

# 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 ( $2^{16} = 64\text{M}$  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.

**Extensión del archivo: .asm** → `explicacion.asm`

**Código fuente** → `num1 DB 8 ;posicion 1000h`

**Nada de ortografía castellana** → `tab DW 1, 2, 3, 4 ;posiciones 1002h a 1009h`

**El código termina con la directiva END (no es una instrucción)** → `END`

**Nueva línea (ENTER) al final del archivo (si no, da error)** → `; siempre poner una linea nueva despues del END!!`

```
explicacion.asm - Bloc de notas
Archivo Edición Formato Ver Ayuda

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
volver:
CMP CH, CL
JZ termine
ADD AH, AL
DEC CL
JMP volver
termine:
MOV res, AH
HLT
END ; siempre poner una linea nueva despues del END!!
```

Línea 19, columna 60

# 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]:

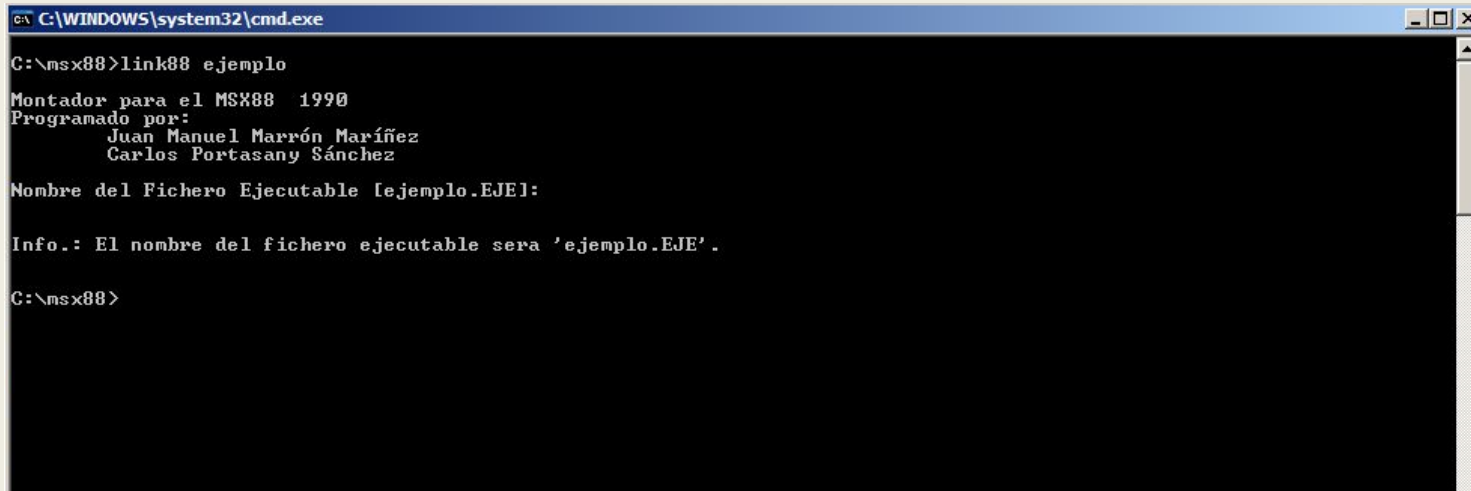
      0 Error(es)      0 Aviso(s)
Info.: El nombre del Fichero Objeto sera 'ejemplo.0'.

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]:

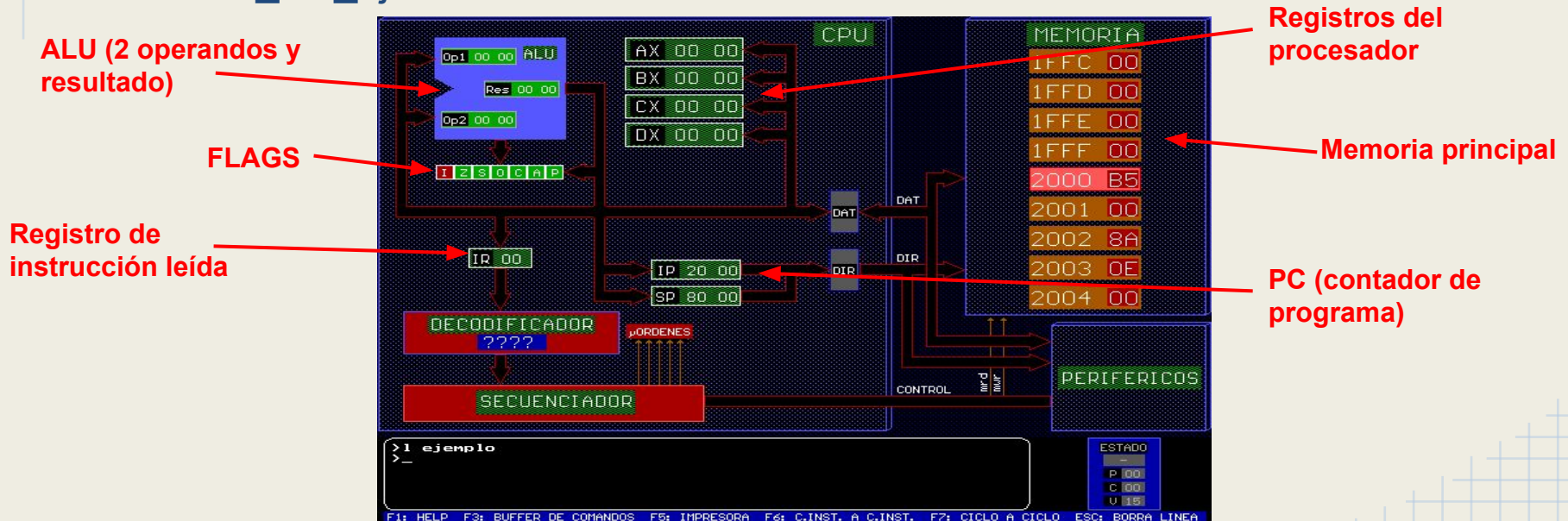
Info.: El nombre del fichero ejecutable sera 'ejemplo.EJE'.

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 ?      ;posicion 1010h

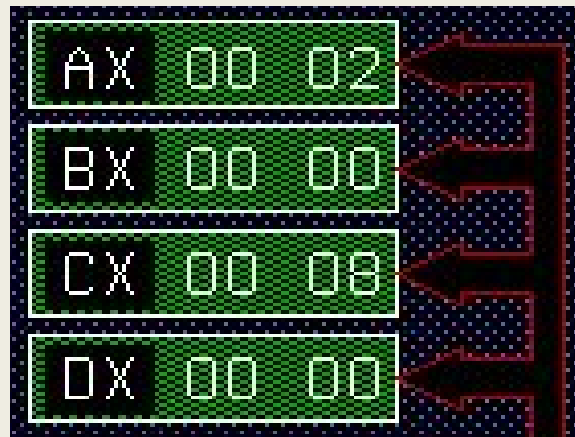
END
```

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

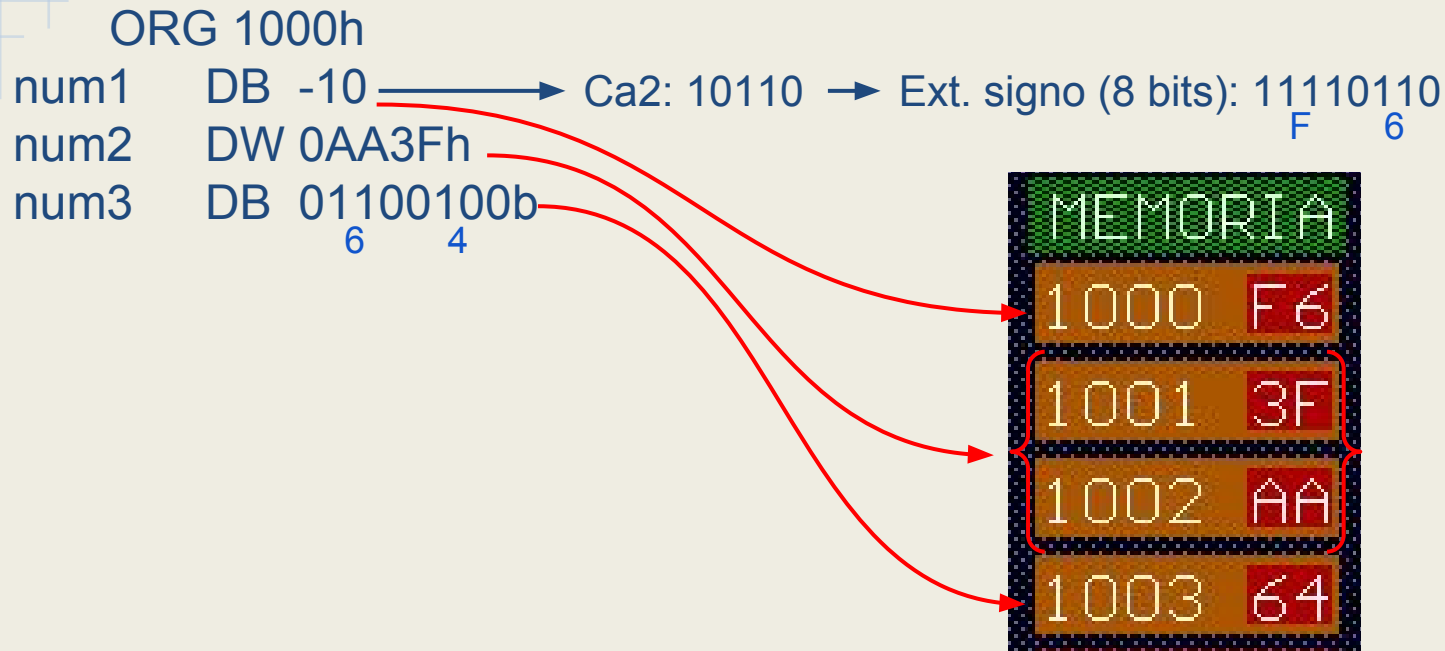
# Ejemplo 1 (cont.)

```
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



# 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
```

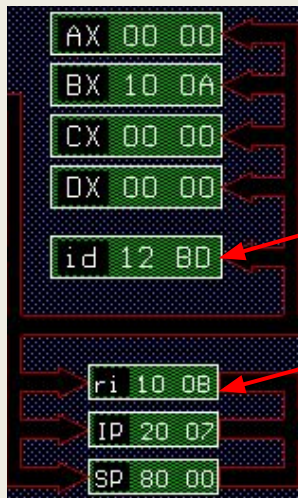
AX	F6	64
BX	F6	00
CX	AA	3F
DX	00	3F

# Ejemplo 3 (reg. temporales)

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

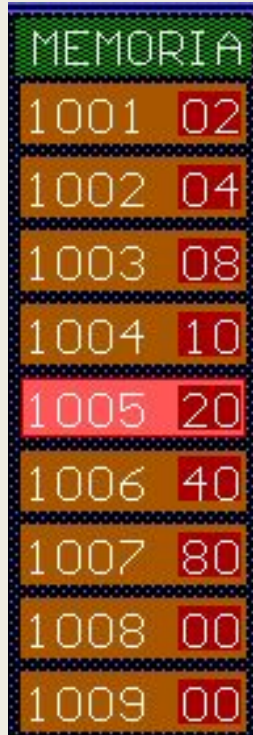
Las siguientes instrucciones alteran el flujo normal del programa modificando el registro IP según distintas condiciones:

<i><b>Instrucción</b></i>	<i><b>Condición</b></i>	<i><b>Opuesta</b></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

	tabla	AL	BX	CL	ZSOC	JS?
ORG 1000h						
num1 DB 8	?	2	1001h	0	0000	-
tabla DB ?						
	2	4	1002h	1	0100	Sí
ORG 2000h						
MOV CL, 0	2, 4	8	1003h	2	0100	Sí
MOV CH, num1						
MOV BX, OFFSET tabla	2, 4, 8	16	1004h	3	0100	Sí
MOV AL, 2						
lazo: MOV [BX], AL	2, 4, 8, 16	32	1005h	4	0100	Sí
ADD AL, AL	2, 4, 8, 16, 32	64	1006h	5	0100	Sí
INC BX						
INC CL	2, 4, 8, 16, 32, 64	128	1007h	6	0100	Sí
CMP CL, CH						
JS lazo	2, 4, 8, 16, 32, 64, 128	0	1008h	7	0100	Sí
HLT						
END	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?*