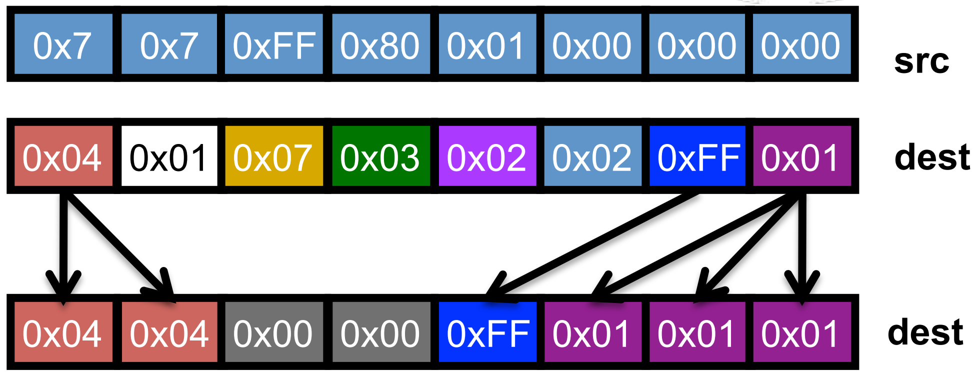


SSE 2 | SSE 4.1

Insert and extract serialize the code

Byte Permutation

PSHUFB mm, mm/m64
PSHUFB xmm, xmm/m128



- Byte-granularity permutation
- Variable control by source field
- Each byte of the source field selects the origin of the corresponding destination byte
- Also includes force-byte-to-zero flag (bit 7)

Byte permute is a very powerful operation for data preparation

SSSE 3

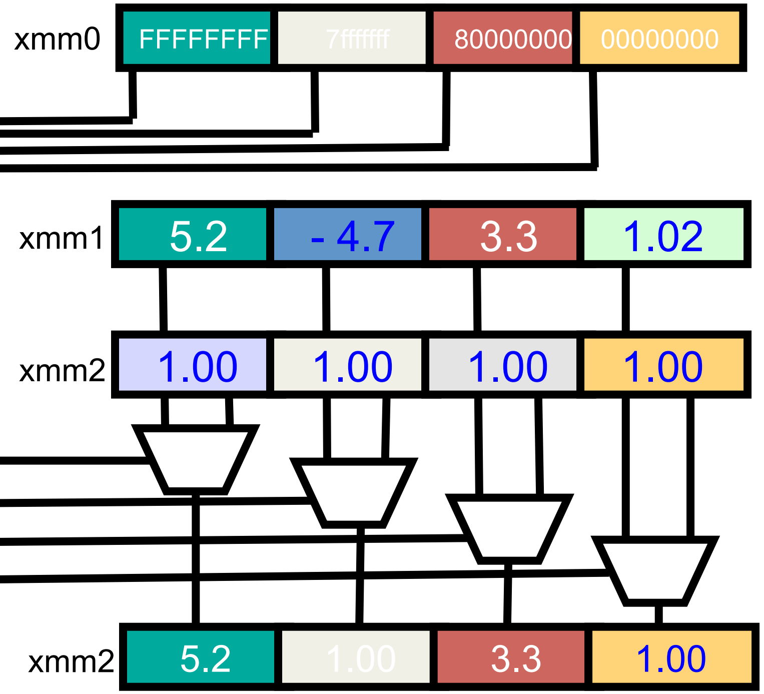
Blends

BLENDVP{B,S,D} xmm, xmm/m128 <xmm0>

BLENDVP{B,S,D} xmm, xmm/m128 imm8

- Based on implicit input xmm0 or imm8, copy fields from source to dest.
  - Selectively copy 4 sp FP
  - Selectively copy 2 dp FP
  - Selectively copy 16 bytes
- The control bit is the MSB in the corresponding fields in xmm0
- Copy if the corresponding MSB is 1

BLENDVPS xmm2, xmm1 <xmm0>



Blending allows merging results of 2 code paths

SSE 4.1

Agenda

- Introduction to SIMD
- SIMD arithmetic
- Conditional Code with SIMD
- Search and String Operations
- Data layout for SIMD
- **Using SIMD for Full-Table Scans**
- Summary

Recap: TREX Stores Values in Columns

Sales Table

ID	Amount
2	12.23
3	245.92
4	92.77
…	…

Tables for storing “Amount” attribute

Attribute Table

DocId	ValueId
2	42
3	100000
4	4711
5	128
6	31455

Dictionary

ValueId	Value
1	0.02
2	0.10
3	1.00
…	…
100000	245.92

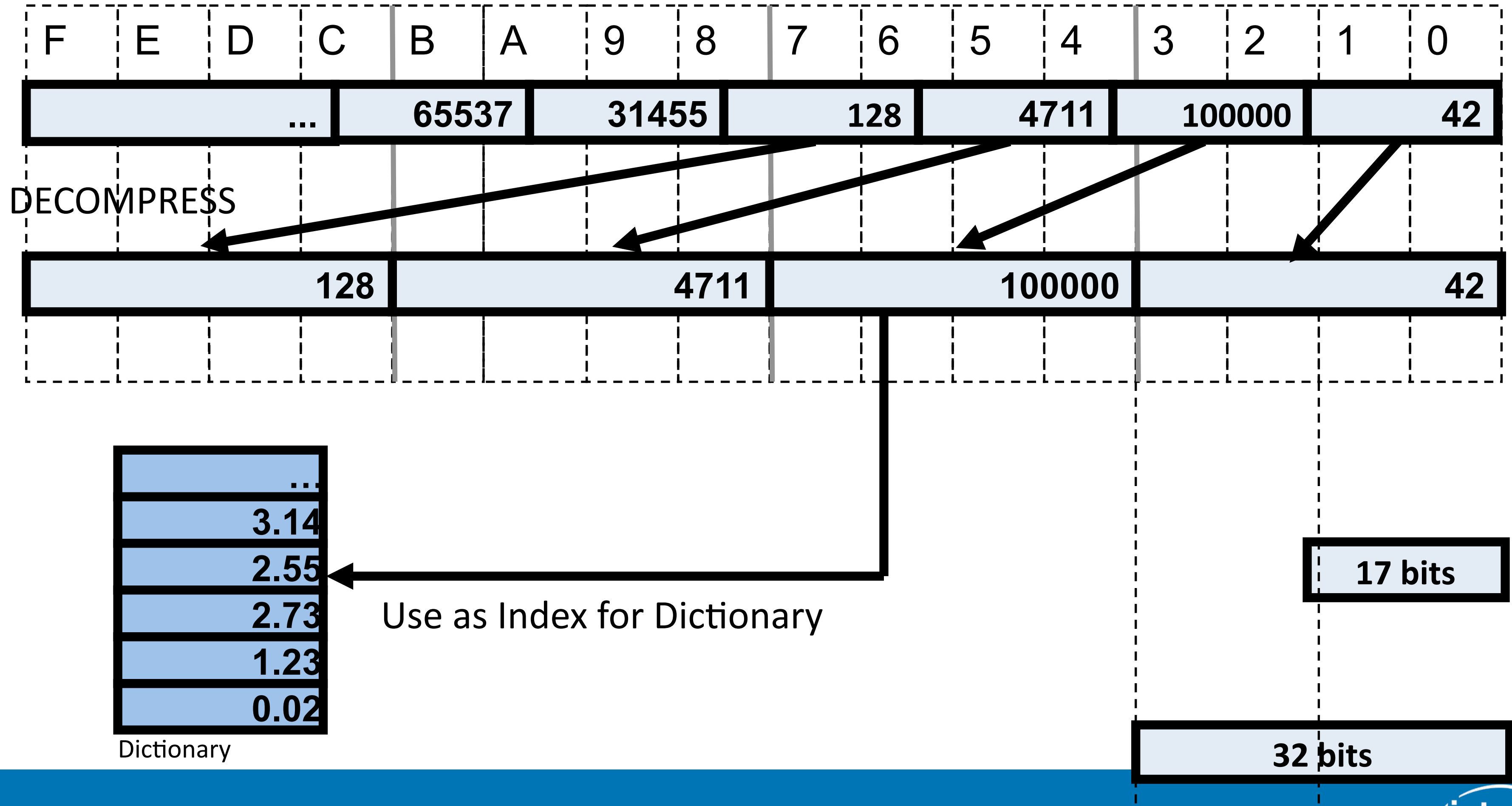
…

Our focus is this column

- From “Dictionary”, those values are {0, 1, 2 ,3, … ,100000}
- Max is 100000, which needs 17 bits to represent (2^{17}-1)
- Idea: instead of 32-bits, use 17-bits fields to store each ValueID
- Accessing “Value” needs decompression into 32-bits

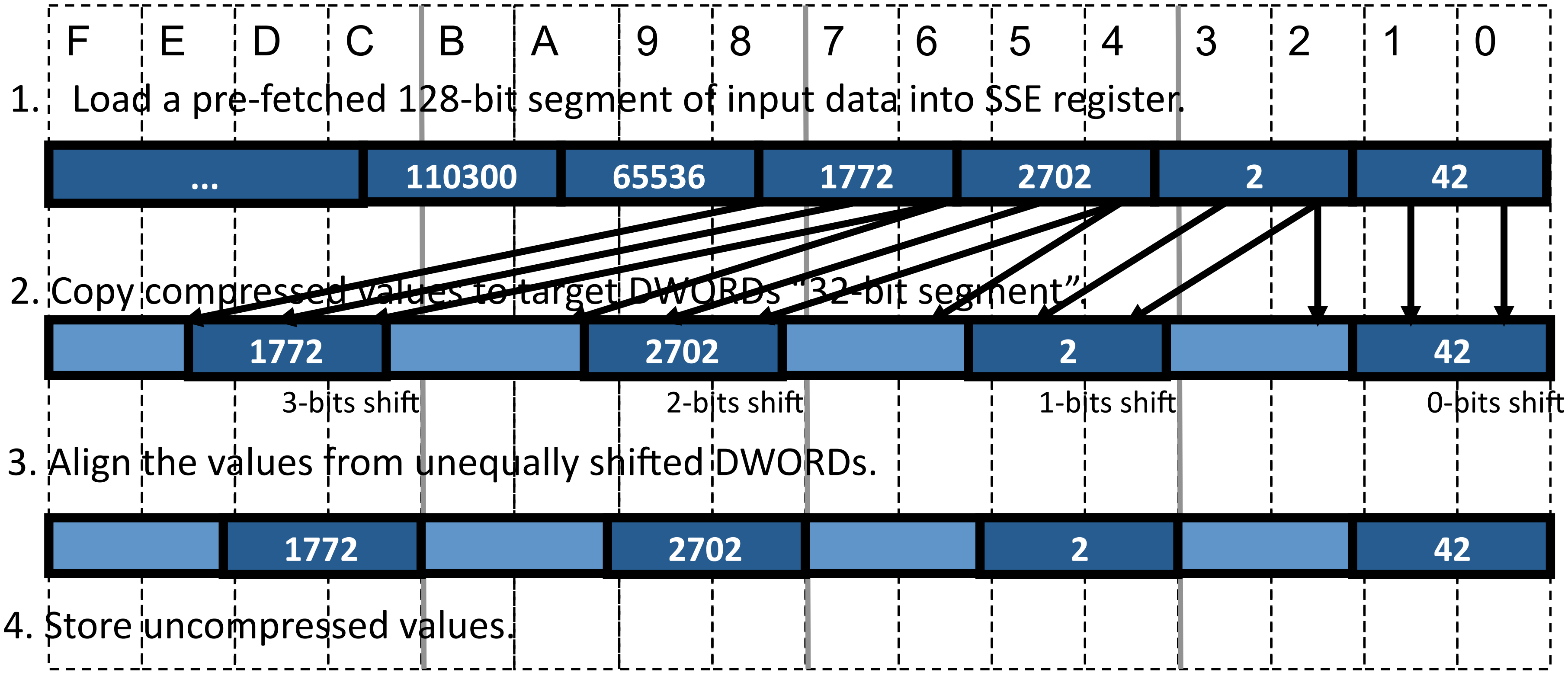
Integers are compressed as packed bit-fields

Example: packed 17-bit fields



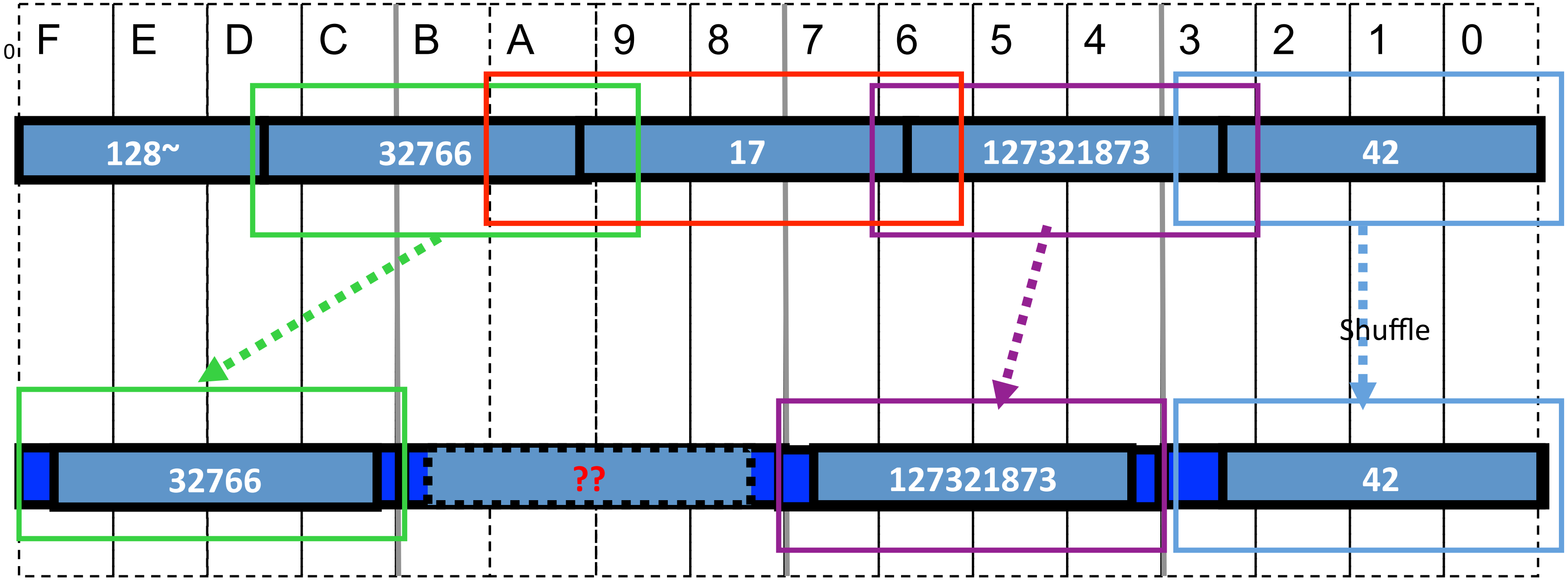
DECOMPRESS unaligned bit fields

Example: packed 17-bit fields



Problem: There are values that span across 5 Bytes

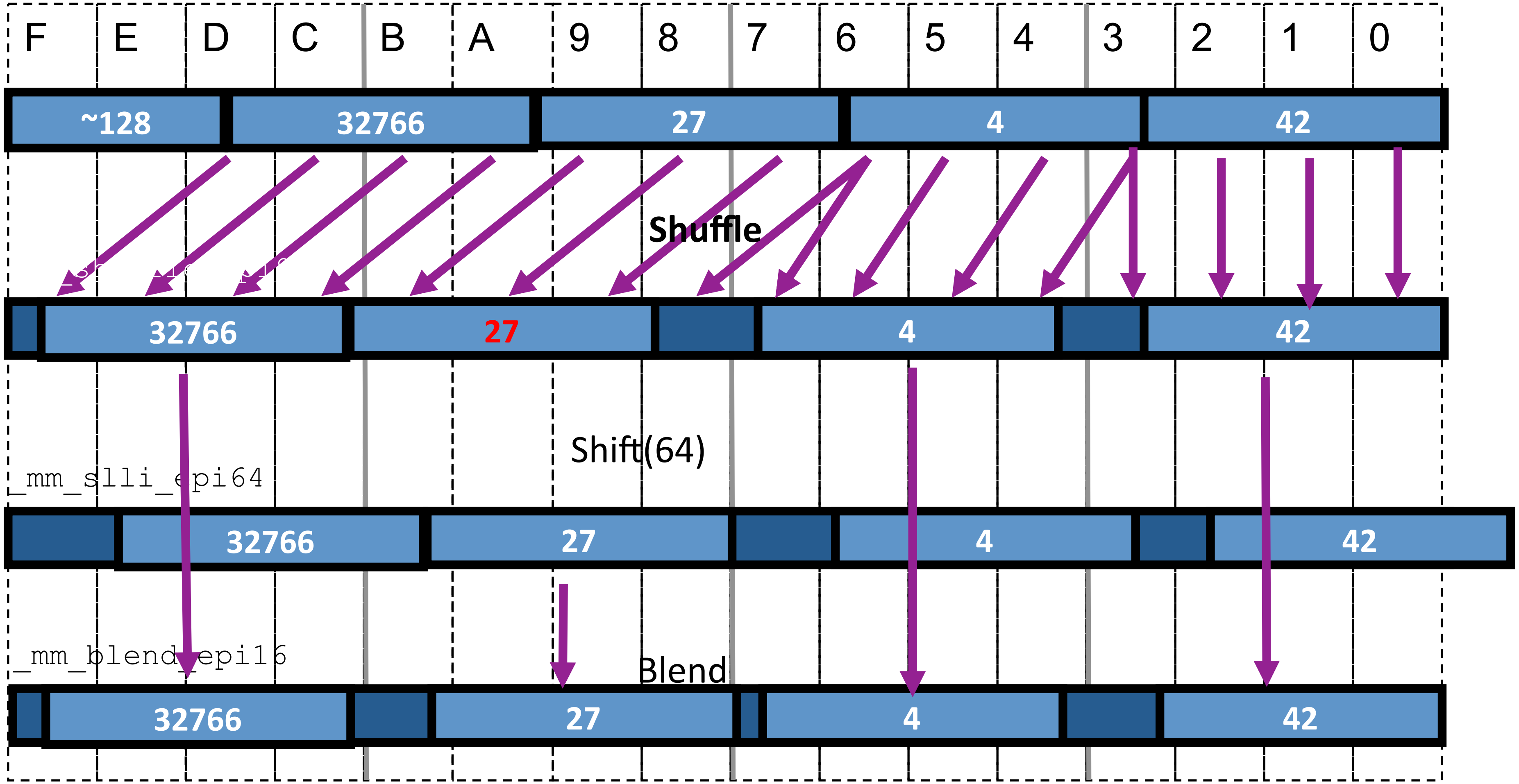
Example: packed 27-bit fields



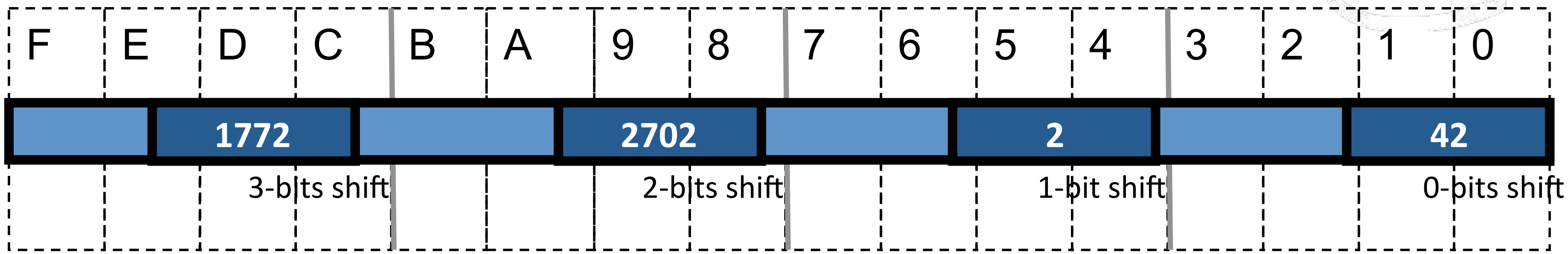
- The 3rd value spans across 5 Bytes.
- Cannot use Shuffle to copy the FULL bits into a 4-Byte space directly.

Solution: Shift 5-Bytes values into 4 Bytes and blend

Example: packed 27-bit field



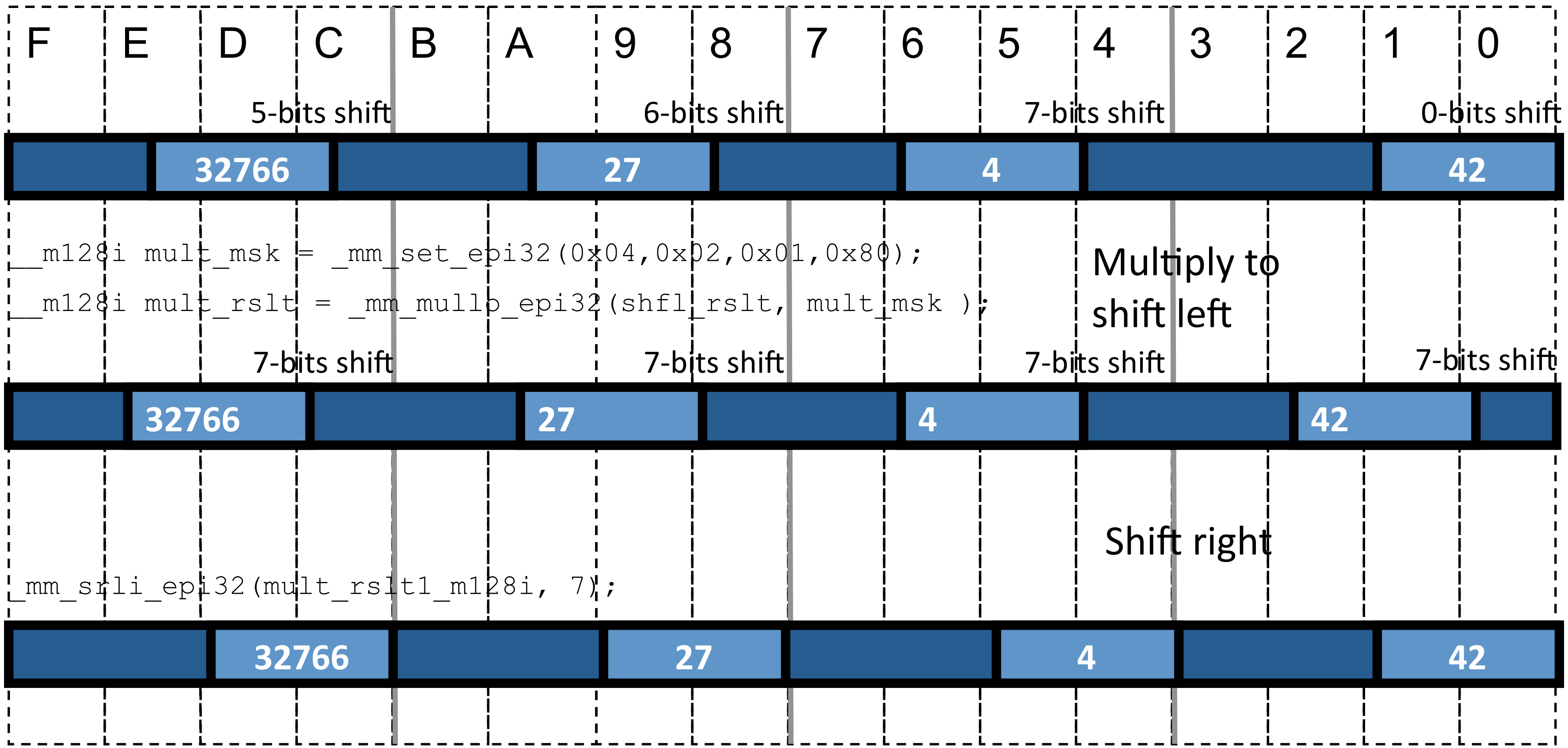
Example: Simulate Independent Shift



- Problem: Intel® SSE shift instruction shifts all values by the same shift amount
  - Quiz: How to compute n*16 efficiently?
  - Answer: use shift to multiply n<<4
- Solution: use multiplication to shift

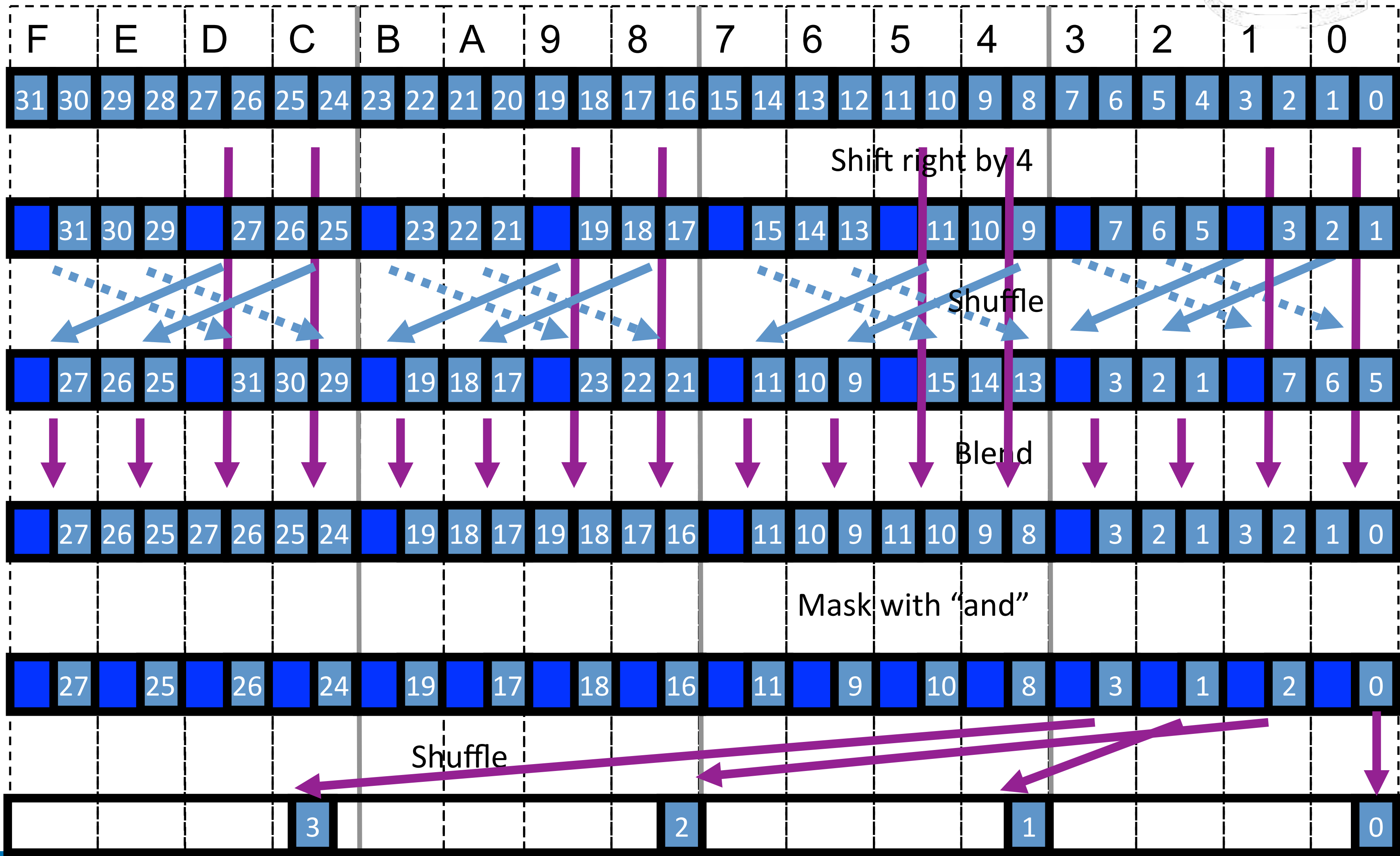
Example: Simulate Independent Shift

Example: packed 15-bit fields



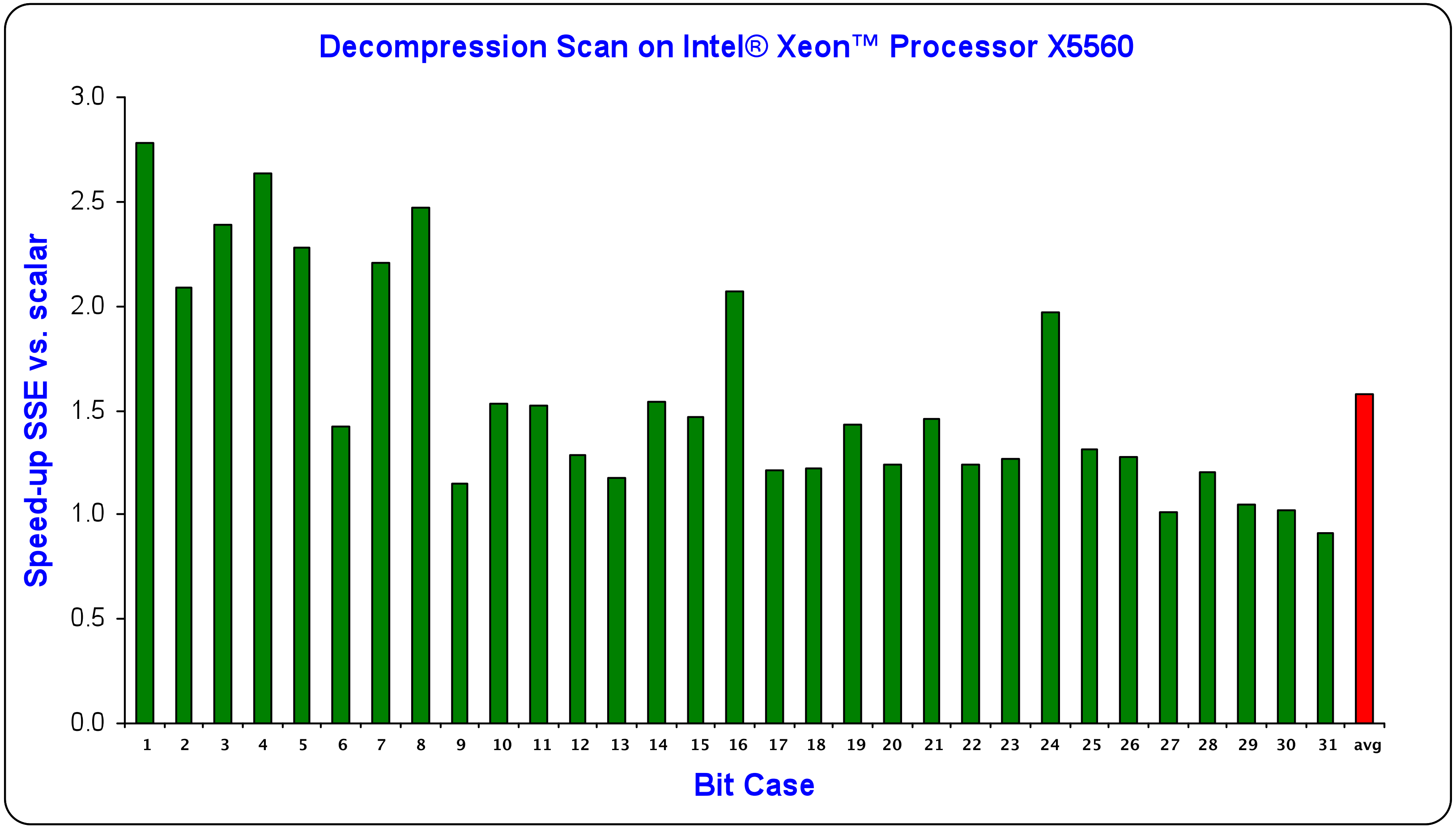
Blend results of different shift amount

Example: packed 4-bit fields



Use second Blend for remaining values

DECOMPRESS is 1.6x faster with SIMD

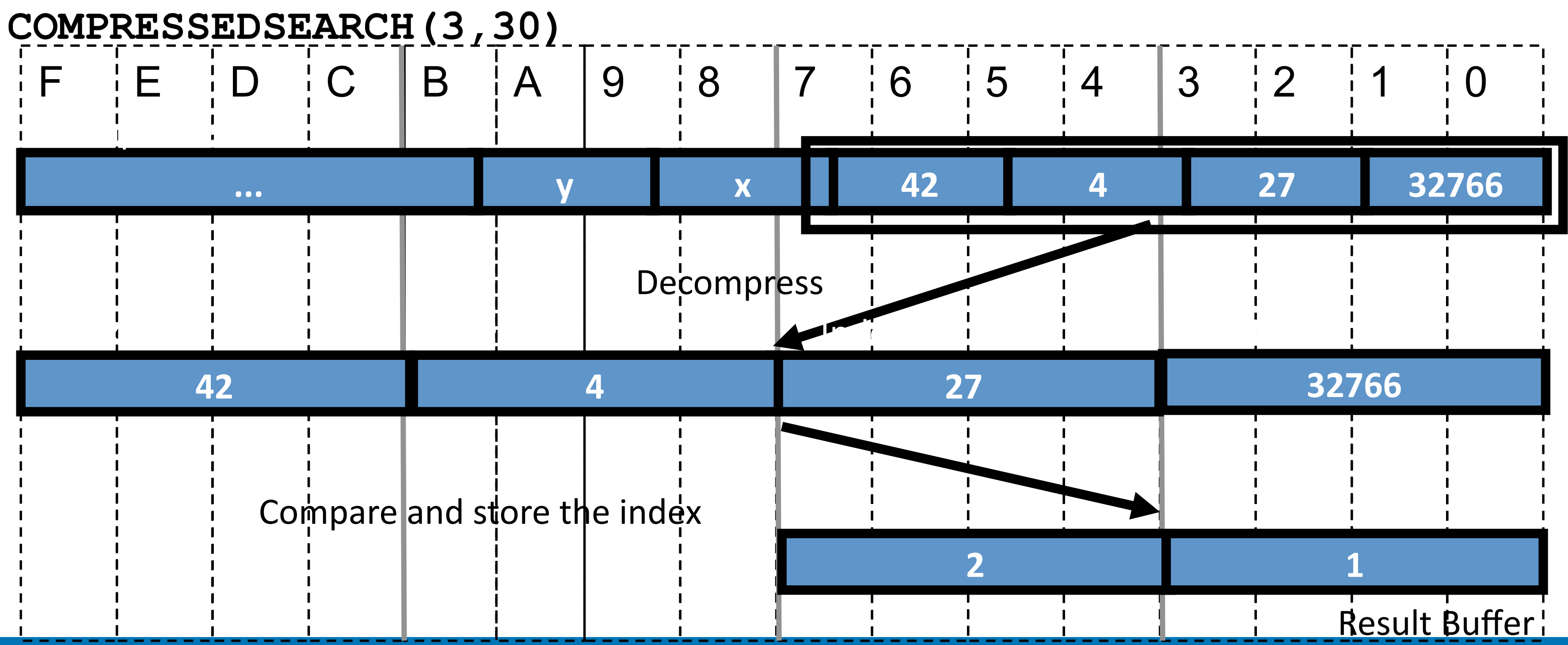


Software & Services Group

Full-Table Scan searches on compressed values

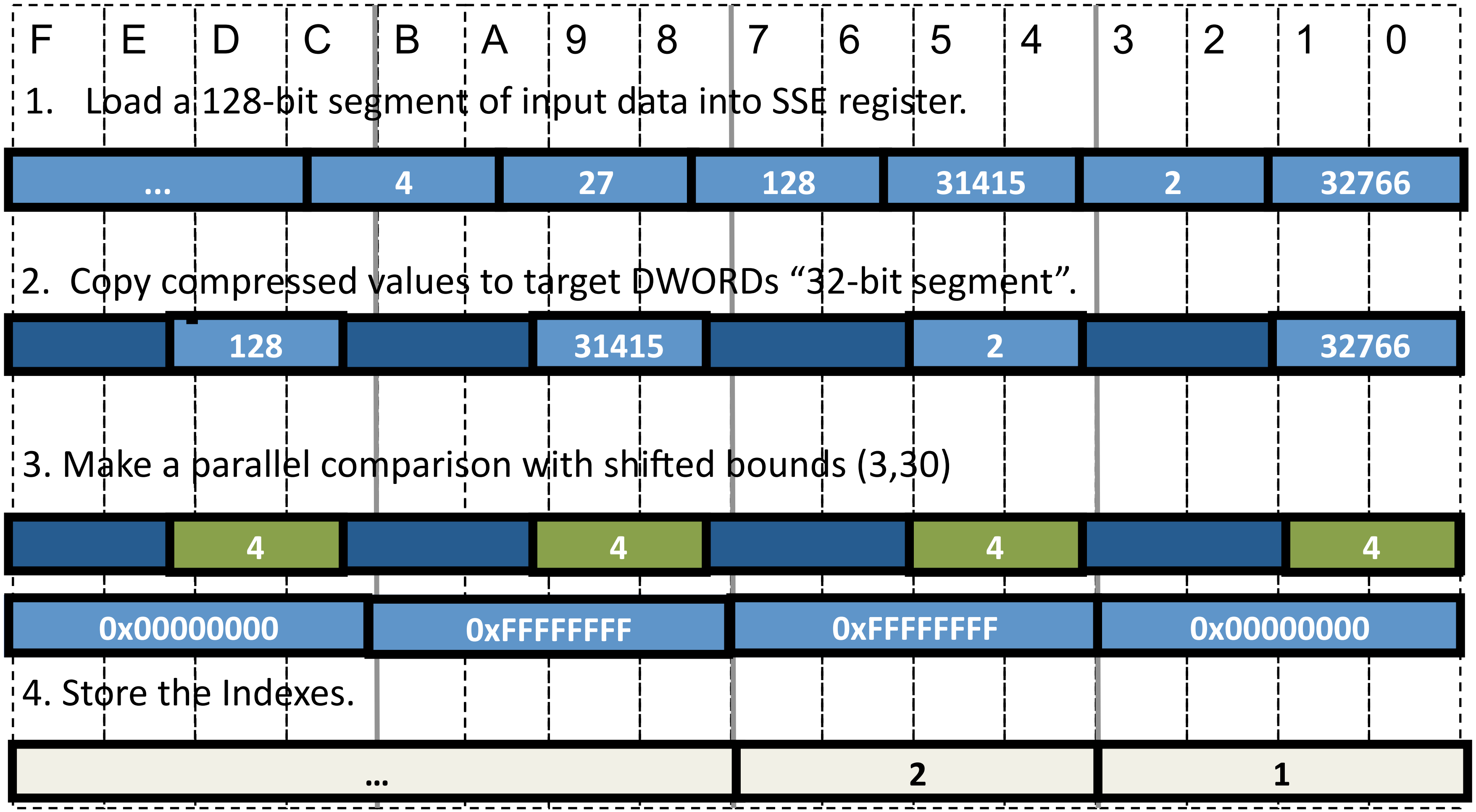
Algorithmic optimization by only decompressing the range of values that are of interest:

- DECOMPRESS
- And returns indexes of “Index Values” instead of decompressed “Index Values”



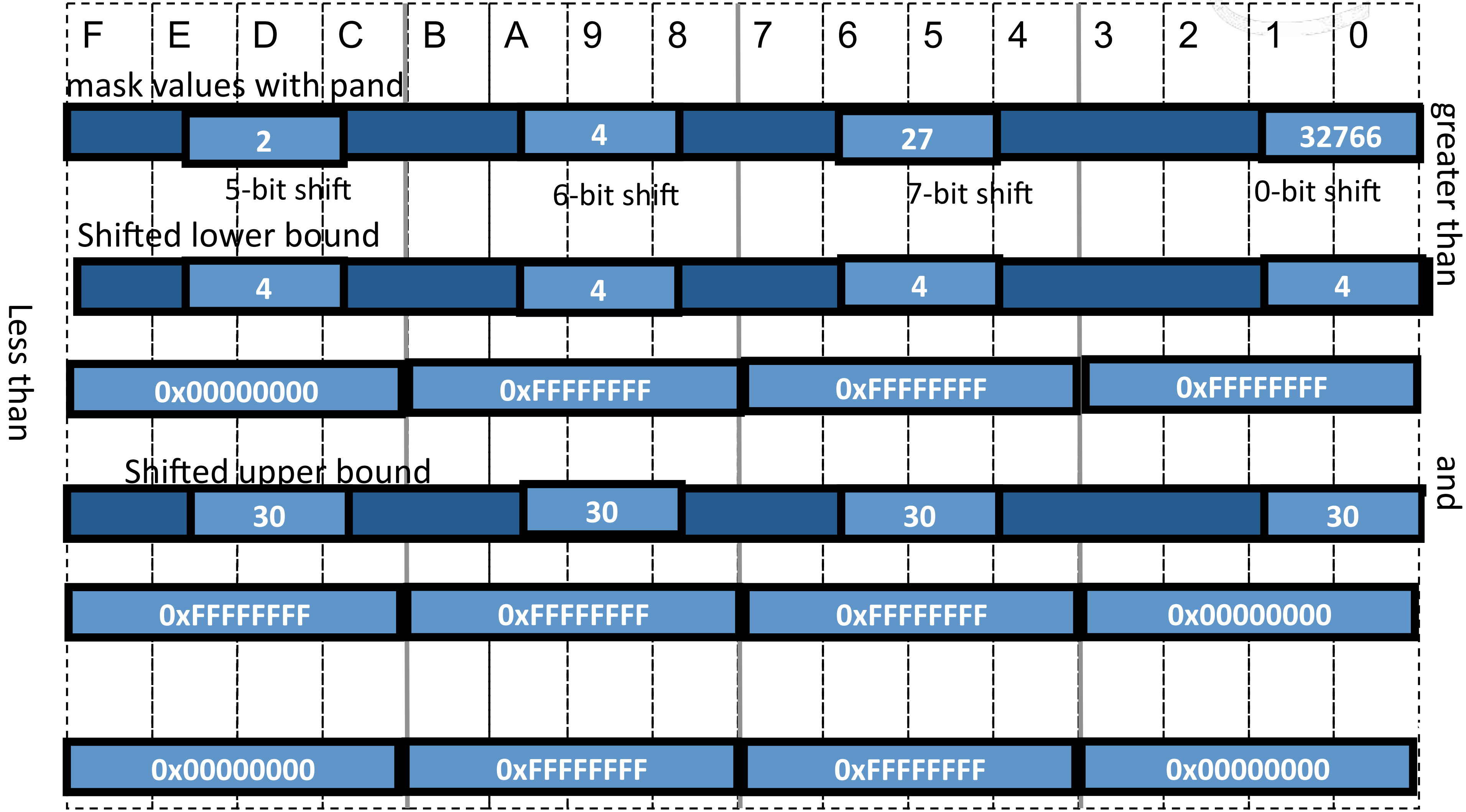
Basic idea of COMPRESSEDSEARCH

Example: COMPRESSEDSEARCH(3,30) for packed 17-bit fields



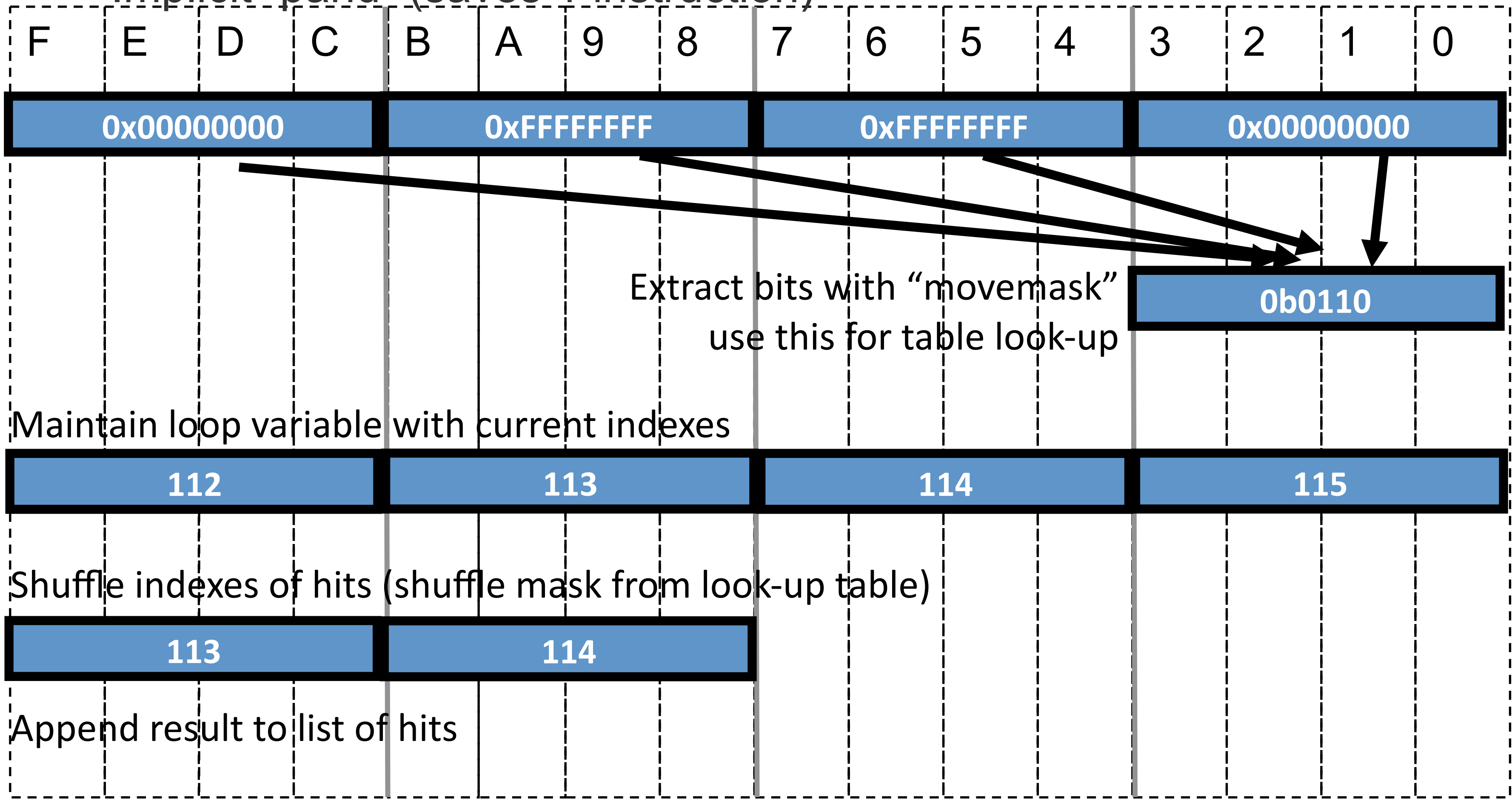
Compare shifted values

Example: COMPRESSEDSEARCH(3,30) for packed 15-bit fields

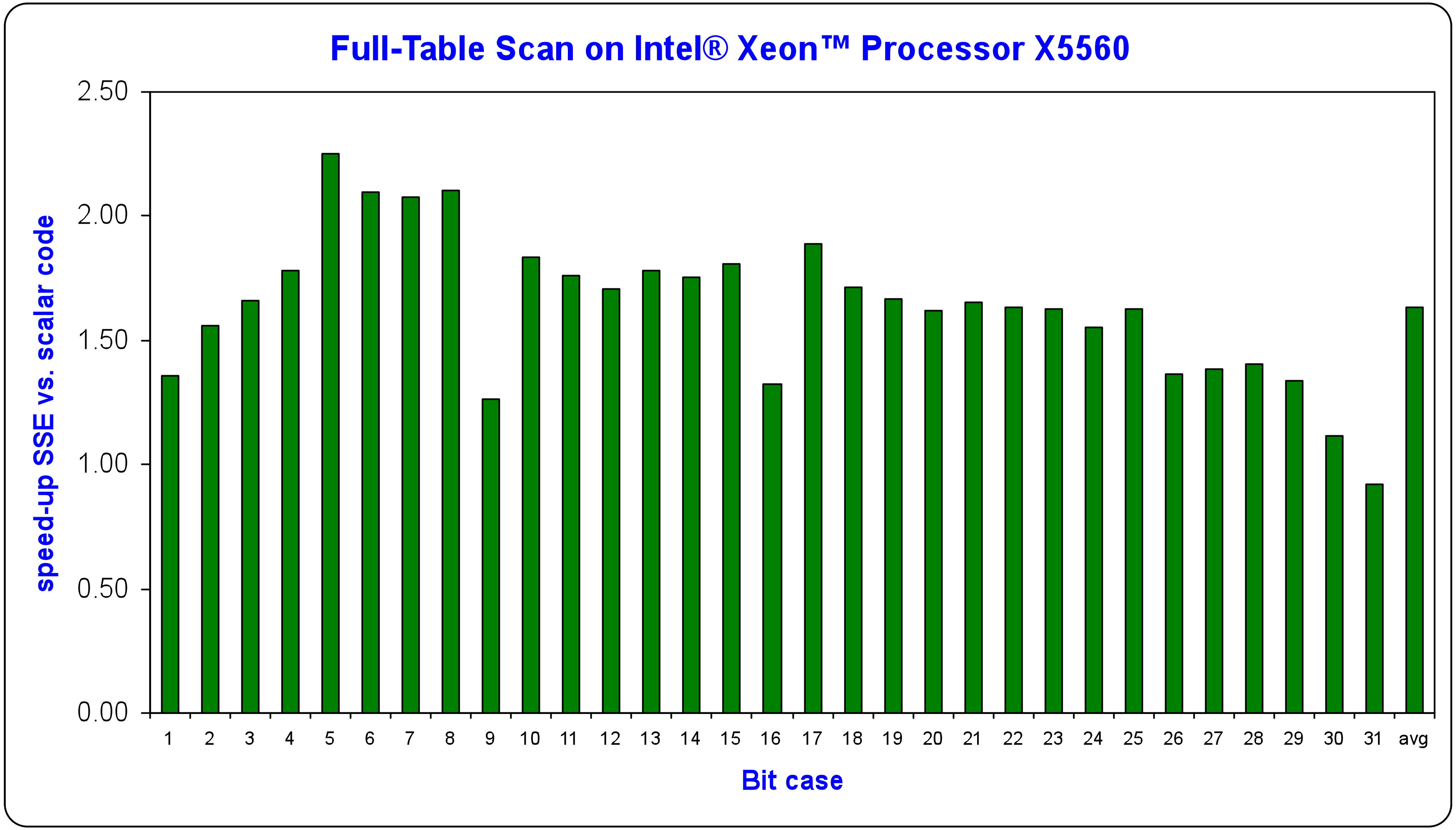


Hits are stored with look-up table

- Test first, if there are any hits with _mm_testz_si128
  - Implicit "pand" (saves 1 instruction)



Full-Table Scan is 1.6x faster with SIMD



Summary

- A single instruction processes multiple data
- Standard arithmetic operations are available
- Use masks for conditional code
- Special instrucions for searching and strings
- Column-orientation favors SIMD
- Shuffle and blend to arrange data
- Full-table scan is 1.6x faster with SIMD

Get creative and find new ways to use SIMD!

References

- Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A and 2B http://www.intel.com/products/processor/manuals/
- Intrinsics Reference http://software.intel.com/sites/products/documentation/studio/composer/en-us/2009/compiler_c/
- Interactive Intel Intrinsics Guide http://software.intel.com/en-us/avx/
- Intel® Advanced Vector Extensions (Intel® AVX) http://software.intel.com/en-us/avx/
- C++ Larrabee Prototype Library http://software.intel.com/en-us/articles/prototype-primitives-guide/

Software