

Arquitectura *software* x86-16bits

Índice

1. Organización interna
2. Segmentación de memoria
3. Modos de direccionamiento
4. Formato de instrucciones

Arquitectura *software* x86-16bits

¿Qué es arquitectura *software*?

- Es el conjunto de atributos que ve el programador
 - ➔ ...o que ve el compilador, como generador automático de código
- Es el “contrato” entre el fabricante del procesador y el programador
- No requiere correspondencia real con la capa física
- La arquitectura *software* x86-16bits data de 1978
 - ➔ Típico repertorio CISC

Arquitectura *software* x86-16bits

1. Organización interna

	15	0	PROPÓSITO GENERAL
Acumulador	AH	AX AL	Producto, división, E/S y transferencias optimizadas
Base	BH	BX BL	Puntero a dirección base (datos)
Contador	CH	CX CL	Contador de bucles, desplazamientos, rotaciones y repetición de cadenas
Datos	DH	DX DL	Producto, división y E/S
Puntero base	BP		Puntero a dirección base (pila)
Índice fuente	SI		Fuente en manejo de cadenas
Índice destino	DI		Destino en manejo de cadenas
Puntero de pila	SP		Puntero a la cima de la pila

© Rafael Rico López

3/47

Arquitectura *software* x86-16bits

1. Organización interna

Registros de propósito general:

- Almacenamiento temporal de datos
- Algunos se acceden como palabra (16 bits) o como byte
 - ➔ Registro X
 - ➔ Registro L y H
- Usos dedicados y limitaciones
- Modelo híbrido (acumulador y banco de registros)
 - ➔ Heredado de arquitecturas anteriores

© Rafael Rico López

4/47

Arquitectura *software* x86-16bits

1. Organización interna

	15	0
Registro de segmento	CS	
Segmento de código		
Segmento de datos	DS	
Segmento de pila	SS	
Segmento extra (datos)	ES	

- Los programas manejan diferentes áreas de memoria:
 - Código (texto)
 - Datos
 - Pila
 - *Heap*
- Estos registros apuntan a dichas áreas de memoria

© Rafael Rico López

5/47

Arquitectura *software* x86-16bits

1. Organización interna

Registros de segmento:

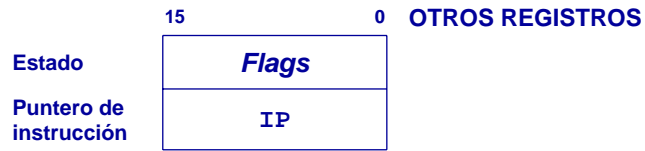
- Todas las direcciones son relativas a alguno de los registros de segmento (por defecto)
- **CS**: la memoria de este segmento contiene instrucciones
- **DS**: datos declarados por el programa
- **SS**: es la pila
- **ES**: datos extra; datos de cadenas

© Rafael Rico López

6/47

Arquitectura *software* x86-16bits

1. Organización interna

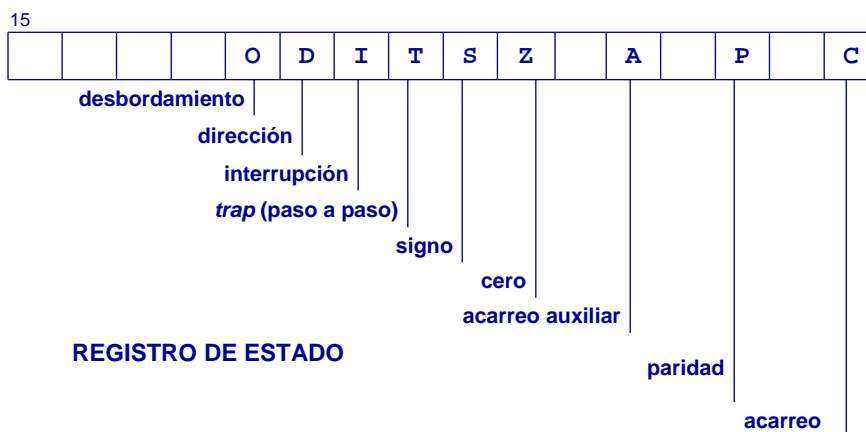


© Rafael Rico López

7/47

Arquitectura *software* x86-16bits

1. Organización interna



© Rafael Rico López

8/47

Arquitectura *software* x86-16bits

1. Organización interna

- **Acarreo:**
 - ➔ Es 1 si el resultado de una operación genera acarreo
- **Paridad:**
 - ➔ Es 1 si el resultado contiene un número par de bits a 1
- **Acarreo auxiliar:**
 - ➔ Es 1 si el resultado genera un acarreo en los 4 bits de menor peso. Se usa en aritmética BCD
- **Cero:**
 - ➔ Es 1 si el resultado es cero
- **Signo:**
 - ➔ Copia el bit de mayor peso del resultado
 - ➔ (independientemente de la interpretación del resultado)

Arquitectura *software* x86-16bits

1. Organización interna

- **Trap:**
 - ➔ Si es 1 el procesador genera una interrupción de paso a paso después de ejecutar cada instrucción
- **Interrupción:**
 - ➔ Si es 1 las interrupciones serán reconocidas
- **Dirección:**
 - ➔ Si es 1 las operaciones con cadenas se realizan de las posiciones altas a las bajas

Arquitectura *software* x86-16bits

1. Organización interna

- **Desbordamiento:**

- ➔ Es 1 si el resultado es demasiado grande (o pequeño) para los límites de representación con signo (en C-2)

$$OF = c_{n-1} \oplus c_{n-2}$$

Arquitectura *software* x86-16bits

1. Organización interna

- **Flags de estado**

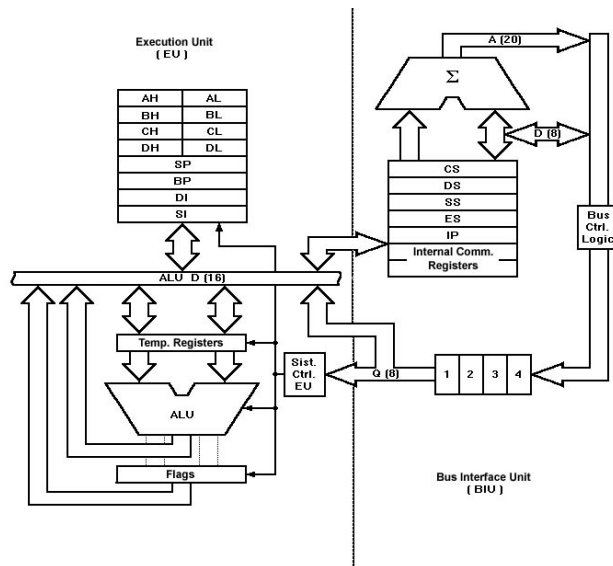
- ➔ **Acarreo, paridad, acarreo auxiliar, cero, signo y desbordamiento**
 - ➔ Describen el resultado
 - ➔ Se aplican a operaciones de tamaño byte ($n=8$) –ya sea la parte alta o la baja– o palabra ($n=16$)
 - ➔ Proporcionan códigos de condición por separado o en conjunto

- **Flags de control**

- ➔ **Trap, interrupción y dirección**
 - ➔ Afectan al modo de funcionamiento del procesador

Arquitectura software x86-16bits

1. Organización interna



13/47

Arquitectura software x86-16bits

2. Segmentación de memoria

- El mapa de memoria que “ve” el x86-16bits NO es “plano”
- Los registros de 16 bits sólo pueden direccionar 2^{16} posiciones de memoria (65.535 posiciones = 64K)
- ¿Cómo alcanzar un espacio de direcciones mayor?
- El acceso a memoria se realiza en segmentos de 64K

© Rafael Rico López

14/47

Arquitectura *software* x86-16bits

2. Segmentación de memoria

- La memoria está organizada en bytes (B)
- La capacidad de direccionamiento es de 1MB (2^{20} B)
- La dirección completa se consigue mediante la combinación de dos punteros: base y desplazamiento (*offset*):

base:desplazamiento

- La dirección física se calcula:

base x 16 + desplazamiento

© Rafael Rico López

15/47

Arquitectura *software* x86-16bits

2. Segmentación de memoria

Ejemplos:

- La dirección 53C2:107A es:

53C20 h	x16 (x10 h)
<u>+107A h</u>	
54C9A h	dirección física

- La dirección B100:046C es:

B1000 h	x16 (x10 h)
<u>+046C h</u>	
B146C h	dirección física

© Rafael Rico López

16/47

Arquitectura *software* x86-16bits

2. Segmentación de memoria

- La misma dirección física puede ser accedida con diferentes combinaciones *base:desplazamiento*

Ejemplo:

- La dirección física 7A26B h puede ser:
 - ➔ 7A26:000B
 - ➔ 7A00:026B
 - ➔ 751C:50AB
- Problemas de seguridad
- La base (segmento) apunta a párrafos (de 16 posiciones)

© Rafael Rico López

17/47

Arquitectura *software* x86-16bits

2. Segmentación de memoria

- En ensamblador no se representan las direcciones como *base:desplazamiento* aunque si lo admiten los depuradores
- Los desplazamientos llevan asociada una base por defecto
- Cuando es necesario especificar la **base** se hace de manera simbólica (usando el nombre del registro de segmento como prefijo de la instrucción)

© Rafael Rico López

18/47

Arquitectura *software* x86-16bits

2. Segmentación de memoria

base offset	CS	SS	DS	ES
IP	sí	no	no	no
SP	no	sí	no	no
BP	prefijo	por defecto	prefijo	prefijo
BX	prefijo	prefijo	por defecto	prefijo
SI	prefijo	prefijo	por defecto	prefijo
DI	prefijo	prefijo	por defecto	por defecto

© Rafael Rico López

19/47

Arquitectura *software* x86-16bits

2. Segmentación de memoria

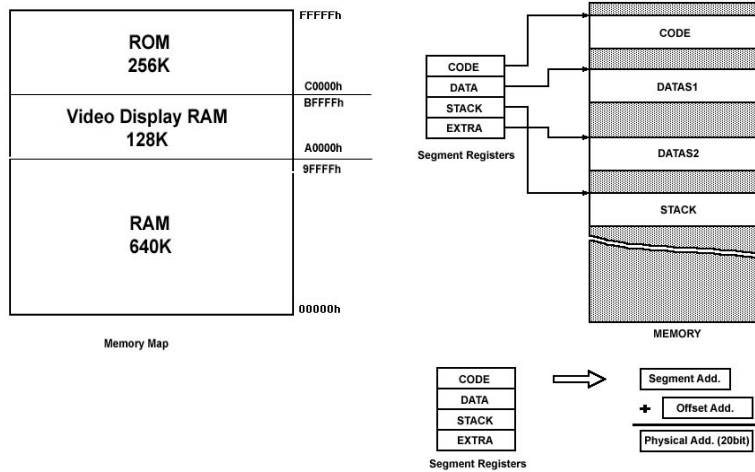
- Las direcciones pueden ser:
 - ➔ *Near*
 - ➔ *Far*
- *Near* : sólo el **desplazamiento**; la **base** toma el valor en curso de la dada por defecto
- *Far* : se requiere la especificación de la **base** y del **desplazamiento**

© Rafael Rico López

20/47

Arquitectura software x86-16bits

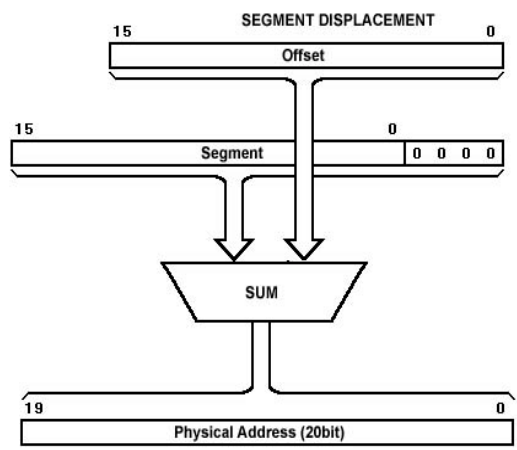
2. Segmentación de memoria



© Rafael Rico López

Arquitectura software x86-16bits

2. Segmentación de memoria



© Rafael Rico López

Arquitectura *software* x86-16bits

3. Modos de direccionamiento

- **Determinan la ubicación de los operandos**
- **Tres posibles ubicaciones:**
 - ➔ Inmediatos (en la propia instrucción)
 - ➔ En registros
 - ➔ En memoria
 - ➔ (Implícitos) → no se identifican, son sobreentendidos

© Rafael Rico López

23/47

Arquitectura *software* x86-16bits

3. Modos de direccionamiento

- **Momento en el que se actualiza el modo de direccionamiento:**
 - ➔ Inmediatos → tiempo de ensamblado
 - ➔ En registros → tiempo de programación o compilación
 - ➔ En memoria
 - Absolutos → tiempo de carga
 - Relativos → tiempo de ejecución

interviene el SO

© Rafael Rico López

24/47

Arquitectura software x86-16bits

3. Modos de direccionamiento

- Clasificación de estructura de computadores:

inmediato		
directo	absoluto	a registro a memoria a página base
	relativo	a registro base + desplazamiento base + índice + desplazamiento contador de programa puntero de pila
indirecto		
implícito		

© Rafael Rico López

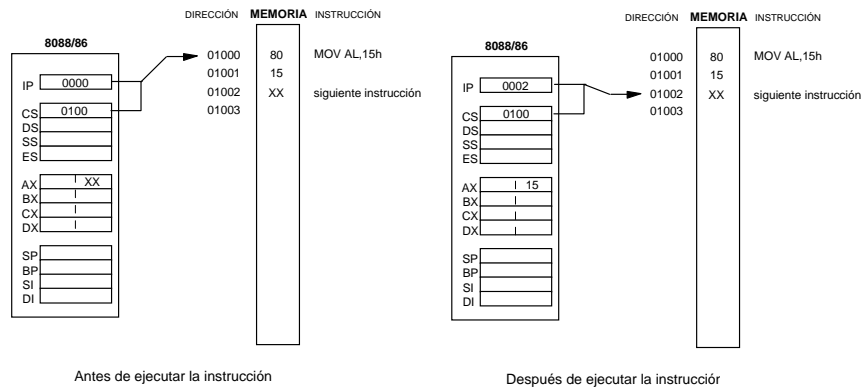
25/47

Arquitectura software x86-16bits

3. Modos de direccionamiento

3.1. Inmediato

➔ Ejemplo: MOV AL,15h



© Rafael Rico López

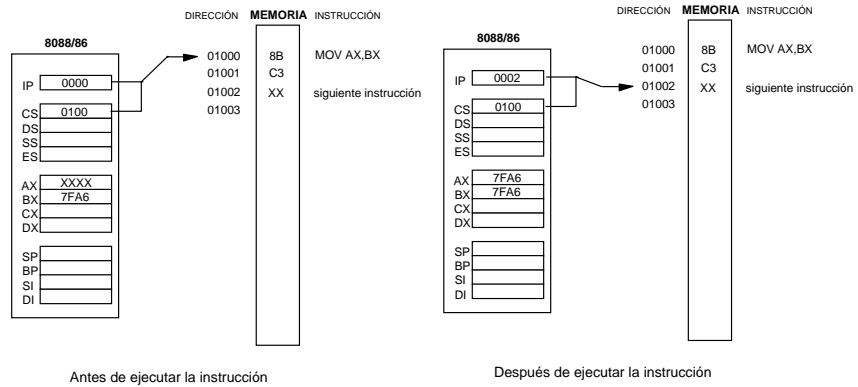
26/47

Arquitectura software x86-16bits

3. Modos de direccionamiento

3.2. Directo absoluto a registro

➔ Ejemplo: MOV AX,BX



© Rafael Rico López

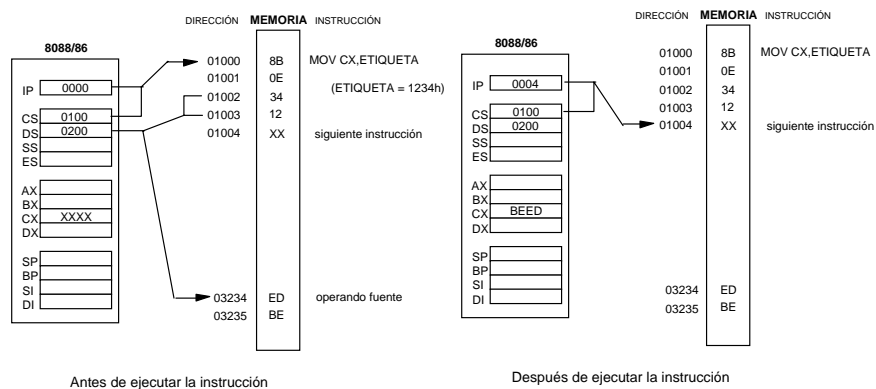
27/47

Arquitectura software x86-16bits

3. Modos de direccionamiento

3.3. Directo absoluto a memoria (nombre INTEL: directo a memoria)

➔ Ejemplo: MOV CX, ETIQUETA



© Rafael Rico López

28/47

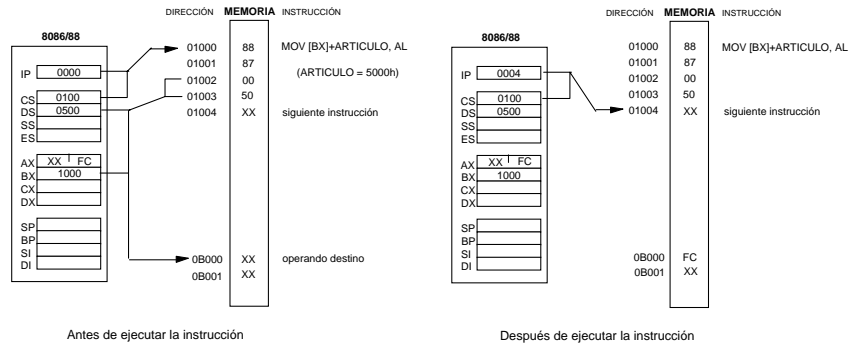
Arquitectura software x86-16bits

3. Modos de direccionamiento

3.4. Directo relativo a registro (nombre INTEL: indirecto a memoria)

➔ Ejemplo: `MOV [BX]+ARTICULO, AL`

registro base



© Rafael Rico López

29/47

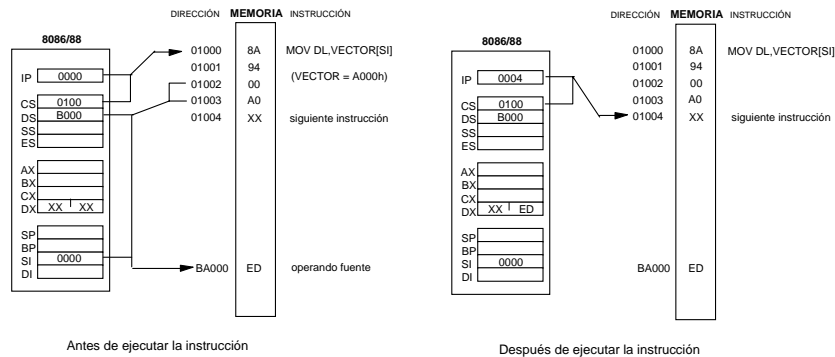
Arquitectura software x86-16bits

3. Modos de direccionamiento

3.4. Directo relativo a registro (nombre INTEL: indirecto a memoria)

➔ Ejemplo: `MOV DL, VECTOR[SI]`

registro índice



© Rafael Rico López

30/47

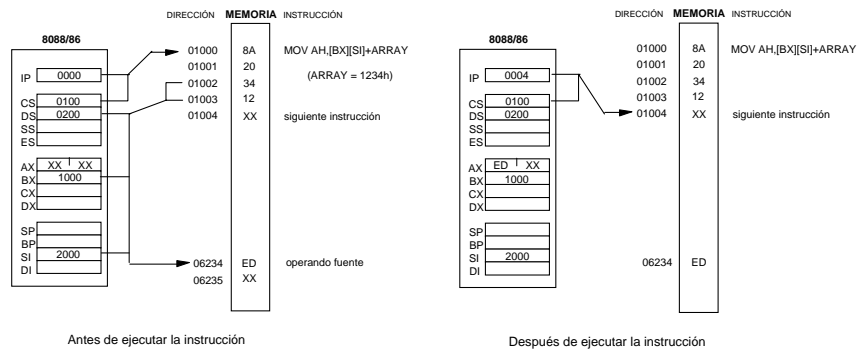
Arquitectura software x86-16bits

3. Modos de direccionamiento

3.4. Directo relativo a registro (nombre INTEL: indirecto a memoria)

➔ Ejemplo: `MOV AH, [BX][SI]+ARRAY`

registro índice y base



© Rafael Rico López

31/47

Arquitectura software x86-16bits

4. Formato de instrucciones

- Es la codificación binaria de las instrucciones
- Debe proporcionar información acerca de:
 - ➔ Operación a realizar
 - ➔ Operandos y resultado
 - ➔ Siguiete instrucción (secuencia implícita)
- El x86-16bits cuenta con dos formatos:
 - ➔ Formato general
 - ➔ Formato especial (campo de extensión y/o prefijo)
- Está diseñado para minimizar el espacio de representación

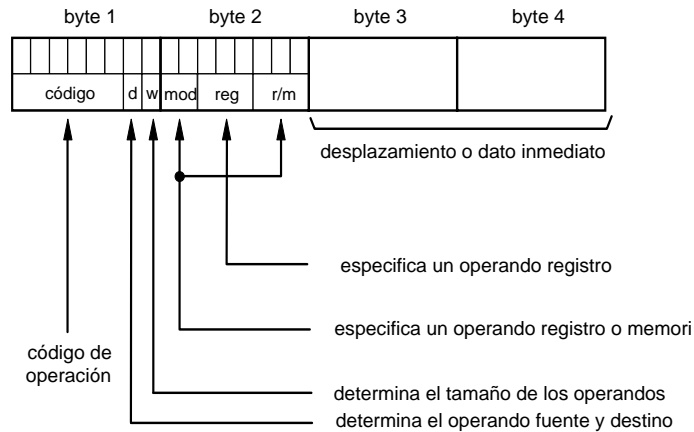
© Rafael Rico López

32/47

Arquitectura software x86-16bits

4. Formato de instrucciones

4.1. Formato general



© Rafael Rico López

33/47

Arquitectura software x86-16bits

4. Formato de instrucciones

4.1. Formato general

- El primer byte contiene 3 clases de información:
 - ➔ **Código de operación:** los 6 primeros bits contienen el código de la operación a realizar
 - ➔ **El bit de dirección de registro (D):** especifica si el operando dado por el campo de *registro operando* REG en el segundo byte es el operando fuente o destino:
 - si D = 1 tengo que REG = operando destino
 - si D = 0 tengo que REG = operando fuente
 - ➔ **El bit de tamaño del dato (W):** especifica si la operación será realizada sobre datos de 8 o de 16 bits:
 - si w = 0 los datos son de 8 bits
 - si w = 1 los datos son de 16 bits

© Rafael Rico López

34/47

Arquitectura *software* x86-16bits

4. Formato de instrucciones

4.1. Formato general

- El segundo byte contiene los operandos:
 - ➔ Uno siempre es un registro
 - ➔ Viene dado por REG
 - ➔ El otro puede ser registro o memoria
 - ➔ Viene dado por MOD y R/M
 - ➔ NO se admiten las operaciones entre dos operandos ubicados en memoria

Arquitectura *software* x86-16bits

4. Formato de instrucciones

4.1. Formato general

- Campo REG:

REG	w = 0	w = 1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Arquitectura software x86-16bits

4. Formato de instrucciones

4.1. Formato general

- Campo MOD y R/M:

R/M	MOD = 11		Cálculo de la dirección efectiva		
	w = 0	w = 1	MOD = 00	MOD = 01	MOD = 10
000	AL	AX	[BX]+[SI]	[BX]+[SI]+D8	[BX]+[SI]+D16
001	CL	CX	[BX]+[DI]	[BX]+[DI]+D8	[BX]+[DI]+D16
010	DL	DX	[BP]+[SI]	[BP]+[SI]+D8	[BP]+[SI]+D16
011	BL	BX	[BP]+[DI]	[BP]+[DI]+D8	[BP]+[DI]+D16
100	AH	SP	[SI]	[SI]+D8	[SI]+D16
101	CH	BP	[DI]	[DI]+D8	[DI]+D16
110	DH	SI	dirección	[BP]+D8	[BP]+D16
111	BH	DI	[BX]	[BX]+D8	[BX]+D16

© Rafael Rico López

37/47

Arquitectura software x86-16bits

4. Formato de instrucciones

4.1. Formato general

- ➔ Registros implicados en el cálculo de direcciones (se incluye el registro de segmento por defecto):

R/M	Dirección efectiva	Reg. de segmento
000	DSx16+[BX]+[SI]	DS
001	DSx16+[BX]+[DI]	
010	SSx16+[BP]+[SI]	SS
011	SSx16+[BP]+[DI]	
100	DSx16+[SI]	DS
101	DSx16+[DI]	
110	SSx16+[BP]	SS
111	DSx16+[BX]	DS

© Rafael Rico López

38/47

Arquitectura software x86-16bits

4. Formato de instrucciones

dec	0	16	32	48	64	80	96	112
hex	0	1	2	3	4	5	6	7
0	ADD reg8→r/m	ADD reg16→r/m	ADD r/m→reg8	ADD r/m→reg16	ADD inm8→AL	ADD inm16→AX	PUSH ES	POP ES
1	ADC reg8→r/m	ADC reg16→r/m	ADC r/m→reg8	ADC r/m→reg16	ADC inm8→AL	ADC inm16→AX	PUSH SS	POP SS
2	AND reg8→r/m	AND reg16→r/m	AND r/m→reg8	AND r/m→reg16	AND inm8→AL	AND inm16→AX	ES: DAA	DAA
3	XOR reg8→r/m	XOR reg16→r/m	XOR r/m→reg8	XOR r/m→reg16	XOR inm8→AL	XOR inm16→AX	SS: AAA	AAA
4	INC AX	INC CX	INC DX	INC BX	INC SP	INC BP	INC SI	INC DI
5	PUSH AX	PUSH CX	PUSH DX	PUSH BX	PUSH SP	PUSH BP	PUSH SI	PUSH DI
6								
7	JO d8	JNO d8	JB/JNAE d8	JNB/JAE d8	JE/JZ d8	JNE/JNZ d8	JBE/JNA d8	JNBE/JA d8
8	INMEDIATO inm(8-16)→r/m				TEST reg8.r/m	TEST reg16.r/m	XCHG reg8.r/m	XCHG reg16.r/m
9	NOP	XCHG AX: CX	XCHG AX: DX	XCHG AX: BX	XCHG AX: SP	XCHG AX: BP	XCHG AX: SI	XCHG AX: DI
10	MOV m→AL	MOV m→AX	MOV AL→m	MOV AX→m	MOVSB	MOVSW	CMPSB	CMPSW
11	MOV inm8→AL	MOV inm8→CL	MOV inm8→DL	MOV inm8→BL	MOV inm8→AH	MOV inm8→CH	MOV inm8→DH	MOV inm8→BH
12		RET inm16	RET		LES m→reg16	LDS m→reg16	MOV inm8→r/m	MOV inm16→r/m
13	DES PLAZAMIENTO 1→r/m		DES PLAZAMIENTO CL→r/m		AAA	AAD		XLAT
14	LOOPE d8	LOOPE d8	LOOP d8	JKZ d8	IN inm8→AL	IN inm8→AX	OUT AL→inm8	OUT AX→inm16
15	LOCK		REP	REPZ	HLT	CMC	GRUPO 1	

© Rafael Rico López

4.1. Formato general

39/47

Arquitectura software x86-16bits

4. Formato de instrucciones

dec	128	144	160	176	192	208	224	240
hex	8	9	A	B	C	D	E	F
0	OR reg8→r/m	OR reg16→r/m	OR r/m→reg8	OR r/m→reg16	OR inm8→AL	OR inm16→AX	PUSH CS	
1	SBB reg8→r/m	SBB reg16→r/m	SBB r/m→reg8	SBB r/m→reg16	SBB inm8→AL	SBB inm16→AX	PUSH DS	POP DS
2	SUB reg8→r/m	SUB reg16→r/m	SUB r/m→reg8	SUB r/m→reg16	SUB inm8→AL	SUB inm16→AX	CS: DAS	DAS
3	CMP reg8.r/m	CMP reg16.r/m	CMP r/m.reg8	CMP r/m.reg16	CMP inm8:AL	CMP inm16:AX	DS: AAS	AAS
4	DEC AX	DEC CX	DEC DX	DEC BX	DEC SP	DEC BP	DEC SI	DEC DI
5	POP AX	POP CX	POP DX	POP BX	POP SP	POP BP	POP SI	POP DI
6								
7	JS d8	JNS d8	JP/JPE d8	JNP/JPO d8	JL/JNGE d8	JNL/JGE d8	JLE/JNG d8	JNLE/JG d8
8	MOV reg8→r/m	MOV reg16→r/m	MOV r/m→reg8	MOV r/m→reg16	MOV r/m→seg	LEA m→reg16	MOV seg→r/m	POP r/m
9	CBW	CWD	CALL b:d16	WAIT	PUSHF	POPF	SAHF	LAHF
10	TEST AL:inm8	TEST AX:inm16	STOSB	STOSW	LODSB	LODSW	SCASB	SCASW
11	MOV inm16→AX	MOV inm16→CX	MOV inm16→DX	MOV inm16→BX	MOV inm16→SP	MOV inm16→BP	MOV inm16→SI	MOV inm16→DI
12		RET inm16	RET	INT 3	INT inm8	INTO	IRET	
13	ESC 0	ESC 1	ESC 2	ESC 3	ESC 4	ESC 5	ESC 6	ESC 7
14	CALL d16	JMP d16	JMP b:d16	JMP d8	IN DX→AL	IN DX→AX	OUT AL→DX	OUT AX→DX
15	CLC	STC	CLI	STI	CLD	STD	GRUPO 2	

© Rafael Rico López

4.1. Formato general

40/47

Arquitectura *software* x86-16bits

4. Formato de instrucciones

- **Leyenda:**

- ➔ **reg8, reg16:** registros de tamaño byte o palabra
- ➔ **inm8, inm16:** inmediatos de tamaño byte o palabra
- ➔ **r/m:** operando registro o memoria
- ➔ **m:** operando memoria
- ➔ **d8, d16:** desplazamiento
- ➔ **b:** base
- ➔ **b:d16** es una dirección completa de memoria
- ➔ **'→':** se escribe
- ➔ **' ':** no se escribe

© Rafael Rico López

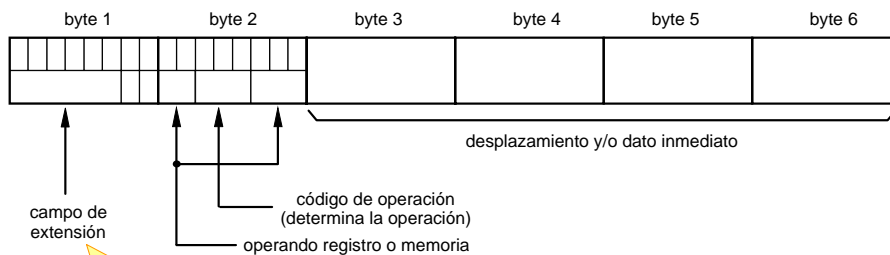
41/47

Arquitectura *software* x86-16bits

4. Formato de instrucciones

4.2. Formato especial con campo de extensión

- **Se necesitan 2 bytes para determinar la operación**



no determina la operación

© Rafael Rico López

42/47

Arquitectura software x86-16bits

4. Formato de instrucciones

4.2. Formato especial con campo de extensión

- ➔ Los campos MOD y R/M dan lugar al operando
- ➔ El campo REG da el código de operación completando el campo de extensión

CAMPO DE EXTENSIÓN	GRUPO
80, 81, 82, 83	Inmediatos
D0, D1, D2, D3	Desplazamientos
F6, F7	Grupo I
FE, FF	Grupo II

© Rafael Rico López

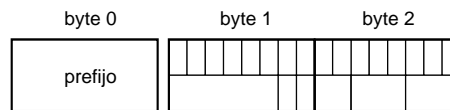
	000	001	010	011	100	101	110	111
mod r/m								
INMEDIATO	ADD <small>inm→r/m</small>	OR <small>inm→r/m</small>	ADC <small>inm→r/m</small>	SBB <small>inm→r/m</small>	AND <small>inm→r/m</small>	SUB <small>inm→r/m</small>	XOR <small>inm→r/m</small>	CMP <small>inm/r/m</small>
DESPLAZAMIENTO	ROL	ROR	RCL	RCR	SHL/SAL	SHR		SAR
GRUPO 1	TEST <small>inm/r/m</small>		NOT <small>r/m</small>	NEG <small>r/m</small>	MUL <small>r/m</small>	IMUL <small>r/m</small>	DIV <small>r/m</small>	IDIV <small>r/m</small>
GRUPO 2	INC <small>r/m</small>	DEC <small>r/m</small>	CALL <small>indirecto</small>	CALL <small>indirecto</small>	JMP <small>indirecto</small>	JMP <small>indirecto</small>	PUSH <small>m</small>	

43/47

Arquitectura software x86-16bits

4. Formato de instrucciones

4.3. Formato especial con prefijo



Prefijo	Efecto
CS	Modifican la base por defecto
DS	
SS	
ES	
REP	Repetición de instrucción de cadena
REPZ/REPE	
REPNZ/REPNE	
LOCK	Sincronización con coprocesador

Arquitectura software x86-16bits

4. Formato de instrucciones

- **Ejemplo:**

- ➔ `MOV AX,[BX][SI]`

- ➔ `DSX16+[BX][SI]`

- ➔ `CS:MOV AX,[BX][SI]`

- ➔ `CSX16+[BX][SI]`

Arquitectura software x86-16bits

4. Formato de instrucciones

- **EJEMPLO:** La instrucción `MOV BL,AL` mueve el byte contenido en el registro fuente `AL` al registro destino `BL`. ¿Cuál es el código máquina de la instrucción sabiendo que el código de operación es 100010_2 ?

- ➔ **SOLUCION:** En el primer byte los primeros 6 bits especifican la operación y deben ser:

`CODIGO DE OPERACION = 1000102`

- ➔ El bit `D` indica si el registro que señala el campo `REG` del 2º byte es el operando fuente o el destino. Codificaremos el registro `AL` en el campo `REG` y, por tanto, `D` será `0`

- ➔ El bit `W` será `0` para indicar una operación de tamaño byte

- ➔ **PRIMER BYTE = $1000\ 1000_2 = 88h$**

Arquitectura *software* x86-16bits

4. Formato de instrucciones

- ➔ En el segundo byte, REG indica el operando fuente que es AL. Su código es: REG = 000
- ➔ Como el segundo operando es también un registro, MOD debe valer 11. El campo R/M debe especificar que el registro destino es BL y su codificación es 011. Esto da:

MOD = 11 R/M = 011

- ➔ SEGUNDO BYTE = $1100\ 0011_2 = C3h$

- ➔ Y el código hexadecimal completo para la instrucción es:

MOV BL,AL = 88 C3h