(2)

UNIVERSIDAD AUTONOMA DE MADRID

Unidad 2
Modelo de Programación del 80x86 de Intel

SISTEMAS BASADOS EN MICROPROCESADORES

Grado en Ingeniería Informática
EPS - UAM

Índice

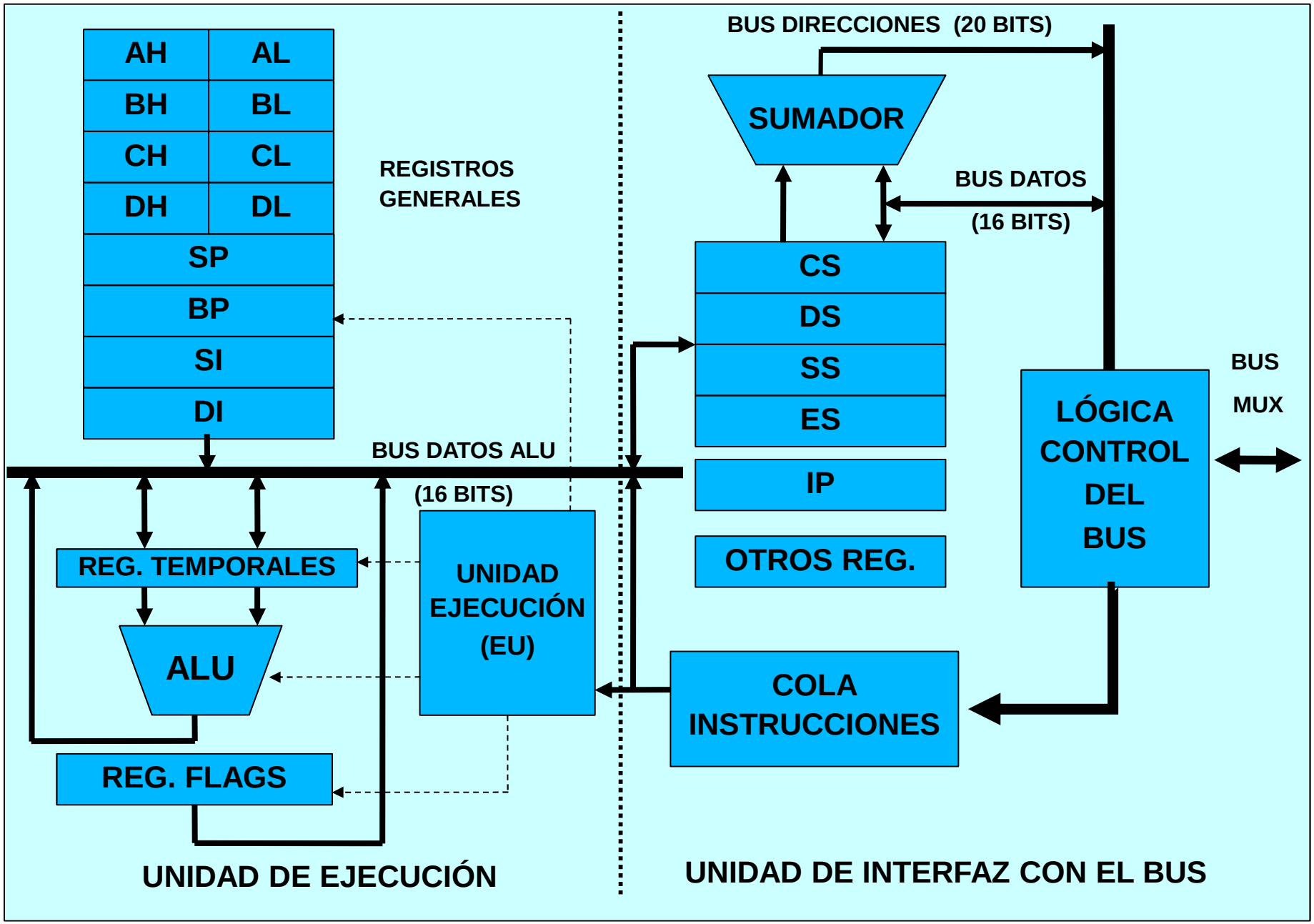
(2)

2.1. Familia 80x86 como caso particular

- Microprocesadores aparecen en los 70 (1971-…) con 4 bits y luego 8 bits (**8085** con 64KB de memoria).
- Inventados por **Intel** como circuitos integrados digitales y programables para sustituir circuitos digitales cableados.
- Familia 80x86 nace en 1978 con el **8086** (16 bits y 1 MB memoria). Continúa con: **80186**, **80286**, **80386**, **80486**, …
- En paralelo aparece **8088** (ordenador personal de IBM o PC): **8086** de 8 bits.
- Competidor inicial: **Motorola 6800** (8 bits) y **68000** (16 bits).
- Intel garantiza compatibilidad de sus microprocesadores desde los inicios e introduce la segmentación de memoria (segmentos de 64 KB)
- Tecnología **CISC** vs. **RISC** (más actual)

(2)

2.2. Registros internos y arquitectura del 80x86 (I)

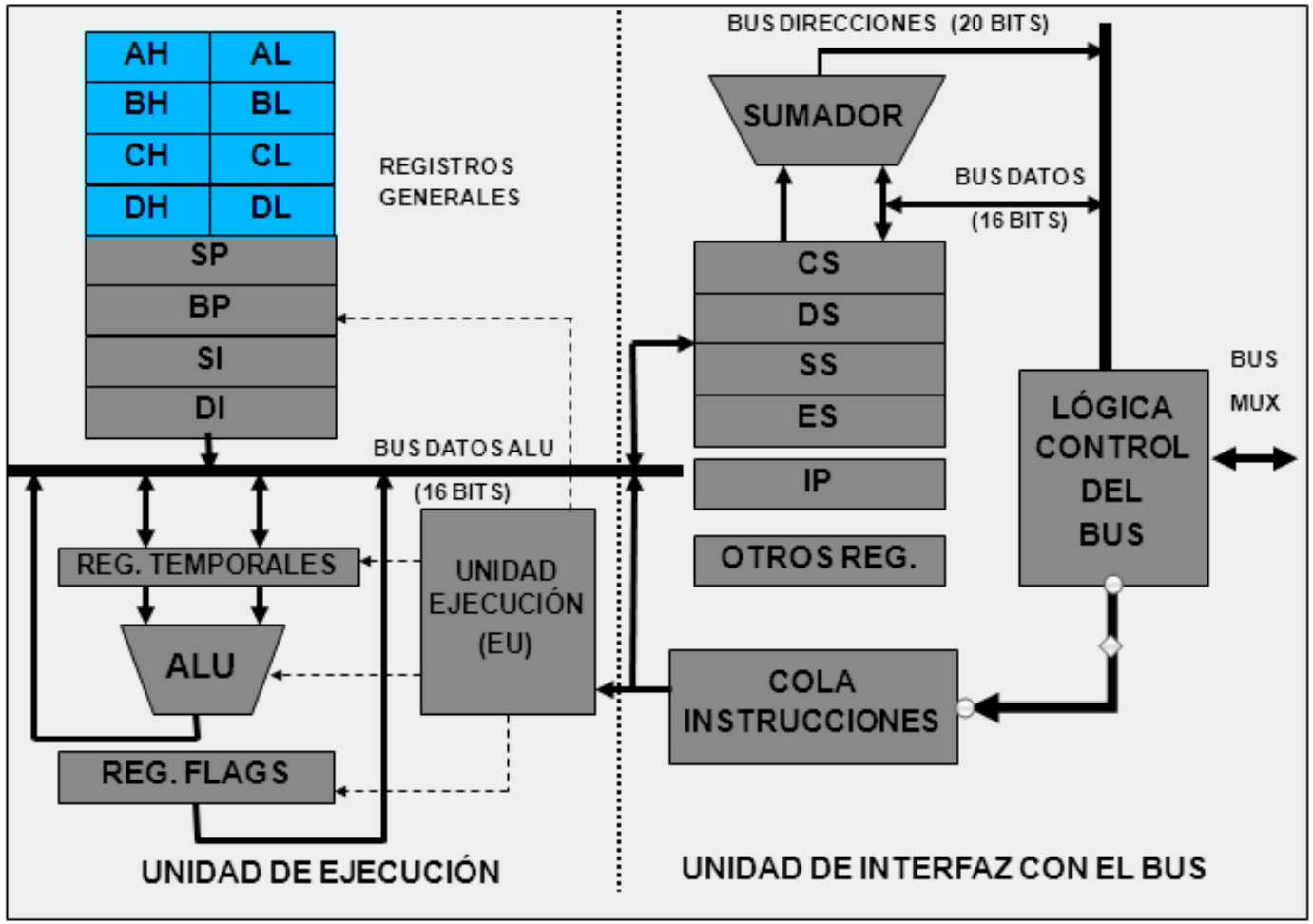


(2)

2.2. Registros internos y arquitectura del 80x86 (II)

- Registros de datos

  A**X** (A**H**-A**L**), B**X** (B**H**-B**L**), C**X** (C**H**-C**L**), D**X** (D**H**-D**L**)



(2)

2.2. Registros internos y arquitectura del 80x86 (III)

- Registros de datos

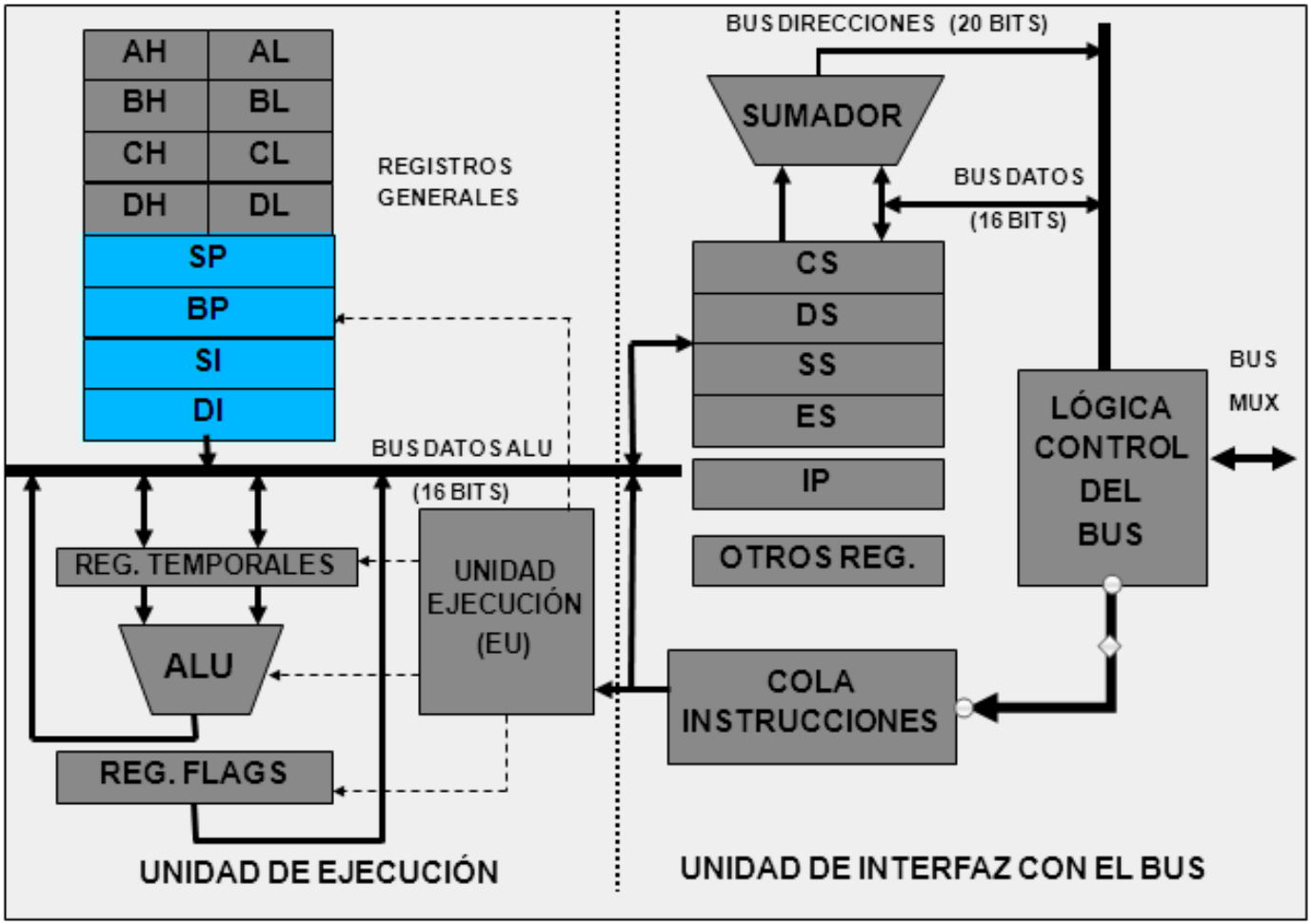
  A**X** (A**H**-A**L**), B**X** (B**H**-B**L**), C**X** (C**H**-C**L**), D**X** (D**H**-D**L**)

  - Actúan como acumuladores en instrucciones de transferencia, lógicas y aritméticas.
  - Cada uno de 16 bits, divisible en 2 registros de 8 bits.
  - Tareas específicas en algunos casos (para cualquier uso si están libres):
    - **AX**: Multiplicar, dividir y operaciones de E/S.
    - **BX**: Registro base para direccionamiento indirecto (apunta a la base de una tabla)
    - **CX**: Contador de bucles.
    - **DX**: Multiplicar, dividir, operaciones de E/S.

(2)

2.2. Registros internos y arquitectura del 80x86 (IV)

- Registros punteros: **SP , BP , SI , DI**



(2)

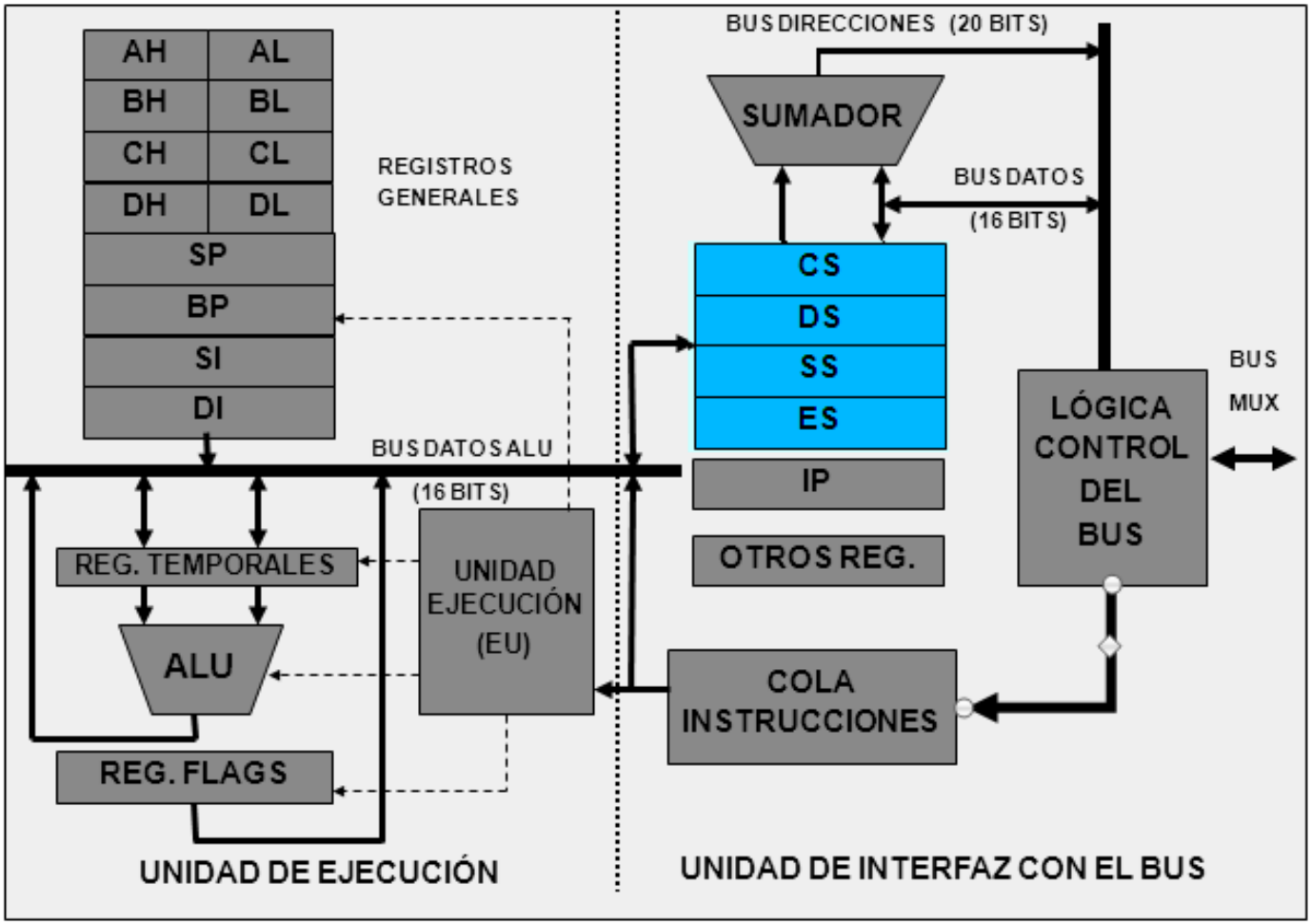
2.2. Registros internos y arquitectura del 80x86 (V)

- Registros punteros: **SP , BP , SI , DI**
  - Intervienen en el direccionamiento de memoria como desplazamientos (*offsets*) respecto a las zonas de memoria indicadas en registros de segmento.
  - **SP** (*Stack Pointer*): Usado junto al registro de segmento de pila **SS**. Interviene en:
    - Llamadas a subrutinas
    - Interrupciones
    - Instrucciones de manejo de pila
  - **BP** (*Base Pointer*): Usado junto al registro de segmento de pila **SS**. Útil para acceder a los parámetros de subrutinas pasados por pila.
  - **SI** (*Source Index*): Usado para indexar tablas en memoria (lectura). Para cualquier uso si está libre.
  - **DI** (*Destination Index*): Usado para indexar tablas en memoria (escritura). Para cualquier uso si está libre.

(2)

2.2. Registros internos y arquitectura del 80x86 (VI)

- Registros de segmento: **CS , SS , DS , ES**



(2)

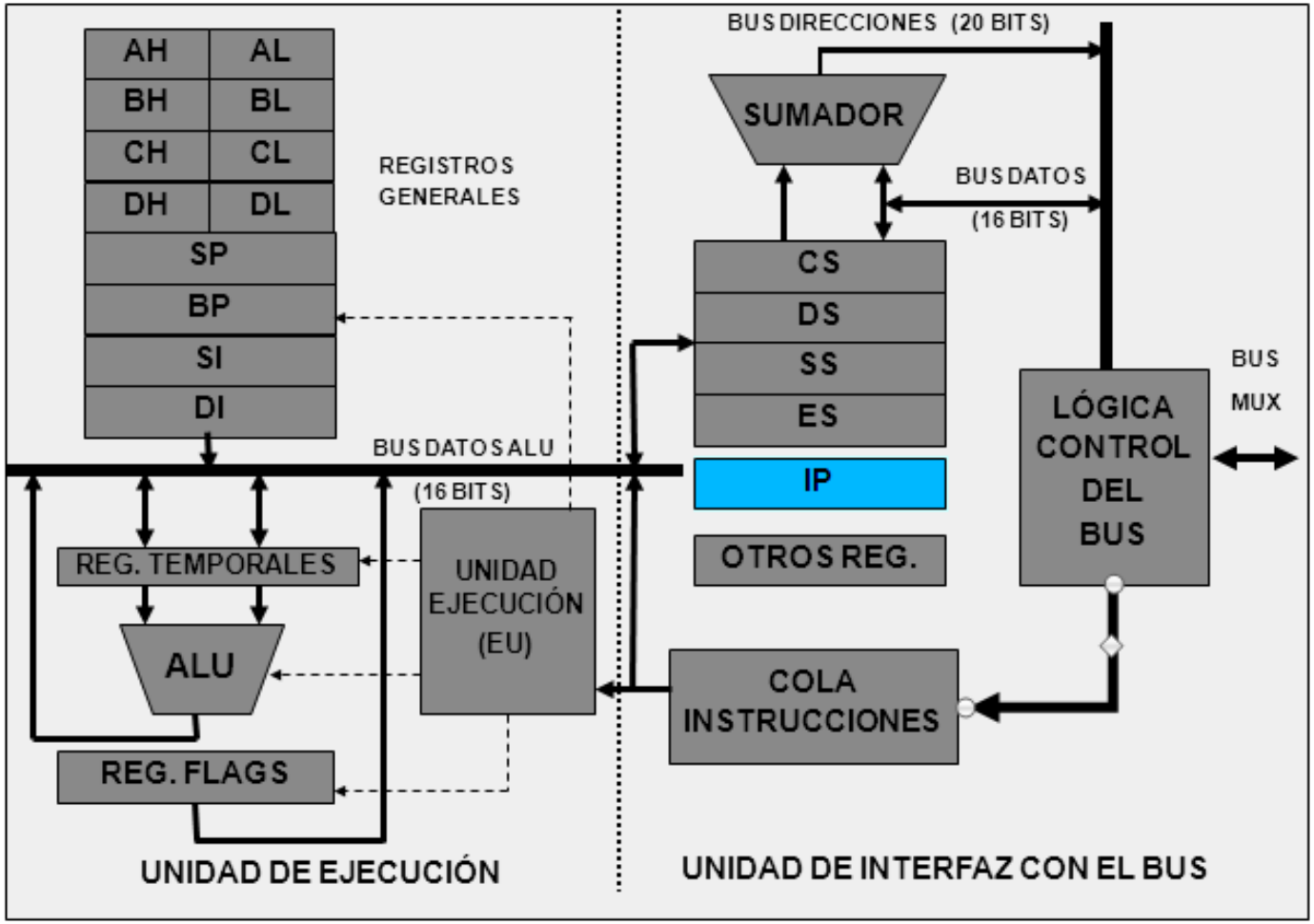
2.2. Registros internos y arquitectura del 80x86 (VII)

- Registros de segmento: **CS , SS , DS , ES**
  - Intervienen en el direccionamiento de memoria indicando zonas de 64KB de memoria (segmentos).
  - **CS** (*Code Segment*): Indica el segmento de código máquina (programa). Junto con el puntero de instrucciones **IP** constituye el *contador de programa*.
  - **SS** (*Stack Segment*): Indica el segmento de pila. Junto con **SP** o **BP** indica una posición absoluta de memoria en la pila.
  - **DS** (*Data Segment*): Indica el segmento principal de datos (variables globales).
  - **ES** (*Extra Segment*): Indica el segmento adicional de datos (variable globales).

(2)

2.2. Registros internos y arquitectura del 80x86 (VIII)

- Registro puntero de instrucciones: **IP**



(2)

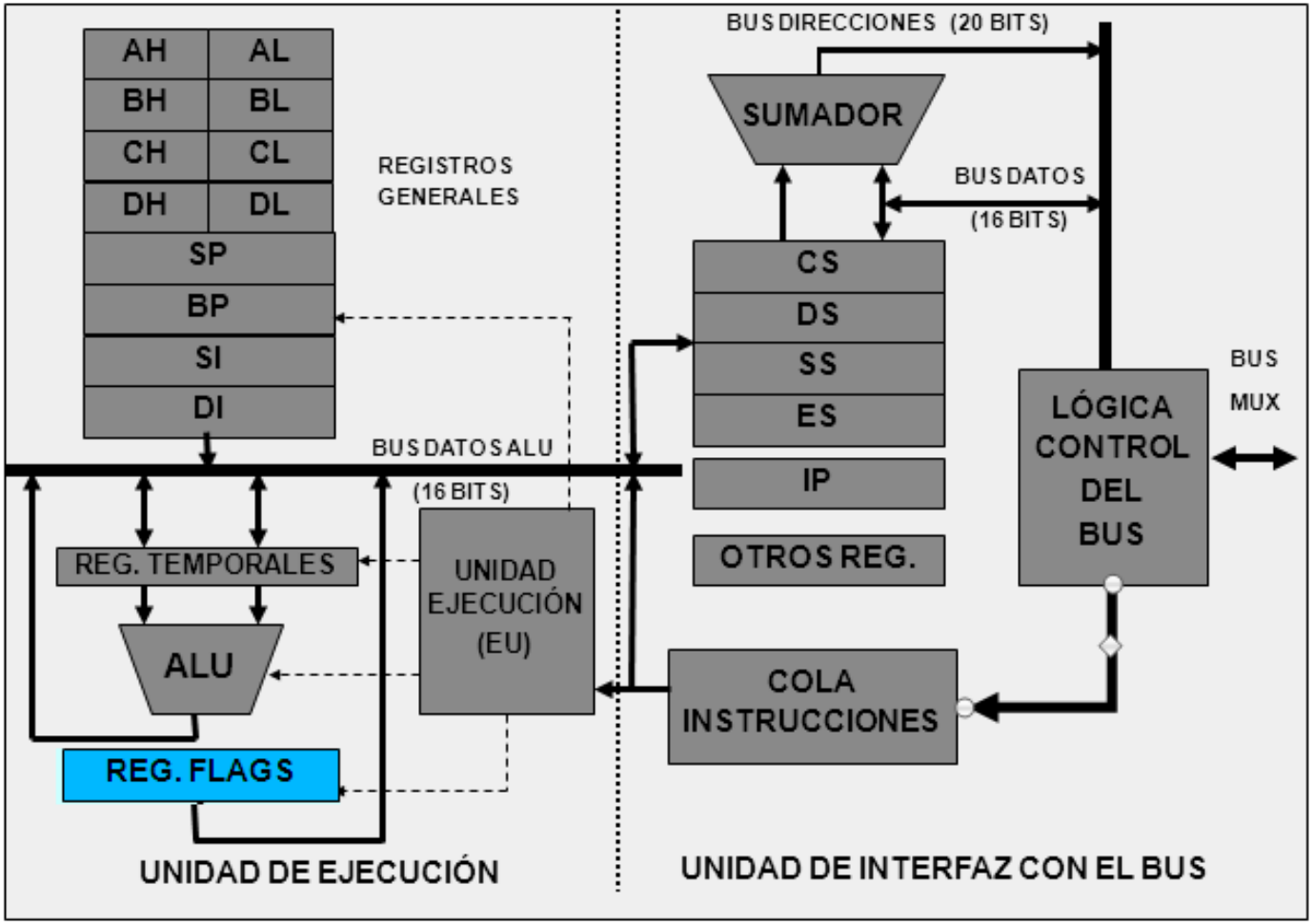
2.2. Registros internos y arquitectura del 80x86 (IX)

- Registro puntero de instrucciones: **IP**
  - Indica el desplazamiento (*offset*) dentro del segmento indicado por **CS** donde se encuentra la siguiente instrucción de código máquina que va a ser ejecutada (contador de programa).

(2)

2.2. Registros internos y arquitectura del 80x86 (X)

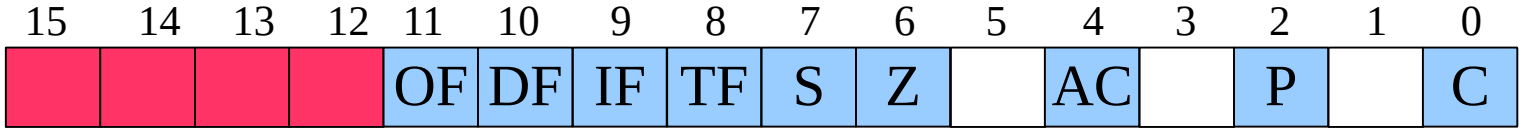
- Registro de estado (*FLAGS*)



(2)

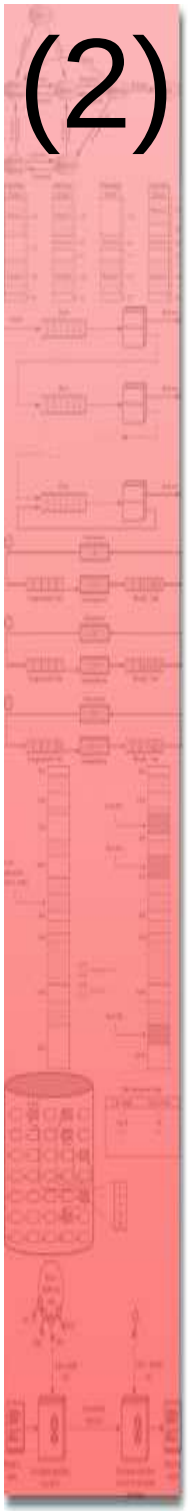
2.2. Registros internos y arquitectura del 80x86 (XI)

- Registro de estado (*FLAGS*)
  - Algunos de sus 16 bits indican información de estado del procesador y de la última operación de la ALU.



IF: bit de interrupciones
DF: bit de dirección
OF: bit de *overflow*
Z: bit de cero
S: bit de signo
TF: bit de *trap*
C: bit de acarreo
P: bit de paridad
AC: bit de acarreo auxiliar

  - Las banderas (*flags*) **C**, **AC**, **S**, **P**, **Z** y **OF** dependen del resultado de la última operación ejecutada por la ALU.
  - La bandera **IF** habilita o deshabilita las interrupciones *hardware*.
  - La bandera **TF** habilita o deshabilita la ejecución “paso a paso”.
  - La bandera **DF** incrementa o decrementa los punteros índice en instrucciones de cadena.
  - Todos los bits pueden ponerse a **0** o a **1** con instrucciones específicas.



2.3. Acceso y organización de la memoria (I)

- Memoria física de un sistema basado en 8086 organizada como 2^{20} posiciones de 1 byte (1 MB).
- Memoria física de 1 MB dividida a nivel lógico en "segmentos" de 64 KB.
- Los segmentos empiezan en direcciones múltiplo de 16.
- Dos segmentos consecutivos están separados por 16 bytes.
- En un programa, las instrucciones suelen estar en un segmento, los datos en uno o varios segmentos distintos y la pila en otro (hay casos en que esto no se cumple).
- La CPU puede acceder a la vez hasta a cuatro segmentos distintos (registros **CS**, **DS**, **ES** y **SS** con valores distintos).
- Puede haber solapamiento total o parcial de segmentos (caso extremo: **CS**, **DS**, **ES** y **SS** con mismo valor).
- El programa puede cambiar en cualquier momento el valor de los registros de segmento.

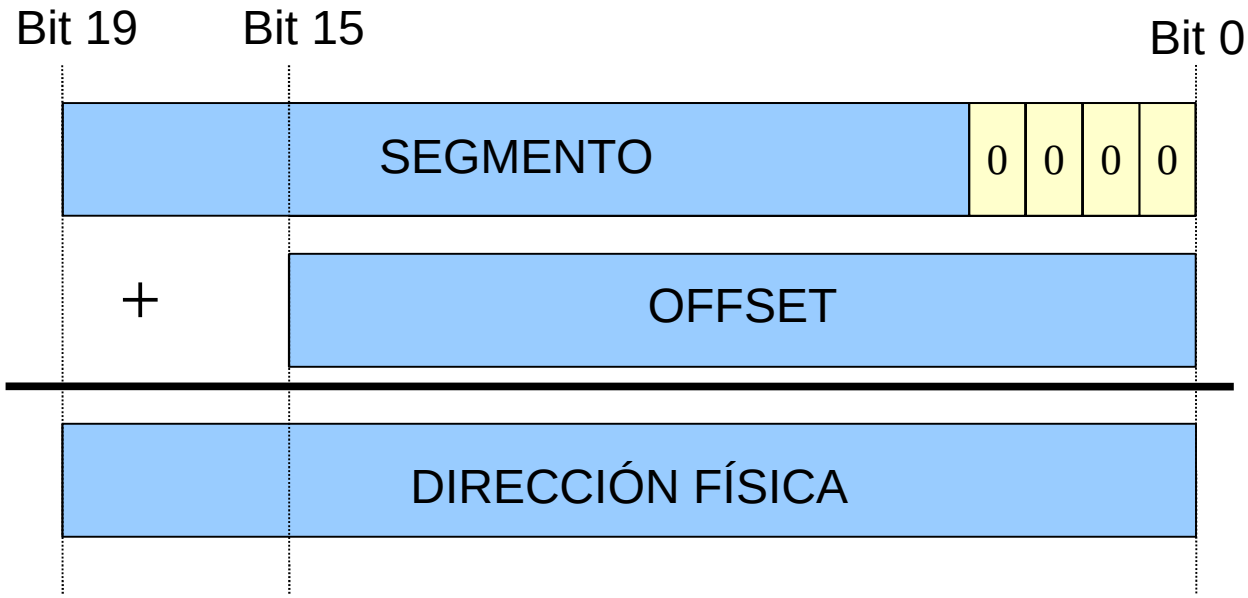
(2)

2.3. Acceso y organización de la memoria (II)

- Acceso a memoria (modo real)

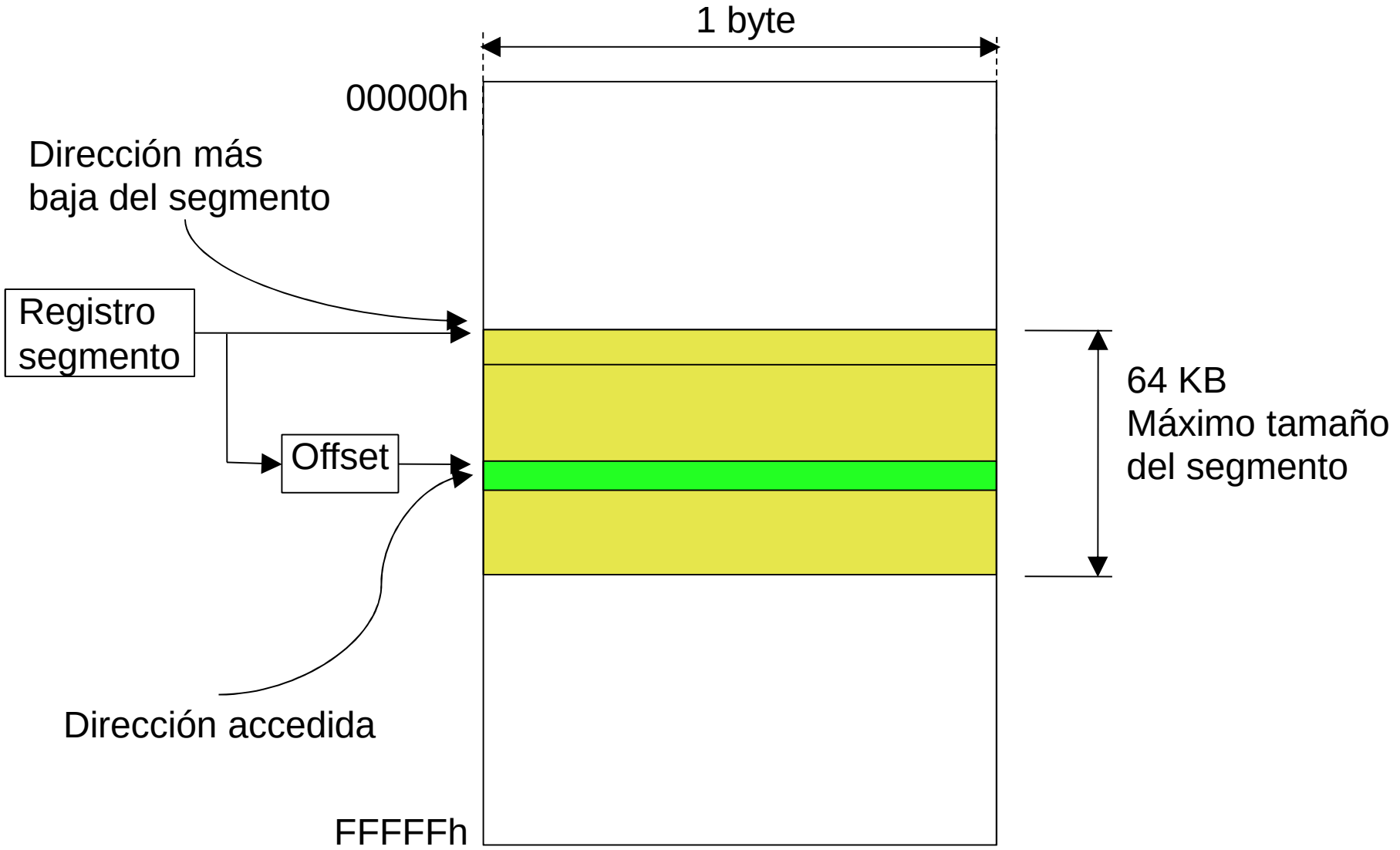
  Hardware: 20 bits de dirección (A19-A0)
  Software: 32 bits (16 bits de Segmento y 16 bits de Offset)

DIRECCIÓN FÍSICA = **Segmento** x **16** + **Offset**



(2)

2.3. Acceso y organización de la memoria (III)



2.3. Acceso y organización de la memoria (IV)

- Ejemplos de acceso a memoria (modo real)
  - ***CS* = A783h** ***(segmento)***
    IP* = 403Eh** ***(offset)

    Dirección física = **A783h** x **16** + **403Eh** =
    A783h x **10h** + **403Eh =**
    A7830h + **403Eh** = **AB86Eh**

  - ***ES* = 54A3h** ***(segmento)***
    DI* = 1F2Bh** ***(offset)

    Dirección física = **54A30h** + **1F2Bh** = **5695Bh**

  - ***SS* = 4675h**
    ***SP* = A001h**

    Dirección física = **46750h** + **A001h** = **50751h**

(2)

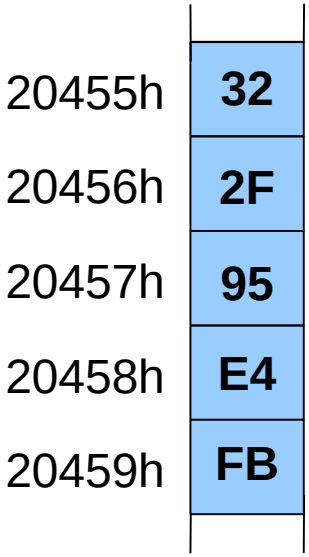
2.3. Acceso y organización de la memoria (V)

- Acceso a memoria desde programas
  - El acceso puede ser a un byte o dos consecutivos (una palabra) según el registro que intervenga en la instrucción.
  - ***Ejemplo***: si previamente se han ejecutado las siguientes instrucciones:

```
mov AX, 2000h
mov DS, AX
```

el resultado de las siguientes operaciones es:

```
mov AX, [455h]      ;  AX = 2F32h
mov AX, [456h]      ;  AX = 952Fh
mov AH, [457h]      ;  AH = 95h
mov AL, [458h]      ;  AL = E4h
```



2.4. Modos de direccionamiento (I)

- Siete modos de direccionamiento:
  - Inmediato
  - Por registro
  - Directo
  - Indirecto
  - Relativo
  - Indexado
  - Implícito
- Los modos directo e indirecto consisten en “punteros” a memoria.

(2)

2.4. Modos de direccionamiento (II)

- **Direccionamiento inmediato**

  El operando fuente siempre es un valor y el destino un registro.

  Ejemplos:

```
mov CL, 3Fh          ; 3Fh ⇒ CL
mov SI, 4567h        ; 4567h ⇒ SI
```

- **Direccionamiento por registro**

  Ambos operandos son siempre registros.

  Ejemplos:

```
mov DX, CX           ; CX ⇒ DX
mov BH, CL           ; CL ⇒ BH
```

(2)

2.4. Modos de direccionamiento (III)

- Direccionamiento directo

  El *offset* de la posición de memoria a la que se quiere acceder se especifica en la instrucción. Por defecto, el segmento lo indica **DS**.

  Ejemplos si DS = **3000h**:

```
mov DX, [678Ah]   ;  carga en DL el contenido de la posición
                  ;  de memoria 3678Ah y en DH el
                  ;  contenido de la posición de memoria
                  ;  3678Bh.
mov AL, [32h]     ;  carga en AL el contenido de la posición
                  ;  de memoria 30032h
mov [800h], BL    ;  carga en la posición de memoria 30800h
                  ;  el contenido de BL.
```

(2)

2.4. Modos de direccionamiento (IV)

- Direccionamiento indirecto por registro

  La dirección efectiva del operando está contenida en uno de los registros **BX**, **BP**, **SI** o **DI**.

  Ejemplo:

  ```
  mov AX, [BX]
  ```

- Direccionamiento relativo a base

  La dirección efectiva se obtiene sumando un desplazamiento al registro **BX** o al **BP**.

  Ejemplos equivalentes si *offset* de la TABLA es 4:

  ```
  mov AX, [BX]+4
  mov AX, 4[BX]
  mov AX, TABLA[BX]
  mov AX, [BX+4]
  ```

2.4. Modos de direccionamiento (V)

- Direccionamiento indexado

  La dirección efectiva se calcula sumando un desplazamiento al contenido de **SI** o **DI**.

  Ejemplos equivalentes si *offset* de la TABLA es 4:

  ```
  mov AX, [SI]+4
  mov AX, 4[SI]
  mov AX, TABLA[SI]
  mov AX, [SI+4]
  ```

- Direccionamiento indexado a base

  La dirección efectiva se obtiene sumando **BX** o **BP** con **SI** o **DI** y/o un offset directo.

  Ejemplos:

  ```
  mov AX, TABLA[BX][SI]
  mov AX, TABLA+[BX]+[SI]
  ```

2.4. Modos de direccionamiento (VI)

- ## Direccionamiento relativo

  Usado en saltos condicionales: El operando es un desplazamiento de 8 bits con signo (-128 a 127) que se suma al puntero de instrucciones **IP**.

  Ejemplos:

```
jnc 26
jz etiqueta   ; si la etiqueta está a una distancia
              ; mayor o igual que -128 y menor que 128
```

- ## Direccionamiento implícito

  No es necesario indicar el operando (es implícito).

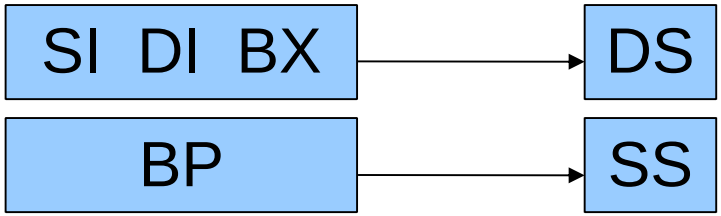
  Ejemplos:

```
cli     ; pone a 0 el flag de interrupciones
stc     ; pone a 1 el flag de acarreo
```

(2)

2.4. Modos de direccionamiento (VII)

- Registros de segmento por defecto en direccionamiento indirecto, relativo e indexado:



- Uso forzado de otro registro de segmento:

  La dirección se prefija con el registro deseado.

- Ejemplos con **DS** = **3000h**, **CS** = **2000h**, **ES** = **A000h**, **SS** = **E000h**, **SI** = **100h** y **BP** = **500h**

```
mov DX, [678Ah]              ; [3678Ah] y [3678Bh]  ⇒ DX
mov DX, CS:[678Ah]           ; [2678Ah] y [2678Bh]  ⇒ DX
mov ES:[SI], AL              ; AL  ⇒  [A0100h]
mov SS:[1000h+SI], CX        ; CL ⇒ [E1100h]  y CH ⇒ [E1101h]
mov SI, [BP]                 ; [E0500h] y [E0501h]  ⇒  SI
mov DS:[BP], DI              ; DI  ⇒  [30500h] y [30501h]
```

(2)

2.4. Modos de direccionamiento (VIII)

- Si en la instrucción no aparece ningún registro, el número de bytes de la transferencia se indica explícitamente:

```
mov BYTE PTR [ 3Ah ], 4Fh           ;  4Fh ⇒ [3003Ah]
mov WORD PTR [ 3Ah ], 4Fh           ;  4Fh ⇒ [3003Ah] ,  0 ⇒ [3003Bh]
mov WORD PTR ES:[ 3Ah ], 2000h      ; 0 ⇒ [A003Ah] ,  20h ⇒ [A003Bh]
mov BYTE PTR [ 3Ah+SI ], 4          ; 4 ⇒ [3013Ah]
```

(2)

2.5. Directivas y operadores del ensamblador del 80x86 (I)

- Las directivas son instrucciones para el ensamblador.
- No se traducen a instrucciones de código máquina.
- Tres tipos principales de directivas:
  - Definición de símbolos
  - Definición de datos
  - Definición de segmentos y procedimientos

(2)

2.5. Directivas y operadores del ensamblador del 80x86 (II)

- Directivas de definición de símbolos

  Asignan nombres simbólicos a expresiones. Después de la asignación, el nombre puede ser empleado en cualquier parte del programa.

  - EQU puede usarse para asignar texto o expresiones numéricas.
  - = sólo permite asignaciones numéricas y puede redefinirse.

Ejemplos:

```
K           EQU     1024
TABLA       EQU     TABLA[BX+SI]
K2          EQU     K
CONTADOR    EQU     CX
DOBLEK      EQU     2*K
MIN_DIAS    EQU     60*24

CONSTANTE   =       20h
CONSTANTE   =       CONSTANTE + 1
```

(2)

2.5. Directivas y operadores del ensamblador del 80x86 (III)

- Directivas de definición de datos

  Reservan espacio de memoria, asignan un valor y definen un nombre para la variable.

  - **DB** reserva 1 byte
  - **DW** reserva 2 bytes (1 palabra)
  - **DD** reserva 4 bytes (2 palabras)
  - **DQ** reserva 8 bytes (4 palabras)

- Ejemplos:

```
NUMEROS     DB      4, 5*9, 10h+4, 23h, 'A'          ; 1 byte por elemento
TEXTO  DB   "Final", 13, 0Ah
NUM         DW      1000, -200, 400/60, 80h          ; 2 bytes por elemento
NUMMM       DD      200000h                          ; 4 bytes por elemento
            DB      6 dup (10h)                      ; 10h seis veces seguidas
            DB      10h dup("Pila")                  ; PilaPilaPilaPila .....
LETRA       DB      ?               ; reserva 1 byte sin asignar valor
LETRAS      DB      8 dup (?)
CEERCANO    DW      LETRA           ; almacena offset de LETRA
LEJANO      DD      LETRA           ; almacena offset y segmento de LETRA
```

(2)

2.5. Directivas y operadores del ensamblador del 80x86 (IV)

- Directivas de definición de segmentos y procedimientos
  - **SEGMENT** y **ENDS**: Delimitan el inicio y final de un segmento lógico (conjunto de variables o instrucciones de ensamblador) y le dan un nombre. El segmento de pila se llama **STACK**.
  - **ASSUME reg_seg : nombre_segmento[, ...]**: Indican el registro de segmento por defecto para direccionar las variables contenidas en el segmento lógico indicado.
  - **PROC** y **ENDP**: Delimitan el principio y final de un procedimiento (rutina, subrutina, …).
    - Un procedimiento es una parte de un programa que puede ser accedido desde diferentes lugares de un programa.
    - El procedimiento puede ser **NEAR** (cercano) o **FAR** (lejano).
    - Cercano: sólo se puede llamar desde mismo segmento.
    - Lejano: puede ser llamado desde cualquier segmento.

(2)

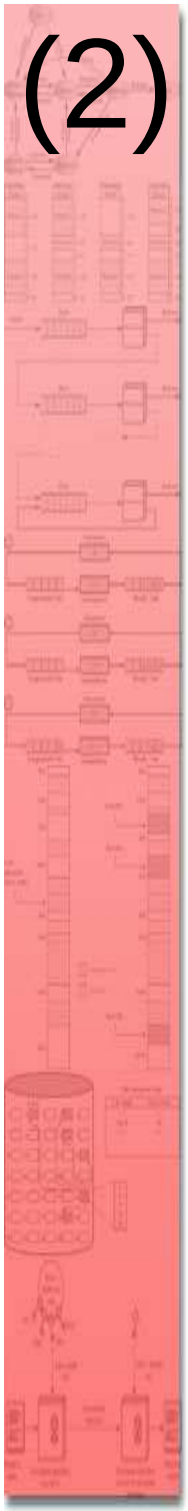
2.5. Directivas y operadores del ensamblador del 80x86 (V)

- Directivas de definición de segmentos y procedimientos
  - **PUBLIC**: Indica al montador (*linker*) que una etiqueta (variable o procedimiento) declarada en el fichero puede ser referenciada desde otros ficheros (es pública).
  - **EXTRN**: Indica al montador (*linker*) que una etiqueta está declarada en otro fichero (es externa).
  - **ORG *offset***: Fuerza que la siguiente variable o instrucción de código máquina empiece en el desplazamiento (*offset*) indicado.
  - **END:** Indica final del programa. Si va seguido por una etiqueta, indica al ensamblador la primera instrucción que debe ser ejecutada.

(2)

2.5. Directivas y operadores del ensamblador del 80x86 (VI)

- Los operadores son modificadores que aparecen en una directiva o instrucción de ensamblador.
- Su valor se calcula en tiempo de ensamblado, por lo que no pueden contener variables ni registros.
- Cuatro tipos de operadores:
  - Operadores aritméticos
  - Operadores lógicos
  - Operadores que devuelven valores
  - Operadores de atributo



2.5. Directivas y operadores del ensamblador del 80x86 (VII)

- Operadores aritméticos: **+** , **-** , ***** , **/** , **MOD** , **SHL** , **SHR**
  - Combinan operandos numéricos para dar un resultado.
  - Ejemplos:

```
PI EQU 31415 / 10000            ;  cociente de división entera
MOV AX, 2 * PI
MOV CX, 31415 MOD 10000         ;  resto de división entera
VAL     EQU 10011101b
VAL2    EQU VAL SHL 2     ;   VAL2 vale 1001110100b
VAL3    EQU VAL SHR 2     ;   VAL3 vale 100111b
```

(2)

2.5. Directivas y operadores del ensamblador del 80x86 (VIII)

- Operadores lógicos: **OR** , **AND** , **XOR** , **NOT**
  - Combinan operandos numéricos para dar un resultado.
  - Ejemplos:

```
MASCARA     DB    4 AND 80
NUM         EQU   20
NUMNEG      EQU   (NOT NUM) +1
```

(2)

2.5. Directivas y operadores del ensamblador del 80x86 (IX)

- Operadores que devuelven valores
  - **$**: Devuelve el desplazamiento (offset) de la instrucción o directiva donde se encuentra. Suele usarse para calcular el tamaño de cadenas de caracteres.
  - **OFFSET** y **SEG**: devuelven el desplazamiento y número de segmento de una variable.
  - Ejemplos:

```
TEXTO          DB    "Hola qué tal"
LONG_TEXTO     DB    $-TEXTO

mov AX, SEG TEXTO
mov DX, OFFSET TEXTO
```

(2)

2.5. Directivas y operadores del ensamblador del 80x86 (X)

- Operadores de atributo
  - **PTR**: Modifica el tipo de datos (BYTE, WORD, DWORD) de un operando.
  - Ejemplos:

```
TABLA          DB    100   dup (0)
...
mov       AL, TABLA[0]
mov       AX, WORD PTR TABLA[0]
```

(2)

2.6. Estructura de un programa en ensamblador (I)

```
;  Segmento de variables globales
datos segment

      ;  Declaración de variables

datos ends
```

```
;  Pueden haber varios segmentos
;  de variables globales
```

```
;  Segmento de pila
pila segment stack "stack"

 ;  Declaración de vector de bytes

pila ends
```

```
;  Segmento de instrucciones
codigo segment
        assume cs:codigo, ds:datos
        assume ss:pila

    ;  procedimiento principal
    inicio proc far

        ;  Instrucciones en ensamblador

    inicio endp
....
        ;  otros procedimientos
....

codigo ends
end inicio
```

(2)

2.6. Estructura de un programa en ensamblador (II)

Ejemplo 1

Innecesarias en este ejemplo

```
datos segment
   tamano    dw 5
   tabla     db "clave"
   tabla2    db "clava"
   resultado db 0
datos ends
```

```
pila segment stack "stack"
          db 64 dup (?)
pila ends
```

```
codigo segment
        assume cs:codigo, ds:datos, ss:pila
inicio proc far
          mov ax, datos
          mov ds, ax
          mov ax, pila
          mov ss, ax
          mov sp, 64
          mov si, 0
          mov resultado, 0
seguir:   mov al, tabla[ si ]
          cmp al, tabla2[ si ]
          jnz distinto
          inc si
          cmp si, tamano
          jnz seguir
          mov resultado, 1
distinto: mov ax, 4C00h
          int 21h
inicio endp

codigo ends
        end inicio
```

(2)

2.6. Estructura de un programa en ensamblador (III)

Ejemplo 2

```
datos segment
   tamano dw 5
   tabla     db "clave"
datos ends
```

```
datos2 segment
   tabla2    db "clava"
datos2 ends
```

```
pila segment stack "stack"
          db 64 dup (?)
pila ends
```

```
codigo segment
          assume cs:codigo, ds:datos, es: datos2, ss:pila
inicio proc far
          jmp lab1
resultado db 0
lab1:     mov ax, datos
          mov ds, ax
          mov ax, datos2
          mov es, ax
          mov ax, pila
          mov ss, ax
          mov sp, 64
          mov resultado, 0
          mov si, 0
seguir: mov al, tabla[ si ]
          cmp al, tabla2[ si ]
          jnz distinto
          inc si
          cmp si, tamano
          jnz seguir
          mov resultado, 1
distinto: mov ax, 4C00h
          int 21h
inicio endp
end inicio
```

Innecesarias en este ejemplo

(2)

2.6. Estructura de un programa en ensamblador (IV)

Ejemplo 3

```
datos segment
        org 100
tabla   db "clave1"
        org 200
tabla2  db "clave2"
datos ends
```

```
datos2 segment
tabla3  dw 3 dup (0)
datos2 ends
```

```
pila segment stack "stack"
        db 64 dup (?)
pila ends
```

```
codigo segment
        assume cs:codigo, ds:datos, es: datos2, ss:pila
inicio proc far
        mov ax, datos
        mov ds, ax
        mov ax, datos2
        mov es, ax
        mov ax, pila
        mov ss, ax
        mov sp, 64
        mov si, offset tabla
        mov di, offset tabla2
        mov cx, 6
seguir: mov al, [ si ]
        cmp al, [ di ]
        jnz distinto
        inc si
        inc di
        dec cx
        jnz seguir
        call guarda
distinto: mov ax, 4C00h
        int 21h
inicio endp
```

```
guarda proc
        mov si, 0
reptir:  mov ax, word ptr tabla[ si ]
        mov tabla3[ si ], ax
        inc si
        inc si
        cmp si, 6
        jnz repetir
        ret
guarda endp


codigo ends
end inicio
```

2.7. Instrucciones del ensamblador (I)

- Tipos de instrucciones básicas
  - Transferencia de datos
  - Operaciones aritméticas
  - Operaciones lógicas
  - Transferencia de control
  - Interrupciones
  - Activación de banderas (flags)

(2)

2.7. Instrucciones del ensamblador (II)

Transferencia de datos

- **MOV**: transfiere datos entre registros o entre registros y posiciones de memoria.

  MOV destino, fuente

- **XCHG**: intercambia el contenido de dos registros o un registro y una posición de memoria.

  XCHG destino, fuente

- **PUSH**: almacena en la pila.

  PUSH fuente

- **POP**: saca de la pila.

  POP fuente

2.7. Instrucciones del ensamblador (III)

Entrada / Salida

- **IN**: lee un dato de un puerto.
  IN acumulador, puerto
- **OUT**: envía un dato a un puerto.
  OUT puerto, acumulador

2.7. Instrucciones del ensamblador (IV)

Transferencia de direcciones

- **LEA**: carga dirección efectiva. Transfiere el offset de una posición de memoria a un registro de 16 bits.

  LEA reg16, mem16

- **LDS**: carga puntero usando DS. Transfiere el contenido de la palabra de memoria especificada al registro indicado y el contenido de la siguiente palabra a DS.

  LDS reg16, mem16

- **LES**: carga puntero usando ES. Transfiere el contenido de la palabra de memoria especificada al registro indicado y el contenido de la siguiente palabra a ES.

  LES reg16, mem16

2.7. Instrucciones del ensamblador (V)

Transferencia del registro de banderas

- **PUSHF**: almacena el registro de banderas en la pila.

  PUSHF

- **POPF**: carga el registro de banderas de la pila.

  POPF

(2)

2.7. Instrucciones del ensamblador (VI)

Operaciones aritméticas

- Operan enteros de 8 o 16 bits.
- Enteros sin signo:
  - **0** a **255** (8 bits), **0** a **65535** (16 bits)
- Enteros con signo:
  - **-128** a **127** (8 bits), **-32768** a **32767** (16 bits)
  - Bit más significativo (signo): **0** (positivo), **1** (negativo)

2.7. Instrucciones del ensamblador (VII)

Operaciones aritméticas

- **ADD**: suma el operando fuente y el destino, dejando el resultado en el destino.

  ADD destino, fuente

- **ADC**: suma al operando fuente el destino y el valor de la bandera de acarreo.

  ADC destino, fuente

- **INC**: incrementa en uno el registro o posición de memoria.

  INC operando

2.7. Instrucciones del ensamblador (VIII)

Operaciones aritméticas

- **SUB**: resta del operando destino el fuente, dejando el resultado en el destino.

  SUB destino, fuente

- **SBB**: resta del operando destino el fuente y el valor de la bandera de acarreo, dejando el resultado en el destino.

  SBB destino, fuente

- **DEC**: decrementa en uno el registro o posición de memoria.

  DEC operando

2.7. Instrucciones del ensamblador (IX)

Operaciones aritméticas

- **MUL**: multiplica el operando por **AX** (operando de 8 bits) o el par **DX**:**AX** (operando de 16 bits).

  MUL operando

- **IMUL**: multiplica con signo.

  IMUL operando

- **DIV**: divide **AX** (operando de 8 bits) o **DX**:**AX** (operando de 16 bits) por el operando sin signo. Cociente en **AL** y resto en **AH** (operando de 8 bits). Cociente en **AX** y resto en **DX** (operando de 16 bits).

  DIV operando

- **IDIV**: divide con signo.

  IDIV operando

2.7. Instrucciones del ensamblador (X)

Operaciones aritméticas

- **NEG**: realiza el complemento a dos de un registro o posición de memoria.

  NEG operando

- **CMP**: Resta del operando destino el fuente sin modificar destino (actualiza banderas).

  CMP destino, fuente

2.7. Instrucciones del ensamblador (XI)

Operaciones lógicas

- **AND**: operación AND entre registros o entre registro y posición de memoria, dejando el resultado en destino.

  AND destino, fuente

- **OR**: operación OR entre registros o entre registro y posición de memoria, dejando el resultado en destino.

  OR destino, fuente

- **XOR**: operación OR EXCLUSIVA entre registros o entre registro y posición de memoria, dejando el resultado en destino.

  XOR destino, fuente

- **NOT**: complemento a uno de un registro o posición de memoria.

  NOT operando

2.7. Instrucciones del ensamblador (XII)

Operaciones lógicas

- **TEST**: operación AND entre operando fuente y destino sin modificar destino (actualiza banderas).

  TEST destino, fuente

- **SAL** / **SHL**: desplazamiento aritmético o lógico a la izquierda.

  SAL operando, 1
  SAL operando, CL

- **SAR**: desplazamiento aritmético a la derecha.

  SAR operando, 1
  SAR operando, CL

- **SHR**: desplazamiento lógico a la derecha.

  SHR operando, 1
  SHR operando, CL

2.7. Instrucciones del ensamblador (XIII)

Operaciones lógicas

- **ROL**: rotación a izquierda.

  ROL operando, 1
  ROL operando, CL

- **ROR**: rotación a derecha.

  ROR operando, 1
  ROR operando, CL

- **RCL**: rotación a izquierda con bandera de acarreo.

  RCL operando, 1
  RCL operando, CL

- **RCR**: rotación a derecha con bandera de acarreo.

  RCR operando, 1
  RCR operando, CL

2.7. Instrucciones del ensamblador (XIV)

Transferencia de control

- **CALL**: Inicia la ejecución de un procedimiento o subrutina. Puede estar en mismo segmento o en otro.

  CALL operando

- **RET**: retorno de un procedimiento o subrutina.

  RET

  RET desplazamiento (retorna y suma desplazamiento a **SP** para descartar parámetros de entrada en la pila)

- **JMP**: salta a la instrucción indicada por el operando.

  JMP operando

2.7. Instrucciones del ensamblador (XV)

Transferencia de control

- **Saltos condicionales**:
  - Saltan a la instrucción indicada por el operando si se cumple la condición de un flag. Siguen con siguiente instrucción si la condición no se cumple.
  - Suelen ejecutarse tras una operación aritmética o lógica (habitualmente tras una comparación con **CMP** o **TEST**).
  - El salto no puede ser nunca mayor de **127** bytes adelante o **128** atrás.

2.7. Instrucciones del ensamblador (XVI)

Transferencia de control (saltos condicionales)

Operación entre enteros con signo		
=	JE/JZ	Z=1
<>	JNE/JNZ	Z=0
>	JG	Z=0 y S=0
>=	JGE	S=0
<	JL	S<>0
<=	JLE	Z=1 o S<>0

Operación entre enteros sin signo		
=	JE/JZ	Z=1
<>	JNE/JNZ	Z=0
>	JA	Z=0 y C=0
>=	JAE	C=0
<	JB	C<>0
<=	JBE	Z=1 o C<>0

JCXZ	CX=0	si CX=0
JO	O=1	si overflow
JNO	O=0	si no overflow
JS	S=1	si signo -
JNS	S=0	si signo +
JC	C=1	si acarreo
JNC	C=0	si no acarreo
JP/JPE	P=1	si paridad par
JNP/JPO	P=0	si paridad impar

G: greater
L: less
E: equal
A: above
B: below

2.7. Instrucciones del ensamblador (XVII)

Interrupciones

- **INT**: ejecuta la rutina de servicio a la interrupción indicada por el número.

  INT número

- **IRET**: retorno de la rutina de servicio.

  IRET

2.7. Instrucciones del ensamblador (XVIII)

Activación de banderas (flags)

- STC: acarreo **C :=** 1.

  STC
- CLC: acarreo **C :=** 0.

  CLC
- CMC: complementa acarreo **C**.

  CMC
- STI: interrupciones **IF :=** 1 (activa interrupciones)

  STI
- CLI: interrupciones **IF :=** 0 (desactiva interrupciones)

  CLI

(2)

2.8. Mapa de memoria del sistema PC

#00000-#0007F	Zona de vectores de interrupción del BIOS
#00080-#000FF	Zona de vectores de interrupción del DOS
#00100-#001FF	Zona de vectores de interrupción de usuario
#00200-#003FF	Zona de vectores de interrupción del BASIC
#00400-#004FF	Área de datos del BIOS
#00500-#005FF	Área de datos del DOS y del BASIC
#00600-#9FFFF	Memoria del usuario para programas
#A0000-#AFFFF	Área de memoria de expansión de pantalla
#B0000-#BFFFF	Área de memoria de pantalla
#C0000-#EFFFF	Extensiones BIOS, no ocupada completamente
#F0000-#FFFFF	Área de ROM BIOS

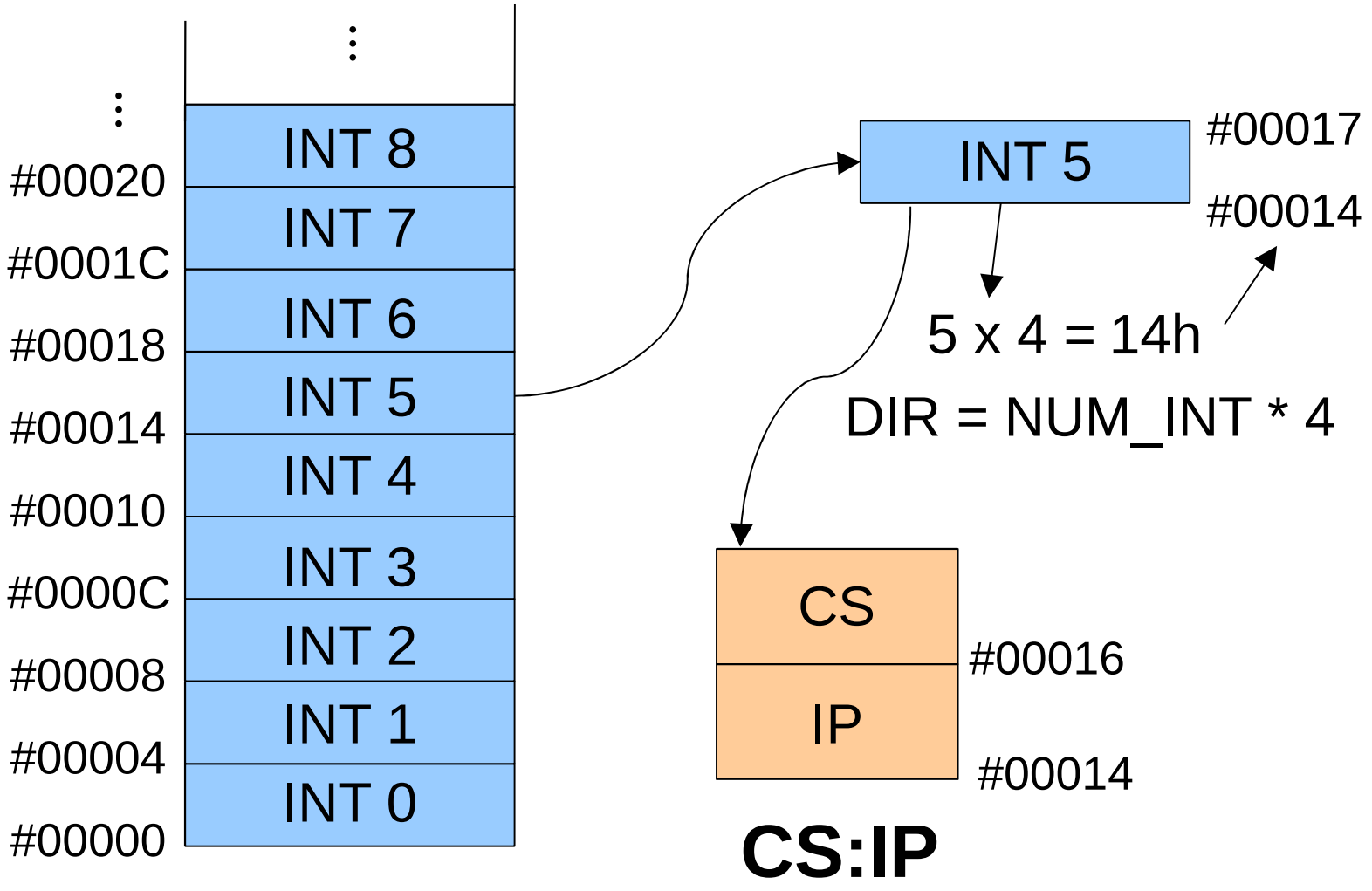
(2)

2.9. Interrupciones: mecanismo y vectores de interrupción (I)

- Las interrupciones son llamadas a rutinas del sistema (normalmente servicios del BIOS o del SO).
- Estas rutinas están “residentes” en memoria.
- Las posiciones de memoria donde empiezan las rutinas se guardan en una tabla en memoria.
- Esta tabla se encuentra al principio de la memoria en DOS: desde la dirección 0 a la 3FFh.
- Cada 4 bytes de esta tabla constituyen un vector de interrupción (offset y segmento donde comienza la rutina de servicio a esa interrupción).

(2)

2.9. Interrupciones: mecanismo y vectores de interrupción (II)



(2)

2.9. Interrupciones: mecanismo y vectores de interrupción (III)

- Instalación de una rutina de servicio a interrupción:

```
DIR     equ    4 * NUM_INT

mov ax, 0
mov es, ax
cli
mov es:[ DIR ], OFFSET rutina_servicio
mov es:[ DIR + 2 ], SEG rutina_servicio
sti
```

(2)

2.9. Interrupciones: mecanismo y vectores de interrupción (IV)

- **Interrupciones software**: Se provocan desde un programa con la instrucción **INT** *n* (*n* entre 0 y 255). No pueden desactivarse (enmascararse).

- **Interrupciones hardware**: Se provocan a través de dos pines del microprocesador: *INTR*, *NMI*.
  - *Enmascarables*:
    - Se activan por hardware poniendo a 1 el pin INTR.
    - Se enmascaran con la bandera **IF** a 0.
    - Los dispositivos indican el número de interrupción en el bus de datos.
  - *No enmascarables*:
    - Se activan por hardware mediante flanco ascendente en el pin NMI.
    - Equivalen a la **INT** 2 de software

(2)

2.9. Interrupciones: mecanismo y vectores de interrupción (V)

- Fases de ejecución de una interrupción por la CPU:
  1. Se apilan banderas y dirección de retorno:
     - Registro de estado (2 bytes)
     - Segmento de código de dirección de retorno.
     - Desplazamiento de dirección de retorno.
  2. Se ponen a 0 bit de interrupción **IF** y de traza **TF** (enmascarando interrupciones hardware y desactivando ejecución paso a paso).
  3. Se lee vector de interrupción (**CS**:**IP**) con dirección de primera instrucción de la rutina de servicio.
  4. Se ejecuta la rutina de servicio.
  5. La rutina de servicio acaba con instrucción **IRET**.
  6. Se desapilan dirección de retorno y estado:
     - **IP** := Desplazamiento @ retorno
     - **CS** := Segmento @ retorno
     - Registro de estado := banderas