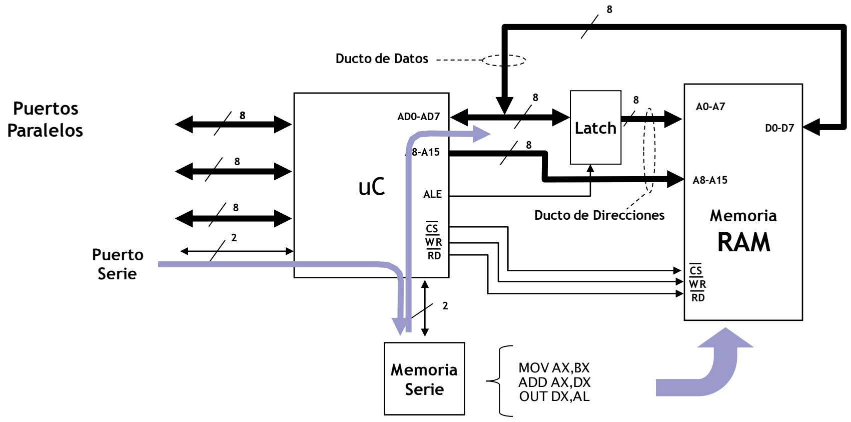
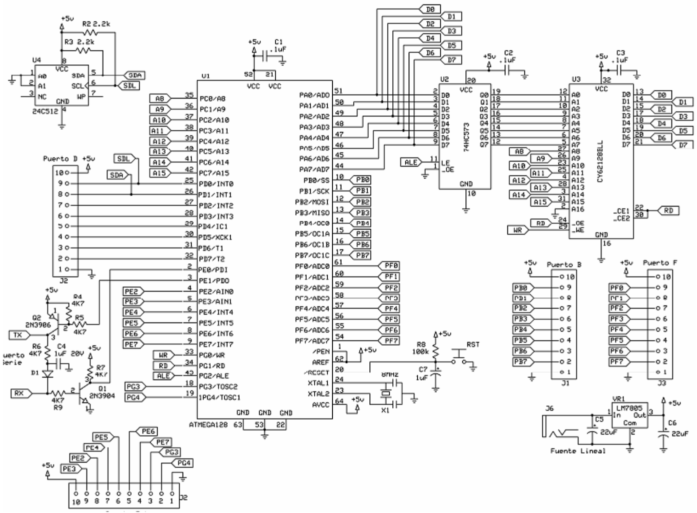
Interprete de código de 16 bits 80x86

Diagrama Simplificado

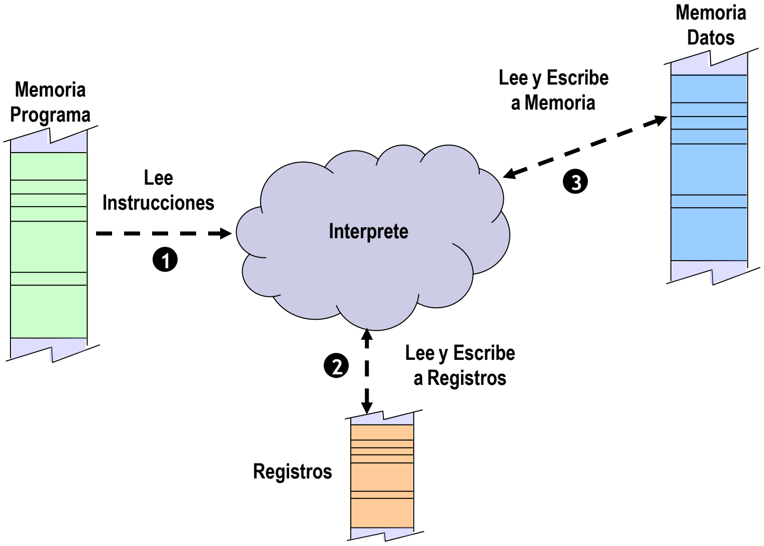


```
MOV AX,BX
ADD AX,DX
OUT DX,AL
```

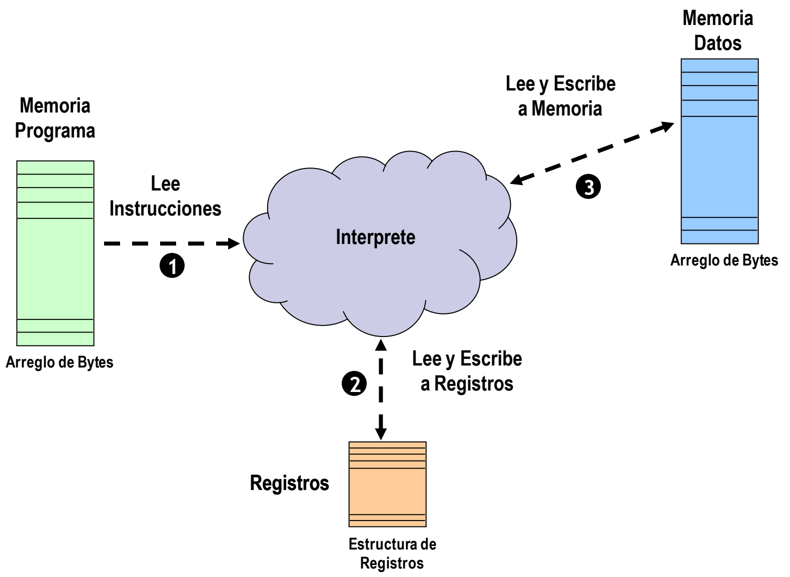
Descripción



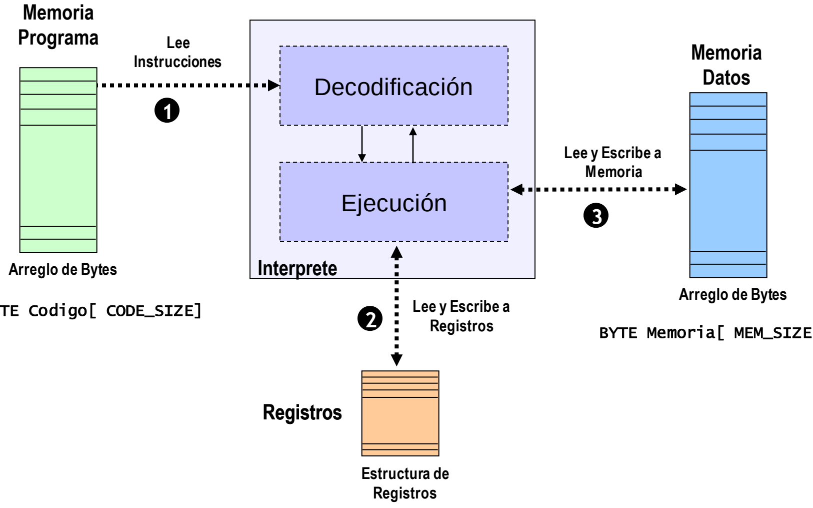
Interprete



Interprete



Interprete

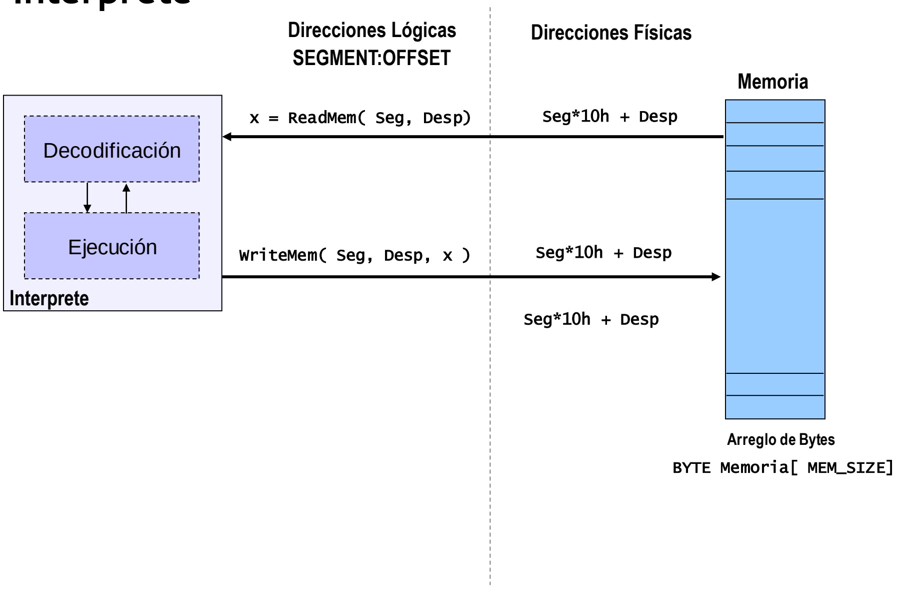


```
BYTE Codigo[ CODE_SIZE]
```

```
BYTE Memoria[ MEM_SIZE]
```

```
Reg_t  Registro[8]
/* Flags, IP, CS, DS, ES, SS */
```

Interprete



Direcciones Lógicas
SEGMENT:OFFSET

Direcciones Físicas

Memoria

Decodificación

Ejecución

Interprete

`x = ReadMem( Seg, Desp)`

`Seg*10h + Desp`

`WriteMem( Seg, Desp, x )`

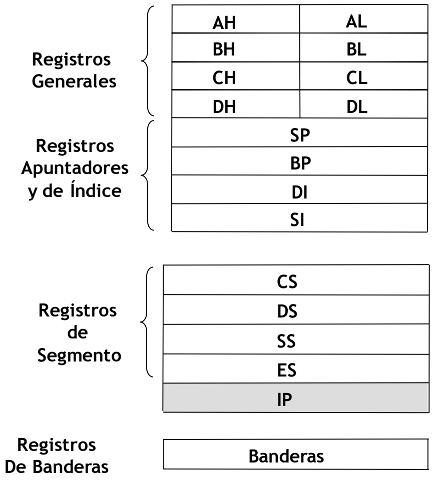
`Seg*10h + Desp`

`Seg*10h + Desp`

Arreglo de Bytes

`BYTE Memoria[ MEM_SIZE]`

Registros



Registros Generales	AH	AL
	BH	BL
	CH	CL
	DH	DL
Registros Apuntadores y de Índice	SP	
	BP	
	DI	
	SI	
Registros de Segmento	CS	
	DS	
	SS	
	ES	
	IP	
Registros De Banderas	Banderas	

Instrucciones (1 o 2 bytes)

INC = Increment:

Register/memory	1 1 1 1 1 1 1 w	mod 0 0 0 r/m
Register	0 1 0 0 0 reg	

OUT = Output to:

Fixed port	1 1 1 0 0 1 1 w	port
Variable port	1 1 1 0 1 1 1 w	

PUSH = Push:

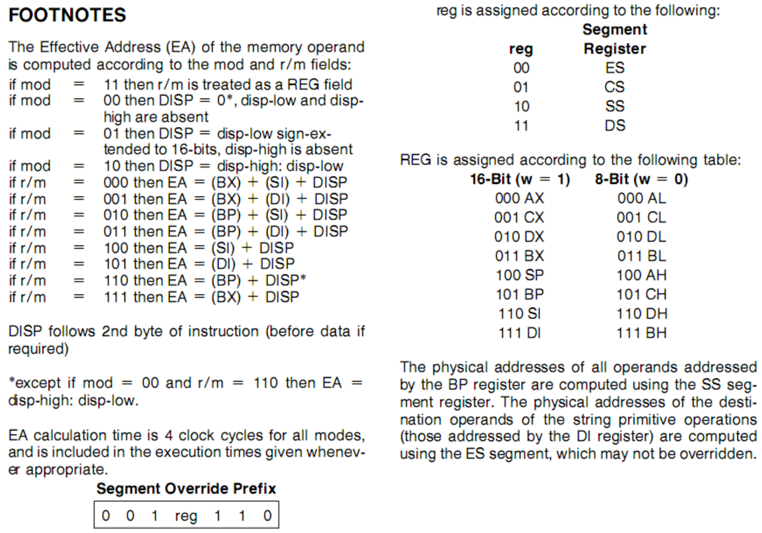
Memory	1 1 1 1 1 1 1 1	mod 1 1 0 r/m
Register	0 1 0 1 0 reg	

Instrucciones (2, 3 o mas bytes)

Immediate to register	1 0 1 1 w reg	data	data if w = 1
Memory to accumulator	1 0 1 0 0 0 0 w	addr-low	addr-high
Accumulator to memory	1 0 1 0 0 0 1 w	addr-low	addr-high

Immediate to register/memory	1 1 0 0 0 1 1 w	mod 000 r/m	data	data if w = 1

Instrucciones



FOOTNOTES

The Effective Address (EA) of the memory operand is computed according to the mod and r/m fields:

if mod = 11 then r/m is treated as a REG field
if mod = 00 then DISP = 0*, disp-low and disp-high are absent
if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
if mod = 10 then DISP = disp-high: disp-low
if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/m = 101 then EA = (DI) + DISP
if r/m = 110 then EA = (BP) + DISP*
if r/m = 111 then EA = (BX) + DISP

DISP follows 2nd byte of instruction (before data if required)

*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low.

EA calculation time is 4 clock cycles for all modes, and is included in the execution times given whenever appropriate.

Segment Override Prefix

0	0	1	reg	1	1	0

reg is assigned according to the following:

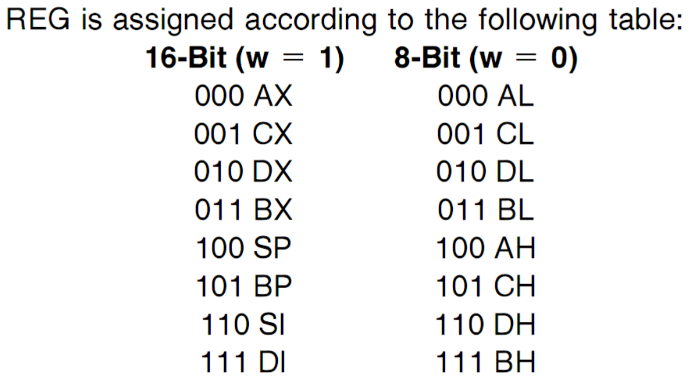
reg	**Segment Register**
00	ES
01	CS
10	SS
11	DS

REG is assigned according to the following table:

16-Bit (w = 1)	**8-Bit (w = 0)**
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
100 SP	100 AH
101 BP	101 CH
110 SI	110 DH
111 DI	111 BH

The physical addresses of all operands addressed by the BP register are computed using the SS segment register. The physical addresses of the destination operands of the string primitive operations (those addressed by the DI register) are computed using the ES segment, which may not be overridden.

Instrucciones



REG is assigned according to the following table:

16-Bit (w = 1)	**8-Bit (w = 0)**
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
100 SP	100 AH
101 BP	101 CH
110 SI	110 DH
111 DI	111 BH

Instrucciones

INC = Increment:

Register/memory	1 1 1 1 1 1 1 w	mod 0 0 0 r/m
Register	0 1 0 0 0 reg	

Ejemplo:

Código	Campos		Mnemónico
	oper	reg	
40h	0 1 0 0 0	**0 0 0**	INC AX
41h	0 1 0 0 0	**0 0 1**	INC CX
	⋮		
47h	0 1 0 0 0	**1 1 1**	INC DI

Decodificación y Ejecución

Ejemplo:

✗

```
switch (codigo){

        case 0x40:    AX++;  /* incrementa AX */
                      break;

        case 0x41:    CX++;  /* incrementa cx */
                      break;

                 :

        case 0x47:    DI++;  /* incrementa cx */
                      break;

}
```

Decodificación y Ejecución

Ejemplo:

```
/* a) Tener un arreglo de registros que conforman a AX, CX etc. */

T_Registro Registro[8];

/* b) Tener un registro de instrucción formado por campos de bits */

/* teniendo así: Codigo.oper y Codigo.reg  */

/* ahora si */        #define INC_R16 0x08

if ( Codigo.oper ==   INC_R16   )    ✓
    Registro[ Codigo.reg ]++;
```

INC Reg$_{16}$

0 1 0 0 0	r e g

Esquema de Registros

INC = Increment:

Register/memory	1 1 1 1 1 1 1 w	mod 0 0 0 r/m
Register	0 1 0 0 0 reg	

Registros Generales	AH	AL
	BH	BL
	CH	CL
	DH	DL
Registros Apuntadores y de Índice	SP	
	BP	
	DI	
	SI	
Registros de Segmento	CS	
	DS	
	SS	
	ES	
	IP	

REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
100 SP	100 AH
101 BP	101 CH
110 SI	110 DH
111 DI	111 BH

Esquema de Registros

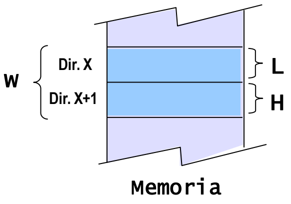
INC = Increment:

Register/memory	1 1 1 1 1 1 1 w	mod 0 0 0 r/m

DIR. LÓGICA	CÓDIGO	MNEMÓNICO
136E:0100	FEC0	INC AL
136E:0102	FEC4	INC AH
136E:0108	FE060001	INC BYTE PTR [0100]
136E:0104	FF060001	INC WORD PTR [0100]

Declaración de Registros

```
#define WORD unsigned int
#define BYTE unsigned char

union u_reg {
        WORD w;         /* Registro de 16 bits        */
        BYTE L;         /* Registros de 8 bits        */
        BYTE H;         /* Registros de 8 bits        */

};
```



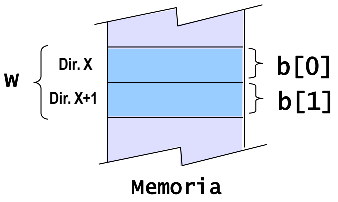
Memoria

Ejemplo:

```
/* declarando a AX como:       */
union u_reg AX;

/* se puede accesar como       */
AX.w = 0x1234; /* 16 bits      */

/* 8 bits */
AX.L = 0x23;   /* AL   */
AX.H = 0xef;   /* AH   */
```

Declaración de Registros

```
#define WORD unsigned int
#define BYTE unsigned char

union u_reg {
        WORD w;         /* Registro de 16 bits        */
        BYTE b[2];      /* Registros de 8 bits        */
      ~~BYTE L;~~
      ~~BYTE H;~~
};
```



Ejemplo:

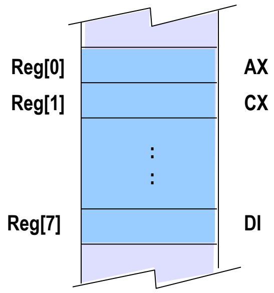
```
/* declarando a AX como:      */
union u_reg AX;

/* se puede accesar como      */
AX.w = 0x1234; /* 16 bits     */

/* 8 bits */
AX.b[0] = 0x23;        /* AL   */
AX.b[1] = 0xef;        /* AH   */
```

Declaración de Registros

```
#define WORD unsigned int
#define BYTE unsigned char

union u_reg {
        WORD w;         /* Registro de 16 bits        */
        BYTE b[2];      /* Registros de 8 bits        */
};

/* nuevo tipo de dato  */
typedef union u_reg Registro;

/* Ejemplo de Acceso    */

Registro AX;

AX.w = 0x1234;          /* Como registro de 16 bits         */
AX.b[0]= 0xff;          /* Como registro de 8 bits (baja)   */
AX.b[1]= 0x3f;          /* Como registro de 8 bits (alta)   */
```

Declaración de Registros

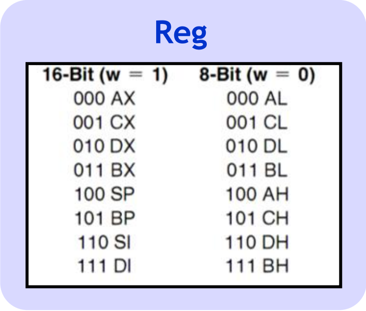
```
#define WORD unsigned int
#define BYTE unsigned char

union u_reg {
        WORD w;         /* Registro de 16 bits        */
        BYTE b[2];      /* Registros de 8 bits        */
};

typedef union u_reg Registro; /* nuevo tipo de dato  */


Registro Reg[8];
/* Acceso a nuevo tipo de dato */
```



Acceso a Registros

- Caso de Registros de 16 bits

```
/* si REG=7 */
dato=Reg[REG].w;

/* si REG=0 */
dato=Reg[REG].w;

/* si REG=3 */
dato=Reg[REG].w;    /
```



Reg

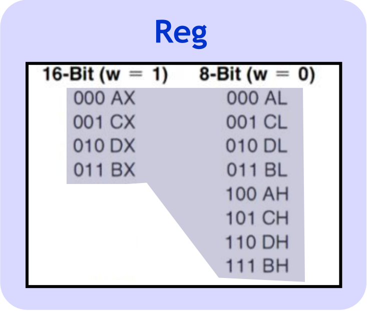
16-Bit (w = 1)	8-Bit (w = 0)
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
100 SP	100 AH
101 BP	101 CH
110 SI	110 DH
111 DI	111 BH

Acceso a Registros

- Caso de Registros de 8 bits

/* si REG=0 → AL*/
~~dato=Reg[REG].b[REG];~~

/* si REG=1 → CL */
~~dato=Reg[REG].b[REG];~~



Reg

16-Bit (w = 1)	8-Bit (w = 0)
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
	100 AH
	101 CH
	110 DH
	111 BH

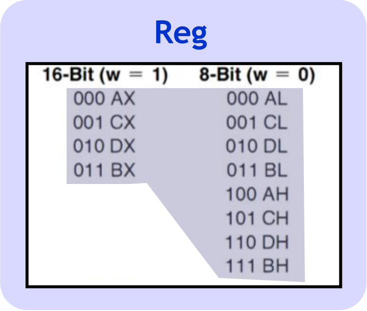
Acceso a Registros

- Caso de Registros de 8 bits

/* si REG=0 → AL*/
~~dato=Reg[REG].b[REG];~~

/* si REG=1 → CL */
~~dato=Reg[REG].b[REG];~~

```
/* si REG=1 → CL */
dato=Reg[ REG % 4].b[ REG / 4];

#define  REG_PG16( reg ) reg & 0x03
#define  REG_HL (reg)  reg>>2

dato=Reg[ REG_PG( REG) ].b[ REG_HL( REG) ];
```



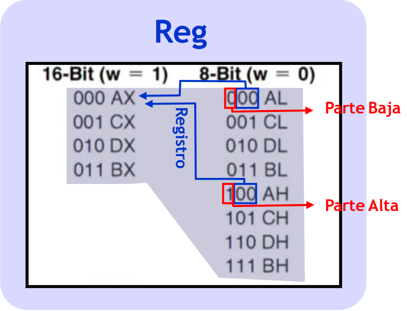
Reg

16-Bit (w = 1)	8-Bit (w = 0)
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
	100 AH
	101 CH
	110 DH
	111 BH

Acceso a Registros

- Caso de General

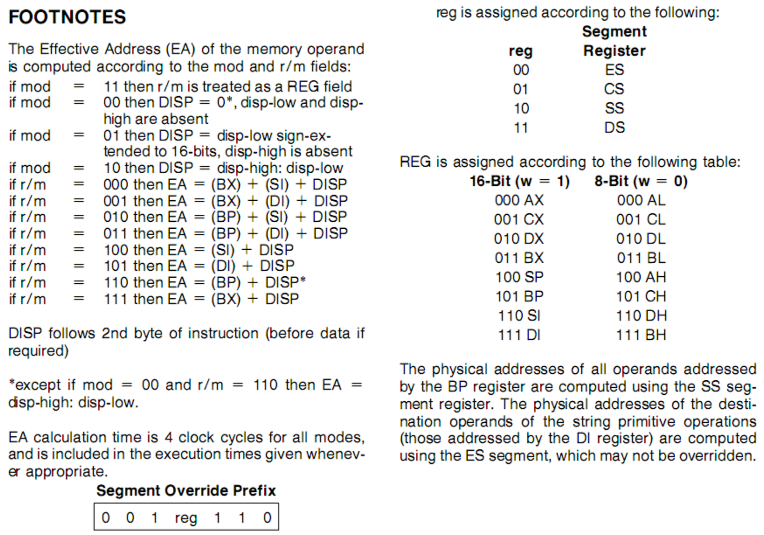
```
#define  REG_PG16( reg ) reg & 0x03
#define  REG_HL (reg)  reg>>2

registro=codigo.reg;

If( codigo.w == 1)

          Reg[ registro  ].w++;
else

          Reg[ REG_PG( registro ) ].b[ REG_HL( registro ) ]++;
```



Reg

16-Bit (w = 1)	8-Bit (w = 0)
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
	100 AH
	101 CH
	110 DH
	111 BH

Decodificación de Instrucciones (2 bytes)



FOOTNOTES

The Effective Address (EA) of the memory operand is computed according to the mod and r/m fields:

```
if mod   =   11 then r/m is treated as a REG field
if mod   =   00 then DISP = 0*, disp-low and disp-
             high are absent
if mod   =   01 then DISP = disp-low sign-ex-
             tended to 16-bits, disp-high is absent
if mod   =   10 then DISP = disp-high: disp-low
if r/m   =   000 then EA = (BX) + (SI) + DISP
if r/m   =   001 then EA = (BX) + (DI) + DISP
if r/m   =   010 then EA = (BP) + (SI) + DISP
if r/m   =   011 then EA = (BP) + (DI) + DISP
if r/m   =   100 then EA = (SI) + DISP
if r/m   =   101 then EA = (DI) + DISP
if r/m   =   110 then EA = (BP) + DISP*
if r/m   =   111 then EA = (BX) + DISP
```

DISP follows 2nd byte of instruction (before data if required)

*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low.

EA calculation time is 4 clock cycles for all modes, and is included in the execution times given whenever appropriate.

Segment Override Prefix

0	0	1	reg	1	1	0

reg is assigned according to the following:

reg	Segment Register
00	ES
01	CS
10	SS
11	DS

REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
100 SP	100 AH
101 BP	101 CH
110 SI	110 DH
111 DI	111 BH

The physical addresses of all operands addressed by the BP register are computed using the SS segment register. The physical addresses of the destination operands of the string primitive operations (those addressed by the DI register) are computed using the ES segment, which may not be overridden.

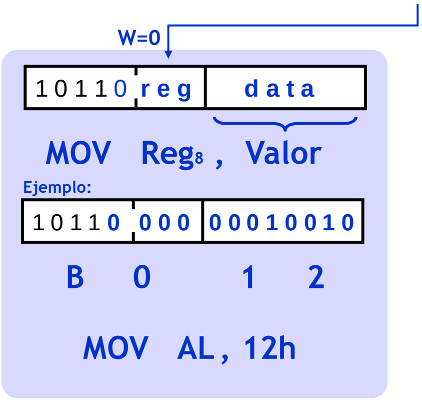
Decodificación de Instrucciones (2 bytes)

DATA TRANSFER
MOV = Move:

Register to Register/Memory	1 0 0 0 1 0 0 w	mod reg r/m		
Register/memory to register	1 0 0 0 1 0 1 w	mod reg r/m		
Immediate to register/memory	1 1 0 0 0 1 1 w	mod 000 r/m	data	data if w = 1
Immediate to register	1 0 1 1 w reg	data	data if w = 1	
Memory to accumulator	1 0 1 0 0 0 0 w	addr-low	addr-high	
Accumulator to memory	1 0 1 0 0 0 1 w	addr-low	addr-high	
Register/memory to segment register	1 0 0 0 1 1 1 0	mod 0 reg r/m		
Segment register to register/memory	1 0 0 0 1 1 0 0	mod 0 reg r/m		

Decodificación de Instrucciones

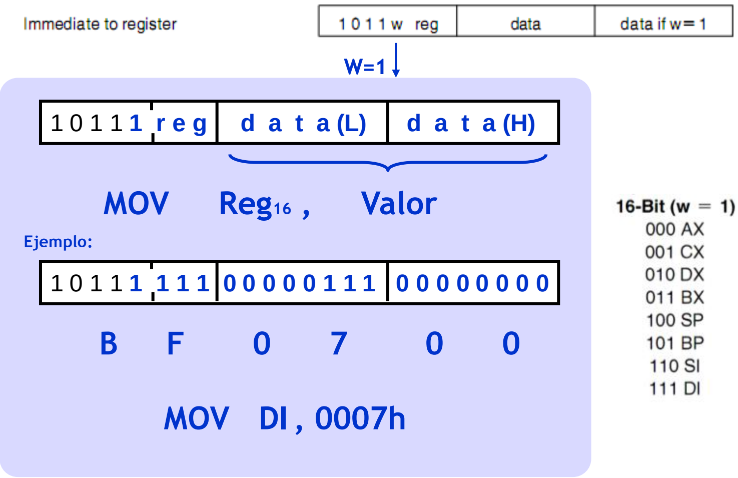
Immediate to register	1 0 1 1 w reg	data	data if w = 1



REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
100 SP	100 AH
101 BP	101 CH
110 SI	110 DH
111 DI	111 BH

Decodificación de Instrucciones



Immediate to register

1 0 1 1 w reg	data	data if w = 1

W=1

1 0 1 1 1 reg	d a t a (L)	d a t a (H)

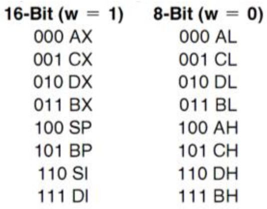
MOV Reg16 , Valor

Ejemplo:

1 0 1 1 1 1 1 1	0 0 0 0 0 1 1 1	0 0 0 0 0 0 0 0

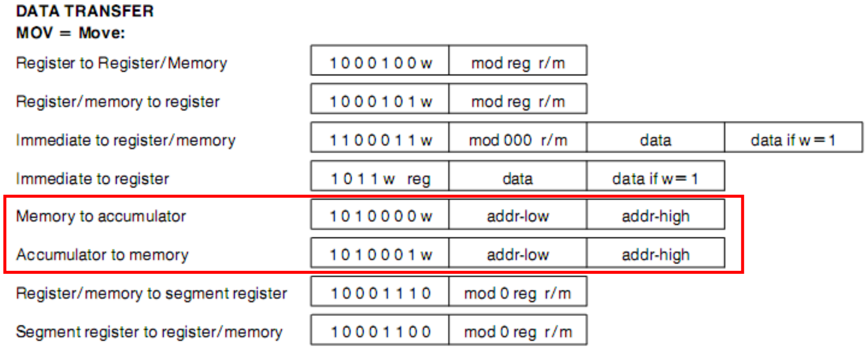
B F 0 7 0 0

MOV DI , 0007h



16-Bit (w = 1)	8-Bit (w = 0)
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
100 SP	100 AH
101 BP	101 CH
110 SI	110 DH
111 DI	111 BH

Decodificación de Instrucciones (2 bytes)



DATA TRANSFER
MOV = Move:

Register to Register/Memory	1 0 0 0 1 0 0 w	mod reg r/m		
Register/memory to register	1 0 0 0 1 0 1 w	mod reg r/m		
Immediate to register/memory	1 1 0 0 0 1 1 w	mod 000 r/m	data	data if w = 1
Immediate to register	1 0 1 1 w reg	data	data if w = 1	
Memory to accumulator	1 0 1 0 0 0 0 w	addr-low	addr-high	
Accumulator to memory	1 0 1 0 0 0 1 w	addr-low	addr-high	
Register/memory to segment register	1 0 0 0 1 1 1 0	mod 0 reg r/m		
Segment register to register/memory	1 0 0 0 1 1 0 0	mod 0 reg r/m		

Decodificación de Instrucciones (2 bytes)

Memory to accumulator	1 0 1 0 0 0 0 w	addr-low	addr-high
Accumulator to memory	1 0 1 0 0 0 1 w	addr-low	addr-high

Memoria al Acumulador:

MOV AX , [offset]

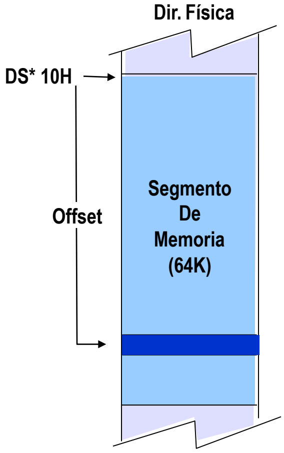
MOV AL , [offset]

Acumulador a Memoria:

MOV [offset] , AX

MOV [offset] , AL

Offset: Desplazamiento (valor de 16 bits)



Decodificación de Instrucciones

Memory to accumulator	1 0 1 0 0 0 0 w	addr-low	addr-high

W=1

MOV AX , [offset]

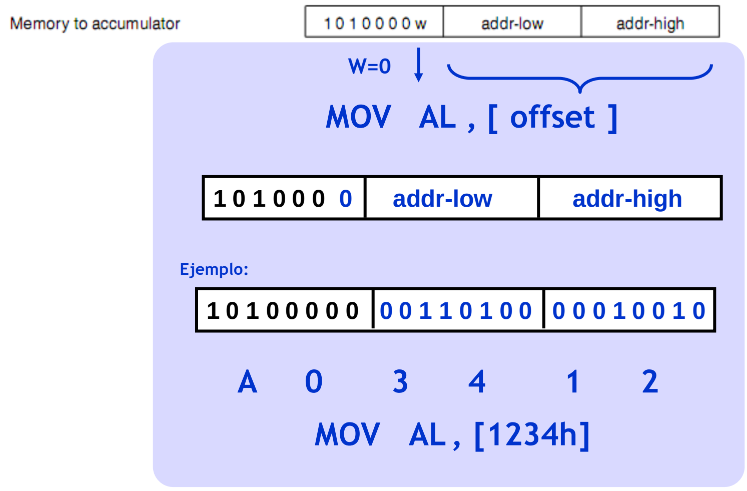
1 0 1 0 0 0 1	addr-low	addr-high

Ejemplo:

1 0 1 0 0 0 0 1	0 0 1 1 0 1 0 0	0 0 0 1 0 0 1 0
A 1	3 4	1 2

MOV AX , [1234h]

Decodificación de Instrucciones



Memory to accumulator	1 0 1 0 0 0 0 w	addr-low	addr-high

W=0

MOV AL , [offset]

1 0 1 0 0 0 0	addr-low	addr-high

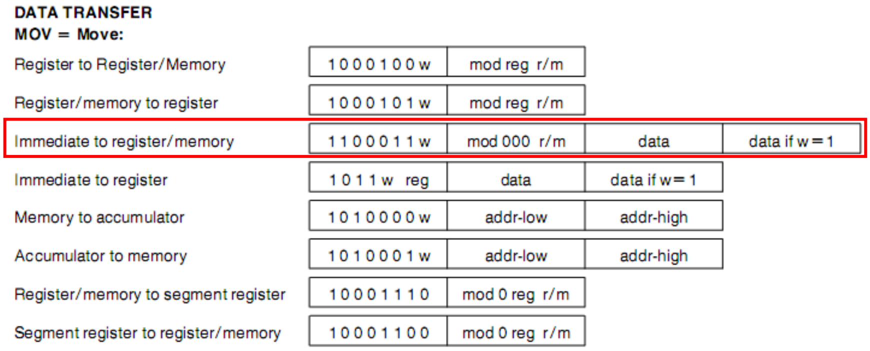
Ejemplo:

1 0 1 0 0 0 0 0	0 0 1 1 0 1 0 0	0 0 0 1 0 0 1 0

A 0 3 4 1 2

MOV AL , [1234h]

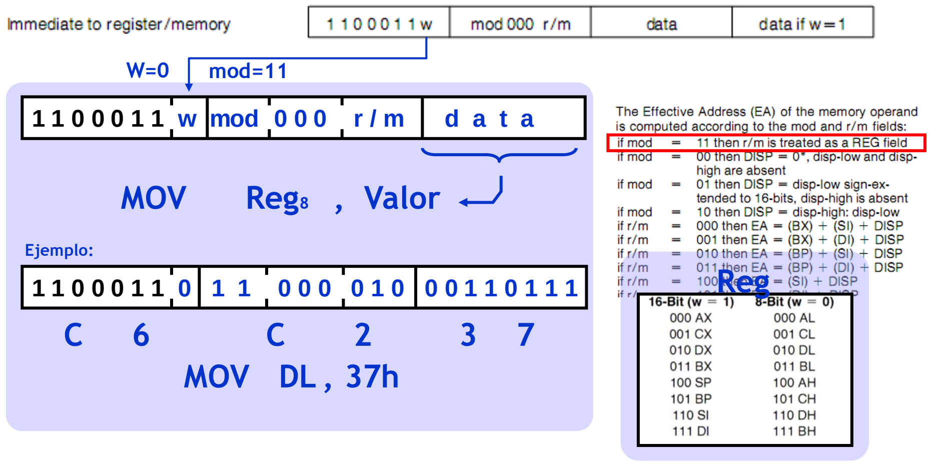
Decodificación de Instrucciones (2 bytes)



DATA TRANSFER
MOV = Move:

Register to Register/Memory	1 0 0 0 1 0 0 w	mod reg r/m		
Register/memory to register	1 0 0 0 1 0 1 w	mod reg r/m		
Immediate to register/memory	1 1 0 0 0 1 1 w	mod 000 r/m	data	data if w = 1
Immediate to register	1 0 1 1 w reg	data	data if w = 1	
Memory to accumulator	1 0 1 0 0 0 0 w	addr-low	addr-high	
Accumulator to memory	1 0 1 0 0 0 1 w	addr-low	addr-high	
Register/memory to segment register	1 0 0 0 1 1 1 0	mod 0 reg r/m		
Segment register to register/memory	1 0 0 0 1 1 0 0	mod 0 reg r/m		

Continuación…



Immediate to register/memory

1 1 0 0 0 1 1 w	mod 000 r/m	data	data if w = 1

W=0 mod=11

1 1 0 0 0 1 1 w mod 0 0 0 r / m d a t a

MOV Reg8 , Valor

Ejemplo:

1 1 0 0 0 1 1 0 1 1 0 0 0 0 1 0 0 0 1 1 0 1 1 1

C 6 C 2 3 7

MOV DL , 37h

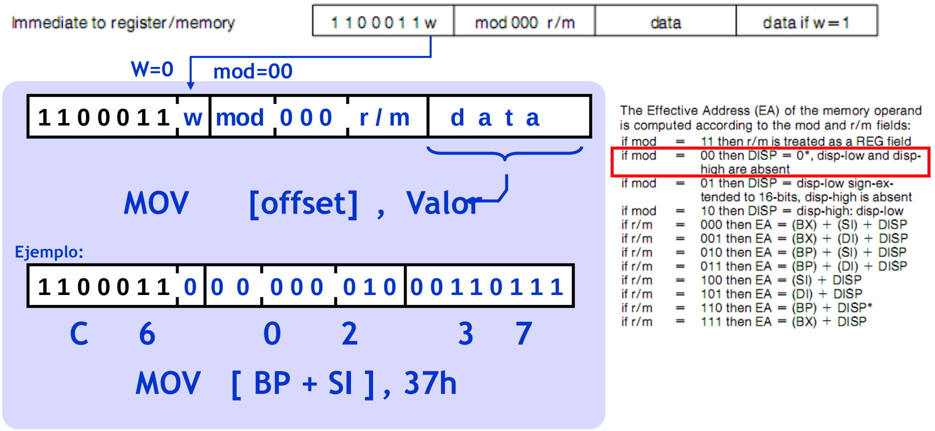
The Effective Address (EA) of the memory operand is computed according to the mod and r/m fields:

if mod = 11 then r/m is treated as a REG field
if mod = 00 then DISP = 0*, disp-low and disp-high are absent
if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
if mod = 10 then DISP = disp-high: disp-low
if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/

Reg

16-Bit (w = 1)	8-Bit (w = 0)
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
100 SP	100 AH
101 BP	101 CH
110 SI	110 DH
111 DI	111 BH

Continuación…



Immediate to register/memory

1 1 0 0 0 1 1 w	mod 000 r/m	data	data if w = 1

W=0 mod=00

1 1 0 0 0 1 1 w mod 0 0 0 r / m d a t a

MOV [offset] , Valor

Ejemplo:

1 1 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 1 1 0 1 1 1

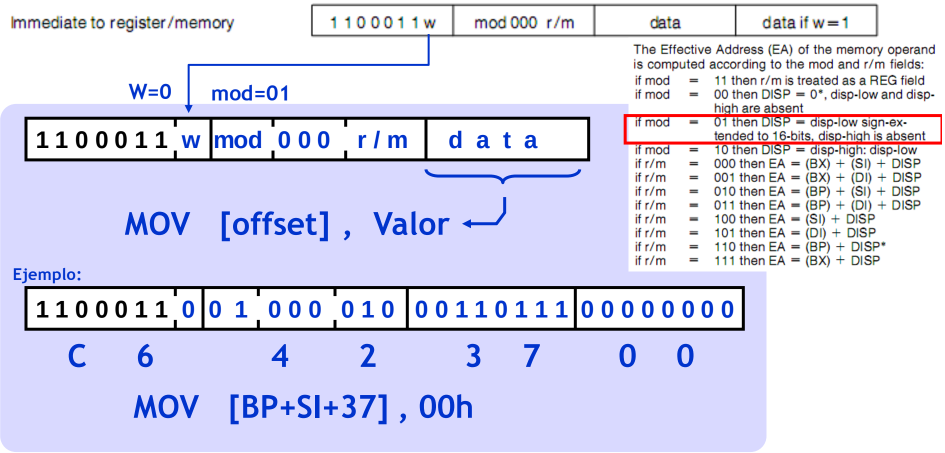
C 6 0 2 3 7

MOV [BP + SI] , 37h

The Effective Address (EA) of the memory operand is computed according to the mod and r/m fields:

if mod = 11 then r/m is treated as a REG field
if mod = 00 then DISP = 0*, disp-low and disp-high are absent
if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
if mod = 10 then DISP = disp-high: disp-low
if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/m = 101 then EA = (DI) + DISP
if r/m = 110 then EA = (BP) + DISP*
if r/m = 111 then EA = (BX) + DISP

Continuación...



Immediate to register/memory

1 1 0 0 0 1 1 w	mod 000 r/m	data	data if w=1

The Effective Address (EA) of the memory operand is computed according to the mod and r/m fields:

if mod	=	11 then r/m is treated as a REG field
if mod	=	00 then DISP = 0*, disp-low and disp-high are absent
if mod	=	01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
if mod	=	10 then DISP = disp-high: disp-low
if r/m	=	000 then EA = (BX) + (SI) + DISP
if r/m	=	001 then EA = (BX) + (DI) + DISP
if r/m	=	010 then EA = (BP) + (SI) + DISP
if r/m	=	011 then EA = (BP) + (DI) + DISP
if r/m	=	100 then EA = (SI) + DISP
if r/m	=	101 then EA = (DI) + DISP
if r/m	=	110 then EA = (BP) + DISP*
if r/m	=	111 then EA = (BX) + DISP

1 1 0 0 0 1 1	w	mod	0 0 0	r / m	d a t a

MOV [offset] , Valor

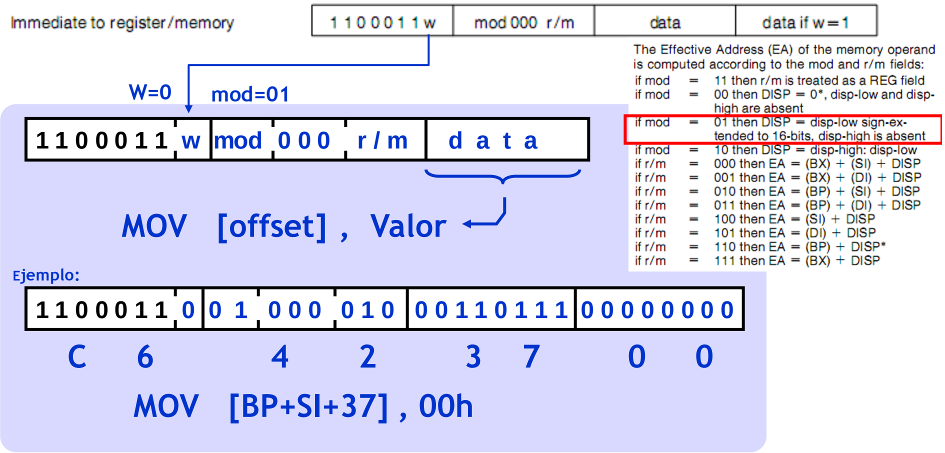
Ejemplo:

1 1 0 0 0 1 1	0	0 1	0 0 0	0 1 0	0 0 1 1 0 1 1 1	0 0 0 0 0 0 0 0

C 6 4 2 3 7 0 0

MOV [BP+SI+37] , 00h

Continuación...



Immediate to register/memory

1 1 0 0 0 1 1 w	mod 000 r/m	data	data if w=1

The Effective Address (EA) of the memory operand is computed according to the mod and r/m fields:

if mod	=	11 then r/m is treated as a REG field
if mod	=	00 then DISP = 0*, disp-low and disp-high are absent
if mod	=	01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
if mod	=	10 then DISP = disp-high: disp-low
if r/m	=	000 then EA = (BX) + (SI) + DISP
if r/m	=	001 then EA = (BX) + (DI) + DISP
if r/m	=	010 then EA = (BP) + (SI) + DISP
if r/m	=	011 then EA = (BP) + (DI) + DISP
if r/m	=	100 then EA = (SI) + DISP
if r/m	=	101 then EA = (DI) + DISP
if r/m	=	110 then EA = (BP) + DISP*
if r/m	=	111 then EA = (BX) + DISP

W=0
mod=01

1 1 0 0 0 1 1	w	mod	0 0 0	r / m	d a t a

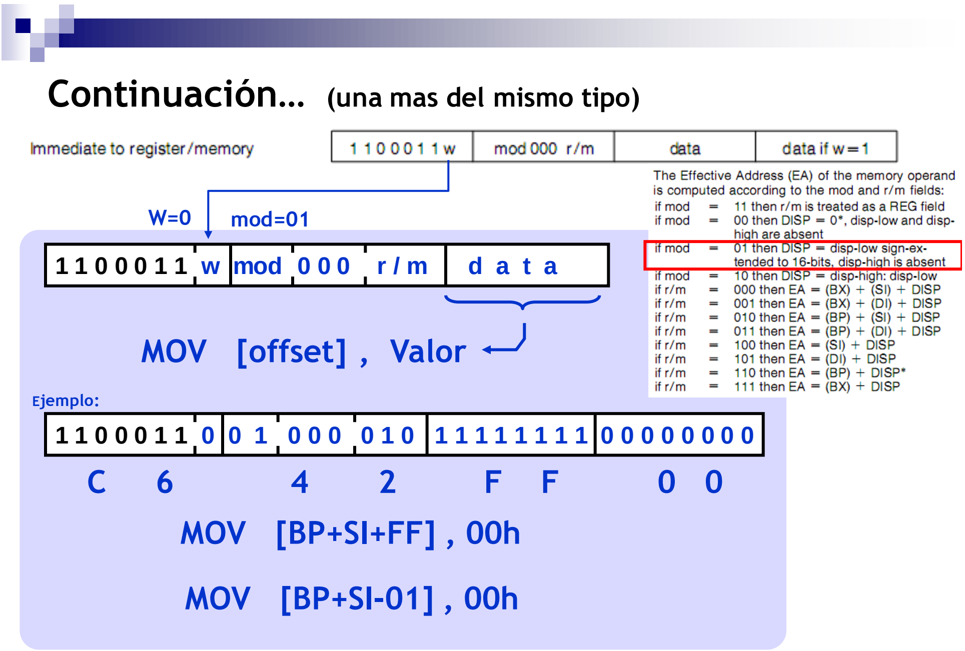
MOV [offset] , Valor

Ejemplo:

1 1 0 0 0 1 1	0	0 1	0 0 0	0 1 0	0 0 1 1 0 1 1 1	0 0 0 0 0 0 0 0

C 6 4 2 3 7 0 0

MOV [BP+SI+37] , 00h



Continuación... (una mas del mismo tipo)

Immediate to register/memory	1 1 0 0 0 1 1 w	mod 000 r/m	data	data if w = 1

The Effective Address (EA) of the memory operand is computed according to the mod and r/m fields:

if mod	=	11 then r/m is treated as a REG field
if mod	=	00 then DISP = 0*, disp-low and disp-high are absent
if mod	=	01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
if mod	=	10 then DISP = disp-high: disp-low
if r/m	=	000 then EA = (BX) + (SI) + DISP
if r/m	=	001 then EA = (BX) + (DI) + DISP
if r/m	=	010 then EA = (BP) + (SI) + DISP
if r/m	=	011 then EA = (BP) + (DI) + DISP
if r/m	=	100 then EA = (SI) + DISP
if r/m	=	101 then EA = (DI) + DISP
if r/m	=	110 then EA = (BP) + DISP*
if r/m	=	111 then EA = (BX) + DISP

W=0 mod=01

1 1 0 0 0 1 1	w	mod	0 0 0	r / m	d a t a

MOV [offset] , Valor

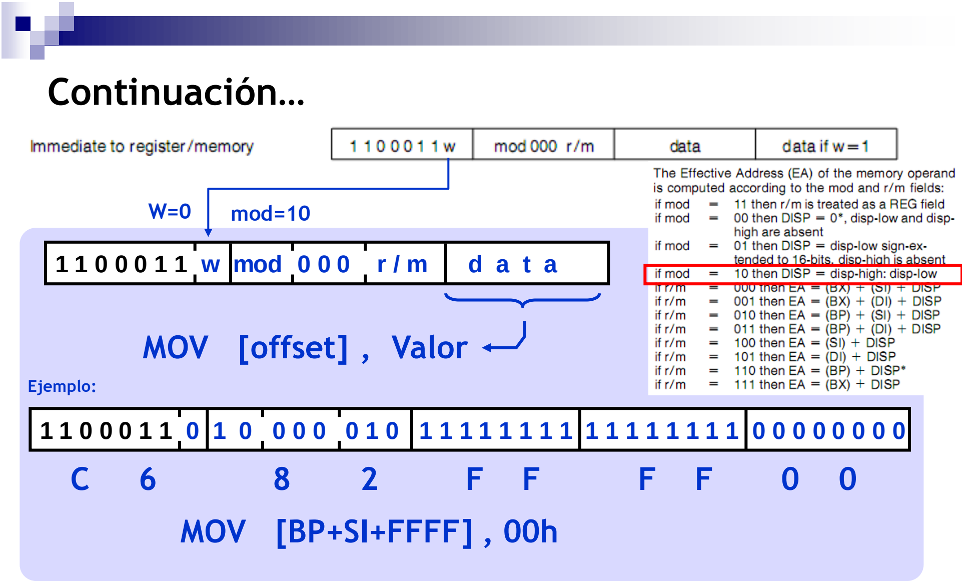
Ejemplo:

1 1 0 0 0 1 1	0	0 1	0 0 0	0 1 0	1 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0

C 6 4 2 F F 0 0

MOV [BP+SI+FF] , 00h

MOV [BP+SI-01] , 00h



Continuación...

Immediate to register/memory	1 1 0 0 0 1 1 w	mod 000 r/m	data	data if w = 1

The Effective Address (EA) of the memory operand is computed according to the mod and r/m fields:

if mod	=	11 then r/m is treated as a REG field
if mod	=	00 then DISP = 0*, disp-low and disp-high are absent
if mod	=	01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
if mod	=	10 then DISP = disp-high: disp-low
if r/m	=	000 then EA = (BX) + (SI) + DISP
if r/m	=	001 then EA = (BX) + (DI) + DISP
if r/m	=	010 then EA = (BP) + (SI) + DISP
if r/m	=	011 then EA = (BP) + (DI) + DISP
if r/m	=	100 then EA = (SI) + DISP
if r/m	=	101 then EA = (DI) + DISP
if r/m	=	110 then EA = (BP) + DISP*
if r/m	=	111 then EA = (BX) + DISP

W=0 mod=10

1 1 0 0 0 1 1	w	mod	0 0 0	r / m	d a t a

MOV [offset] , Valor

Ejemplo:

1 1 0 0 0 1 1	0	1 0	0 0 0	0 1 0	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0

C 6 8 2 F F F F 0 0

MOV [BP+SI+FFFF] , 00h

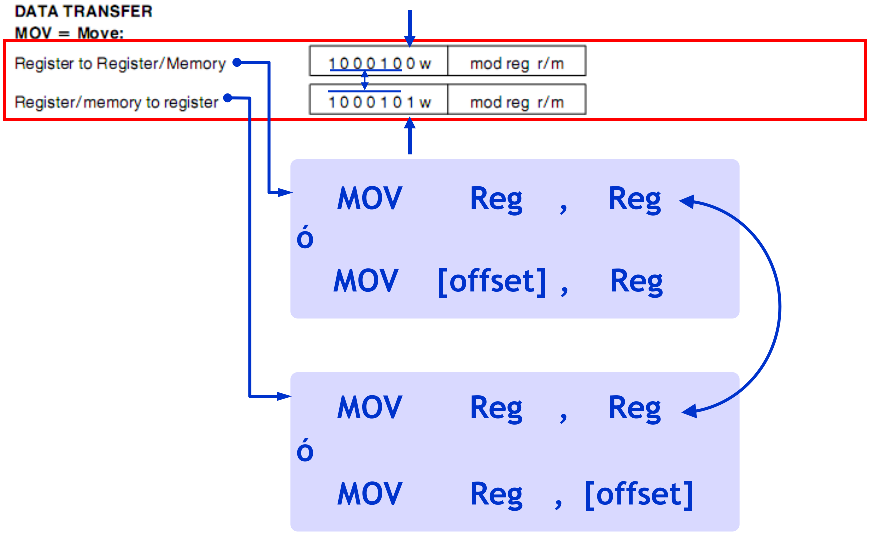
Decodificación de Instrucciones (2 bytes)

DATA TRANSFER
MOV = Move:

Register to Register/Memory	1 0 0 0 1 0 0 w	mod reg r/m		
Register/memory to register	1 0 0 0 1 0 1 w	mod reg r/m		
Immediate to register/memory	1 1 0 0 0 1 1 w	mod 000 r/m	data	data if w = 1
Immediate to register	1 0 1 1 w reg	data	data if w = 1	
Memory to accumulator	1 0 1 0 0 0 0 w	addr-low	addr-high	
Accumulator to memory	1 0 1 0 0 0 1 w	addr-low	addr-high	
Register/memory to segment register	1 0 0 0 1 1 1 0	mod 0 reg r/m		
Segment register to register/memory	1 0 0 0 1 1 0 0	mod 0 reg r/m		

Decodificación de Instrucciones (2 bytes)

DATA TRANSFER
MOV = Move:

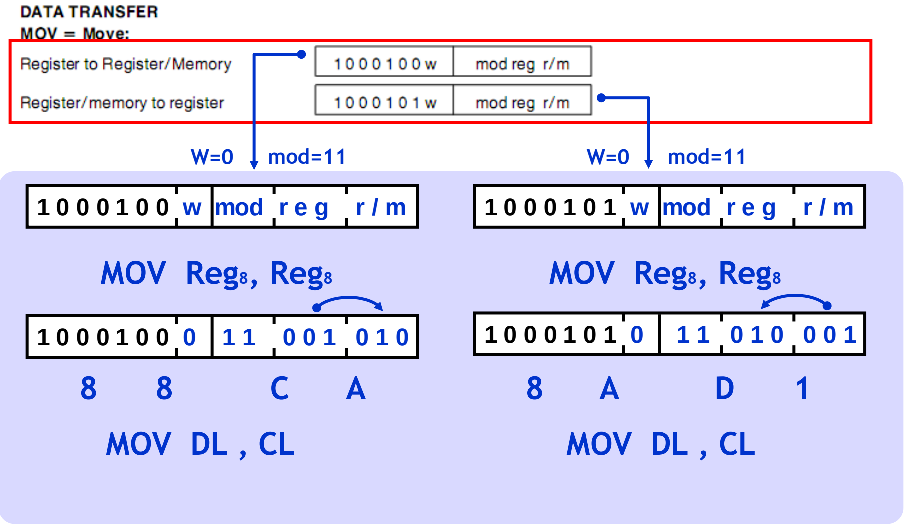


Register to Register/Memory	1 0 0 0 1 0 0 w	mod reg r/m
Register/memory to register	1 0 0 0 1 0 1 w	mod reg r/m

MOV Reg , Reg
ó
MOV [offset] , Reg

MOV Reg , Reg
ó
MOV Reg , [offset]

Decodificación de Instrucciones (2 bytes)



DATA TRANSFER
MOV = Move:

Register to Register/Memory	1 0 0 0 1 0 0 w	mod reg r/m
Register/memory to register	1 0 0 0 1 0 1 w	mod reg r/m

1 0 0 0 1 0 0	w	mod	r e g	r / m

MOV Reg8, Reg8

1 0 0 0 1 0 0	0	1 1	0 0 1	0 1 0

8 8 C A

MOV DL , CL

1 0 0 0 1 0 1	w	mod	r e g	r / m

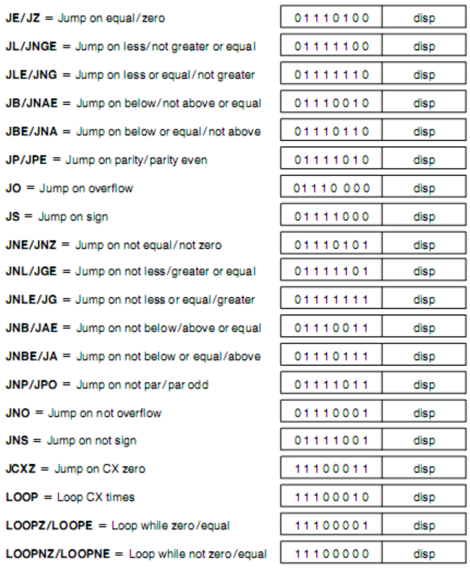
MOV Reg8, Reg8

1 0 0 0 1 0 1	0	1 1	0 1 0	0 0 1

8 A D 1

MOV DL , CL

Decodificación de Instrucciones JMP condicional

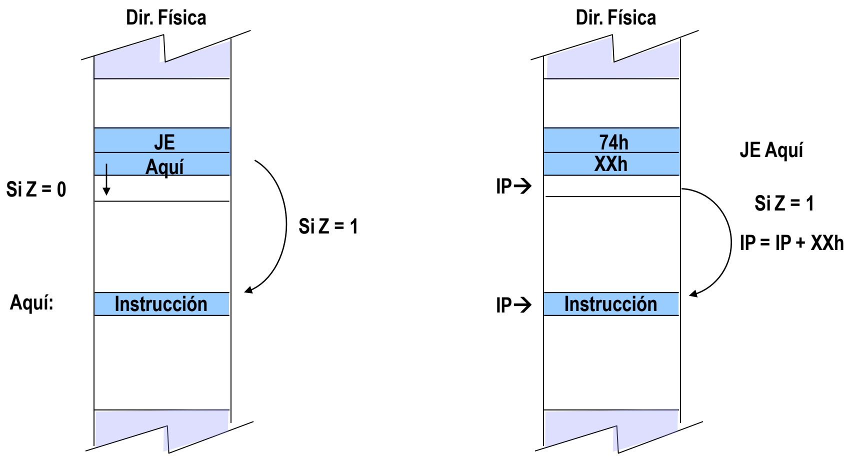


JE/JZ = Jump on equal/zero	0 1 1 1 0 1 0 0	disp
JL/JNGE = Jump on less/not greater or equal	0 1 1 1 1 1 0 0	disp
JLE/JNG = Jump on less or equal/not greater	0 1 1 1 1 1 1 0	disp
JB/JNAE = Jump on below/not above or equal	0 1 1 1 0 0 1 0	disp
JBE/JNA = Jump on below or equal/not above	0 1 1 1 0 1 1 0	disp
JP/JPE = Jump on parity/parity even	0 1 1 1 1 0 1 0	disp
JO = Jump on overflow	01 1 1 0 0 0 0	disp
JS = Jump on sign	0 1 1 1 1 0 0 0	disp
JNE/JNZ = Jump on not equal/not zero	0 1 1 1 0 1 0 1	disp
JNL/JGE = Jump on not less/greater or equal	0 1 1 1 1 1 0 1	disp
JNLE/JG = Jump on not less or equal/greater	0 1 1 1 1 1 1 1	disp
JNB/JAE = Jump on not below/above or equal	0 1 1 1 0 0 1 1	disp
JNBE/JA = Jump on not below or equal/above	0 1 1 1 0 1 1 1	disp
JNP/JPO = Jump on not par/par odd	0 1 1 1 1 0 1 1	disp
JNO = Jump on not overflow	0 1 1 1 0 0 0 1	disp
JNS = Jump on not sign	0 1 1 1 1 0 0 1	disp
JCXZ = Jump on CX zero	1 1 1 0 0 0 1 1	disp
LOOP = Loop CX times	1 1 1 0 0 0 1 0	disp
LOOPZ/LOOPE = Loop while zero/equal	1 1 1 0 0 0 0 1	disp
LOOPNZ/LOOPNE = Loop while not zero/equal	1 1 1 0 0 0 0 0	disp

Decodificación de Instrucciones JMP y CALL

JE/JZ = Jump on equal/zero

01110100	disp



Decodificación de Instrucciones JMP y CALL

JMP = Unconditional jump:

Short/long	11101011	disp-low		
Direct within segment	11101001	disp-low	disp-high	
Register/memory indirect within segment	11111111	mod 100 r/m		
Direct intersegment	11101010	segment offset		
		segment selector		
Indirect intersegment	11111111	mod 101 r/m		(mod ≠ 11)