

# Lenguaje máquina (x86)

---

## Historia

### IBM - i8086

IBM se hizo con el mercado con el procesador de **16 bits** llamado i8086.

Era de números enteros, **no podías hacer operaciones aritméticas con decimales** (solo podías simularlas).

Hizo un **chip aritmético** para aumentar la velocidad de computación.

### IBM - i80286

Aumenta el espacio de direcciones a **24 bits**.

### IBM - i80386 (el de las prácticas)

Se extiende a **32 bits**. Todos los registros sirven para todo.

Recordad que en las prácticas hay que poner la opción -m32 para compilar con 32 bits.

### Intel - Pentium MMX (1997)

Intel hizo un conjunto de instrucciones (**instrucciones multimedia**) para procesamiento vectorial de datos. Es decir, con una sola operación se podían modificar vectores.

### AMD (2003)

Extiende a **64 bits**.

## Instrucciones

Lenguaje máquina x86 que usaremos es CISC.

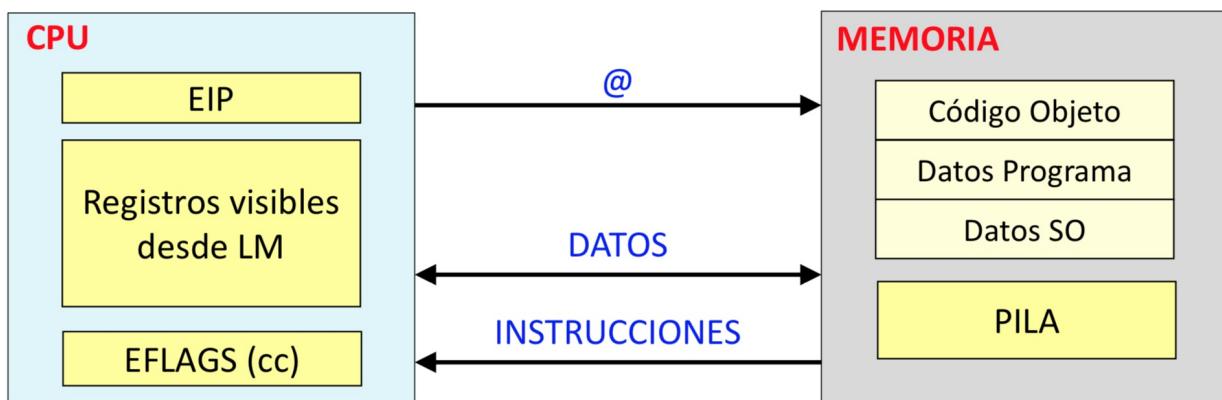
Por lo tanto hay diferencias:

### Lenguaje Máquina MIPS (características RISC)

- Instrucciones aritméticas acceden sólo a registros
  - En algunos casos un operando puede ser inmediato
- Solo las instrucciones Load y Store pueden acceder a memoria.
- Referencias a memoria con modos de direccionamiento simples
  - Base + Desplazamiento
- Instrucciones de longitud fija
- Muchos registros

### Lenguaje Máquina x86 (características CISC)

- Instrucciones pueden referenciar diferentes tipos de operandos
  - inmediato, registros, memoria
- Instrucciones aritméticas pueden leer/escribir en memoria, pero sólo 1 de los 2 operandos puede estar en memoria
- Referencias a memoria pueden suponer cálculos complejos
  - $Rb + S \cdot Ri + D$
- Instrucciones pueden tener diferente longitud
- Pocos registros



## Tipos de datos

- Enteros
- Reales (con coma flotante)

Cada tipo de dato tiene un rango de número que caben dentro (ambos incluidos):

- Naturales:
  - byte: 0 .. 255
  - word: 0 .. 65.535
  - long: 0 .. 4.294.967.215
- Enteros:
  - byte: -128 .. 127
  - word: -32.768 .. 32.767
  - long: -2.147.483.648 .. 2.147.483.647
- Reales:
  - byte (32 bits):  $1,18 \cdot 10^{-38} .. 3,40 \cdot 10^{38}$

- word (64 bits):  $2,23 \times 10^{-308} \dots 1,79 \times 10^{308}$
  - long (80 bits):  $3,37 \times 10^{-4932} \dots 1,18 \times 10^{-4932}$

MOV[ B (8 bits)| L (32 bits)|W (16 bits) ]

No hay structs, ni matrices, ni ningún tipo estructurado

# Registros

32 bits	16 bits	8 bits	
%eax	%ax	%ah , %al	
%ebx	%bx	%bh , %bl	
%ecx	%cx	%ch , %cl	
%edx	%dx	%dh , %dl	
%esi	%si		
%edi	%di		
%esp	%sp		Reservados para uso específico de subrutinas
%ebp	%bp		
%eip			Contador programa
%eflags			Palabra de estado

// TO-DO: revisar (arriba)

Son registros little endian

0	34	byte 8:	0x21
1	22	byte 3:	0x3B
2	5A	word 8:	0x2A21
3	3B	word 3:	0xC13B
4	C1	longword 8 :	0x7B2C2A21
5	45	longword 3 :	0xFF45C13B
6	FF	quadword 8 :	0xFF1143907B2C2A21
7	00	quadword 3 :	0x2C2A2100FF45C13B
8	21		
9	2A		
10	2C		
11	7B		
12	90		
13	43		
14	11		
15	FF		

**i Little Endian !**

0	34	byte 8:	0x21
1	22	byte 3:	0x3B
2	5A	word 8:	0x212A
3	3B	word 3:	0x3BC1
4	C1	longword 8 :	0x212A2C7B
5	45	longword 3 :	0x3BC145FF
6	FF	quadword 8 :	0x212A2C7B904311F
7	00	quadword 3 :	0x3BC145FF00212A2
8	21		
9	2A		
10	2C		
11	7B		
12	90		
13	43		
14	11		
15	FF		

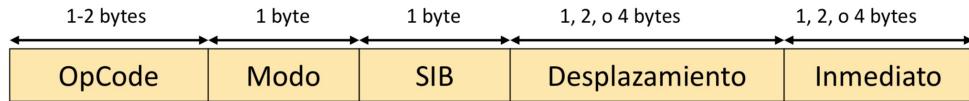
**i Big Endian !**

## Ejemplos de instrucciones

```
// Inmediatos  
$19  
$-3  
$0x2A  
$0x2A45
```

```
// Registros
%eax
%ah
%esi

// Memoria: D(Rb, Ri, s) -> M[Rb + Ri * s + D]
a(b,c,d) = a + b + c * d
```

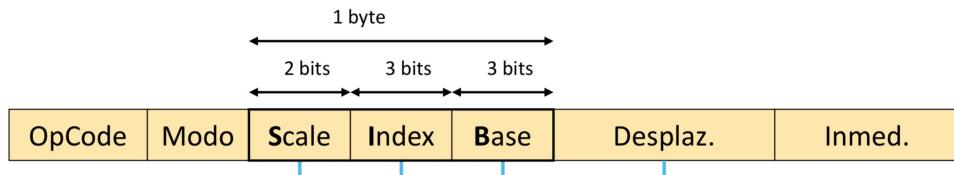


**MOVL \$37, -40(%ebp,%esi, 4)**

## Codificación de memoria

(es bastante jodido)

// TO-DO: explicarlo



## INSTRUCCIONES DE MOVIMIENTO

Instrucciones	Descripción	Notas	Ejemplo
<b>MOVx op1, op2</b>	$op2 \leftarrow op1$	$x = \{L, W, B\}$	MOVB \$-1,%AL
<b>MOV\$xy op1, op2</b>	$op2 \leftarrow \text{ExtSign}(op1)$	$xy = \{BW, BL, WL\}$	MOVSBW %CH,%AX
<b>MOVZxy op1, op2</b>	$op2 \leftarrow \text{ExtZero}(op1)$	$xy = \{BW, BL, WL\}$	MOVZWL %BX,%EDX
<b>PUSHL op1</b>	$\%ESP \leftarrow \%ESP - 4;$ $M[\%ESP] \leftarrow op1$		PUSHL 12(%EBP)
<b>POPL op1</b>	$op1 \leftarrow M[\%ESP];$ $\%ESP \leftarrow \%ESP + 4;$		POPL %EAX
<b>LEAL op1, op2</b>	$op2 \leftarrow \&op1$	op1: memoria	LEAL (%EBX,%ECX),%EAX

En el manual de Microsoft va al revés (MOVx op2, op1)

**MOV\$xy** --> Extiende el signo.

**MOVZxy** --> Pone ceros.

Si se te olvida la -m32 el push y el pop se vuelven locos.

**LEAL** --> Mueve la dirección al registro de destino.

En todos se puede mover de op1 a op1

## INSTRUCCIONES ARITMÉTICAS

Instrucciones	Descripción	Notas	Ejemplo
<b>ADDx op1, op2</b>	$op2 \leftarrow op2 + op1$	$x = \{L, W, B\}$	ADDL \$13,%EAX
<b>SUBx op1, op2</b>	$op2 \leftarrow op2 - op1$	$x = \{L, W, B\}$	SUBW %CX,%AX
<b>ADCx op1, op2</b>	$op2 \leftarrow op2 + op1 + CF$	$x = \{L, W, B\}$	ADCL %EDX,%EAX
<b>SBBx op1, op2</b>	$op2 \leftarrow op2 - op1 - CF$	$x = \{L, W, B\}$	SBBL %ECX,%EAX
<b>INCx op1</b>	$op1 \leftarrow op1 + 1$	$x = \{L, W, B\}$	INCL %EAX
<b>DECx op1</b>	$op1 \leftarrow op1 - 1$	$x = \{L, W, B\}$	DECW %BX
<b>NEGx op1</b>	$op1 \leftarrow -op1$	$x = \{L, W, B\}$	NEGL %EAX

Instrucciones	Descripción	Notas	Ejemplo
<b>IMUL op1, op2</b>	$op2 \leftarrow op2 \cdot op1$	op2: registro	IMUL (%EBX),%EAX
<b>IMUL inm,op1,op2</b>	$op2 \leftarrow op1 \cdot inm$	inm: constante	IMUL \$3,%EAX,%ECX
<b>IMULL op1</b>	$\%EDX\%EAX \leftarrow op1\cdot\%EAX$	op1: mem. o reg. <b>(Enteros)</b>	IMULL (%EBX)
<b>MULL op1</b>	$\%EDX\%EAX \leftarrow op1\cdot\%EAX$	op1: mem. o reg. <b>(Naturales)</b>	MULL (%EBX)
<b>CLTD</b>	$\%EDX\%EAX \leftarrow ExtSign(\%EAX)$		CLTD
<b>IDIVL op1</b>	$\%EAX \leftarrow \%EDX\%EAX / op1$ $\%EDX \leftarrow \%EDX\%EAX \% op1$	op1: mem. o reg. <b>(Enteros)</b>	IDIVL (%EBX)
<b>DIVL op1</b>	$\%EAX \leftarrow \%EDX\%EAX / op1$ $\%EDX \leftarrow \%EDX\%EAX \% op1$	op1: mem. o reg. <b>(Naturales)</b>	DIVL %ESI

**IMULL** --> Sé que el resultado puede tener más de 32 bits.

**IMUL** --> Sé que el resultado no tiene más de 32 bits.

**MULL** --> Multiplica el %eax por el op1 que puede tener 64 bits.

**CLTD** --> Convierte 32 bits (%eax) en 64 bits (%edx %eax).

**IDIVL** --> Divide enteros.

**DIVL** --> Divide naturales.

## INSTRUCCIONES LÓGICAS

Instrucciones	Descripción	Notas	Ejemplo
<b>ANDx op1, op2</b>	$op2 \leftarrow op2 \& op1$	$x = \{L, W, B\}$	ANDL \$13,%EAX
<b>ORx op1, op2</b>	$op2 \leftarrow op2   op1$	$x = \{L, W, B\}$	ORW %CX,%AX
<b>XORx op1, op2</b>	$op2 \leftarrow op2 ^ op1$	$x = \{L, W, B\}$	XORL %EDX,%EAX
<b>NOTx op1</b>	$op1 \leftarrow \sim op1$	$x = \{L, W, B\}$	NOTB %AH
<b>SALx k,op1</b>	$op1 \leftarrow op1 << k$ (aritm.)	$x = \{L, W, B\}$ , k: inm. o %CL	SALL \$1,%EAX
<b>SHLx k,op1</b>	$op1 \leftarrow op1 << k$ (log.)	$x = \{L, W, B\}$ , k: inm. o %CL	SHLW %CL,%DX
<b>SARx k,op1</b>	$op1 \leftarrow op1 >> k$ (aritm.)	$x = \{L, W, B\}$ , k: inm. o %CL	SARL \$1,%EAX
<b>SHRx k,op1</b>	$op1 \leftarrow op1 >> k$ (log.)	$x = \{L, W, B\}$ , k: inm. o %CL	SHRW %CL,%DX
<b>CMPx op1, op2</b>	$op2 - op1$	$x = \{L, W, B\}$ , activa flags	CMPL \$13,%EAX
<b>TESTx op1, op2</b>	$op2 \& op1$	$x = \{L, W, B\}$ , activa flags	TESTW %CX,%AX

## INSTRUCCIONES DE SALTO

Instrucciones	Descripción	Notas	Ejemplo
<b>JMP etiq</b>	$EIP \leftarrow EIP + \text{despl.}$	$EIP \leftarrow \&etiq$	JMP loop
<b>JMP op</b>	$EIP \leftarrow op$	op: reg. o mem.	JMP (%ebx,%esi,4)
<b>Jcc etiq</b>	if (cc) $EIP \leftarrow EIP + \text{despl.}$	cc = {E, NE, G, GE, L, LE, ...} ( <b>Z</b> )	JLE else
<b>Jcc etiq</b>	if (cc) $EIP \leftarrow EIP + \text{despl.}$	cc = {A, AE, B, BE, ...} ( <b>N</b> )	JA loop
<b>Jcc etiq</b>	if (cc) $EIP \leftarrow EIP + \text{despl.}$	cc = {Z, NZ, C, NC, O, ...} (flags)	JNC error
<b>CALL etiq</b>	$\%ESP \leftarrow \%ESP - 4$ $M[\%ESP] \leftarrow EIP$ $EIP \leftarrow EIP + \text{despl.}$	Guardar @retorno $EIP \leftarrow \&etiq$	CALL sub
<b>CALL op</b>	$\%ESP \leftarrow \%ESP - 4$ $M[\%ESP] \leftarrow EIP$ $EIP \leftarrow op$	op: reg. o mem.	CALL (%EBX)
<b>RET</b>	$EIP \leftarrow M[\%ESP];$ $\%ESP \leftarrow \%ESP + 4$		RET

## FLAGS DE CONDICIONES

Instrucciones	Flags	Descripción
<b>JE etiq</b>	ZF	Igual / cero
<b>JNE etiq</b>	$\sim ZF$	No igual / no cero
<b>JS etiq</b>	SF	Negativo
<b>JNS etiq</b>	$\sim SF$	No negativo
<b>JG etiq</b>	$\sim(SF \wedge OF) \& \sim ZF$	Mayor (con signo)
<b>JGE etiq</b>	$\sim(SF \wedge OF)$	Mayor o igual (con signo)
<b>JL etiq</b>	$(SF \wedge OF)$	Menor (con signo)
<b>JLE etic</b>	$(SF \wedge OF) \mid ZF$	Menor o igual (con signo)
<b>JA etiq</b>	$\sim CF \& \sim ZF$	Mayor (sin signo)
<b>JAE etiq</b>	$\sim CF$	Mayor o igual (sin signo)
<b>JB etiq</b>	CF	Menor (sin signo)
<b>JBE etiq</b>	$CF \wedge ZF$	Menor o igual (sin signo)

## Ejemplos

Instrucciones	OF	DF	IF	TF	SF	ZF	AF	PF	CF
<b>ADD op1, op2</b>	X				X	X	X	X	X
<b>AND op1, op2</b>	0				X	X	?	X	0
<b>DEC op1</b>	X				X	X	X	X	
<b>NOT op1</b>									
<b>STC</b>									1
<b>MOV op1, op2</b>									
<b>MUL op1</b>	X				?	?	?	?	X
<b>SAL k,op1</b>	i				X	X	?	X	X
<b>LEAL op1, op2</b>									

Significado	
X	Depende resultado
0	cero
1	uno
?	No definido
	No modificado
i	Consultar manual

```
.data           # el código empieza aquí
    .align 4      # empieza en una dirección múltiple de 4
    .string "Esto es un string.\n"
.text
    .align 4      # empieza en una dirección múltiple de 4
    .globl main    # hace que empiece haciendo el main
    .type main, @function
```

```
main:    ...
        ...
```

## Traducción C --> Assembly

El profesor se ha saltado los ifs, whiles, etc: "Ya lo habéis hecho en IC/EC."

Los punteros son de 4 bytes. Ya que nuestra máquina es de 32 bits.

TIPO	sizeof(TIPO)
char	1 byte
char*	4 bytes
int	4 bytes
int*	4 bytes
double	8 bytes

## Vectores

```
int V[8];
```

**V[i]** --> @V + (i) \* sizeof(int)

```
int Vi(int V[100], int i) {
    return V[i];
}
```

### Traducción:

```
Vi: pushl %ebp
    movl %esp, %ebp
    movl 8(%ebp), %ecx      # @V → 8[%ebp]
    movl 12(%ebp), %edx     # i → 12[%ebp]
    movl (%ecx,%edx,4), %eax
    popl %ebp                # resultado en %eax
    ret
```

## Matrices

```
int mat[10][10];
```

```
mat[i][j] --> @mat + (Ncolumnas * i + j) * sizeof(int)
```

```
int mfc(int mat[50][80], int fil, int col) {  
    return mat[fil][col];  
}
```

### Traducción:

```
Mfc: pushl %ebp  
      movl %esp, %ebp  
      movl 8(%ebp),%ecx  
      imull $80,12(%ebp),%eax # fil → 12[%ebp]  
      addl 16(%ebp),%eax      # col → 16[%ebp]  
      movl 8(%ebp),%ecx      # @M → 8[%ebp]  
      movl (%ecx,%eax,4),%eax  
      popl %ebp              # resultado en %eax  
      ret
```