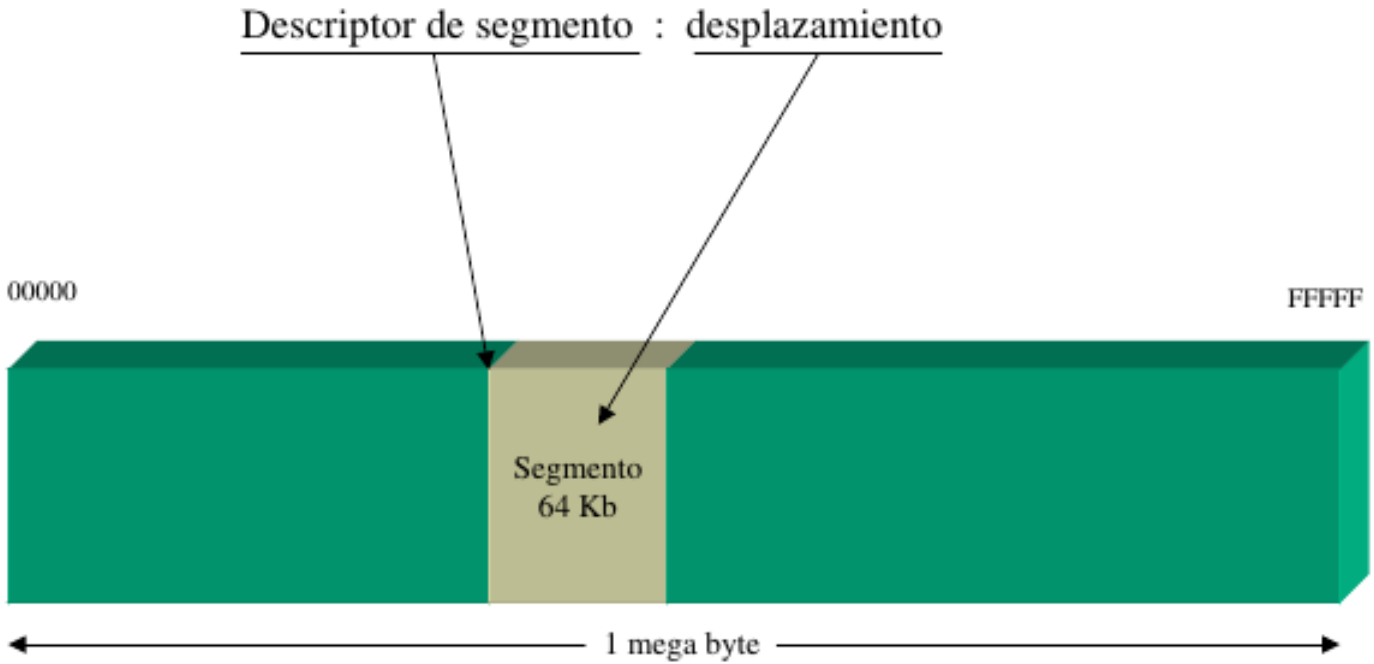
ARQUITECTURA 8086 Y ARM

Ing Marlon Moreno

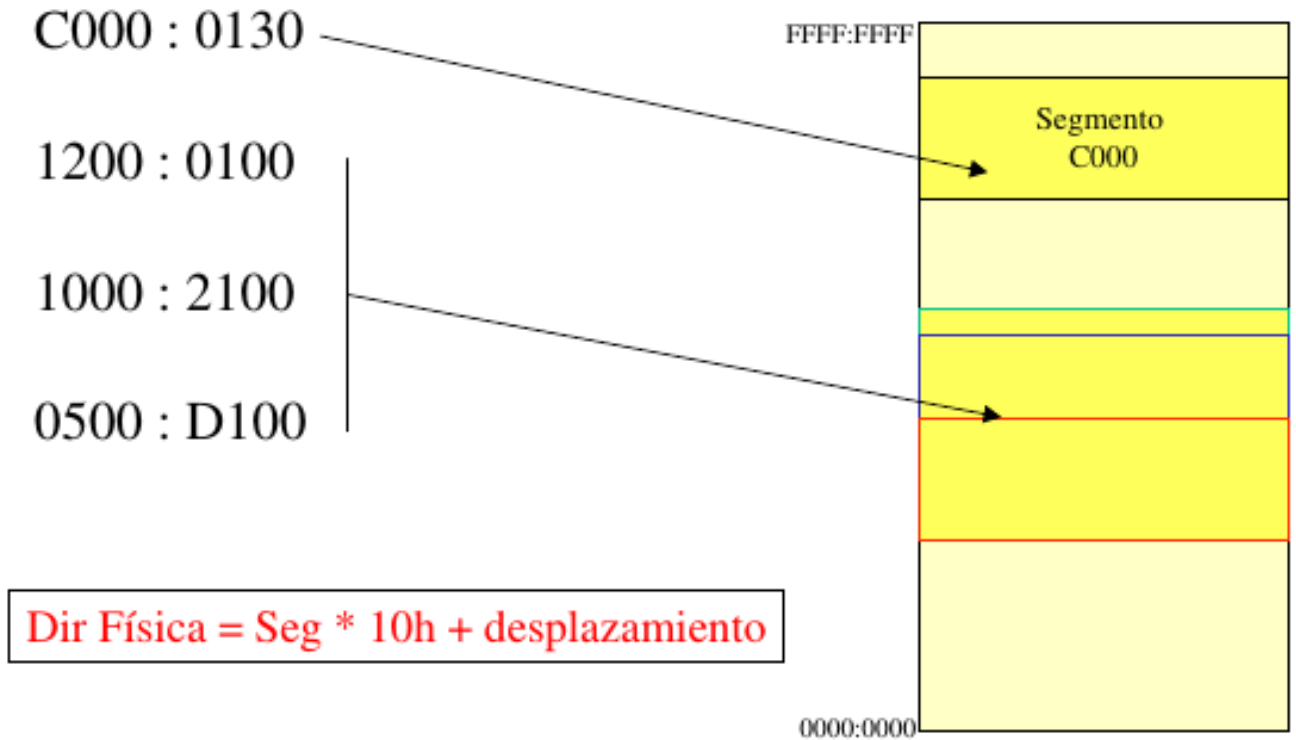
Características generales 8086.

- Procesador 16 bits.
- 20 bits de dirección de memoria, 1 MB.
- 16 bits de datos internos.
- Bus externos de 8 y 16 bits.
- 89 instrucciones.
- No posee cooprocesador.

Memoria.


Descriptor de segmento : desplazamiento
00000
FFFFF
Segmento
64 Kb
1 mega byte

Memoria.


C000 : 0130
1200 : 0100
1000 : 2100
0500 : D100
FFFF:FFFF
Segmento
C000
0000:0000
Dir Física = Seg * 10h + desplazamiento

Representación de datos.

- Bits – unidad.
- Byte – 8 unidades.
- Word – 16 unidades.
- ASCII
- BCD - *binary coded decimal*

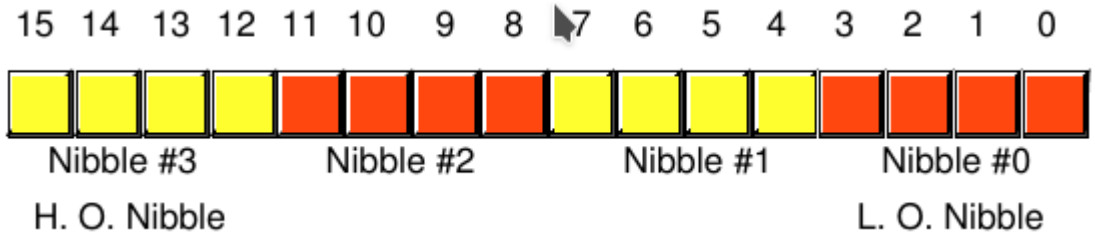
Tipo de datos enteros

Tipo de dato	Bytes
BYTE , **DB** (byte)	1 (de 0 a 255)
SBYTE (byte con signo)	1 (de –128 a +127)
WORD, **DW** (word)	2 (de 0 a 65.535)
SWORD (word con signo)	2 (de –32.768 a +32.767)
DWORD, **DD** (doubleword)	4 (de 0 a 4.294.967.295)
SDWORD (doubleword con signo)	4 (de –2.147.483.648 a +2.147.483.647)

Tipo de datos enteros.

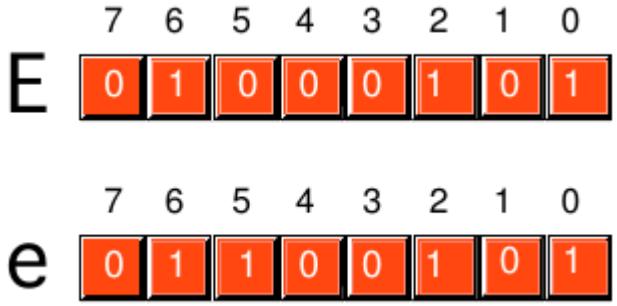
```
entero      BYTE      16          ; inicializa un byte en 16
negativo    SBYTE     -16         ; inicializa un byte con signo en -16
expression  WORD      4*3         ; inicializa un word en 12 a través de la expresión 4*3
positivo    SWORD     4*3         ; inicializa un worq con signo en 12
vacio       DWORD     ?           ; declara una variable dword pero no la inicializa
lista       BYTE      1,2,3,4,5,6 ; inicializa 6 bytes
```

BCD *decimal codificado en binario*.



- Colección de cuatro bits.
- Con un nibble se puede representar 16 valores diferentes, pero en BCB se toman los 10 primeros valores.

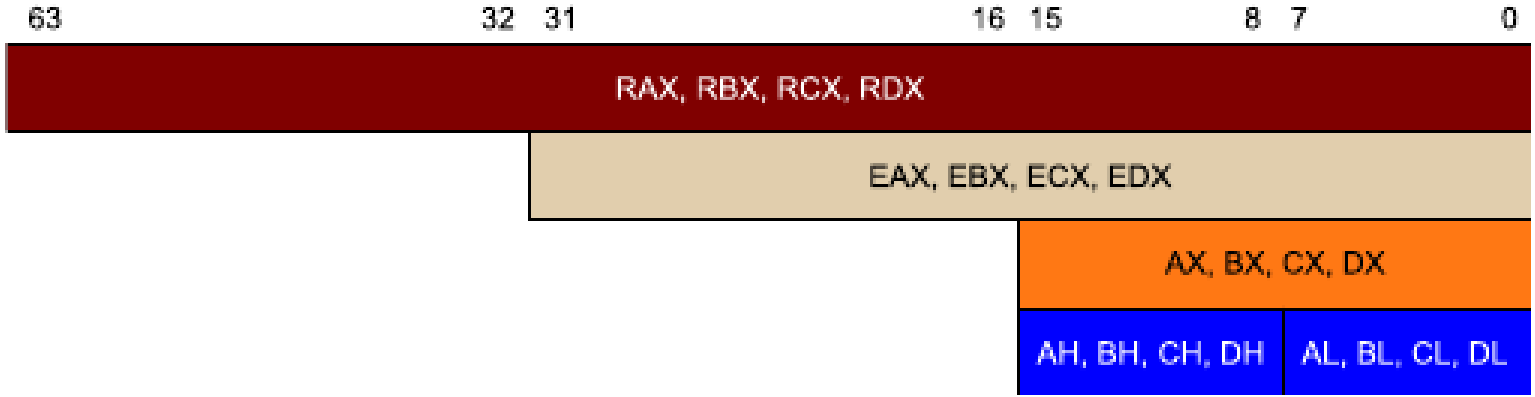
ASCII.



- *Código Estándar Estadounidense para el Intercambio de Información. Creado de 1963 por el comité estadounidense de estándares*
- Código de caracteres del alfabeto latino, utiliza 7 bits.

Registros de proposito general.

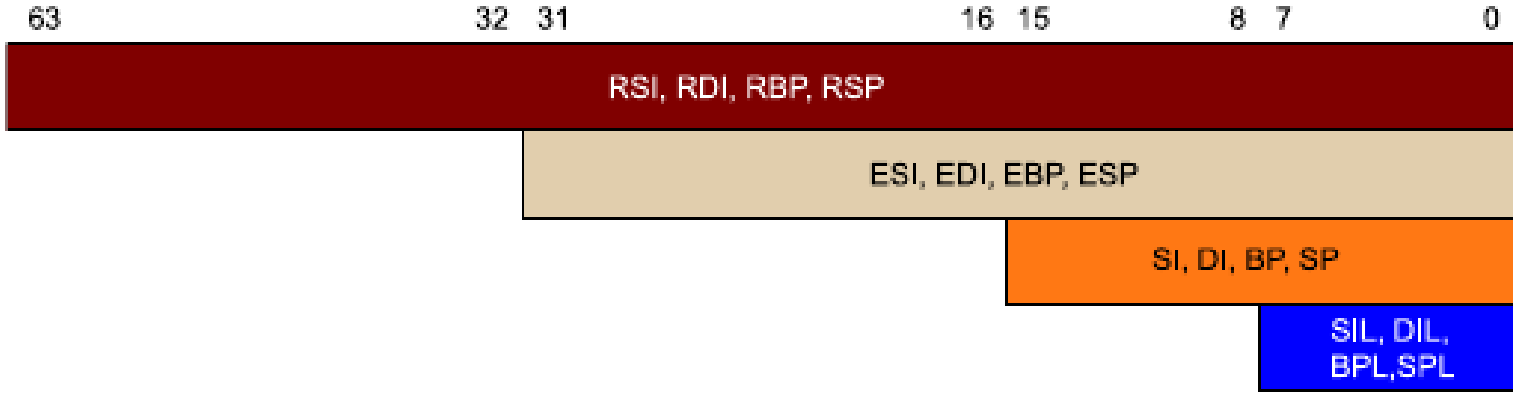
$$AX = Registro\ acumulador\ .$$
$$BX = Registro\ base\ de\ dirección\ .$$
$$CX = Registro\ de\ conteo\ .$$
$$DX = Registro\ de\ datos\ .$$



Registros apuntadores.

$$SP = Apuntador\ de\ pila\ .$$

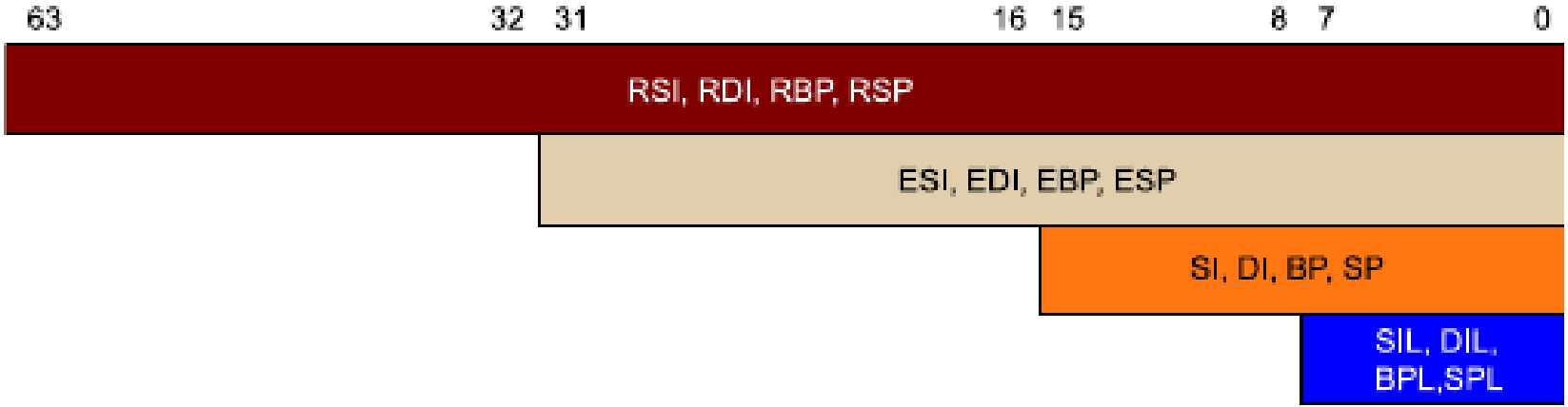
$$BP = Apuntador\ base\ .$$



Registros índices.

$$SI = Índice\ fuente\ .$$

$$DI = Índice\ destino\ .$$



Registros de segmento.

$$CS = Segmento\ de\ código\ .$$
$$DS = Segmento\ de\ datos\ .$$
$$ES = Segmento\ extra\ .$$
$$SS = Segmento\ de\ pila\ .$$

Registro apuntador de instrucciones.

- IP (Apuntador de instrucciones) Este registro contiene la dirección de desplazamiento del lugar de memoria donde está la siguiente instrucción que será ejecutada por el microprocesador.

Registro de estado o Flags register.

- CF (Acarreo) Esta bandera indica el acarreo o préstamo después de una suma o resta.
- OF (Sobre flujo) Esta bandera indica cuando el resultado de una suma o resta de números con signo sobrepasa la capacidad de almacenamiento de los registros.
- SF (Signo) Esta bandera indica si el resultado de una operación es positivo o negativo. SF=0 es positivo, SF=1 es negativo.
- DF (Dirección) Indica el sentido en el que los datos serán transferidos en operaciones de manipulación de cadenas. DF=1 es de derecha a izquierda y DF=0 es de izquierda a derecha.

Registro de estado o Flags register.

- ZF (Cero) Indica si el resultado de una operación aritmética o lógica fue cero o diferente de cero. ZF=0 es diferente y ZF=1 es cero.
- IF (interrupción) Activa y desactiva la terminal INTR del microprocesador.
- PF (paridad) Indica la paridad de un número. Si PF=0 la paridad es impar y si PF=1 la paridad es par.
- AF (Acarreo auxiliar) Indica si después de una operación de suma o resta ha ocurrido un acarreo de los bits 3 al 4.
- TF (Trampa) Esta bandera controla la ejecución paso por paso de un programa con fines de depuración.

Ejecución de un programa.

1. Extrae de la memoria la instrucción que va a ejecutar y la coloca en el registro interno de instrucciones.
2. Cambia el registro apuntador de instrucciones (IP) de modo que señale a la siguiente instrucción del programa.
3. Determina el tipo de instrucción que acaba de extraer.
4. Verifica si la instrucción requiere datos de la memoria y, si es así, determina donde están situados.
5. Extrae los datos, si los hay, y los carga en los registros internos del CPU.
6. Ejecuta la instrucción.
7. Almacena los resultados en el lugar apropiado.
8. Regresa al paso 1 para ejecutar la instrucción siguiente.

Modo de direccionamiento

- **Modo registro:** El operando es un registro.
- **Modo inmediato:** El operando es una constante.
- **Modo directo: E**l operando es una dirección efectiva (explícita).
- **Modo registro indirecto:** similar al anterior pero la dirección efectiva está contenida en un registro (BX, BP, SI , DI).

Modo de direccionamiento

- **Modo relativo a base:** La dirección efectiva se encuentra sumando un desplazamiento a BX o BP.
- **Modo indexado directo:** Igual al anterior usando SI o DI.
- **Modo indexado a base:** combinación de los dos anteriores. La dirección efectiva se calcula como la suma de un registro base, un registro índice y opcionalmente un desplazamiento.

Modelos de memoria.

- tinty: Código + datos <=64 k, .com
- Smal: Código <=64k, datos <=64k.
- medium: Datos <=64 k
- compact: Código <=64 k.
- large: Múltiples segmentos de datos y código
- huge: Permite que las matrices excedan los 64 k.
- flat: No hay segmentos, se usan 32 bits de direcciones

Modelos de memoria.

Category	Name	Description	Return result
Segment [a] Information	@code	Returns the name of the current code segment.	Text value
	@data	Returns the name of the current data segment.	Text value
	@FarData?	Returns the name of the current far data segment.	Text value
	@Word-Size	Returns two if this is a 16 bit segment, four if this is a 32 bit segment.	Numeric value
	@Code-Size	Returns zero for Tiny, Small, Compact, and Flat models. Returns one for Medium, Large, and Huge models.	Numeric value
	@DataSize	Returns zero for Tiny, Small, Medium, and Flat memory models. Returns one for Compact and Large models. Returns two for Huge model programs.	Numeric value
	@Model	Returns one for Tiny model, two for Small model, three for Compact model, four for Medium model, five for Large model, six for Huge model, and seven for Flag model.	Numeric value
	@CurSeg	Returns the name of the current code segment.	Text value
	@stack	The name of the current stack segment.	Text value

Conceptos de programación.

- Sentencia ***if.***
- *Lenguaje* ***C.***

  if ((a != b) || (a>=1 && a<=5)) {b = a;}
- **ASM.**

```
    mov rax, qword [a]  ;Se cargan las variables en registros
    mov rbx, qword [b]
    cmp rax, rbx            ;Se hace la comparación (a != b)
    jne cierto                    ;Si se cumple la condición salta a la etiqueta cierto
                                        ;Si no se cumple, como es una OR, se puede cumplir la otra condición.
    cmp rax, 1                    ;Se hace la comparación (a >= 1), si no se cumple,
    jl fin                  ;como es una AND, no hay que mirar la otra condición.
     cmp rax, 5                   ; Se hace la comparación (a <= 5)
     jg fin                  ;Si no salta, se cumple que (a>=1 && a<=5)
  cierto:
     mov qword [b], rax             ;Hacemos la asignación cuando la condición es cierta.
  end:
```

Conceptos de programación.

- Sentencia ***if.***
- *Lenguaje* ***C.***

  if (a > b) { maxA = 1; maxB = 0; }
- **ASM**

```
    mov ax, qword [a] ;Se cargan las variables en registros
    mov bx, qword [b]
    cmp ax, bx              ;Se hace la comparación
    jg cierto               ;Si se cumple la condición, salta a la etiqueta cierto
    jmp fin                 ;Si no se cumple la condición, salta a la etiqueta fin
cierto:
    mov byte [maxA], 1      ;Estas instrucciones solo se ejecutan
    mov byte [maxB], 0      ;cuando se cumple la condición
fin:
```

Conceptos de programación.

- Sentencia ***if - else.***
- *Lenguaje* ***C.***

```
if (a > b) {
    max = 'a';
}
else { // (a <= b)
    max = 'b';
}
```

Conceptos de programación.

- Sentencia ***if - else.***
- ***ASM***

```
    mov rax, qword [a]          ;Se cargan las variables en registros
    mov rbx, qword [b]
    cmp rax, rbx                ;Se hace la comparación
    jg cierto                   ;Si se cumple la condición, se salta a cierto
    mov byte [max], 'b'         ;else (a <= b)
    jmp fin
cierto:
    mov byte [max], 'a'         ;if (a > b)
fin:
```

Conceptos de programación.

- Sentencia ***for.***
- *Lenguaje* ***C.***

```
resultado=1;
for (i = num; i > 1; i--)
{
    resultado=resultado*i;
}
```

Conceptos de programación.

- Sentencia ***for***
- ***ASM.***

```
    mov ax, 1                ;ax será [resultado]
    mov cx, qword [num]      ;cx será [i] que inicializamos con [num]
for:
    cmp cx, 1                ;Se hace la comparación
    jg cierto                ;Si se cumple la condición, salta a cierto
    jmp fin
cierto:
    imul ax,cx
    dec cx
    jmp for
fin:
    mov qword [resultado], ax
    mov qword [i], rcx
```