Multicore Architecture Programming – SIMD

Olivier Aumage

Inria & LaBRI lab.

olivier.aumage @ inria.fr

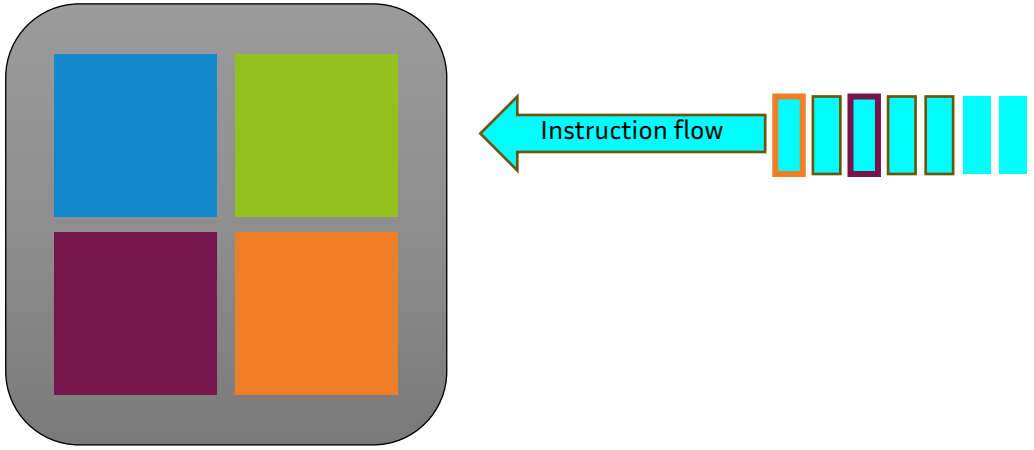
2024 – 2025

Introduction

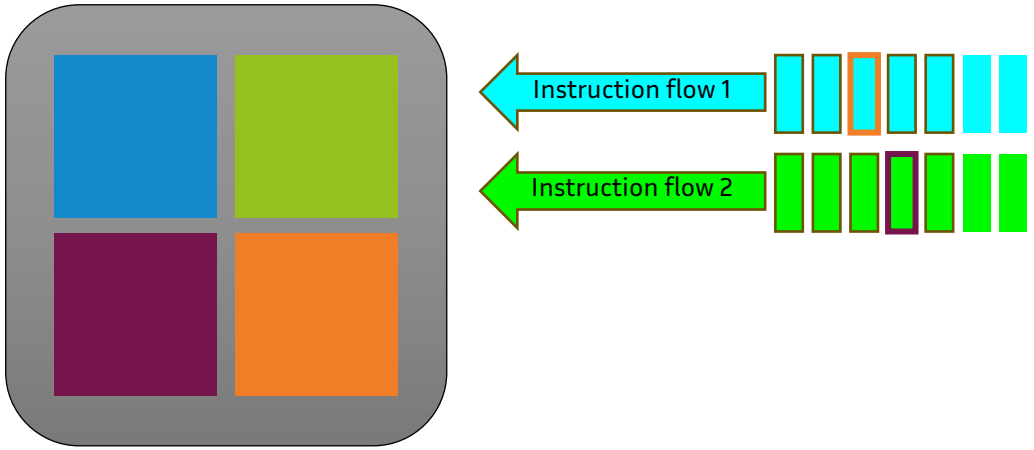
Doing More by Unit of Time

Hardware parallelism

- **Instruction-level parallelism (ILP)**
  - Parallelism within a single instruction flow



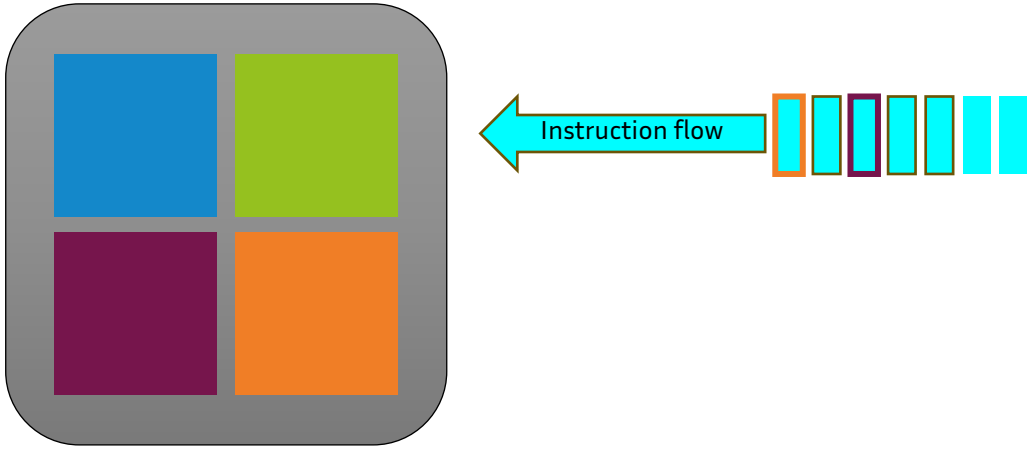
- **Thread-level parallelism (TLP)**
  - Parallelism across multiple instruction flows



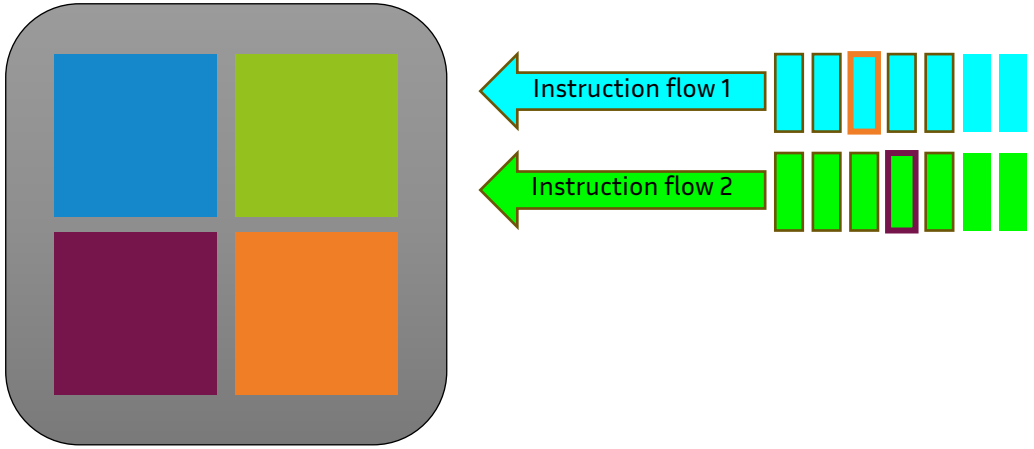
Doing More by Unit of Time

Hardware parallelism

- **Instruction-level parallelism (ILP)**
  - Parallelism within a single instruction flow



- **Thread-level parallelism (TLP)**
  - Parallelism across multiple instruction flows



Instruction-Level Parallelism

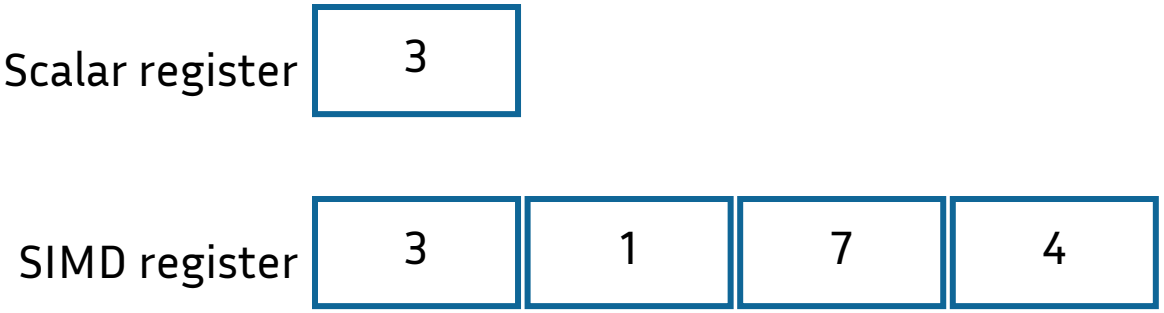
Extract parallelism within a single flow of instructions

- **Overlap the processing of successive instructions**
  - Pipelining
- **Process several instructions at a given time**
  - Overlapping / offload
  - Superscalar processing
- **Apply an instruction on multiple data**
  - SIMD instruction sets

SIMD Instruction Sets

Single Instruction Multiple Data

- **General principle**
  - SIMD register == small vector

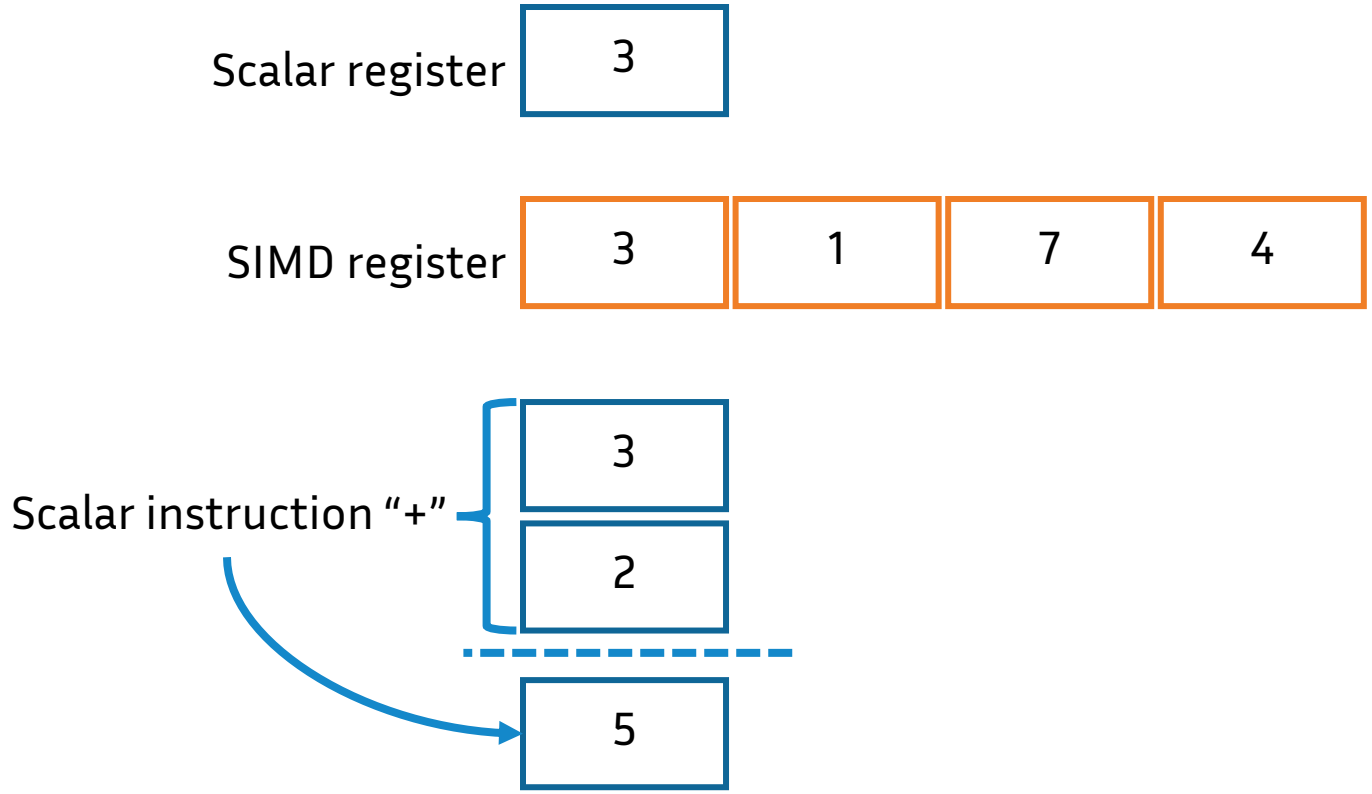


Typical RISC instruction processing sequence

SIMD Instruction Sets

Single Instruction Multiple Data

- **General principle**
  - SIMD register == small vector
  - SIMD arithmetic instructions
    - Ops applied to vector items…
    - … in parallel

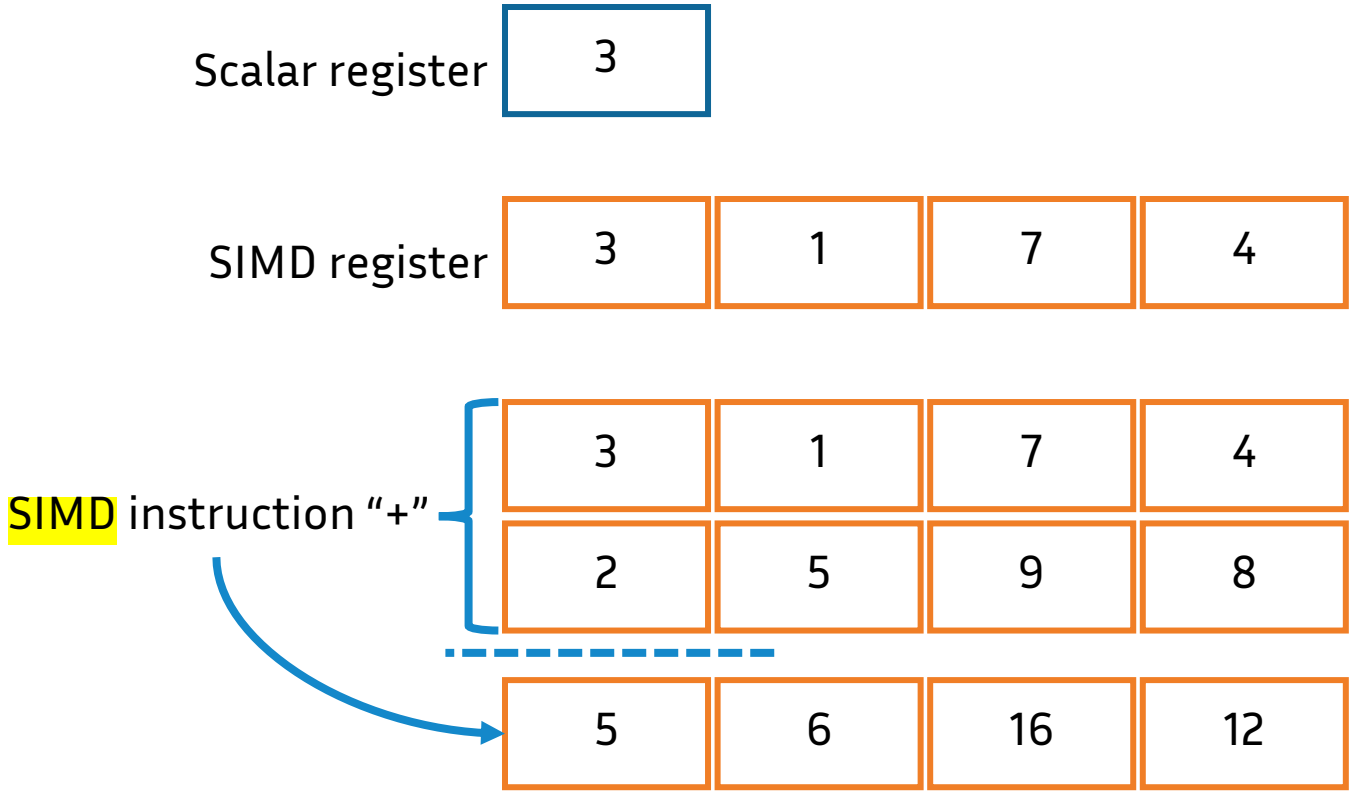


Typical RISC instruction processing sequence

SIMD Instruction Sets

Single Instruction Multiple Data

- **General principle**
  - SIMD register == small vector
  - SIMD arithmetic instructions
    - Ops applied to vector items…
    - … in parallel



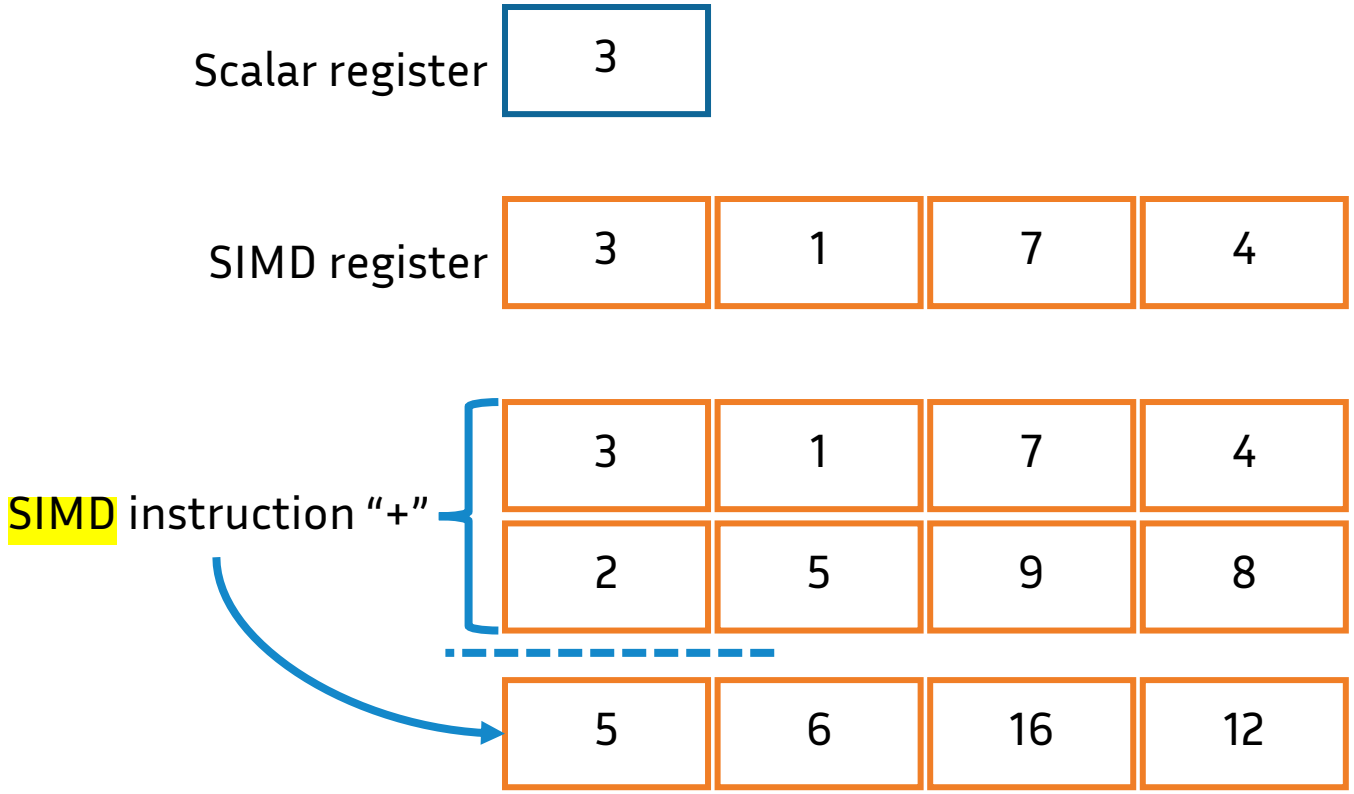
Typical RISC instruction processing sequence

SIMD Instruction Sets

Single Instruction Multiple Data

- **General principle**
  - SIMD register == small vector
  - SIMD arithmetic instructions
    - Ops applied to vector items…
    - … in parallel

- **Example**
  - Intel Intrinsics Guide [link]



Typical RISC instruction processing sequence

Short History of SIMD Instruction Sets

x86 & x86-64 architectures (Intel, AMD)

- **MMX**: Intel, 1997
  - 64-bit wide registers (`mmx0..`)
  - Operations on 2 x 32-bit, 4 x 16-bit, 8 x 8-bit integer elements
- **3DNow!**: AMD, 1998, (deprecated)
  - Add 2 x single precision (32-bit) floating-point elements support
- **SSE**: Intel, 1999
  - 128-bit wide registers (`xmm0..`)
  - 4 x single precision floating-point
- **SSE2**: Intel, 2000
  - Add 2 x double precision (64-bit) floating-point elements support
  - Add integer elements support: 2 x 64-bit, 4 x 32-bit, 8 x 16-bit, 16 x 8-bit
  - Follow-ups: **SSE3**, **SSSE3**, **SSE4**...
- **AVX**: Intel, 2011 (Sandy Bridge proc.)
  - 256-bit wide registers (`ymm0..`)
  - Floating point instructions:
    4 x 64-bit (DP) & 8 x 32-bit (SP)
- **AVX2**: Intel, 2013 (Haswell proc.) [PlaFRIM miriel]
  - Add integer instructions: 64-bit, 32-bit, 16-bit, 8-bit elts
  - *Fused Multiply-Accumulate* (FMA), AMD 2012
- **AVX-512**: Intel
  - Multiple sub-sets of new instructions
  - 512-bit wide registers (`zmm0..`)
  - Masked operations
  - [link]

Short History of SIMD Instruction Sets

Some other contemporary architectures

- **ARM**
  - **NEON**: 128-bit registers, 8/16/32/64-bit integers, single precision floating point (+double prec. on 64-bit ARM archi. / AArch64)
  - **SVE**: abstract, implementation-defined register width from 128-bit to 2048-bit

- **RISC-V**
  - **RISC-V "V" vector extension**, (spec. v1.0, 2021)
  - ELEN: implementation defined max element size in bit
    - ELEN >= 8
  - VLEN: implementation defined max register len
    - VLEN >= ELEN
    - VLEN <= 2^16

Programming with SIMD Instruction Sets

Multiple means

- **Vectorizing compilers**
  - Origin
    - Libre &open source: GNU GCC, LLVM Clang
    - Vendor compilers
  - Languages
    - Parallel languages: OpenMP, OpenACC
    - Language extensions & pragmas
- **Wrapper libraries**
  - Mainly C++
- **Intrinsics pseudo-routines**
  - C / C++
  - [link Intel MMX / SSE / AVX], [link ARM SVE]
- **Assembly language**
  - Asm code

Programming with SIMD Instruction Sets

Multiple means

- **Vectorizing compilers**
  - Origin
    - Libre &open source: GNU GCC, LLVM Clang
    - Vendor compilers
  - Languages
    - Parallel languages: OpenMP, OpenACC
    - Language extensions & pragmas
- **Wrapper libraries**
  - Mainly C++
- **Intrinsics pseudo-routines**
  - C / C++
  - [link Intel MMX / SSE / AVX], [link ARM SVE]
- **Assembly language**
  - Asm code

```
#include <immintrin.h>

void xpy (float *x, float *y, float *z, int vector_size)
{
  for (i=0; i<vector_size; i+=REG_NB_ELEMENTS)
  {
    __m256 reg_x;
    __m256 reg_y;
    __m256 reg_z;

    /* Load A and B arrays in SIMD registers */
    reg_x = _mm256_load_ps(&x[i]);
    reg_y = _mm256_load_ps(&y[i]);

    /* Perform SIMD add operation */
    reg_z = _mm256_add_ps(reg_x, reg_y);

    /* Store SIMD register in C array */
    _mm256_store_ps(&z[i], reg_z);
  }
}
```

Addition of two vectors of single precision float elements in AVX2
(Note: assuming vectors are properly aligned, and `vector_size` is a multiple of the number of REG_NB_ELEMENT)

Examples of Common SIMD Instructions – AVX2

Load / Store / Set*

- **Load**
  - Load data from memory into a SIMD register:
    - Aligned data: `_mm256_load_*`
    - Unaligned data: `_mm256_loadu_*`
- **Store**
  - Store data from a SIMD register into memory
    - Aligned data: `_mm256_store_*`
    - Unaligned data: `_mm256_storeu_*`
- **Set**
  - Set the value of a SIMD register
    - Set all reg. elements to zero: `_mm256_setzero_*`
    - Set all reg. element to the same value: `_mm256_set1_*`
    - Set all reg. elements to distinct values: `_mm256_set_*`

Examples of Common SIMD Instructions – AVX2

Load / Store / Set*

- **Load**
  - Load data from memory into a SIMD register:
    - Aligned data: `_mm256_load_*`
    - Unaligned data: `_mm256_loadu_*`
- **Store**
  - Store data from a SIMD register into memory
    - Aligned data: `_mm256_store_*`
    - Unaligned data: `_mm256_storeu_*`
- **Set**
  - Set the value of a SIMD register
    - Set all reg. elements to zero: `_mm256_setzero_*`
    - Set all reg. element to the same value: `_mm256_set1_*`
    - Set all reg. elements to distinct values: `_mm256_set_*`

Allocating aligned memory

`malloc(size)` → `aligned_alloc(alignment, size)`

Available in standard C 11

Examples of Common SIMD Instructions – AVX2

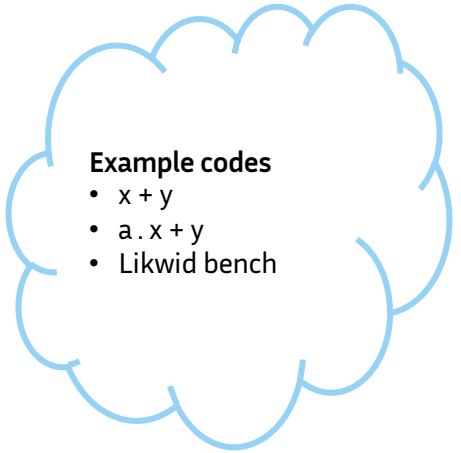
Arithmetics

- **Add**
  - Add two registers, element by element: **_mm256_add_***
- **Sub, Mul**
  - Subtract two registers, element by element: **_mm256_sub_***
  - Multiply two registers, element by element: **_mm256_mul_***
- **Horizontal Add, Sub**
  - Add adjacent pairs of elements : **_mm256_hadd_***
  - Subtract adjacent pairs of elements : **_mm256_hsub_***
- **Fused Multiply-Accumulate** (FMA)
  - "d = a x b + c" in a single operation: **_mm256_fmadd_***

Examples of Common SIMD Instructions – AVX2

Arithmetics

- **Add**
  - Add two registers, element by element: _mm256_add_*
- **Sub, Mul**
  - Subtract two registers, element by element: _mm256_sub_*
  - Multiply two registers, element by element: _mm256_mul_*
- **Horizontal Add, Sub**
  - Add adjacent pairs of elements : _mm256_hadd_*
  - Subtract adjacent pairs of elements : _mm256_hsub_*
- **Fused Multiply-Accumulate** (FMA)
  - “d = a x b + c” in a single operation: _mm256_fmadd_*



Examples of Common SIMD Instructions – AVX2

Comparison operators

- **Cmpeq**
  - Compare two registers element by element for equality: **_mm256_cmpeq_***
- **Cmpgt**
  - Compare two registers element by element for “<” (greater-than) operator: **_mm256_cmpgt_***

Examples of Common SIMD Instructions – AVX2

Boolean operators

- **And**
  - Apply ”and” operator, bit by bit: **_mm256_and_***
- **Not, Andnot, Or, Xor**
  - **_mm256_[not | andnot | or | xor]_***
- **Shift left, right**
  - Shift each element bit-wise by some number of bits to the left: **_mm256_[sll | sla]***
  - Shift to the right: **_mm256_[srl | sra]***

Examples of Common SIMD Instructions – AVX2

Shuffle / Permute / Blend / Broadcast

- **Permute / shuffle**
  - Pick selected elements from a register, from two registers (naming not entirely consistent):
    - _mm256_permute*
    - _mm256_shuffle_*
- **Blend**
  - Pick selected elements from either a first register or a second register: _mm256_blend_*
- **Broadcast**
  - Copy a register element value to the all the other elements.
    - Example: _mm256_broadcastd_epi32
- **Note**
  - Control by *immediate value*: compile-time constant mandatory
  - Control by *register*: can be computer at runtime

Examples of Common SIMD Instructions – AVX2

Extract / Insert / Pack / Unpack

- **Extract**
  - Extract a specific element or part from a register
  - **_mm256_extract***
- **Insert**
  - Set the value of a specific register element
  - **_mm256_insert***
- **Pack**
  - Convert elements of two registers to half-shorter elements, and interleave them in a single register
  - **_mm256_packs_***
- **Unpack**
  - Interleave the higher [or lower] part of two registers and interleave them in a single register:
  - **_mm256_unpackhi_***
  - **_mm256_unpacklo_***

Examples of Common SIMD Instructions – AVX2

Gather

- **Gather**
  - Collect indirectly addressed data elements from memory into a register
  - _mm256_[i32 | i64]gather_*

TD

Introduction to using *intrinsics* routines for the Intel AVX2 instruction set

- **Assignment, source codes, additional slides**
  - Moodle ENSEIRB, IT390 course