MMX and SSE (lesson 7)

Arquitectura de Computadores Avançada
Universidade de Aveiro – DETI

José Luís Azevedo / Nuno Lau

Outline

- Introduction to MMX programming model
  - MMX data types
  - Data transfer instructions
  - Basic sub-set of arithmetic instructions
- Introduction to SSE programming model
  - SSE data types
  - Data transfer instructions
  - Basic sub-set of arithmetic instructions
- Embedding MMX/SSE assembly instructions in C code

Multimedia extensions

- Extensions to the base ISA that allow a form of vector computation
- “Vectors” are implemented in dedicated registers
- Registers of N bits may be used as vectors of 2x(N/2) elements, 4x(N/4), 8x(N/8) elements, etc.
- Single Instruction Multiple Data paradigm (SIMD)
  - **One multimedia instruction applies simultaneously to all elements of a register**

MMX

- Multimedia extensions for Intel x86 architecture, introduced in 1997
- SIMD
- MMX provides only integer operations
- A specific set of registers (shared with FP unit)
- 64-bit registers
- 8 registers: **MM0, MM1, ..., MM7**

SSE (streaming SIMD extensions)

- Evolution of MMX (introduced in 1999)
- Integer and floating point operations (single precision operands, i.e., 32 bits)
- 128-bit registers
- 8 registers: **XMM0**, **XMM1**, ..., **XMM7**

- **SSE2** (2001), **SSE3** (2004), **SSE4** (2006)

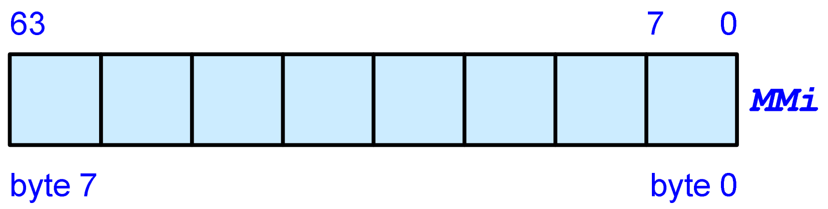
SSE evolution

- **AVX** (Advanced Vector Extensions) – first supported by Intel Sandy Bridge processor, 2011, and later on by AMD (Bulldozer processor)
  - 256-bit registers (**YMM0**...**YMM15**)

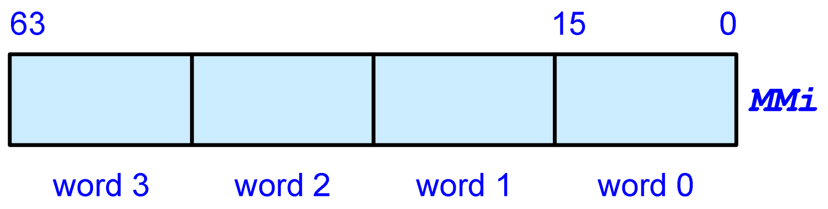
- **AVX-512** expands AVX to 512-bit support (first supported by Intel with the Knights Landing processor scheduled to ship in 2015)
  - 512-bit registers (**ZMM0**...**ZMM31**)

MMX data types (64-bit registers)

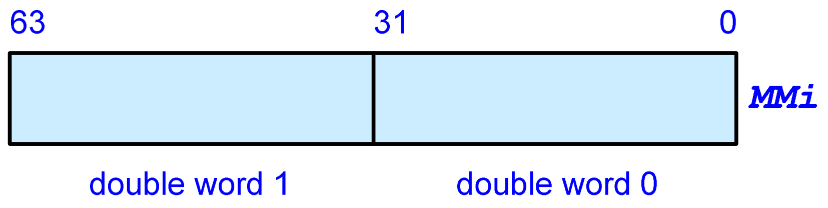
- Packed bytes (8)



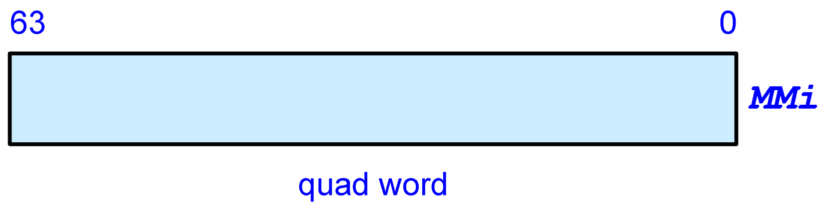
- Packed words (4)
  (a word is 16 bits)



- Packed double words
  (a double word is 32 bits)



- Quad word



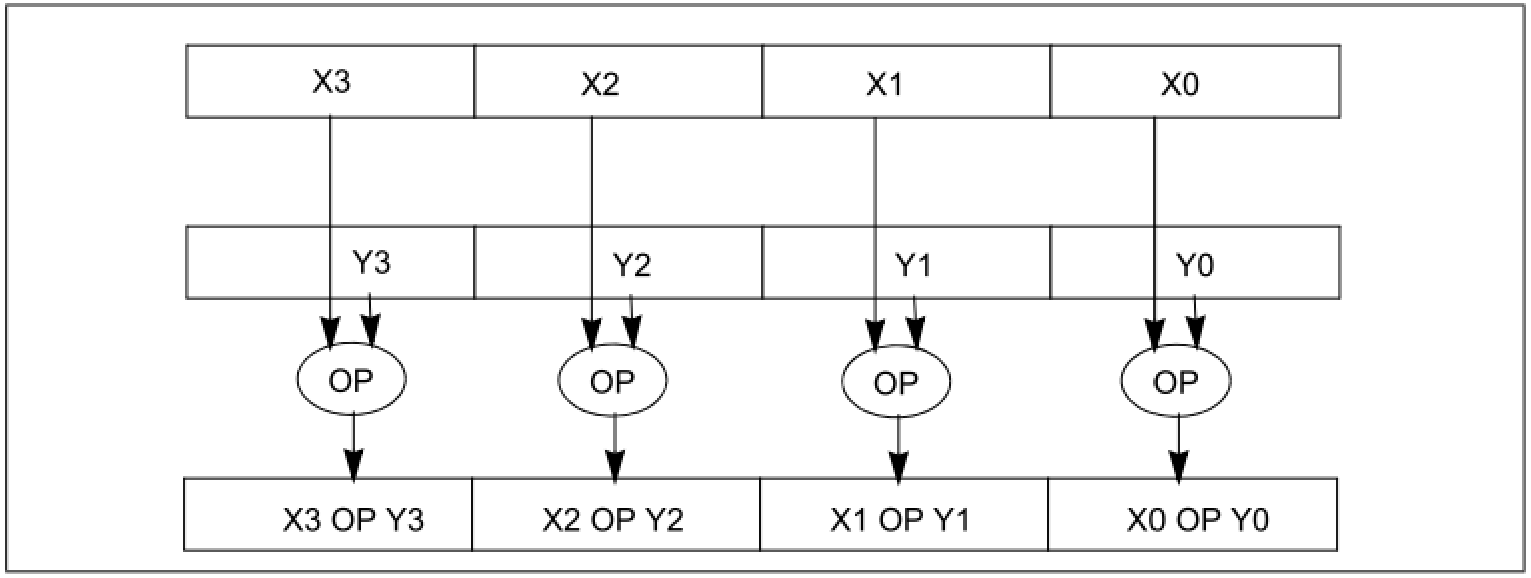
Data transfer instructions

- `movd` - move double (32-bit transfer)
  - copies data between a 32-bit integer register or a double word memory location and an MMX register
  - if the destination is an MMX register (64-bits), this instruction zero-extends the value while moving it
  - if the destination is a 32-bit register or memory location, this instruction copies the low-order 32-bits of the MMX register to the destination
- **movd** $\mathbf{reg_{32}}$**,mmi**   **;**$\mathbf{reg_{32}}$**->mmi**
- **movd mmi,**$\mathbf{reg_{32}}$ **;mmi->**$\mathbf{reg_{32}}$
- **movd** $\mathbf{mem_{32}}$**,mmi**   **;[**$\mathbf{mem_{32}}$**]->mmi**
- **movd mmi,**$\mathbf{mem_{32}}$ **;mmi->[**$\mathbf{mem_{32}}$**]**

Data transfer instructions

- **movq** - move quad (64-bit transfer)
  - copies data between two MMX registers or between an MMX register and memory
  - if either the source or destination operand is a memory object, it must be a **qword** variable
- **movq mmi,mem**$_{64}$ **;mmi->[mem**$_{64}$**]**
- **movq mem**$_{64}$**,mmi ;[mem**$_{64}$**]->mmi**
- **movq mmi,mmj ;mmi->mmj**

Packed Arithmetic Instructions



Packed Arithmetic Instructions

- Add Packed Bytes
  - **`paddb mem64,mmi`** **;[mem_{64}]+mmi->mmi**
  - **`paddb mmj,mmi`** **;mmi+mmj->mmi**

- the **PADDB** instruction adds the individual bytes (8), in the two 64-bit operands using a wrap-around (i.e., non-saturating) addition
- any carry out of a sum is lost

Packed Arithmetic Instructions

- Add Packed Words
  - `paddw mem64,mmi ;[mem64]+mmi->mmi`
  - `paddw mmj,mmi ;mmi+mmj->mmi`

- the **PADDW** instruction adds the individual words (4), in the two 64-bit operands using a wrap-around (i.e., non-saturating) addition
- any carry out of a sum is lost

Packed Arithmetic Instructions

- Add Packed Double Words
  - `paddd mem64,mmi ;[mem64]+mmi->mmi`
  - `paddd mmj,mmi ;mmi+mmj->mmi`

- the **PADDD** instruction adds the individual double words (2), in the two 64-bit operands using a wrap-around (i.e., non-saturating) addition
- any carry out of a sum is lost

Packed Arithmetic Instructions

- Add Packed Saturation Bytes, signed
  - `paddsb` mem_{64}`,mmi` `;[`mem_{64}`]+mmi->mmi`
  - `paddsb mmj,mmi   ;mmi+mmj->mmi`

- The **PADDSB** instruction adds the individual bytes (8), in the two 64-bit operands using a signed saturation arithmetic (results are in the range: -128, +127)

Packed Arithmetic Instructions

- Add Packed Saturation Words, signed
  - `paddsw mem64,mmi ;[mem64]+mmi->mmi` (mem_{64})
  - `paddsw mmj,mmi ;mmi+mmj->mmi`

- The **PADDSW** instruction adds the individual words (4), in the two 64-bit operands using a signed saturation arithmetic (results are in the range:
  -32768, +32767)

Packed Arithmetic Instructions

- Add Packed Saturation Bytes, unsigned
  - **paddusb mem$_{64}$,mmi ;[mem$_{64}$]+mmi->mmi**
  - **paddusb mmj,mmi ;mmi+mmj->mmi**

- The **PADDUSB** instruction adds the individual bytes (8), in the two 64-bit operands using an unsigned saturation arithmetic (results are in the range: 0, 255)

Packed Arithmetic Instructions

- Add Packed Saturation Words, unsigned
  - `paddusw` mem_{64},mmi ;[mem_{64}]+mmi->mmi
  - `paddusw mmj,mmi ;mmi+mmj->mmi`

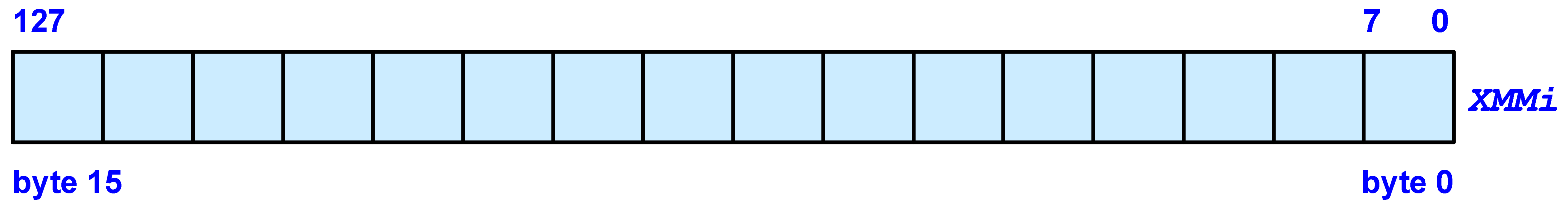
- The **PADDUSW** instruction adds the individual words (4), in the two 64-bit operands using an unsigned saturation arithmetic (results are in the range: 0, 65535)

SSE (streaming SIMD extensions)

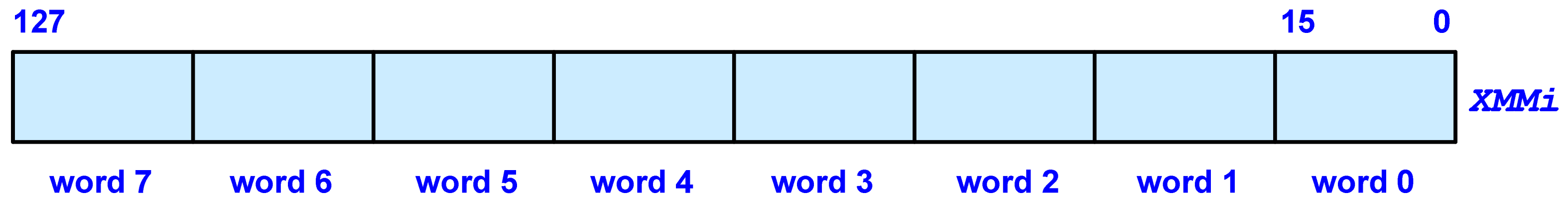
- 128-bit registers
- 8 registers: **XMM0**, **XMM1**, ..., **XMM7**
- New instructions, but all MMX arithmetic instructions are supported (using **XMM*i*** registers)
- Hence, MMX instructions can be used with SSE registers

SSE data types (128-bit registers)

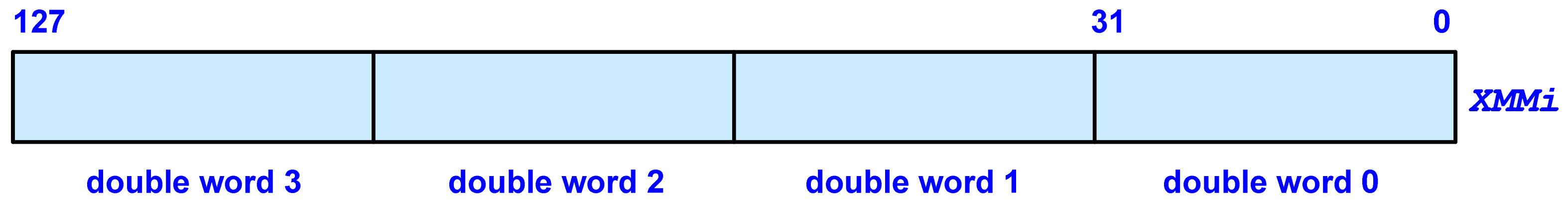
- Packed bytes (16)



- Packed words (8)

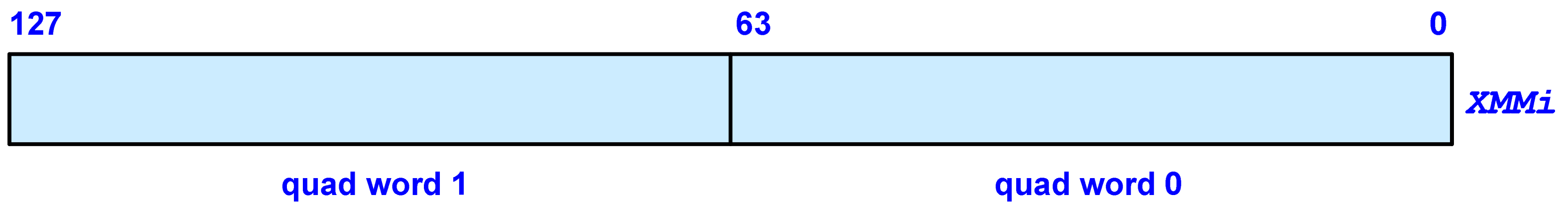


- Packed double words (4)

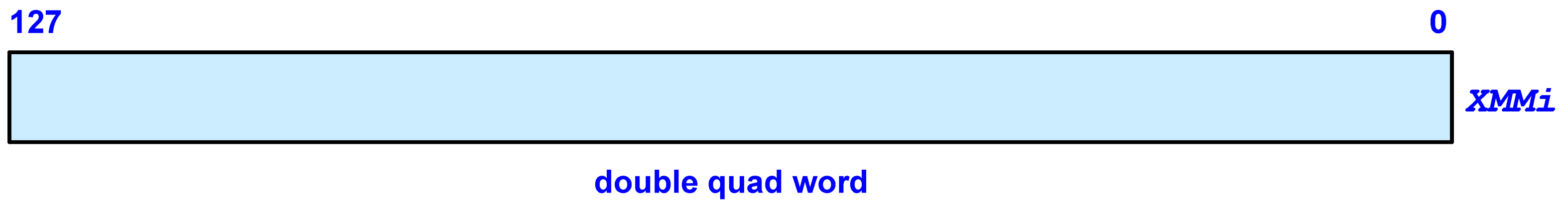


SSE data types (128-bit registers)

- Packed quad words (2)



- Double quad word



SSE data transfer instructions

- **movdqa** - move aligned double quad word (128-bit transfer)
  - copies data between two SSE registers or between an SSE register and memory
  - **the memory address must be aligned to a 16-byte boundary**; otherwise, a general-protection exception is generated
- **movdqa xmmi,mem$_{128}$** **;xmmi->[mem$_{128}$]**
- **movdqa mem$_{128}$,xmmi** **;[mem$_{128}$]->xmmi**
- **movdqa xmmi,xmmj** **;xmmi->xmmj**

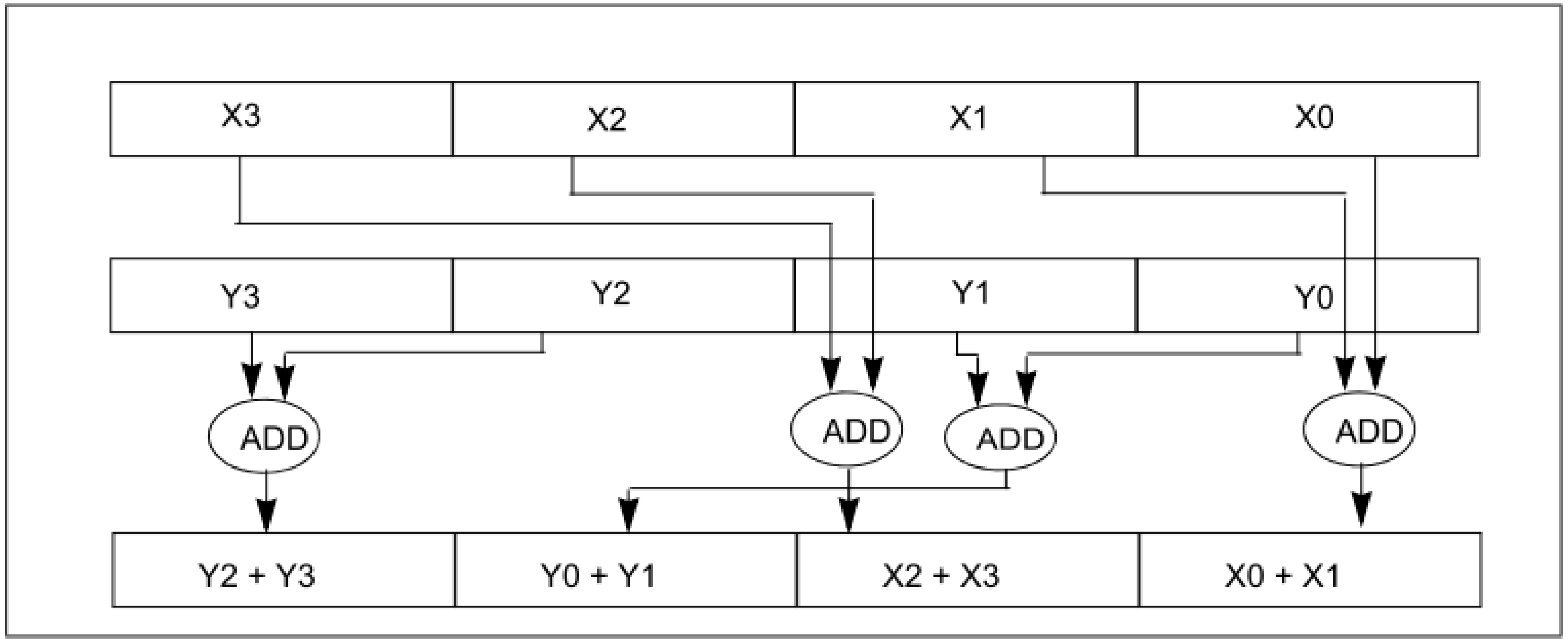
SSE Packed Arithmetic Instructions

- All MMX arithmetic instructions are supported (using *XMMi* registers)

- **paddb**	- add packed bytes
- **paddw**	- add packed words
- **paddd**	- add packed double words
- **paddsb**	- add packed saturation byte (signed)
- **paddsw**	- add packed saturation words (signed)
- **paddusb**	- add packed saturation bytes (unsigned)
- **paddusw**	- add packed saturation words (unsigned)

SSE – PHADDD instruction

- **phaddd** - horizontal add packed double words



Horizontal Data Movement in PHADDD

Embedding MMX/SSE assembly Instructions in C code

```
void sum(int *a, int *b, int *c, int size)
{
  for(int i=0; i < size; i += 2) {
    __asm__ volatile
      (
      "\n movq    %1,%%mm0"
      "\n movq    %2,%%mm1"
      "\n paddd   %%mm0,%%mm1"
      "\n movq    %%mm1,%0"
      : "=m" (c[i])   // %0
      : "m"  (a[i]),  // %1
        "m"  (b[i])   // %2
      );
  }
  __asm__("emms" : : ); // Exit MMX Machine State
}
```

why 2?

Output C variable (referred as %0)

Input C variables (referred as %1, %2)

Alignment restrictions

- When using the "**movdqa**" instruction, **the memory address must be aligned to a 16-byte boundary**
- To achieve that, variables must be declared in C with a special syntax
- Example of a declaration of an integer array:

```
int a[size] __attribute__((aligned(16)));
```