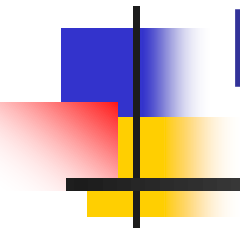


Sistemas de Procesamiento de Datos

Assembler (parte 2)
8086

Profesor: Fabio Bruschetti
Aydte: Pedro Iriso
Ver 2019

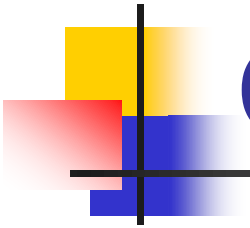
Assembler – Real Mode

- Modelos de Memoria
  - En Real Mode se puede elegir los siguientes modelos

Modelo	Codigo	Datos
Tiny (DS=CS=SS=ES)	< 64Kb	< 64Kb
Small (CS, DS=SS)	< 64Kb	< 64Kb
Medium	> 64Kb	< 64Kb
Compact	< 64Kb	> 64Kb
Large / Huge	> 64Kb	> 64Kb

Modelo	Punteros
Tiny, Small, Medium y Compact	Near
Large / Huge	Far / Huge
Referencia	**Punteros**
En el mismo segmento	Near
En diferente segmento	Far

  - En Protected Mode la memoria es simple contigua (Tiny model o Small model)



Cargando un Programa

- Nuestro progama debe ser cargado en memoria para poder ser ejecutado
  - Hecho por el Loader (Herramienta que es parte del sistema operativo)
  - El Loader decide que segmentos actuales usará
  - El Loader inicializa SS:SP (para uso del stack) y CS:IP (para apuntar a la primera instrucción a ejecutar)
  - El Loader inicializa DS pero no apunta al segmento de datos
  - En lugar de esto, el Loader sabe que segmento usó como segmento de datos
  - Cuando el programa está cargado, el Loader reemplaza todo ocurrencia de "@data" con el número de segmento de datos
- Que significa todo esto para nuestro programa?

Cargando un Programa

- Recordar: El procesador usa el contenido de DS como el valor de 16bits del segmento cada vez que accede a las variables de memoria (incluyendo el fetch de operandos durante el fetch de instrucciones)
  - El programador solo necesita suministrar el offset de 16 bits en las instrucciones
- DS debe ser inicializado antes de cualquier acceso al segmento de datos
  - Antes de cualquier referencia a etiquetas de memoria
- DS es inicializado dinámicamente (en runtime)
  - Debe ser la primer cosa que el programa realice
  - Específicamente, ninguna variable definida en el segmento de datos puede ser referenciada hasta que DS esté inicializado.
- Como se inicializa DS:
  - Forma Errónea: MOV DS, @data MOV(RegSeg, inmed) (No permitido)
  - Forma Correcta: MOV AX, @data Modo inmed
    MOV DS, AX Modo Registro

Listando un Programa

Address

Machine Encoding

Assembly Programming

```
                                    ; This program displays "Hello World"
                                .model small
                                .stack 100h
0000                            .data
0000 48 65 6C 6C 6F 2C          message db "Hello, world!", 0dh, 0ah, '$'
     20 77 6F 72 6C 64
     21 0D 0A 24
0000                            .code
0000                            main PROC
0000  B8 ---- R                      MOV     AX, @data
0003  8E D8                          MOV     DS, AX

0005  B4 09                          MOV     AH, 9
0007  BA 0000 R                      MOV     DX, OFFSET message
000A  CD 21                          INT     21h

000C  B8 4C00                        MOV     AX, 4C00h
000F  CD 21                          INT     21h
0011                            main ENDP
                                END main
```

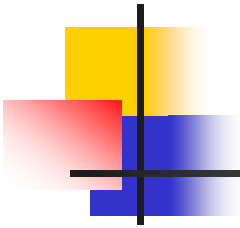
SubPrograma Ejemplo

- ***main.asm*:**

```
.MODEL SMALL
.STACK 200
.DATA
    s1 DB Good ', 0
    s2 DB ‘morning!', '$', 0
    FinalS DB 50 DUP (?)
    PUBLIC FinalS   ; reemplazo GLOBAL FinalS:BYTE
.CODE
    EXTRN Concatenar:PROC
Start:
    mov ax, @data
    mov ds, ax                  ;Carga el Data Segment
    mov ax, OFFSET s1
    mov bx, OFFSET s2
    call Concatenar             ; FinalS:=s1+s2
    mov ah, 9
    mov dx, OFFSET FinalS
    int 21h                     ;Imprime FinalS
    mov ah, 4ch
    int 21h                     ; Fin del programa
END  Start
```

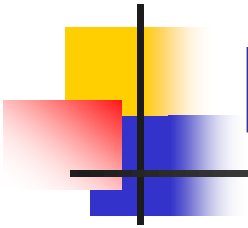
- ***sub.asm*:**

```
.MODEL SMALL
.DATA
EXTRN FinalS:BYTE    ; reemplazo GLOBAL FinalS:BYTE
.CODE
PUBLIC Concatenar
Concatenar PROC
    cld
    mov di, SEG FinalS
    mov es, di
    mov di, OFFSET FinalS   ;es:di <- dirección de FinalS
    mov si, ax              ;ds:si <- Direccion de S1
    s1Loop:
        lodsb               ;al <- caracter actual
        and al, al          ;Termina en 0?
        jz cont
        stosb               ;Si no, se pone en FinalS
        jmp  s1Loop
    cont:
    mov si, bx              ;ds:si <- dirección de S2
    s2Loop:
        lodsb
        stosb               ;Termina en 0?
        and al, al
        jnz s2Loop
    ret                     ; regresa del procedimiento
Concatenar ENDP
END
```



¿Cuál Microprocesador ?

- Cualquier microprocesador está caracterizado por:
  - El Conjunto de Registros
    - De propósito general, de direccionamiento, de control y estado
  - El Conjunto de Instrucciones
    - Incluye los modos de direccionamiento
  - El mecanismo de Interrupciones (más adelante)
- Se estudiará al Intel 8086 el primero de los 80x86
  - Todos los registros de programa son de 16 bits
  - Bus de datos de 16 bits y bus de direcciones de 20 bits
  - Puertos mapeados de I/O de 8 bits y 16 bits (mas adelante)
  - Todos los desendiente de la familia son compatibles con este
    - Mismo conjunto básico de registros pero mas anchos
    - Mismo conjunto básico de instrucciones pero muchas nuevas
    - Mismo mecanismo de interrupciones



La Familia 80x86

- Intel 8086
  - Registros de 16 bits, bus de datos de 16 bits, bus de direcciones de 20 bits.
- Intel 80286
  - Igual que el 8086, pero el bus de direcciones de 24 bits y tiene “modo protegido” (multitarea)
- IA-32
  - Registros de 32 bits, bus de datos de 32 bits, bus de direcciones de 32 bits.
- P6
  - Extiende la arquitectura IA-32 para lograr mas performance
- Core2
  - extienden la arquitectura P6 (64 bits)

8086 – Modos de Operación

- Modo Real-Access (DOS)
  - El Microprocesador actúa como el 8086
    - Espacio de Direcciones de 1 MB, solo instrucciones 8086, un solo programa en ejecución
    - Acceso directo irrestricto a toda la memoria y hardware E/S
- Modo Protegido
  - Todas las instrucciones y características disponibles
  - Varios programas en ejecución sobre areas de memoria separadas (llamadas segmentos). La CPU previene accesos fuera de segmento.
    - No hay Direcciones de Memoria "Real", sino dentro de su area
- Modo Virtual 8086
  - Mientras se está en modo protegido, se permite a un programa ejecutar en modo Real-Access
  - Ejecutar un Program DOS bajo Windows
    - Pero windows previene acceso a algunas direcciones / hardware
- Modo Manejo de Sistemas
  - Permite tener las funciones de seguridad a un Sistema Operativo

8086 - Conjunto de Registros

- Registros de Propósito general de 16 bits
  - Se puede acceder a los 16 bits de una vez
  - Se puede acceder al byte (H) alto y al byte (L) bajo
  - AX (Acumulador)
  - BX (Base)
  - CX (Count)
  - DX (Data)
  - EAX

		AH	AL
		BH	BL
		CH	CL
		DH	DL
__	__	AH	AL

(IA 32 bits)

- Registros de Direccionamiento de Segmentos de 16 Bits
  - CS Code Segment
  - DS Data Segment
  - SS Stack Segment
  - ES Extra Segment
- Registros de Desplazamiento en Segmentos de 16 Bits
  - SP Stack Pointer
  - BP Base Pointer
  - SI Source Index
  - DI Destination Index

8086 - Conjunto de Registros

- Registros de Control y Estado de 16 bits
  - IP Instruction Pointer
  - FLAGS Registro de 16 bits
    - No se trata de un valor de 16 bits sino un colección de flags de 9 bits (seis son usados)
    - Un flag está "**Set**" cuando es igual a 1
    - Un flag está "Clear" cuando es igual a 0
    - Flags de Control
      - **Dirección** Usado en instrucciones sobre STRINGs para moverse hacia delante o atrás sobre el string
      - **Interrupt** Usado para habilitar o deshabilitar las interrupciones (mas adelante)
      - **Trap** Usado para habilitar o deshabilitar el trap de paso a paso

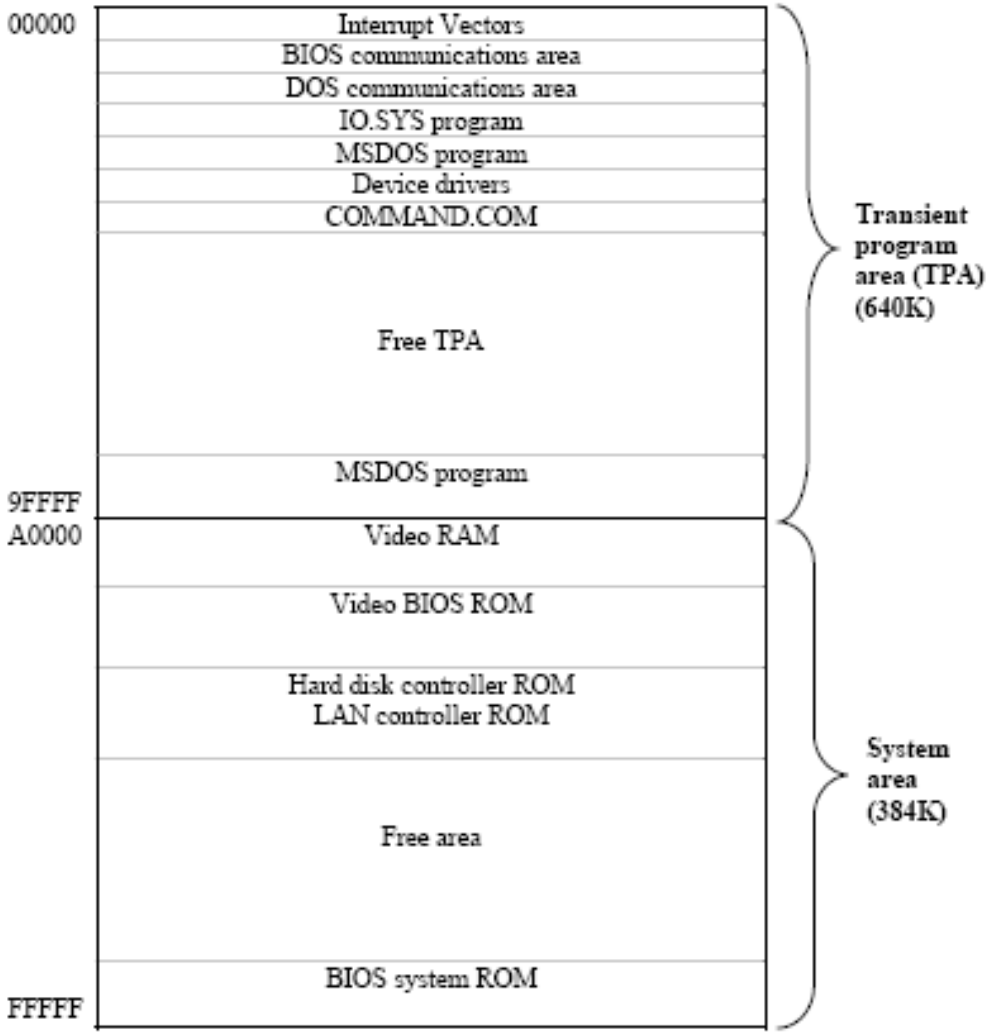
8086 - Conjunto de Registros

- Flags de Estado
  - Los flags son seteados o limpiados según efectos colaterales de una instrucción ejecutada
  - Parte del aprendizaje de una instrucción es conocer que flags modifica
  - Hay instrucciones que leen un flag e indican si está o no "seteado"
    - Flag de Estado — Nombre
      - CF — Carry (Acarreo)
      - AF — Auxiliary Carry
      - OF — Overflow
      - SF — Sign
      - ZF — Zero

8086 - Conjunto de Registros

- Otros Registros:
  - Hay otros registros que son parte del modelo del programador pero están dentro de la CPU
    - Ellos soportan la ejecución de instrucciones
      - Ej: IR — Instruction Register
      - Ej: registros E/S de ALU — Son registros temporales
    - No pueden ser accedidos directamente por el programador
    - Pueden ser más anchos que 16 bits

8086 – Mapa de Memoria 1Mb

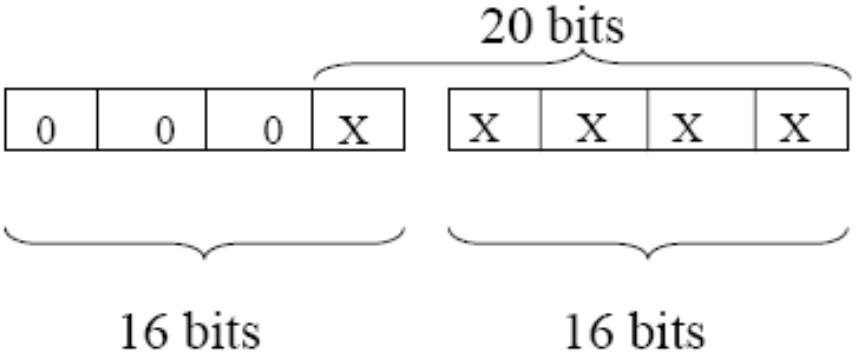


8086 – Conjunto de Registros

- Otros Registros:
  - Hay otros registros que son parte del modelo del programador pero están dentro de la CPU
    - Ellos soportan la ejecución de instrucciones
      - Ej: IR — Instruction Register
      - Ej: registros E/S de ALU — Son registros temporales
    - No pueden ser accedidos directamente por el programador
    - Pueden ser más anchos que 16 bits

8086 – Memoria Segmentada

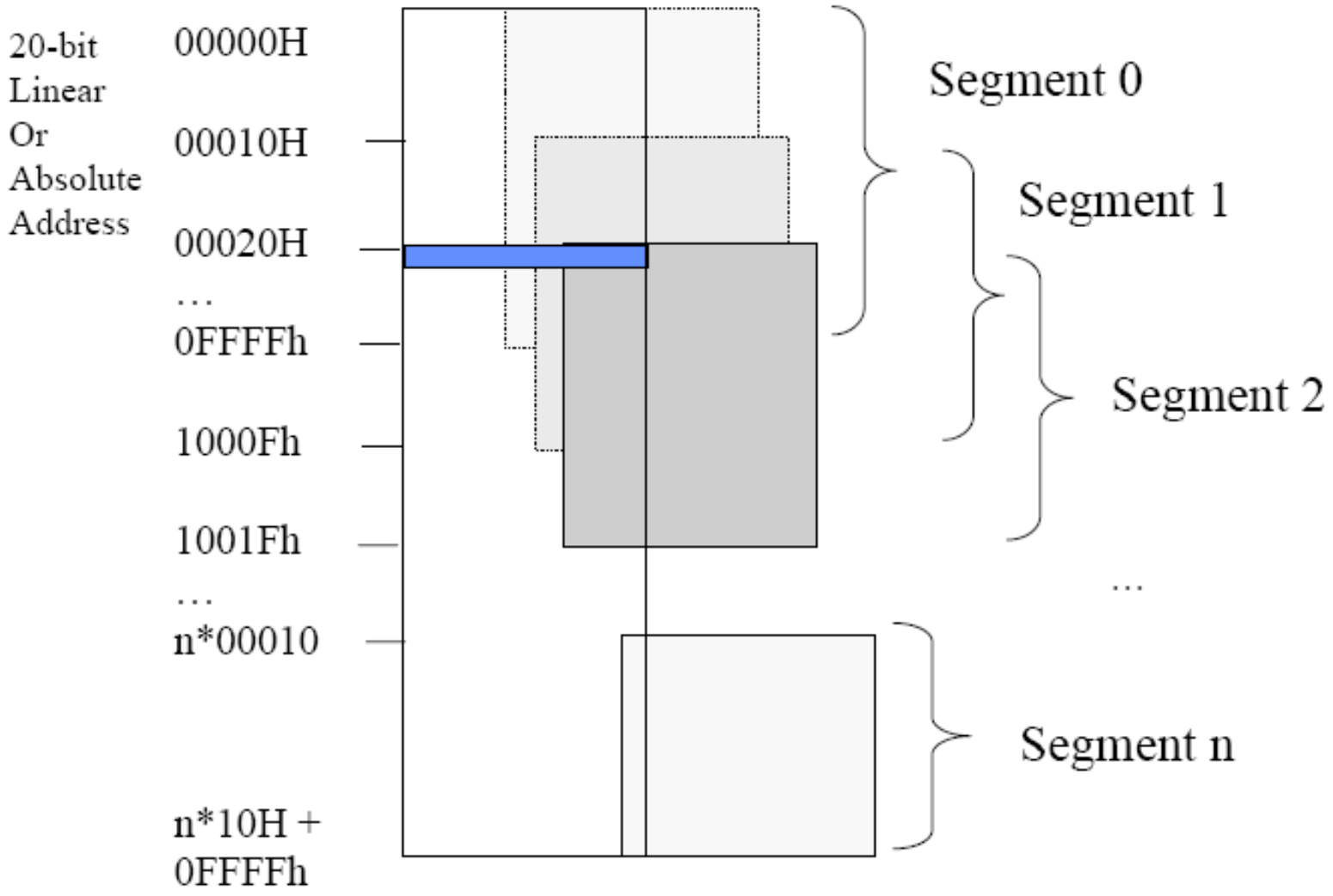
- Modelo de Memoria Segmentada para 8086 con 20 bits de espacio de Direccionamiento
  - Problema de Diseño de Procesador
    - ¿Como usar registros y valores de 16 bits para especificar direcciones de 20 bits?
    - Una forma: Usar dos registro "uno al lado del otro"



8086 – Memoria Segmentada

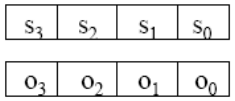
- Modo Real-Access (8086)
- Dentro del espacio lineal de direcciones (0 a 1M[2^{20}]), se pueden solapar conjuntos de segmentos.
  - Espacio lineal de direcciones, se conocen como dirección absoluta (20 bits).
  - Un segmento se define como una secuencia de bytes que:
    - Comienza cada 16 bytes = 10h bytes
      - Cada segmento comienza en una **Dirección Absoluta** que termina con 0h.
      - Dirección absoluta = Segmento:Offset
      - Offset = 2 bytes = 16 bits
    - Tienen una longitud de 64K bytes consecutivos (64K=FFFFh).
      - Hint: 216 = 64K y todos los registros del 8086 son de 16 bits
  - Un byte en particular está ubicado dentro de más de un segmento.
    - Segmento 0 Comienza en 00000H y va hasta 0FFFFh
    - Segmento 1 Comienza en 00010H y va hasta 1000Fh (0FFFFh + 10h)
    - Segmento 2 Comienza en 00020H y va hasta 1001Fh (0FFFFh + 20h)

8086 – Memoria Segmentada



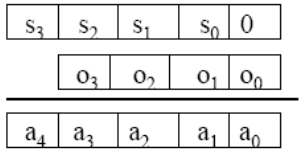
8086 – Memoria Segmentada

- A Nivel de Hardware:
  - Una dirección se coloca en el bus de direcciones de 20 bits como una dirección absoluta
- A Nivel de Programador:
  - Las direcciones NUNCA se especifican como valores de 20 bits
  - Las direcciones SIEMPRE se especifican como dos valores de 16 bits:
    - Segmento:Desplazamiento Segment:offset
- Quien hace la conversión?
  - La CPU, durante el fetch de una instrucción
    - Recordar que cada segmento comienza cada 16 bytes
    - La dirección de un segmento = Número de Segmento * 16_{10}
    - Segmento:
    - Offset:



Determinado por el nro de segmento

    - Segmento * 10h
    - Offset



Dirección de 20 bits (Dirección absoluta)

8086 – Memoria Segmentada

- Ejemplo:
  - Suponamos tener el número de segmento = 6020H y el offset 4267H
  - Segmento * 10h 60200H
  - Offset 4267H
  - Dirección de 20 bits 64467H
- Recordar:
  - Un efecto colateral de la memoria segmentada es que todo byte de memoria tiene mas de una forma de ser referido mediante pares SEGMENT:OFFSET
  - Ejemplo:
    - El único byte que se encuentra en la dirección 00300H puede ser referido por:
      - 0 H : 300 H
      - 1 H : 2F0 H
      - 30 H : 0 H

8086 – Memoria Segmentada

- 8086 incluye cuatro registros de Segmento de 16 bits:
  - CS :Code Segment Register
  - DS :Data Segment Register
  - SS :Stack Segment Register
  - ES :Extra Segment Register
- Algunos de estos registros son usados por default:
  - Todos los fetchs de instrucciones: CS:IP
  - Los accesos a datos: DS:OFFSET
- Hay que tener en cuenta que los segmentos hay que inicializarlos antes de ser usados.

8086 – Ciclo de Ejecución

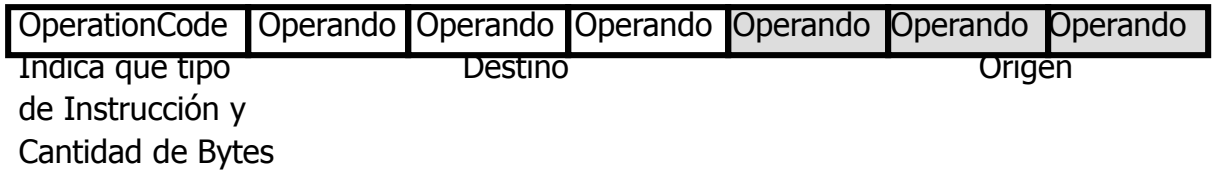
- El Procesador ejecuta instrucciones repitiendo:
  - do {

    Fetch de la instrucción: IR := mem[CS:IP] y se ajusta el IP para apuntar a la siguiente instrucción secuencial.

    Ejecuta la instrucción en IR

    ` Si hay alguna interrupción se ejecuta aquí lo que le corresponde

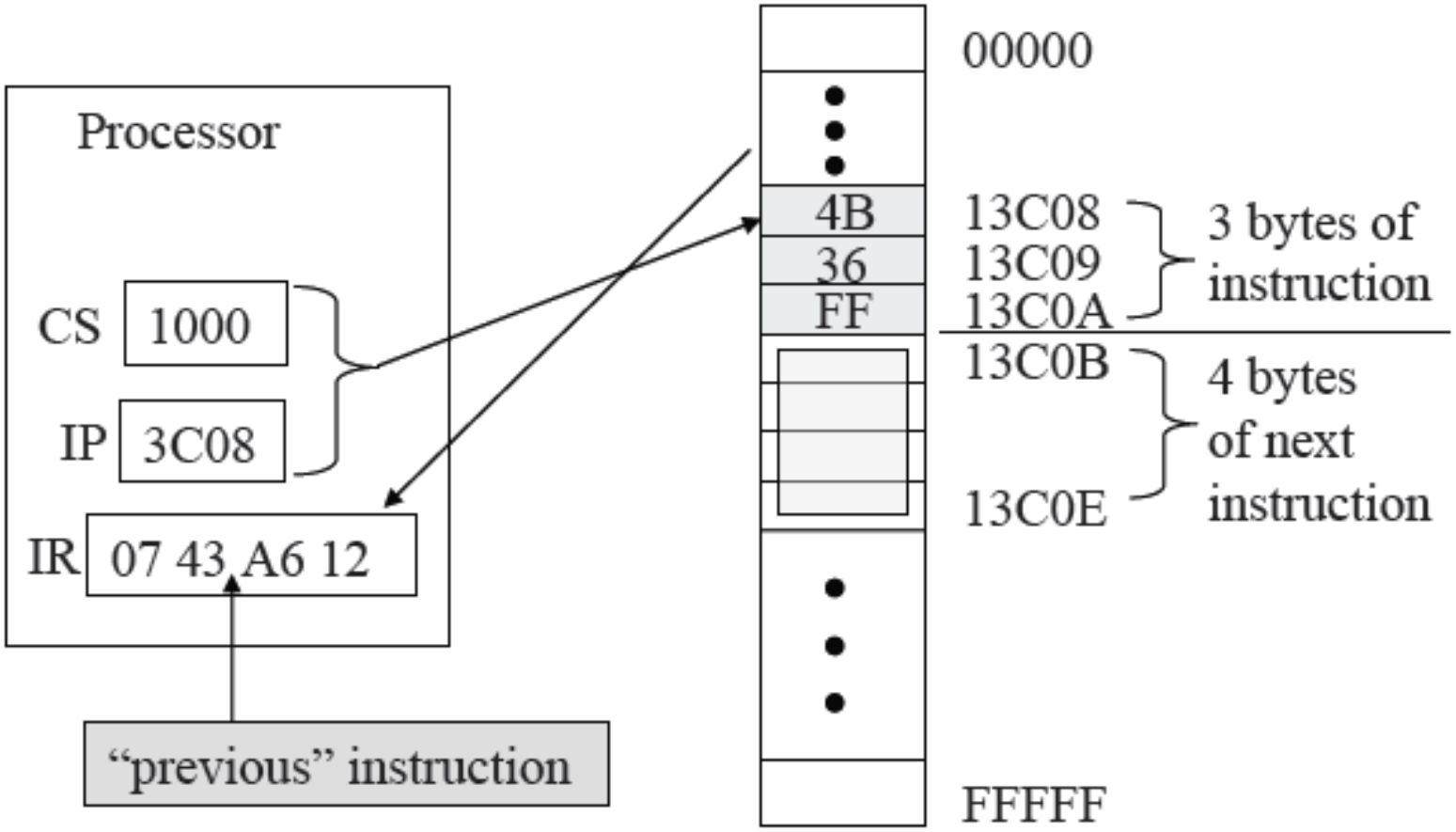
    } hasta que la instrucción HLT sea ejecutada.
- ¿Que es una instrucción?
  - En 8086, una instrucción es una secuencia de 1 a 6 bytes
  - Su estructura es algo similar a la siguiente



- Error común: No se aplica little endian a una instrucción, solo se lo aplica a operaciones con words, no a secuencias de bytes.

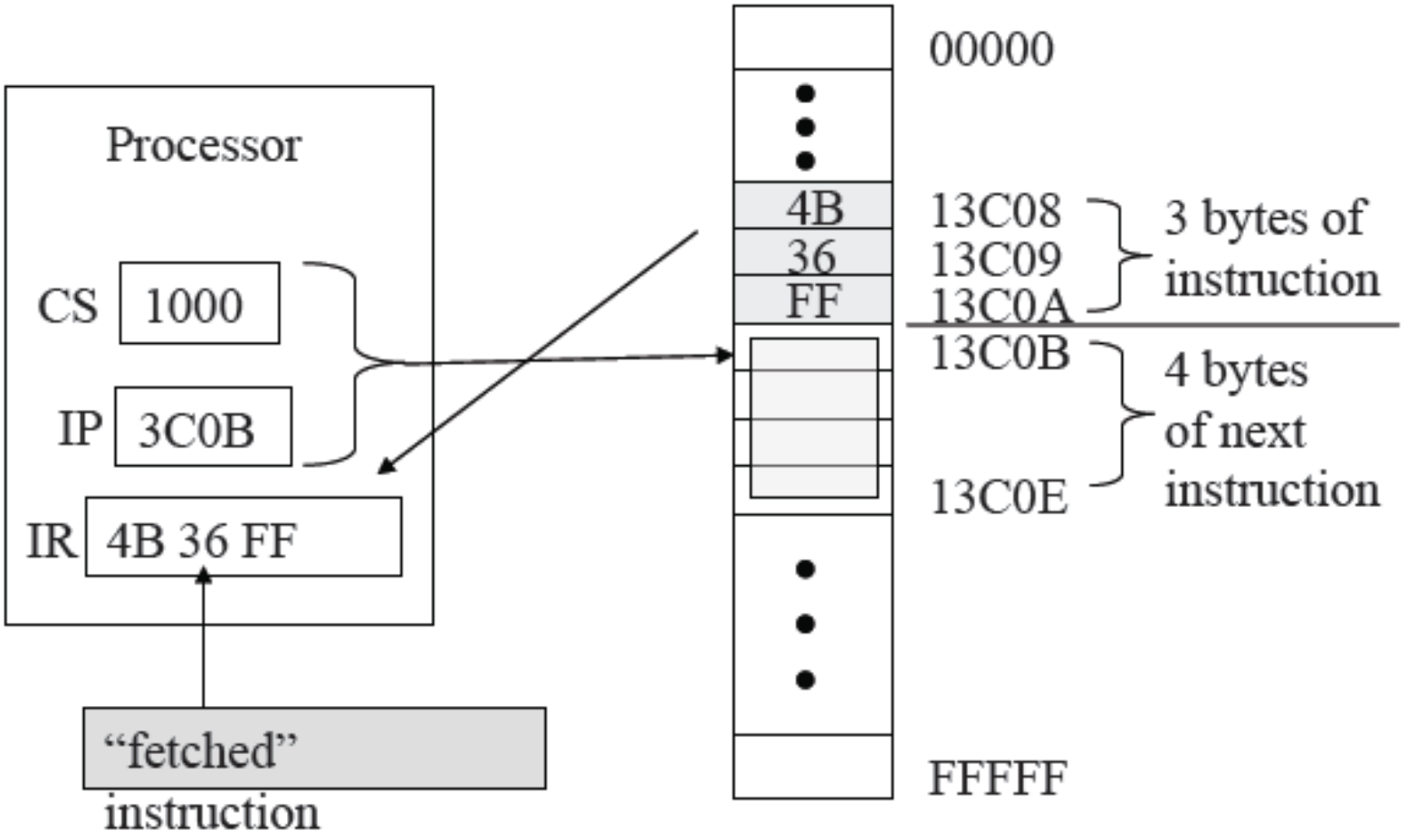
8086 – Ciclo de Ejecución

- Antes del Fetch



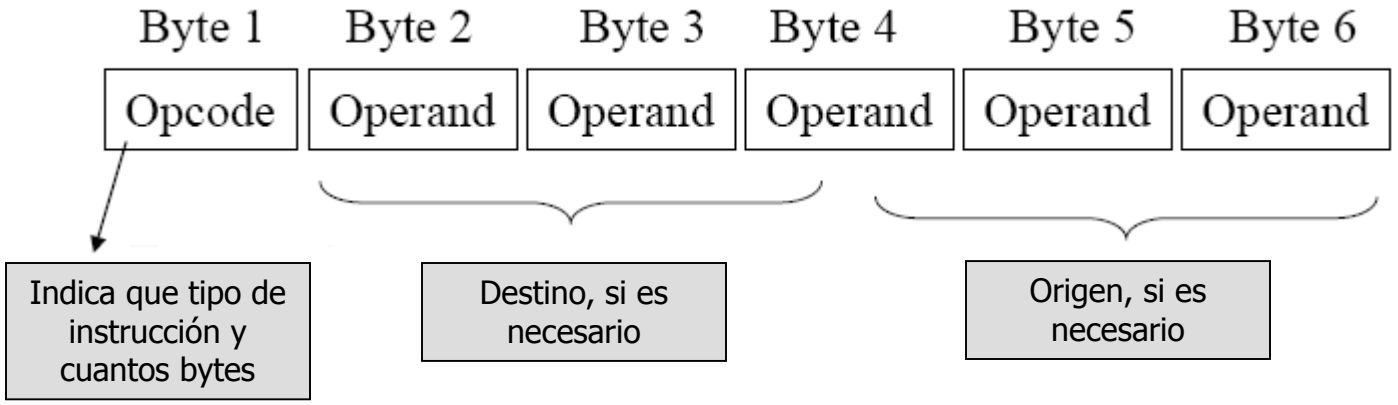
8086 – Ciclo de Ejecución

- Despues del Fetch



8086 – Codificación de Instrucciones

- En el 8086 las instrucciones es una secuencia de bytes de 1 a 6 bytes.



- # de bytes depende de # operandos
  - Nop 10010000
  - INC BX 01000001
  - ADD BX, 1 10000011 11000011 00000001 00000000

8086 – Codificación de Instrucciones

- # de bytes depende del direccionamiento

			BX | AX	
ADD BX, AX	00000001	11000011		
			Registro	Dir. Inmediato
ADD B**X**,1	1000000**1**	11000011	00000001	00000000
ADD B**L**,1	1000000**0**	11000011	00000001	
			Registro	Dir. Directo
ADD B**X**,[1]	0000001**1**	00011110	00000001	00000000
ADD B**L**,[1]	0000001**0**	00011110	00000001	

8086 – Codificación de Instrucciones

- # de bytes depende del direccionamiento

	puntero a WORD	Dir Indirecto	Dir. Inmediato
ADD [BX], 1	10000001 00000111	00000001	00000000

	Base	Constante	Dir. Inmediato
ADD [BX+2],1	10000001 01000111	00000010	00000001 00000000

	Base Indexada	Dir. Inmediato
ADD [BX+SI],1	10000001 00000000	00000001 00000000

	Base Indexada con cte	Dir. Inmediato
ADD [BX+2],1	10000001 01000000	00000010 00000001 00000000

8086 – Codificación de Instrucciones

- Codificación de Saltos
  - En todos los saltos, el destino debe ser un valor a ser usado en el regitro IP.
  - Dirección Absoluta: IP:=Nuevo Valor
  - Dirección Relativa: IP:= IP + Valor
  - Por Registro o Indirecta: IP:= registro
    IP:= mem[Dirección]
  - Saltos Incondicionales siempre usan direcciones relativas.

1) **IP**=034H **IR**=????

2) **IP**=034H **IR**=E91002

3) **IP**=247H **IR**=E91002

Dirección	Memoria	.asm
0034H	E9 10 02	JMP here
0037H	….	….
….	….	….
0247		here: