New AVX extensions

S.Jarp
CERN

A rapid
(incomplete)
overview

28 June 2011
Sverre Jarp
CERN IT Department

Contents

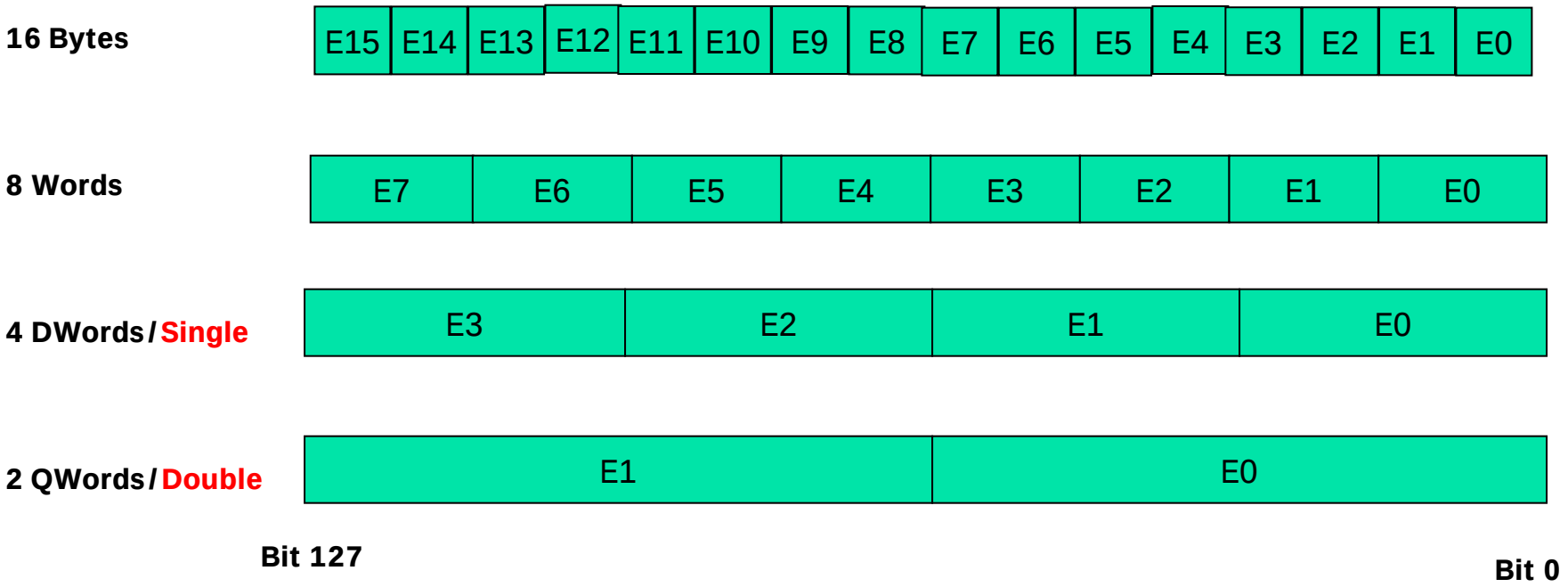
- AVX or “Advanced Vector eXtensions”:
  - Quick reminder of SSE and AVX1
  - New capabilities in AVX2:
    - Fused Multiply and Add
    - 16-bit FLP support
    - Gather instructions
    - 256-bit Integer vector instructions
    - Miscellaneous instructions
      - Bit manipulations, etc.
- Conclusion

FLP = Floating-Point

S.Jarp
CERN

XMM Registers

- 16 registers with 128 bits each



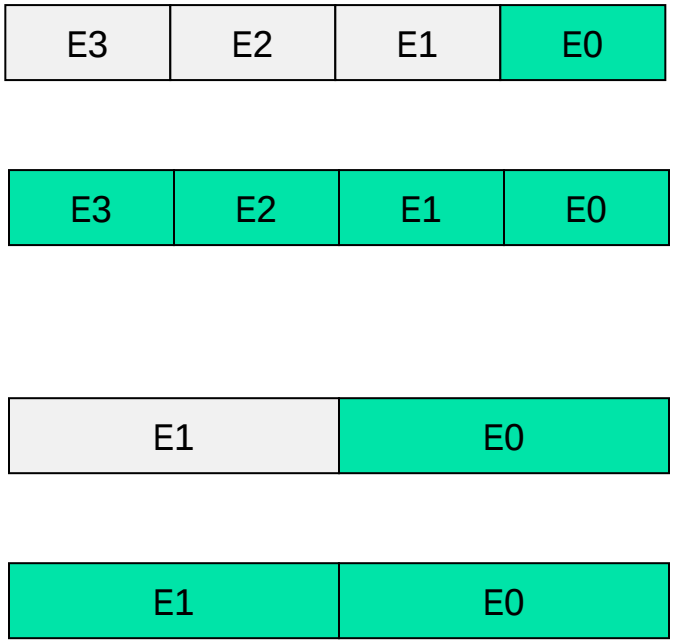
Streaming SIMD Extensions (SSE)

- Long list of instructions exploiting the 128-bit registers
  - Integer work
  - Floating-point work
- Evolution through a decade with incremental extensions:
  - SSE → SSE2 → SSE3 → SSSE3 → SSE4 (4.1 + 4.2)

Four FLP data flavours

S.Jarp
CERN

- Single precision
  - Scalar Single (SS)
  - Packed Single (PS)
- Double precision
  - Scalar Double (SD)
  - Packed Double (PD)



- Scalar SSE instructions replace x87 in Intel-64!

SSE FLP standard arithmetic

S.Jarp
CERN

	Mnemonic	SS	SD	PS	PD
Square root	SQRT	f3 0f 51	f2 0f 51	0f 51	66 0f 51
Normal add	ADD	f3 0f 58	f2 0f 58	0f 58	66 0f 58
Multiplication	MUL	f3 0f 59	f2 0f 59	0f 59	66 0f 59
Normal subtract	SUB	f3 0f 5c	f2 0f 5c	0f 5c	66 0f 5c
Division	DIV	f3 0f 5e	f2 0f 5e	0f 5e	66 0f 5e
Horizontal add	HADD	--	--	f2 0f 7c	66 0f 7c
Horizontal subtract	HSUB	--	--	f2 0f 7d	66 0f 7d
Add and subtract	ADDSUB	--	--	f2 0f d0	66 0f d0
Dot Product	DP	--	--	66 0f 3a 41	66 0f 3a 40
Reciprocal approximation	RCP	f3 0f 53	--	0f 53	--
Reciprocal SQRT approx.	RSQRT	f3 0f 52	--	0f 52	--

Integer instructions (128 bits)

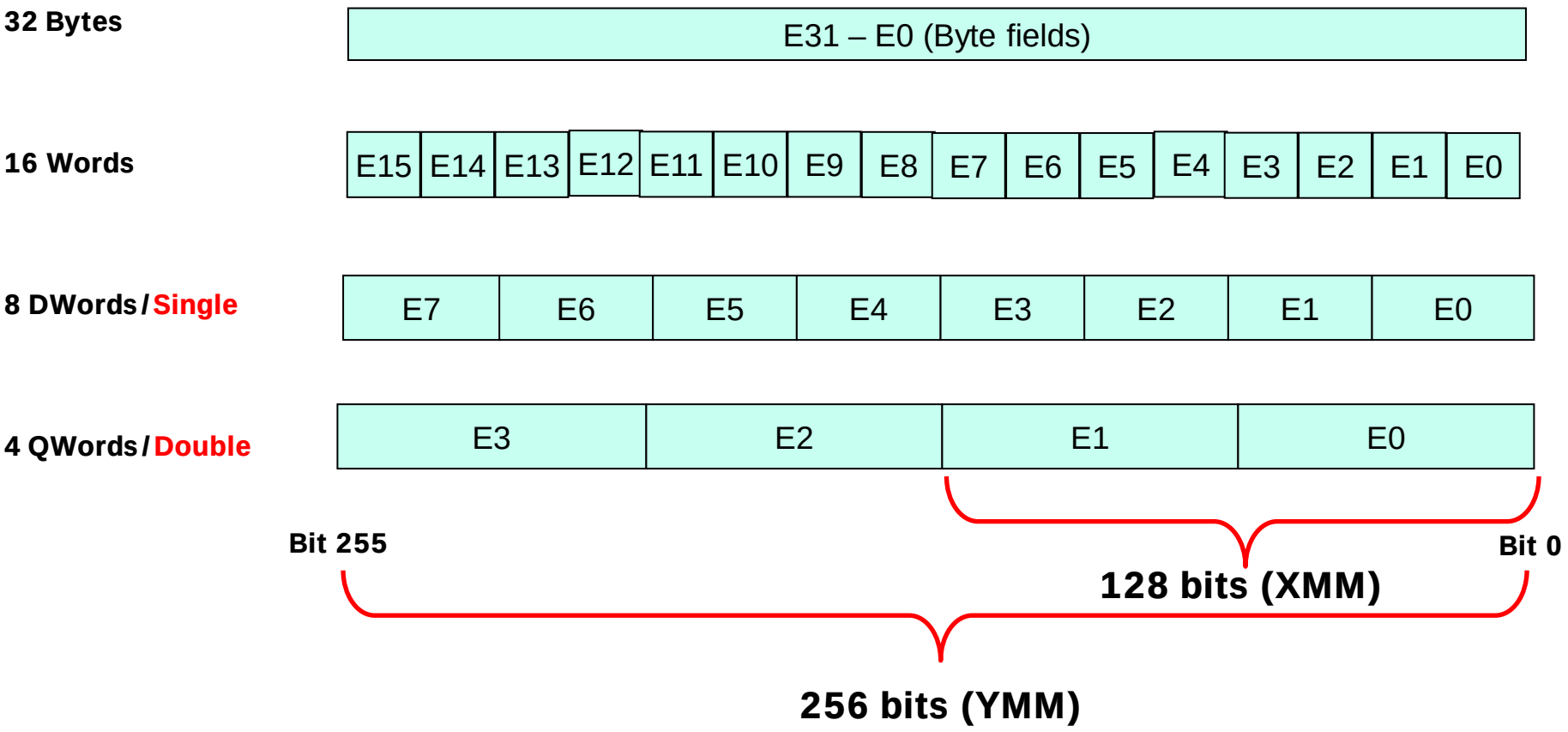
S.Jarp
CERN

- Long list of instructions
  - Manipulate Bytes, Words, Dwords, Qwords
  - Arithmetic (Add, Sub, Multiply)
    - Need to pay attention to details:
    - Are fields signed/unsigned?
    - Using saturation or not?
    - Retaining high/low part of multiplication?
    - etc.
  - Comparisons
  - Logic (And, Or, etc.)
  - Shifts
  - Max, Min, Avg., ...
  - Pack, Unpack, Moves, ...

With AVX: YMM registers

S.Jarp
CERN

- 16 registers with 256 bits each
  - Expands on top of the existing 128-bit XMM registers



Instruction evolution

S.Jarp
CERN

- Typically three flavours available:
  - SSE-variant (binary):
    - packusdw xmm1, xmm2/m128
  - AVX variant (ternary):
    - vpackusdw xmm1, xmm2, xmm3/m128
  - AVX2 variant (ternary, 256 bits):
    - vpackusdw ymm1, ymm2, ymm3/m256

AVX1 versus AVX2

S.Jarp
CERN

- AVX1: Introduced FLP support only
  - Single or double precision:
    - Traditional math operations
    - Logical operations: and, andn, or, xor
    - Blend and conditional merge
    - Conversions
    - Loads/stores and broadcasts
    - Miscellaneous:
      - Extract, insert, round, setzero, unpack, cast
    - Packed test operations

AVX1 versus AVX2 (cont'd)

- AVX2:
  - Fused Multiply and Add (FMA)
  - Gather instructions for floating-point
  - Broadcast, permute
  - 256-bit versions of 16-bit FLP support (in memory)
  - 256-bit integer vector instructions
    - Including gather
  - Miscellaneous (non-vector) instructions

Fused Multiply Add

- First idea found in VAX instruction set in the late 70s
- Now included in many other architectures
  - IBM RISC (Power), HP-PA, Itanium, LRBni, Fujitsu SPARC64
- Useful to minimise instructions in general
  - In particular, it is useful with polynomials
    - A * X2 + B*X + C = ((A * X + B) * X + C)

FMA in AVX2: How many?

- Naively, only two variants are needed:
  - (A * B + C)
  - (A * B – C)
- Then, we need instructions for single or double precision
- Then, we need scalar or packed
- So, since there is no SSE legacy in this case, are 8 instructions what they implemented?

FMA combinations

S.Jarp
CERN

- Six basic versions [96 instructions]
  - Multiply and Add [18]
    - vf**madd**
  - Multiply and Subtract [18]
    - vf**msub**
  - Negative Multiply and Add [18]
    - vf**nmadd**
  - Negative Multiply and Subtract [18]
    - vf**nmsub**
  - Multiply, then Add (even elements), Subtract (odd) [12]
    - vf**maddsub**
  - Multiply, then Subtract (even elements), Add (odd) [12]
    - vf**msubadd**

Two reasons for “bloat”

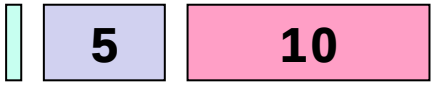
- Specify where the memory operand resides:
  - vfadd132pd ymm0, ymm1, ymm2/m256
  - vfadd213pd ymm0, ymm1, ymm2/m256
  - vfadd231pd ymm0, ymm1, ymm2/m256

- Also, they provide separate instructions for xmm registers:
  - vfadd132pd xmm0, ymm1, xmm2/m128
  - vfadd213pd xmm0, xmm1, xmm2/m128
  - vfadd231pd xmm0, xmm1, xmm2/m128

16-bit floating-point

S.Jarp
CERN

- Only in storage
  - 16 bits (S + E + M)
  - Approximate Normalized Range
    - 3.1×10^{-5} to 6.50×10^{4}



- Conversion to/from single
  - 32 bits (S + E + M)
  - Approximate Normalized Range
    - 1.18×10^{-38} to 3.40×10^{38}
  - Instructions:
    - vcvtph2ps ymm1, xmm2/m128
    - vcvtps2ph xmm1/m128, ymm2, imm8

FLP Gather Instructions

S.Jarp
CERN

- Brand new capability in x86:
  - Gather packed DP FP values using signed DW/QW indices
    - vgatherdpd xmm1,vm32x,xmm2
    - vgatherqpd xmm1,vm64x,xmm2
    - vgatherdpd ymm1,vm32x,ymm2
    - vgatherqpd ymm1,vm64y,ymm2
  - Gather packed SP FP values using signed DW/QW indices
    - vgatherdps xmm1,vm32x,xmm2
    - vgatherqps xmm1,vm64x,xmm2
    - vgatherdps ymm1,vm32y,ymm2
    - vgatherqpd xmm1,vm64y,xmm2

Integer Gather Instructions

S.Jarp
CERN

- New capability for Dwords and Qwords:
  - Gather packed DW values using signed DW/QW indices
    - vpgatherdd xmm1,vm32x,xmm2
    - vpgatherqd xmm1,vm64x,xmm2
    - vpgatherdd ymm1,vm32y,ymm2
    - vpgatherqd xmm1,vm64y,xmm2
  - Gather packed QW values using signed DW/QW indices
    - vpgatherdq xmm1,vm32x,xmm2
    - vpgatherqq xmm1,vm64x,xmm2
    - vpgatherdq ymm1,vm32x,ymm2
    - vpgatherqq ymm1,vm64y,ymm2

Gather Instructions (addressing scheme)

- Better understood by looking at definition of intrinsics:
  - vpgatherqq __m256i _mm_mask_i64gather_epi64(__m256i *src*, int const* *base*, __m256i *index*, __m256i *mask*, const int *scale*)

  Corresponds to:
  - vpgatherqq ymm1,vm64y,ymm2
- Usage notes:
  - GP (general protection) fault if any pair of the index, mask, destination registers are the same
  - Values are read from memory in any order
  - Faults are delivered right-to-left.
  - A given implementation of the instruction is repeatable
  - The instruction can be suspended by an exception
    - Corresponding fields in destination register and mask register are then updated

Integer instructions (256 bits)

S.Jarp
CERN

- Similar list of instructions (to SSE)
  - Manipulate Bytes, Words, Dwords, Qwords
  - Arithmetic (Add, Sub, Multiply)
  - Comparisons
  - Logic (And, Or, etc.)
  - Shifts
  - Max, Min, Avg., ...
  - Pack, Unpack, Moves, ...
- But, not necessarily a one-to-one upgrade
  - “Vast majority”
- Also new capabilities, such as gather
  - New: Shifts with per-element shift count

Non-vector instructions

- Bit manipulations (BMI1/LZCNT):
  - And not : andn r64a, r64b, r/m64
  - Bit field extract: bexr r64a, r/m64, r64b
  - Extract lowest set isolated bit: blsi r64, r/m64
  - Get mask up to lowest set bit: blsmsk r64a, r/m64
  - Reset lowest set bit: blsr r64, r/m64
  - Zero high bits from specified bit position: bzhi r64a, r/m64, r64b
  - Count number of leading zeroes: lzcnt r64, r/m64
  - Count number of trailing zero bits tzcnt r64, r/m64
- Others (BMI2):
  - Unsigned (H+L) multiply without affecting flags: mulx r64a, r64b, r/m64
  - Parallel bits deposit: pdep r64a,r64b,r/m64
  - Parallel bits extract: pext r64a,r64b,r/m64
  - Rotate right logical without affecting flags: rorx r64a,r/m64,imm8
  - Shift without affecting flags: Multiple (see manual)

Special instructions

- Random number generator:
  - rdrand r16
  - rdrand r32
  - rdrand r64

- From the manual: “RDRAND returns random numbers that are supplied by a cryptographically secure, deterministic random bit generator (DRBG). The DRBG is designed to meet the NIST SP 800-90 standard. The DRBG is re-seeded frequently from a on-chip non-deterministic entropy source to guarantee data returned by RDRAND is statistically uniform, non-periodic and non-deterministic.”

Conclusions

- Every x86 computer is a real vector computer!
- AVX2 continues to add powerful instructions
  - Available in a couple of year (w/Haswell μ-architecture)
- Impossible to “know” all the instructions
  - But, useful to know they exist
- Important to understand what the compiler will do for your inner loops
- Last word: There is probably (much) more to come!

Intel AVX Programming reference: 319433-011