

MMX

MMX es un conjunto de instrucciones SIMD (Single Instruction Multiple Data) creadas por Intel. La tecnología MMX fue diseñada para mejorar de forma sustancial el rendimiento de las aplicaciones multimedia y de telecomunicaciones. Se basa en un nuevo juego de instrucciones, que se añadían a las ya existentes en la arquitectura 80x86, y en nuevos tipos de datos de 64 bits. Dichas instrucciones trabajaban en paralelo sobre múltiples elementos de datos empaquetados en cantidades de 64 bits.

SSE (Streaming SIMD Extensions)

Las SSE (SSE2, SSE3, SSSE3, SSE4 (4.1, 4.2, a)) son unas extensiones al grupo de instrucciones MMX, que también fueron desarrolladas por Intel, concretamente, para sus procesadores Pentium III (1999). Están especialmente pensadas para decodificación de MPEG2 (códec vinculado a los DVD), procesamiento de gráficos tridimensionales y software de reconocimiento de voz. Con la tecnología SSE, los microprocesadores x86 fueron dotados de setenta nuevas instrucciones y de ocho registros nuevos: del xmm0 al xmm7. Estos registros tienen una extensión de 128 bits (es decir que pueden almacenar hasta 16 bytes de información cada uno).

NET Bubble Sort SSE (comparando assembler + C)

- ¿Qué hace el programa?

```
Ordena arrays con contenido aleatorio de valor double, usando el algoritmo
BubbleSort implementado en C y en Inline Assembly. Se ordenan arrays de longitud
2 cuya longitud crece exponencialmente después de cada iteración hasta tener la
longitud 66537, y muestra el tiempo transcurrido en cada caso, comparando ambas
implementaciones.
```

- ¿Se utilizan extensiones MMX o SSE?

```
SSE, ya que se pueden reconocer instrucciones del set SSE2 tales como minpd,
maxpd, movapd...
```

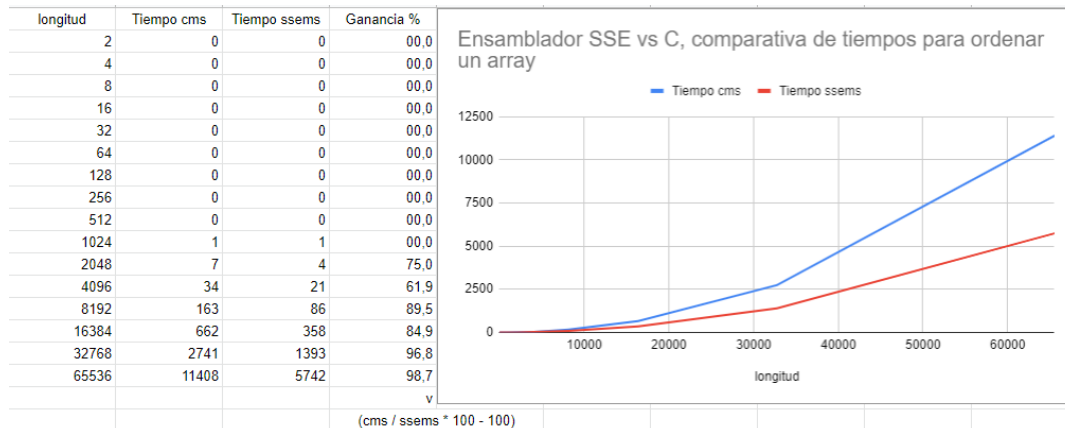
- Comenta el funcionamiento de la función: void sortDoublesSSE(int32 byteCount, double* values)

```
// TO DO
```

- ¿Qué ganancia obtenemos con el algoritmo MMX/SSE con respecto al algoritmo secuencial?

```
A medida que la longitud del array va aumentando, podemos observar que la
ganancia se va estabilizando en el 100%, siendo la implementación en SSE el doble
de rápida que en C.
```

- Realiza una batería de pruebas y muéstralo utilizando gráficas explicativas.



NET Data Transfer (comparando assembler + C)

- ¿Qué hace el programa?

Medir cuanto tarda el sistema en transferir datos (4 bytes (sizeof: int)) de un registro de memoria a otro una cantidad de veces (definido en ITERATION), implementado en C y en Inline Assembly, mostrando la diferencia entre ambos. Además permite reejecutar dicha prueba para obtener unos datos concluyentes sobre este.

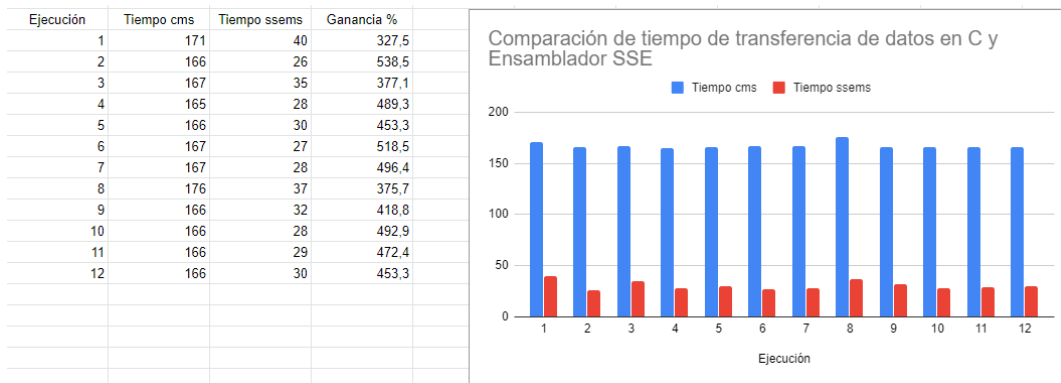
- Explica las siguientes instrucciones: shufpd, cmpltpd, movmskpd
 - shufpd: Baraja dos pares de valores de precisión de coma flotante de mxx1 y mxx2, y devuelve el resultado a mxx1.
 - cmpltpd: Compara si dos valores de precisión de coma flotante son "menores que" entre si.
 - movmskpd: Extrae la máscara de signo de un valor de precisión de coma flotante en mmx.
- Explica el funcionamiento de la siguiente función: int DataTransferOptimised(int* piDst, int* piSrc, unsigned long SizeInBytes)

```
// TO DO
```

- ¿Qué ganancia obtenemos con el algoritmo optimizado mediante extensiones SIMD con respecto al algoritmo secuencial?

Si comparamos los resultado tras varias iteraciones, podemos comprobar que la implementación en ensamblador llega a ser hasta 5 veces más rápida que la implementada en C.

- Realiza una batería de pruebas y muéstralo utilizando gráficas explicativas.



NET Inner Product (comparando assembler + C)

- ¿Qué hace el programa?

Suma la multiplicación de arrays rellenos de valores aleatorios de distinto tipo implementado de diferentes maneras, en C, en ensamblador usando instrucciones de SSE2 y otra con instrucciones de SS3.

- Comenta la siguiente función: float sse3_inner(const float* a, const float* b, unsigned int size)

```
// TO DO
```

- ¿Qué ganancia obtenemos con el algoritmo optimizado mediante extensiones SIMD con respecto al algoritmo secuencial?

Por lo que podemos ver segun los resultados, la ganancia esta vez depende del set que se usa y sobre que tipo de valor se usa, aún así, el resultado es el de siempre, favorable para la implementacion en ensamblador, los calculos realizados con floats parecen ser el triple de rapidos para el set sse3, y en el resto de tipos parece llegar a duplicar la velocidad de la implementacion en C.

- Realiza una batería de pruebas y muéstralo utilizando gráficas explicativas.

NET MatrixVectorMult (comparando assembler + C)

- ¿Que hace el programa?
- ¿Qué ganancia obtenemos con el algoritmo optimizado mediante extensiones SIMD con respecto al algoritmo secuencial?
- Realiza una batería de pruebas y muéstralo utilizando gráficas explicativas.

Fuentes:

- <https://books.google.es/books?id=bt3ZjeB4v9gC&printsec=frontcover&hl=es#v=onepage&q&f=false>
- <https://books.google.es/books?id=c3SlgrgMid4C&printsec=frontcover&hl=es#v=onepage&q&f=false>
- https://books.google.es/books?id=C3_WIQOYE2EC&printsec=frontcover&hl=es#v=onepage&q&f=false
- <https://books.google.es/books?id=FGxOp0IAjUC&printsec=frontcover&hl=es#v=onepage&q&f=false>

- <http://softpixel.com/~cwright/programming/simd/>
- https://docs.oracle.com/cd/E18752_01/html/817-5477/eojdc.html
- https://docs.oracle.com/cd/E26502_01/html/E28388/eojde.html
- <http://www.jegerlehner.ch/intel/IntelCodeTable.pdf>
- https://cs.brown.edu/courses/cs033/docs/guides/x64_cheatsheet.pdf

Preguntas:

- Porque en data transfer se multiplica por size(int) y despues se divide?
- Porque en Inner product, la funcion "sse_inner_d" y "sse_inner" no está realmente implementada en ningun bloque __asm?