Teórica 1

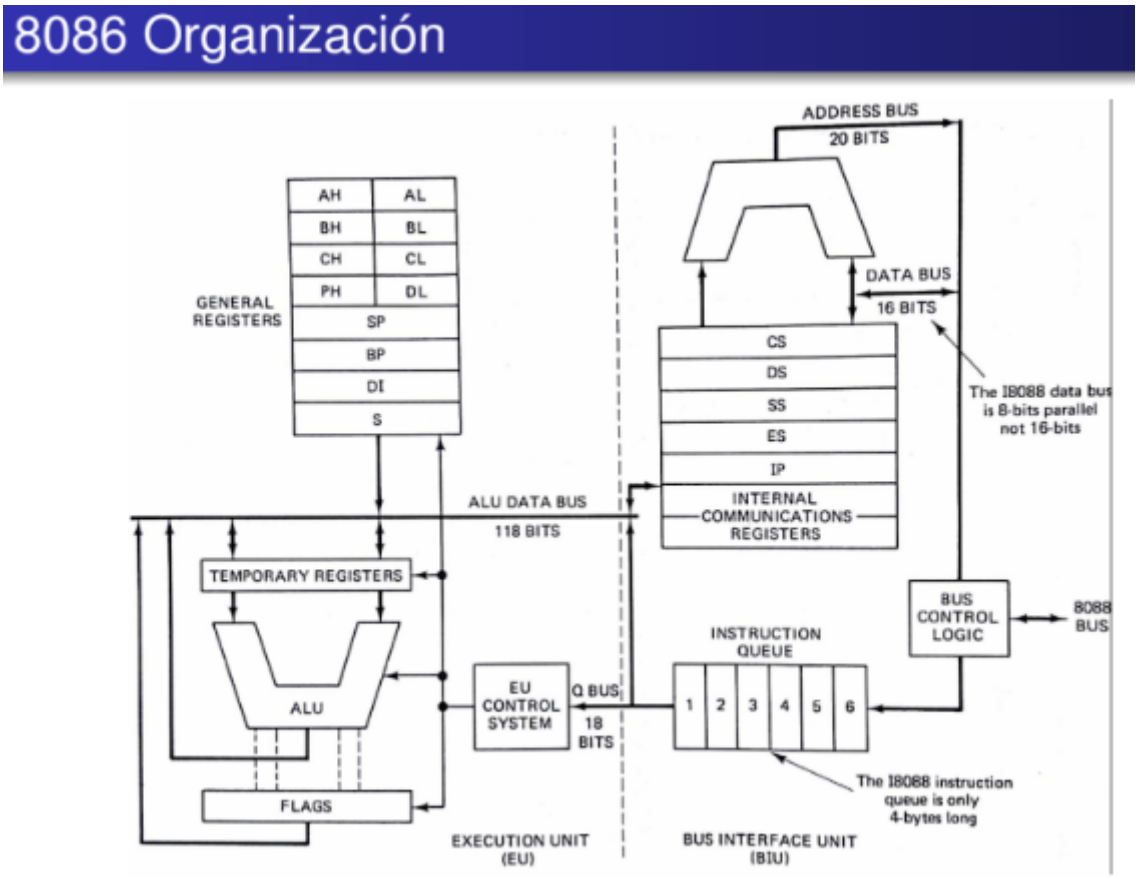
Orígenes

Las instrucciones complejas tienen sentido en un paradigma donde la prioridad era minimizar el consumo de memoria. Con un pequeño patrón de bits, la CPU puede realizar operaciones que del contrario estarían almacenadas en memoria. Hoy en día se diseñan procesadores con el foco puesto en el mínimo consumo de energía.

Si tenemos instrucciones complejas nuestro código va a ser más compacto y va a ocupar menos memoria.

Este paradigma estuvo presente durante décadas, e Intel se comprometió a mantener compatibilidad.

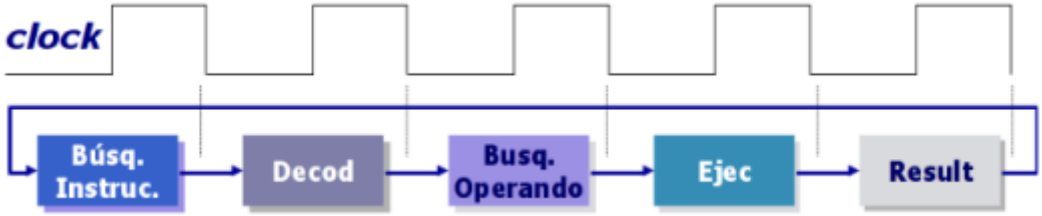
En 1978 presenta la familia iAPx86 de la mano de su nave insignia procesador 8086, ye compromete a mantener compatibilidad ascendente en los modelos subsiguientes de procesadores para satisfacer esta demanda.
Esto fue uno de los motivos de la decisión de IBM de adoptar esta familia de procesadores como base para su PC.



Pipeline

Maquina de estados elemental

En la década del 40 Von Newman definió un modelo básico de CPU, que a esta altura esta más que superado. Sin embargo algunos conceptos de ese modelo viven en los más modernos procesadores. Uno de ellos es la maquina de ejecución.



En las primeras generaciones de microprocesadores cada etapa de este ciclo se ejecutaba en un ciclo de clock, y la CPU entera estaba dedicada a esa tarea. Ejecutar una instrucción insumía de varios ciclos de clock...

Pipeline

Arquitectura que permite crear el efecto de superponer en el tiempo, la ejecución de varias instrucciones a la vez.

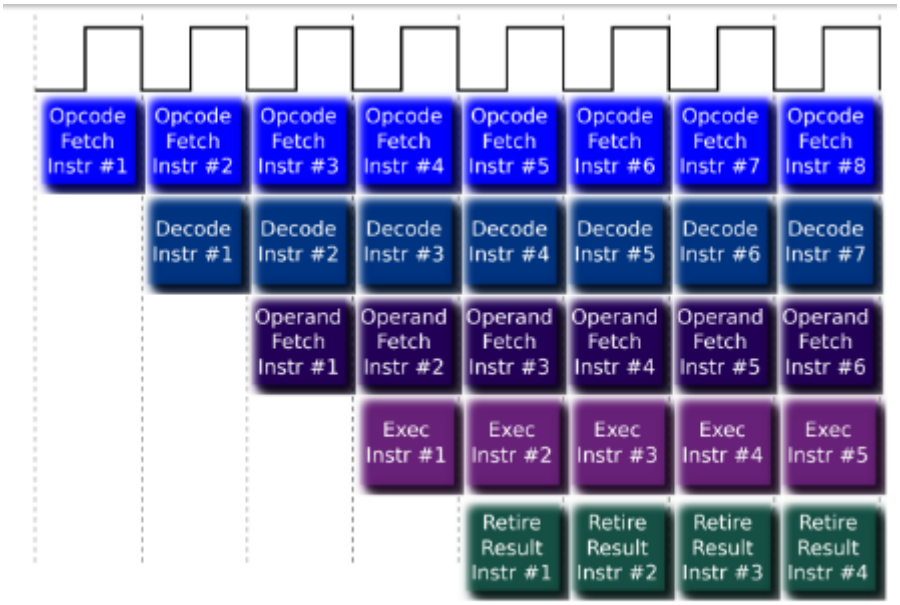
- Con el se formaliza el concepto de Instruction Level Paralelism (ILP).

Requiere muy poco o ningún hardware adicional.
Solo necesita que los bloques del procesador que resuelven la maquina de estados para la ejecución de una instrucción, operen en forma simultanea.

- Se logra si todos los bloques funcionales trabajan en paralelo pero cada uno en una instrucción diferente.

Es algo parecido al concepto de una línea de montaje, en donde cada operación se descompone en partes, y se ejecutan en un mismo momento diferentes partes de diferentes operaciones.

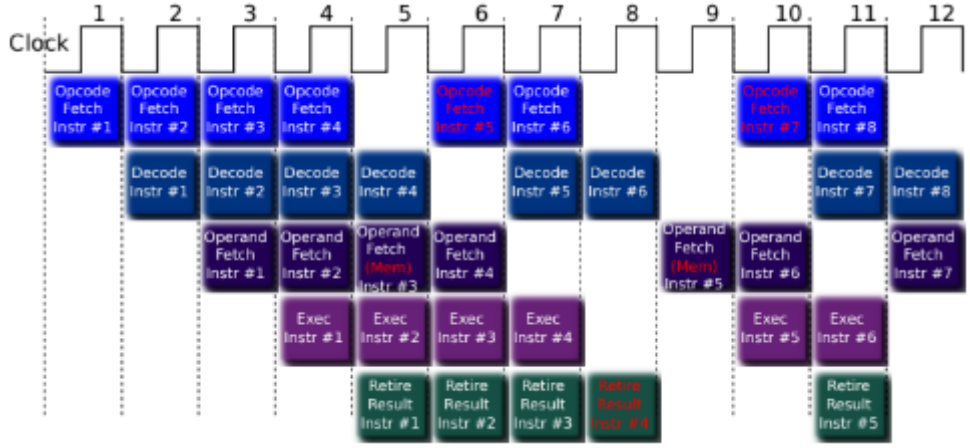
- Cada parte se denomina etapa (stage).



Este modelo es teórico y representa la situación ideal.

Obstaculos Estructurales: Ejemplo

- Tenemos un procesador que solo tiene una etapa para acceder a memoria y la comparte para acceso a datos e instrucciones.
- En el caso de que se necesite un operando de memoria, el acceso para traer este operando interferirá con la búsqueda del operando de una instrucción mas adelante del programa.
- También interferirá con el Fetch de la siguiente instrucción.

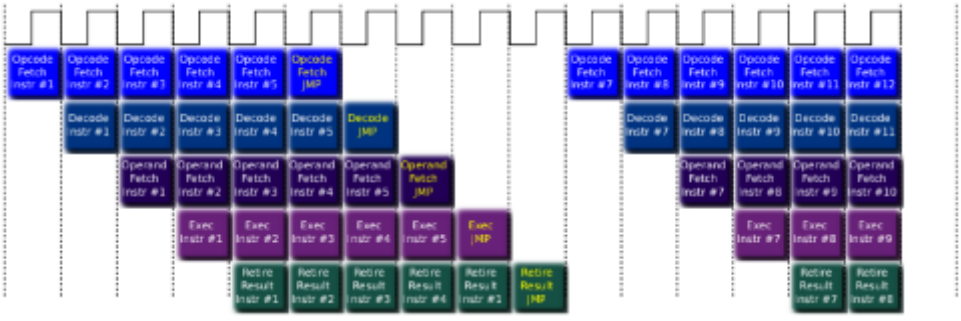


- Por cada Obstaculo, pospone una operación. *CPI = CPI + 1*
- En general la cantidad de CPI que se incrementan es igual a la cantidad de concurrencias menos 1, en el lapso considerado.

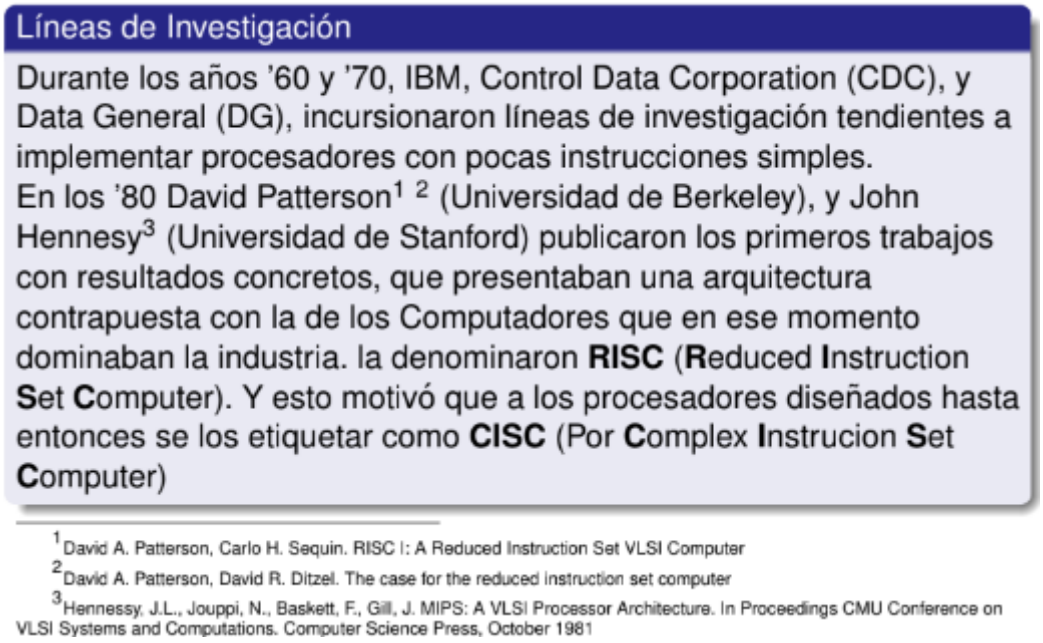
Obstaculos de Control

- Un branch es la peor situación en pérdida de performance.
- Un branch es una discontinuidad en el flujo de ejecución.
- El pipeline busca instrucciones en secuencia.
- El branch hace que todo lo que estaba pre procesado deba descartarse. Y el pipeline se vacía debiendo transcurrir *n-1* ciclos de clock hasta el próximo resultado. Siendo **n** la cantidad de etapas del pipeline. Esto se conoce como **branch penalty**.

El problema del branching es que corta la secuencialidad del pipeline. Es decir, hay que vaciar todo el pipeline de donde veníamos procesando instrucciones y empezar de cero con la nueva subrutina o el lugar en memoria a donde saltemos.



RISC



- Se analizaron los obstáculos en un pipeline.
- En particular los obstáculos de datos impactan cuando involucran accesos a memoria para buscar operandos.
- Claramente si se evita que los operandos se accedan en memoria, en lugar de cargarlos previamente en un registro tal vez el pipeline pueda mejorarse.

Los Mandamientos RISC

1. Se dispone de un juego de registros numeroso, todos de propósito general.
2. Las instrucciones se ejecutan en un solo ciclo de clock.
3. Las instrucciones derivan en codigos de operacion de igual formato y tamano.
4. Las instrucciones deben ser sencillas de decodificar. Los numeros de registros se deben tener la misma ubicacion en los codigos de la instruccion y deben requerir la misma cantidad de bits para su decodificacion.
5. No se utiliza micro codigo para decodificar instrucciones (no hay instrucciones complejas, como DIV o MUL).
6. Los datos en memoria se acceden mediante instrucciones simples de transferencia: LOAD y STORE.

Procesadores IA-32 e Intel64

Modos de Operación (los procesadores IA-32)

1. Modo Real
2. Modo Protegido
3. Modo Mantenimiento del Sistema
4. Modo extendido a 64 bits (IA-32e)

- - Submodo Compatibilidad
- - Submodo 64 bits

Modo Real

En este modo el procesador implementa el entorno de operación del 8086, con algunas extensiones:

- 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)
- Desde este modo se puede pasar por software al Modo Protegido o al Modo Mantenimiento del Sistema.
- Aunque nadie trabaja en este modo, es el modo de arranque de cualquier procesador IA-32 e Intel 64 actual y futuro. . . Delicias de la **compatibilidad**.
  En este modo es el que arrancan los procesadores.

Modo protegido

Modo preferido de uso de todos los procesadores.

- En este modo se implementa multitasking
- Este es el modo por excelencia de los procesadores de esta familia inaugurado por el procesador 80286, primer procesador con capacidad multitarea, a pesar de su arquitectura de 16 bits similar a la de su antecesor 8086.
- A partir del 80386 se pasa a 32 bits (nace IA-32) y se despliega un espacio de direccionamiento de 4 Gbytes, extensible a 64 Gbytes.
- 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 di ferente del DOS original, y es en general una tarea mas.

Modo Mantenimiento

El procesador ingresa a este modo por dos caminos:

- Activación de la señal de interrupción #SMM.
- mediante un mensaje SMI desde su APIC local (Ver Interrupciones y SMP mas adelante)

- 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 en el modo de operación en el que se encontraba.

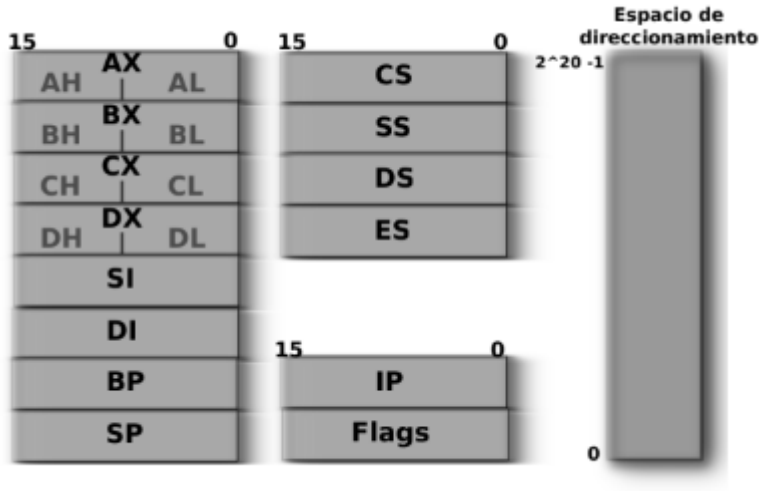
Extendido a 64 bits

Los procesadores Intel 64 ademas de los modos de trabajo de los procesadores IA-32 incluyen un modo IA-32e en el que se activa la arquitectura de 64 bits.
Para pasar a este modo, el procesador debe estar trabajando en modo protegido, con paginacion habilitada, y PAE activo (Como estudiaremos en Paginacion)
En IA-32e a su vez existen dos sub-modos (nada es simple en este mundo).

1. Sub modo Compatibilidad (correr en modo 64 bits aplicaciones de 32)
2. Sub modo 64 bits.

Modelo del Programador de Aplicaciones

Entorno básico de ejecución en 16 bits



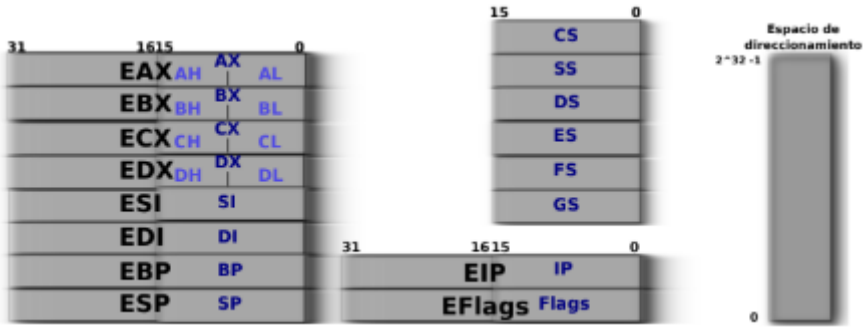
Los primeros 4 registros pueden ser divididos en registros independientes de 8 bits.

- El registro AX/AL funcionan como el viejo registro *acumulador*, soporta todas las operaciones posibles. Es el registro más rico en cuestión de cantidad de cosas que podemos hacer con el.
- El BX/BL se llama registro *base*. Obra como puntero base a memoria.
- El CX, o *counter*, puede usarse como contador, y de hecho, muchas instrucciones de alto nivel lo utilizan como tal.
- El DX sirve para almacenar datos. Su próposito es no tener ningún propósito en especial.
- Los registros SI y DI funcionan también como punteros a memoria. Puedo usarlos para otras cosas, pero si los uso como punteros trabajan como índices. Es decir, registros que se pueden autoincrementar para recorrer un vector, por ejemplo.
- - S y D también se llaman así por 'source' y 'destination'.
- BP y SP son *base pointer* y *stack pointer*
- CS, SS, DS y ES son registros de **segmento**[1]. Cada segmento define un trozo de memoria donde puede estar el código (CS), la pila, *o stack* (SS), y dos para datos (DS y ES).
- Por último, el IP o *instruction pointer* y el registro de flags.

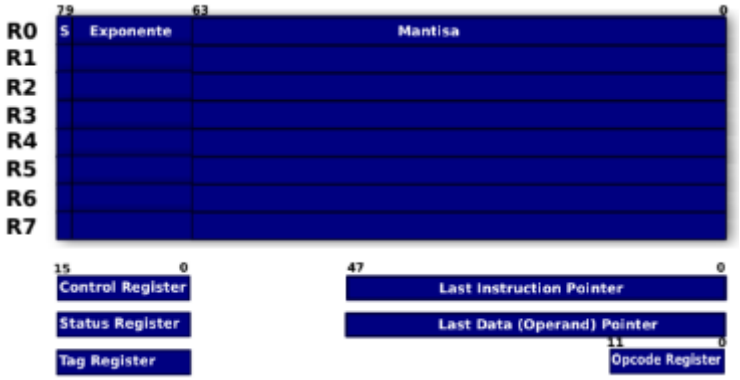
- El espacio de direccionamiento de memoria es de 1 MB.

Arquitectura de 32 bits compatible

El modelo de 16 bits esta incluido en el de 32.

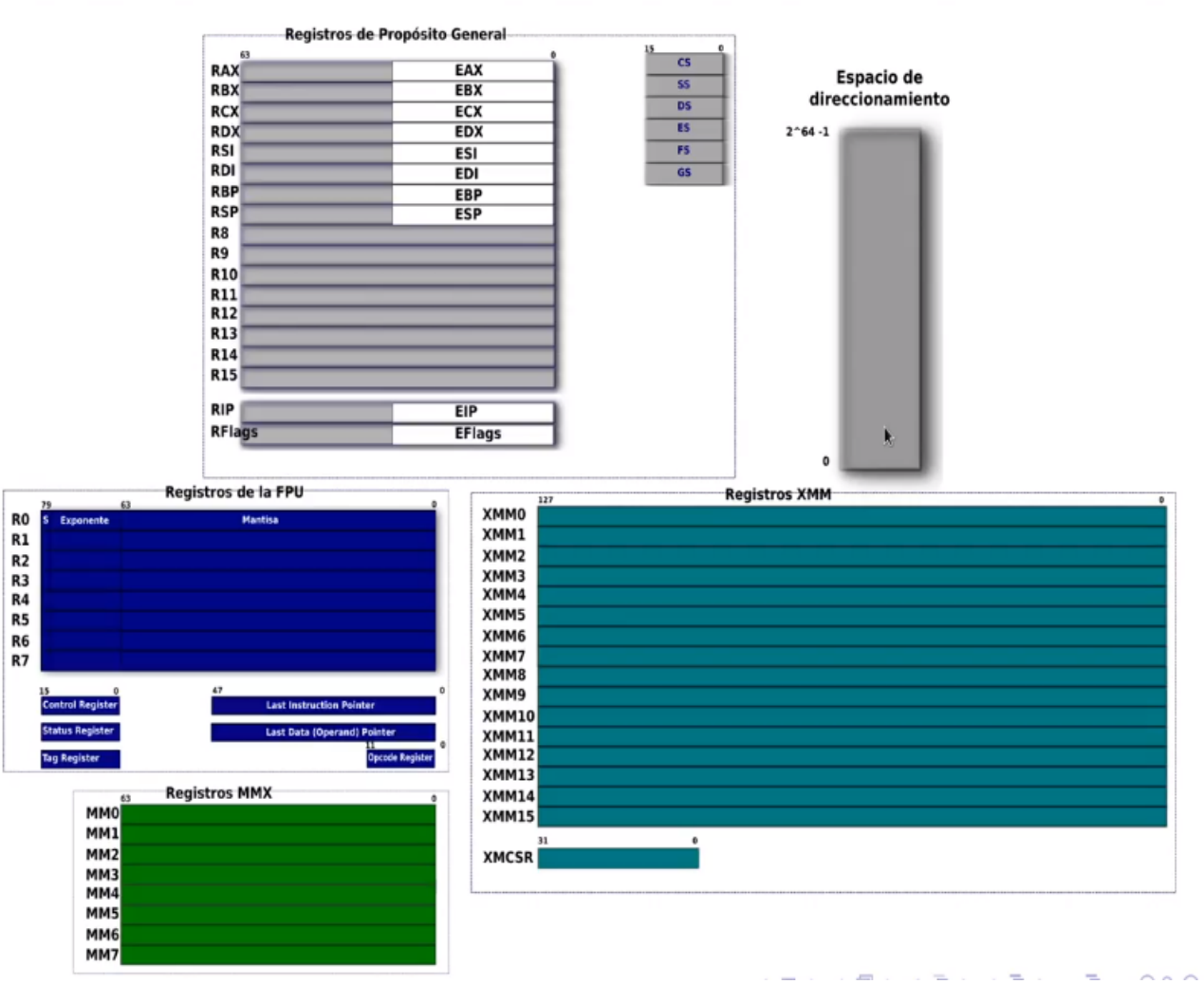


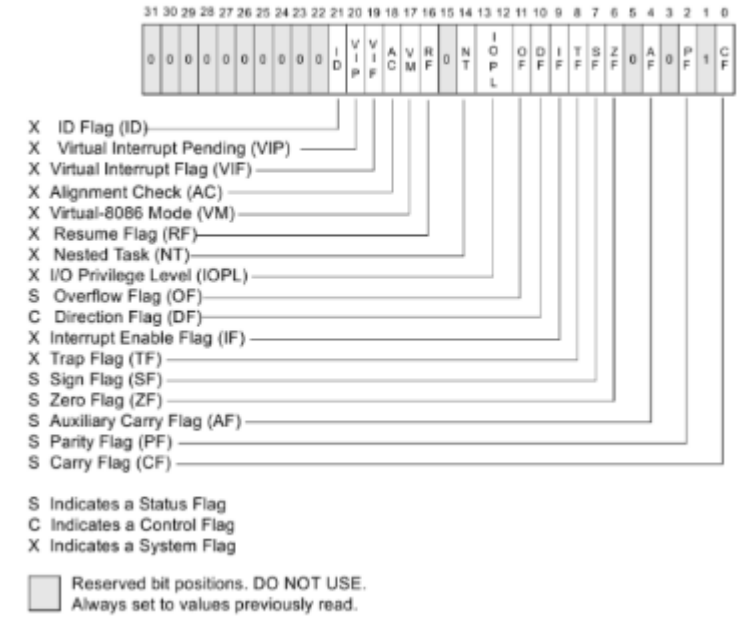
- El espacio de direccionamiento pasa a ser de 4 GB.
- Floating Point Unit.
- - 8 registros de 80 bits de ancho.



- Recursos DSP (digital signal processing, o video)
- - Registros MMX, *data level paralellism*.

Extensiones de 64 bits





Operandos

Obtención

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

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

Almacenamiento

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

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

Modos de 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), esta 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),

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.

```
5 ascii :
6        add al , ' 0 '
7        cmp al , ' 9 '
8        jle listo
9        add al , 'A '- ' 9 '-1
10 listo : ret
```

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 segun sea IA-32 o Intel 64 respectivamente,
- Registros de Control
- Registros de Debug
- MSR's (Model Specific Registers)

```
1 inc rd x                  ; Inc rementa 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 ) , s t ( 3 ) ; ST( 0 ) = ST( 0 ) * ST( 3 )
5 sqrt p d xmm2,xmm6        ; Raız cuadrada de dos double precision FP
6                           ; empaquetados en xmm6. Resultados en xmm2.
```

Memoria

Generalmente los procesadores RISC no tienen direccionamiento a memoria.

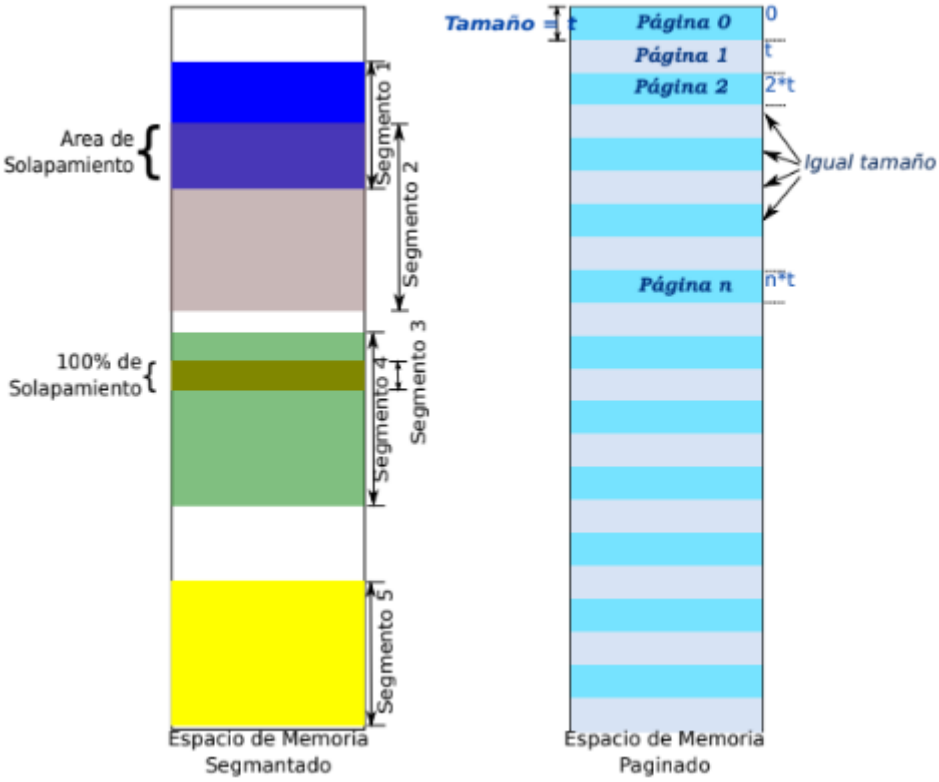
Espacio físico

Los procesadores IA-32 organizan la memoria como una secuencia de bytes, direccionables a través de su Bus de Address.

- La memoria conectada a este bus se denomina **memoria física**.
- El espacio de direcciones que pueden volcarse sobre este bus se denomina **direcciones físicas**.

Segmentación vs paginación

Por diversos motivos que en su momento tuvieron sentido, Intel definio organizar el espacio de direccionamiento de la Familia iAPx86 en segmentos. El compromiso de compatibilidad ato a los siguientes procesadores a mantener este esquema.



Espacio lógico

- 4 registros de segmento para almacenar hasta 4 selectores de segmento.
- Registros de 16 bits los segmentos tienen a lo sumo 64K de tamaño
- Expresión de las direcciones en el modelo de programación mediante dos valores:

1. Identificador del segmento en el que se encuentra la variable o la instrucción que se desea direccionar,

2. Desplazamiento, offset, o dirección efectiva a partir del inicio de ese segmento en donde se encuentra efectivamente

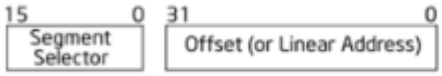


Figura: IA-32: Dirección lógica

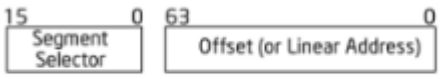


Figura: Intel® 64 : Dirección Lógica

Segmentos para cada dirección lógica

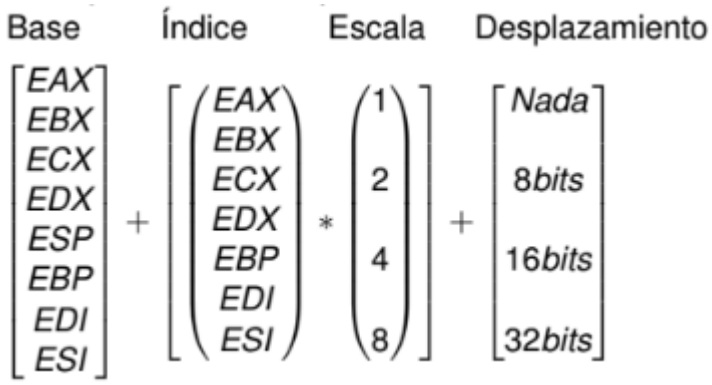
- El valor del segmento en una dirección lógica, puede especificarse de manera implícita.
- Por lo general este es el modo en que se hace, aunque es posible explicitar con que segmento se desea direccionar un operando de memoria determinado.
- Si no se especifica explícitamente el segmento como parte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

Desplazamiento para ubicar operandos en memoria

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.
- Indice: 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 Indice: Puede valer 2, 4, u 8.



En general la expresión general que representa el calculo interno del procesador es:
DireccionEfectiva = Base + (Indice ∗ escala) + desplazamiento

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

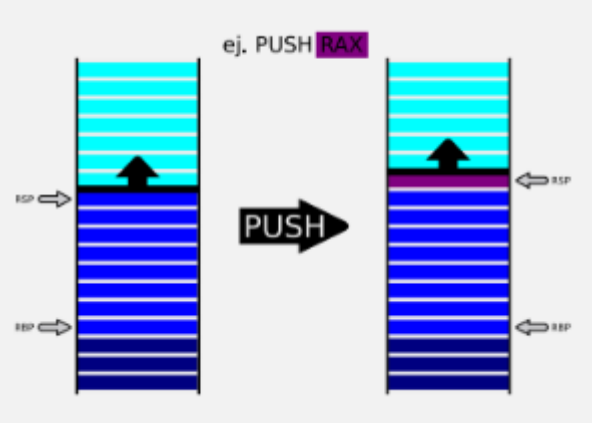
No voy a copiar la data de todos los modos de direccionamiento, es una cantidad absurda de texto

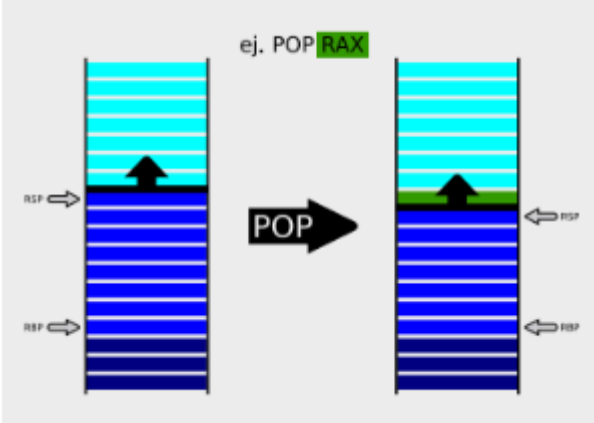
Pila

La pila (stack) es un´area 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.
- 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).

Funcionamiento básico





- 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 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´ıtem apuntado por el par SS : stack pointer, y luego incrementa´este ultimo registro.

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

Usos del stack

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

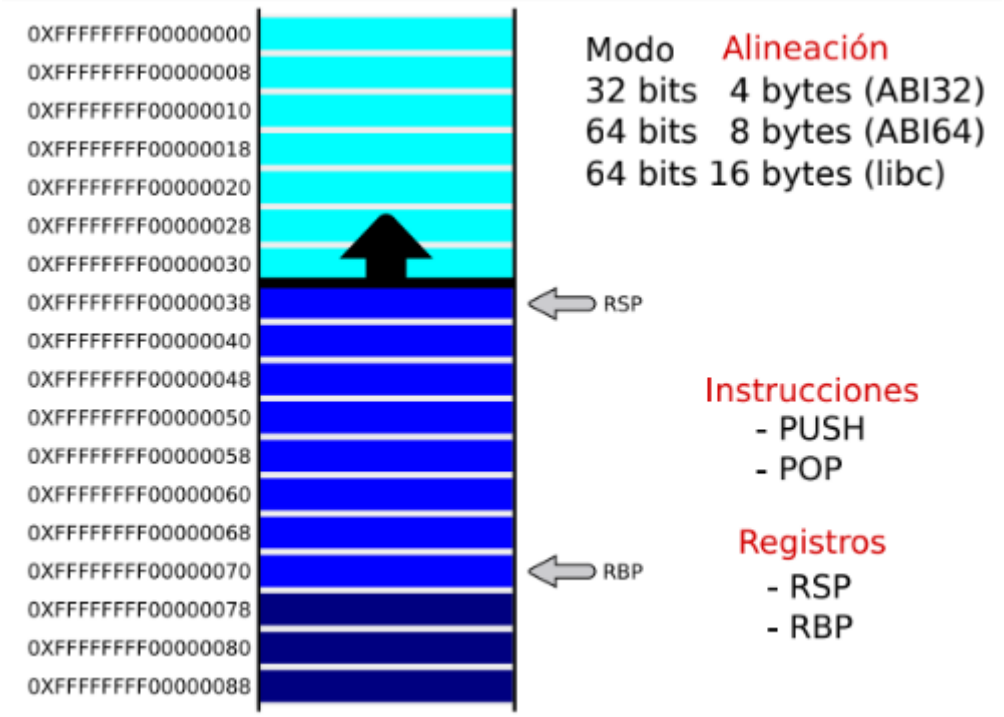
- Cuando llamamos a una subrutina desde un programa en Assembler, mediante la instrucción CALL.
- Cuando el hardware mediante la interfaz adecuada envía una Interrupción al Procesador.
- Cuando desde una aplicación, ejecutamos una Interrupción de software mediante la instrucción INT type.

- Cuando desde un lenguaje como el C se invoca a una función cualquiera.

Alineación del stack

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).



---

1. Hay dos formas de administrar memoria. Por segmento y por página. ↩