# ARQUITECTURA DE COMPUTADORAS.

EL MICROPROCESADOR 8086.

2C – 2024.

FCyT – UADER – Oro Verde.

UADER FCyT

# CONTENIDOS

- La arquitectura interna del 8086.
- Los registros de 8086 y sus funciones específicas.
- El funcionamiento de las banderas de 8086.
- La técnica de segmentación de la memoria utilizada por la familia x86.
- Cómo realizar cálculos de direcciones de memoria.
- Diferentes modos de direccionamiento del 8086.

# 8086 -> CARACTERISTICAS

**Arquitectura de 16 bits:** El 8086 es un procesador de 16 bits, lo que significa que maneja instrucciones y datos en palabras de 16 bits.

**Bus de Datos de 16 bits.**

**Bus de direcciones de 20 bits.**

**Registros de Propósito General:** Cuenta con ocho registros de propósito general (AX, BX, CX, DX, SI, DI, BP, SP), cada uno de 16 bits, que se pueden usar para realizar operaciones aritméticas y lógicas.

**Registro de Segmento:** Tiene registros de segmento (CS, DS, ES, SS) que apuntan a segmentos de memoria. Estos registros se combinan con los registros de desplazamiento para formar direcciones físicas de 20 bits.

# 8086 -> CARACTERISTICAS

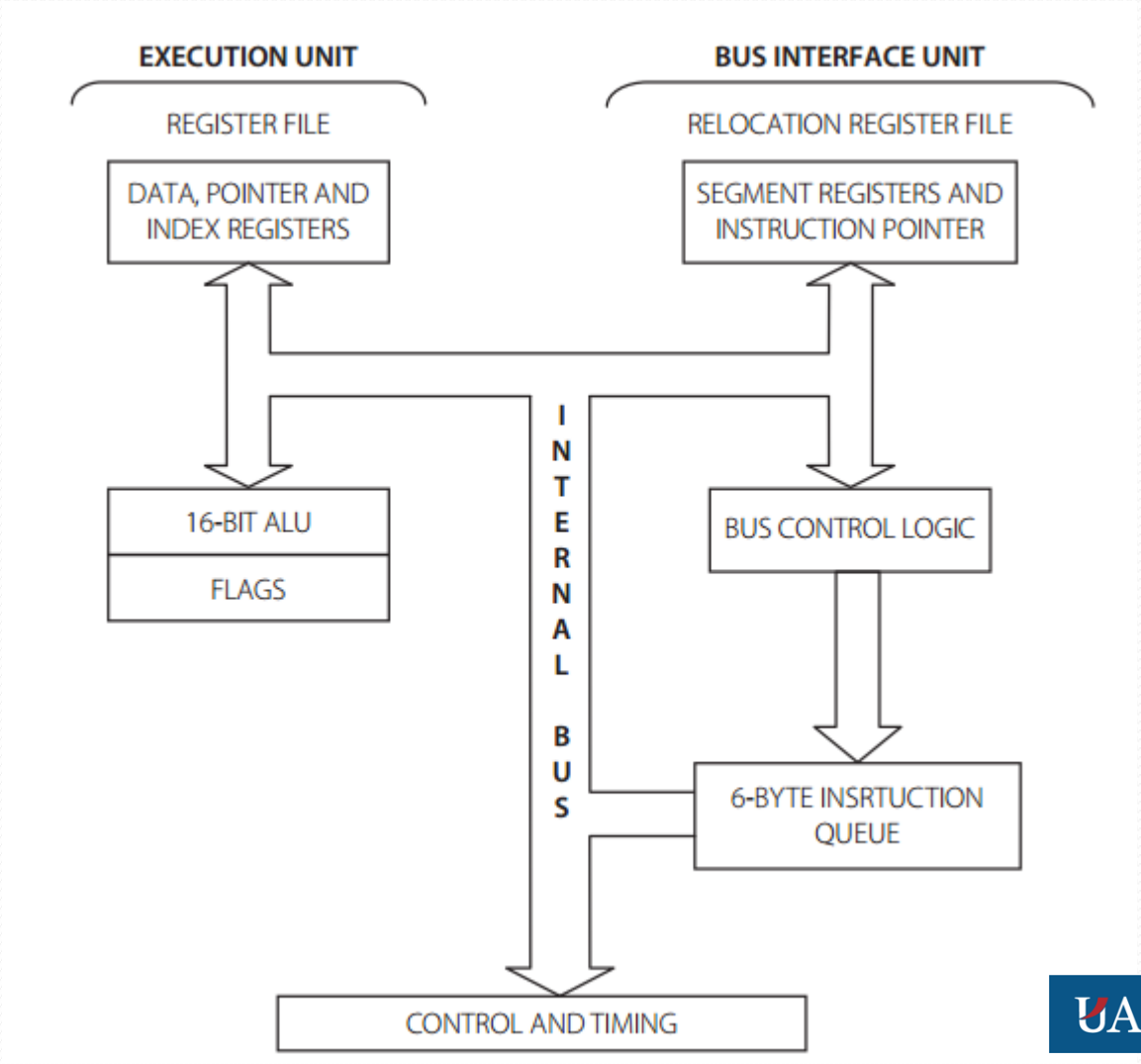
**Registro de Puntero de Instrucción (IP):** Indica la dirección de la próxima instrucción que se ejecutará.

**Arquitectura Von Neumann:** El 8086 sigue una arquitectura Von Neumann, lo que significa que las instrucciones y los datos se almacenan en la misma memoria.

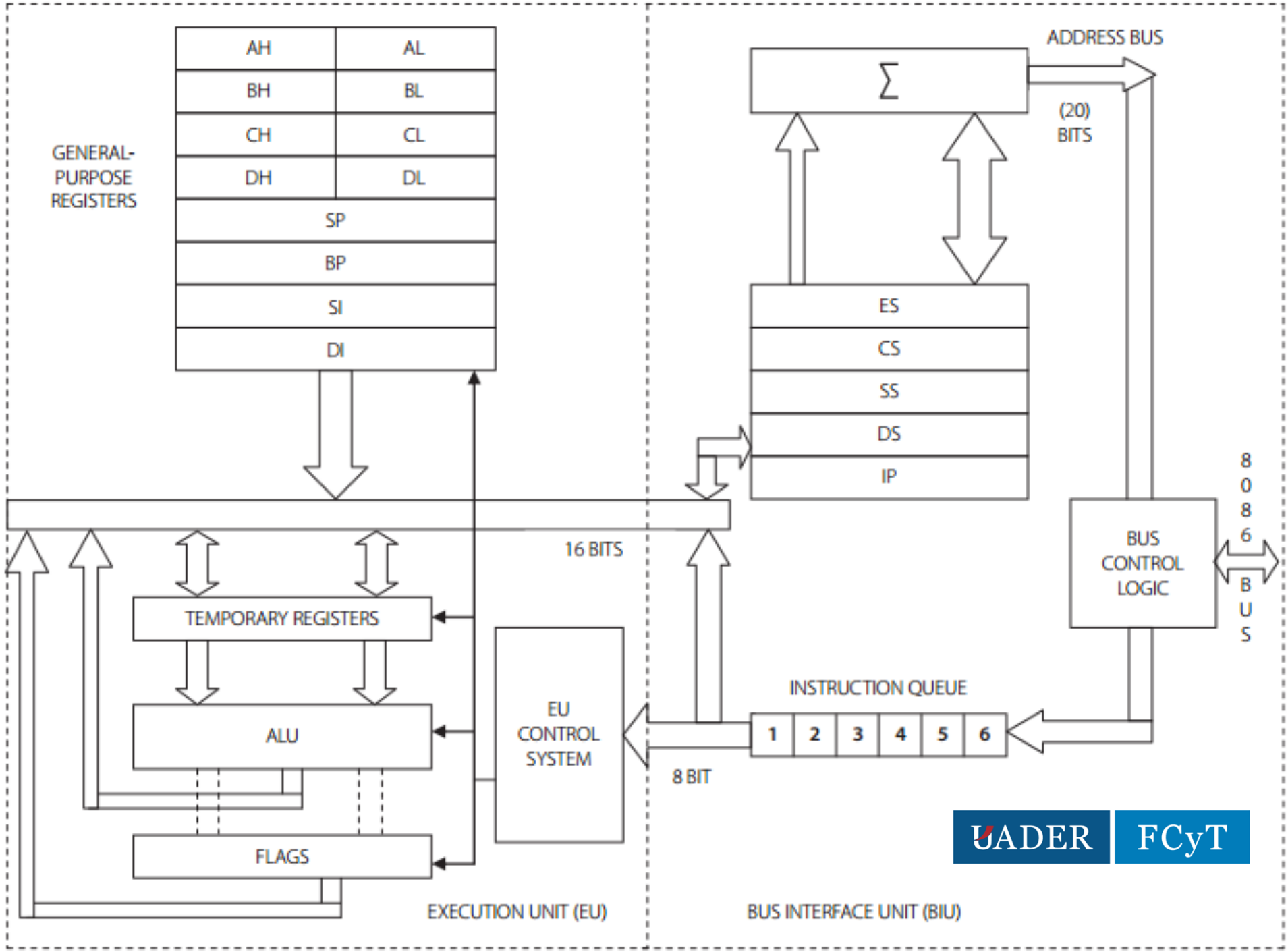
**Velocidad de Reloj:** Originalmente operaba a una velocidad de reloj de 5 MHz, pero versiones posteriores aumentaron esta velocidad.

**Instrucciones de Transferencia de Bloques:** Introduce instrucciones como MOVSB, MOVSW para transferir bloques de datos entre memoria y registros.

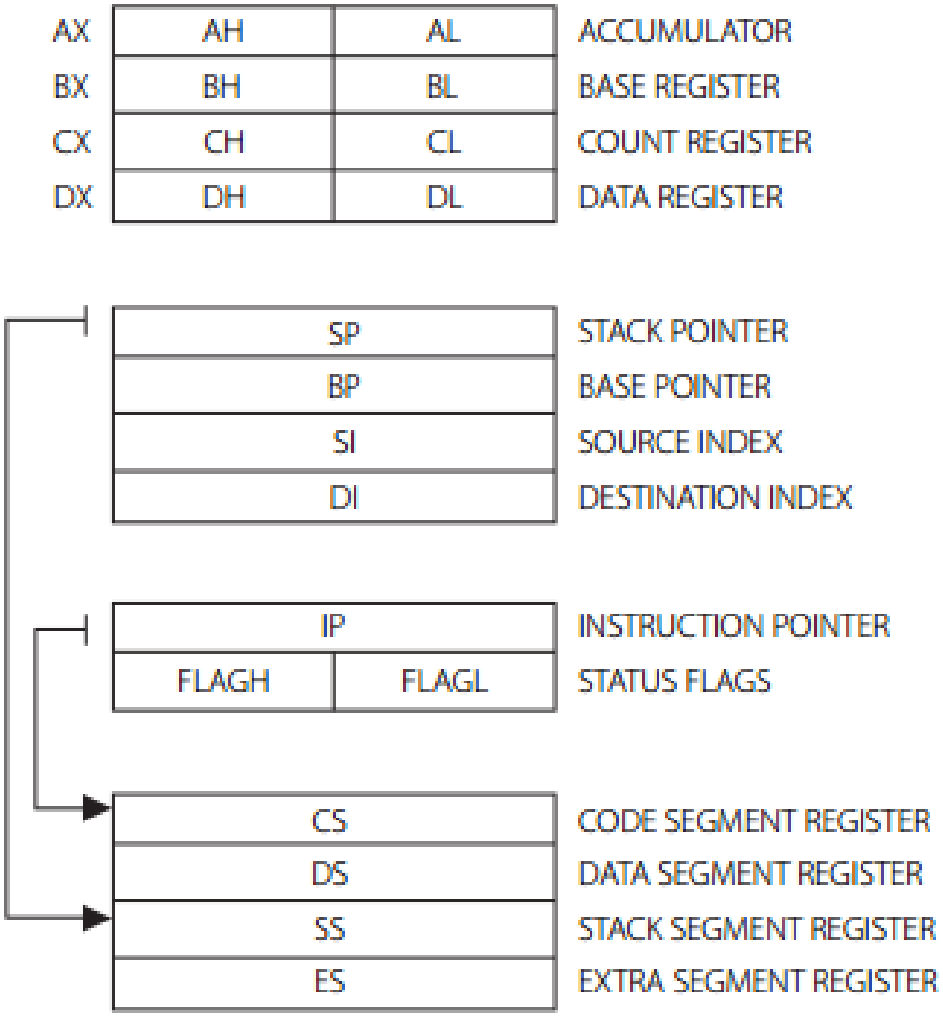
# 8086 -> DIAGRAMA DE BLOQUES INTERNO




UADER FCyT

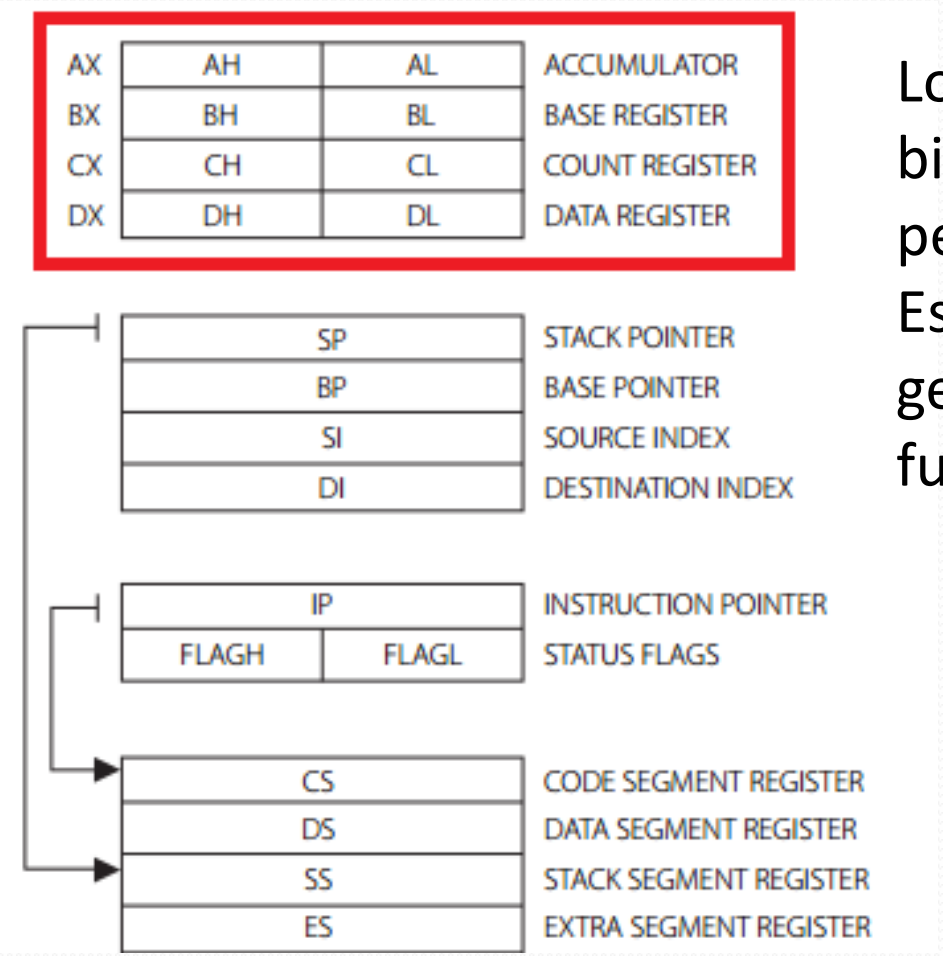

AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
SI
DI
GENERAL-
PURPOSE
REGISTERS
ADDRESS BUS
Σ
(20)
BITS
ES
CS
SS
DS
IP
16 BITS
8
0
8
6
BUS
CONTROL
LOGIC
B
U
S
TEMPORARY REGISTERS
ALU
FLAGS
EU
CONTROL
SYSTEM
INSTRUCTION QUEUE
1
2
3
4
5
6
8 BIT
EXECUTION UNIT (EU)
BUS INTERFACE UNIT (BIU)
UADER
FCyT

# 8086 -> REGISTROS INTERNOS




UADER FCyT

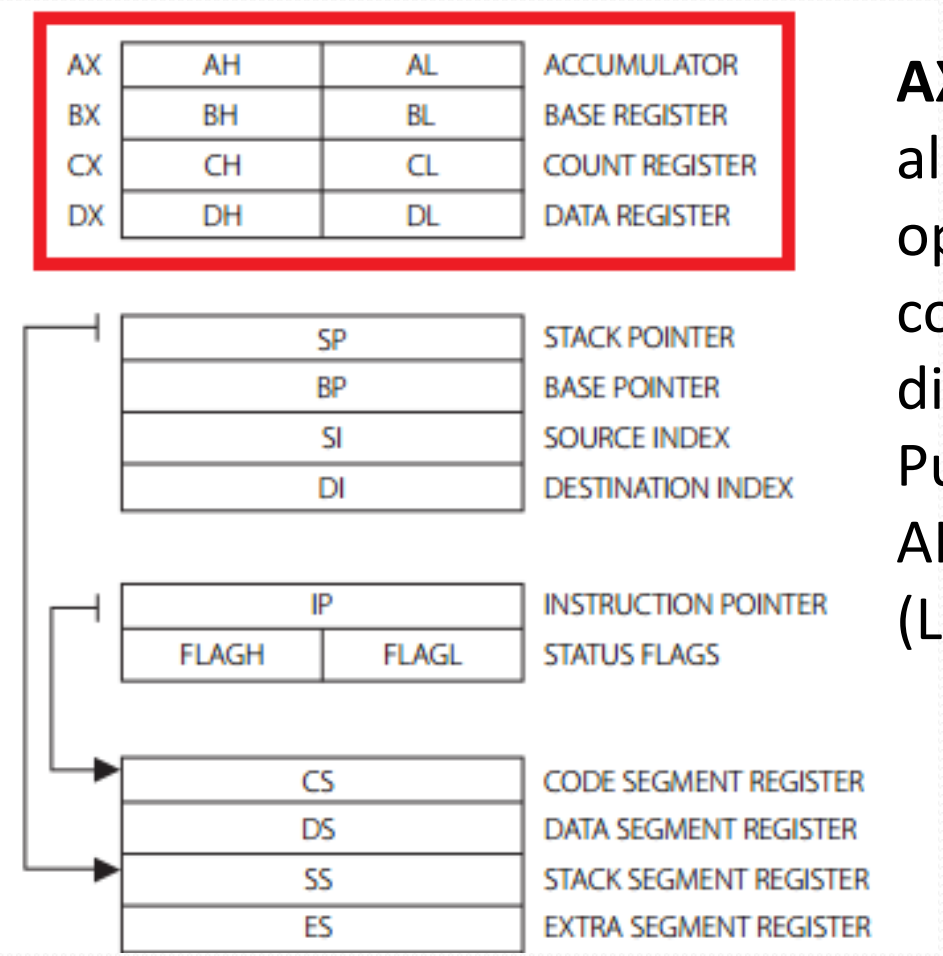
# 8086 -> REGISTROS DE DATOS




Los registros de datos son de 16 bits, aunque están divididos. lo que permite su acceso en 8 bits.
Estos registros son de propósito general aunque todos tiene alguna función por defecto.

UADER FCyT

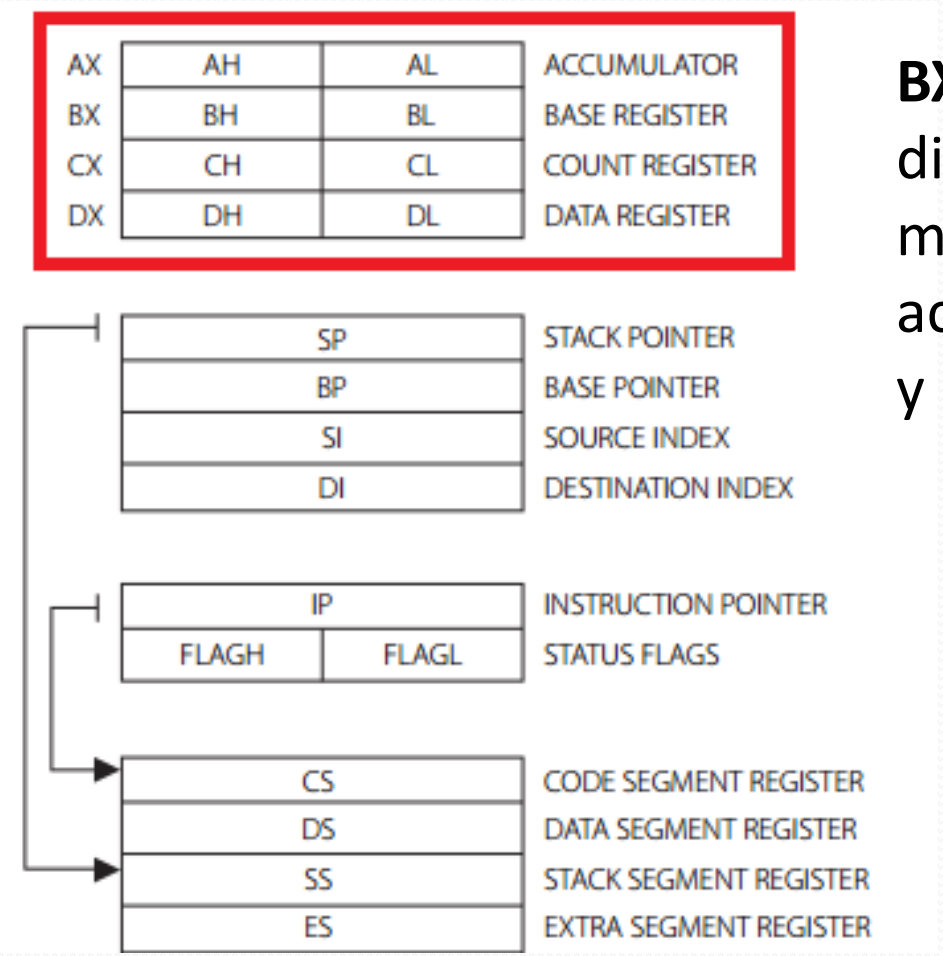
# 8086 -> REGISTROS DE DATOS




**AX (acumulador):** se usa para almacenar el resultado de las operaciones, es al único registro con el que se puede hacer divisiones y multiplicaciones. Puede ser accedido en 8 bits como AH para la parte alta (HIGH) y AL (LOW) para la parte baja.

UADER FCyT

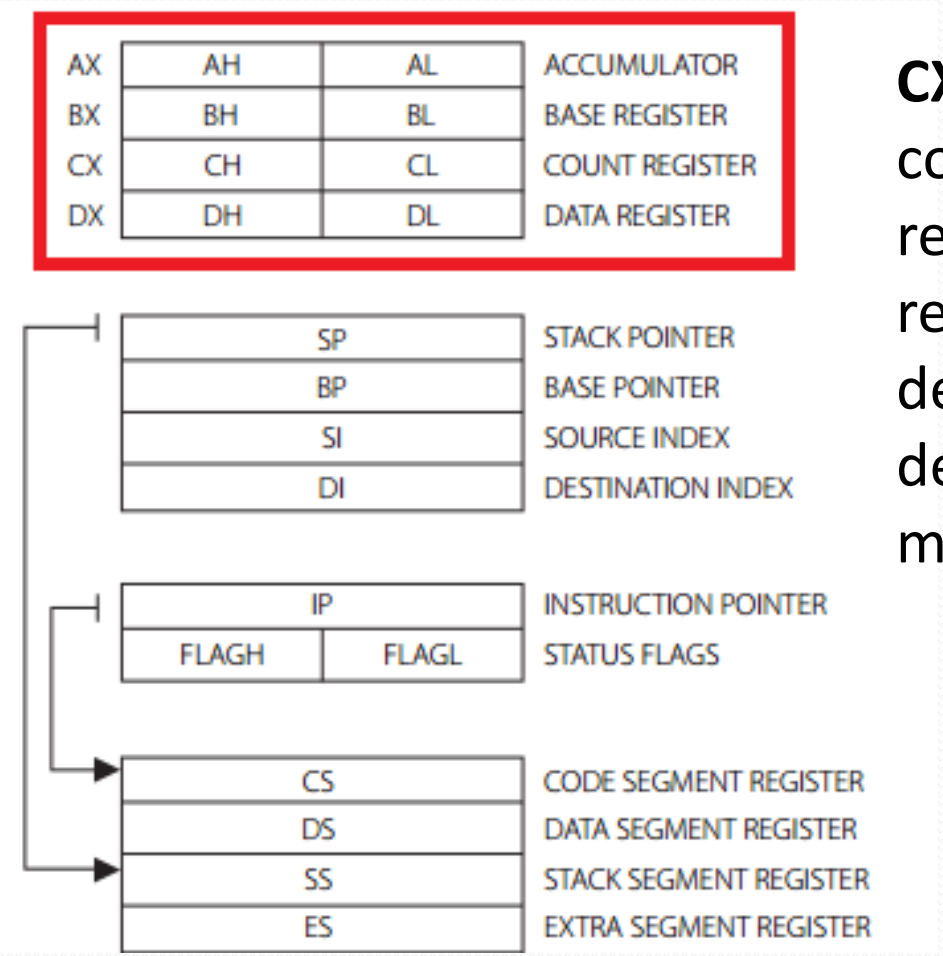
# 8086 -> REGISTROS DE DATOS




**BX (registro base):** almacena la dirección base para los accesos a memoria. También puede accederse como BH y BL, parte alta y baja respectivamente.

UADER FCyT

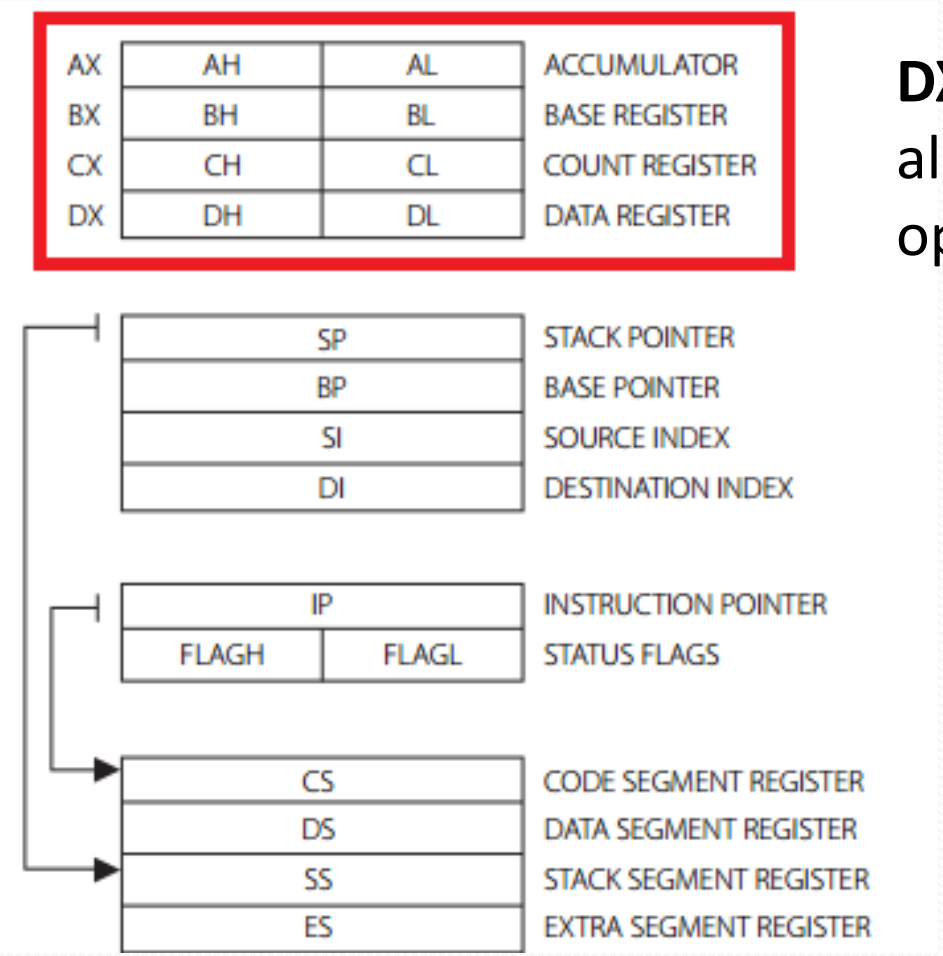
# 8086 -> REGISTROS DE DATOS




**CX (contador):** actúa como contador en los bucles de repetición. CL (parte baja del registro) almacena el desplazamiento en las operaciones de desplazamiento y rotación de múltiples bits.

UADER FCyT

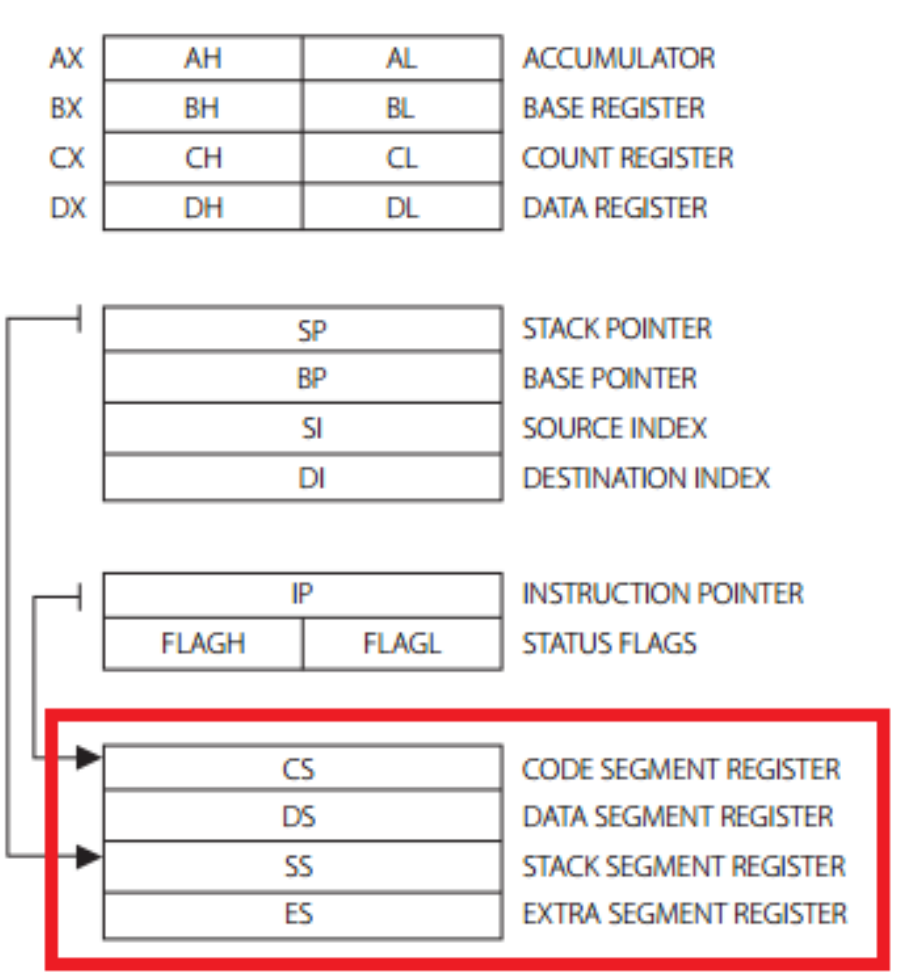
# 8086 -> REGISTROS DE DATOS




**DX (datos):** es usado para almacenar los datos de las operaciones.

UADER FCyT

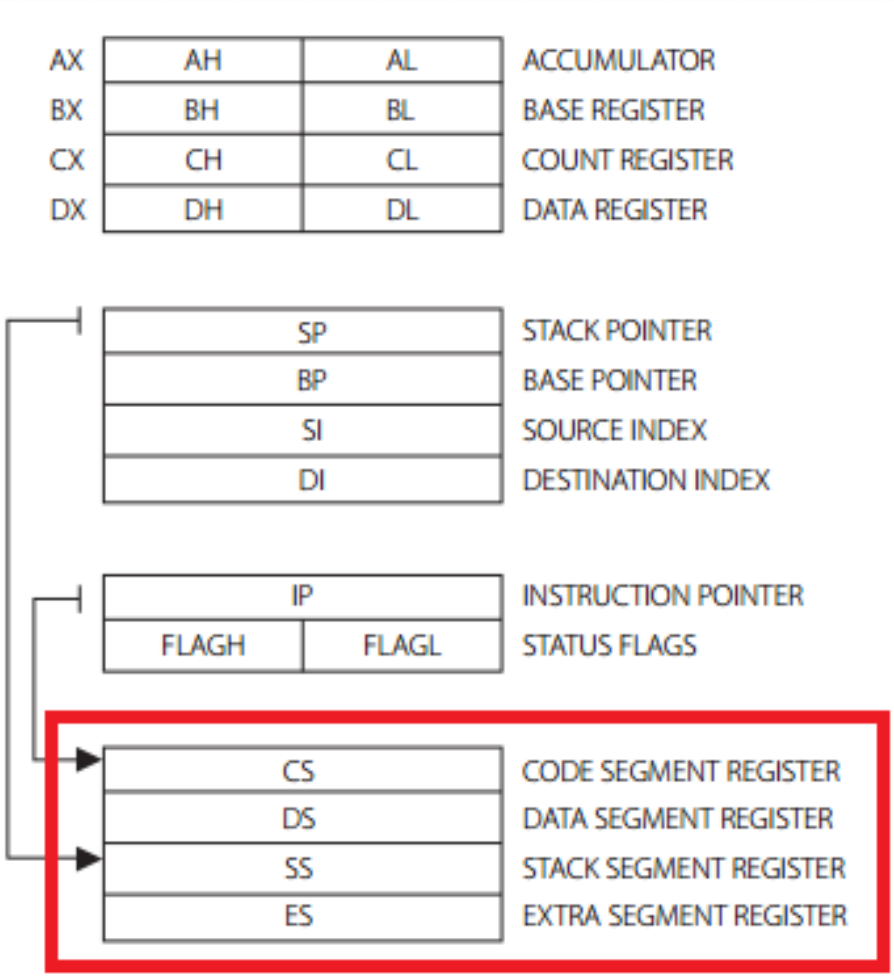
# 8086 -> REGISTROS DE SEGMENTO




Los registros de segmento son de 16 bits y contienen el valor de segmento.

UADER FCyT

# 8086 -> REGISTROS DE SEGMENTO

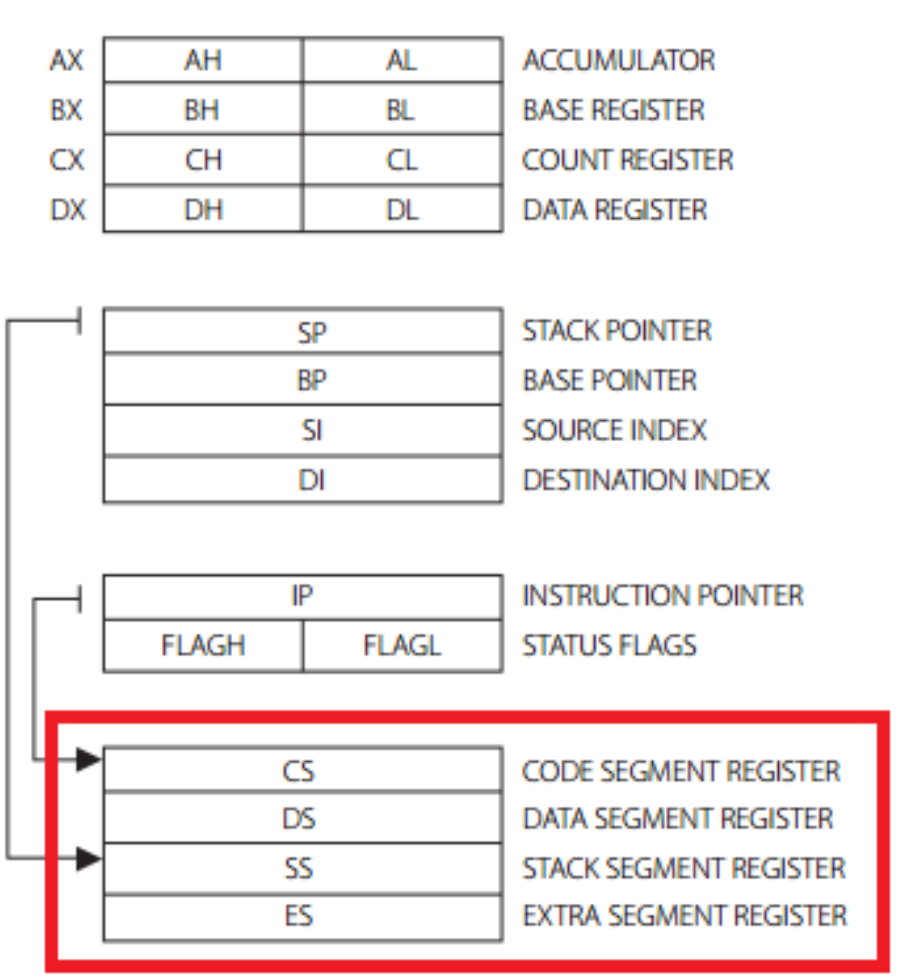



**CS (segmento de código):** contiene el valor de segmento donde se encuentra el código. Actúa en conjunción con el registro IP para obtener la dirección de memoria que contiene la próxima instrucción.

Este registro es modificado por las instrucciones de saltos lejanos.

UADER FCyT

# 8086 -> REGISTROS DE SEGMENTO

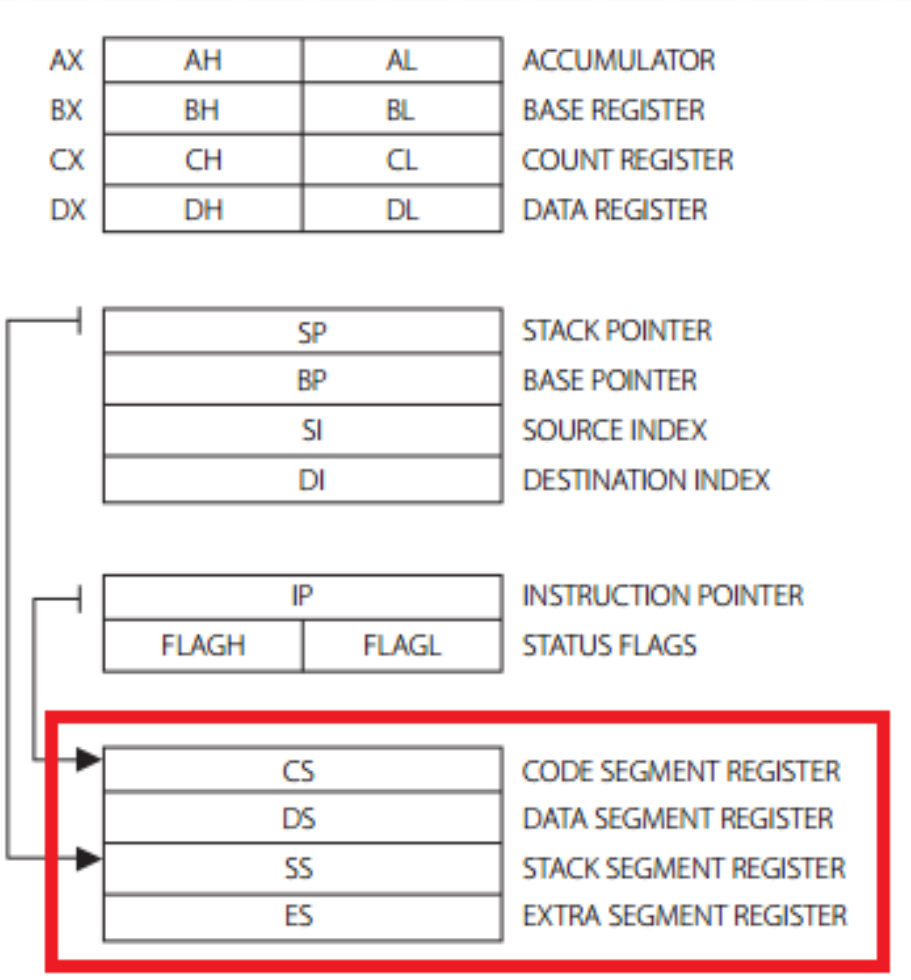



**DS (segmento de datos):** contiene el segmento donde están los datos.

**ES (segmento extra de datos):** es usado para acceder a otro segmento que contiene más datos.

UADER FCyT

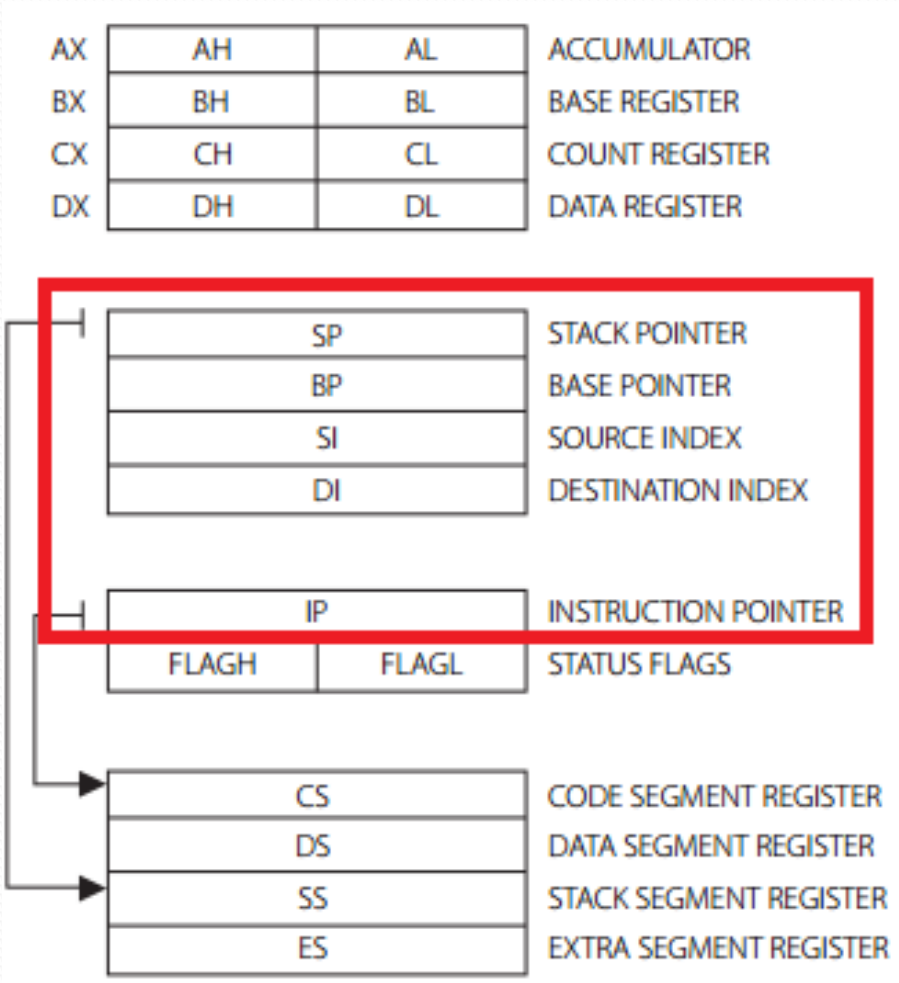
# 8086 -> REGISTROS DE SEGMENTO




**SS (segmento de pila):** contiene el valor del segmento donde está la pila. Se usa conjuntamente con el registro SP para obtener la dirección donde se encuentra el último valor almacenado en la pila por el procesador

UADER FCyT

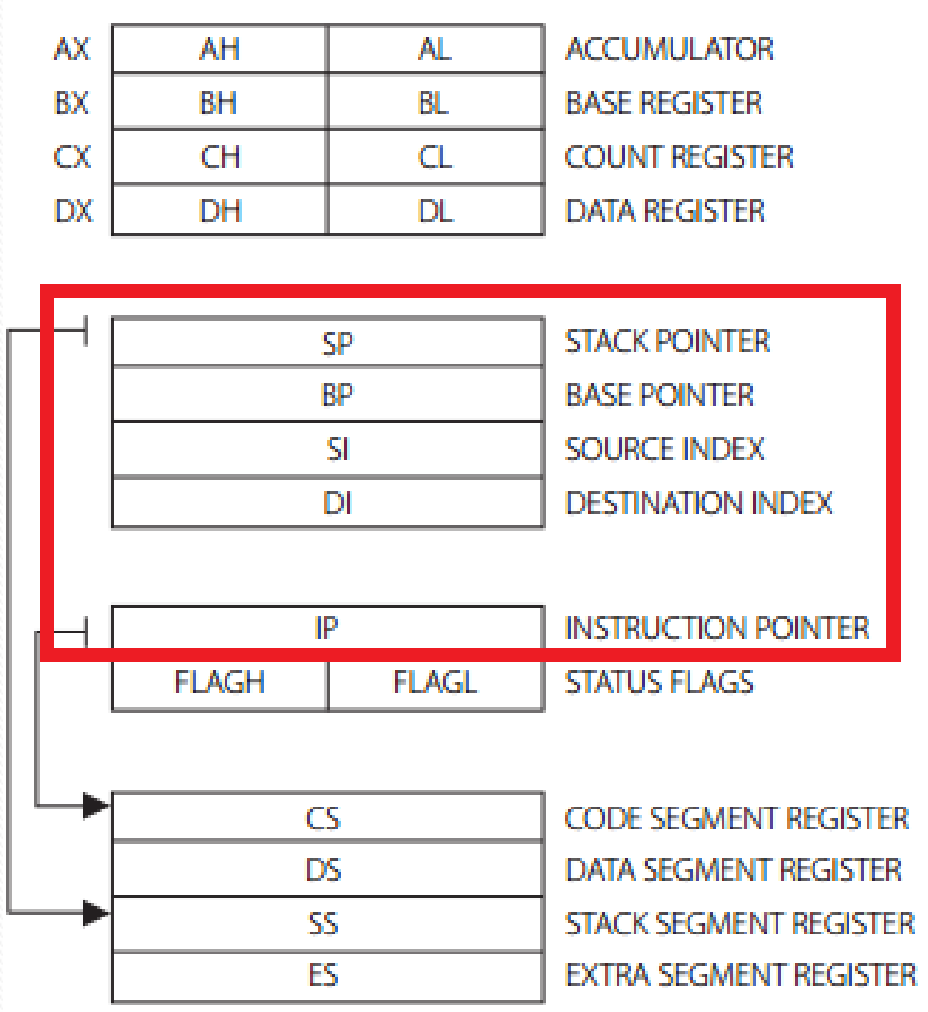
# 8086 -> REGISTROS DE ÍNDICE




Estos registros son usados como índices por algunas instrucciones. También pueden ser usados como operandos (excepto el registro IP).

UADER FCyT

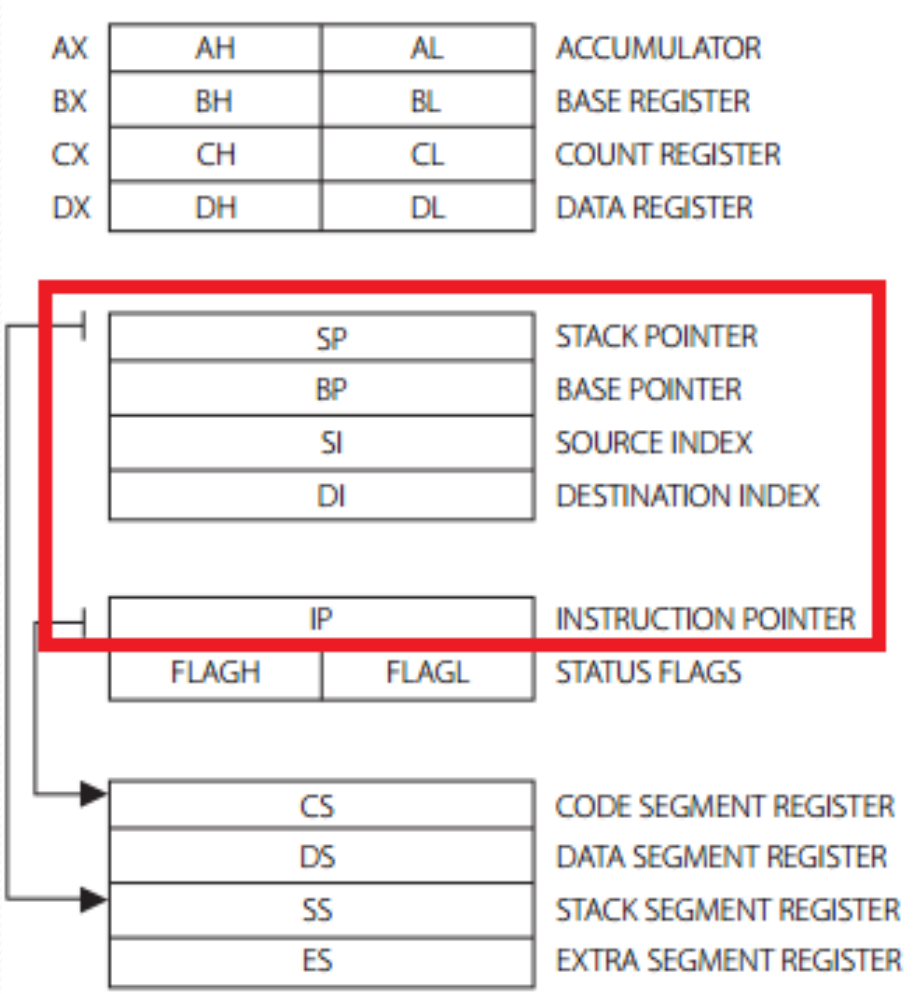
# 8086 -> REGISTROS DE ÍNDICE




**IP (índice de programa):** almacena el desplazamiento dentro del segmento de código. Este registro junto al registro CS apunta a la dirección de la próxima instrucción. No puede ser usado como operando en operaciones aritmético/lógicas.

UADER FCyT

# 8086 -> REGISTROS DE ÍNDICE

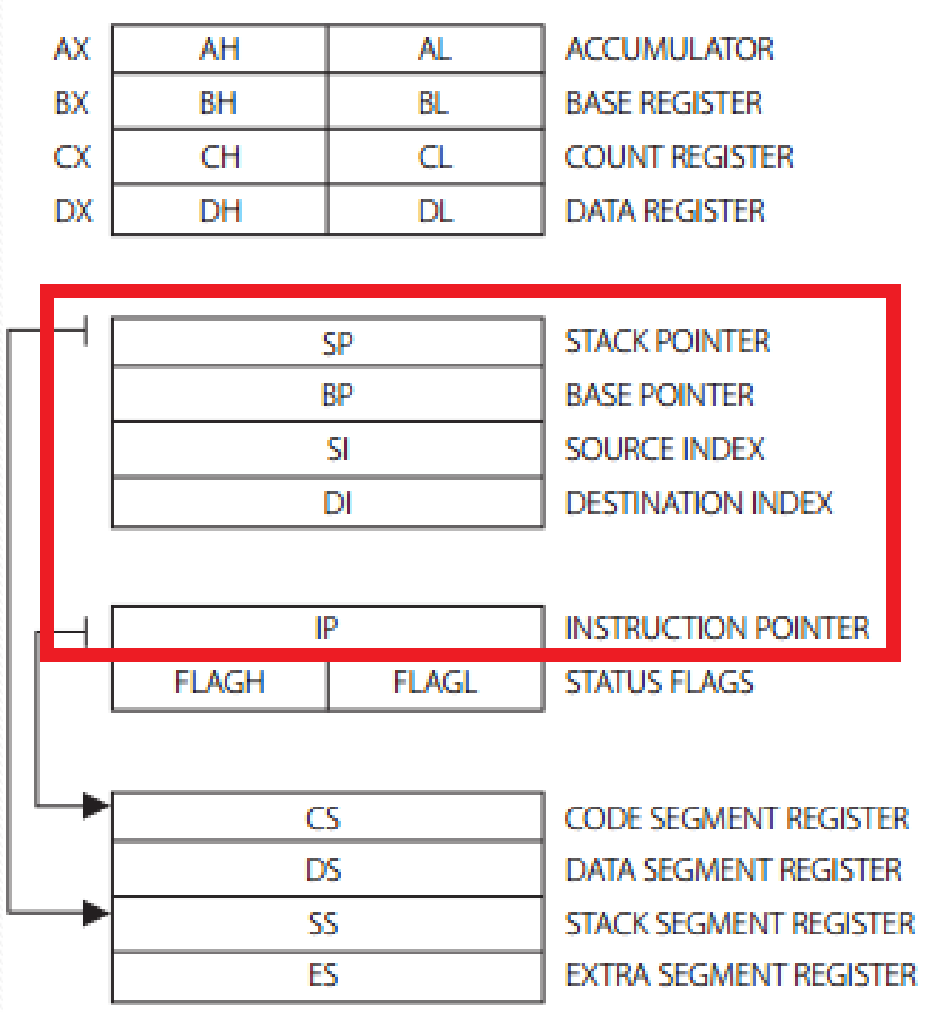



**SI (índice de origen):** almacena el desplazamiento del operando de origen en memoria en algunos tipos de operaciones (operaciones con operandos en memoria).

**DI (índice de destino):** almacena el desplazamiento del operando de destino en memoria en algunos tipos de operaciones (operaciones con operandos en memoria)

UADER FCyT

# 8086 -> REGISTROS DE ÍNDICE

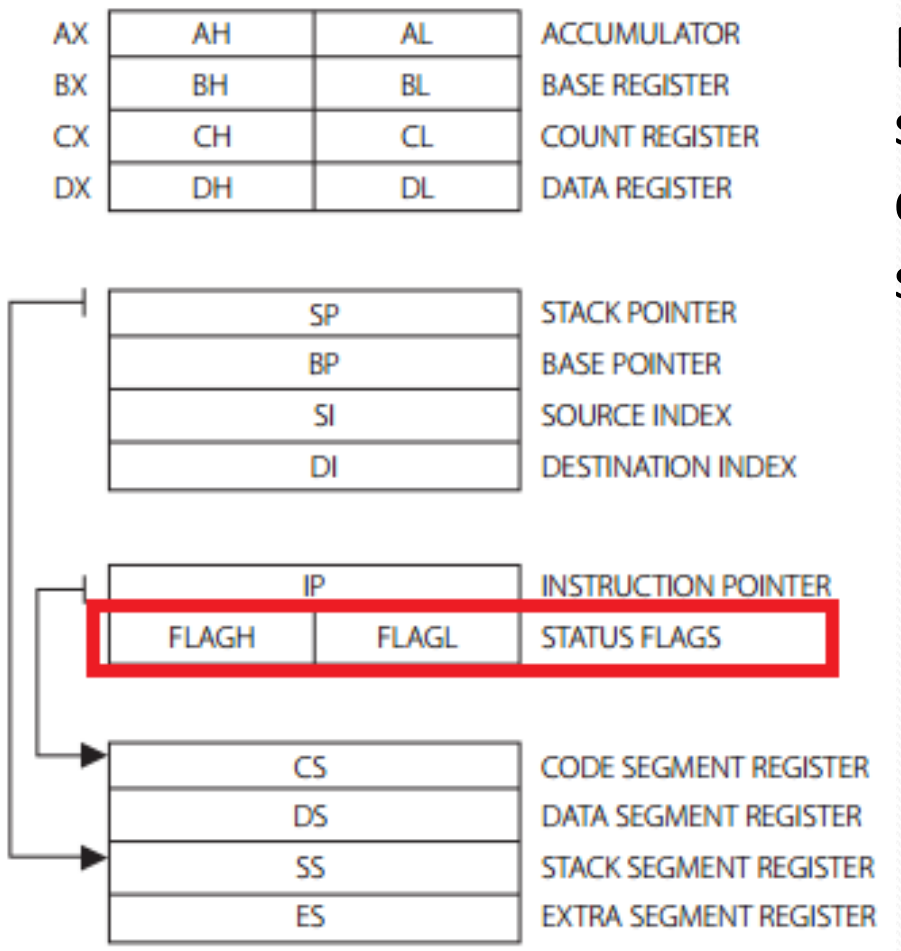



**SP (índice de pila):** almacena el desplazamiento dentro del segmento de pila, y apunta al último elemento introducido en la pila. Se usa conjuntamente con el registro SS.

**BP (índice de base):** se usa para almacenar desplazamiento en los distintos segmentos. Por defecto es el segmento de la pila.

UADER FCyT

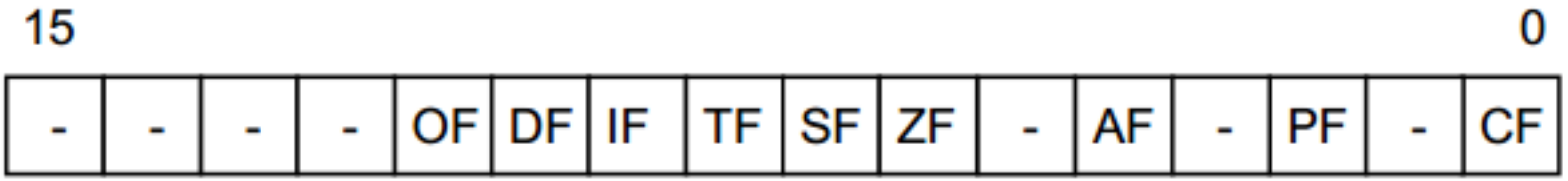
# 8086 -> REGISTRO DE ESTADO




El registro de estado contiene una serie de banderas que indican distintas situaciones en las que se encuentra el procesador.

UADER FCyT

# 8086 -> REGISTRO DE ESTADO



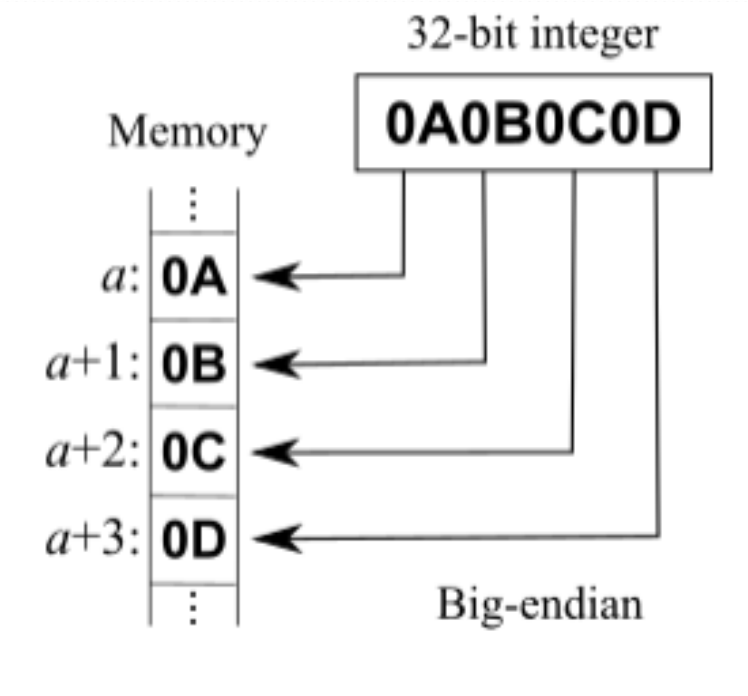

OF: Desbordamiento
DF: Dirección en operaciones con cadenas
IF: Indicador de interrupción
TF: Modo traza
SF: Indicador de signo en operaciones con signo
ZF: Indicador de cero
AF: Acarreo del bit 3 en AL
PF: Bit de paridad
CF: Acarreo

# FORMATO ENDIAN

El formato **endian** es un atributo de los datos que describe el orden de los bytes. Cuando las aplicaciones intercambian datos, deben conocer el convenio de clasificación para los datos de varios bytes. En caso contrario, los datos podrían malinterpretarse.

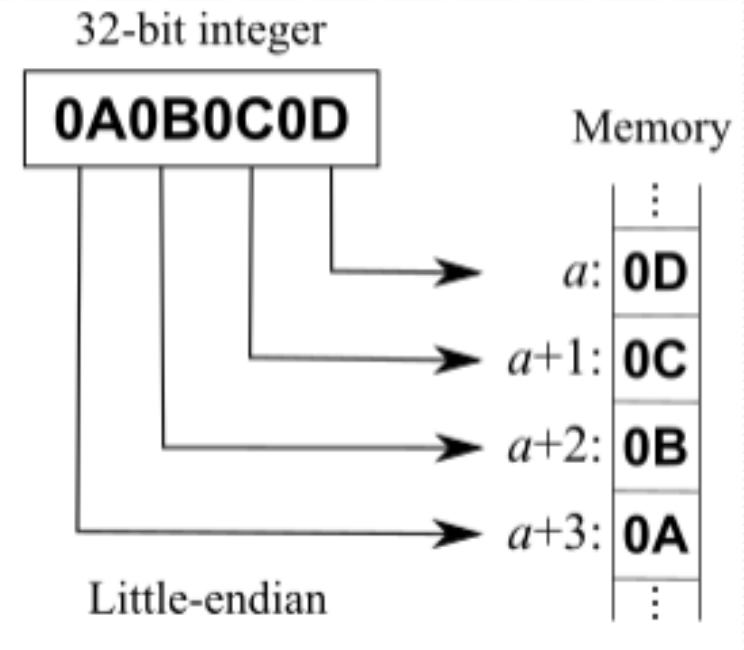
# BIG ENDIAN

Formato en el que el byte más significativo se almacena en primer lugar. Los demás bytes le siguen en orden de significado descendente. Por ejemplo, para una palabra de cuatro bytes, el orden de bytes es 0, 1, 2, 3. Para una palabra de dos bytes, es 0, 1.



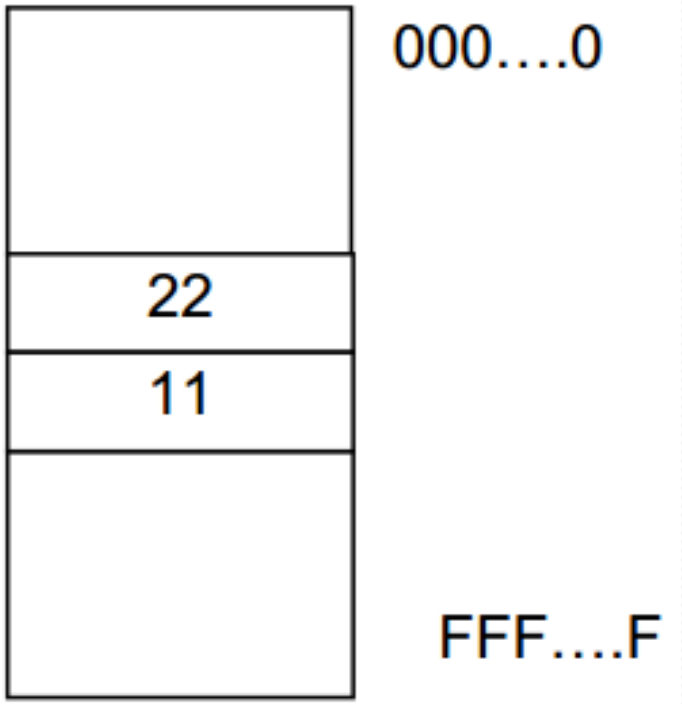
# 8086 -> LITTLE ENDIAN

Formato en el que el byte menos significativo se almacena en primer lugar. Los demás bytes le siguen en orden de significado ascendente. Por ejemplo, para una palabra de cuatro bytes, el orden de bytes es 3, 2, 1, 0. Para una palabra de dos bytes, es 1, 0.



# 8086 -> LITTLE ENDIAN

El 8086/88 usa el formato de almacenamiento denominado “little endian”, esto quiere decir que el byte menos significativa (LSB) del dato es guardada en la parte baja de la memoria. Por ejemplo
el dato 0x1122 será almacenado en memoria:



# 8086 -> SEGMENTACIÓN

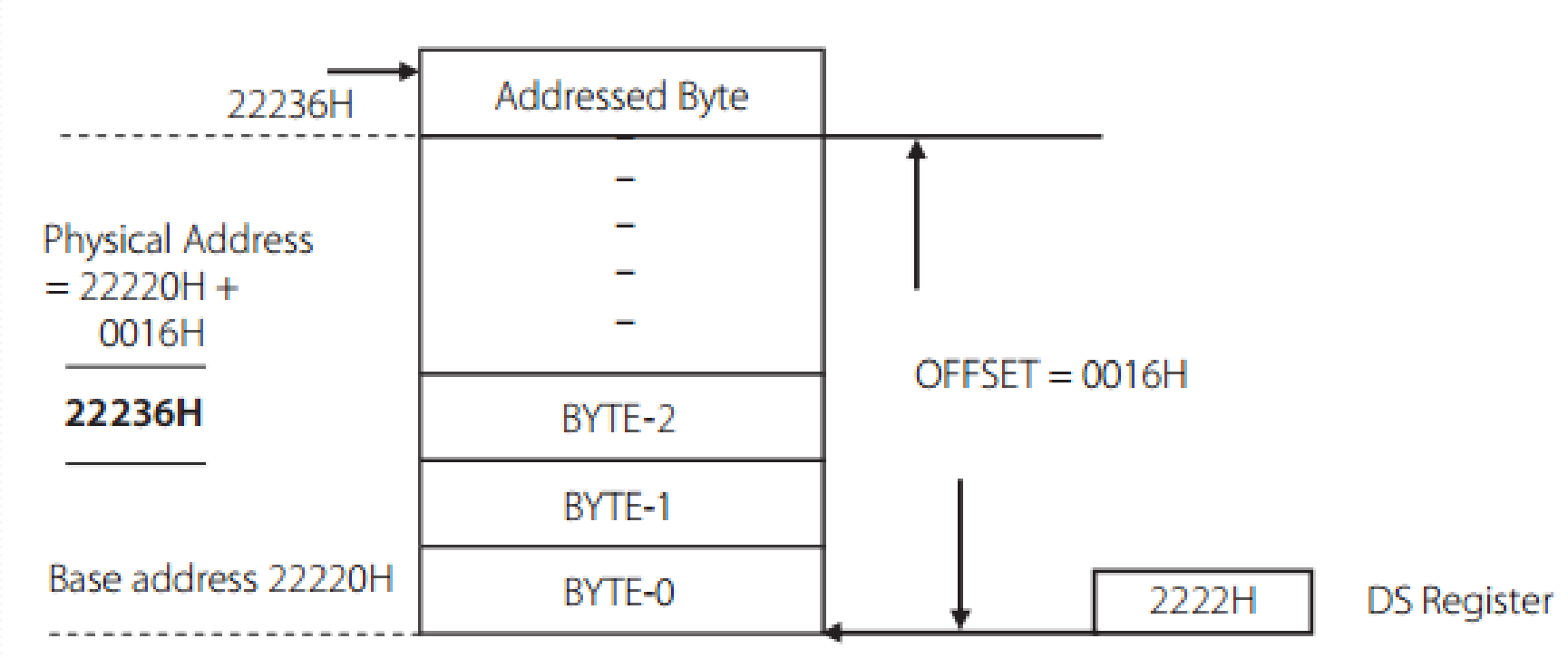
El 8086/88 tiene un ancho de bus de datos de 16 bits y un ancho de bus de direcciones de 20 bits.
Con 20 bits de direcciones se puede acceder a 2^20 = 1 Mega posiciones de memoria. Como cada dirección de memoria contiene un byte, el total de memoria accedido por el procesador es de 1 Mbyte. El bus de datos de 16 bits lo que implica que en cada acceso a memoria se leen dos posiciones.

# 8086 -> SEGMENTACIÓN

El problema que se les planteó a los diseñadores fue que siendo los registros internos del procesador de 16 bits, y el bus de direcciones de 20; faltaban 4 bits para poder aprovechar al máximo las capacidades de direccionamiento del procesador. Para resolver esto, cada dirección de memoria será especificada como un segmento y un desplazamiento dentro de ese segmento.

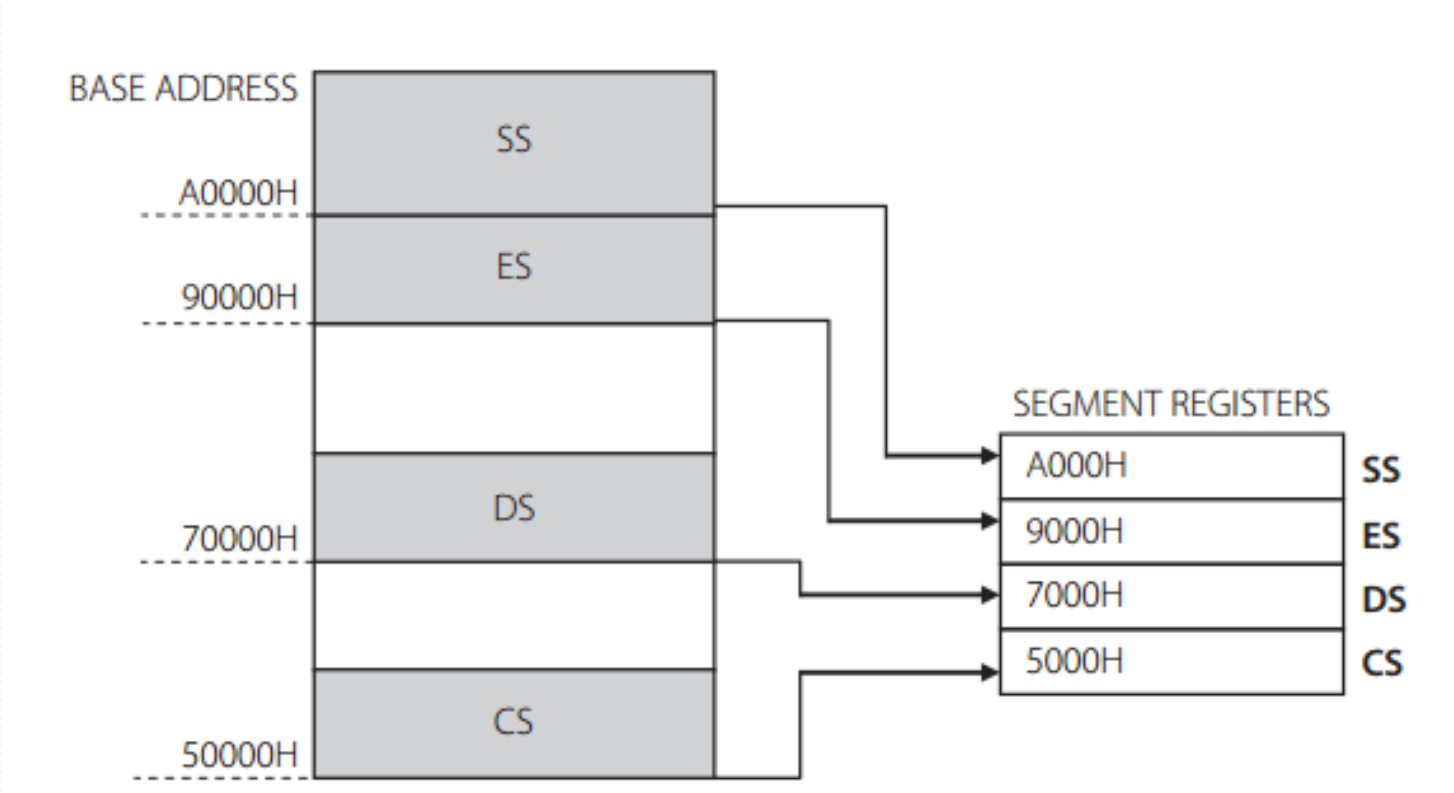
Esta solución divide la memoria en segmentos de 64 K.

# 8086 -> SEGMENTACIÓN



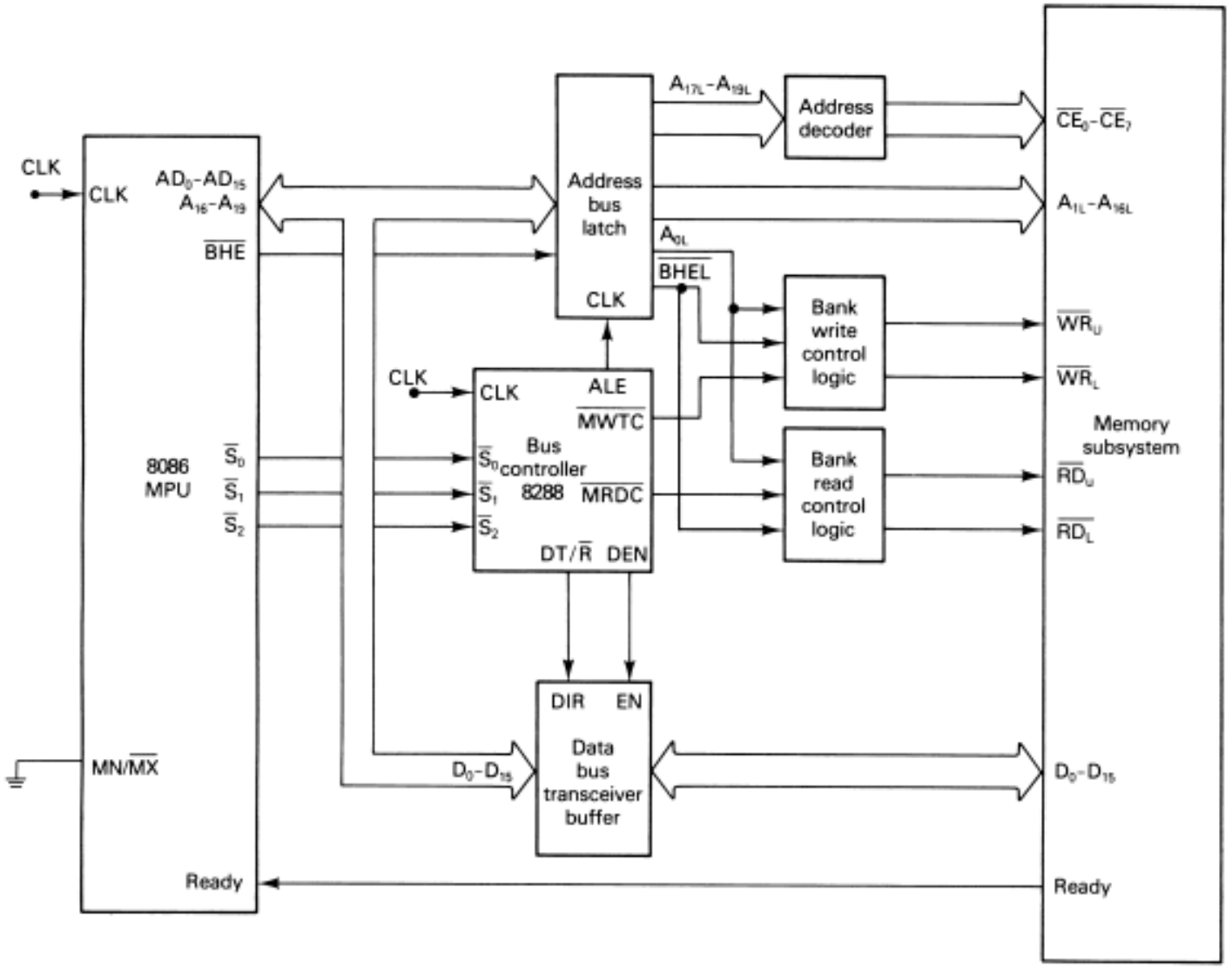

UADER FCyT

# 8086 -> SEGMENTACIÓN



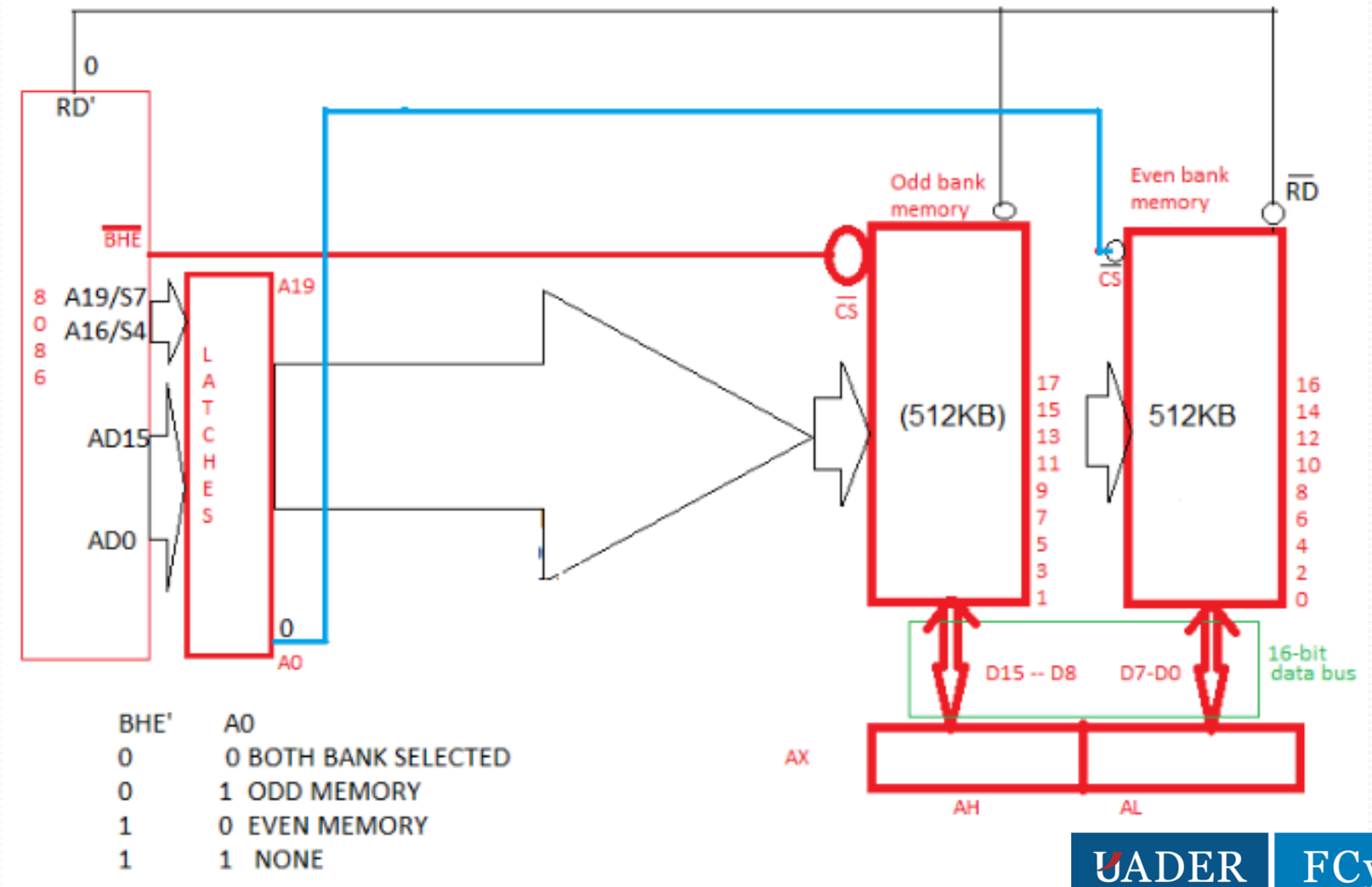

UADER FCyT

# 8086 -> CONEXIÓN DE LAS MEMORIAS




UADER FCyT

# 8086 -> CONEXIÓN DE LAS MEMORIAS




UADER FCyT

# 8086 -> MODOS DE DIRECCIONAMIENTO

El modo de direccionamiento es una técnica utilizada en la programación y el diseño de microprocesadores para especificar cómo se accederá a los operandos (datos o direcciones de memoria) en una instrucción de máquina. Define la forma en que se obtendrán los datos necesarios para realizar una operación y cómo se calcularán las direcciones de memoria en función de los registros y valores presentes en la instrucción.

# 8086 -> MODOS DE DIRECCIONAMIENTO

- Implícito. El dato está implícito en la propia instrucción. Ej. STI (Set Interrupts).
- Inmediato.
- Registro.
- Directo.
- Indirecto:
-- Base. Ej. MOV AX, [BX].
-- Índice. Ej. MOV AX, [SI].
-- Base + Desplazamiento. Ej. MOV AX, [BP + 7]
-- Índice + Desplazamiento. Ej. MOV AX, [DI + 7]
-- Base + Índice. Ej. MOV AX, [BX + SI]
-- Base + Índice + Desplazamiento. Ej. MOV AX, [BX + SI + 9]

# 8086 -> MODOS DE DIRECCIONAMIENTO

El tipo de direccionamiento se determina en función de los operandos de la instrucción.

La instrucción MOV realiza transferencia de datos desde un operando origen a un operando destino.

Su formato es el siguiente:

MOV destino, origen.

# 8086 -> MODOS DE DIRECCIONAMIENTO

**Direccionamiento inmediato.**

Cuando el operando origen es una constante.

Ejemplo:

MOV AX,500; carga en AX el valor 500.

# 8086 -> MODOS DE DIRECCIONAMIENTO

**Direccionamiento de registro.**

Cuando ambos operandos son un registro.

Ejemplo:

MOV AX,BX; transfiere el contenido de BX en AX.

# 8086 -> MODOS DE DIRECCIONAMIENTO

**Direccionamiento directo.**

Cuando el operando origen es una dirección de memoria. Ésta puede ser especificada con su valor entre [ ], o bien mediante una variable definida previamente.

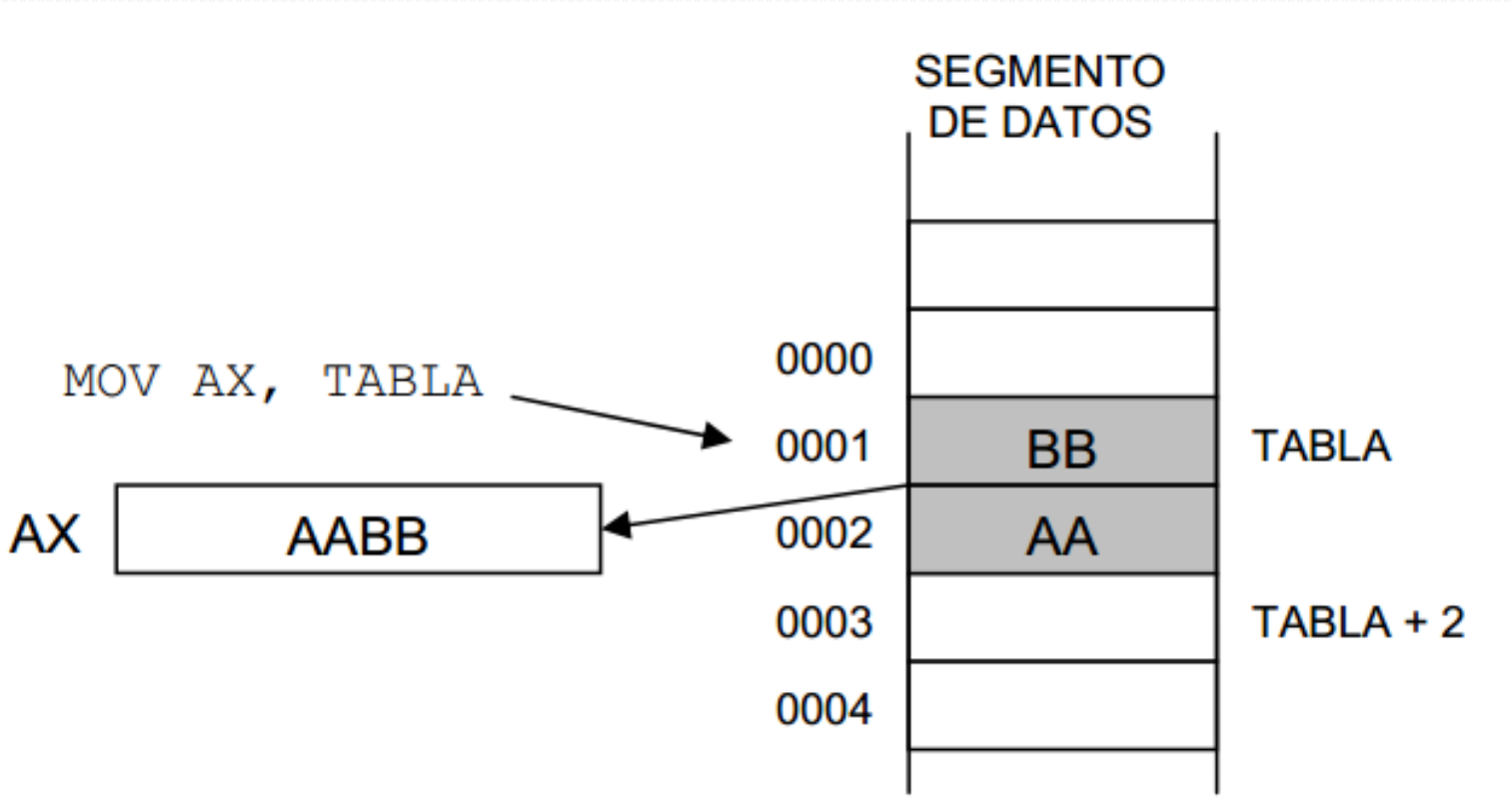
Ejemplo:

MOV BX,[1000]; almacena en BX el contenido de la dirección de memoria DS:1000.

MOV AX,TABLA; almacena en AX el contenido de la dirección de memoria DS:TABLA.

# 8086 -> MODOS DE DIRECCIONAMIENTO

**Direccionamiento directo.**



# 8086 -> MODOS DE DIRECCIONAMIENTO

**Direccionamiento indirecto: base/indice.**

Cuando el operando origen/destino esta en memoria en una posición contenida en un registro (BX, BP, SI o DI).

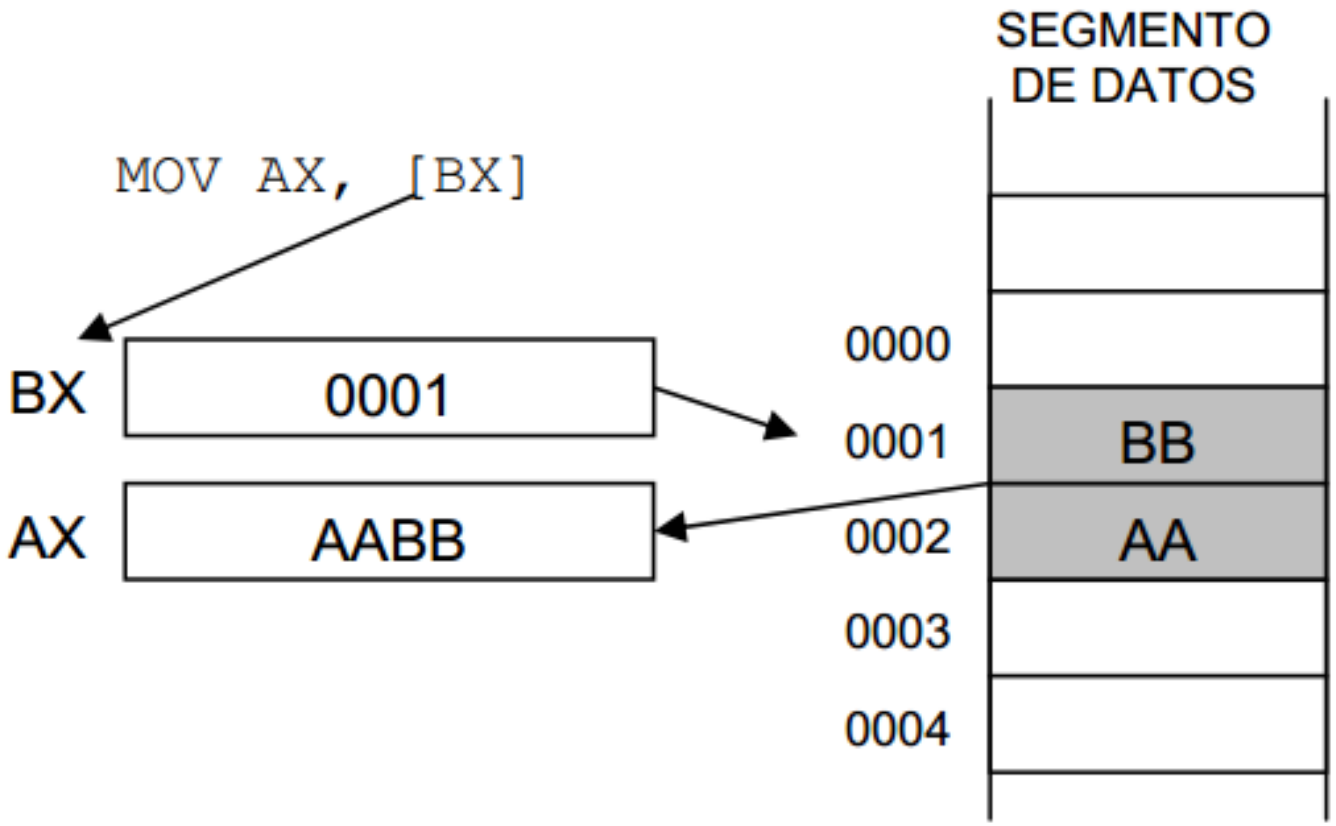
Ejemplo:

MOV AX,[BX] ; almacena en AX el contenido de la dirección de memoria DS:[BX] (apuntada por BX).

MOV [BP],CX ; almacena en al dirección apuntada por BP en contenido de CX.

# 8086 -> MODOS DE DIRECCIONAMIENTO

**Direccionamiento indirecto: base/indice.**



# 8086 -> MODOS DE DIRECCIONAMIENTO

**Direccionamiento indirecto: B+D.**

Cuando el operando esta en memoria en una posición apuntada por el registro BX o BP al que se le añade un determinado desplazamiento.

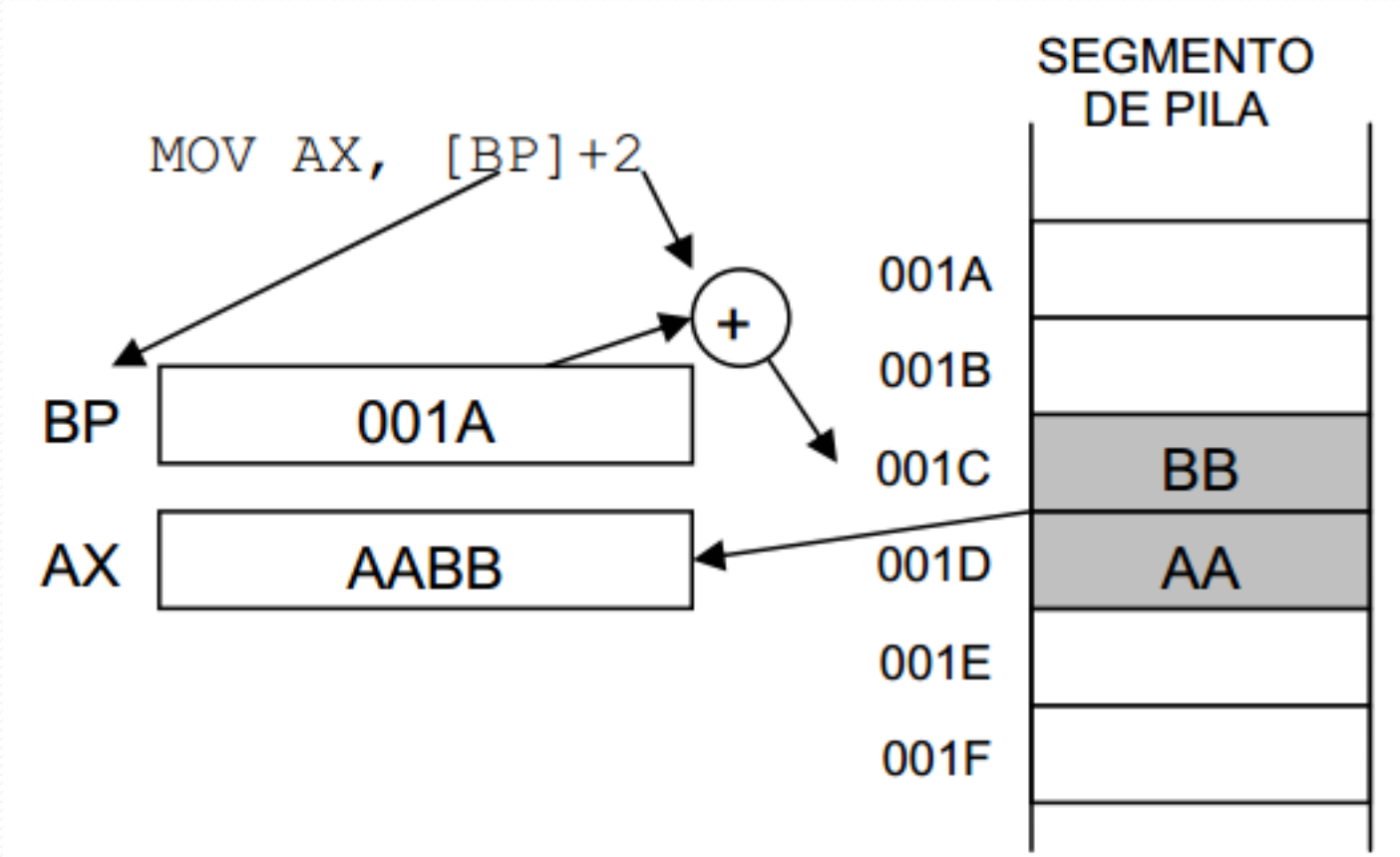
Ejemplo:

MOV AX, [BP] + 2; almacena en AX el contenido de la posición de memoria que resulte de sumar 2 al contenido de BP

Equivalente a MOV AX, [BP + 2].

# 8086 -> MODOS DE DIRECCIONAMIENTO

**Direccionamiento indirecto: B+D.**



# 8086 -> MODOS DE DIRECCIONAMIENTO

**Direccionamiento indirecto: I+D.**

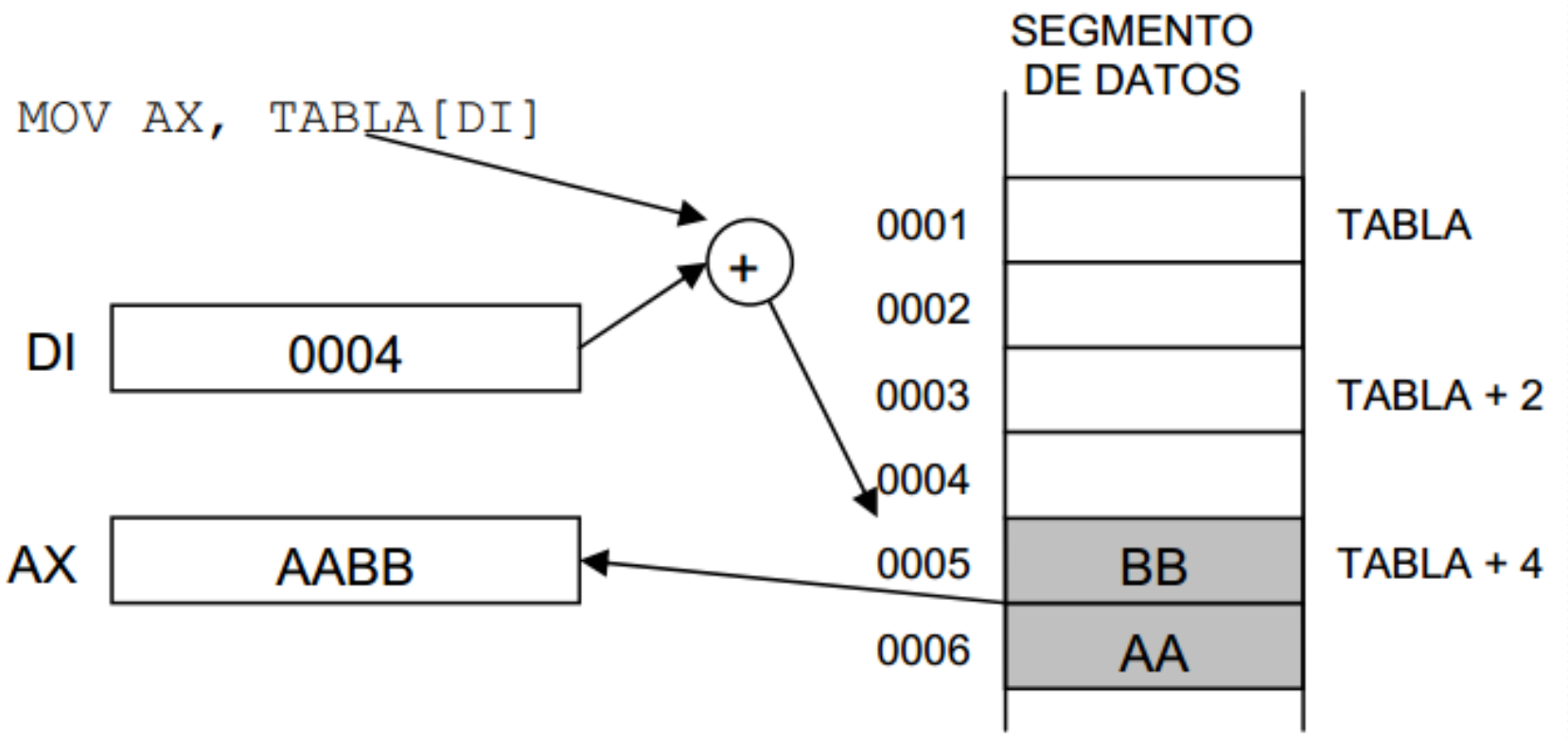
Cuando la dirección del operando es obtenida como la suma de un desplazamiento más un índice (DI, SI).

Ejemplo:
MOV AX, TABLA[DI] ; almacena en AX el contenido de la posición de memoria apuntada por el resultado de sumarle a TABLA el contenido de DI.

MOV AH, [SI + 1733];

# 8086 -> MODOS DE DIRECCIONAMIENTO

**Direccionamiento indirecto: I+D.**




UADER FCyT

# 8086 -> MODOS DE DIRECCIONAMIENTO

**Direccionamiento indirecto con base e índice/B+I+D.**
Cuando la dirección del operando se obtiene de la suma de un registro base (BP o BX), de un índice (DI, SI) y opcionalmente un desplazamiento.

Ejemplo:
MOV AX, TABLA[BX][DI] ; almacena en AX el contenido de la posición de memoria apuntada por la suma de TABLA, el contenido de BX y el contenido de DI.
MOV AH, [BX + SI + 2731];

# PIPELINING.

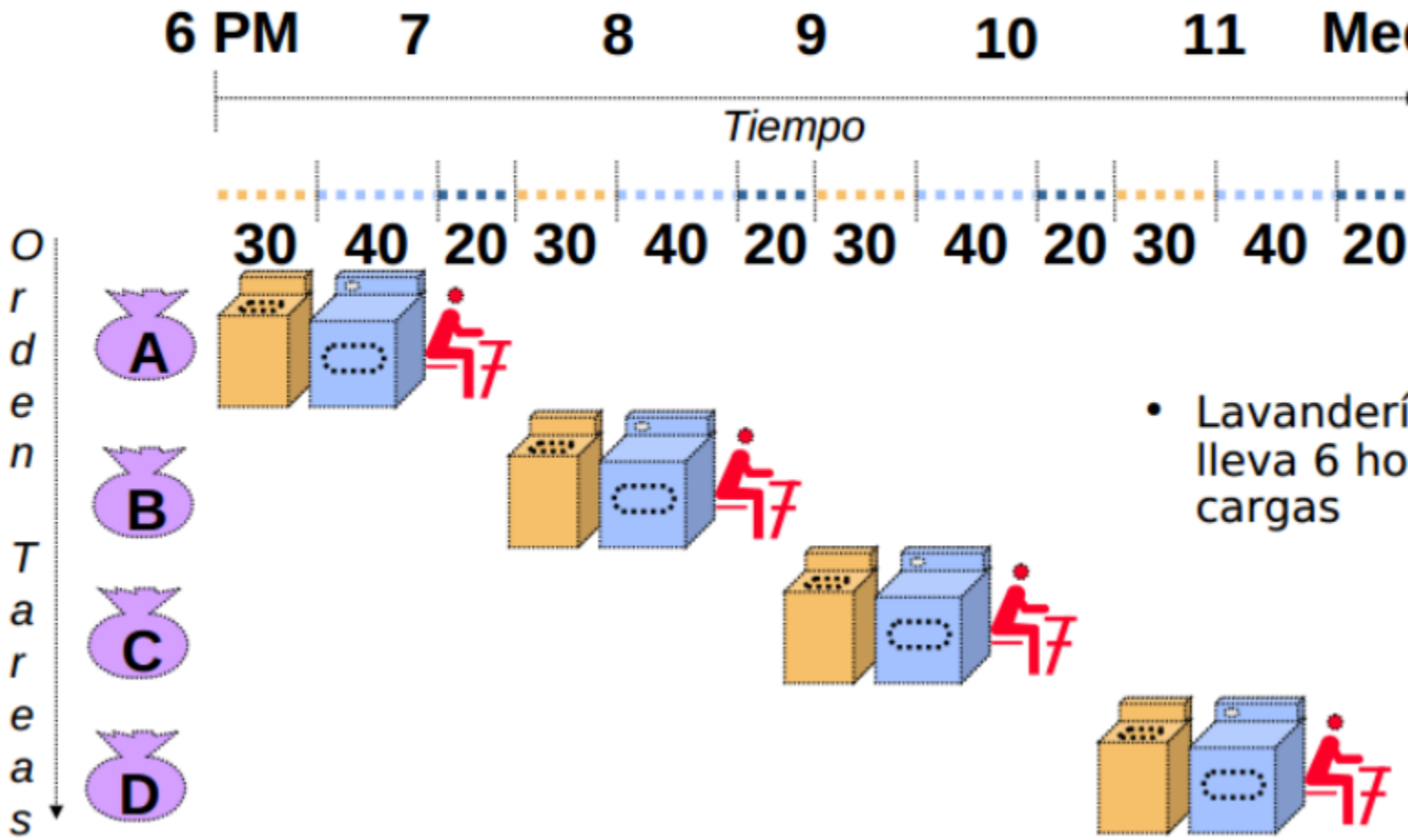
- Concepto: Para una tarea ‘larga’, que se debe repetir, y que se puede dividir en etapas: paralelizar varias ejecuciones de la tarea, solapando las etapas que puedan realizarse al mismo tiempo.
- Responde a la idea de ‘línea de producción’ utilizada en las fábricas.

# PIPELINING.

Supongamos el proceso de lavado de ropa en una lavandería con los siguientes pasos:

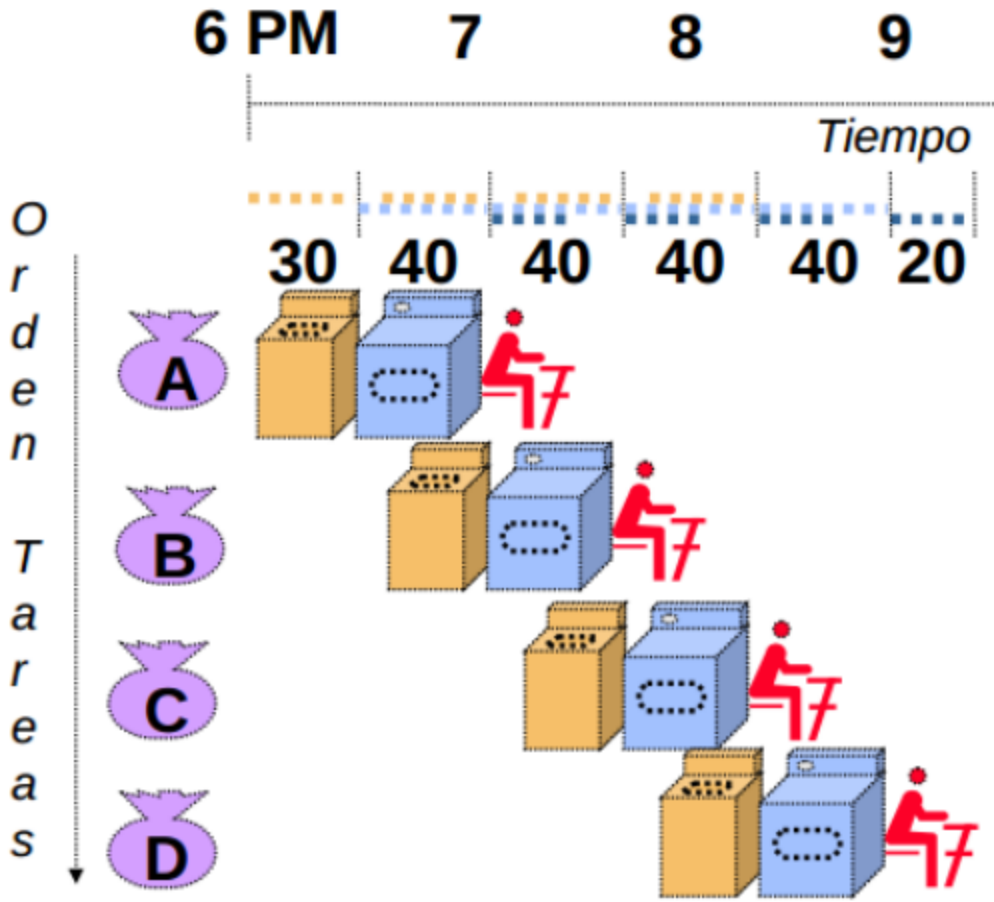
- Lavado: 30 minutos.
- Secado: 40 minutos.
- Doblado de ropa: 20 minutos.

# PIPELINING.



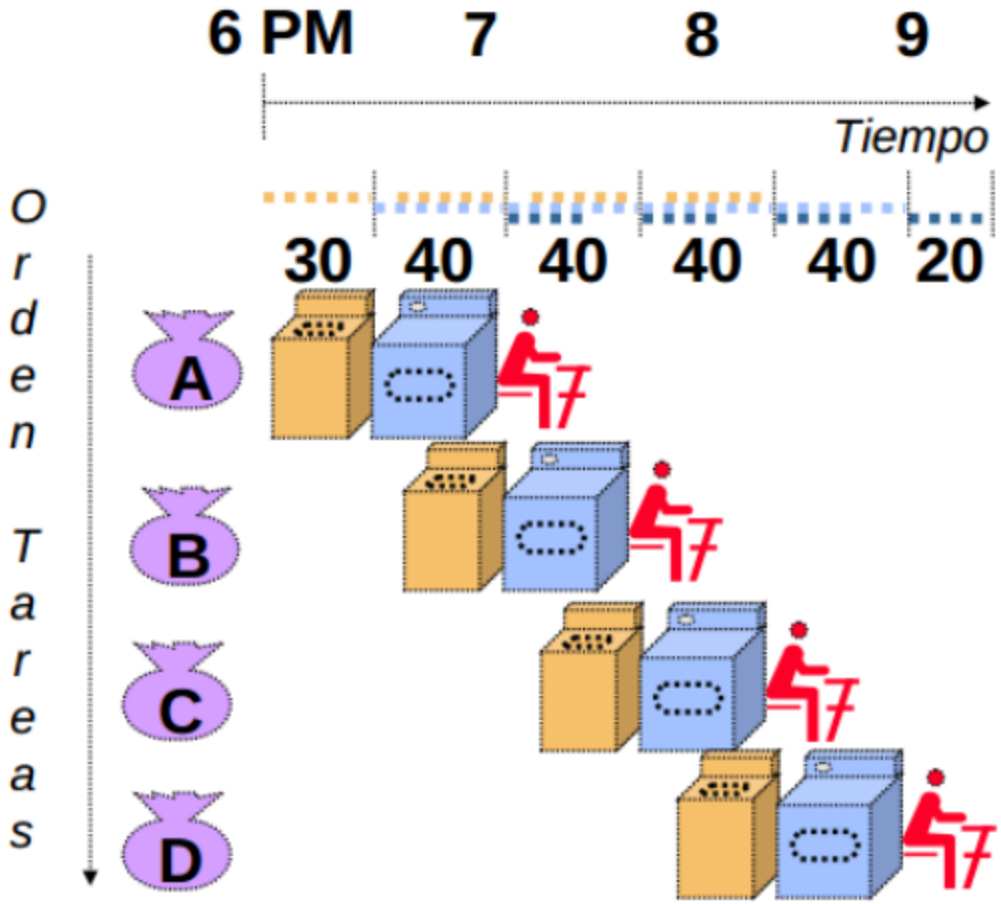

- Lavandería secuencial lleva 6 horas para 4 cargas

# PIPELINING.



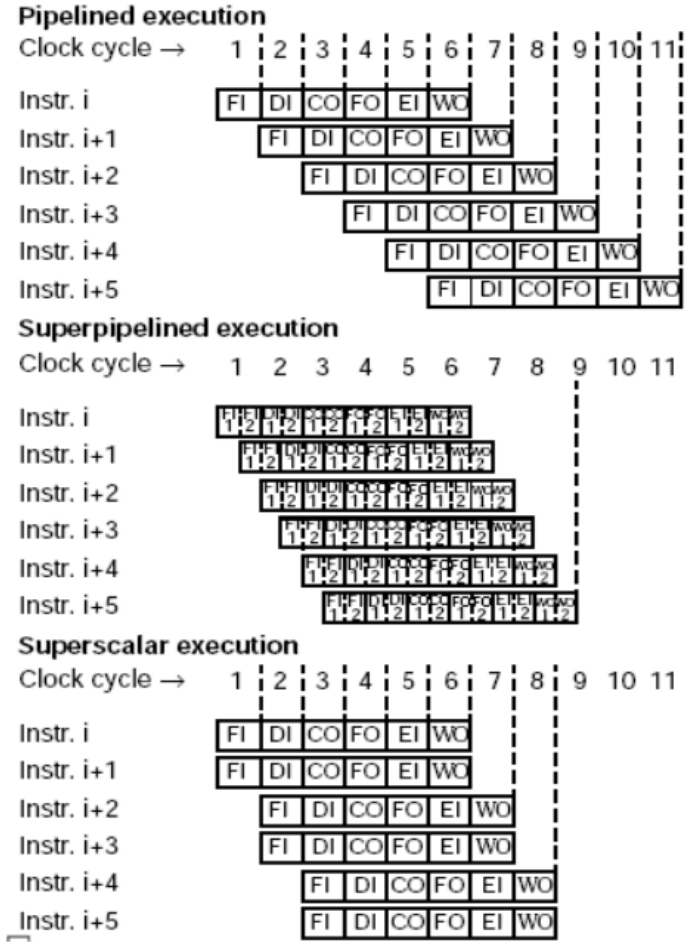

- Lavandería en Pipeline lleva 3.5 horas para 4 cargas!

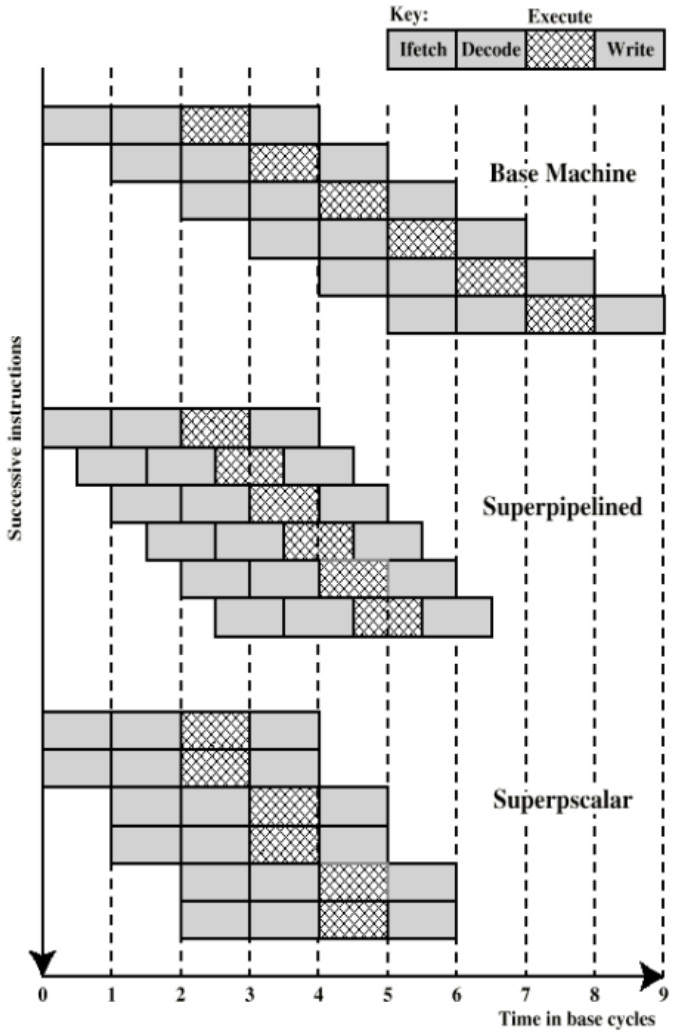
# PIPELINING.




- Pipelining no mejora la latencia de cada tarea, sino el throughput de toda la carga de trabajo
- Velocidad del Pipeline limitada por el paso más lento
- Aceleración potencial = Cantidad de pasos del pipeline
- Tiempo para “llenar” y “vaciar” el pipeline reduce la aceleración

# PIPELINING.



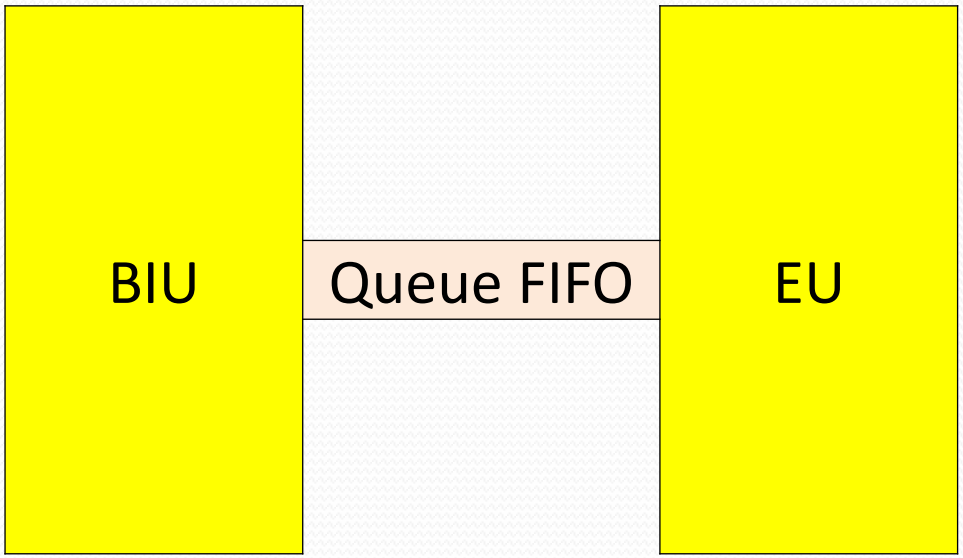




Productividad: 1 o 2 inst/ciclo

UADER FCyT

# PIPELINING.

El 8086 está compuesto por 2 máquinas conocidas como unidad de interface bus (UIB) y unidad de ejecución (UE) que trabajan en modo PIPELINE, conectadas por una cola.



# 8086 -> PIPELINING.

- El proceso de obtener la siguiente instrucción cuando se está ejecutando la instrucción actual se denomina PIPELINING.

- El PIPELINING se ha vuelto posible debido al uso de la cola.

- La BIU completa la cola hasta que se llena completamente.

- La BIU vuelve a llenar la cola cuando al menos dos ubicaciones del final están vacantes.

# 8086 -> PIPELINING.

Ventajas del PIPELINING:

- La unidad de ejecución siempre lee el siguiente byte de instrucción de la cola en la BIU. Esto es más rápido que enviar una dirección a la memoria y esperar a que llegue el siguiente byte de instrucción.
- En resumen, el PIPELINING elimina el tiempo de espera de la UE y acelera el procesamiento.
- La BIU del 8086 no iniciará una búsqueda a menos que haya dos bytes vacíos en su cola.
- La BIU normalmente obtiene dos bytes de instrucción por búsqueda.

# PIPELINE HAZARDS.

En el diagrama anterior se asume que todas las etapas pueden progresar de forma ideal, pero como se mostrará a continuación, la aplicación de la técnica de Pipeline introduce complejidades que a menudo impiden la ejecución ideal.

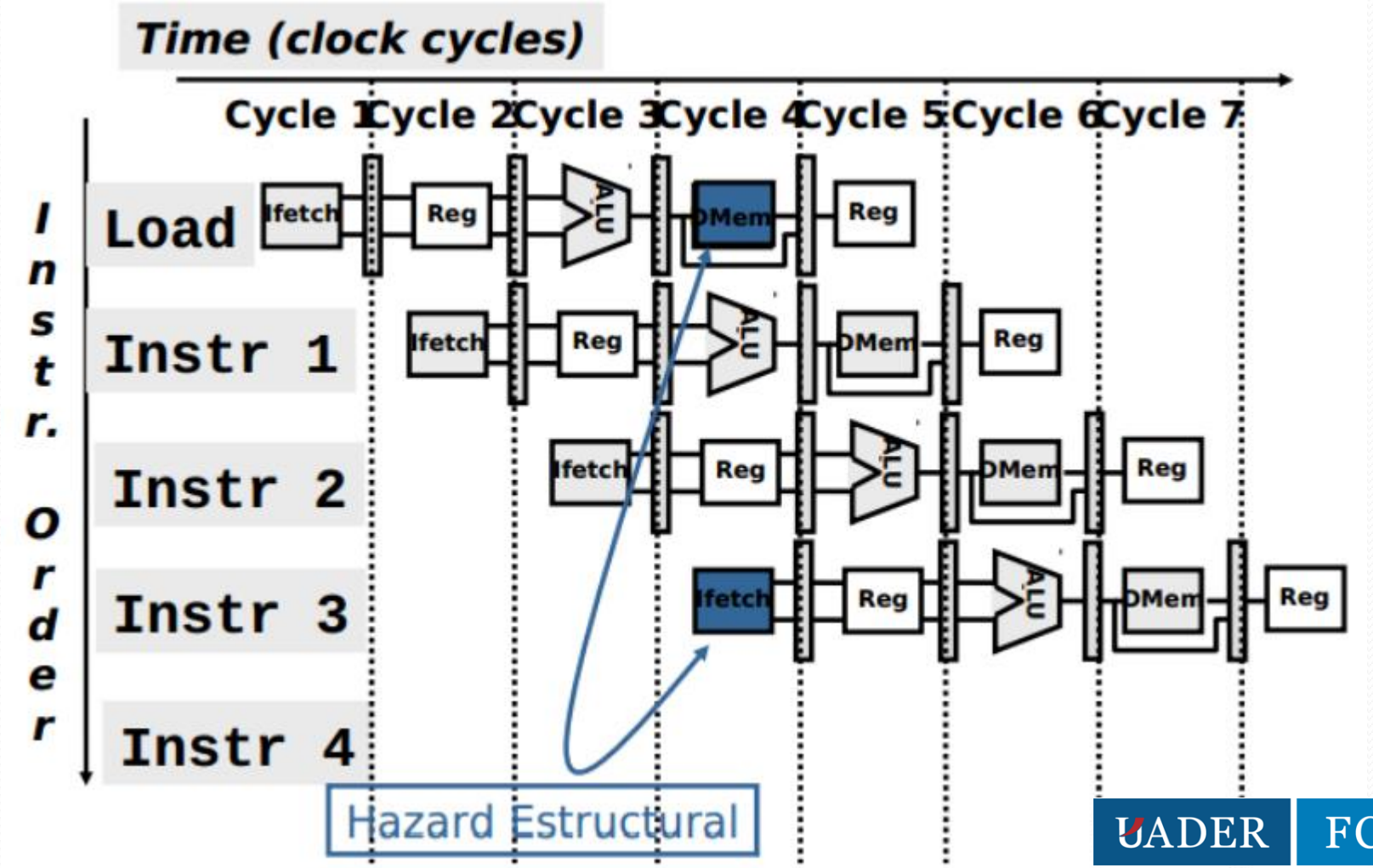
Cuando se detiene el Pipeline se dice que se está en presencia de un hazard.

# PIPELINE HAZARDS.

Tipos de hazards:

- **Hazards estructurales:** se dan cuando hay un conflicto de hardware para alguna combinación de instrucciones.
- **Hazards de datos:** se dan cuando por alguna dependencia de datos en las instrucciones y el uso de pipeline, se altera el flujo de datos del programa.
- **Hazards de control:** son causados por instrucciones de saltos u otras modificaciones del PC.

# HAZARDS ESTRUCTURALES.




UADER FCyT

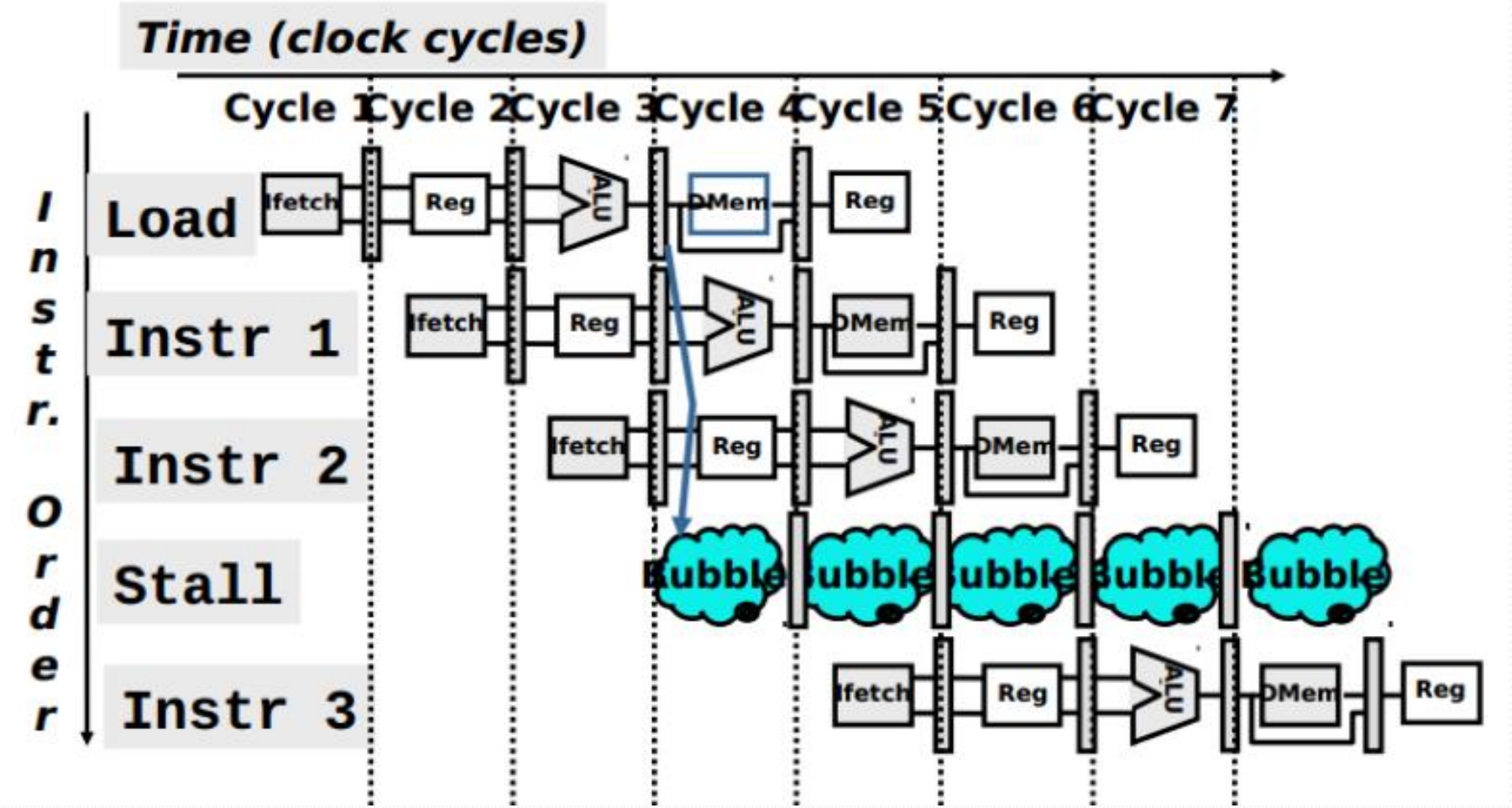
# HAZARDS ESTRUCTURALES.

Un hazard estructural se produce cuando dos instrucciones quieren acceder al mismo recurso de hardware al mismo tiempo.

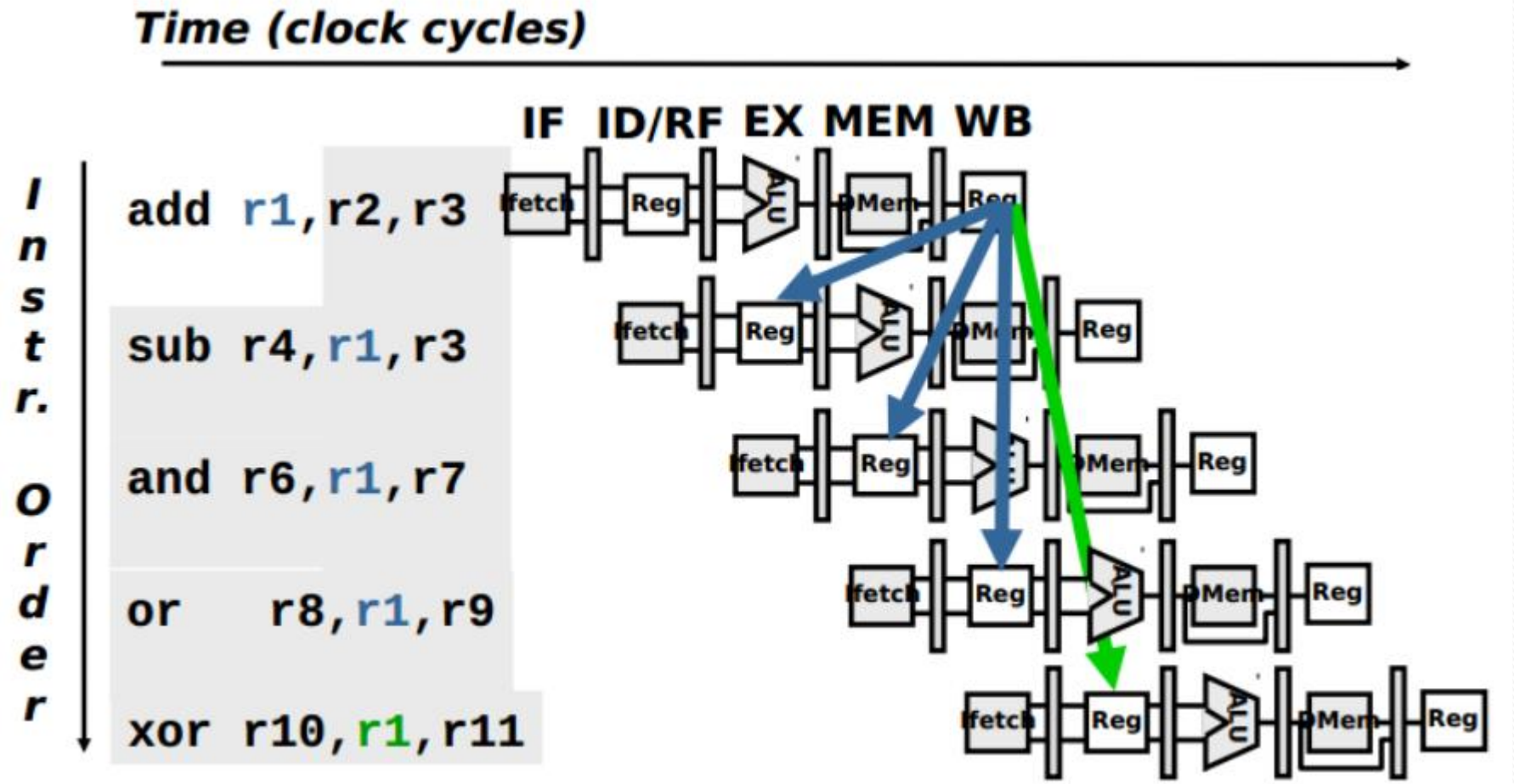
Dos opciones para solucionar:

- Esperar. Se debe generar una señal de stall.
- Agregar más hardware.

# HAZARDS ESTRUCTURALES.



# HAZARD DE DATOS.




UADER FCyT

# HAZARDS DE DATOS.

**Read after Write (RaW):** Un operando es modificado para ser leído posteriormente. Si la primera instrucción no ha terminado de escribir el operando, la segunda estará utilizando datos incorrectos. Dependencia verdadera.

**Write after Read (WaR):** Leer un operando y escribir en él en poco tiempo. Si la escritura finaliza antes que la lectura, la instrucción de lectura utilizará el nuevo valor y no el antiguo. Anti-dependencia.

**Write after Write (WaW):** Dos instrucciones que escriben en un mismo operando. La primera en ser emitida puede que finalice en segundo lugar, de modo que el operando final no tenga el valor adecuado. Dependencia de salida.

# HAZARDS DE SALTO O DE CONTROL.

**Los riesgos de salto** o de control ocurren cuando el procesador se ve obligado a saltar a una instrucción que no tiene por qué ser necesariamente la inmediatamente siguiente en el código. En ese caso, el procesador no puede saber por adelantado si debería ejecutar la siguiente instrucción u otra situada más lejos en el código.

Esto puede resultar en acciones no deseadas por parte de la CPU.