

Procesadores IA-32 e Intel®64

Alejandro Furfaro

Abril 2012

Agenda

1

Inroducción

- Genealogía
- Arquitectura Básica

2

Modelo del Programador de aplicaciones

- Arquitectura de 16 bits básica
- IA-32
- Arquitectura Intel[®] 64

3

Modos de Direccionamiento

- Modo Implícito
- Modo Inmediato
- Modo Registro
- Modos de Direccionamiento a memoria
- Modo Desplazamiento
- Modo Base Directo
- Base + Desplazamiento
- Base + Desplazamiento
- Índice * escala + desplazamiento

Orígenes

La arquitectura IA-32 deriva de la familia iAPx86 que se presentó en 1978 de la mano del procesador de Intel 8086. Fue el primer procesador de 16 bits. Meses luego Motorola lanzaría el 68000 y Zilog el Z8000A y Z8000B. Pero IBM ya había decidido utilizar el procesador de Intel para su modelo de Personal Computer (PC) que revolucionaría el mercado de computadores.

Por entonces, los usuarios de computadores demandaban disminuir el costo de actualización de sus equipos. En parte este elevado costo se debía a la necesidad de reemplazar no solo el hardware sino además el sistema operativo y las aplicaciones. Al lanzar el procesador 8086, Intel se compromete a mantener compatibilidad ascendente en los modelos subsiguientes de procesadores para satisfacer esta demanda de los usuarios. Esto fue uno de los motivos de la decisión de IBM de adoptar esta familia de procesadores como base para su PC.

Orígenes

Unos meses luego del lanzamiento del 8086, Intel presenta una variante del mismo procesador: el 8088, que era idéntico en su arquitectura interna de 16 bits, salvo por su bus de datos que a diferencia del bus de 16 bits del 8086 tenía solo 8 bits, lo cual le daba una mas directa conectividad con la innumerable cantidad de periféricos de 8 bits que por entonces había disponibles. Finalmente por esta mejor adaptación a nivel de hardware con los dispositivos existentes, IBM lanzaría su primer PC basada en un 8088. A partir de entonces se afirmaría el liderazgo de Intel en esta industria.

En 1982 Intel presenta el 80286, que mejoraba la CPU 8086 en la ejecución de numerosas instrucciones e incorporando multitasking. Su arquitectura de 16 bits seguía siendo la misma. A partir de este procesador IBM diseña el modelo de PC conocido como AT.

Orígenes

En 1984 Intel lanza el primer procesador de 32 bits, el 80386, y con él la arquitectura IA-32. Ya la industria comienza a hablar de los procesadores x86. Aparecen fabricantes que producirán procesadores IA-32, bajo licencia de Intel primero, y a partir de que ésta lanza el procesador Pentium en 1993 ya en forma independiente. Entre ellos se destaca AMD.

A fines de los años 90 Intel comienza una sociedad con Hewlett Packard que derivaría en una nueva arquitectura llamada IA-64 y cuyos procesadores de 64 bits Itanium e Itanium 2 no tendrían el éxito de los x86. AMD en cambio apostó a la arquitectura dominante, e ideó las extensiones de 64 bits para los procesadores IA-32. Llamó a esta nueva arquitectura x86-64 y posteriormente AMD64. Intel por primera vez debe incorporar a sus procesadores IA-32, funciones desarrolladas por sus competidores. Llama a esta arquitectura Intel® 64 .

Modos de Operación “Legacy”

Los procesadores IA-32 tienen tres modos de operación, de los cuales surgen sus capacidades arquitecturales e instrucciones disponibles:

- **Modo Real:** En este modo el procesador implementa el entorno de operación del 8086, con algunas extensiones: puede pasar por software al Modo Protegido o al Modo Mantenimiento del Sistema, puede utilizar registros de 32 bits, puede reconfigurar la ubicación del vector de interrupciones, ya que a pesar de que el 8086 no lo tenía, ahora existe y es accesible desde modo Real el registro IDTR (ver Interrupciones mas adelante). Para honrar su compromiso de compatibilidad ascendente, todos los procesadores IA-32 e Intel® 64 arrancan en este Modo de Trabajo.

Modos de Operación “Legacy”

- **Modo Protegido:** Este es el modo por excelencia de los procesadores de esta familia inaugurado por el procesador 80286. En este modo se implementa multitasking y se despliega un espacio de direccionamiento de 4 Gbytes, extensible a 64 Gbytes. A partir del procesador 80386 se lleva el modelo arquitectural a 32 bits y se introduce un sub-modo al que puede ponerse a una determinada tarea, denominado Virtual-8086 que permite a un programa diseñado para ejecutarse en un procesador 8086, poder ejecutarse como una tarea en Modo Protegido. Esto fue muy útil para implementar en Windows la “Ventana DOS”. Actualmente no es utilizado, ya que la consola que se ejecuta utiliza un código diferente del DOS original, y es en general una tarea mas.

Modos de Operación “Legacy”

- **Modo Mantenimiento del Sistema:** El procesador ingresa a este modo por dos caminos: Activación de la señal de interrupción #SMM, o mediante un mensaje SMI desde su APIC local. Este modo fue introducido a partir de los modelos 386SL y 486SL para realizar funciones específicas para la plataforma de hardware en la cual se desempeña el procesador, como lo son ahorro de energía y seguridad. Estos procesadores fueron los primeros diseñados para notebooks. Al ingresar a este modo el procesador resguarda en forma automática el contexto completo de la tarea o programa interrumpido, y pasa a ejecutar en un espacio separado. Una vez efectuadas las operaciones necesarias y cuando debe salir de este modo el procesador reasume la tarea o programa interrumpida independientemente del modo de operación en el que se encuentra.

Modos de Operación de 64 bits

Hasta la aparición de las extensiones de 64 bits, éstos eran los modos de trabajo estándar de los procesadores IA-32. Los procesadores Intel® 64 además de los modos de trabajo de los IA-32 incluyen un modo IA-32e, al que se pasa estando en modo protegido, con paginación habilitada y PAE activo (Como estudiaremos en Paginación). En este modo a su vez existen dos sub-modos (nada es simple en este mundo):: Compatibilidad y Modo 64 bits

Modos de Operación de 64 bits

- **Modo Compatibilidad:** Pensado tal vez para garantizar la transición, permite a las aplicaciones de 16 y 32 bits ejecutarse sin recompilación bajo un sistema operativo de 64 bits. El entorno de ejecución es el que veremos para la arquitectura IA-32 (registros de 32 bits, etc). No soporta el manejo de tareas del modo IA-32 (TSS mediante), ni el modo Virtual 8086 (en esto nos basamos para asegurar que está obsoleto al referirnos a este modo algunos párrafos atrás. Incluye los mecanismos de protección del modo 64 bits.

En el modo compatibilidad un Sistema Operativo de 64 bits puede ejecutar junto con aplicaciones puras de 64 bits, tareas o aplicaciones de 16 y 32 bits sobre la base de diferentes segmentos de código. La aplicación o tarea accede a una arquitectura IA-32 pura, utilizando direcciones de 16 y 32 bits, con 4 Gbytes de espacio de direccionamiento y con la posibilidad de acceder por encima de ese límite habilitando PAE.

Modos de Operación de 64 bits

- **Modo 64 bits:** Este modo habilita a un Sistema Operativo de 64 bits a ejecutar tareas escritas utilizando direcciones lineales de 64 bits. En este modo se extienden de 8 a 16 los Registros de propósito general cuyo ancho de palabra ahora es de 64 bits (para ello se introduce el prefijo REX para las instrucciones que deseen acceder a las versiones de Registros de propósito general de 64 bits), agregándose los registros R8 a R15, y los registros SIMD también se extienden a 16 manteniendo su ancho de 128 bits, XMM0 a XMM15.

Registros y espacio de direccionamiento

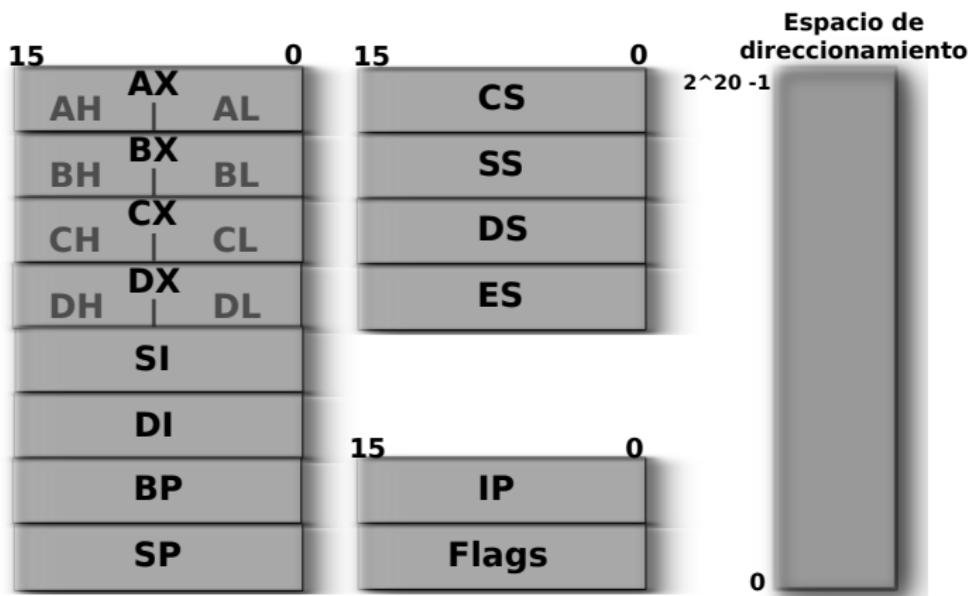


Figura: Entorno Básico de ejecución en 16 bits

Arquitectura de 32 bits compatible

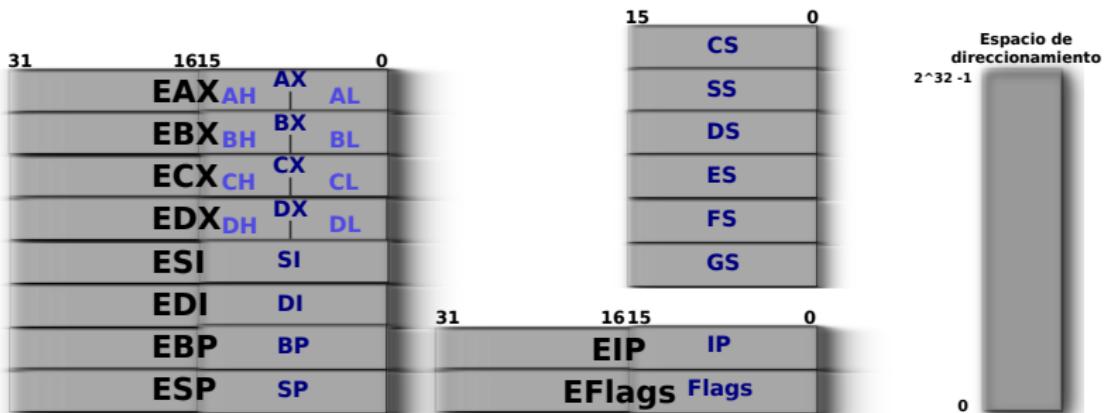


Figura: Entorno Básico de ejecución del 80386

Floating Point Unit

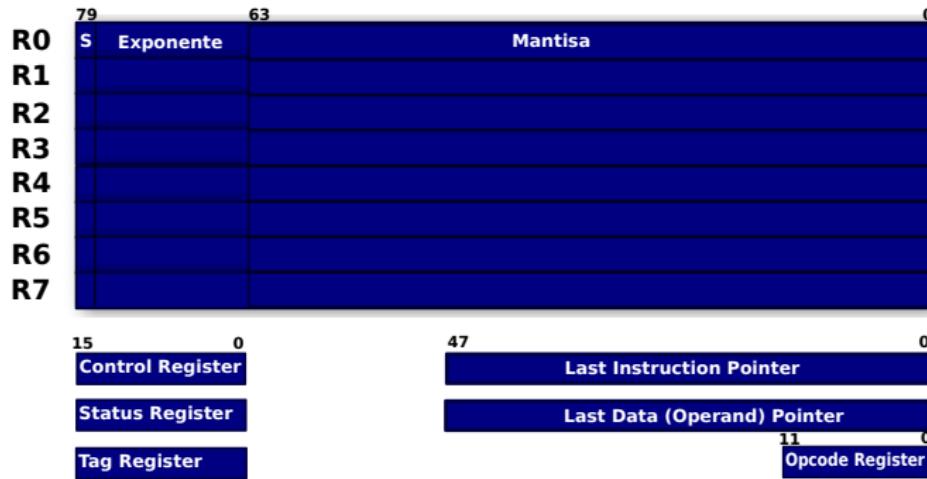


Figura: Entorno Básico de ejecución de la FPU

Floating Point Unit

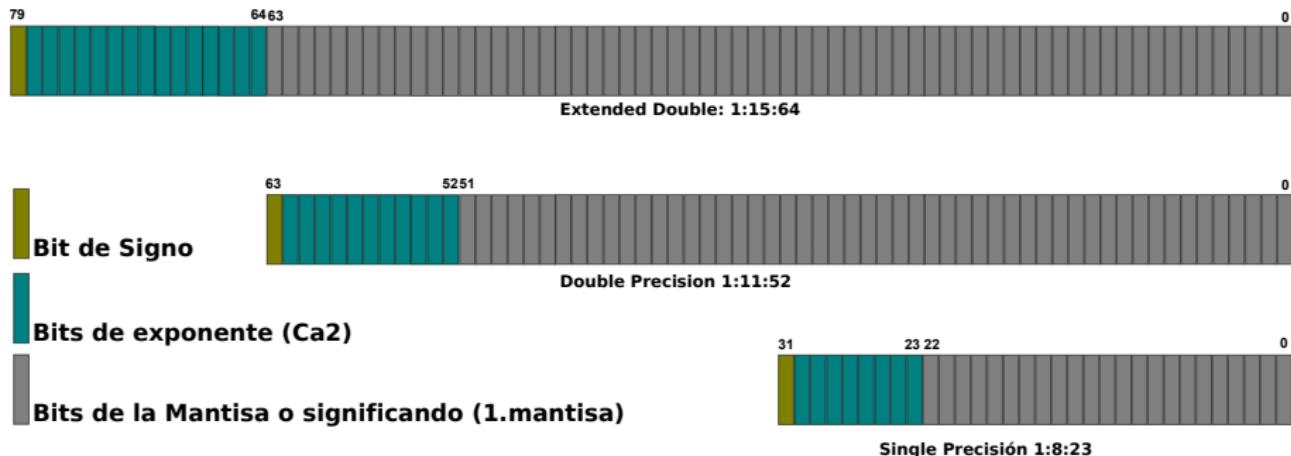


Figura: Formatos de los Datos de Punto Flotante de la FPU

Floating Point Unit on board

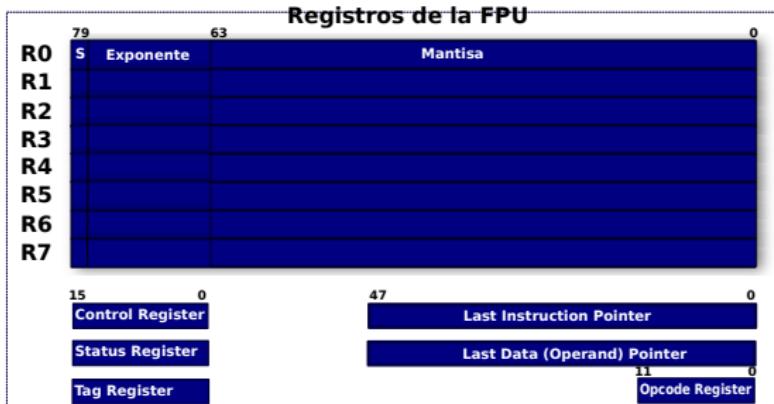


Figura: Entorno Básico de ejecución con la FPU on board

Extensiones MMX

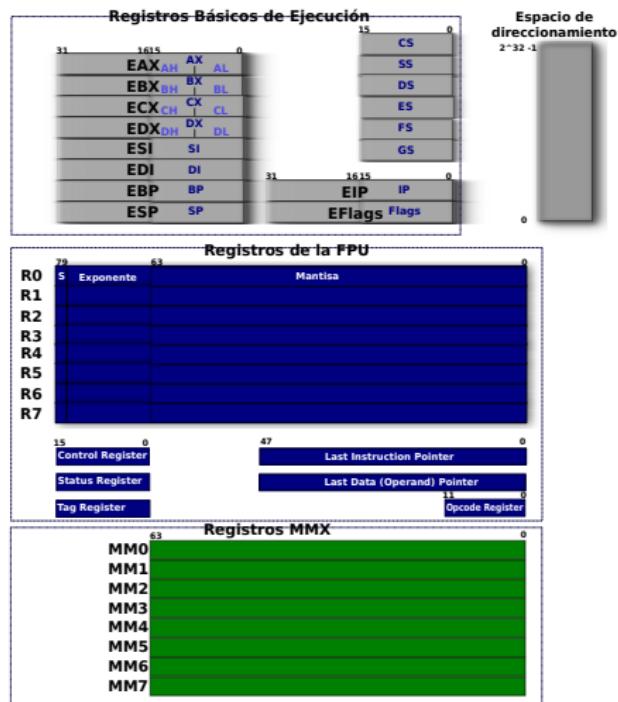


Figura: Entorno Básico de ejecución con tecnología MMX

Llega el Pentium III

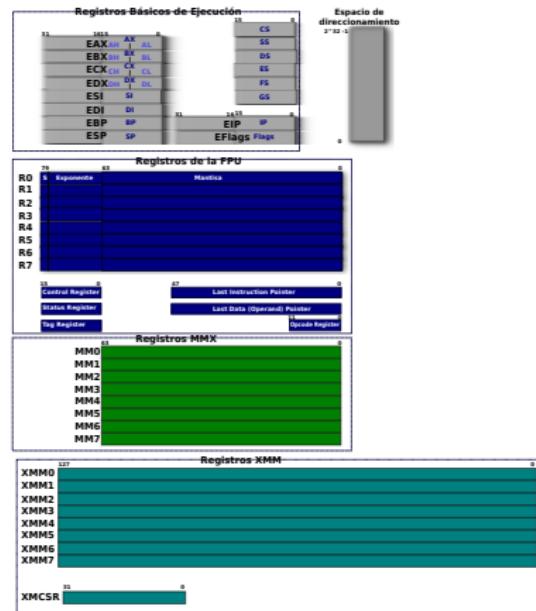
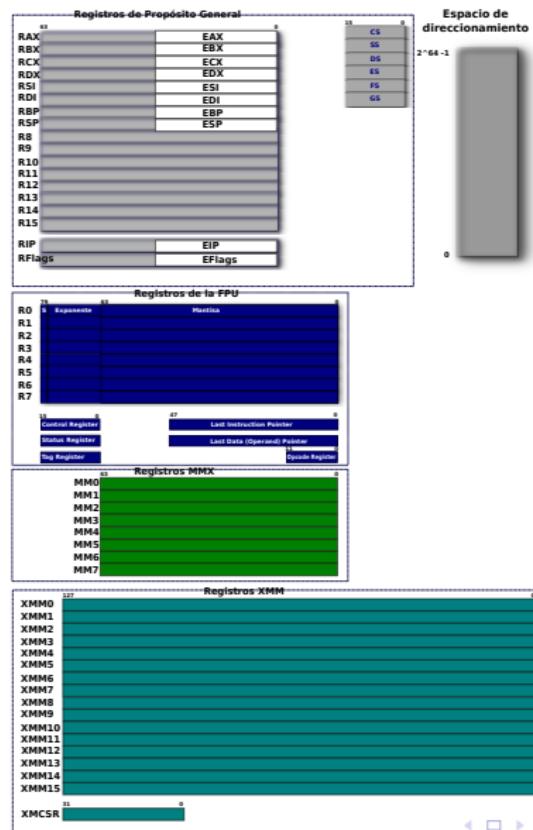


Figura: Entorno Básico de ejecución IA-32 actual

Extensiones de 64 bits



Extensiones de 64 bits

En este modo los registros de Propósito General se extienden a 64 bits y aparecen 8 registros de 64 bits adicionales. El procesador a pesar de estar en modo 64 bits puede acceder a diferentes tamaños de operador. Para ello es necesario que utilice el prefijo REX, antecediendo a cada instrucción que requiera este tipo de operandos.

Registros de Propósito General

	63	0
RAX		EAX
RBX		EBX
RCX		ECX
RDX		EDX
RSI		ESI
RDI		EDI
RBP		EBP
RSP		ESP
R8		R8D
R9		R9D
R10		R10D
R11		R11D
R12		R12D
R13		R13D
R14		R14D
R15		R15D

Extensiones de 64 bits

Registros de Propósito General

	63	0
RAX		AX
RBX		BX
RCX		CX
RDX		DX
RSI		SI
RDI		DI
RBP		BP
RSP		SP
R8		R8W
R9		R9W
R10		R10W
R11		R11W
R12		R12W
R13		R13W
R14		R14W
R15		R15W

Figura: Intel® 64 : Registros de Propósito general para operandos word, utilizando prefijo REX

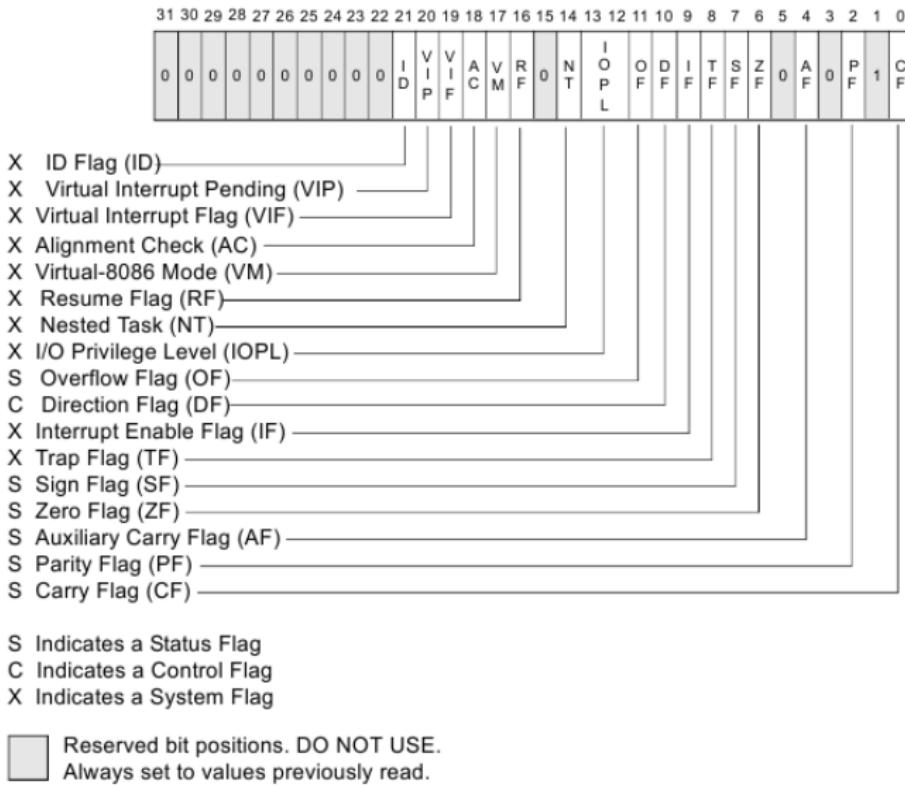
Extensiones de 64 bits

Registros de Propósito General

	63	0
RAX		AL
RBX		BL
RCX		CL
RDX		DL
RSI		SIL
RDI		DIL
RBP		BPL
RSP		SPL
R8		R8L
R9		R9L
R10		R10L
R11		R11L
R12		R12L
R13		R13L
R14		R14L
R15		R15L

Figura: Intel® 64 : Registros de Propósito general para operandos byte, utilizando prefijo REX

Flags, EFLAGS, RFLAGS



Como Obtener los Operandos

Como reglas generales, de acuerdo a la documentación disponible, las instrucciones de estos procesadores pueden obtener los operandos desde:

- La instrucción en si misma, es decir que el operando está implícito en la instrucción)
- Un registro
- Una posición de memoria
- Un port de E/S

Como Almacenar los resultados

Del mismo modo el resultado de una instrucción puede tener como destino para su almacenamiento:

- Un registro
- Una posición de memoria
- Un port de E/S.

¿Que significa Direccionamiento Implícito?

Se trata de instrucciones en las cuales el código de operación es suficiente como para establecer que operación realizar, y cual es el operando.

Son ejemplos de este Modo, las instrucciones que operan sobre los Flags, como por ejemplo:

- CLC: Clear Carry. El operando (el Flag CF), está implícito en la operación. Lo mismo ocurre con STC (Set Carry), CMC (Complement Carry),
- CLD (Clear Direction Flag), STD (Set Direction Flag),
- CLI y STI para limpiar y setear el flag de Interrupciones (IF),

entre otros.

Mas Instrucciones con Direccionamiento Implícito

Otras instrucciones con modo de direccionamiento implícito son algunas que en realidad no requieren operandos, ya que tienen por objeto indicar al procesador alguna acción específica. Por ejemplo:

- HLT, que suspende la ejecución del procesador y lo pone en un estado que se define como HALT, en el que está suspendido, en un modo de consumo de energía mínimo, y del cual se puede salir solo mediante alguna Interrupción de Hardware, incluidas NMI y SMI, alguna excepción de debug, o una señal #BINIT, #INIT, o RESET,
- NOP que provoca que el procesador no haga nada durante el intervalo de una instrucción en el flujo de instrucciones.
- Ajustes para operaciones aritméticas sobre números en formato BCD o ASCII empaquetado, como AAA (ASCII Adjust after Adition), AAD (ASCII Adjust after Division), AAM (ASCII Adjust after Multiplication), AAS ASCII Adjust after Subtraction), DAA (Decimal Adjust after Adition), DAS (Decimal Adjust after Subtraction). Todas estas instrucciones operan sobre el Acumulador AX, razón por la cual no es necesario indicar el operando en la instrucción.
- CWD/CDQ/CQO Convert Word to Double Word y Convert Double Word to Quadword, que operan con el contenido de AX EAX y RAX extendiéndolo con signo a DX:AX, EDX:EAX, y RDX:RAX respectivamente,

¿Que significa direccionar en Modo Inmediato?

En este modo, el operando fuente viene dentro del código de la instrucción, con lo cual no es necesario ir a buscarlo a memoria luego de decodificada la instrucción.

```
1 ; /////////////////////////////////
2 ; ascii recibe en al un byte decimal no empaquetado
3 ; y retorna su ascii en el mismo registro
4 ; /////////////////////////////////
5 ascii:
6     add al, '0'
7     cmp al, '9'
8     jle listo
9     add al, 'A'-'9'-1
10 listo:    ret
```

¿Que significa direccionar a Registro?

Todos los operandos involucrados son Registros del procesador. No importa si hay uno o dos operandos, son todos Registros.

- Registros de Propósito General tanto de 64, 32, 16 u 8 bits de acuerdo con las arquitecturas IA-32 e Intel® 64
- Registros de segmentos
- EFlags (o RFlags)
- Registros de la FPU,
- MM0 a MM7,
- XMM0 a XMM7 o XMM15 según sea IA-32 o Intel® 64 respectivamente,
- Registros de Control
- Registros de Debug
- MSR's (Model Specific Registers)

```
1 inc rdx ; Incrementa en contenido del registro RDX
2 mov eax,ebp ; Mueve al registro EAX el contenido del EBP
3 mov cr0,eax ; Mueve al registro cr0 el contenido del EAX
4 fmul st(0),st(3) ; ST(0) = ST(0) * ST(3)
5 sqrtpd xmm2,xmm6 ; Raíz cuadrada de dos double precision FP
6 ; empaquetados en xmm6. Resultados en xmm2.
```

¿Como nos referimos desde el software a un Operando en Memoria?

Para identificar un operando (fuente o destino) de una instrucción en memoria, se utiliza lo que Intel denomina dirección lógica.

Este nombre obedece a que es una dirección abstracta expresada en términos de su arquitectura pero que necesita ser procesada para convertirse finalmente en la dirección física que es la que saldrá hacia el bus del sistema por las líneas de address.

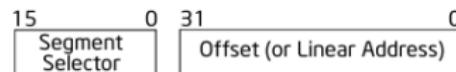


Figura: IA-32: Dirección lógica

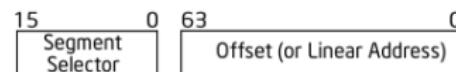


Figura: Intel® 64 : Dirección Lógica

Segmentos para cada dirección Lógica

- El valor del segmento en una dirección lógica, puede especificarse de manera implícita.
- Por lo general este es el modo en que se hace, aunque es posible explicitar con que segmento se desea direccionar un operando de memoria determinado.
- Si no se especifica explícitamente el segmento como parte de la dirección lógica el procesador lo establecerá automáticamente de acuerdo a la tabla

Referencia a:	Reg.	Segmento	Regla de selección por defecto
Instrucciones	CS	Segmento de Código	Cada opcode fetch
Pila	SS	Segmento de Pila	Todos los push y pop, cualquier referencia a memoria que utilice como registro base ESP o EBP.
Datos Locales	DS	Segmento de datos	Cualquier referencia a un dato, excepto en el stack o un destino de instrucción de string
Strings Destino	ES	Segmento de datos extra direccionado por ES	Destino de Instrucciones de manejo de strings

Cuadro: Reglas de selección de segmento predefinidos

Desplazamiento para ubicar operandos en Memoria

La riqueza de modos de direccionamiento de operandos en memoria la da el desplazamiento. Aquí es Intel puso todo el esfuerzo dando una gran cantidad de alternativas para calcular el desplazamiento.

Básicamente un desplazamiento tiene al menos uno de los siguientes componentes, o cualquiera de las combinaciones posibles:

- Desplazamiento directo: Se trata de un valor de 8, 16, o 32 bits, explícitamente incluido en la instrucción.
- Base: Se trata de un valor contenido en un registro de propósito general, que indica una dirección a partir de la cual se calcula el desplazamiento. Es un valor de 32 bits en el modo IA-32 y de 64 bits en IA-32e.
- Índice: Se trata de un valor contenido en un registro de propósito general, que se representa la dirección a la cual nos queremos referir. Típicamente es un valor que al incrementarse permite recorrer por ejemplo un buffer de memoria. Es un valor de 32 bits en el modo IA-32 y de 64 bits en IA-32e.
- Escala: Es un valor por el cual se multiplica el valor del Índice: Puede valer 2, 4, u 8.

Calculando el desplazamiento

A partir de estos cuatro componentes o combinación de algunos de ellos, se obtiene lo que Intel denomina Dirección Efectiva. Esta denominación intenta representar el significado del offset dentro de un segmento: es la dirección que ocupa efectivamente el elemento direccionado respecto del inicio del segmento.

Los cuatro componentes anteriores pueden ser positivos o negativos en representación Ca2 (excepto el valor del factor de escala que es siempre positivo). A continuación se presenta los diferentes registros y valores que pueden integrar cada uno de los cuatro componentes descriptos:

$$\begin{array}{c}
 \text{Base} \quad \text{Índice} \quad \text{Escala} \quad \text{Desplazamiento} \\
 \left[\begin{array}{l} EAX \\ EBX \\ ECX \\ EDX \\ ESP \\ EBP \\ EDI \\ ESI \end{array} \right] + \left[\begin{array}{l} (EAX) \\ (EBX) \\ (ECX) \\ (EDX) \\ (ESP) \\ (EBP) \\ (EDI) \\ (ESI) \end{array} \right] * \left[\begin{array}{l} (1) \\ (2) \\ (4) \\ (8) \end{array} \right] + \left[\begin{array}{l} \text{Nada} \\ 8bits \\ 16bits \\ 32bits \end{array} \right]
 \end{array}$$

En general la expresión general que representa el cálculo interno del procesador es:

$$\text{DireccionEfectiva} = \text{Base} + (\text{Indice} * \text{escala}) + \text{Desplazamiento}$$

Casos particulares

Cuestiones a destacar, respecto de la matriz anterior:

- El registro ESP (o RSP), no puede utilizarse de Índice. Esto es bastante lógico ya que su uso privilegiado es como puntero de pila. Por lo tanto modificarlo para recorrer otro array de datos que no sea la pila es poco menos que imprudente.
- Cuando se emplean como registros base ESP y EBP (o RSP y RBP), se utilizan asociados al registro de segmento SS. Para el resto de los usos se asocian al DS.
- El factor de escala solo se puede emplear cuando se utiliza un Registro Índice. En otro caso no se emplea.

¿El Offset puede venir directo en la Instrucción?

Se incluye en la instrucción un valor en forma explícita que representa en forma directa el valor del offset.

En el listado siguiente se muestran diversos ejemplos de instrucciones de este Modo.

```
1 or    ecx, dword [0x300040A0] ; Calcula la or lógica entre  
2                                ; ECX y la doble word contenida  
3                                ; a partir de la dirección de  
4                                ; memoria 0x300040A0.  
5 inc   byte [0xAF007600]       ; Incrementa el byte contenido  
6                                ; por la dirección de memoria  
7                                ; 0xAF007600  
8 dec   dword [ i ]           ; El valor de la dirección de la  
9                                ; variable i se calcula directa-  
10                               ; mente y el valor se reemplaza  
11                               ; en tiempo de compilación
```

Una de las formas de direccionar en forma indirecta

- En este modo el offset está contenido directamente en un registro Base.
- El procesador simplemente lo toma desde el registro, sin otro cálculo.

```
1 | mov    edx , i      ; edx = desplazamiento de i en el segmento
2 |                   ; de datos
3 | inc    [ edx ]     ; incrementamos i direccionada a través de
4 |                   ; un registro puntero base.
```

El ejemplo anterior es trivial ya que la variable como vimos puede incrementarse de manera directa

Formas mas útiles de direccionar en forma indirecta

- Combina el valor contenido en un registro que apunta a la base de un bloque de datos con un valor explícito puesto en la instrucción, que permite calcular la dirección efectiva del operando.
- También resulta útil para acceder a una estructura mas compleja de datos, apuntando a la base de la estructura con un registro y utilizando el Desplazamiento para acceder al campo deseado de la estructura.

Formas mas útiles de direccionar en forma indirecta

```

1 ORG 8000h
2 use16          ;Estamos en Modo Real=>Código de 16 bits
3 start: jmp main      ;Salto al inicio del programa.
4 ALIGN 8
5 gdt:    resb 8       ;NULL Descriptor. Dejamos 8 bytes sin usar.
6 Data_sel equ $-gdt   ;Calcula dinámicamente la posición del selector
7           dw 0xffff    ;límite 15.00
8           dw 0x0000    ;base 15.00
9           db 0x00      ;base 23.16
10          db 10010010b;Presente Segmento Datos Read Write
11          db 0xCF      ;G = 1, D/B = 1, y límite 0Fh
12          db 0x00      ;base 31.24
13 gdt_size equ $-gdt   ;calcula dinámicamente el tamaño de la gdt
14 main:
15 ...
16 ...
17         mov ebx,Data_sel ;ebx apunta a la base del descriptor
18 ;//////////;Lee con direccionamiento base + desplazamiento los atributos del descriptor
19 ;de segmento de datos definido en la tabla anterior
20         mov al, byte [ebx + 5]
21 ;//////////
22         test al,0x80    ;Testea bit de presente
23         jz NoPresente;Si no está presente, salta
24

```

Formas mas útiles de direccionar en forma indirecta

- Es una forma eficiente de acceder a elementos de un array cuyo tamaño es 2, 4, u 8 bytes
- El Desplazamiento puede ubicar el inicio del array y el valor índice se guarda en un registro que al incrementar pasa al siguiente elemento permitiendo con la escala ajustar al tamaño del mismo

```

1 %define Dir_Tabla      0x2000F000
2 %define mascara        0xFFFFFFF
3     mov    ecx , size_tabla ;ecx = cantidad de elementos de
4                           ;4 bytes de la tabla.
5     xor    esi , esi       ;esi apunta al inicio de la tabla
6 mas:
7     and    [esi*4 + Dir_Tabla], mascara
8                           ;borra bit menos significativo
9                           ;del elemento de la tabla
10    inc   esi             ;esi apunta al siguiente
11                           ;elemento de 4 bytes
12    loop  mas             ;va por el siguiente elemento
13                           ;hasta que exc sea 0

```

Formas mas útiles de direccionar en forma indirecta

- Este modo es especial para acceder a matrices bidimensionales.
- Un ejemplo obligado es el buffer de video
- Cuando trabaja en modo texto el buffer de video es una matriz de 25 filas por 80 columnas.
- Cada elemento consta de dos bytes: el primero contiene el ASCII del carácter a presentar y el segundo los atributos (intensificado, video inverso, y colores de carácter y fondo).
- Vamos a escribir un código que aprovechando este modo de direccionamiento sirva para limpiar el contenido de la pantalla.
- El registro base apunta a cada línea de la pantalla y el índice apunta a cada elemento de la línea.

Formas mas útiles de direccionar en forma indirecta

```
1 ; El siguiente código limpia la pantalla dejándola en modo
2 ; blanco sobre negro. El buffer de video comienza en la
3 ; dirección física 0x000B8000. Utilizamos este valor como
4 ; desplazamiento en el cálculo de la dirección efectiva.
5 xor    ebx,ebx      ;resultado ebx = 0.
6                      ;ebx apunta a la 1er. fila de 80 caracteres
7 col:
8 xor    edi,edi      ;resultado edi = 0.
9                      ;edi apunta al primer elemento de la fila
10 mov   ecx,size_row ;ecx = cantidad de filas para loop
11 row:
12 ;caracter nulo no imprime nada en pantalla
13 mov   byte [ebx + edi + 0x000B8000],0x00
14
15 add   edi,2          ;edi apunta al porx. elemento de 2 bytes
16 loop  row            ;si CX = 0 se completó fila
17 add   ebx,160         ;Apunta a la siguiente fila
18 cmp   ebx,0x1000       ;fin del buffer?
19 jle   col
```



Formas mas útiles de direccionar en forma indirecta

```
1 ; El siguiente código limpia la pantalla dejándola en modo
2 ; blanco sobre negro. El buffer de video comienza en la
3 ; dirección física 0x000B8000. Utilizamos este valor como
4 ; desplazamiento en el cálculo de la dirección efectiva.
5 xor    ebx,ebx      ;resultado ebx = 0.
6                      ;ebx apunta a la 1er. fila de 80 caracteres
7 col:
8 xor    edi,edi      ;resultado edi = 0.
9                      ;edi apunta al primer elemento de la fila
10 mov   ecx,size_row ;ecx = cantidad de filas para loop
11 row:
12 ;caracter nulo no imprime nada en pantalla
13 mov   byte [ebx + edi*2 + 0x000B8000],0x00
14
15 inc   edi          ;edi apunta al porx. elemento de 2 bytes
16 loop  row          ;si CX = 0 se completó fila
17 add   ebx,160        ;Apunta a la siguiente fila
18 cmp   ebx,0x1000     ;fin del buffer?
19 jle   col
```

Tipos básicos de datos

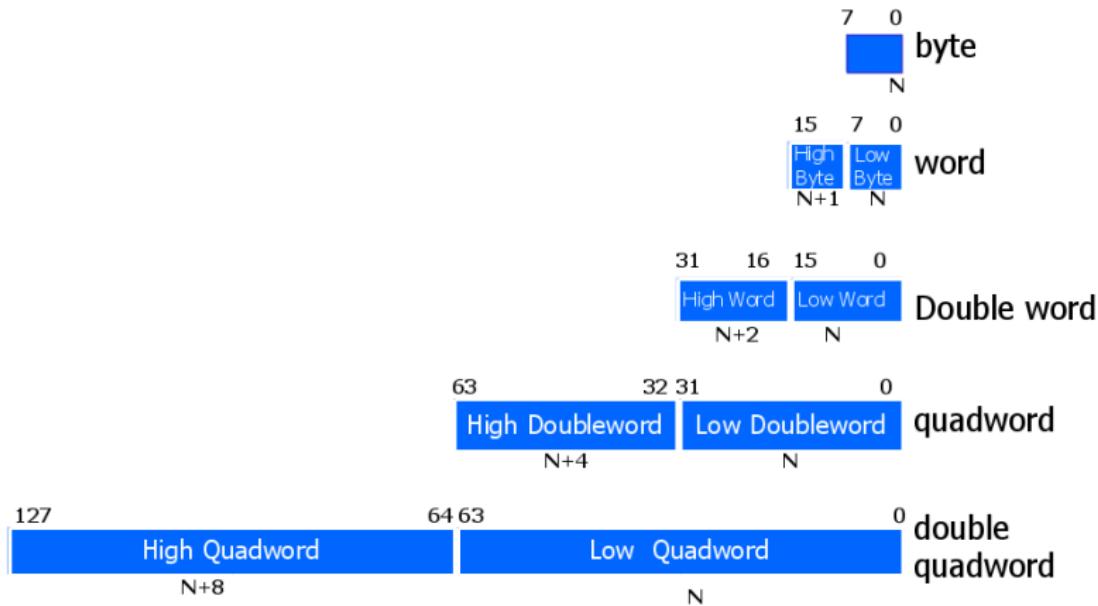


Figura: IA-32 e Intel® 64 : Tipos de datos fundamentales

Little endian

- Desde el procesador 8086, esta familia maneja el almacenamiento en memoria de las variables en el formato little endian.
- Dicho de otra forma: una variable de varios bytes de tamaño almacena su byte menos significativo en la dirección con que se referencia la variable y a partir de allí coloca el resto de los bytes en orden de significancia, terminando con el almacenamiento del byte mas significativo, en la dirección de memoria mas alta (es decir termina con el menor, de allí little endian).
- Esta situación se representa en próximo slide. A simple vista pareciera que están almacenados al revés, ya que si lo miramos en la memoria está de atrás hacia adelante.
- Otros procesadores utilizan el formato BigEndian, es decir colocando la información en el orden en el que normalmente esperamos encontrarla.

Little endian

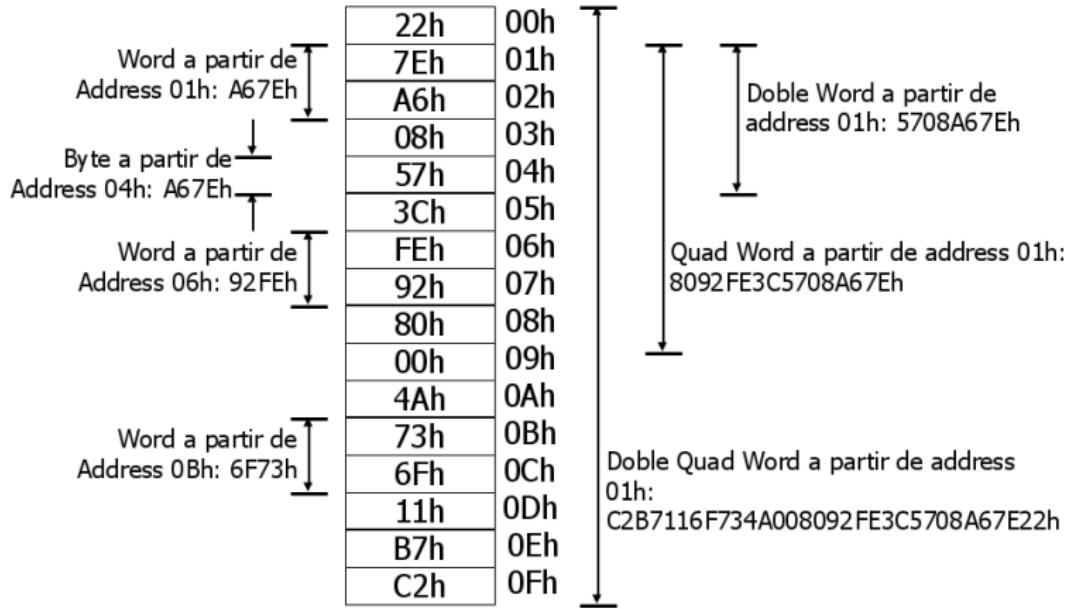
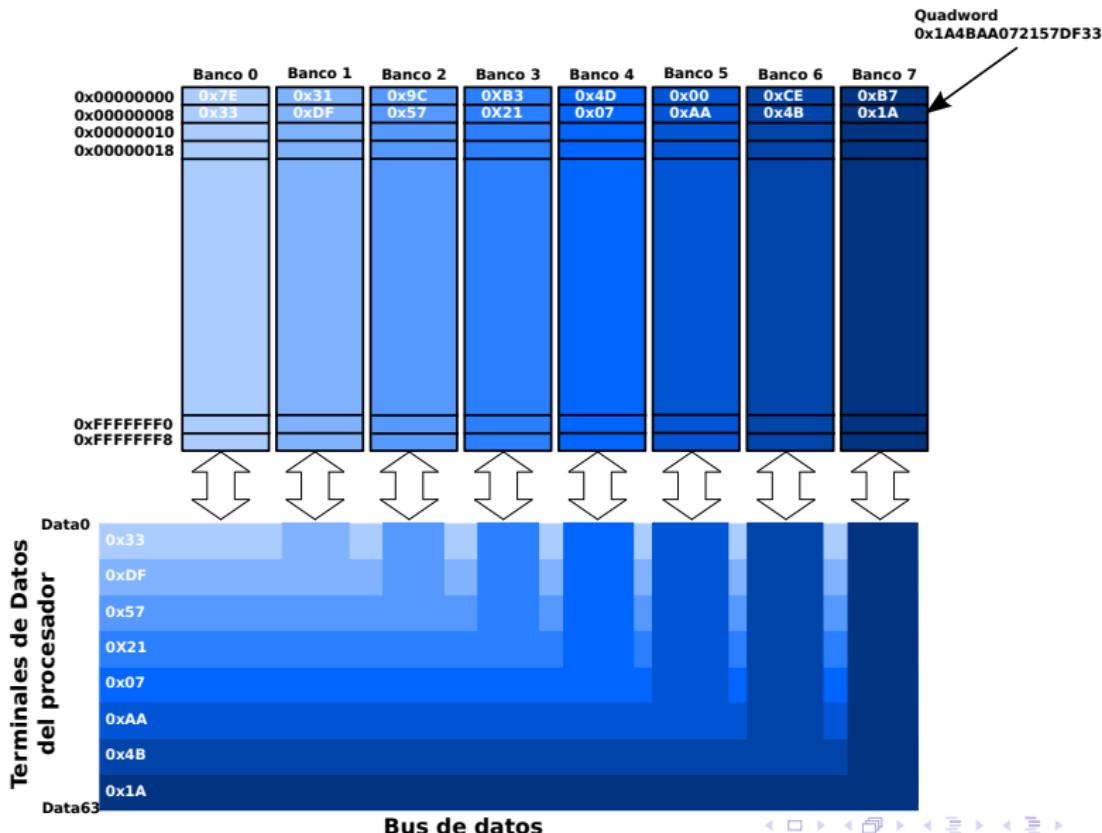


Figura: IA-32 e Intel® 64 : Alineamiento en memoria para los diferentes tipos de datos

Little endian

- La razón por la que Intel adoptó Little Endian, obedece a que el procesador 8086 (y sus sucesores por cuestión de compatibilidad), administra la memoria de bytes.
- Esto significa que cada dirección de memoria tiene una capacidad de almacenamiento de 8 bits.
- Por lo tanto, y debido a esta decisión de diseño, es que un bus de datos de 16 bits primero, 32 mas tarde, y 64 actualmente, se conecta a bancos de memoria RAM Dinámica organizados en bytes.
- Por lo tanto, cuando el procesador lee un dato de 64 bits a través del bus de datos, cada byte de las direcciones de memoria que se leen viaja por un byte del bus de datos, de acuerdo al ordenamiento que la información tiene en la memoria, de la manera en que se muestra en el próximo slide.

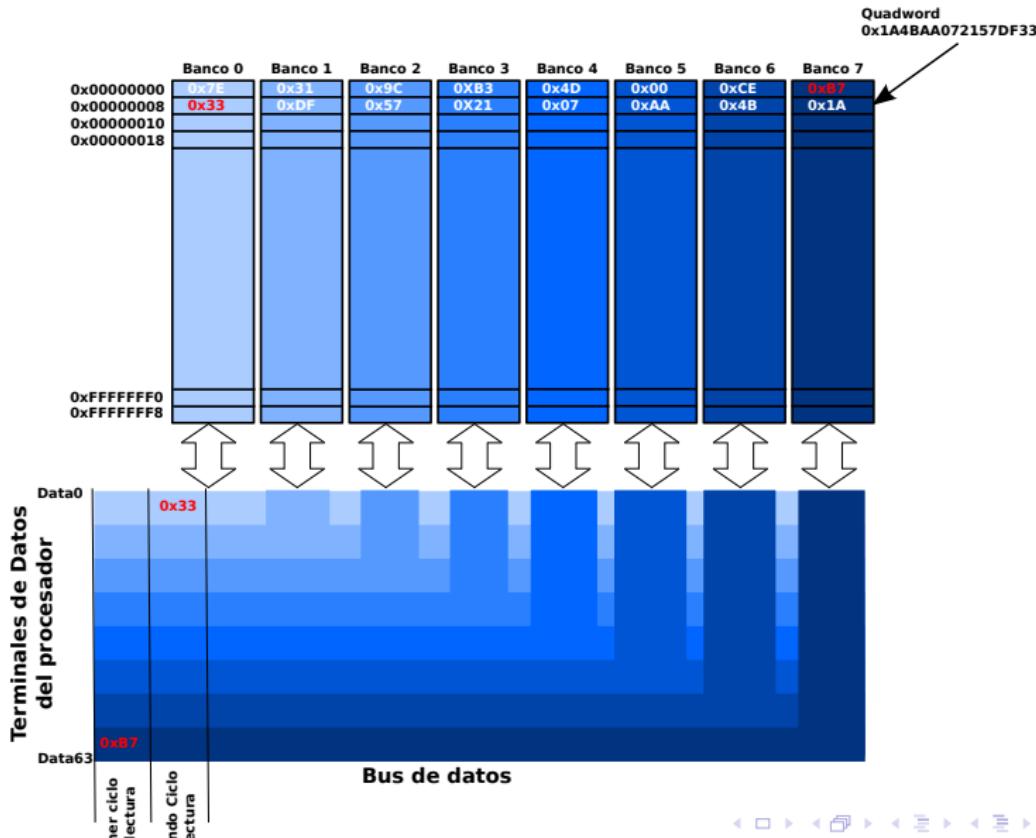
Little endian



¿Porque conviene alinear los datos?

- Los procesadores IA-32 e Intel® 64 no ponen restricciones respecto de la alineación en memoria para las diferentes variables de los programas (consecuencia favorable de la administración de memoria de a bytes).
- Esto otorga gran flexibilidad a la hora de aprovechar al máximo la memoria.
- Pero si una variable queda repartida en dos filas diferentes, se requerirán dos ciclos de lectura para accederla.
- Esta situación se representa en el siguiente slide.
- La variable se trae al procesador con dos lecturas de memoria
- Estas dos lecturas de memoria son transparentes a nivel de software (la aplicación no debe ser modificada en absoluto), ya que el procesador autónomamente realiza las dos lecturas.
- Entonces ¿cuál es el problema?. La respuesta es: **performance**. Dos ciclos de lectura en lugar de uno solo tornan mas lento el acceso a la variable. Esto puede evitarse usando las directivas de alineación que todos los lenguajes poseen.

Acceso a datos no alineados



Formato de Enteros con y sin signo

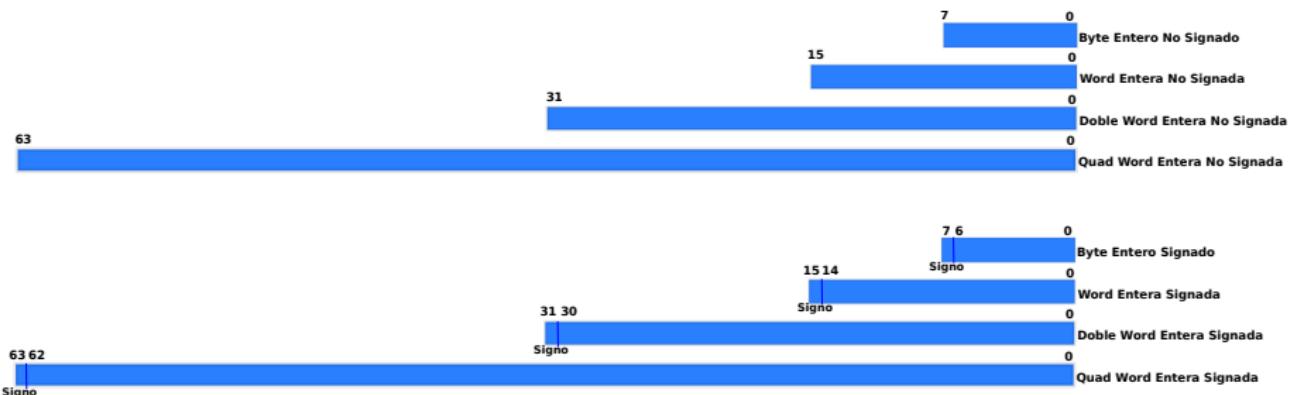


Figura: IA-32 e Intel® 64 : Tamaños de datos enteros con y sin signo

Rangos de Enteros con y sin signo

Formato		Codificación Ca2	
		Signo	
Positivo	Máximo	0	11....11
		.	.
	Mínimo	0	00....01
Cero		0	00....00
Negativo	Mínimo	1	11....11
		.	.
	Máximo	1	00....01
Indefinido		1	00....00
		Byte entero Signado	← 7 bits →
		Word entero Signado	← 15 bits →
		DoubleWord entero Signado	← 31 bits →
		QuadWord Signado	← 63 bits →

Cuadro: Codificación de enteros signados

Representación en punto flotante

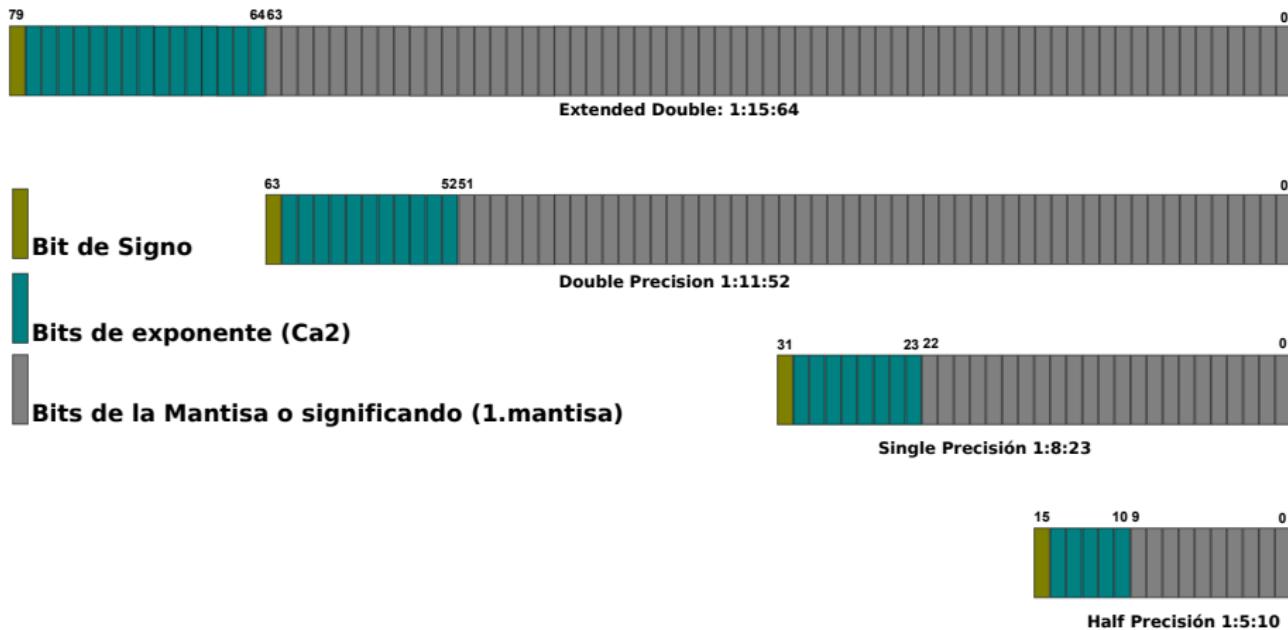


Figura: Formato de Datos en representación de Punto Flotante

Rangos de Números en punto flotante

Tipo de Dato	Longitud	Precisión	Rango Normalizado (aproximado)	
			Binario	Decimal
Half Precision	16	11	2^{-14} a 2^{15}	$3,1 \times 10^{-5}$ a $6,50 \times 10^4$
Single Precision	32	24	2^{-126} a 2^{127}	$1,18 \times 10^{-38}$ a $3,40 \times 10^{38}$
Double Precision	64	53	2^{-1022} a 2^{11023}	$2,23 \times 10^{-308}$ a $1,79 \times 10^{308}$
Extended Double Precision	80	64	2^{-16382} a 2^{16383}	$3,37 \times 10^{-4932}$ a $1,18 \times 10^{4932}$

Cuadro: Codificación de números en Punto Flotante

Funcionamiento básico

- La pila (stack) es un área de memoria contigua, referenciada por un segmento cuyo selector está siempre en el registro SS del procesador.
- El tamaño de este segmento en el modo IA-32, puede llegar hasta 4 Gbytes de memoria, en especial cuando el sistema operativo utiliza el modelo de segmentación Flat (como veremos en clases subsiguientes).
- El segmento se recorre mediante un registro de propósito general, denominado habitualmente en forma genérica stack pointer, y que en estos procesadores según el modo de trabajo es el registro SP, ESP, o RSP (16, 32, o 64 bits respectivamente).
- Para guardar un dato en el stack el procesador tiene la instrucción PUSH, y para retirarlo, la instrucción POP.
- Cada vez que ejecuta PUSH, el procesador primero decrementa el stack pointer (SP, ESP, o RSP) y luego escribe el dato en el stack, en la dirección apuntada por el registro de segmento SS, y el stack pointer correspondiente al modo de trabajo.
- Cada vez que ejecuta un POP, el procesador lee el ítem apuntado por el pas SS y el stack pointer, y luego incrementa éste último registro.

Funcionamiento básico

El stack es un segmento expand down, ya que a medida que lo utilizamos (PUSH) su registro de desplazamiento se decremente hacia las direcciones mas bajas de memoria (numéricamente menores).

Estas operaciones se pueden realizar en cualquier momento, pero hablando mas generalmente, podemos afirmar que la pila se usa cuando:

- Cuando llamamos a una subrutina desde un programa en Assembler, mediante la instrucción CALL.
- Cuando el hardware mediante la interfaz adecuada envía una Interrupción al Procesador.
- Cuando desde una aplicación, ejecutamos una Interrupción de software mediante la instrucción INT type.
- Cuando desde un lenguaje como el C se invoca a una función cualquiera.

Funcionamiento básico

- El stack pointer debe apuntar a direcciones de memoria alineadas de acuerdo con su ancho de bits.
- Por ejemplo, el ESP (32 bits) debe estar alineado a double words.
- Al definir un stack en memoria se debe cuidar el detalle de la alineación.
- El tamaño de cada elemento de la pila se corresponde con el atributo de tamaño del segmento (16, 32, o 64 bits), es decir, con el modo de trabajo en el que está el procesador, y no con el del operando en sí.
- Ej: PUSH AL, consume 16, 32, o 64 bits dependiendo del tamaño del segmento. Nunca consume 8 bits.
- El valor en que se decrementa el Stack Pointer se corresponde con el tamaño del segmento (2, 4, u 8 bytes).