Módulo Assembler

Programación en bajo nivel

Autores:
Alejandro Héctor Gonzalez
Silvana Lis Gallo
Junio 2021

RESUMEN

En esta clase repasamos los conceptos fundamentales de la organización de una computadora. Se presenta el lenguaje de programación en bajo nivel para el Assembler del 8088. Se revisa el set de instrucciones

Palabras clave

assembler, 8088, set de instrucciones, ensamblador, progarmación de bajo nivel

Esquema componentes de una computadora

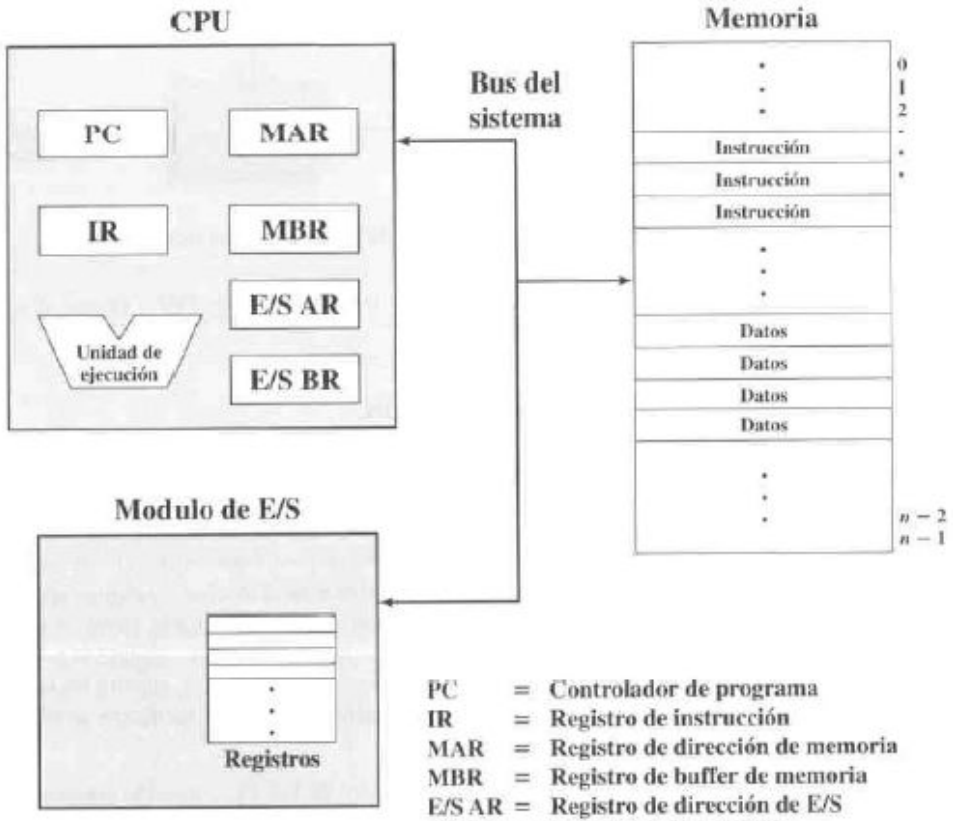
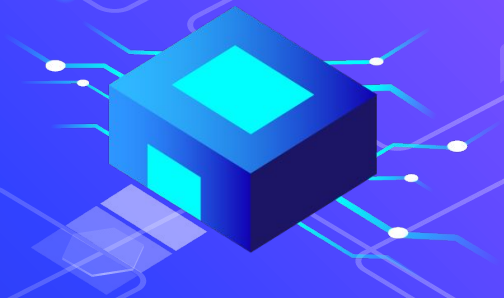
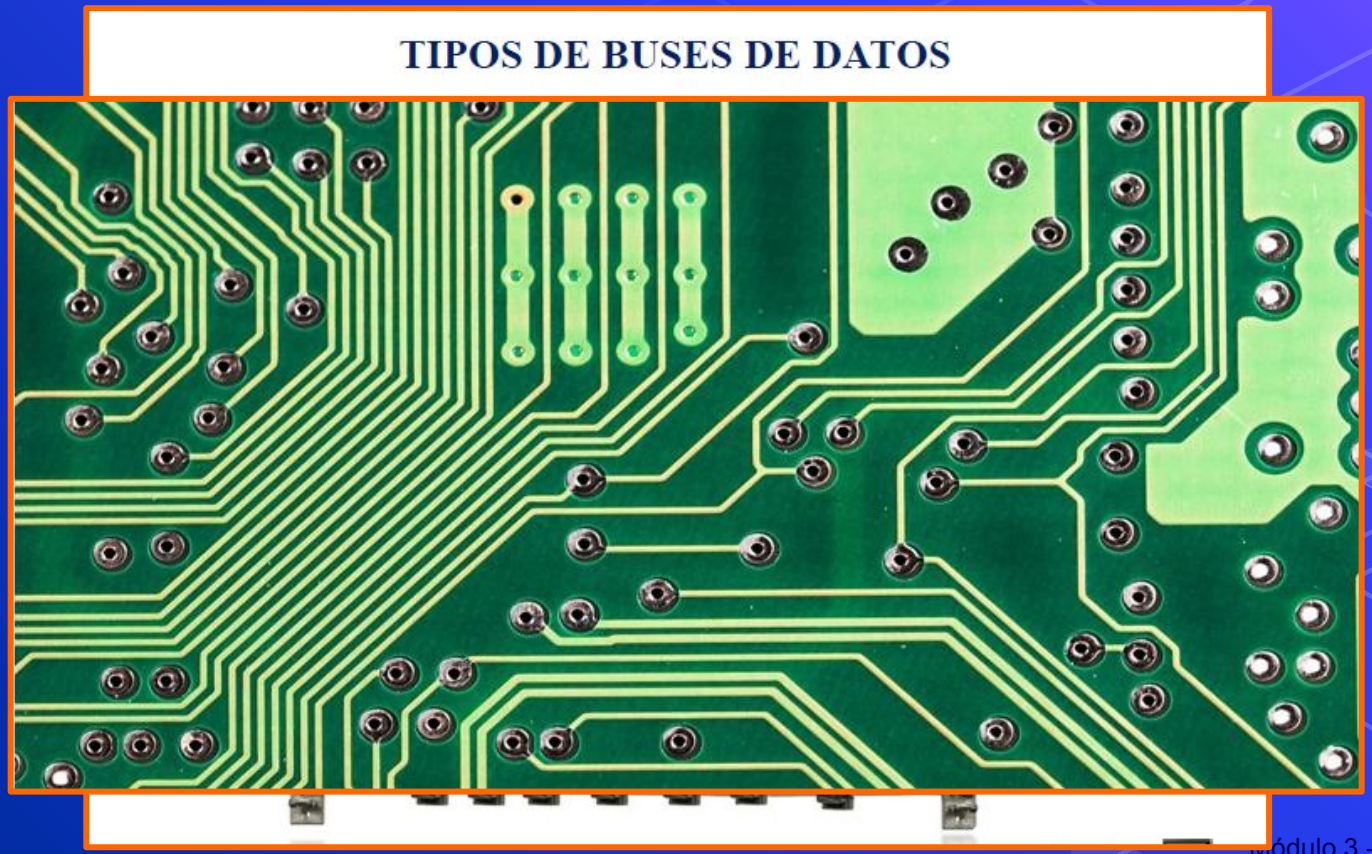


Figura 3.2. Componentes del computador: esquema de dos niveles.



Bus (canal)



En arquitectura de computadores, el **bus** (o canal) es un sistema digital que transfiere datos entre los componentes de una computadora. Está formado por cables o pistas en un circuito impreso

Esquema de buses

- Control
- Dirección
- Datos

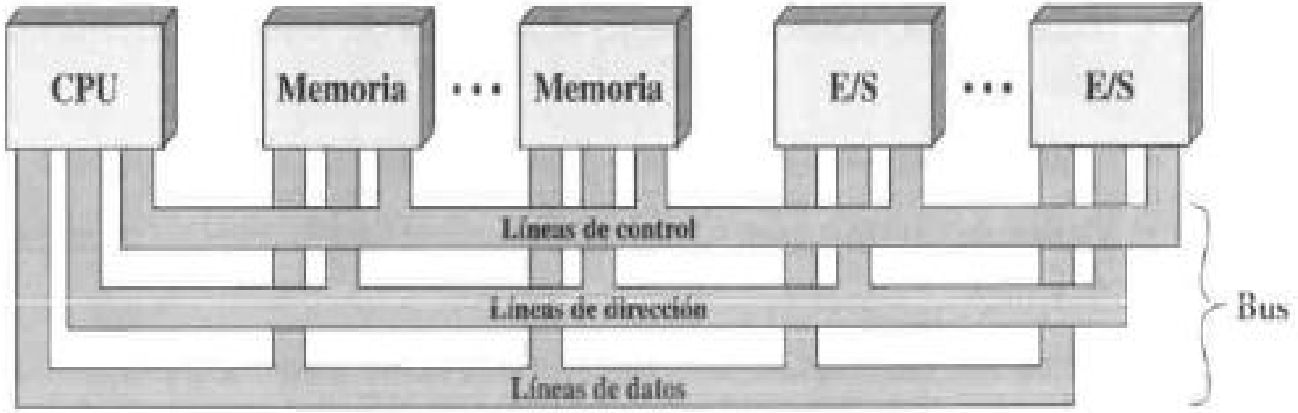
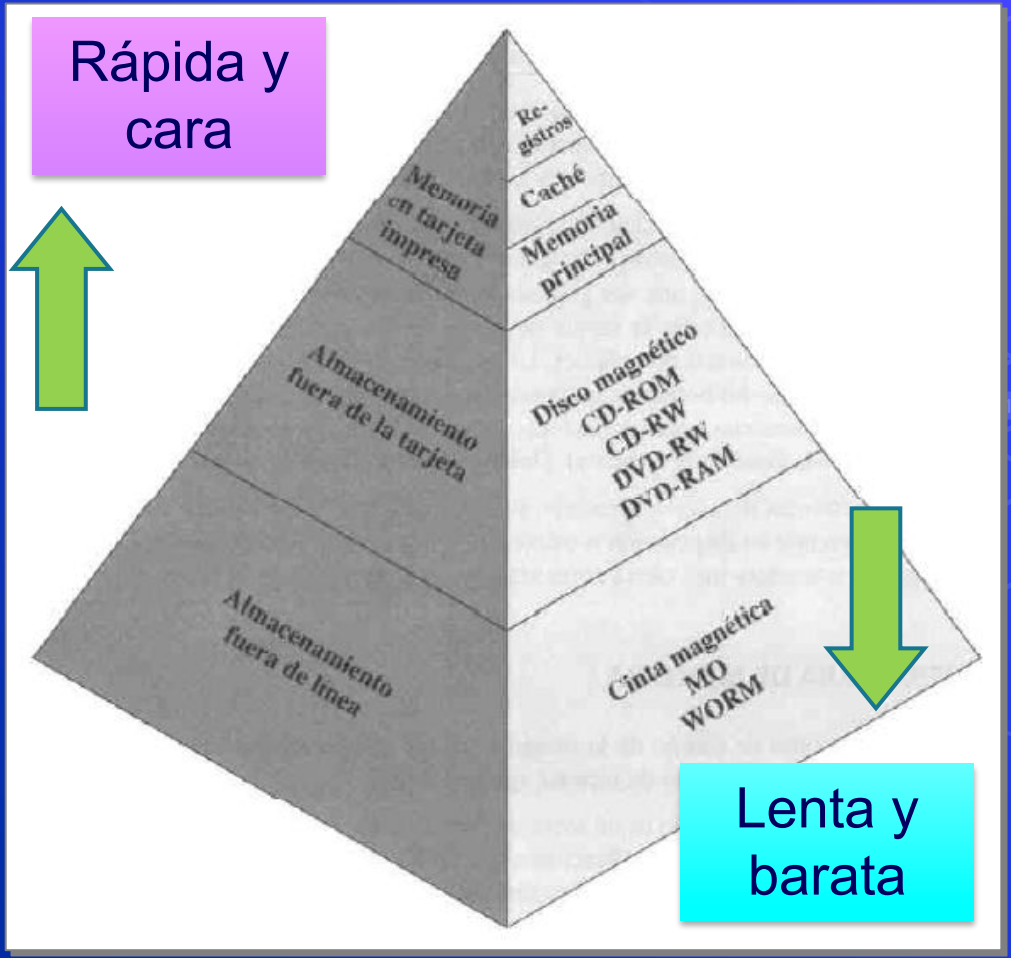


Figura 3.16. Esquema de interconexión mediante un bus.

Memoria



Al bajar en la jerarquía se observa:

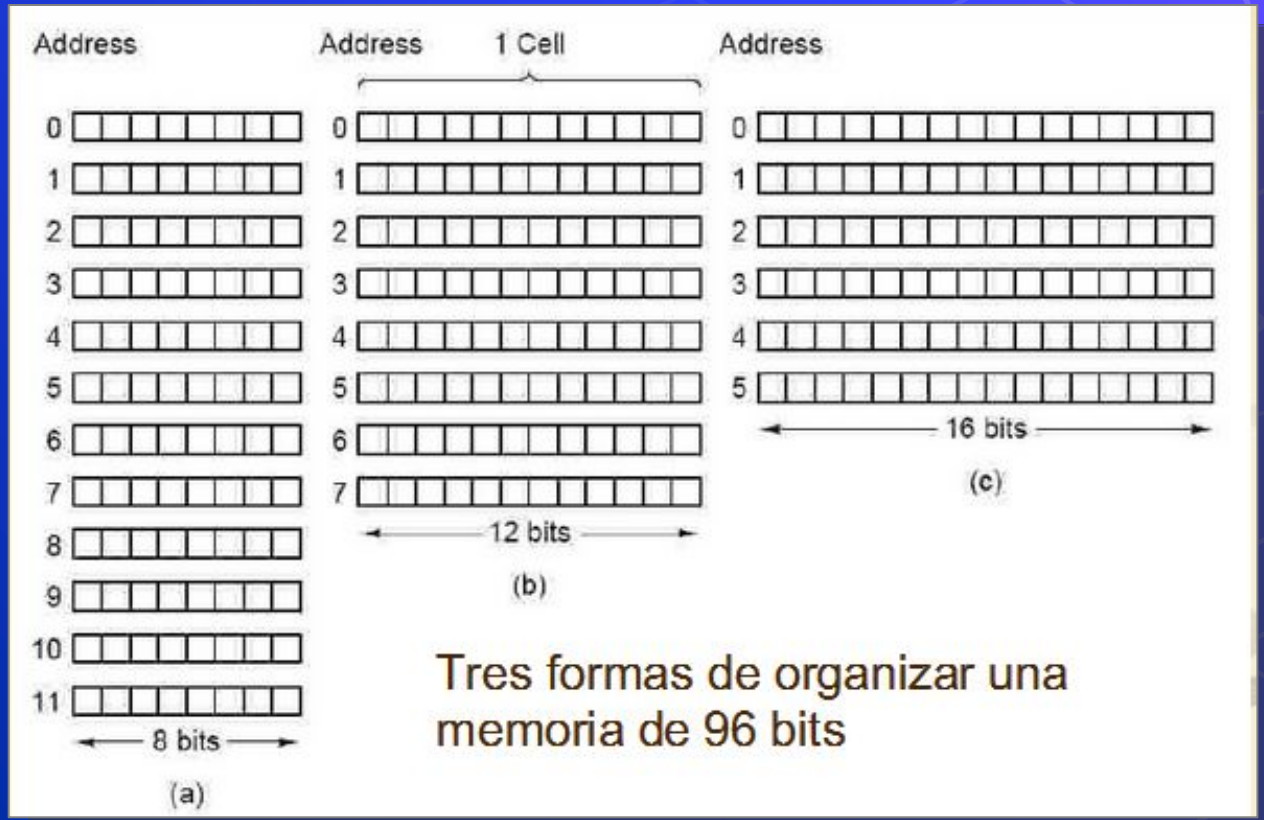
a) Disminuye el costo por bit

b) Aumenta la capacidad

c) Aumenta el tiempo de acceso

d) Disminuye la frecuencia de acceso a la memoria por parte del procesador

Memoria



¿Cómo se organiza la memoria RAM?

¿Qué es un byte?

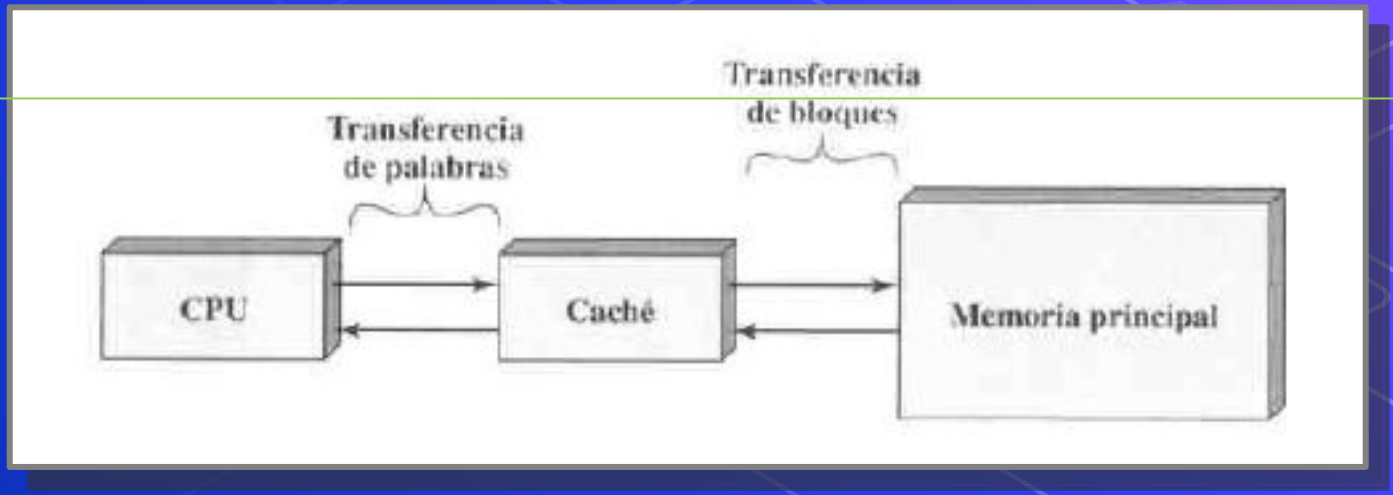
8 bits

¿Qué es un Word?

16 bits

Memoria Cache - Objetivo

- Lograr que la velocidad de la memoria sea lo mas rápida posible.
- Obtener un tamaño grande al precio de las memorias menos costosas.



Memoria Cache

Principios

1. Principio de **localidad espacial** de referencia:

- cuando se accede a una palabra de memoria, es "muy probable" que el próximo acceso sea en la vecindad de la palabra anterior.

2. Principio de **localidad temporal** de referencia:

- cuando se accede a una posición de memoria, es "muy probable" que un lapso de "tiempo corto", dicha posición de memoria sea accedida nuevamente.

Memoria Cache

Localidad espacial

Se sustenta en:

- Ejecución secuencial del código
- Tendencia de los programadores a hacer próximas entre sí variables relacionadas
- Acceso a estructuras tipo matriz ó pila

Localidad Temporal

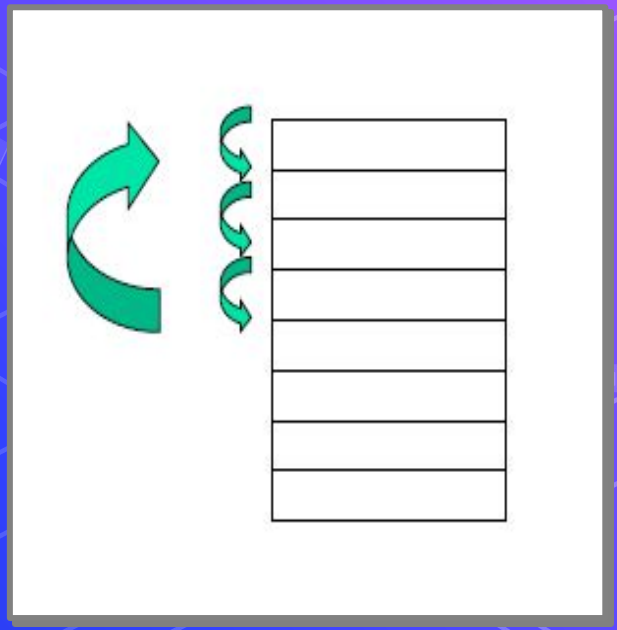
Se sustenta en:

- Formación de ciclos o bucles en el código
- Subrutinas (Procedimientos o Funciones)
- Pilas

Memoria Cache

Estas 2 sentencias exhiben los dos principios antes mencionados:

For i=1 to i=10, do
A[i]:=0;



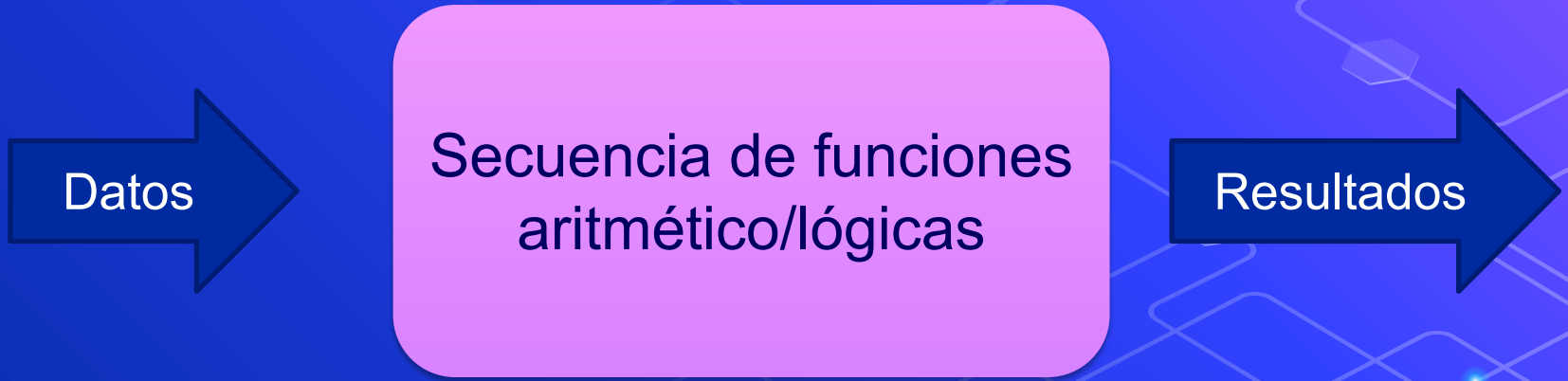
En cada ciclo se consulta cuanto vale i. (Temporal)

Cada asignación **A[i]:=0** almacena un 0 en un elemento del arreglo (Espacial)

“ Programación del procesador 8088

Programa

- En los INICIOS se tenían **sistemas cableados**



- **Programación en hardware**: cuando cambiamos las tareas, debemos cambiar el hardware

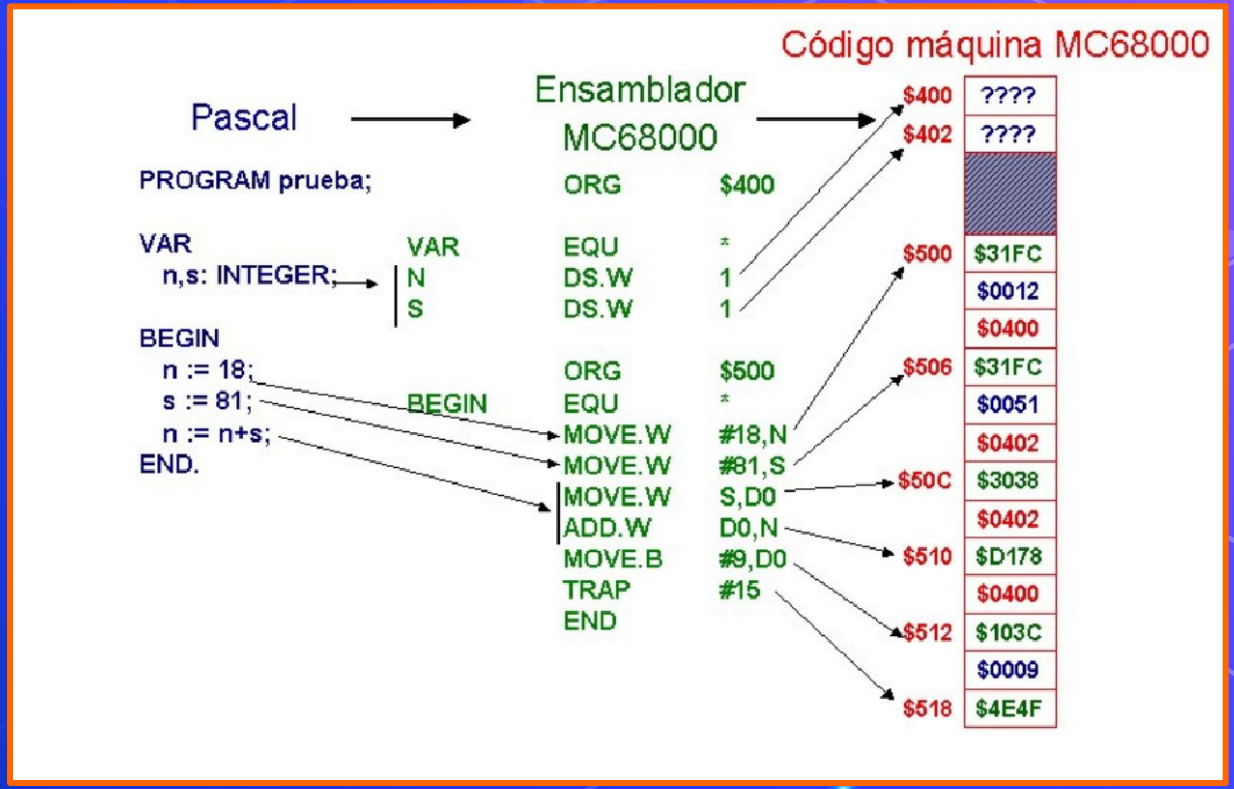
Programa

AHORA

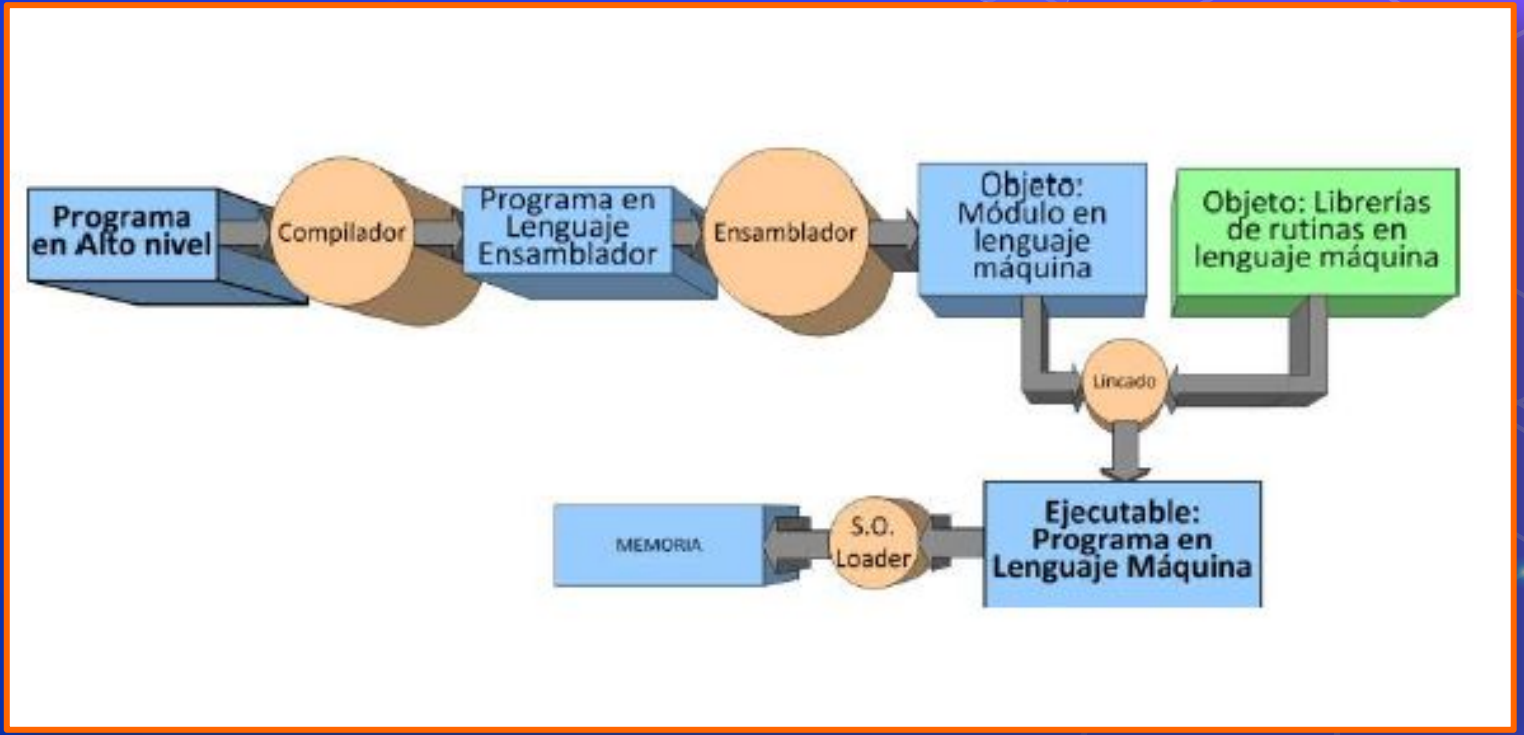
- Programación en software: en cada paso se efectúa alguna operación sobre los datos

Como se va de un programa de Alto nivel a uno de Bajo nivel

Se llaman ensambladores a los programas encargados de traducir los programas escritos en mnemónico a lenguaje binario.



Programa
Del Alto nivel al Bajo nivel



Programa
Del Alto nivel al Bajo nivel

Video con explicación de la ejecución de un programa paso a paso

- https://www.youtube.com/watch?v=PJV7GRCCJSc

Programación Bajo Nivel - Características

- Los lenguajes de programación de bajo nivel **son dependientes de la CPU**, están hechos a medida del hardware de la misma y por lo tanto aprovechan al máximo sus características.

- El fabricante del microcontrolador le asigna un nombre propio denominado “mnemónico” a cada byte que representa una **instrucción en lenguaje de máquina**. Este nombre mnemotécnico es una palabra corta o abreviatura que trata de que su lectura implique el entendimiento de la acción que realiza. Ejemplo **DEC** es decrementar, **INC** incrementar.

Programación Bajo Nivel - Características

- El programa escrito en lenguaje ensamblador es de **difícil lectura** ya que su estructura se acerca al lenguaje máquina.
- El lenguaje ensamblador es **difícilmente portable**, es decir, un código escrito para un microcontrolador, puede requerir modificación, para poder ser usado en otra máquina distinta del mismo fabricante. Al cambiar de fabricante es necesario reescribirlo completamente.
- Los programas creados por un programador experto en lenguaje ensamblador **son generalmente mucho más rápidos** y consumen menos recursos del sistema (memoria de datos y memoria de programa) que el programa equivalente proveniente de un lenguaje de alto nivel.

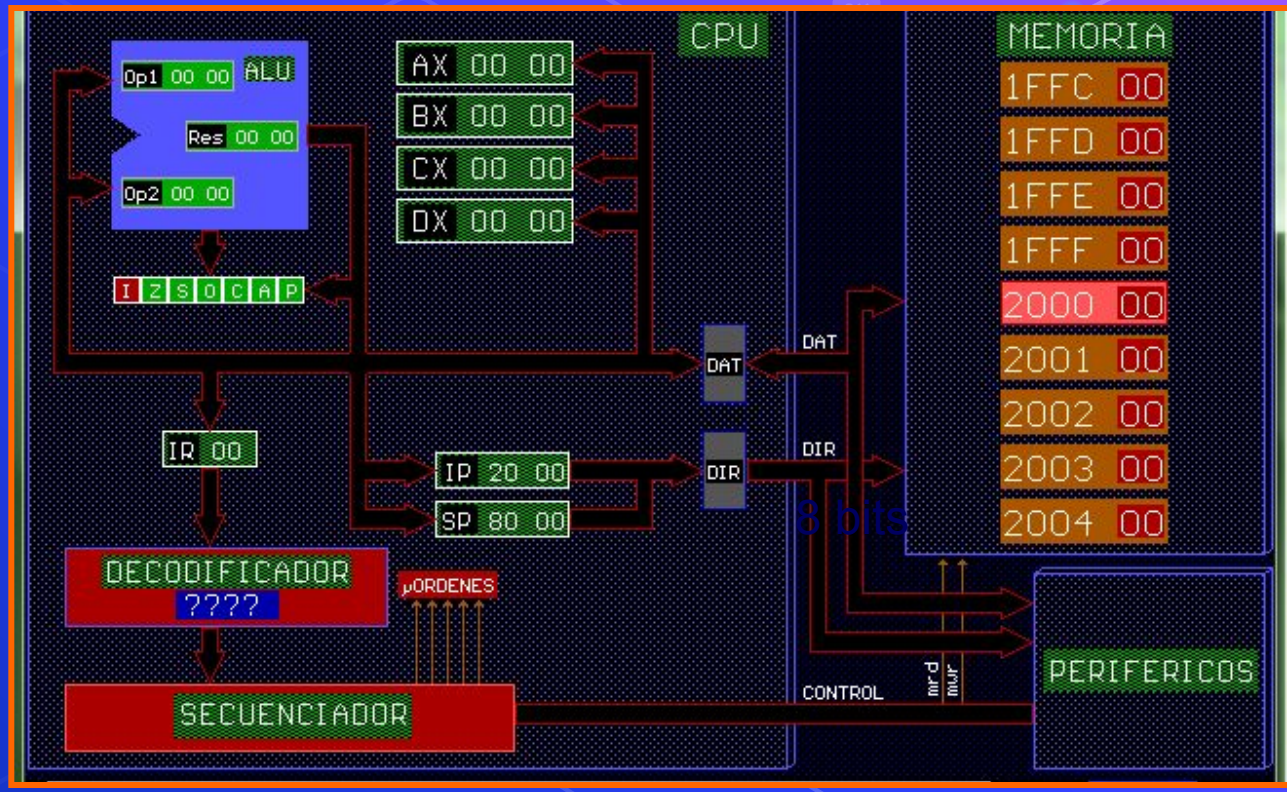
Programación Bajo Nivel - Características

- Al **programar cuidadosamente** en lenguaje ensamblador se pueden crear programas que se ejecutan más rápidamente y ocupan menos espacio que con lenguajes de alto nivel.
- Con el lenguaje ensamblador **se tiene un control muy preciso de las tareas realizadas por un microprocesador** ya que en el lenguaje ensamblador se dispone de instrucciones del CPU que generalmente no están disponibles en los lenguajes de alto nivel.
- Se **puede controlar el tiempo en que tarda una rutina** en ejecutarse, e impedir que se interrumpa durante su ejecución

8088 Conexión entre los componentes

Enlace a Vonsim:

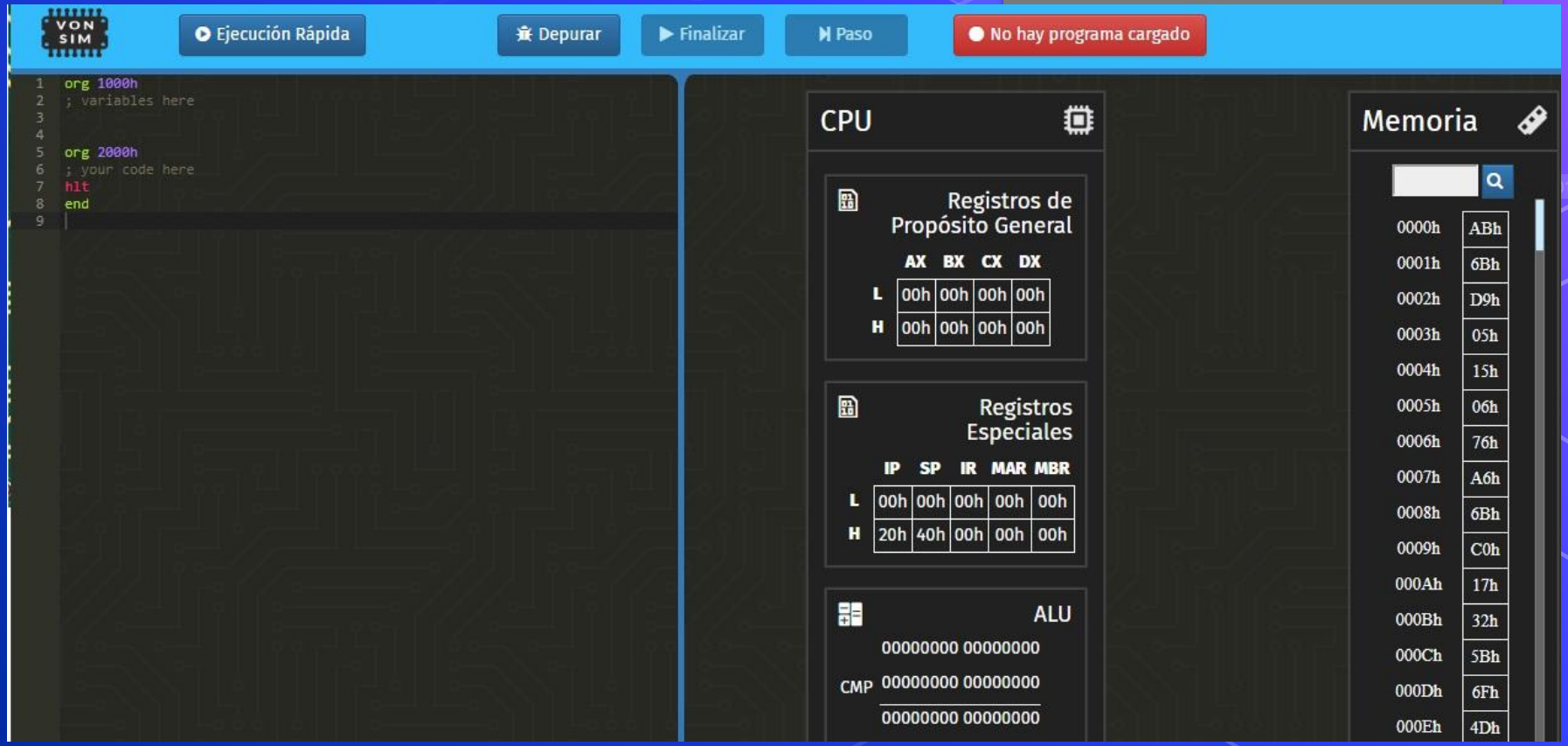
http://facundoq.github.io/unlp/vonsim/assets/index.html



Simulador

Enlace a Vonsim:

http://facundoq.github.io/unlp/vonsim/assets/index.html



Assembler 8088 – Registros

Registros **AX**, **BX**, **CX** y **DX** : uso general, 16 bits de longitud, se pueden dividir en 2 partes de 8 bits cada uno. Ejemplo: **AX** en **AH** y **AL**.

Registro **IP** (**I**nstrucción **P**ointer) contiene la dirección de memoria de la próxima instrucción a ser ejecutada.

Registro **SP** (**S**tack **P**ointer) contiene la dirección de memoria del tope de la pila.

Assembler 8088 – Registros

Registro de **flags**: muestra el estado de las banderas o flags luego de cada operación.

- Bandera de cero: identificada por la letra **Z**.
- Bandera de overflow: identificada por la letra **O**.
- Bandera de carry/borrow: identificada por la letra **C**.
- Bandera de signo del número: identificada por la letra **S**.

Orden de los bytes

Número de 64 bits a representar en memoria de 8 bits: **98765432h**

¿?

Big Endian

Dir. memoria	Contenido
0000	98
0001	76
0002	54
0003	32

El byte más significativo se coloca en la dirección de memoria con valor numérico más bajo

Little Endian

Dir. memoria	Contenido
0000	32
0001	54
0002	76
0003	98

Usaremos esta representación

El byte menos significativo se coloca en la dirección de memoria con valor numérico más bajo

Assembler 8088 – Definición de variables

nombre_variable especificador_tipo valor_inicial

Especificador	Tipo	Tamaño
DB	Byte	8 bits
DW	Word	16 bits

Var1 DB 10
Var2 DW 0A000h

Podemos observar que los valores numéricos se interpretan en decimal, a menos que terminen con una letra 'h', que en cuyo caso se interpretarán como valores en hexadecimal. Además, como los números deben comenzar con un dígito decimal, en el caso del A000h, se **antepone un cero para evitar que se la confunda con una variable que se pueda llamar A000h.**

Memoria de Datos y Memoria de Programa

¿Cómo defino/diferencio cada zona?

Utilizando la instrucción ORG

Ejemplo

```
ORG 1000
total DW 5432h
cont DW 0000
ORG 2000
MOV AX,cont
```

Ver ejercicio 1

La **memoria de datos** es una zona de la memoria RAM donde se cargan los datos /variables que se van a utilizar a lo largo de programa

La **memoria de instrucciones** es una zona de la memoria RAM donde se cargan las instrucciones del programa para ser ejecutado

Dir. memoria	Contenido
1000	32
1001	54
1002	00
1003	00
...	...
2000	FA
2001	O1
2002	C2
2003	AB
2003	03

Assembler 8088 – Modos de Direccionamiento

INMEDIATO

MOV AX, 1000h

El operando contiene la información sobre la que hay que operar. (útil para inicializar registros)

DIRECTO DE MEMORIA O ABSOLUTO

MOV BL, var_byte

La instrucción contiene la dirección de memoria exacta donde se encuentra el operando. El operando se encuentra en memoria.

DIRECTO DE REGISTRO

MOV BX, AX

El operando se encuentra contenido en un registro

INDIRECTO CON REGISTRO

MOV AX, [BX]

La instrucción contiene una dirección que se emplea para leer en memoria una dirección intermedia que será la verdadera dirección del objeto buscado. El operando se encuentra en memoria.

Ver ejercicio 2

INDIRECTO CON DESPLAZAMIENTO

MOV AX, 20h+[BX]

Similar al anterior al que se le agrega un desplazamiento para obtener la dirección final donde se encuentra el operando.

Definición constantes

- Se definen con la instrucción **EQU**

NOMBRE_CONSTANTE EQU valor

El ensamblador reemplazará cualquier ocurrencia indicada, pero dicho valor no va a ocupar ninguna dirección de memoria. Nombre de la constante debe escribirse en mayúscula.

Ejemplo:
MAXIMO EQU 0

Definición de tablas

- Tablas, se definen como

nombre_variable especificador_tipo valores

Ejemplo

tabla DB 1, 2, 4, 8, 16, 32, 64, 128

Esto genera una tabla con los ocho valores especificados, uno a continuación del otro. Esto se puede ver como un arreglo de ocho bytes pero en el que se inicializaron sus celdas con dichos valores.

Definición de tablas

Si quisiéramos definir algo equivalente a un string, podemos aplicar la misma idea de la tabla anterior, en donde en cada celda se almacenaría cada carácter del string.

Sin embargo, escribir los códigos ASCII de cada carácter no simplifica mucho las cosas, así que existe una sintaxis alternativa:

```
string DB “Esto es un String.”
```

MSX88 Instrucciones de transferencia

INSTRUCCIÓN	COMENTARIO	OPERACIÓN	OBS
MOV *dest,fuente*	Copia *fuente* en *dest*	(*dest*)←(*fuente*)	1
PUSH *fuente*	Carga *fuente* en el tope de la pila	(SP)←(SP)-2;[SP+1:SP] ←(*fuente*)	2
POP *dest*	Desapila el tope de la pila y lo carga en *dest*	(*fuente*)←[SP+1:SP]; (SP)←(SP)+2	2
PUSHF	Apila los flags	(SP)←(SP)-2;[SP+1:SP] ←(flags)	-
POPF	Desapila los flags	(flags)←[SP+1:SP]; (SP)←(SP)+2	-
IN *dest,fuente*	Carga el valor en el puerto *fuente* en *dest*	(*dest*)←(*fuente*)	3
OUT *dest,fuente*	Carga en el puerto *dest* el valor en *fuente*	(*dest*)←(*fuente*)	4

1.Las posibilidades para *dest/fuente* son: *reg/reg, reg/mem, reg/op.inm, mem/reg, mem/op.inm. mem* puede ser una etiqueta (dir.directo) o [BX], siendo (BX) una dirección de memoria (dir.indirecto).
2.*dest* y *fuente* solo pueden ser registros de 16 bits.
3. Las posibilidades para *dest/fuente* son: *AL/mem, AX/mem, AL/DX, AX/DX. mem* debe ser una dirección entre 0 y 255. Puede ser un operando inmediato o una etiqueta.
4. Las posibilidades para *dest/fuente* son: *mem/AL, mem/AX, DX/AL, DX/AX. mem* debe ser una dirección entre 0 y 255. Puede ser un operando inmediato o una etiqueta.

MSX88 Instrucciones aritmético - lógicas

INSTRUCCIÓN	COMENTARIO	OPERACIÓN	OBS
ADD *dest,fuente*	Suma *fuente* y *dest*	(*dest*)← (*dest*)+(*fuente*)	1
ADC *dest,fuente*	Suma *fuente, dest y flag C*	(*dest*)← (*dest*)+(*fuente*)+C	1
SUB *dest,fuente*	Resta *fuente a dest*	(*dest*)← (*dest*)-(*fuente*)	1
SBB *dest,fuente*	Resta *fuente y flag C a dest*	(*dest*)← (*dest*)-(*fuente*)-C	1
CMP *dest,fuente*	Compara *fuente* con *dest*	(*dest*)-(*fuente*)	1
NEG *dest*	Negativo de *dest*	(*dest*)← CA2(*dest*)	5
INC *dest*	Incrementa *dest*	(*dest*)← (*dest*)+1	5
DEC *dest*	Decrementa *dest*	(*dest*)← (*dest*)-1	5
AND *dest,fuente*	Operación *fuente* AND *dest* bit a bit	(*dest*)← (*dest*) AND (*fuente*)	1
OR *dest,fuente*	Operación *fuente* OR *dest* bit a bit	(*dest*)← (*dest*) OR (*fuente*)	1
XOR *dest,fuente*	Operación *fuente* XOR *dest* bit a bit	(*dest*)← (*dest*) XOR (*fuente*)	1
NOT *dest*	Complemento a 1de *dest*	(*dest*)← CA1(*dest*)	5

1.Las posibilidades para *dest/fuente* son: *reg/reg, reg/mem, reg/op.inm, mem/reg, mem/op.inm. mem* puede ser una etiqueta (dir.directo) o [BX], siendo (BX) una dirección de memoria (dir.indirecto).

5.*dest* solo puede ser *mem* o *reg. mem* puede ser una etiqueta (dir.directo) o [BX], siendo (BX) una dirección de memoria (dir.indirecto).

MSX88 Instrucciones de control

INSTRUCCIÓN	COMENTARIO	OPERACIÓN	OBS
CALL *etiqueta*	Llama a subrutina cuyo inicio es *etiqueta*		6
RET	Retorna de la subrutina		-
JZ *etiqueta*	Salta si el último valor calculado es cero	Si Z=1, (IP)←*mem*	6
JNZ *etiqueta*	Salta si el último valor calculado no es cero	Si Z=0, (IP)←*mem*	6
JS *etiqueta*	Salta si el último valor calculado es negativo	Si S=1, (IP)←*mem*	6
JNS *etiqueta*	Salta si el último valor calculado no es negativo	Si S=0, (IP)←*mem*	6
JC *etiqueta*	Salta si el último valor calculado produjo carry	Si C=1, (IP)←*mem*	6
JNC *etiqueta*	Salta si el último valor calculado no produjo carry	Si Z=1, (IP)←*mem*	6
JO *etiqueta*	Salta si el último valor calculado produjo overflow	Si O=1, (IP)←*mem*	6
JNO *etiqueta*	Salta si el último valor calculado no produjo overflow	Si O=0, (IP)←*mem*	6
JMP *etiqueta*	Salto incondicional a *etiqueta*	(IP)←*mem*	6

6.*mem* es la dirección de memoria llamada *etiqueta*.

Instrucción ORG

```
            ORG 1000h
contador    DW 1234h
cantidad    DB 0
                ORG 2000h
arreglo     DB 0A0h, 15, 0Fh, 15
cadena      DB "Un string es un arreglo de bytes."
            END
```

¿Cómo se ve en memoria de datos?

1000h	contador	34h
1001h	contador	12h
1002h	cantidad	00h
2000h	arreglo	A0h
2001h	arreglo	0Fh
2002h	arreglo	0Fh
2003h	arreglo	0Fh
2004h	cadena	U
2005h	cadena	n
2006h	cadena	
2007h	cadena	s
2008h	cadena	t

Instrucción Move

```
 ORG 1000h
var_byte  DB 20h
var_word DW ?
 ORG 2000h
 MOV AX, 1000h
 MOV BX, AX
 MOV BL, var_byte
 MOV var_word, BX
 END
```

¿Cómo se ven los registros de la CPU en cada momento de ejecución del programa?

Instante	AX		BX	
	AH	AL	BH	BL
0	00	00	00	00
1	**10**	**00**	00	00
2	10	00	**10**	**00**
3	10	00	10	**20**
4	10	00	10	20

Inconvenientes en ADD y SUB

- Supongamos que queremos **sumar valores de 32 bits**. Dado que nuestra CPU opera con valores de 8 o 16 bits, no sería posible hacerlo en un solo paso. Sin embargo, podríamos sumar la parte baja (los 16 bits menos significativos) por un lado y la parte alta (los 16 bits más significativos) por otro usando dos instrucciones ADD.

- El **problema** se presenta cuando se produce un **acarreo** al realizar la suma en la parte baja, ya que no podemos simplemente ignorarlo pues el resultado no sería el correcto, como se muestra en este ejemplo:

```
   Correcto:                Incorrecto:
     0015  FFFF                0015        FFFF
+    0002  0011          +     0002   +    0011
     0018  0010                0017        1 0010   ———— 0017 0010
```

Uso de ADC

- Para resolver el problema del acarreo se usa ADC (carry)o SUB (borrow), que suman el acarreo o borrow según corresponda.

```
ORG 1000h
dato1_l DW 0FFFFh
dato1_h DW 0015h
dato2_l DW 0011h
dato2_h DW 0002h

ORG 2000h
MOV AX, dato1_l
ADD AX, dato2_l
MOV BX, dato1_h
ADC BX, dato2_h
END
```

Instante	AX	BX	Flag C
0	0000	0000	0
1	FFFF	0000	0
2	0010	0000	1
3	0010	0015	1
4	0010	0018	0

Operador OFFSET

- Este operador **permite obtener la dirección de una etiqueta**.

Por ejemplo si la etiqueta TOTAL esta en la dirección de memoria 2034h al colocar

OFFSET TOTAL

El ensamblador reemplaza a OFFSET TOTAL por 2034h