Microprocesadores x86-32/x86-64

Diseño Basado en Microprocesadores

Víctor Manuel Sánchez Corbacho

Dpto. de Automática, Electrónica, Arquitectura y Redes de Computadores

2017

Contenido

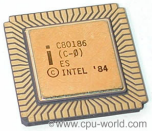
1. Evolución resumida (modelos más destacados de Intel)
2. Modos de operación
3. Registros de enteros
4. Tipos de datos enteros
5. Memoria
6. Modos de direccionamiento
7. Unidad de punto flotante FPU
8. Microprocesadores x86 de 64 bits
9. Extensiones SIMD

Evolución (1/8)

- **8086**
  - Año de introducción: 1978.
  - Extensión a 16 bits de las arquitecturas de 8 bits existentes (8080, 8085).
  - Bus de datos de 16 bits.
  - Bus de direcciones de 20 bits (1 Mbyte de memoria).
  - Frecuencia de reloj: 5 a 10 MHz.
  - Posibilidad de coprocesador matemático 8087.

- **8088**
  - Año de introducción: 1979.
  - Versión del 8086 con bus de datos de 8 bits.
  - Permitió reutilizar los componentes para micros de 8 bits.
  - Bus de direcciones de 20 bits.
  - IBM lo elige para construir el IBM PC.

- **80186/80188**
  - Año de introducción: 1982.
  - Integra un 8086/8088 más circuito generador de reloj, temporizadores, controlador de interrupciones y controlador de DMA.



=www.cpu-world.com

Evolución (2/8)

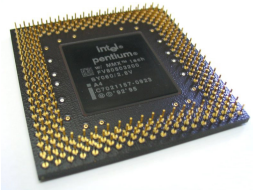
- **80286**
  - Año de introducción: 1982.
  - Arquitectura de 16 bits.
  - Bus de datos de 16 bits.
  - Bus de direcciones de 24 bits (16 Mbytes de m. física).
  - Frecuencia de reloj: de 5 a 20 MHz.
  - Introduce el modo protegido:
    - Hasta 1 Gbyte de memoria virtual.
    - Multitarea.
  - Chip coprocesador matemático 80287.

- **80386**
  - Año de introducción: 1985.
  - Inaugura la arquitectura Intel de 32 bits (IA-32).
  - Bus de datos de 32 bits.
  - Bus de direcciones de 32 bits (4 Gbytes m. física).
  - Frecuencia de reloj: de 12.5 a 40 MHz.
  - Modo protegido mejorado:
    - Hasta 64 Tbytes de memoria virtual.
    - Multitarea.
    - Modo virtual 8086 que permite ejecutar programas 8086 en un entorno multitarea.
  - Chip coprocesador matemático 80387.

Evolución (3/8)

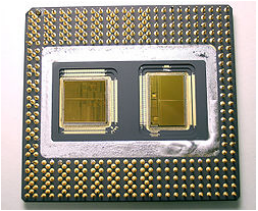
- **80486**
  - Año de introducción: 1989.
  - Arquitectura de 32 bits.
  - Bus de datos de 32 bits.
  - Bus de direcciones de 32 bits.
  - Frecuencia de reloj: 25 a 50 MHz.
  - Arquitectura segmentada (pipelining).
  - Cache interna de 8 Kbytes.
  - Unidad de punto flotante integrada.
  - 80486SX: Versión sin unidad de punto flotante.
  - 80486SL: Versión para portátiles.
  - 80486DX2, 80486DX4: Núcleo del procesador funciona a mayor frecuencia que el bus.

- **Pentium**
  - Año de introducción: 1993.
  - Arquitectura de 32 bits.
  - Bus de datos externo de 64 bits
  - Bus de direcciones de 32 bits.
  - Frecuencia de reloj: 60 a 200 MHz.
  - Arquitectura superescalar: dos pipelines.
  - Cache integrada de 16 Kbytes (8 Kbytes instrucciones + 8 Kbytes datos).
  - Protocolo de cache MESI para sistemas multiprocesador.
  - Versiones con extensiones MMX.



Evolución (4/8)

- **Pentium Pro**
  - Año de introducción: 1995.
  - Introduce la microarquitectura P6.
  - Núcleo RISC.
  - Pipeline de 12 etapas.
  - Bus de datos externo de 64 bits.
  - Bus de direcciones de 36 bits (64 Gbytes m. física).
  - Frecuencia de reloj: 150 a 200 MHz.
  - Soporte para multiprocesamiento simétrico (SMP).
  - Cache interna L1 de 32 Kbytes (16 Kbytes instrucciones + 16 Kbytes datos).
  - Cache L2 en un sustrato aparte conectada mediante un bus dedicado.



- **Pentium II**
  - Año de introducción: 1997.
  - Microarquitectura P6.
  - Bus de datos externo de 64 bits.
  - Bus de direcciones de 36 bits.
  - Frecuencia de reloj: de 233 a 400 MHz.
  - Incorpora extensiones MMX.
  - Encapsulado SEC.
  - Cache L1 de 32 Kbytes.
  - Cache L2 conectada a través de un bus dedicado.

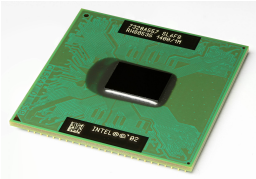
Evolución (5/8)

- **Pentium III**
  - Año de introducción: 1999.
  - Microarquitectura P6.
  - Bus de datos externo de 64 bits.
  - Bus de direcciones de 36 bits.
  - Frecuencia de reloj: 450 MHz a 1.5 GHz.
  - Nuevas ext. multimedia SSE: SIMD p. flotante.
  - Cache interna L1 de 32 Kbytes.
  - Cache externa L2 de 512 Kbytes.

- **Pentium 4**
  - Año de introducción: 2000.
  - Microarquitectura NetBurst.
  - Bus de datos externo de 64 bits
  - Bus de direcciones de 36 bits.
  - Frecuencia de reloj: 1.3 a3.8 Ghz.
  - Las ALUs funcionan al doble de frecuencia.
  - Hiperpipeline de 20 etapas.
  - Ampliación de las extensiones multimedia: SSE2.
  - Cache interna L1 de 32 Kbytes.
  - Cache L2 de 256 Kbytes conectada a través de un bus de 256 bits.
  - Versiones con "Hyper-Threading", mayores frecuencias de bus y cache L2.

Evolución (6/8)

- **Pentium M**
  - Año de introducción: 2003
  - Microarquitectura: variante de P6.
  - Optimizado para bajo consumo.
  - Equipos portátiles.
  - Frecuencia de reloj: 900 MHz a 2.26 Ghz.



- **Intel Core**
  - Año de introducción: 2006
  - Microarquitectura: variante de P6 ya usada en el Pentium M.
  - Optimizado para bajo consumo.
  - Dos versiones:
    - Core Solo: un sólo núcleo.
    - Core Duo: dos núcleos.
  - Cache L2 de 2MB.
  - Frecuencia de reloj: 1.06 GHz a 2.33 Ghz.

Evolución (7/8)

- **Intel Core 2**
  - Año de introducción: 2006
  - Microarquitectura: Core.
  - Arquitectura de 64 bits.
  - Juego de instrucciones x86-64.
  - Optimizaciones en la decodificación de instrucciones, unidades de ejecución y buses dan mayor rendimiento a menores frecuencias de reloj y menores consumos.
  - Versiones:
    - Solo: un sólo núcleo.
    - Duo: dos núcleos.
    - Quad: cuatro núcleos.
  - Tecnología de virtualización: permite que un S.O. "invitado" se ejecute bajo un S.O. "anfitrion" sin modificación del "invitado".
  - Frecuencia de reloj: 1.6 GHz a 3.33 Ghz.

Evolución (8/8)

- **Intel Core i3, i5, i7**
  - Año de introducción: 2008
  - Actualmente (2015), 6ª generación.
  - Arquitectura de 64 bits.
  - Juego de instrucciones x86-64.
  - Mejor relación prestaciones/consumo que Intel Core 2.
  - 7 nuevas instrucciones SSE.
  - Versiones con 2, 4, 6 y 8 núcleos.
  - Gráficos integrados en el mismo chip.
  - Controlador de memoria DDR3/DDR4 integrado.
  - Tres líneas de productos en tres escalones de prestaciones.
  - i5, i7:
    - Nuevas instrucciones para encriptación/desencriptación AES.
    - Nuevas instrucciones AVX de tipo SIMD de 256 bits.
    - Turbo Boost: "Overclocking" dinámico puntual a petición del S.O.

Alcance de nuestro estudio de la familia x86-32/x86-64

- Los microprocesadores x86-32/x86-64 actuales son muy complejos.
- El tiempo disponible impide tratar todos los detalles.
- Enfoque para una aplicación práctica inmediata.
- **Temas que se tratarán:**
  - Arquitectura interna para el programador de aplicación.
  - Programación en ensamblador del nivel de aplicación.
  - Uso de ensamblador desde lenguaje C/C++.
- **Temas que no se tratarán:**
  - Aspectos hardware internos o externos.
  - Programación del nivel de sistema.

¿Por qué aprender programación en ensamblador?

- En comparación con los lenguajes usados actualmente, que permiten construir programas multiplataforma y desarrollados en poco tiempo, el lenguaje ensamblador tiene desventajas:
  - Es poco portable (depende de la CPU usada).
  - Es lento de escribir.
  - Es propenso a errores.
- Así que, ¿por qué aprender ensamblador?
  - Permite obtener las máximas prestaciones de la CPU.
  - Permite usar características de la CPU no accesibles desde lenguajes de alto nivel.
  - Exige conocer bien cómo funciona la CPU. Esto ayuda a crear programas en lenguajes de alto nivel más eficientes.

Nombres genéricos

- **x86**:
  - Todos los descendientes del 8086 hasta la actualidad.
- **IA-32/x86-32**:
  - Arquitectura x86 de 32 bits.
    - Intel la denomina IA-32 (Intel Architecture 32 bits).
- **AMD64/Intel 64/x86-64/x64**:
  - Arquitectura x86 de 64 bits.
    - AMD la denomina AMD64.
    - Intel la denomina Intel 64.
    - Denominaciones genéricas: x86-64/x64.

Modos de operación de un microprocesador x86-64

- **Long mode (AMD) / IA-32e mode (Intel)**:
  - Modo de ejecución de un S.O. de 64 bits.
  - Tiene dos submodos:
    - **Modo 64-bit**: modo de ejecución de las aplicaciones de 64 bits en un S.O. de 64 bits.
    - **Modo de compatibilidad**: permite ejecutar aplicaciones de 16 y 32 bits en un S.O. de 64 bits sin necesidad de recompilación.
- **Legacy mode (AMD) / IA-32 mode (Intel)**:
  - Modo de compatibilidad con S.O. de 32 bits y aplicaciones de 32 y 16 bits.
  - Tiene tres submodos:
    - **Modo real**.
    - **Modo protegido**.
    - **Modo de gestión de sistema (SMM)**.

Registros del 8086 original

- Pocos registros comparado con otros microprocesadores de 16 bits.
- Especialización de los registros. Arquitectura no ortogonal.
- Cuatro tipos:
  - Registros de datos:

    AX, BX, CX, DX
  - Registros punteros e índices:

    IP, SP, BP, SI, DI
  - Registros de segmento:

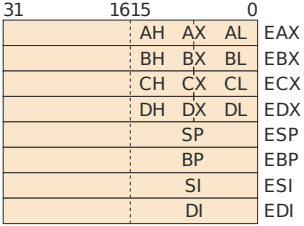
    CS, DS, ES, SS
  - Registro de indicadores:

    FLAGS

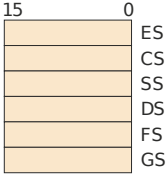
CF: acarreo. Desbordamiento en aritmética sin signo.
PF: paridad. A 1 si byte bajo del resultado tiene un n° par de unos, a 0 si impar.
AF: acarreo auxiliar. Acarreo del bit 3. Usado en aritmética BCD.
ZF: cero. A 1 cuando el resultado es cero, a 0 en caso contrario.
SF: signo. MSB del último resultado.
TF: trampa. Habilitación de la interrupción paso a paso para depuración.
IF: interrupción. Habilitación de interrupciones enmascarables externas.
DF: dirección. Dirección de procesamiento de cadenas.
OF: desbordamiento. Desbordamiento en aritmética en complemento a dos.

Registros de enteros en modo de 32 bits

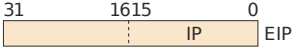
REGISTROS DE "PROPÓSITO GENERAL"



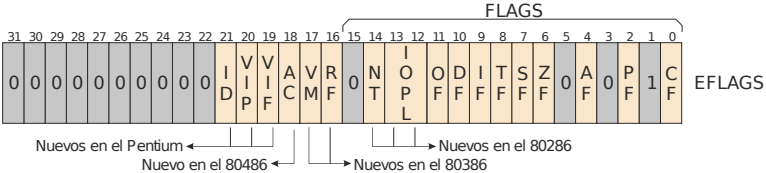
REGISTROS DE SEGMENTO



PUNTERO DE INSTRUCCIÓN



REGISTRO DE INDICADORES



Todos los registros del nivel de aplicación en modo 32 bits

Registros de "propósito general"

EAX
EBX
ECX
EDX
ESI
EDI
EBP
ESP

Registro de estado y control

EFLAGS

Puntero de instrucción

EIP

Registros de segmento

CS
DS
SS
ES
FS
GS

Registros de la FPU/MMX

R0/MMX0
R1/MMX1
R2/MMX2
R3/MMX3
R4/MMX4
R5/MMX5
R6/MMX6
R7/MMX7

Registro de datos SSE/AVX

XMM0/YMM0
XMM1/YMM1
XMM2/YMM2
XMM3/YMM3
XMM4/YMM4
XMM5/YMM5
XMM6/YMM6
XMM7/YMM7

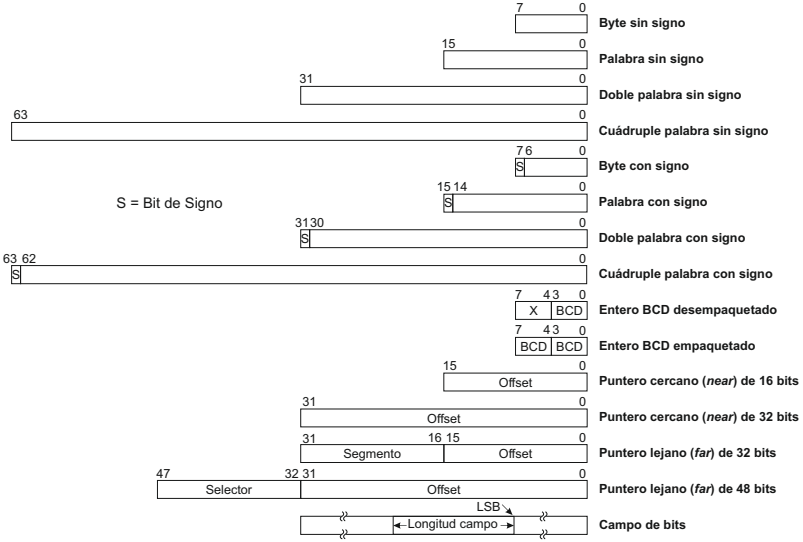
Registros de estado, control y tags de la FPU

Estado y control SSE/AVX

MXCSR

El tamaño de los registros no está a escala.

Tipos de datos enteros

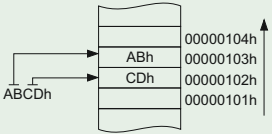


Características generales del sistema de memoria

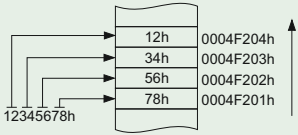
- Arquitectura de memoria de tipo **Von Neumann**:
  - Array continuo de elementos direccionables común para datos e instrucciones.
- **Elemento direccionable: byte.**
- En general, los datos **no necesitan estar alineados**, aunque es conveniente para acelerar la ejecución.
- Los datos se almacenan en la memoria ordenados en formato ***little endian***.

Almacenamiento en formato *little endian*

- Palabra 0xABCD almacenada a partir de la dirección 0x00000102.

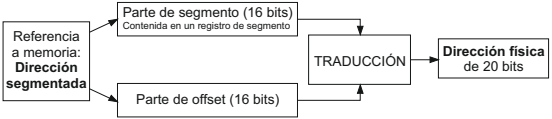


- Doble palabra 0x12345678 almacenada a partir de la dirección 0x0004F201.

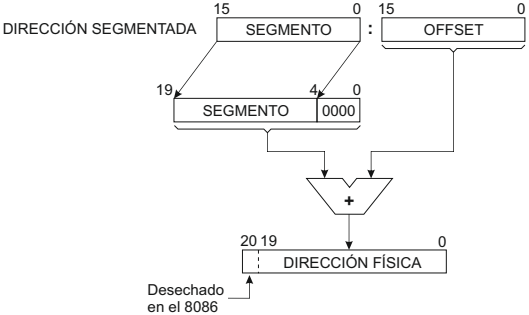


Acceso a memoria en modo real (I)

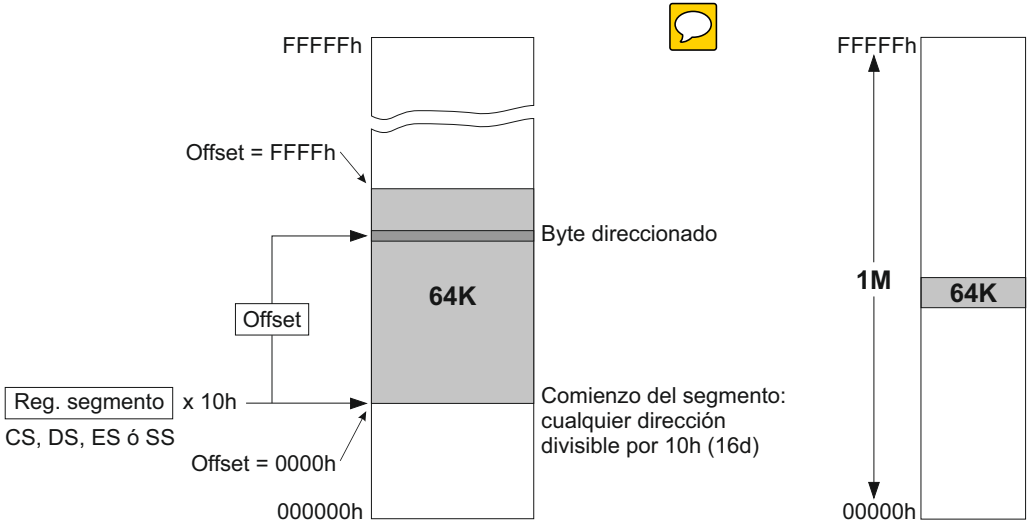
- Segmentación en modo real.



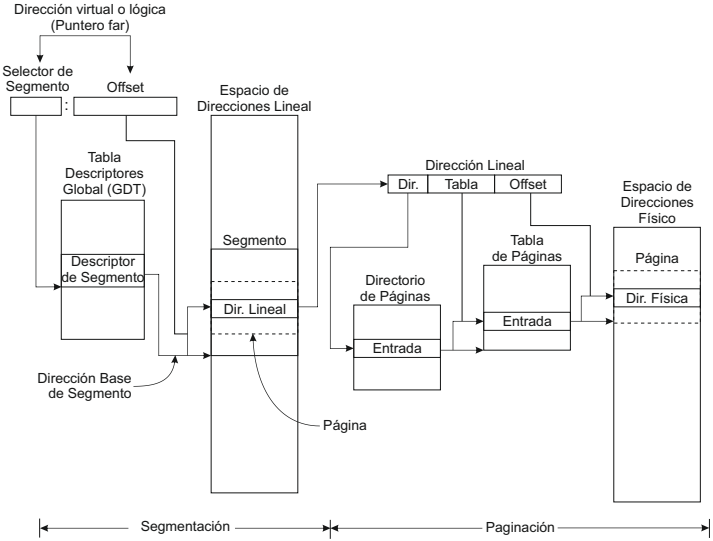
- Traducción de direcciones en modo real.



Acceso a memoria en modo real (II)



Acceso a memoria en modos de compatibilidad y protegido



Modos de direccionamiento

- **De registro**.
- **Inmediato**.
- **Implícito**.
- Modos de direccionamiento **de memoria de 32 bits**. Se usan en:
  - Modo Legacy (AMD) / IA-32 (Intel).
  - Submodo de compatibilidad del modo Long (AMD) / IA-32e (Intel).
- Modos de direccionamiento **de memoria de 64 bits**.
  - Usados en el modo de 64 bits.

Modos de direccionamiento

- De registro

```
mov     al, ah
mov     ebx, ecx
mov     rax, rdx
```

- Inmediato

```
mov     cl, 35
mov     edx, 0x0F7ABD5E4
mov     rax, 23
```

- Implícito

```
daa                 ; Operando AL implícito.
mul     ecx         ; Operandos EAX y EDX implícitos.
loop    etiqueta    ; Operando ECX/RCX implícito.
```

Modos de direccionamiento de memoria 32 bits (I)

- Direccionammiento de memoria **directo**
  - La instrucción codifica directamente una dirección efectiva de 32 bits.

```
mov ecx, [1000]
mov ax, [variable] ; Usando una etiqueta.
```

  - Por defecto, el registro **DS** proporciona el segmento.

Modos de direccionamiento de memoria 32 bits (II)

- Direccionamiento de memoria **indirecto por registro(s)**

Base		Índice		Escala		Desplazamiento (con signo)
Ninguno	+	Ninguno	*	1	+	Ninguno
EAX		EAX		2		8 bits
ECX		ECX		4		32 bits
EDX		EDX		8		
EBX		EBX				
ESP		---				
EBP		EBP				
ESI		ESI				
EDI		EDI				
Elegir uno		Elegir uno		Elegir uno		Elegir uno

Offset = Base + Indice*Escala + Desplazamiento

```
mov  eax,[eax]                ; Sólo base.
mov  [ebp + eax*4], bx        ; Base e índice escalado.
mov  [ecx + esi], edx         ; Si la escala es 1 puede obviarse.
mov  edi, [esi*2 - 3]         ; Índice escalado y desplazamiento.
mov  cl, [esi*2]              ; Sólo índice escalado.
mov  [ebx + ecx*8 + 1000], dh ; Todos los elementos.
```

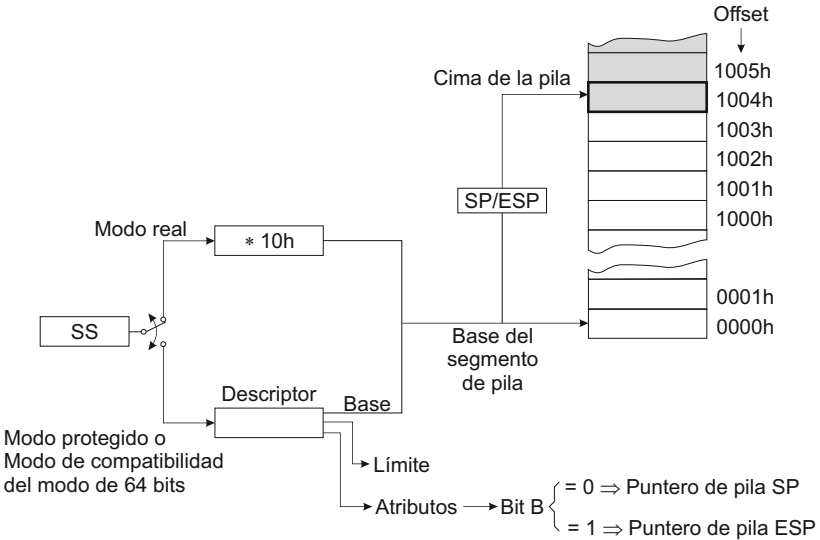
Registro de segmento por defecto

- Registro de segmento por defecto:
  - **DS**: Si no se usan los registros ESP o EBP como base.
  - **SS**: Si se usa el registro ESP o EBP como base.

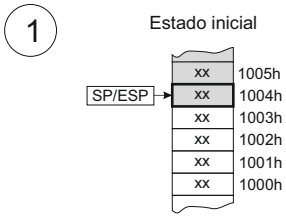
- Puede usarse un registro de segmento distinto al que selecciona por defecto el microprocesador anteponiendo el nombre del registro que se desea utilizar, seguido de dos puntos, al operando de memoria.

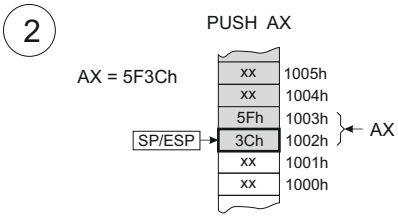
```
mov  al,[es:ebx + edi - 89] ; Cargar AL con el byte en el offset
                            ; EBX + EDI - 89 del segmento designado
                            ; por ES (segmento de datos extra).

mov  [cs:ebp + 71],cx       ; Almacenar CX a partir de la posición
                            ; de offset EBP + 71 del segmento
                            ; designado por CS (segmento de código).
```

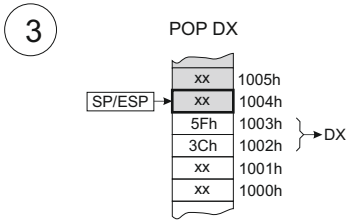
Acceso a la pila en modo real y modos de 32 bits

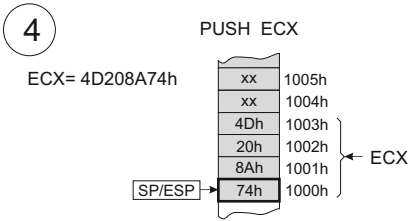


Efecto de las instrucciones PUSH y POP sobre la pila



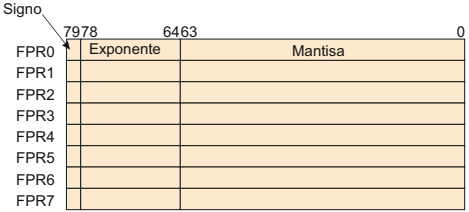






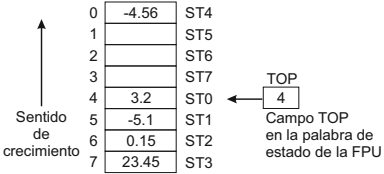
Registros de la FPU

REGISTROS DE DATOS



ORGANIZACIÓN EN FORMA DE PILA DE LOS REGISTROS DE DATOS

ORGANIZACIÓN INTERNA



REGISTROS DE CONTROL

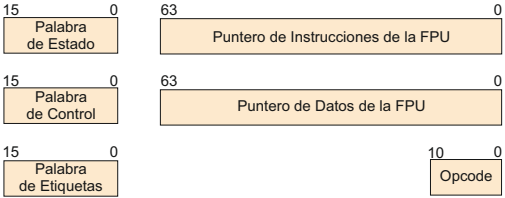
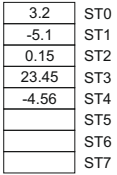
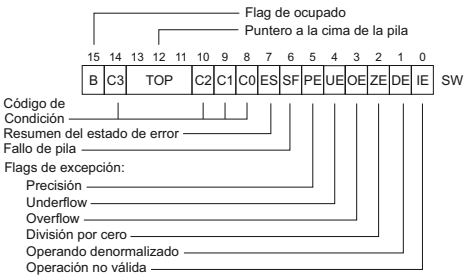


IMAGEN PARA EL PROGRAMADOR

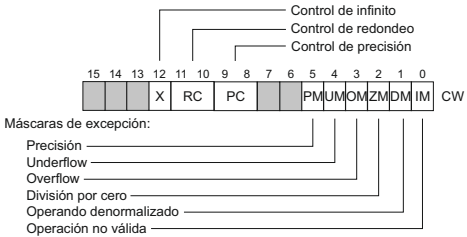


Registros de control de la FPU

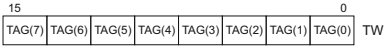
PALABRA DE ESTADO (STATUS WORD)



PALABRA DE CONTROL (CONTROL WORD)

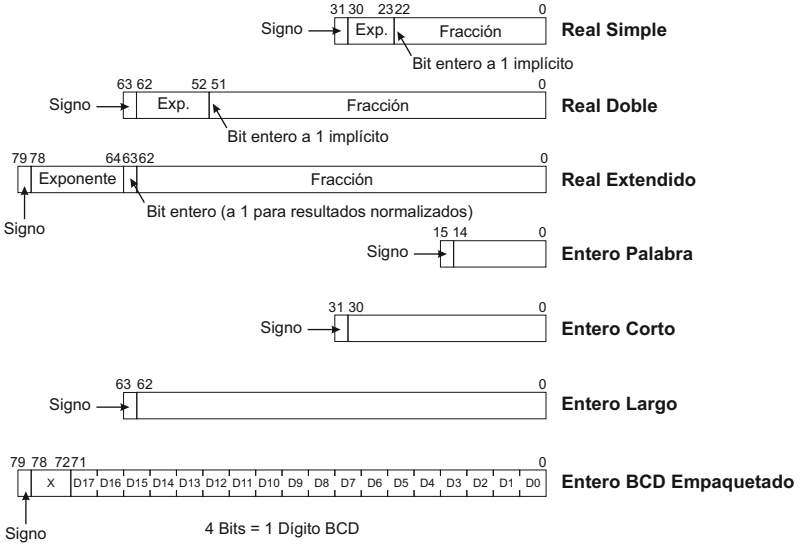


PALABRA DE ETIQUETAS (TAG WORD)



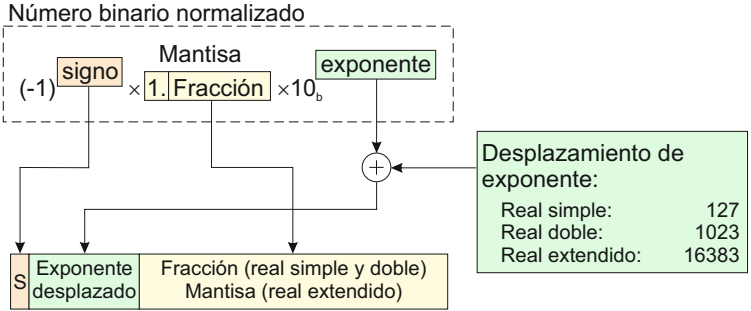
Valores de TAG:
- 00 - Válido
- 01 - Cero
- 10 - Especial: no válido (NaN, no soportado), infinito o denormalizado
- 11 - Vacío

Tipos de datos de punto flotante



Codificación de datos de punto flotante

- Los números reales se representan internamente como números binarios en punto flotante normalizados según el estándar IEEE 754.

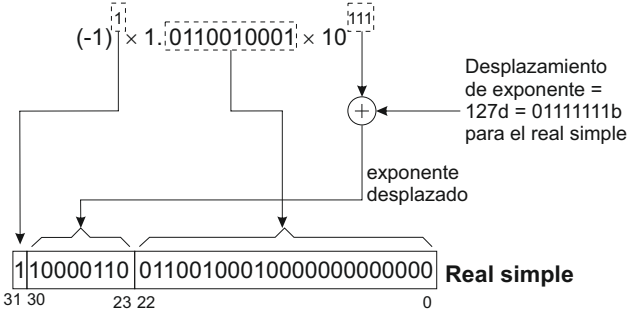


Ejemplo de codificación de un real simple (float)

- Codificación del número decimal -178.125 en un real simple (float).

$$-178.125_d = -10110010.001_b =$$

$$= (-1)^1 \times 10110010.001_b \times 10_b^0 \xrightarrow{\text{Normalización}} (-1)^1 \times 1.0110010001_b \times 10_b^{111_b}$$



En hexadecimal: C3322000h

Microprocesadores x86 de 64 bits

- Introducida por AMD: AMD-64.
- Imitada por Intel: Intel 64.
- Extensión del conjunto de registros:
  - 8 registros de propósito general adicionales: 16 en total.
  - Los 16 los registros de propósito general son de 64 bits.
  - 8 registros XMM adicionales: 16 en total.
- Ampliación de la capacidad de direccionamiento:
  - Direcciones virtuales de 64 bits.
  - Puntero de instrucción RIP de 64 bits.
  - Modo de direccionamiento relativo al puntero de instrucción.

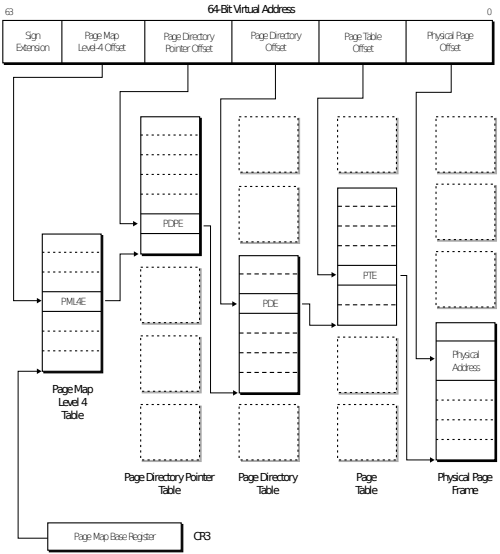
Modos de operación

Operating Mode		Operating System Required	Application Recompile Required	Defaults		Register Extensions	Typical
				Address Size (bits)	Operand Size (bits)		GPR Width (bits)
Long Mode	64-Bit Mode	64-bit OS	yes	64	32	yes	64
	Compatibility Mode		no	32		no	32
				16	16		16
Legacy Mode	Protected Mode	Legacy 32-bit OS	no	32	32	no	32
				16	16		
	Virtual-8086 Mode			16	16		16
	Real Mode	Legacy 16-bit OS					

Acceso a memoria en modo de 64 bits

- Las direcciones virtuales generadas por las instrucciones son cantidades de 64 bits.
- Pero los procesadores actuales generan direcciones físicas de *sólo* 48 bits (256 TBytes).
- En modo de 64 bits no hay segmentación, sólo paginación.
  - El registro de segmento CS se usa de forma muy limitada. DS, ES, SS se ignoran. FS y GS pueden seguir usándose pero cambian su funcionamiento.

Traducción de dirs. virtuales a físicas en modo 64 bits



Todos los registros del nivel de aplicación en modo 64 bits

Registros de propósito general (64 bits)

RAX
RBX
RCX
RDX
RSI
RDI
RBP
RSP
R8
R9
R10
R11
R12
R13
R14
R15

Registro de estado y control (64 bits)

RFLAGS

Puntero de instrucción (64 bits)

RIP

Regs. de segmento (16 bits)

CS
DS
SS
ES
FS
GS

Registro de datos SSE/AVX (128/256 bits)

XMM0/YMM0
XMM1/YMM1
XMM2/YMM2
XMM3/YMM3
XMM4/YMM4
XMM5/YMM5
XMM6/YMM6
XMM7/YMM7
XMM8/YMM8
XMM9/YMM9
XMM10/YMM0
XMM11/YMM11
XMM12/YMM12
XMM13/YMM13
XMM14/YMM14
XMM15/YMM15

El tamaño de los registros no está a escala.

Registros de la FPU/MMX (80/64 bits)

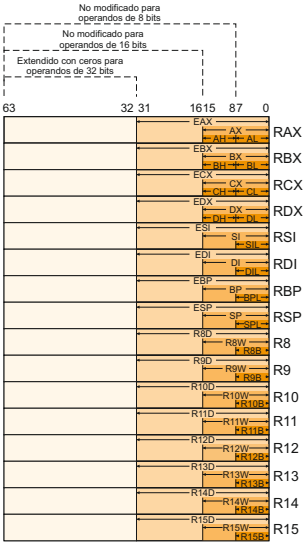
FPR0/MM0
FPR1/MM1
FPR2/MM2
FPR3/MM3
FPR4/MM4
FPR5/MM5
FPR6/MM6
FPR7/MM7

Registros de estado, control y tags de la FPU

Estado y control SSE/AVX (32 bits)

MXCSR

Registros de propósito general en modo de 64 bits



Registro de 32 bits

Registro de 16 bits

Registro de 8 bits

- AMD llama R8B, R9B, R10B, R11B, R12B, R13B, R14B, R15B a los bytes bajos de los registros R8 a R15.

- Intel llama R8L, R9L, R10L, R11L, R12L, R13L, R14L, R15L a los bytes bajos de los registros R8 a R15.

- El ensamblador acepta las dos nomenclaturas.

Uso de registros de 8 bits en modo de 64 bits

- No se pueden emplear en la misma instrucción uno de los registros de 8 bits *altos* (AH, BH, CH, DH) y uno de los nuevos registros de 8 bits (SIL, DIL, BPL, SPL y R8B a R15B).

```
mov     ah, r10b ; Incorrecto.
add     sil, dh  ; Incorrecto.
```

- Sí se pueden seguir empleando los registros AH, BH, CH, DH junto con cualquier otro registro original de 8 bits.

```
mov     ah, cl   ; Correcto.
sub     ch, dh   ; Correcto.
and     bl, dh   ; Correcto.
```

Operaciones con partes de 8, 16 ó 32 bits de un reg. de 64

- Al operar con una parte 8 ó 16 bits de un registro de 64 bits, los bits restantes del registro no cambian.
- Al operar con la parte baja de 32 bits de un registro de 64 bits, los 32 bits superiores del registro se ponen a 0.

Instrucción	Antes	Después
`mov al,0x68`	RAX = 0xAAAAAAAAAAAAAAAA	RAX = 0xAAAAAAAAAAAAAA68
`add al,bl`	RAX = 0xAAAAAAAAAAAAAAAA RBX = 0xBBBBBBBBBBBBBBBB	RAX = 0xAAAAAAAAAAAAAA65 RBX = 0xBBBBBBBBBBBBBBBB
`add cx,ax`	RAX = 0xAAAAAAAAAAAAAAAA RCX = 0xCCCCCCCCCCCCCCCC	RAX = 0xAAAAAAAAAAAAAAAA RCX = 0xCCCCCCCCCCCC7776
`mov eax,2`	RAX = 0xAAAAAAAAAAAAAAAA	RAX = 0x0000000000000002
`mov eax,-1`	RAX = 0xAAAAAAAAAAAAAAAA	RAX = 0x00000000FFFFFFFF
`add eax,ebx`	RAX = 0xAAAAAAAAAAAAAAAA RBX = 0xBBBBBBBBBBBBBBBB	RAX = 0x0000000066666665 RBX = 0xBBBBBBBBBBBBBBBB

Registros de 64 bits y constantes inmediatas

- Sólo MOV admite una constante inmediata de 64 bits.
- El resto de operaciones con registros de 64 bits sólo admiten constantes inmediatas de 8, 16 ó 32 bits.
  - La constante inmediata se **extiende en signo** antes de realizar la operación con el registro de 64 bits.

Código máquina	Instrucción	Valor en reg. destino
48 C7 C0 64 00 00 00	mov rax,100	0x0000000000000064
48 05 C0 C8 00 00 00	add rax,200	0x000000000000012C
48 C7 C1 30 F8 FF FF	mov rcx,-2000	0xFFFFFFFFFFFFF830
48 81 C1 E8 03 00 00	add rcx,1000	0xFFFFFFFFFFFFFC18
48 C7 C2 FF 00 00 00	mov rdx,0xff	0x00000000000000FF
48 81 CA 00 00 00 80	or rdx,0x80000000	0xFFFFFFFF800000FF
48 C7 C2 FF 00 00 00	mov rdx,0xff	0x00000000000000FF
49 B8 00 00 00 80 00 00 00 00	mov r8,0x80000000	0x0000000080000000
49 0B D0	or rdx,r8	0x00000000800000FF

Modos de direccionamiento de memoria indirectos

- Indirecto por registro(s). Formato general:

$$\textbf{Dirección} = \textbf{Base} + \textbf{Indice*Escala} + \textbf{Desplazamiento}$$

  - Base puede ser cualquier registro de 64 bits.
  - Índice puede ser cualquier registro de 64 bits excepto RSP.
  - Escala puede ser 1, 2, 4 u 8.
  - El desplazamiento es una cantidad con signo de 32 bits.

- Relativo al RIP

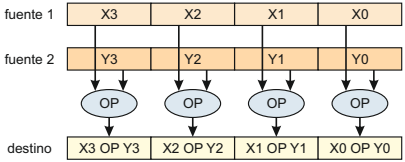
$$\textbf{Dirección} = \textbf{RIP} + \textbf{Desplazamiento}$$

  - El desplazamiento es una cantidad con signo de 32 bits.

```
mov     eax, [rip + 1000]
```

Extensiones SIMD

- Las extensiones SIMD (Single Instruction, Multiple Data) realizan simultáneamente una misma operación sobre un conjunto de datos.



- Aplicaciones:
  - Procesado digital de imágenes.
  - Procesado digital de vídeo.
  - Procesado digital de audio.
  - Reconocimiento de voz.
  - Gráficos 3D.

Extensiones SIMD de diferentes procesadores

- Diferentes fabricantes han desarrollado sus propias extensiones SIMD:
  - **MMX/SSE/AVX**: Intel/AMD.
  - **3DNow!**: AMD.
  - **AltiVec**: Apple/IBM/Motorola.
  - **DSP extensions/NEON**: ARM.

SIMD en los procesadores x86-32/x86-64 (1/5)

- **MMX (MultiMedia eXtensions)**
  - Introducida en el Pentium MMX y Pentium II (1997).
  - SIMD sobre **conjuntos de datos enteros empaquetados**.
- **SSE (Streaming SIMD Extensions)**
  - Introducida son Pentium III (1999).
  - SIMD con **datos en punto flotante de precisión simple (SPFP) empaquetados en 128 bits**.
  - Operaciones con datos **escalares SPFP**.
  - Instrucciones de control de las cachés.
  - Instrucciones de ordenamiento de accesos a memoria.

SIMD en los procesadores x86-32/x86-64 (2/5)

- **SSE2**
  - Introducida con el Pentium 4 (2000).
  - SIMD con **datos SPFP empaquetados en 128 bits**.
  - SIMD con **datos en p. flotante de doble precisión (DPFP) empaquetados en 128 bits**.
  - Operaciones con datos **SPFP escalares**.
  - Operaciones con datos **DPFP escalares**.
  - SIMD con datos **enteros empaquetados en 128 bits**.
  - Instrucciones de control de las cachés adicionales.
- **SSE3**
  - Introducida con la versión Presscott del Pentium 4 (2004).
  - Mismos tipos de datos que SSE2.
  - Suma y resta ***horizontales*** con datos **SPFP** y **DPFP empaquetados**.
  - Instrucciones de transferencia de datos mejoradas.

SIMD en los procesadores x86-32/x86-64 (3/5)

- **SSSE3**
  - Introducida con la microarquitectura Core en junio de 2006.
  - Mismos tipos de datos que SSE2.
  - Suma y resta ***horizontales*** con **enteros empaquetados**.
  - Instrucciones SIMD mejoradas con enteros empaquetados.
- **SSE4.1**
  - Anunciada en septiembre de 2006.
  - Mismos tipos de datos que SSE2.
  - **Producto escalar** de datos **SPFP** y **DPFP** empaquetados.
  - Instrucciones de **mezcla** de datos **SPFP** y **DPFP** empaquetados.
  - Instrucciones de inserción y extracción.
  - Instrucciones de mezcla de datos enteros empaquetados.

SIMD en los procesadores x86-32/x86-64 (4/5)

- **SSE4.2**
  - Introducida en la versión Nehalem del Core-i7.
  - Instrucciones de **procesamiento de cadenas de texto** empaquetadas.
  - Instrucciones de aceleración de **cálculo de CRC**.

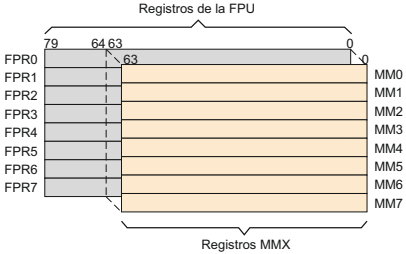
- **AVX (Advanced Vector Extensions)**
  - Introducida en las versiones Sandy Bridge del Core-i5/i7.
  - Datos **SPFP**, **DPFP** y **enteros empaquetados en 128 bits**.
  - Datos **SPPF** y **DPFP empaquetados en 256 bits**.
  - Datos **SPFP** y **DPFP escalares**.
  - **Instrucciones con 3 operandos**.
  - **Carga y almacenamiento condicional** de datos empaquetados.
  - **Difusión y permutación** de datos **SPFP** y **DPFP empaquetados**.
  - Nuevas instrucciones de comparación.
  - Conversión desde/hacia p.f. de media precisión.

SIMD en los procesadores x86-32/x86-64 (5/5)

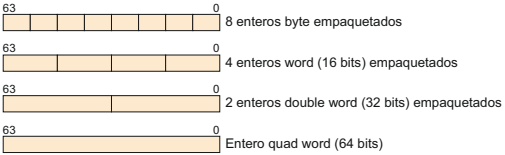
- **AVX2**
  - Introducida Introducida en las versiones Haswell del Core-i5/i7.
  - SIMD con datos **enteros empaquetados en 256 bits**.
  - Instrucciones de reunión de datos.
  - Instrucciones de difusión y permutación mejoradas.
  - Instrucciones de **multiplicación y suma fusionadas (FMA)**.
  - Instrucciones de manipulación de bits mejoradas.
  - Instrucciones de rotación y desplazamiento sin afectar flags.
- **AVX-512**
  - Introducida en el Xeon Knights Landing.
  - Datos **SPFP**, **DPFP** y **enteros empaquetados en 512 bits**.
  - Operaciones condicionales de elementos empaquetados.
  - Anulación de redondeo a nivel de instrucción.
  - Instrucciones de dispersión de datos.

Registros y tipos de datos MMX

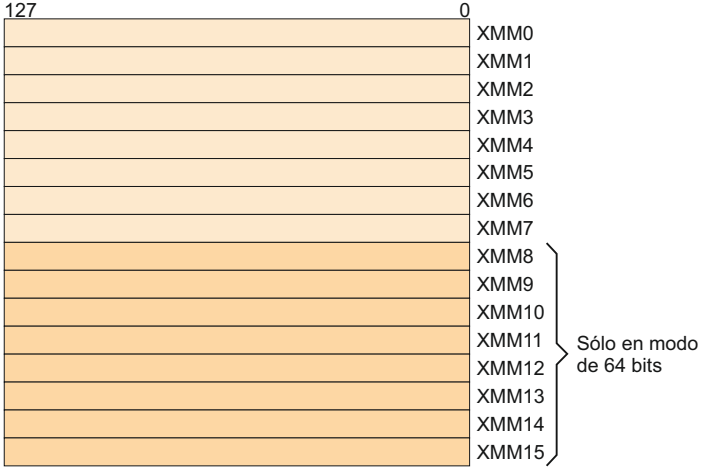
- Registros MMX



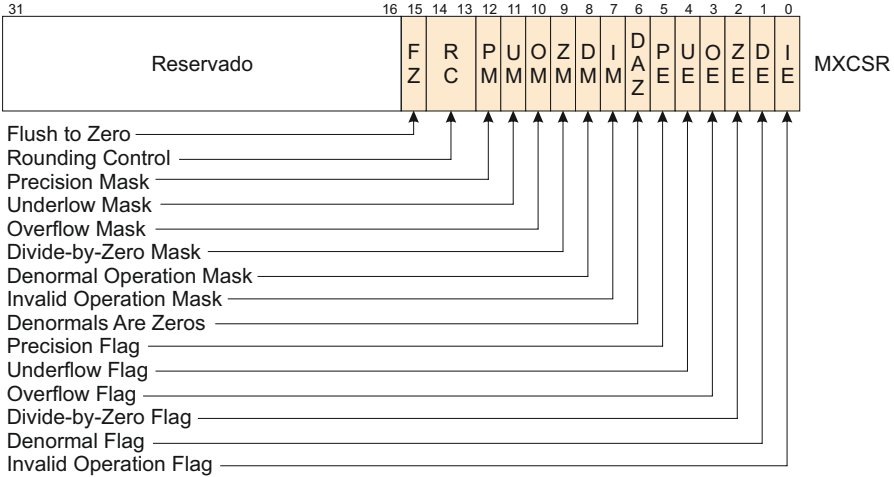
- Tipos de datos MMX



Registros de datos SSE

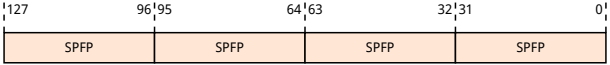


Registro de control y estado SSE

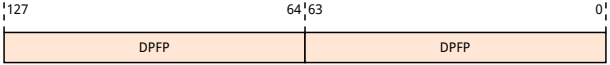


Tipos de datos SSE. Flotantes empaquetados

- Cuatro números en punto flotante de simple precisión (SPFP, Single Precision Floating Point) empaquetados en 128 bits.

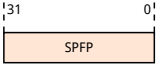


- Dos números en punto flotante de doble precisión (DPFP, Double Precision Floating Point) empaquetados en 128 bits.

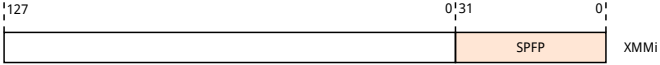


Tipos de datos SSE. Flotantes escalares

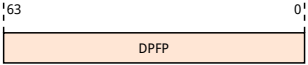
- Un número en punto flotante de simple precisión (SPFP).



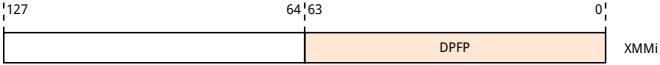
  - Se almacena en los 32 bits inferiores de los registros XMM.



- Un número en punto flotante de doble precisión (DPFP).

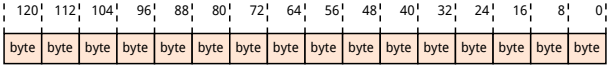


  - Se almacena en los 64 bits inferiores de los registros XMM.

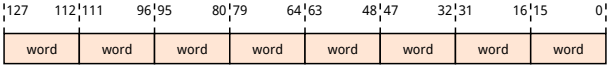


Tipos de datos SSE. Enteros empaquetados

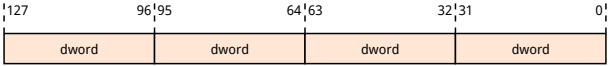
- 16 bytes empaquetados en 128 bits.



- 8 words empaquetados en 128 bits.



- 4 dwords empaquetados en 128 bits.



Indicación del tipo de dato en las instrucciones SSE (1/2)

- Las instrucciones SSE que operan con datos en punto flotante indican el tipo de datos usado mediante los sufijos:
  - **SS**: **S**calar **S**ingle precision (escalar de precisión simple).
  - **SD**: **S**calar **D**ouble precision (escalar de precisión doble).
  - **PS**: **P**acked **S**ingle precision (varios datos precisión simple empaquetados).
  - **PD**: **P**acked **D**ouble precision (varios datos precisión doble empaquetados).

```
addss xmm0,xmm1 ; Sumar a un número de precisión simple en XMM0 un número
                ; de precisión simple en XMM1.

addsd xmm0,xmm1 ; Sumar a un número de precisión doble en XMM0 un número
                ; de precisión doble en XMM1.

addps xmm0,xmm1 ; Sumar a 4 números de precisión simple empquetados en XMM0
                ; 4 números de precisión simple empquetados en XMM1.

addpd xmm0,xmm1 ; Sumar a 2 números de precisión doble empaquetados en XMM0
                ; 2 números de precisión doble empaquetados en XMM1.
```

Indicación del tipo de dato en las instrucciones SSE (2/2)

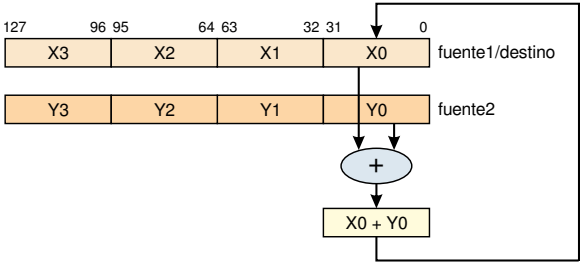
- Las instrucciones SSE que operan con datos enteros tienen el prefijo **P** (Packed) y los sufijos:
  - **B**: **B**ytes.
  - **W**: **W**ords.
  - **D**: **D**ouble words (dwords).
  - **Q**: **Q**uad words (qwords).

```
paddb xmm0,xmm1 ; Sumar a los 16 bytes empaquetados en XMM0 los 16 bytes
                ; empaquetados en XMM1.

paddw xmm0,xmm1 ; Sumar a los 8 words empaquetados en XMM0 los 8 words
                ; empaquetados en XMM1.

paddd xmm0,xmm1 ; Sumar a los 4 double words empaquetados en XMM0 los 4
                ; double words empaquetados en XMM1.

paddq xmm0,xmm1 ; Sumar a los 2 quad words empaquetados en XMM0 los 2 quad
                ; words empaquetados en XMM1.
```

Operaciones SSE con datos en punto flotante escalares

- Ejemplo: **ADDSS**: **ADD S**calar **S**ingle-Precision

```
ADDSS <fuente1/destino>, <fuente2>
```

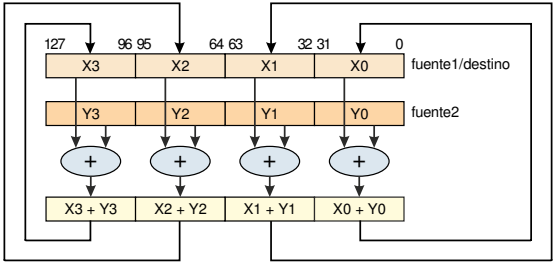


- La instrucción sólo usa los 32 o 64 bits en la parte baja de los datos de 128 bits situados en los registros XMM o la memoria.
- El resto de bits altos del operando destino no cambian.

Operaciones SSE con datos en punto flotante empaquetados

- Ejemplo: **ADDPS**: **ADD P**acked **S**ingle-Precision

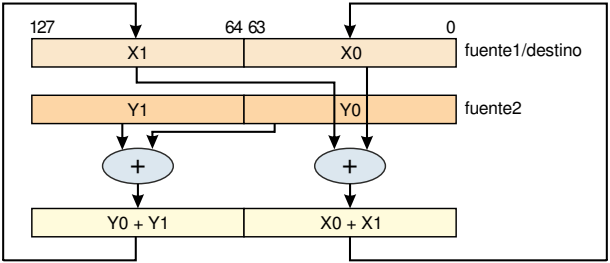
```
ADDPS <fuente1/destino>, <fuente2>
```



Instrucciones SSE horizontales

- Introducidas en SSE3.
- Ejemplo: **HADDPD** (**H**orizontal **ADD** **P**acked **D**ouble-Precision)

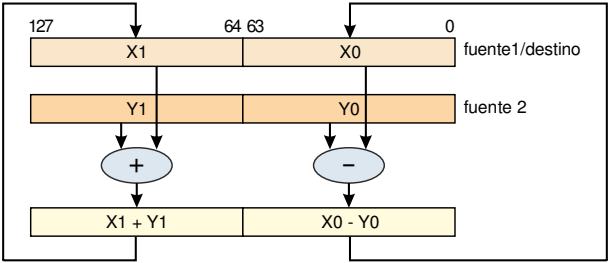
```
HADDPD <fuente1/destino>, <fuente2>
```



Instrucciones SSE asimétricas

- Introducidas en SSE3.
- Ejemplo: **ADDSUBPD** (**ADD SUB**tract **P**acked **D**ouble-Precision)

  `ADDSUBPD <fuente1/destino>, <fuente2>`



Alineamiento de datos empaquetados

- La mayoría de las instrucciones SSE que acceden a **datos empaquetados en memoria** necesitan que éstos estén **alineados**.
  - Deben residir a partir de una posición divisible entre 16.
- Alineamiento de datos en el compilador GCC:

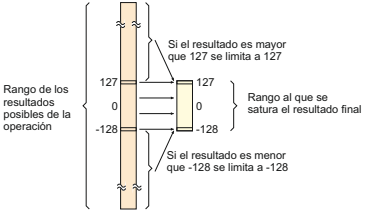
```
float v[4] __attribute__ ((aligned (16))) = {1.0, 2.0, 3.0, 4.0};
```

Aritmética con saturación

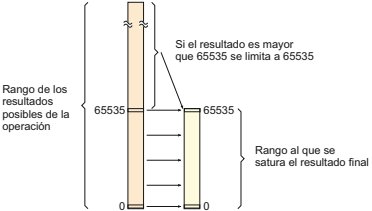
- Varias instrucciones aritméticas SIMD proporcionan 3 formas de tratar los casos en los que el resultado de una operación se sale de rango.
- **Aritmética *wraparound***:
  Ningún mecanismo para evitar la salida de rango (Igual que el funcionamiento de la unidad de enteros).
- **Aritmética con saturación con signo**:
  Los resultados se limitan al rango de números con signo del tipo de datos procesados.
- **Aritmética con saturación sin signo**:
  Los resultados se limitan al rango de números sin signo del tipo de datos procesados.

Aritmética con saturación

- Ejemplo: saturación de resultado a byte con signo.



- Ejemplo: saturación de resultado a palabra sin signo.



Una aplicación de aritmética con saturación

- Aumentar brillo de una imagen de escala de grises de 1 byte por pixel.

SIN ARITMÉTICA CON SATURACIÓN

77	204	230	102	255	0	77	230	Secuencia original de n pixels
+	+	+	+	+	+	+	+	
100	100	100	100	100	100	100	100	Aumentar brillo en 100
=	=	=	=	=	=	=	=	
177	48	74	202	99	100	177	74	Secuencia de pixels resultante

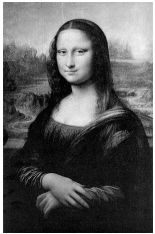
Pixels en donde se ha producido "wraparound".
El brillo de los pixels ha disminuido en lugar de aumentar.

CON ARITMÉTICA CON SATURACIÓN SIN SIGNO

77	204	230	102	255	0	77	230	Secuencia original de n pixels
+	+	+	+	+	+	+	+	
100	100	100	100	100	100	100	100	Aumentar brillo en 100
=	=	=	=	=	=	=	=	
177	255	255	202	255	100	177	255	Secuencia de pixels resultante

Pixels en donde se ha producido saturación.
El brillo de los pixels ha saturado al valor máximo.

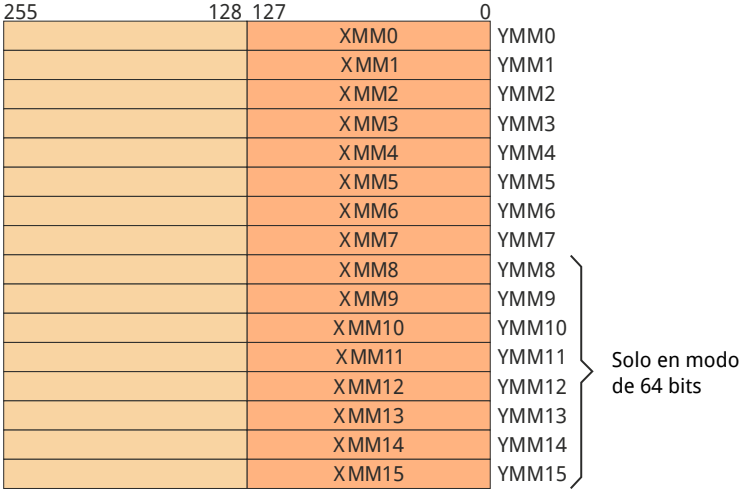
Imagen original



Brillo incrementado **sin usar** aritmética con saturación

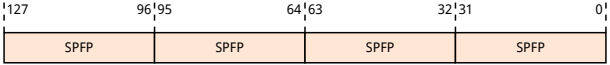
Brillo incrementado **usando** aritmética con saturación sin signo

Registros de datos AVX/AVX2

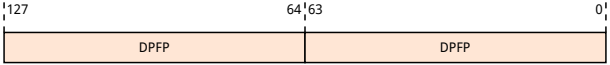


Tipos de datos AVX/AVX2. Flotantes empaquetados

- Ocho números en punto flotante de simple precisión (SPFP, Single Precision Floating Point) empaquetados en 256 bits.

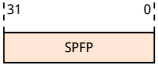


- Cuatro números en punto flotante de doble precisión (DPFP, Double Precision Floating Point) empaquetados en 256 bits.

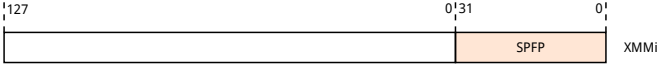


Tipos de datos AVX/AVX2. Flotantes escalares

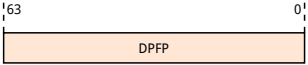
- Un número en punto flotante de simple precisión (SPFP).



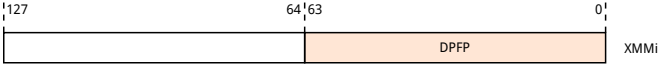
  - Se almacena en los 32 bits inferiores de los registros YMM.



- Un número en punto flotante de doble precisión (DPFP).

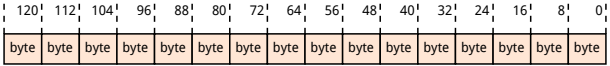


  - Se almacena en los 64 bits inferiores de los registros YMM.

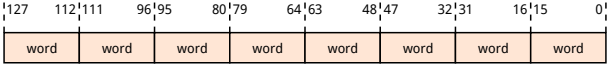


Tipos de datos enteros empaquetados AVX2

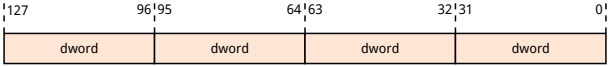
- 32 bytes empaquetados en 256 bits.



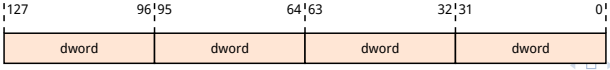
- 16 words empaquetados en 256 bits.



- 8 dwords empaquetados en 256 bits.



- 4 qwords empaquetados en 256 bits.



Instrucciones AVX/AVX2

- Los mnemotécnicos AVX/AVX2 empiezan por la letra V.
- Muchas instrucciones AVX/AVX2 son extensiones a 256 de instrucciones SSE.
- Instrucciones para convertir hacia y desde datos en punto flotante de media precisión.
- Nuevas instrucciones para copiar, mezclar, permutar, extraer e insertar datos empaquetados de forma compleja.
- Instrucciones de multiplicación y suma/resta fusionadas.
- Nuevas instrucciones que operan con los registros de propósito general:
  - Rotaciones y desplazamientos sin afectar a los indicadores.
  - Multiplicación sin signo sin afectar a los indicadores.
  - Muchas usan el nuevo formato de tres operandos.

Instrucciones AVX/AVX2 con tres operandos

- La mayoría de las instrucciones AVX/AVX2 permiten tres operandos:

  `instruccion  destino, fuente1, fuente2`

  - En general, la `instruccion` no modifica los operandos `fuente1` y `fuente2`.
  - Pueden usarse con operandos de 128 bits o de 256 bits.

```
; Con operandos de 128 bits.
vmulps    xmm0, xmm1, xmm2
vpxor     xmm0, xmm1, xmm2

; Con operandos de 256 bits.
vaddps    ymm0, ymm1, ymm2
vpand     ymm0, ymm1, ymm2
```

Calles (lanes) de 128 bits AVX/AVX2

- La mayor parte de las instrucciones AVX realizan la misma operación sobre las partes altas y bajas de 128 bits de los registros YMM.
- En general, las operaciones no pueden cruzar la frontera entre las partes altas y bajas de 128 bits. (Ej: VHADDPS, VSHUFPS, etc.).
- Las instrucciones VPERM2F128 y VPERM2I128 sí permiten ...
- Las instrucciones de carga/almacenamiento de strides ...

Funciones intrínsecas SSE, AVX/AVX2

- Las funciones intrínsecas son funciones incorporadas dentro del compilador que se corresponden con instrucciones de la CPU.
- Por ejemplo, la función

  ```
  __mm128_add_ps
  ```

  corresponde a la instrucción SSE `addps`.
- Permiten acceder a instrucciones ensamblador desde ficheros fuentes en C.
- Están incluidas en la mayor parte de los compiladores modernos.
- Ventajas:
  - Evitan tener que crear ficheros fuente en ensamblador.
  - Permite mezclar las construcciones del lenguaje C con instrucciones en ensamblador.
    - Esto debe hacerse con cuidado para no echar a perder el aumento de velocidad proporcionado por SSE, AVX/AVX2.

Funciones intrínsecas: tipos de datos

- Algunas funciones intrínsecas pueden usar los tipos usuales como `int` o `float` pero la mayoría usa tipos vector específicos.

Tipo de dato	Descripción
`__m128`	Vector de 128 bits con 4 `floats`
`__m128d`	Vector de 128 bits con 2 `doubles`
`__m128i`	Vector de 128 bits con datos enteros de 8, 16 o 32 bits
`__m256`	Vector de 256 bits con 8 `floats`
`__m256d`	Vector de 256 bits con 4 `doubles`
`__m256i`	Vector de 256 bits con datos enteros de 8, 16, 32 o 64 bits

Funciones intrínsecas: nombres de las funciones

- El formato del nombre de las funciones intrínsecas es:

  `__mm<ancho_bits>_<operación>_<tipo_datos>`

  donde
- `<ancho_bits>` identifica el tamaño del vector retornado por la función:
  - Para vectores de 128 bits este elemento está ausente.
  - Para vectores de 256 bits este elemento es 256.
- `<operación>` describe la operación realizada por la función.
- `<tipo_datos>` identifica el tipo de los argumentos principales de la función:
  - `ps`: vectores de `floats`.
  - `pd`: vectores de `doubles`.
  - `epi8/epi16/epi32/epi64`: vectores de enteros con signo de 8/16/32/64 bits.
  - `epu8/epu16/epu32/epu64`: vectores de enteros sin signo de 8/16/32/64 bits.
  - `si128/si256`: vector de 128/256 bits no especificado.
  - `m128/m128i/m128d/m256/m256i/m256d`: identifica los tipos de los vectores de entrada cuando no coinciden con el tipo retornado.

Ejemplos de funciones intrínsecas

```
__m128 _mm_load_ps (float const* mem_addr)
void _mm256_store_pd (double * mem_addr, __m256d a)
__m256d _mm256_add_pd (__m256d a, __m256d b)
__m128i _mm_adds_epu8 (__m128i a, __m128i b)
__m128i _mm_add_epi16 (__m128i a, __m128i b)
__m128i _mm_xor_si128 (__m128i a, __m128i b)
__m256 _mm256_set_m128 (__m128 hi, __m128 lo)
```