

# Explicación de práctica N° 5

## Lenguaje Assembly

# Instrucciones

El procesador:

- sólo puede interpretar cadenas de 1 y 0
- resuelve operaciones muy elementales (suma, resta, lógicas, transferencia de datos)
- tiene una capacidad acotada para el tamaño de los datos

El usuario:

- difícilmente recuerde el código binario asociado a cada instrucción
- requiere resolver operaciones muy complejas
- utiliza datos de tamaño variable y grande

# Lenguaje Assembly

- propone una sintaxis y semántica cercanas al lenguaje “máquina”
- puede implementar estructuras de control y llamados a procedimientos
- se pueden tratar datos de tamaño mayor a una palabra en memoria con diferentes técnicas

*Uso de mnemónicos:*

*MOV dato, 10 => Establecer valor 10 en “variable” dato.*

*Operaciones aritmético-lógicas + saltos condicionales según FLAGS, llamados a subrutina, pasaje de parámetros.*

*Modos de direccionamiento (acceso a sucesivas direcciones de memoria referenciadas en registro)*

# Set de instrucciones

Las instrucciones son cadenas binarias que indican al procesador qué hacer...

→ Cada arquitectura define su propio set de instrucciones.

Cada instrucción tiene una representación binaria diferente...

→ Muchas instrucciones = códigos de instrucción más largos

Y cada modo de direccionamiento también es una instrucción diferente...

→ Más modos de direccionamiento = códigos de instrucción más largos

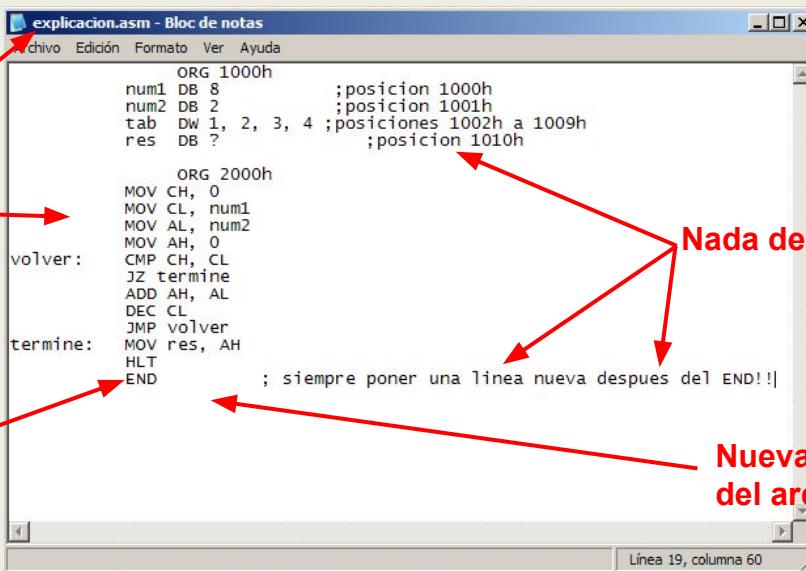
# Arquitectura SX88

El simulador que se utilizará en la práctica tiene una arquitectura con las siguientes características:

- 4 registros de uso general de 16 bits: AX, BX, CX, DX  
Cada uno puede dividirse en dos registros de 8 bits: AH, AL, BH, BL, etc.
- Tamaño de instrucción variable (en múltiplos de 8 bits).
- Tamaño de palabra de 16 bits.
- Bus de datos de 8 bits de ancho (necesita 2 ciclos para recuperar una palabra).
- Direcciones de 16 bits ( $2^{16} = 64M$  direcciones \* 1 byte = 64MB de memoria).
- Modos de direccionamiento:
  - Inmediato
  - Directo (registro)
  - Directo (memoria)
  - Indirecto por registro

# Programando para SX88

Para ejecutar un programa en el simulador MSX88 basta con editar el código fuente en cualquier editor de texto sin formato, ensamblarlo y cargarlo.



The screenshot shows a Windows Notepad window titled "explicacion.asm - Bloc de notas". The code is written in 80x86 assembly language:

```
ORG 1000h
num1 DB 8      ;posicion 1000h
num2 DB 2      ;posicion 1001h
tab DW 1, 2, 3, 4 ;posiciones 1002h a 1009h
res DB ?

ORG 2000h
MOV CH, 0
MOV CL, num1
MOV AL, num2
MOV AH, 0
CMP CH, CL
JZ termine
ADD AH, AL
DEC CL
JMP volver
MOV res, AH
HLT
END

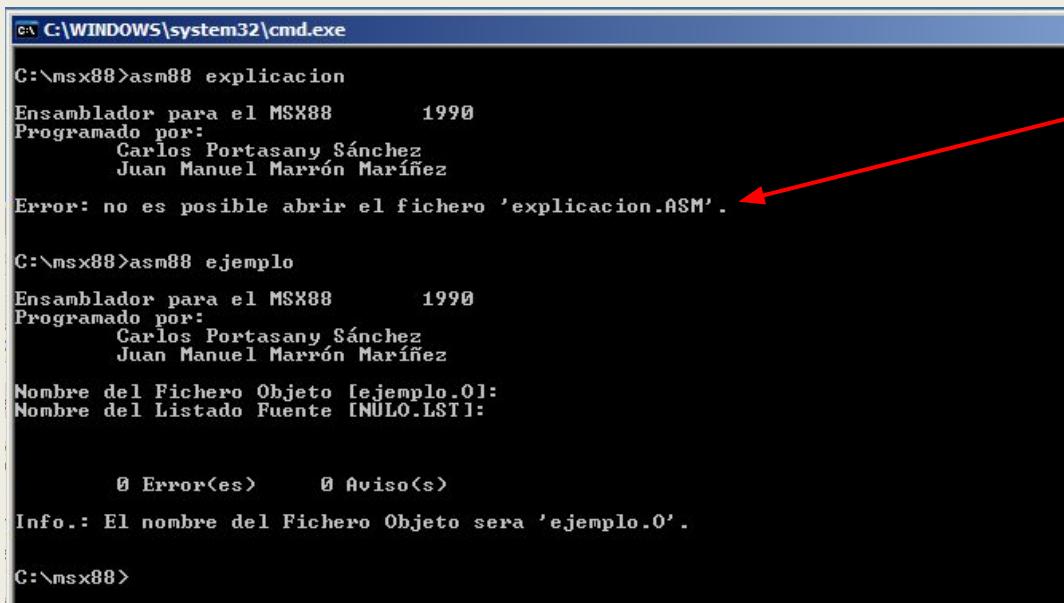
volver:
termine:
```

Annotations in red text with arrows pointing to specific parts of the code:

- "Extensión del archivo: .asm" points to the file tab at the top left.
- "Código fuente" points to the assembly code area.
- "Nada de ortografía castellana" points to the comments in the code.
- "El código termina con la directiva END (no es una instrucción)" points to the "END" directive at the end of the code.
- "; siempre poner una linea nueva despues del END!!" points to the line separator after the "END" directive.
- "Nueva línea (ENTER) al final del archivo (si no, da error)" points to the bottom right corner of the window, indicating where to press Enter.

# Programando para SX88

El siguiente paso es ensamblar el código creado:



```
C:\WINDOWS\system32\cmd.exe
C:\msx88>asm88 explicacion
Ensamblador para el MSX88      1990
Programado por:
    Carlos Portasany Sánchez
    Juan Manuel Marrón Maríñez
Error: no es posible abrir el fichero 'explicacion.ASM'.

C:\msx88>asm88 ejemplo
Ensamblador para el MSX88      1990
Programado por:
    Carlos Portasany Sánchez
    Juan Manuel Marrón Maríñez
Nombre del Fichero Objeto [ejemplo.0]:
Nombre del Listado Fuente [NULO.LST]:

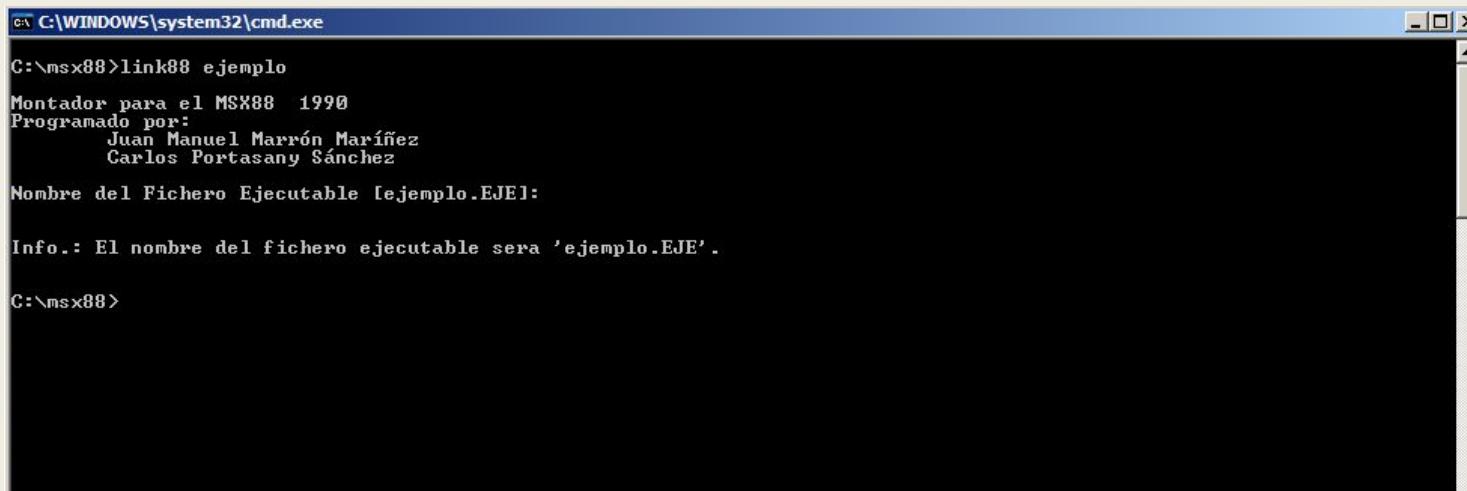
          0 Error(es)      0 Aviso(s)
Info.: El nombre del Fichero Objeto sera 'ejemplo.0'.

C:\msx88>
```

El nombre del archivo  
debe tener como máximo 8  
caracteres

# Programando para SX88

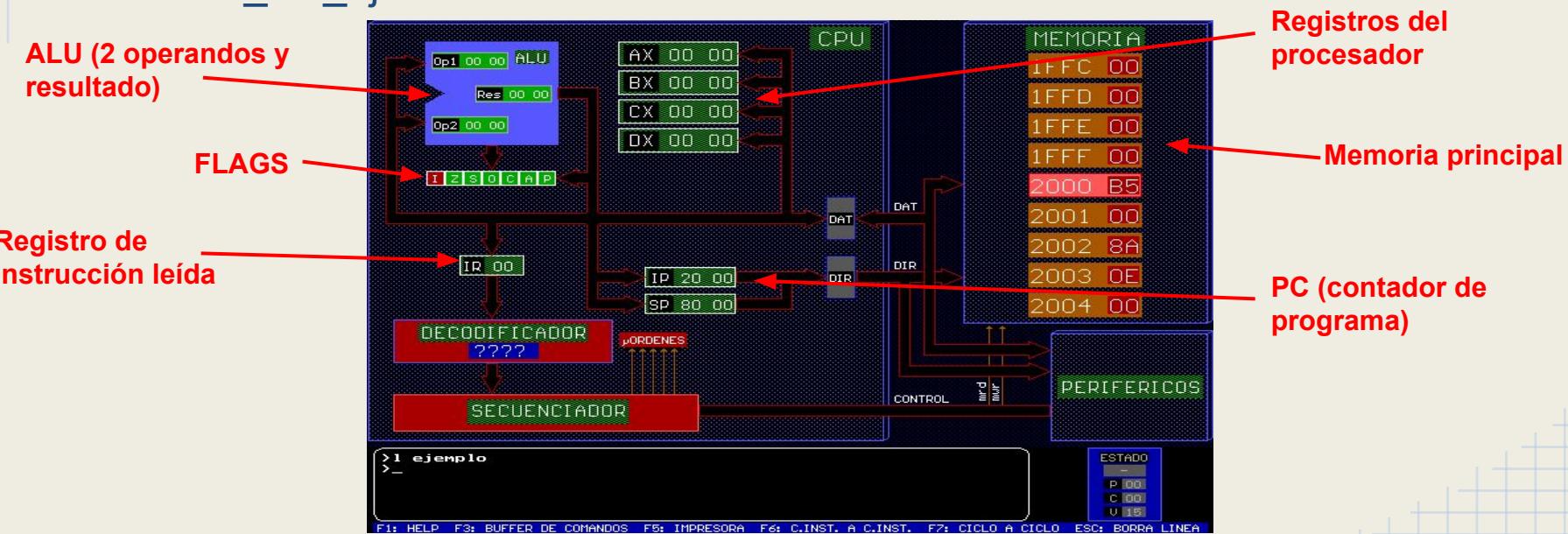
Enlazar los archivos objeto creados (paso requerido pero no hace nada):



```
C:\WINDOWS\system32\cmd.exe
C:\msx88>link88 ejemplo
Montador para el MSX88 1990
Programado por:
    Juan Manuel Marrón Maríñez
    Carlos Portasany Sánchez
Nombre del Fichero Ejecutable [ejemplo.EJE]:
Info.: El nombre del fichero ejecutable sera 'ejemplo.EJE'.
C:\msx88>
```

# Simulador MSX88

Por último, ejecutar la aplicación MSX88.EXE e ingresar el comando:  
L nombre\_del\_ejecutable



# Estructura de un programa

- El procesador siempre busca la primera instrucción en una dirección predeterminada (2000h).
- Por *convención*, los datos se ubican a partir de la dirección 1000h.
- La directiva ORG indica a partir de qué dirección se ubican las siguientes instrucciones y declaraciones.
- Los datos y las instrucciones pueden estar en cualquier posición de memoria, siempre que ubiquemos en 2000h la primera instrucción del programa (o bien cambiando el registro IP del procesador antes de ejecutar).

# Ejemplo 1

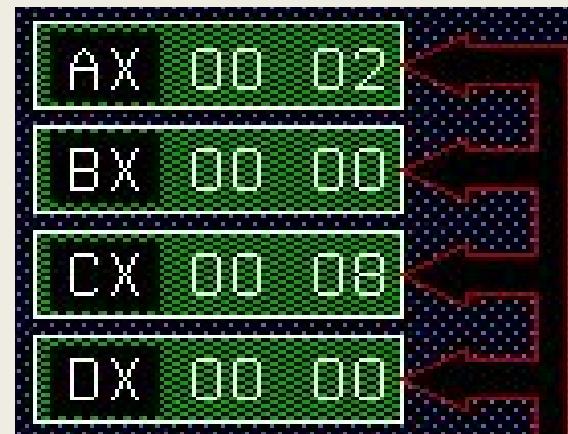
```
ORG 1000h  
num1 DB 8      ;posicion 1000h  
num2 DB 2      ;posicion 1001h  
tab DW 1, 2, 3, 4 ;posiciones 1002h a 1009h  
res DB ?  
END
```

MEMORIA	
1000	08
1001	02
1002	01
1003	00
1004	02
1005	00
1006	03
1007	00
1008	04

# Ejemplo 1 (cont.)

```
ORG 1000h  
num1 DB 8      ;posicion 1000h  
num2 DB 2      ;posicion 1001h  
tab DW 1, 2, 3, 4 ;posiciones 1002h a 1009h  
res DB ?      ;posicion 1010h
```

```
ORG 2000h  
MOV CH, 0  
MOV CL, num1  
MOV AL, num2  
MOV AH, 0  
HLT  
END
```



# Ejemplo 2: representaciones

ORG 1000h

num1 DB -10 → Ca2: 10110 → Ext. signo (8 bits): 11110110  
F 6

num2 DW 0AA3Fh

num3 DB 01100100b  
6 4

MEMORIA	
1000	F6
1001	3F
1002	AA
1003	64

# Ejemplo 2 (cont. - registros de 8 bits)

```
ORG 1000h  
num1    DB -10  
num2    DW 0AA3Fh  
num3    DB 01100100b
```

```
→ ORG 2000h  
MOV AH, num1  
MOV DL, num2  
MOV CX, num2  
MOV AL, num3  
MOV BX, OFFSET num1  
MOV BH, [BX]  
HLT  
END
```

AX	F6	00
BX	00	00
CX	00	00
DX	00	00

# Ejemplo 2 (cont. - registros de 8 bits)

```
ORG 1000h  
num1    DB -10  
num2    DW 0AA3Fh  
num3    DB 01100100b
```

```
ORG 2000h  
MOV AH, num1  
→ MOV DL, num2  
MOV CX, num2  
MOV AL, num3  
MOV BX, OFFSET num1  
MOV BH, [BX]  
HLT  
END
```

AX	F6	00
BX	00	00
CX	00	00
DX	00	3F

# Ejemplo 2 (cont. - registros de 8 bits)

```
ORG 1000h  
num1    DB -10  
num2    DW 0AA3Fh  
num3    DB 01100100b
```

```
ORG 2000h  
MOV AH, num1  
MOV DL, num2  
→ MOV CX, num2  
MOV AL, num3  
MOV BX, OFFSET num1  
MOV BH, [BX]  
HLT  
END
```

AX	F6	00
BX	00	00
CX	AA	3F
DX	00	3F

# Ejemplo 2 (cont. - registros de 8 bits)

```
ORG 1000h  
num1    DB -10  
num2    DW 0AA3Fh  
num3    DB 01100100b
```

```
ORG 2000h  
MOV AH, num1  
MOV DL, num2  
MOV CX, num2  
MOV AL, num3  
MOV BX, OFFSET num1  
MOV BH, [BX]  
HLT  
END
```

AX	F6	64
BX	00	00
CX	AA	3F
DX	00	3F

# Ejemplo 2 (cont. - direccionamiento)

```
ORG 1000h  
num1    DB -10  
num2    DW 0AA3Fh  
num3    DB 01100100b
```

```
ORG 2000h  
MOV AH, num1  
MOV DL, num2  
MOV CX, num2  
MOV AL, num3  
→ MOV BX, OFFSET num1  
MOV BH, [BX]  
HLT  
END
```

AX	F6	64
BX	10	00
CX	AA	3F
DX	00	3F

# Ejemplo 2 (cont. - direccionamiento)

ORG 1000h

num1 DB -10

num2 DW 0AA3Fh

num3 DB 01100100b

ORG 2000h

MOV AH, num1

MOV DL, num2

MOV CX, num2

MOV AL, num3

MOV BX, OFFSET num1

→ MOV BH, [BX]

HLT

END

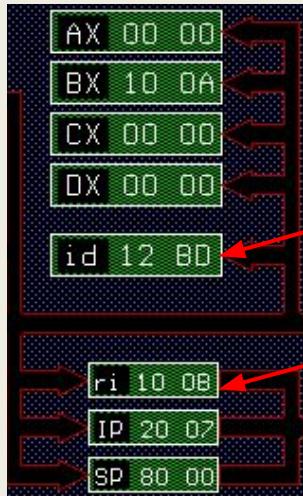
AX	F6	64
BX	F6	00
CX	AA	3F
DX	00	3F

# Ejemplo 3 (reg. temporales)

```
ORG 1000h  
num1 DB 8      ;posición 1000h  
num2 DB 2      ;posición 1001h  
tab DW 1, 2, 3, 4 ;posiciones 1002h a 1009h  
res DB ?       ;posición 1010h
```

```
ORG 2000h  
MOV BX, OFFSET res  
MOV WORD PTR [BX], 12BDh  
END
```

- Cada 'celda' de memoria tiene 1 byte.
- Cada dirección apunta a una 'celda'.
- Esta es una instrucción con dato inmediato.
- El tamaño de los datos que se utiliza lo determina el registro (de 8 ó de 16 bits).
- Esta instrucción además de dato inmediato, usa una referencia a memoria.
- La instrucción sería ambigua si no se hace explícito el tamaño con WORD PTR o BYTE PTR.



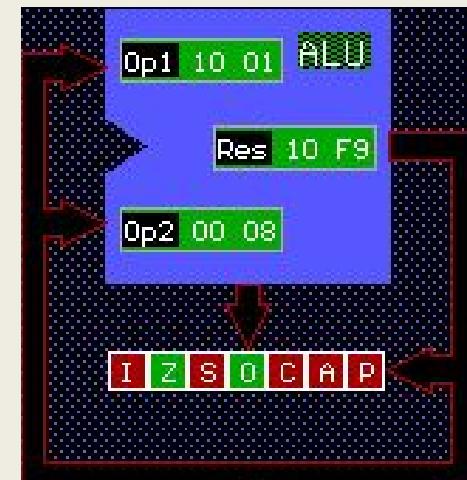
Registro temporal ID: dato inmediato que se va a transferir (las operaciones son entre registros / entre registro y memoria).

Registro temporal RI: dirección que se debe colocar en el bus de direcciones para transferir el dato (origen o destino en memoria principal de la operación).

# Flags

El procesador tiene un registro especial (FLAGS) en el que almacena el estado de las distintas banderas (8 en total) tras ejecutar determinadas operaciones.

- El registro solamente es afectado por las operaciones aritméticas y las operaciones lógicas que ejecuta la ALU.
- Las operaciones de transferencia de datos, transferencia de control, etc, no modifican los FLAGS.
- Hay una operación particular (CMP) que altera los FLAGS tras restar los operandos pero no almacena el resultado en ninguno de ellos.
- En la práctica serán de interés los bits Z, S, O, C de este registro.



# Transferencia de control

Las siguientes instrucciones alteran el flujo normal del programa modificando el registro IP según distintas condiciones:

<i>Instrucción</i>	<i>Condición</i>	<i>Opuesta</i>
JZ	Flag Zero = true	JNZ
JS	Flag Sign = true	JNS
JO	Flag Overflow = true	JNO
JC	Flag Carry = true	JNC
JMP	Incondicional	No aplica

# Ejemplo 4: estructuras de control

```
ORG 1000h
num1 DB 8
tabla DB ?

ORG 2000h
MOV CL, 0
MOV CH, num1
MOV BX, OFFSET tabla
MOV AL, 2
lazo: MOV [BX], AL
      ADD AL, AL
      INC BX
      INC CL
      CMP CL, CH
      JS lazo
      HLT
      END
```

tabla	AL	BX	CL	ZSOC	JS?
?	2	1001h	0	0000	-
2	4	1002h	1	0100	Sí
2, 4	8	1003h	2	0100	Sí
2, 4, 8	16	1004h	3	0100	Sí
2, 4, 8, 16	32	1005h	4	0100	Sí
2, 4, 8, 16, 32	64	1006h	5	0100	Sí
2, 4, 8, 16, 32, 64	128	1007h	6	0100	Sí
2, 4, 8, 16, 32, 64, 128	0	1008h	7	0100	Sí
2, 4, 8, 32, 64, 128, 0	0	1009h	8	1000	No

# Ejemplo 4: estructuras de control

MEMORIA	
1001	02
1002	04
1003	08
1004	10
1005	20
1006	40
1007	80
1008	00
1009	00

tabla	AL	BX	CL	ZSOC	JS?
?	2	1001h	0	0000	-
2	4	1002h	1	0100	Sí
2, 4	8	1003h	2	0100	Sí
2, 4, 8	16	1004h	3	0100	Sí
2, 4, 8, 16	32	1005h	4	0100	Sí
2, 4, 8, 16, 32	64	1006h	5	0100	Sí
2, 4, 8, 16, 32, 64	128	1007h	6	0100	Sí
2, 4, 8, 16, 32, 64, 128	0	1008h	7	0100	Sí
2, 4, 8, 32, 64, 128, 0	0	1009h	8	1000	No

# Estructura de un programa (cont.)

ORG 1500h  
num2 DB ?

ORG 1000h  
num1 DB ?

ORG 2100h  
HLT

ORG 2400  
JMP instr

ORG 2000h  
instr: JMP 2100h  
END

- *¿Qué instrucción se ejecuta primero?*
- *No importa el orden numérico de las directivas ORG. ¿Por qué?*
- *¿A qué posición de memoria hace referencia la etiqueta ‘instr’?*
- *La etiqueta está declarada después que la instrucción ‘JMP instr’. ¿Es correcta esta instrucción?*
- *¿En qué dirección de memoria se ubica esa instrucción?*

# Explicación de práctica N° 5

Lenguaje Assembly - 2da. parte

# Ejercicio 1 (datos)

ORG 1000H

NUM0 DB 0CAH

NUM1 DB 0

NUM2 DW ?

NUM3 DW 0ABCDH

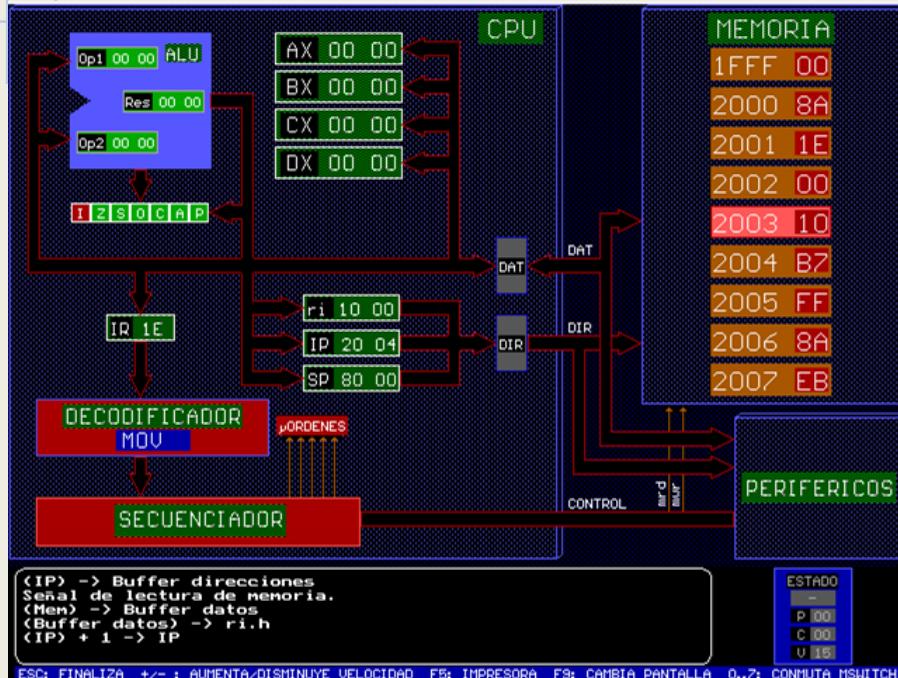
NUM4 DW ?

MEMORIA	
1000	CA
1001	00
1002	00
1003	07
1004	CD
1005	AB
1006	CD
1007	AB
1008	00

# Ejercicio 1 (instrucciones)

Dir.	Código máquina	Lín.	Código ensamble
2000	8A 1E 00 10	8	ORG 2000H
2004	B7 FF	9	MOV BL, NUM0
2006	8A EB	10	MOV BH, 0FFH
2008	8B C3	11	MOV CH, BL
200A	88 06 01 10	12	MOV AX, BX
200E	C7 06 02 10 34 12	13	MOV NUM1, AL
2014	BB 04 10	14	MOV NUM2, 1234H
2017	8A 17	15	MOV BX, OFFSET NUM3
2019	8B 07	16	MOV DL, [BX]
201B	BB 06 10	17	MOV AX, [BX]
201E	C7 07 06 10	18	MOV BX, 1006H
		19	MOV WORD PTR [BX], 1006H

# Ej. 1: MOV BL, NUM0



**Origen** Memoria (NUM0:1000h)

**Destino** Registro (BL)

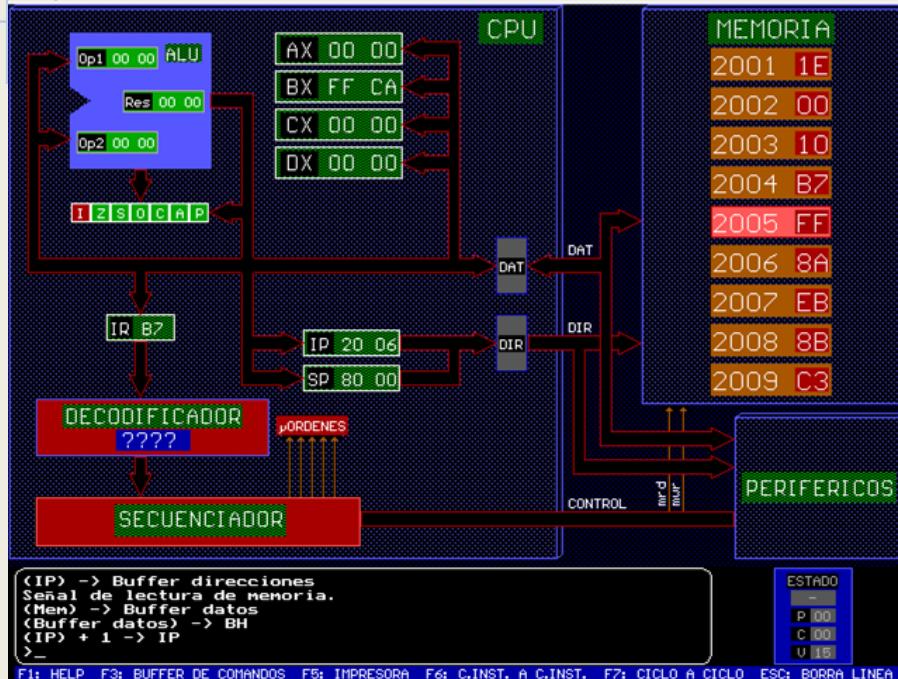
**Tamaño** 8 bits (BL)

**Modo** Directo

**Dato** CAh

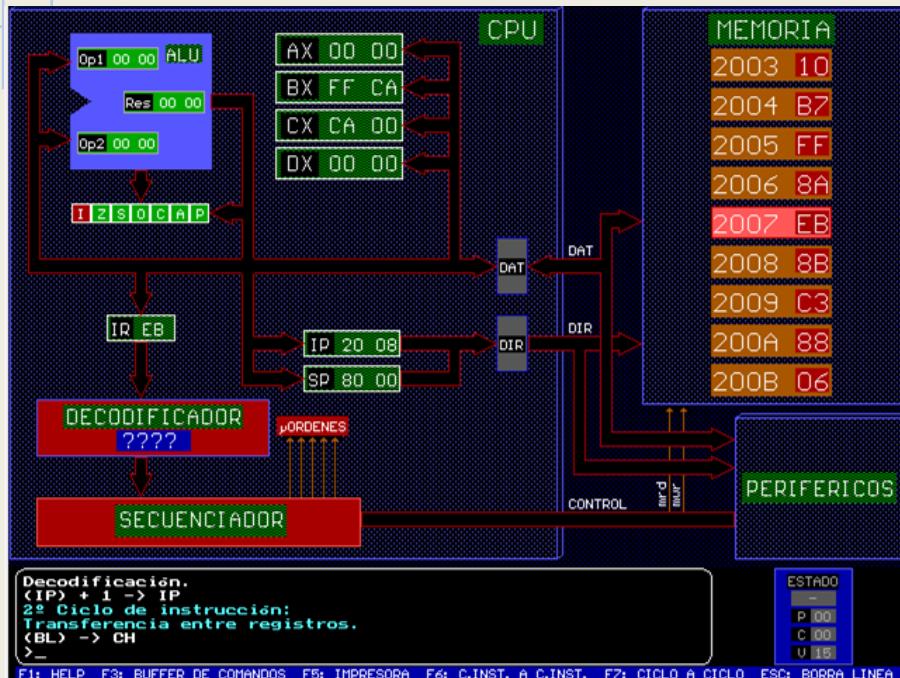
**Resultado** BX = 00CAh

# Ej. 1: MOV BH, OFFH



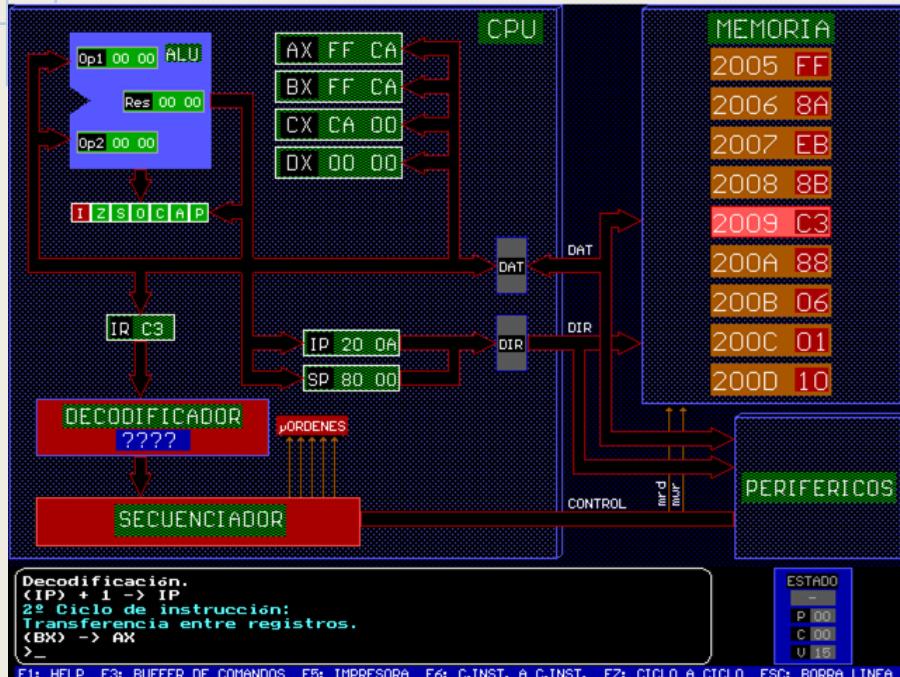
<b>Origen</b>	Memoria (Instrucción:2005h)
<b>Destino</b>	Registro (BH)
<b>Tamaño</b>	8 bits (BH)
<b>Modo</b>	Inmediato
<b>Dato</b>	FFh
<b>Resultado</b>	BX = FFCAh

# Ej. 1: MOV CH, BL



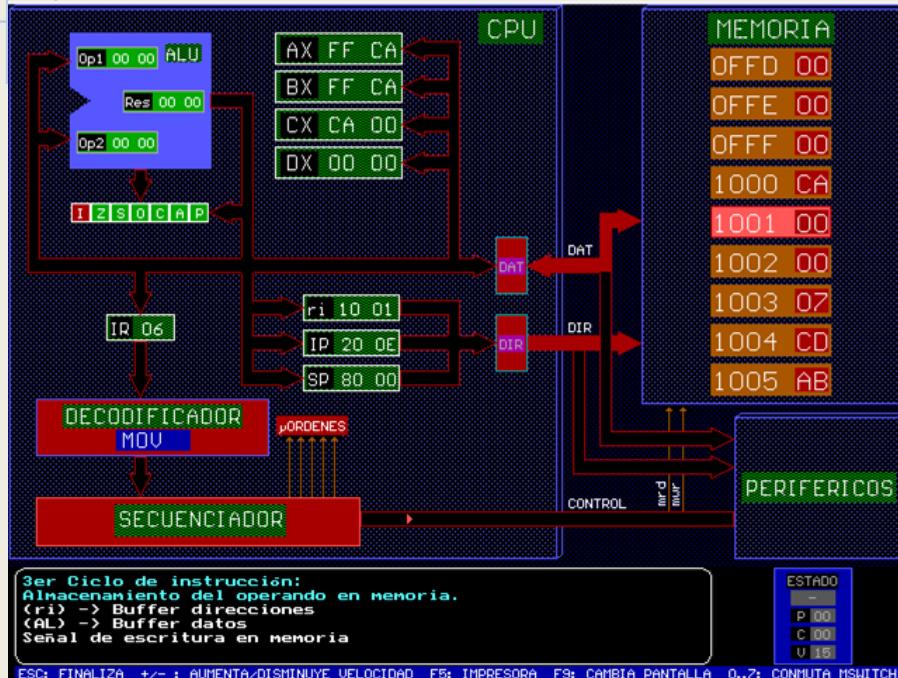
<b>Origen</b>	Registro (BL)
<b>Destino</b>	Registro (CH)
<b>Tamaño</b>	8 bits (CH)
<b>Modo</b>	Directo (registro)
<b>Dato</b>	CAh
<b>Resultado</b>	CX = CA00h

# Ej. 1: MOV AX, BX



<b>Origen</b>	Registro (BX)
<b>Destino</b>	Registro (AX)
<b>Tamaño</b>	16 bits (AX)
<b>Modo</b>	Directo (registro)
<b>Dato</b>	FFCAh
<b>Resultado</b>	AX = FFCAh

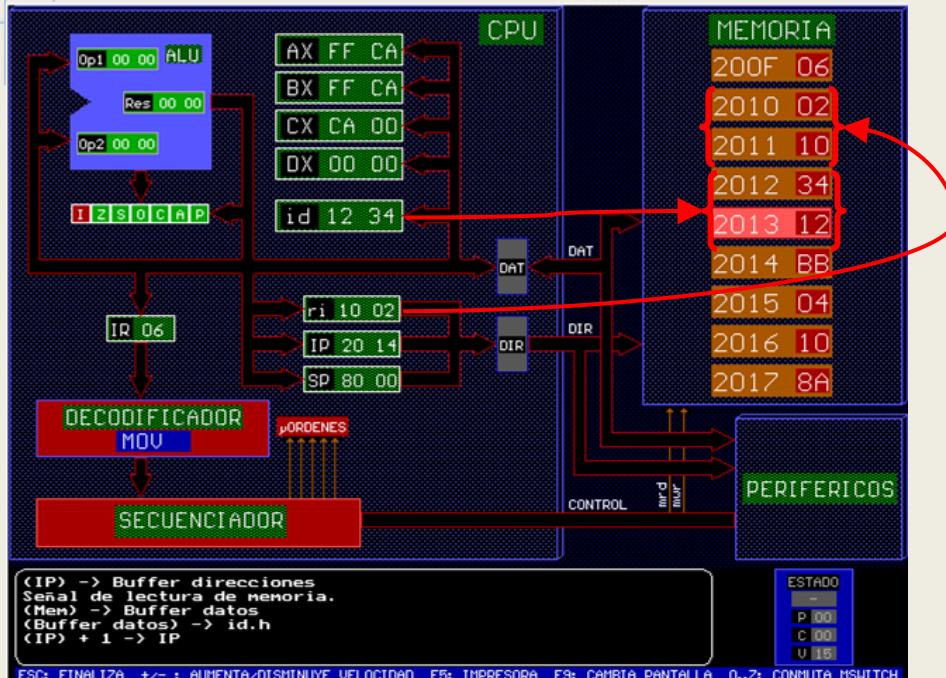
# Ej. 1: MOV NUM1, AL



<b>Origen</b>	Registro (AL)
<b>Destino</b>	Memoria (NUM1:1001h)
<b>Tamaño</b>	8 bits (AL)
<b>Modo</b>	Directo
<b>Dato</b>	CAh
<b>Resultado</b>	NUM1 = CAh

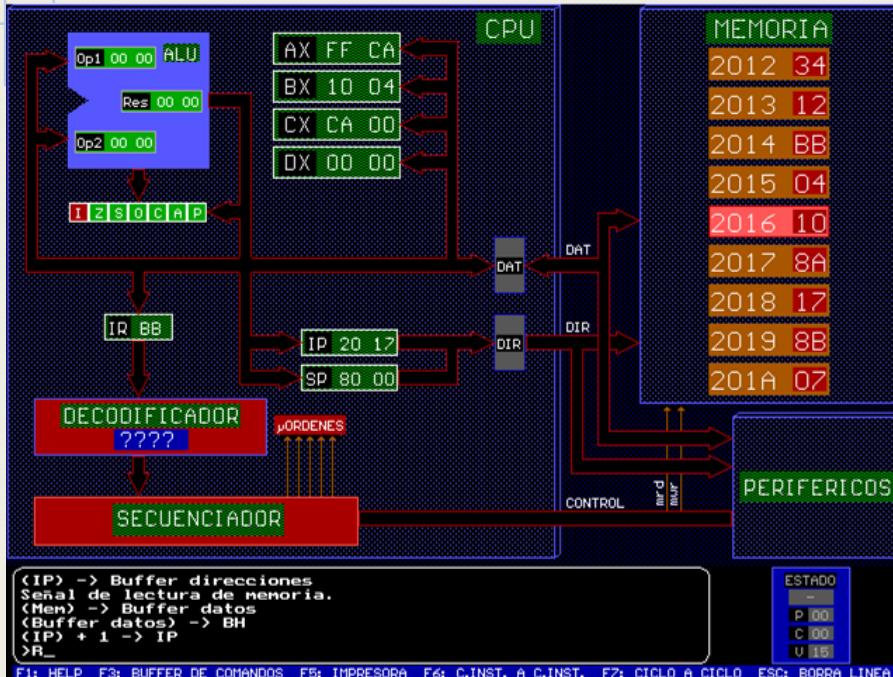
*Nota: la imagen muestra el momento en el que el dato es transferido al bus de datos con la dirección 1001h puesta en el bus de direcciones desde el registro temporal RI.*

# Ej. 1: MOV NUM2, 1234H



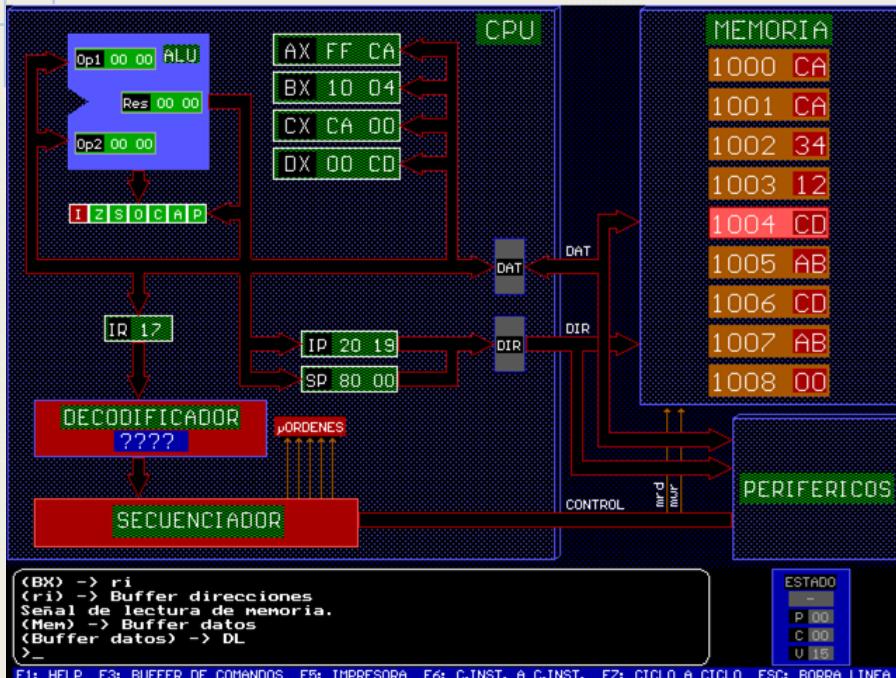
<b>Origen</b>	Memoria (Instrucción:2012h)
<b>Destino</b>	Memoria (NUM2:1002h)
<b>Tamaño</b>	16 bits (AX)
<b>Modo</b>	Inmediato
<b>Dato</b>	1234h
<b>Resultado</b>	NUM2 = 1234h

# Ej. 1: MOV BX, OFFSET NUM3



<b>Origen</b>	Memoria (Instrucción: 2015h)
<b>Destino</b>	Registro (BX)
<b>Tamaño</b>	16 bits (BX)
<b>Modo</b>	Inmediato
<b>Dato</b>	1004h
<b>Resultado</b>	$BX = 1004h$

# Ej. 1: MOV DL, [BX]



Origen

Memoria (\*BX:1004h)

Destino

Registro (DL)

Tamaño

8 bits (DL)

Modo

Indirecto por registro (BX)

Dato

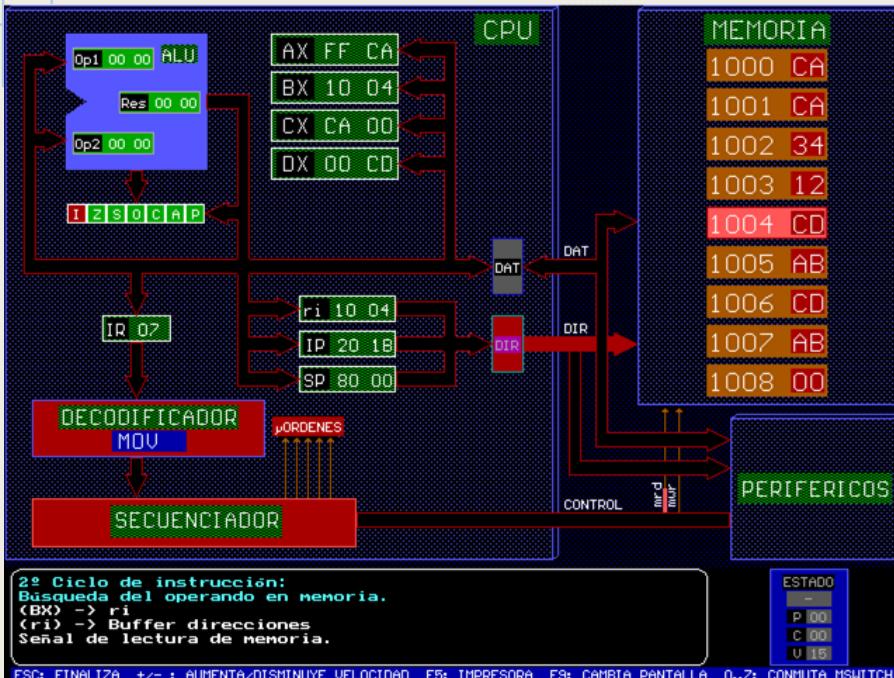
CDh

Resultado

DX = 00CDh

*Nota: el contenido del registro BX (1004h) es transferido al registro temporal RI para poder direccionar a esa posición de memoria.*

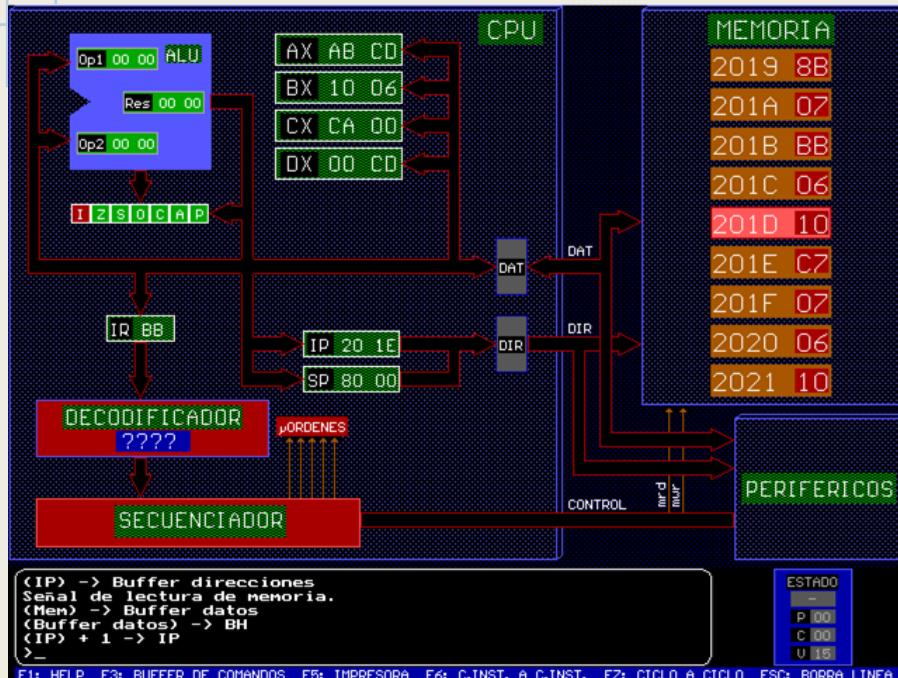
# Ej. 1: MOV AX, [BX]



<b>Origen</b>	Memoria (*BX:1004h)
<b>Destino</b>	Registro (AX)
<b>Tamaño</b>	16 bits (AX)
<b>Modo</b>	Indirecto por registro (BX)
<b>Dato</b>	ABCDh
<b>Resultado</b>	AX = ABCDh

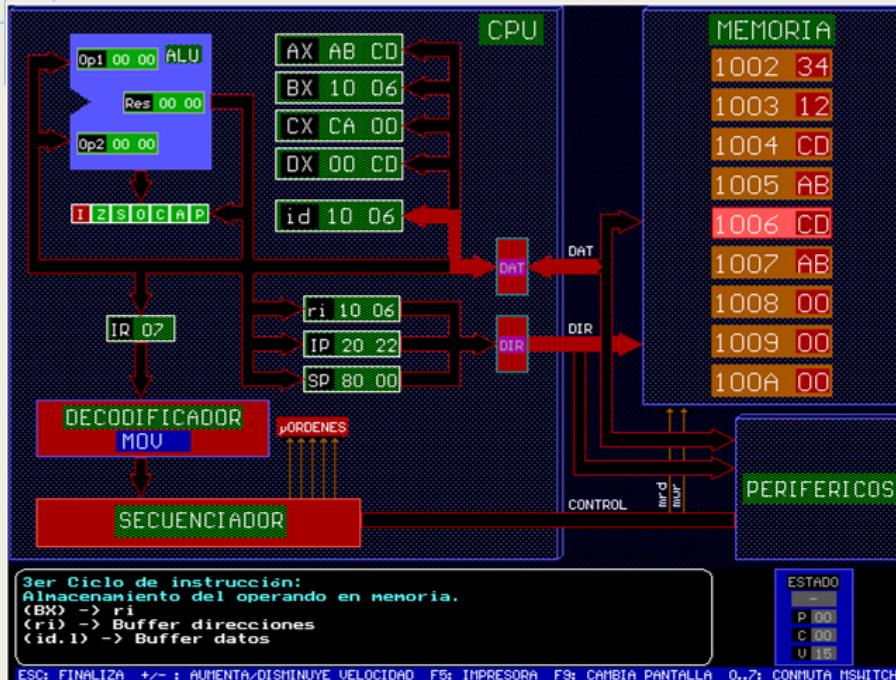
*Nota: la imagen muestra el momento en que se direcciona la memoria con el registro temporal RI (el dato aún no fue almacenado en AX).*

# Ej. 1: MOV BX, 1006H



<b>Origen</b>	Memoria (Instrucción:201Ch)
<b>Destino</b>	Registro (BX)
<b>Tamaño</b>	16 bits (BX)
<b>Modo</b>	Inmediato
<b>Dato</b>	1006h
<b>Resultado</b>	BX = 1006h

# Ej. 1: MOV WORD PTR [BX], 1006H



**Origen** Memoria (Instrucción:2020h)

**Destino** Memoria (\*BX:1006h)

**Tamaño** 16 bits (WORD PTR)

**Modo** Inmediato

**Dato** 1006h

**Resultado** NUM4(1006h) = 1006h

*ID tiene el valor 1006h leído desde la instrucción (dato inmediato).*

*RI tiene el valor 1006h transferido desde BX (direcciónamiento indirecto).*

# Ejercicio 2

Escribir un programa en lenguaje assembly del MSX88 que implemente la sentencia condicional de un lenguaje de alto nivel:

IF A < B THEN

C := A

ELSE C := B;

A en AL, B en BL y C en CL.

1. Identificar la condición como el estado de un FLAG a partir de una operación.
2. Determinar qué instrucción de salto conviene utilizar.
3. Expresar las instrucciones simples en lenguaje assembly y etiquetar los bloques.
4. Etiquetar la salida del condicional.

# Ejercicio 2 (cont.)

*Identificar la condición como el estado de un FLAG a partir de una operación*

- La condición se debe evaluar como una operación aritmética o una operación lógica de las disponibles.
- Tenemos que comparar dos valores.  
    CMP AL, BL
- Observamos lo que pasa con los FLAGS en los distintos casos límite:
  - Si  $AL = 1$  y  $BL = 2$ , ZSOC = 0100 (cuando  $AL < BL$ )
  - Si  $AL = 2$  y  $BL = 2$ , ZSOC = 1000 (cuando  $AL = BL$ )
  - Si  $AL = 2$  y  $BL = 1$ , ZSOC = 0000 (cuando  $AL > BL$ )
- Esta resolución no siempre es tan trivial.  
    ¿Qué pasa si la condición fuera  $A \geq B$ ?

# Ejercicio 2 (cont.)

*Determinar qué instrucción de salto conviene utilizar.*

*El flag a utilizar es el S (signo).*

*Las instrucciones candidatas son JS y JNS.*

- JS: el bit de signo vale 1, AL<BL es verdadero, salta a THEN.
- JNS: el bit de signo vale 0, AL<BL es falso, salta a ELSE.

*CMP AL, BL*

*JS then*

*;instrucciones del bloque ELSE  
then:  
;instrucciones del bloque THEN*

*CMP AL, BL*

*JNS else*

*;instrucciones del bloque THEN  
else:  
;instrucciones del bloque ELSE*

# Ejercicio 2 (cont.)

*Expresar las instrucciones simples en lenguaje assembly y etiquetarlas*

C := A (then)

*then: MOV CL, AL*

C := B (else)

*else: MOV CL, BL*

*CMP AL, BL*

*JNS else*

*;es la forma natural del if*

*MOV CL, AL*

*else: MOV CL, BL*

**Este código no funciona correctamente, ¿por qué?**

# Ejercicio 2 (cont.)

## *Etiquetar la salida del condicional*

```
CMP AL, BL  
JNS else  
    MOV CL, AL  
    JMP fin  
else: MOV CL, BL  
fin: HLT  
END
```

- El procesador ejecuta las instrucciones secuencialmente.
- Las instrucciones de salto modifican el registro PC para alterar el orden normal.
- Si no se ejecuta un salto luego del primer bloque de instrucciones, se sigue ejecutando en secuencia el *else*.

# Ejercicio 2 (cont.: A<=B)

*Si la condición fuese A<=B*

- Tenemos que comparar dos valores.  
    CMP AL, BL
- Observamos lo que pasa con los FLAGS en los distintos casos límite:
  - Si AL = 1 y BL = 2, ZSOC = 0100 (cuando AL < BL)
  - Si AL = 2 y BL = 2, ZSOC = 1000 (cuando AL = BL)
  - Si AL = 2 y BL = 1, ZSOC = 0000 (cuando AL > BL)
- La condición implica evaluar dos FLAGS (o bien S=0 o bien Z=0).
- Se pueden encadenar instrucciones de salto por los bits que valen 1:
  - JS then
  - JZ then
  - else: JMP fin
  - then: hlt

# Ejercicio 2 (cont.: A<=B)

*Si la condición fuese A<=B*

- Una solución más elegante y eficiente es invertir la condición.  
A<=B es *falso* cuando A > B
- Tenemos que comparar dos valores.  
CMP BL, AL
- Observamos lo que pasa con los FLAGS en los distintos casos límite:
  - Si AL = 1 y BL = 2, ZSOC = 0000 (cuando AL < BL)
  - Si AL = 2 y BL = 2, ZSOC = 1000 (cuando AL = BL)
  - Si AL = 2 y BL = 1, ZSOC = 0100 (cuando AL > BL)
- Ahora la condición es verdadera cuando el bit S=0.

# Ejercicio 2 (cont.: A<=B)

Dir.	Cód. máquina	Lín.	Cód. ensamble	
		1	ORG 2000h	
2000	3A D8	2	CMP BL, AL	
2002	79 03	3	JNS else	;S=0 cuando AL>BL
2004	E9 08 20	4	then: JMP fin	
2007	90	5	else: NOP	;instrucciones del else
2008	F4	6	HLT	
		7	END	

# Pila

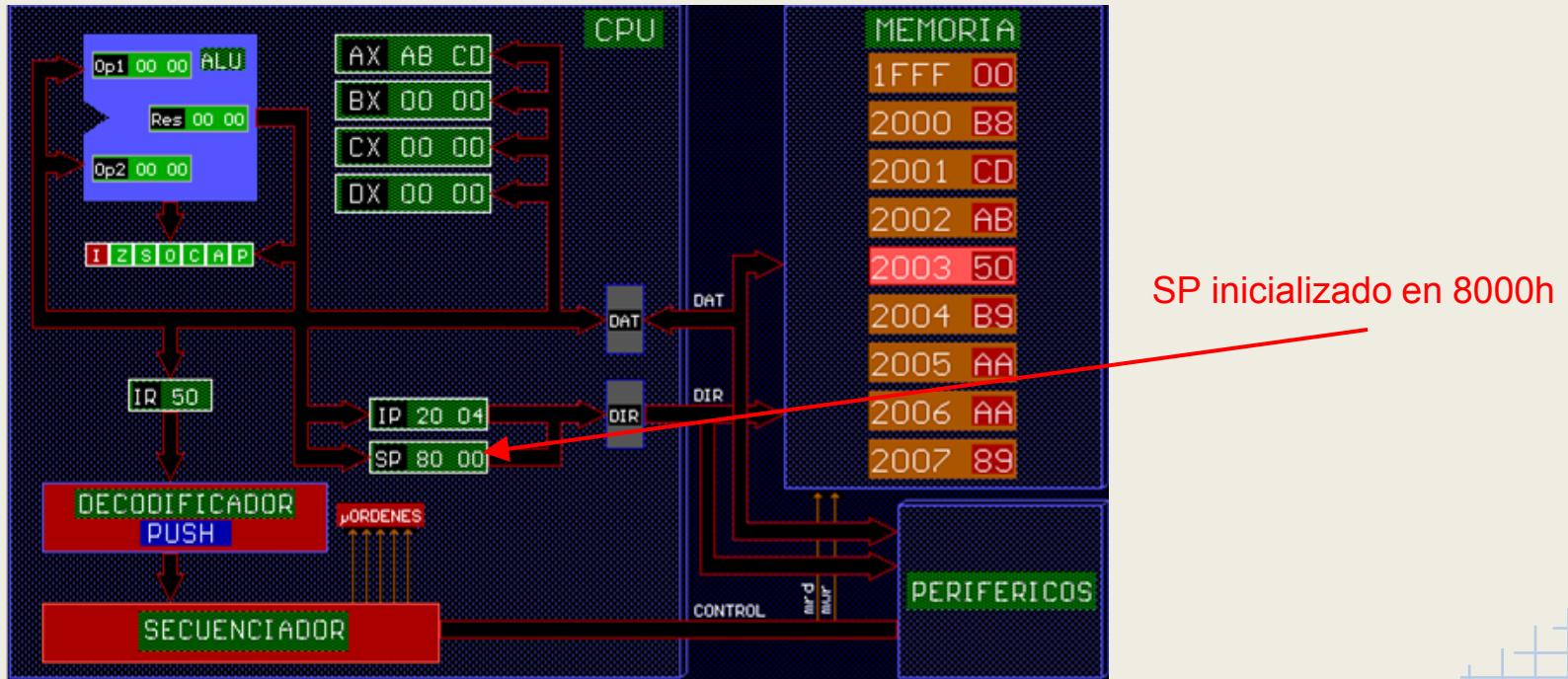
La pila es un área de la memoria que:

- comienza en una dirección fija (en nuestro caso: 7FFFh)
- como estructura, tamaño variable (crece a medida que *apilamos* datos y decrece a medida que los *desapilamos*)
- está apuntada por un registro (SP: stack pointer) que se modifica automáticamente en cada operación (se inicializa en 8000h)
- se corresponde a una estructura LIFO (last in - first out) -> SP apunta al último dato apilado

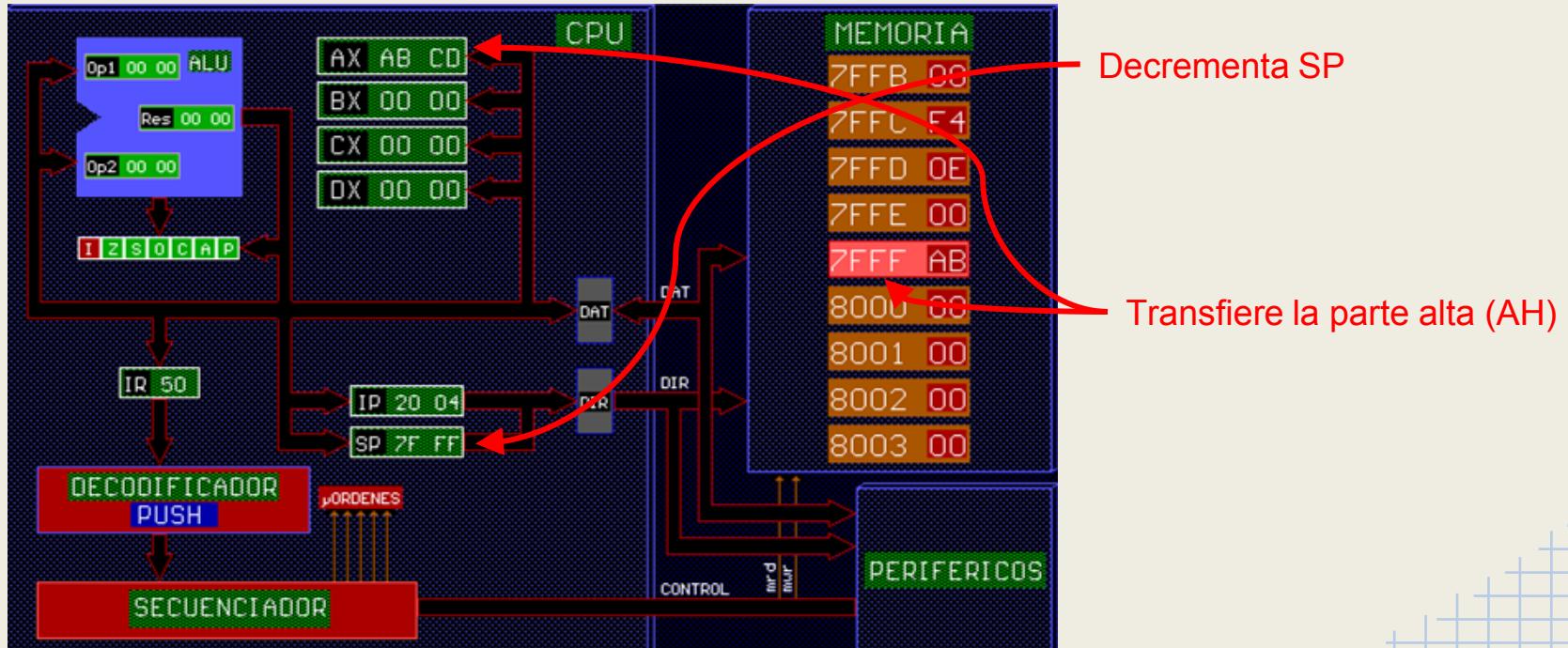
# Pila (instrucciones)

- PUSH: *decrementa* el SP y apila el contenido del registro de 16 bits.
- POP: transfiere 16 bits de la pila al registro e *incrementa* el SP.
- Son instrucciones unarias (requieren un solo operando).
- El destino (PUSH) y el origen (POP) están implícitos.
- El registro SP se modifica automáticamente.
- El espacio de memoria no está protegido (se puede modificar apuntando a direcciones del área 7FFF).

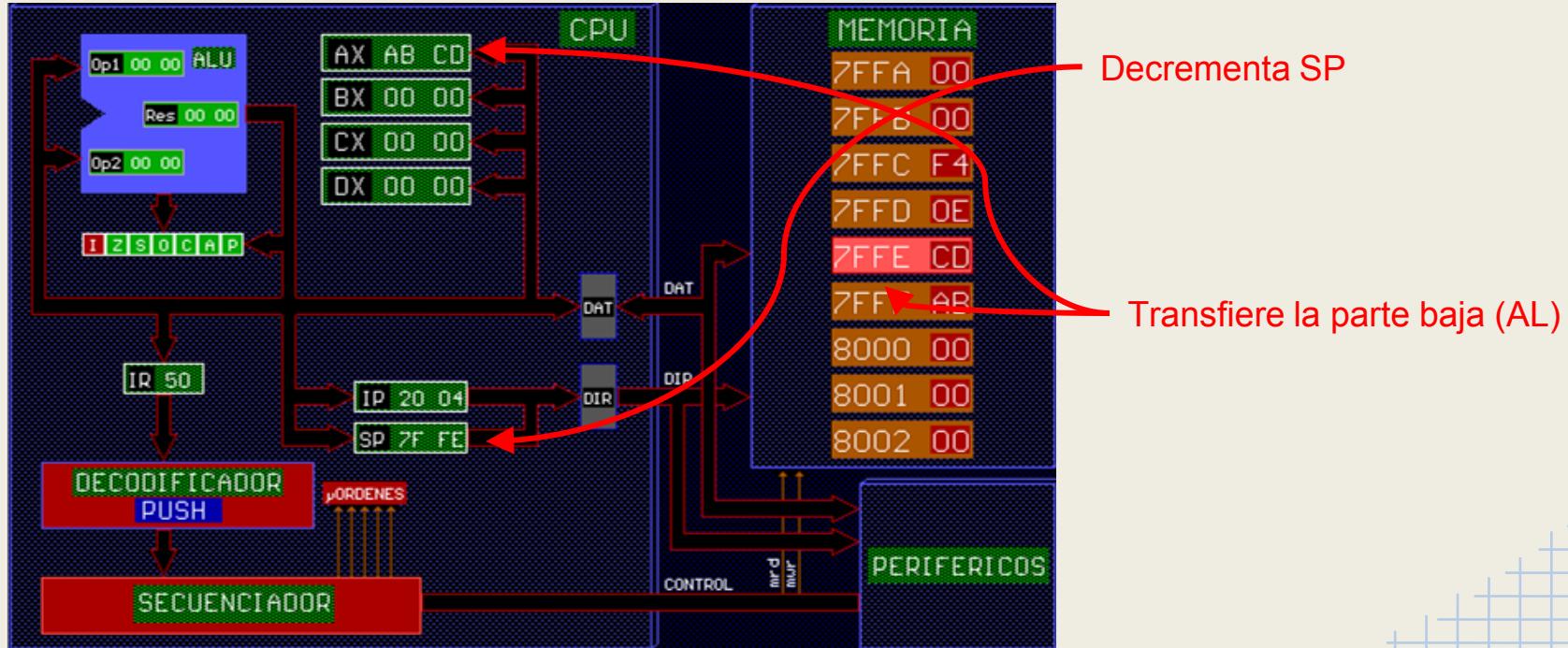
# Pila: PUSH AX



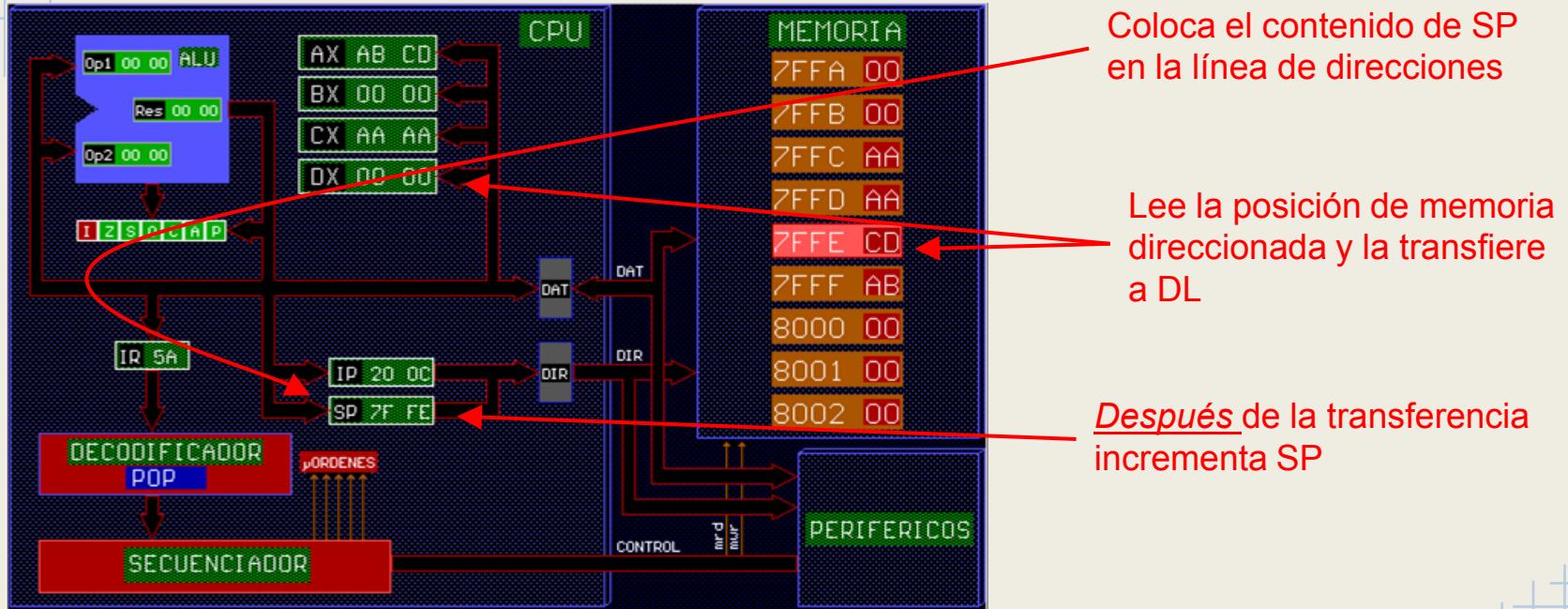
# Pila: PUSH AX



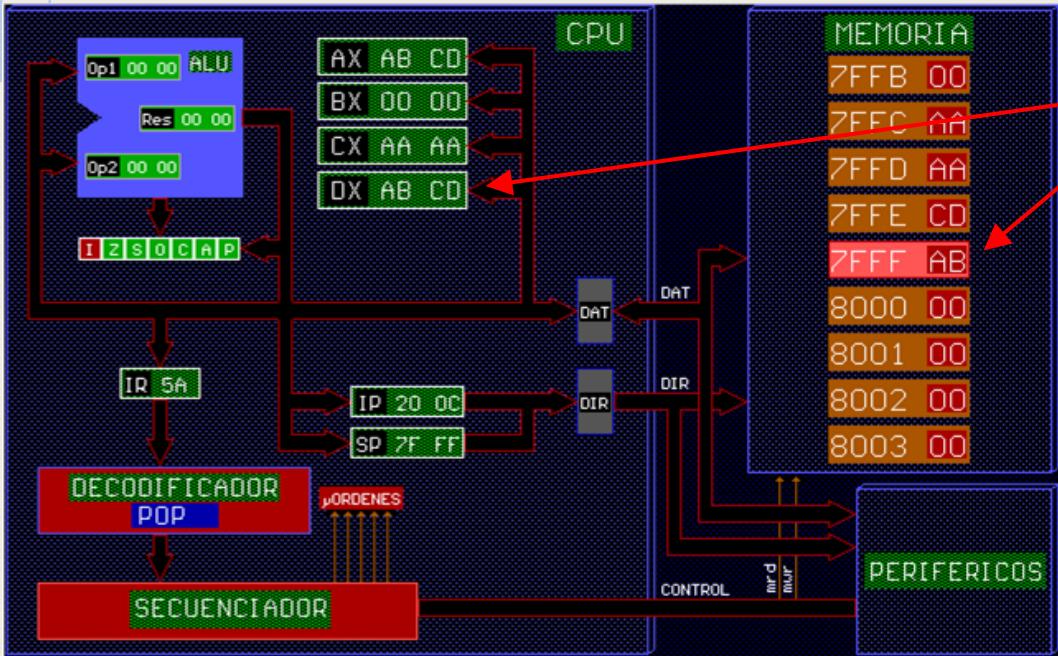
# Pila: PUSH AX



# Pila: POP DX

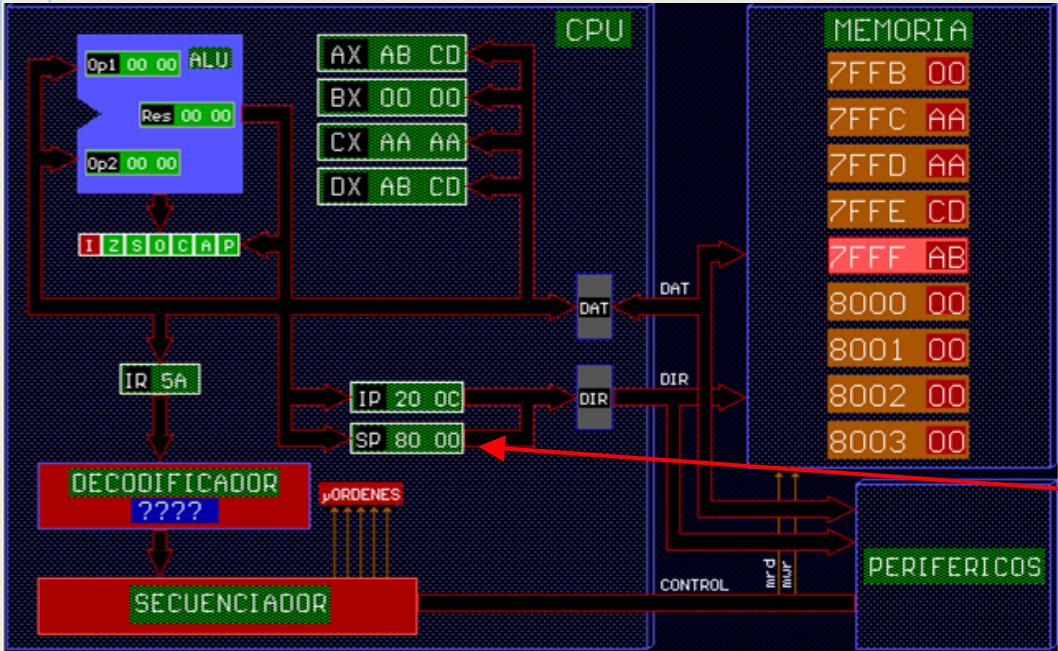


# Pila: POP DX



Transfiere el contenido de la posición apuntada por SP a la parte alta de DX (DH)

# Pila: POP DX



¿Qué dato contiene la posición 7FFEh después de desapilar?

# Llamadas a subrutina

Para mejorar la legibilidad del código es útil implementar subrutinas (similares a los procedimientos y las funciones de lenguajes de alto nivel), modularizando así la solución.

*Las instrucciones son captadas desde memoria en secuencia*



*Existen instrucciones de salto*

*Las llamadas a procedimiento retornan al punto de invocación*



*Se necesita una técnica que permita 'guardar' el estado previo a la invocación*

*Instrucción de salto que almacene el punto de retorno*



*CALL  
...  
RET*

# Subrutinas: CALL

La instrucción CALL modifica el contador de programa para alterar la secuencia del programa *apilando* antes el valor de ese registro.

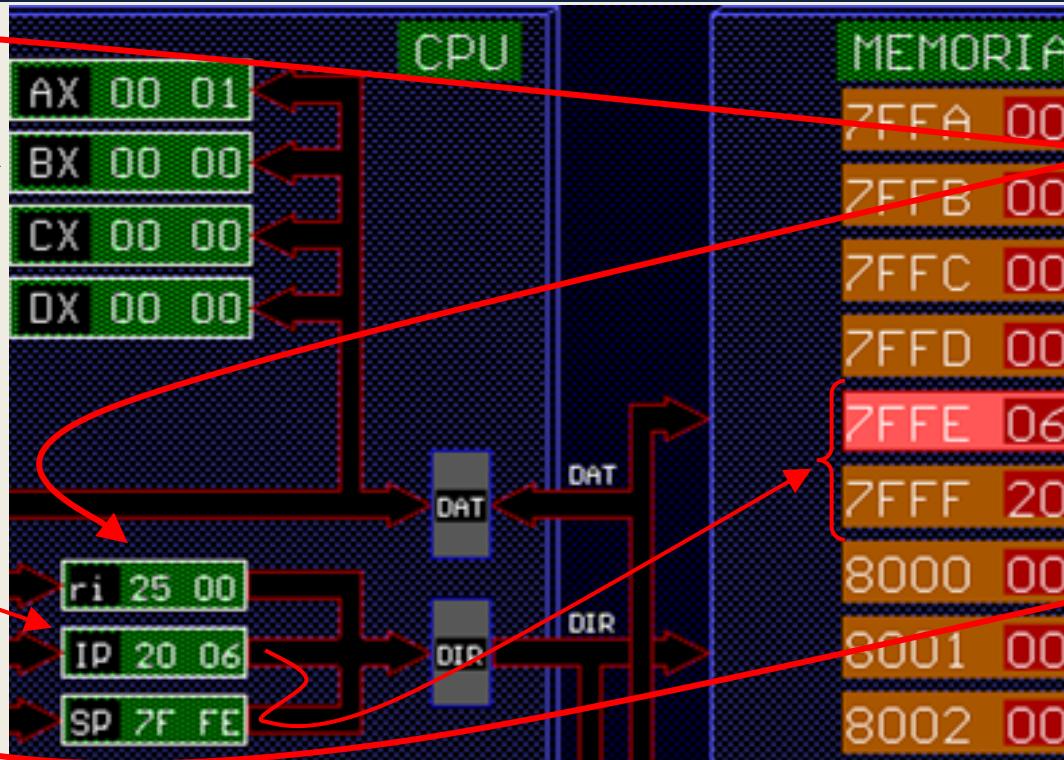
- Es una instrucción unaria (CALL etiqueta\_destino)
- Hace uso de la pila y por lo tanto modifica el SP
- Apila el valor actual del registro PC *antes* de modificarlo
- Coloca en el registro PC la dirección identificada por etiqueta\_destino
- *Nada dice del retorno al bloque de código que hace la invocación*

# Subrutinas: CALL

```
ORG 2500h
SUBR: ADD AX, AX
      CMP AX, 8
      JNZ SUBR
RET
```

RET

```
ORG 2000h  
MOV AX, 1  
CALL SUBR  
MOV num, AX  
HLT  
END
```



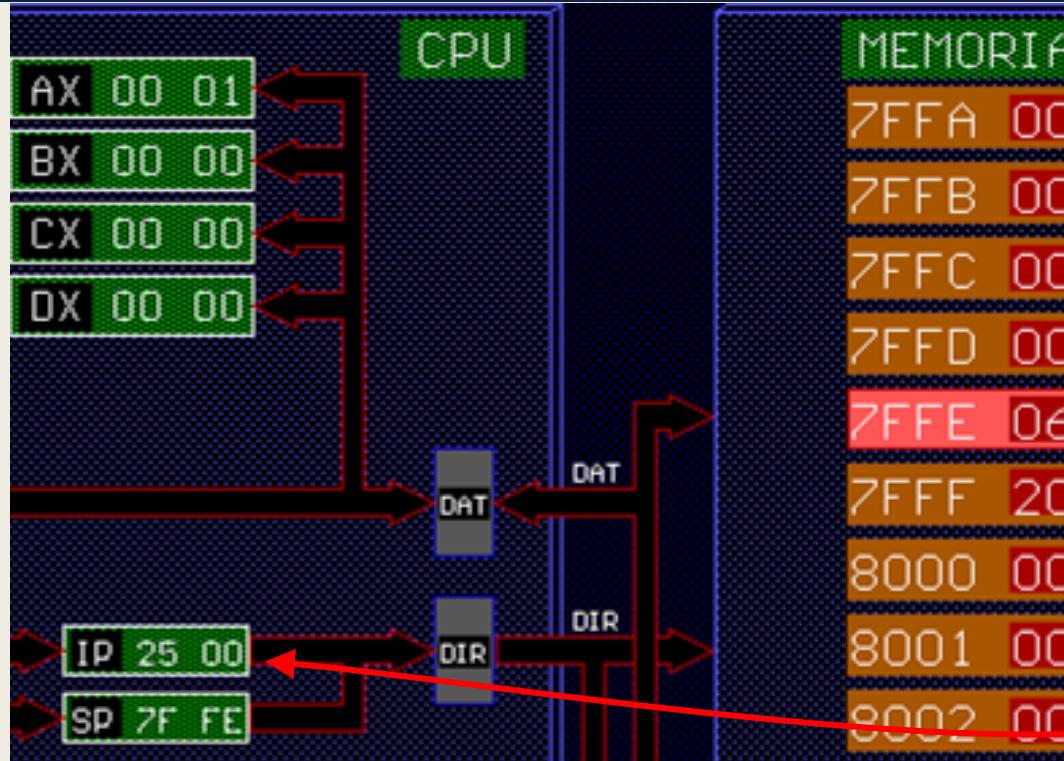
Almacena temporalmente la dirección de SUBR en RI

Apila la próxima instrucción a ejecutar (registro IP)

# Subrutinas: CALL

```
ORG 2500h  
SUBR: ADD AX, AX  
      CMP AX, 8  
      JNZ SUBR  
  
RET
```

```
ORG 2000h  
MOV AX, 1  
CALL SUBR  
MOV num, AX  
HLT  
END
```



Coloca la dirección etiquetada como SUBR en el contador de programa

# Subrutinas: RET

Al finalizar la ejecución de la subrutina se debe volver al lugar de la invocación (instrucción siguiente al CALL). La dirección de esa instrucción fue *apilada* al ejecutar CALL.

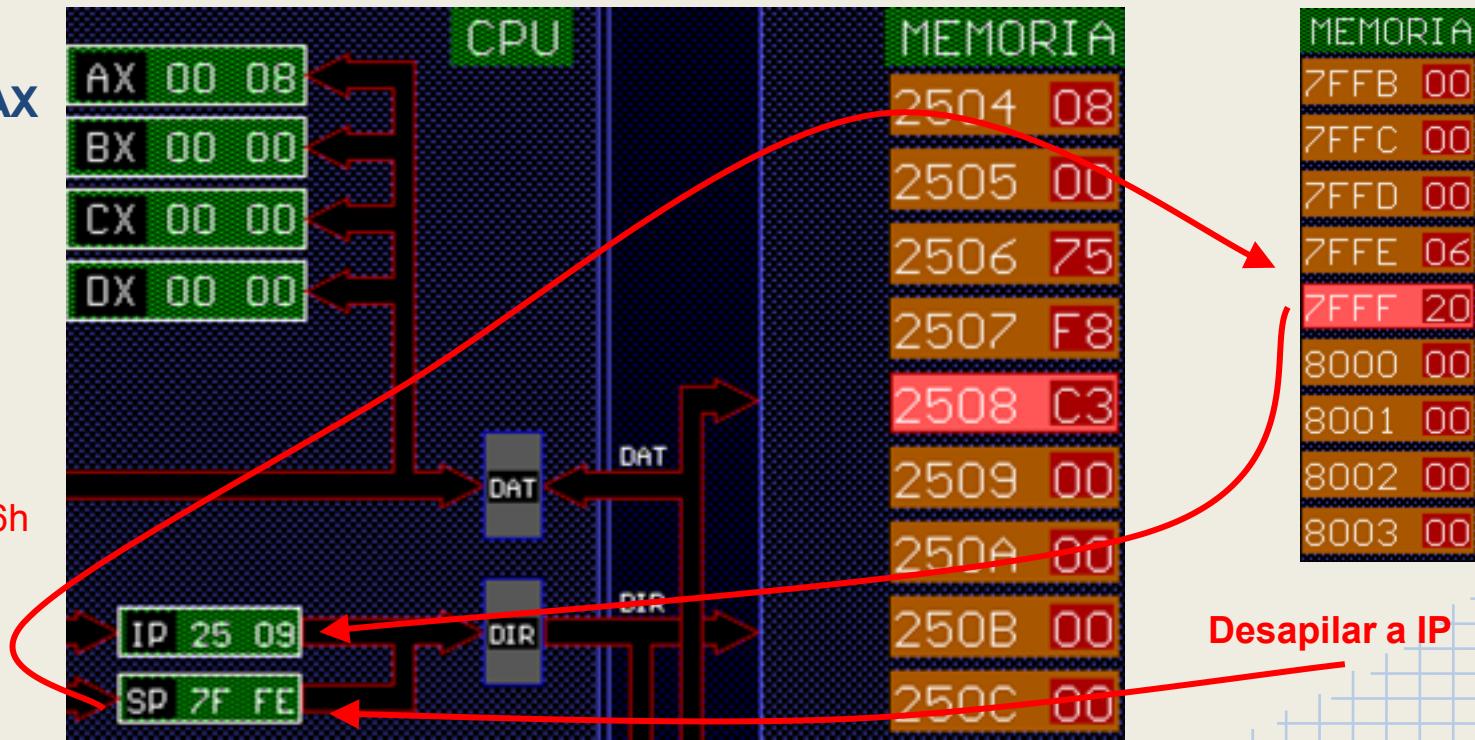
La instrucción RET:

- No necesita operandos
- Desapila un dato (modifica SP)
- El dato desapilado se transfiere al contador de programa
- **Corresponde al usuario asegurar que SP apunta a la dirección donde apiló CALL (si hace uso de la pila dentro de la subrutina)**

# Subrutinas: RET

```
ORG 2500h  
SUBR: ADD AX, AX  
      CMP AX, 8  
      JNZ SUBR  
RET 2508h
```

```
ORG 2000h  
MOV AX, 1  
CALL SUBR  
MOV num, AX 2006h  
HLT  
END
```

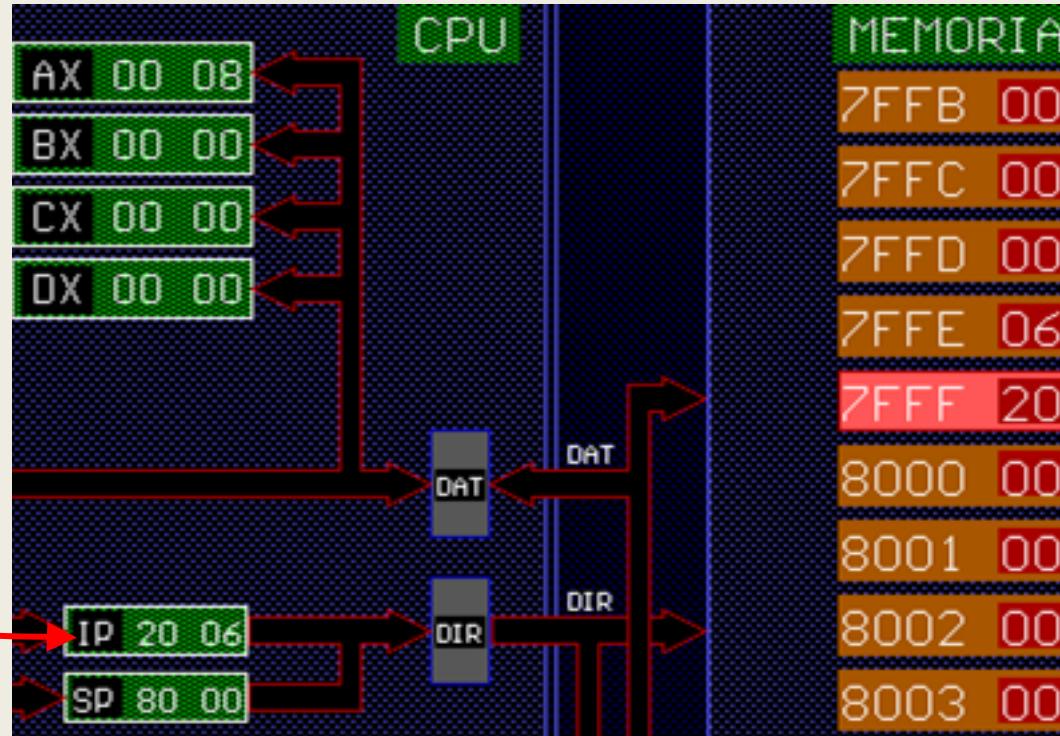


# Subrutinas: RET

```
ORG 2500h  
SUBR: ADD AX, AX  
      CMP AX, 8  
      JNZ SUBR
```

RET

```
ORG 2000h  
MOV AX, 1  
CALL SUBR  
MOV num, AX  
HLT  
END
```



# Pasaje de parámetros

En el ejemplo anterior, el valor inicial a sumar (1) es pasado a través del registro AX. Al retornar, el resultado es devuelto en el mismo registro y luego se almacena en la dirección con etiqueta ‘num’.

- La subrutina recibe el valor a utilizar como parámetro
- Procesa ese valor y devuelve el resultado final en un registro
- El parámetro es pasado *por valor*

# Pasaje de parámetros

ORG 1000h

num DB 1

ORG 2500h

SUBR: MOV CL, [BX]  
lazo: ADD CL, CL  
CMP CL, 8  
JNZ lazo

RET

Al retornar, el dato en la dirección *num* permanece inalterado

ORG 2000h

MOV BX, OFFSET num  
CALL SUBR  
MOV num, CL  
HLT  
END

BX = Dirección de num

SUBR recibe la dirección de memoria donde se encuentra el parámetro

El resultado es devuelto en el registro CL

# Pasaje de parámetros

En este ejemplo, la subrutina recibe a través del registro BX la dirección de memoria desde donde debe cargar el valor inicial a sumar.

- La subrutina recibe como parámetro la dirección del dato a utilizar
- Transfiere el dato desde la dirección indicada a un registro
- Realiza el proceso con este registro
- El dato original en memoria no se modifica en ningún momento
- El parámetro es pasado *por referencia*