



# SWAR: MMX, SSE, SSE 2 Multiplatform Programming

---

Relatore: dott. Matteo Roffilli  
roffilli@csr.unibo.it



# What's SWAR?

---

- ❑ SWAR = SIMD Within A Register
- ❑ SIMD = Single Instruction Multiple Data
- ❑ MMX, SSE, SSE2, Power3DNow =  
Intel\AMD implementation of SWAR

# Why SWAR?

---

- ❑ Enable parallel computing on commercial hardware
- ❑ Boost 2x your application at 0 € cost
- ❑ Enjoy yourself learning assembler 😊

# Why not SWAR?

---

- ❑ Hard low-level programming
- ❑ Require know out of assembler
- ❑ Not portable code (yet!)

# X86 Instruction Set

---

- Pentium I
- General Purpose Instruction
- X87 FPU Instruction
- System Instruction

# General Purpose Instruction

---

- ☐ Data transfer
- ☐ Binary integer arithmetic
- ☐ Decimal arithmetic
- ☐ Logic operations
- ☐ Shift and rotate
- ☐ Bit and byte operations
- ☐ Program control
- ☐ String
- ☐ Flag control
- ☐ Segment register operations

# X87 FPU Instruction

---

- ☐ Floating-point
- ☐ Integer
- ☐ Binary-coded decimal (BCD) operands

# SIMD Instruction

---

- SIMD: Single Instruction Multiple Data
  - MMX , SSE and SSE2 instruction
  - provides a group of instructions that perform SIMD operations on packed integer and/or packed floating-point data elements contained in the 64-bit MMX, the 128-bit XMM or 128-bit MMX registers.
  - enables increased performance on a wide variety of multimedia and communications applications.



# What's new in Pentium II

---

- Pentium II = Pentium I + MMX
- MMX : MultiMedia Extensions
  - 57 new instructions
  - Eight 64-bit wide MMX registers
  - Four new data types

# What's new in Pentium III

---

- Pentium III = Pentium II + SSE
- SSE : Streaming SIMD Extensions
- 70 new instructions
- three categories:
  - SIMD-Floating Point
  - New Media Instruction
  - Streaming Memory Instruction

# The implementation of SSE

---

- SSE has 128-bit architectural width
  - Double-cycling the existing 64-bit data paths.
  - Deliver a realized 1.5 – 2x speedup
  - Only have 10% die size overhead

# SSE in details 1

---

- ❑ Eight 128-bits registers, named XMM0 to XMM7. The principal data type used by SSE instructions is a packed single floating point operand; practically, four 32-bits encoded floats are packed in each of the eight XMM registers
- ❑ SSE instructions set includes some additional integer instructions that operate on classic 64-bits MMX registers (MM0 to MM7)

# SSE in details 2

---

- ☐ 1. Arithmetic instructions
- ☐ 2. Logical instructions
- ☐ 3. Compare instructions
- ☐ 4. Shuffle instructions
- ☐ 5. Conversion instructions
- ☐ 6. Data movement instructions
- ☐ 7. State management instructions
- ☐ 8. Additional SIMD integer instructions
- ☐ 9. Cacheability control instructions

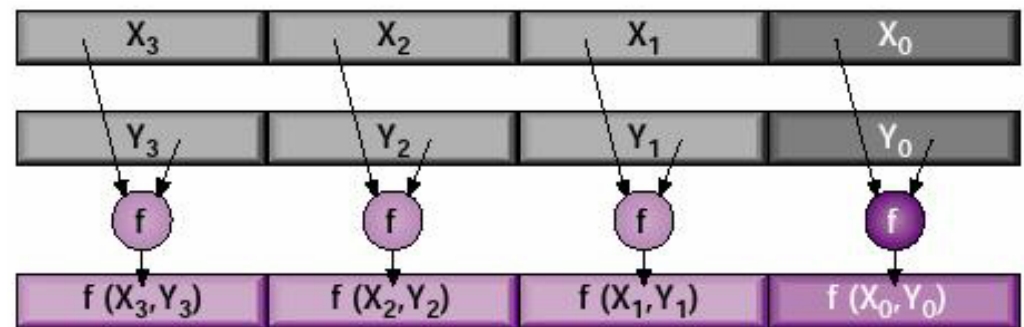
# What's new in Pentium 4

---

- Pentium 4 = Pentium III + SSE2
- SSE2-Streaming SIMD Extensions 2
  - 144 new instructions for Double Precision FP.
  - Cache and Memory Management.
  - Enable 3-D Graphics.
    - Video Decoding/Encoding/ Speech Recognition
- 128 bit MMX Registers.
- Separate Register for Data Movement

# SIMD-FP Instruction

- SIMD feature introduce a new register file containing eight 128-bit registers
  - Capable of holding a vector of four IEEE single precision FP data elements
  - Allow four single precision FP operations to be carried out within a single instruction



# Aligned or not aligned?

---

- Why use memory aligned?
  - Fast memory-to-register load
  - Require dedicated malloc function
  
- Why use memory not aligned?
  - Slow memory-to-register load
  - Fast integration into existing code
  - Not require dedicated malloc function



# Repetition helps learning!

---

## □ MMX

- 54 instructions for **INTEGER**

## □ SSE

- 70 instructions for **FLOAT**

## □ SSE2

- 144 instructions for **DOUBLE**

# Portable Code Writing

---

- At&T vs Intel syntax
- Linux vs Windows syntax
- GNU Gcc 2.95 vs MS Visual Studio 6

# How to write SWAR code?

---

- ☐ Intel compiler 6 Linux/Windows
- ☐ Direct Assembly
- ☐ Inline Assembly
- ☐ Link portable library (libSIMD)

# Real problem

---

- ❑ Intel Compiler: is not FREE!
- ❑ Direct Asm: too hard for newmbies!
- ❑ libSIMB: under developing yet!
- ❑ Inline Asm: not portable but easy!

# Linux Inline Assembly

---

```
1. __asm__ __volatile__(  
2. /* load vector a[1-4] in 128bit reg */  
3. "movups %0, %%xmm0\n\t"  
4. "movups %1, %%xmm1\n\t"  
  
5. /* multiply */  
6. "mulps %%xmm1, %%xmm0 \n\t"  
7. :  
8. : "m"(a[0]), "m"(a[4])  
9. );
```

# Windows Inline Assembly

---

```
1. __asm {  
2.     /* load vector a[1-4] in 128bit reg */  
3.     mov      edi, a  
4.     movups   xmm0, [edi]  
5.     movups   xmm1, [edi+16]  
  
6.     /* multiply */  
7.     mulps    xmm0, xmm1  
8.     }
```

# Side by side

---

- |  |                                      |
|--|--------------------------------------|
| 1. <code>__asm__ __volatile__</code> (     | 1. <code>__asm</code> {              |
| 2. <code>movups %0, %%xmm0\n\t"</code>     | 2. <code>mov edi, a</code>           |
| 3. <code>"mulps %%xmm1, %%xmm0\n\t"</code> | 3. <code>movups xmm0, [edi]</code>   |
| 4. <code>:</code>                          | 4. <code>mulps     xmm0, xmm1</code> |
| 5. <code>:"m"(a[0]),"m"(a[4])</code>       |                                      |
| 6. <code>);</code>                         | 5. <code>}</code>                    |

# How to compile?

---

- ☐ Kernel 2.4.x
- ☐ MS Visual Studio 6
- ☐ Gcc 2.95 / 3.x
- ☐ Assembly  
Power Pack 6
- ☐ Binutils >2.13



# How to debug?

---

☐ Directly in assembler! ☹️

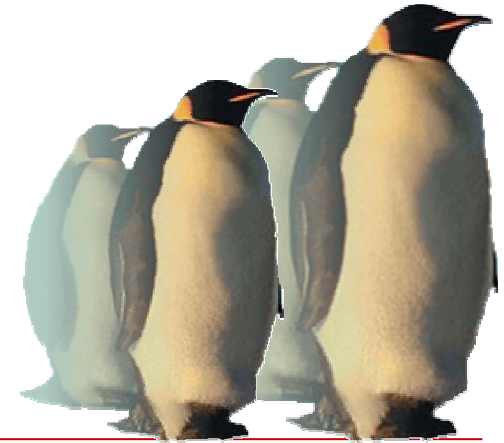
# Online Resource

---

- ☐ Intel tutorial
  - ☐ AMD tutorial and source code
  - ☐ Sourceforge.net
  - ☐ IBM whitepapers
  - ☐ MS tutorial
- 
- ☐ SPRITE Linux Day Proceeding

# THANKS FOR YOUR ATTENTION!

---



## Some questions?

Relatore: dott. Matteo Roffilli  
[roffilli@csr.unibo.it](mailto:roffilli@csr.unibo.it)