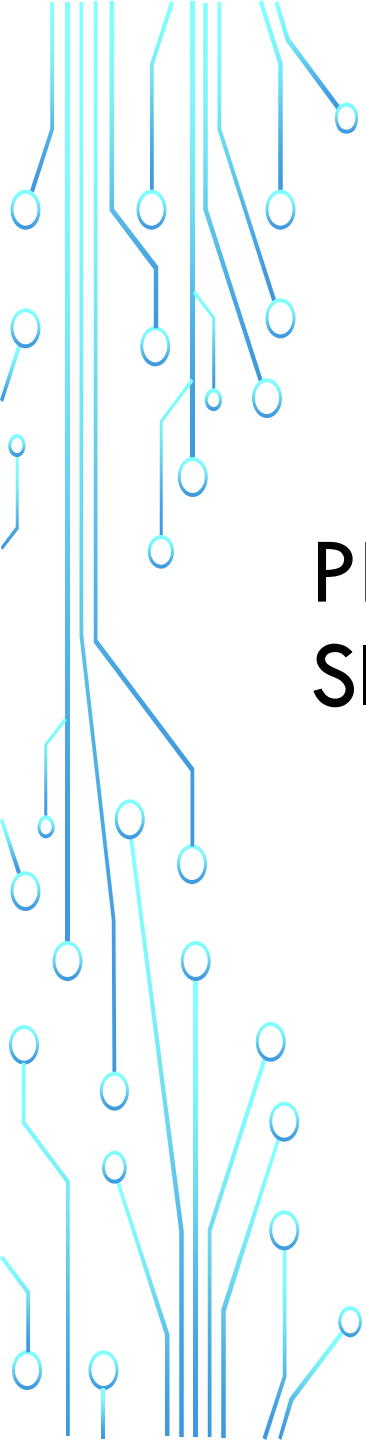


PROGRAMACIÓN EN ASSEMBLER SIMULADOR MSX88

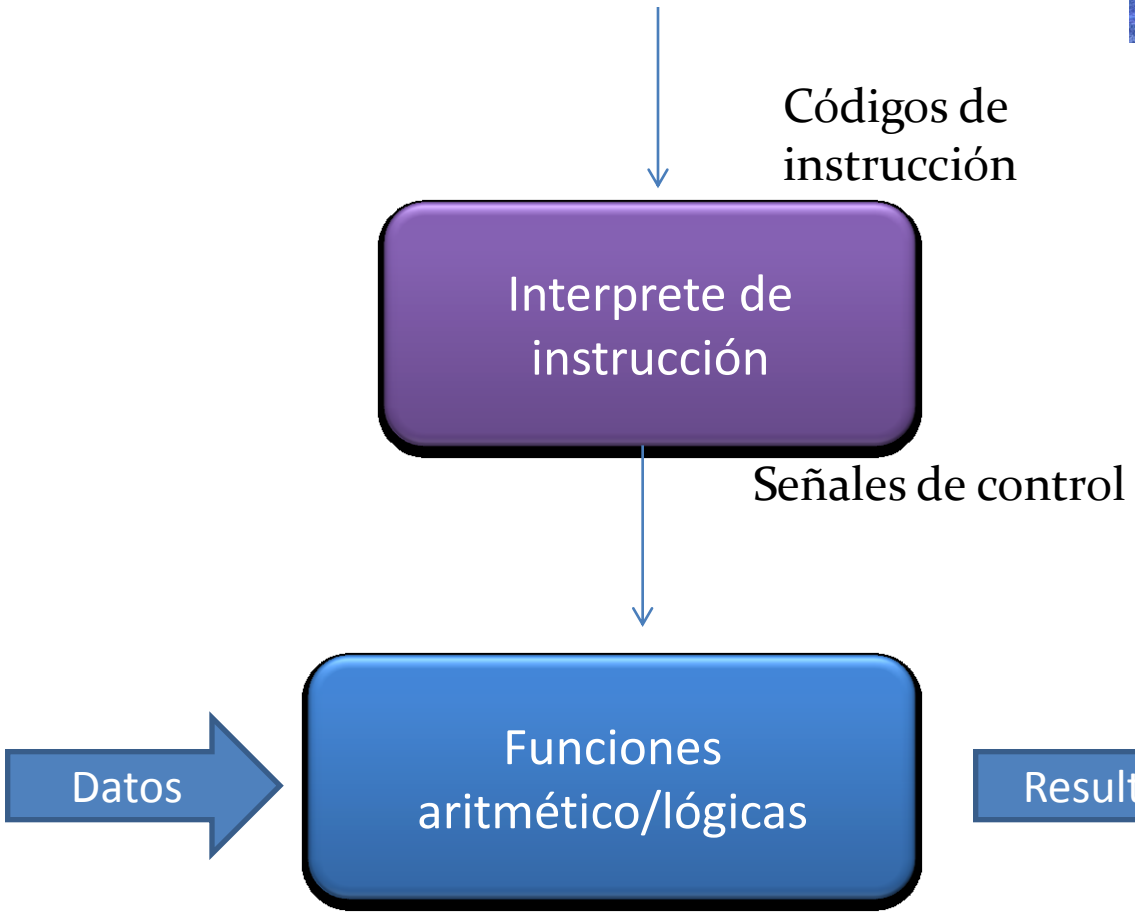
REPASO CONCEPTO DE PROGRAMA

- ANTES se tenían **<u>sistemas cableados</u>**

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

REPASO CONCEPTO DE PROGRAMA

AHORA



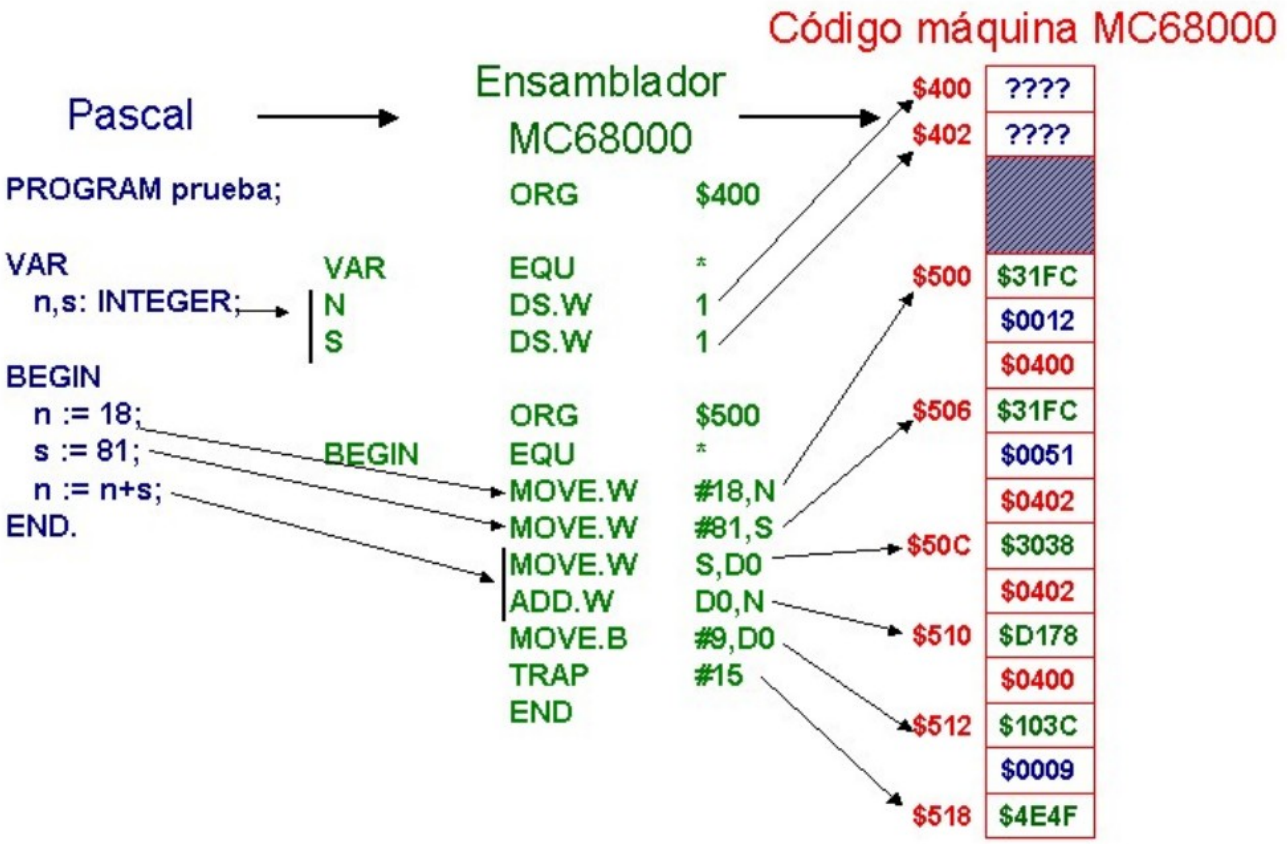
- Programación en software: en cada paso se efectúa alguna operación sobre los datos

REPASO CONCEPTO DE PROGRAMA

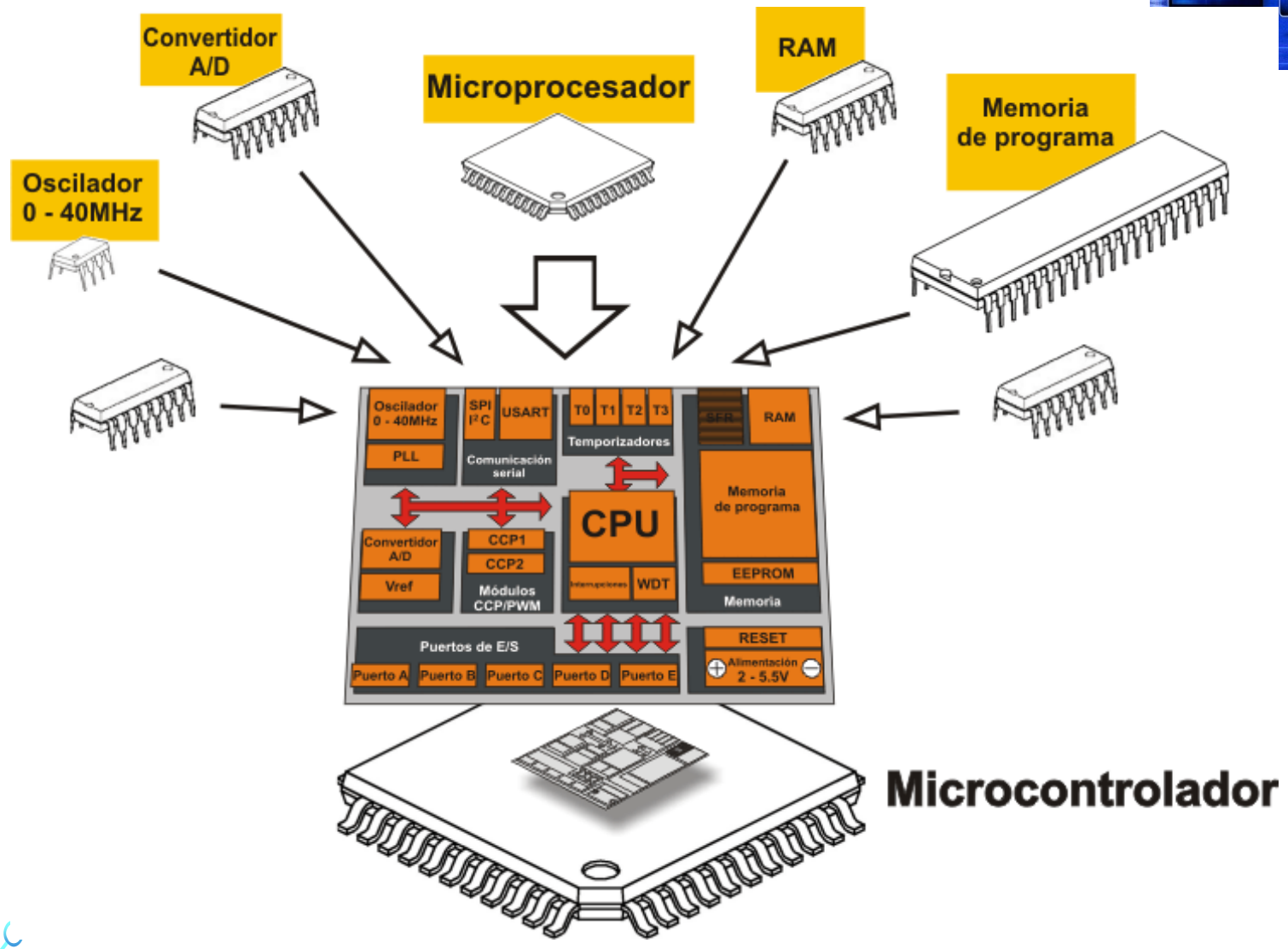
- Para cada paso se necesita un nuevo conjunto de señales de control.
- Las instrucciones proporcionan esas señales de control.
- Aparece el nuevo concepto de programación.

No hay que cambiar el hardware !!!

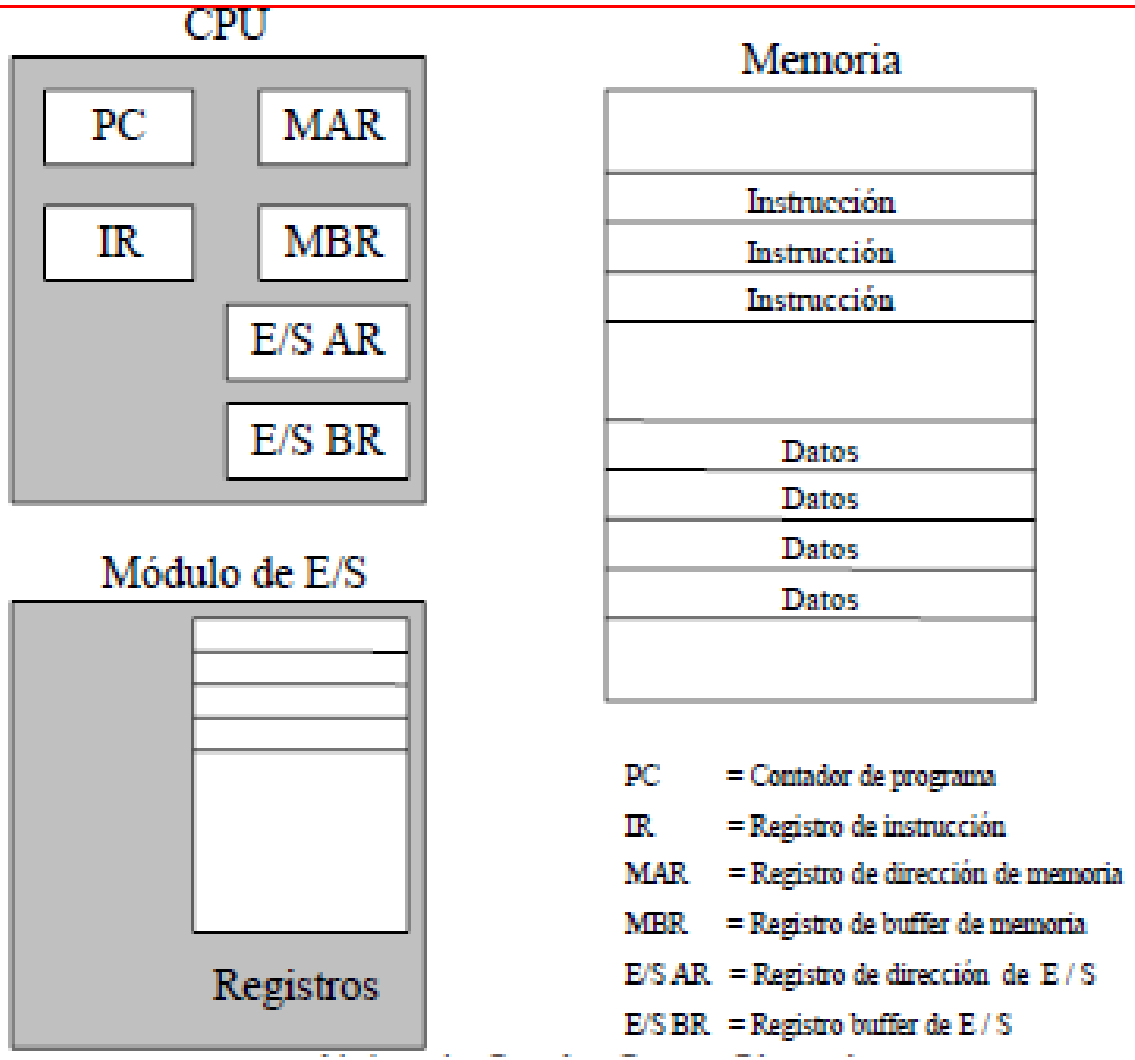
DE LENGUAJES DE ALTO NIVEL A BAJO NIVEL



MICROCONTROLADORES



ESQUEMA GENERAL DE COMPONENTES DE UNA COMPUTADORA



PC = Contador de programa
IR = Registro de instrucción
MAR = Registro de dirección de memoria
MBR = Registro de buffer de memoria
E/S AR = Registro de dirección de E / S
E/S BR = Registro buffer de E / S

DEL ALTO NIVEL AL LENGUAJE DE MAQUINA

Se llaman <u>ensambladores</u> a los programas encargados de traducir los programas escritos en mnemónico a lenguaje binario.

PROGRAMACIÓN A BAJO NIVEL CARACTERISTICAS

- 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 aprovecha 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 A 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 A BAJO NIVEL – CARACTERÍSTICAS

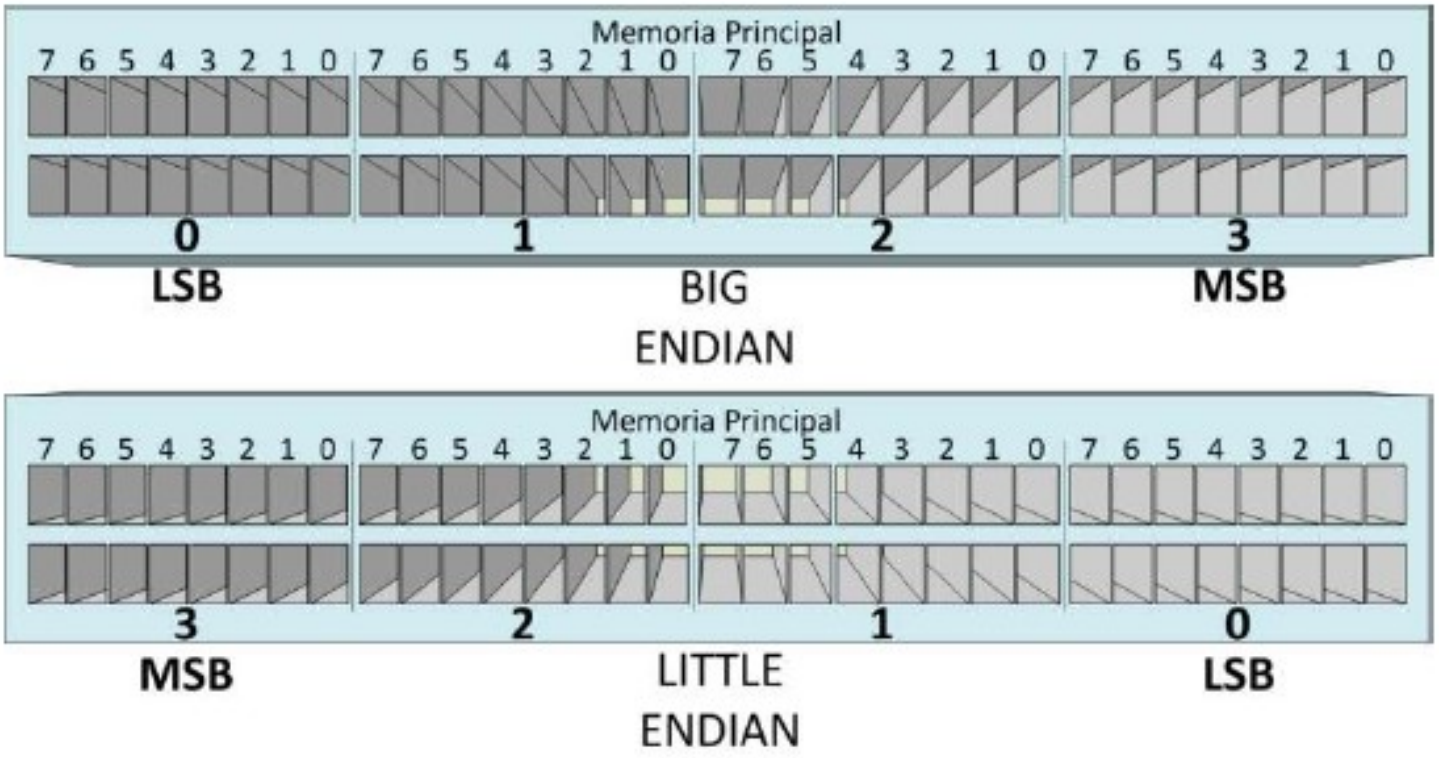
- Al **programar cuidadosamente** en lenguaje ensamblador se pueden crear programas que se ejecutan más rápidamente y ocupan menos espacio que con lenguajes de alto nivel.
- Con el lenguaje ensamblador **se tiene un control muy preciso de las tareas realizadas por un microprocesador** ya que en el lenguaje 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

ORDEN DE LOS BYTES

Supongamos una memoria **direccionable de a byte**

¿En qué orden se leen aquellos números que ocupan más de un byte?

Ejemplo: La palabra doble 98765432H (32 bits) se puede almacenar en 4 bytes consecutivos de las siguientes 2 formas:



ORDEN DE LOS BYTES

Dir. de byte	Forma 1	Forma 2
00	98	32
01	76	54
02	54	76
03	32	98

¿cuál forma uso?

Big endian: el byte más significativo en la dirección con valor numérico más bajo

Little endian: el byte menos significativo en la dirección con valor numérico más bajo

ORDEN DE LOS BYTES

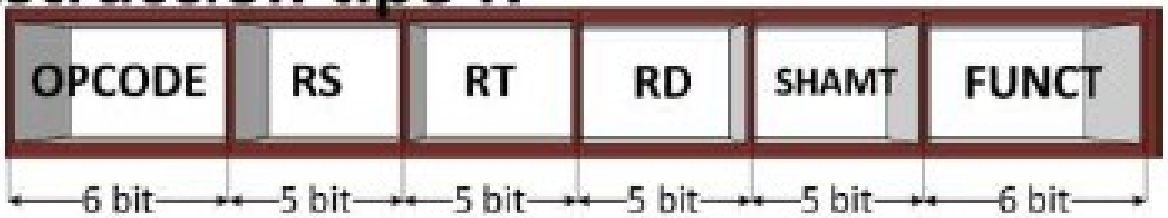
- Intel 80x86, Pentium y VAX son "little-endian".
- IBM S/370, Motorola 680x0 (Mac) y la mayoría de los RISC son "big-endian".

Incompatibilidad !!!

FORMATOS DE INSTRUCCIÓN

Instrucción tipo I

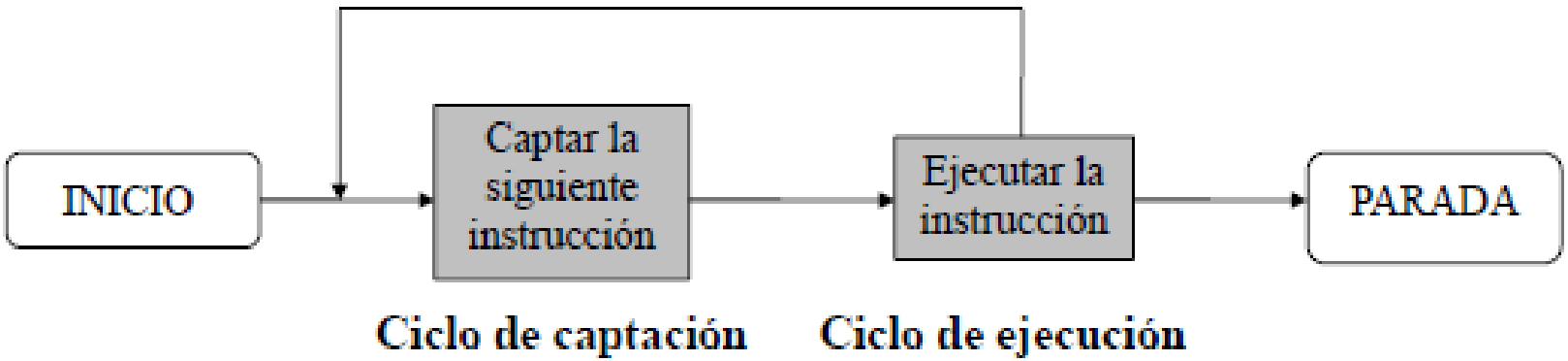
Instrucción tipo R



Instrucción tipo J

CICLO DE INSTRUCCIÓN BÁSICO

- Dos pasos:
  - Captación
  - Ejecución



CICLO DE CAPTACIÓN

- La dirección de la instrucción que se debe captar se encuentra en el registro Contador de Programa (PC)
- La UC **capta** la instrucción desde la Memoria
  - La instrucción va al registro de instrucción (IR)
- El registro PC se incrementa
  - a no ser que se indique lo contrario.
- La UC interpreta la instrucción captada y debe lleva a cabo la acción requerida

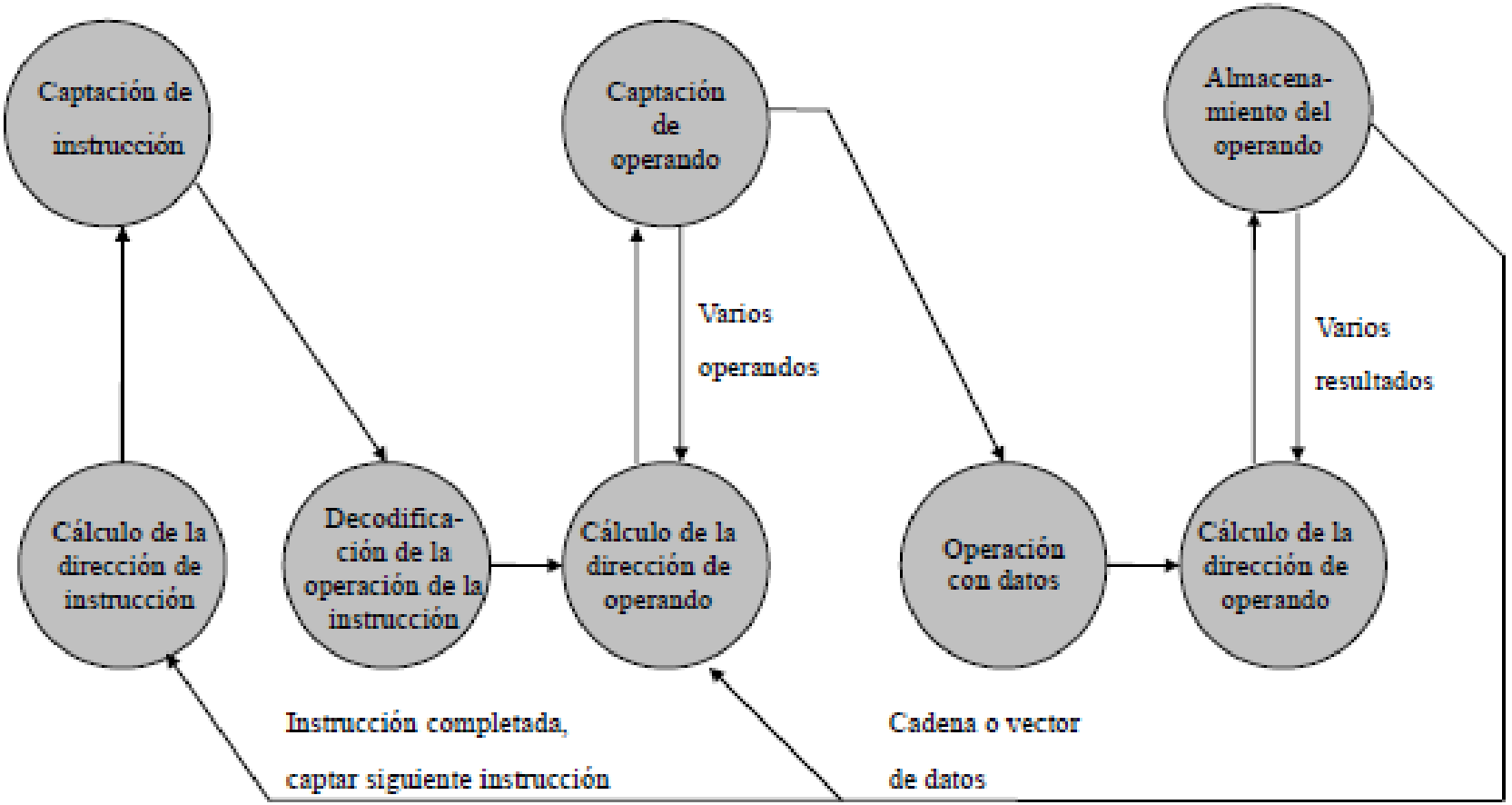
CICLO DE EJECUCIÓN

Acciones posibles:

- Procesador - memoria
  - Transferencia de datos CPU - Memoria.
- Procesador - E/S
  - Transferencias de datos CPU y módulo de E/S.
- Procesamiento de datos
  - Alguna operación aritmética o lógica con los datos.
- Control
  - Alteración de la secuencia de ejecución.
    - Instrucción de salto

ó combinación de las acciones anteriores

DIAGRAMA DE ESTADOS DEL CICLO DE INSTRUCCIÓN



REPERTORIO (SET) DE INSTRUCCIONES

Es el conjunto completo de instrucciones que se realizan en una CPU.

- Código máquina con nombre mnemónico para no tener que escribir en binario

Representado simbólicamente por un conjunto de códigos de ensamblaje

- **de operaciones**: ADD (sumar), SUB (restar), LOAD (cargar datos en un registro)
- **de operandos:** ADD BX, PEPE; se interpreta como sumar contenidos de reg BX y dirección PEPE, el resultado se guarda en reg BX

¿CÓMO SE COMPONE UNA INSTRUCCIÓN?

- Código de operación (“Cod Op”)
- Referencia a operandos fuentes
- Referencia al operando resultado
- Referencia a la siguiente instrucción

¿DÓNDE SE ALMACENAN LOS OPERANDOS?

- Memoria principal
- Memoria virtual o en memoria cache
- Registro de la CPU
- Dispositivo de E/S

TIPOS DE INSTRUCCIONES

- **Procesamiento de datos:**

  instrucciones aritmético-lógicas

- **Almacenamiento de datos:**

  instrucciones de memoria

- **Transferencia de datos:**

  instrucciones de E/S

- **Control:**

  instrucciones de testeo y flujo del programa

TIPOS DE OPERANDOS

- **Direcciones**
- **Números**

  punto fijo ó punto flotante

- **Caracteres**

  ASCII, EBCDIC ...etc.

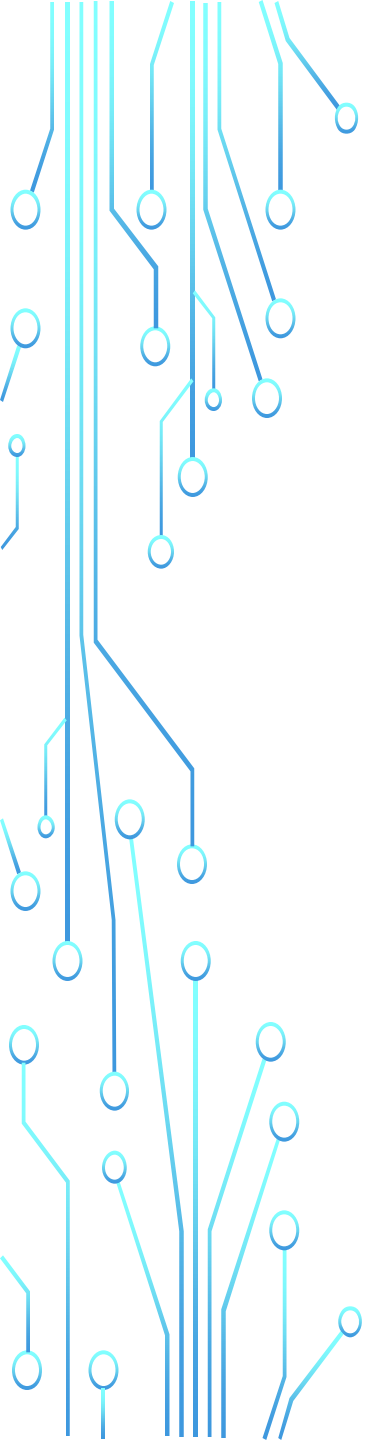
- **Datos lógicos**

  Bits (1 ó 0)

  Ej: flags o indicadores

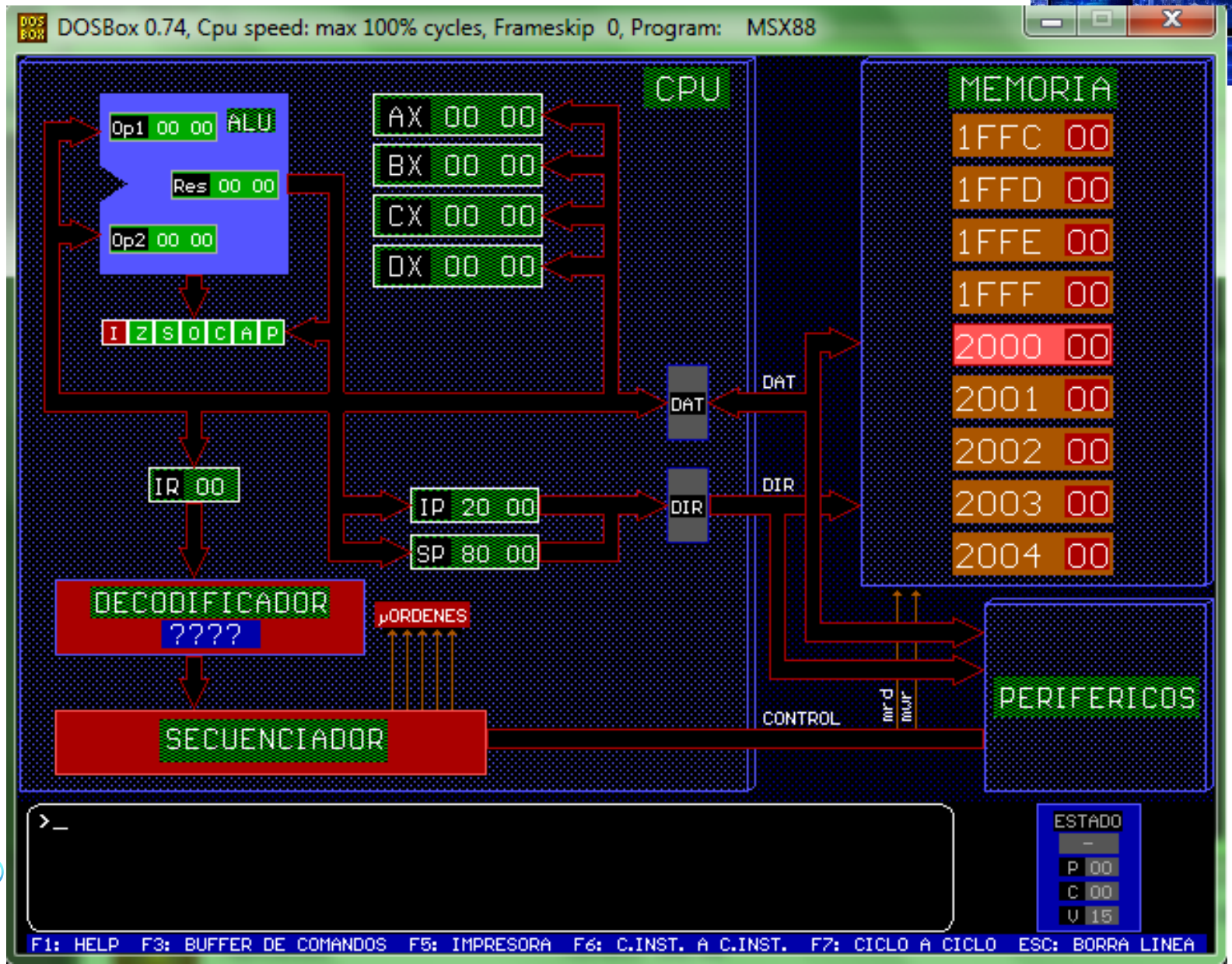
TIPOS DE OPERACIONES

- Transferencias de datos
- Aritméticas
- Lógicas
- Conversión
- Entrada/Salida
- Control del sistema
- Control de flujo



LENGUAJE ENSAMBLADOR

SIMULADOR MSX88



TIPOS DE VARIABLES

- La sintaxis para definir una variables es:

nombre_variable especificador_tipo valor_inicial

- El nombre de la variable debe comenzar con una letra o un underscore (_) seguido por números, letras o underscores.
- El tipo indica el tamaño que ocupa en memoria dicha variable.
- El valor inicial puede no especificarse, usando el carácter '?', lo que le informa al ensamblador que dicho valor será el que se encuentre en la memoria del simulador en el momento de cargar el programa.

Especificador	Tipo	Tamaño
DB	Byte	8 bits
DW	Word	16 bits

Tipos de variables

DEFINICIÓN DE VARIABLES

- Ejemplo de definición de variables:

Var1 DB 10

Otra DW 1034h

➢ 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 llamada A000h.

Cant DW 0A000h

DEFINICIÓN CONSTANTES Y TABLAS

- Constantes . Se definen

  maximo EQU valor

El ensamblador reemplazará cualquier ocurrencia indicada, pero dicho valor no va a ocupar ninguna dirección de memoria,

- Tablas, se defienen como

  nombre_variable especificador_tipo valores

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.”

DEFINICIÓN DE TABLAS

- Si quisiéramos definir una tabla en la que los datos que contienen son iguales o cumplen algún patrón repetitivo, es posible utilizar el modificador DUP en la lista de valores, de la siguiente manera:

cantidad DUP (valores)

En este caso, *cantidad* indica la cantidad de veces que se repiten el o los valores indicados entre paréntesis. Por ejemplo, para definir un arreglo de 20 palabras, inicialmente conteniendo 1234h y 4321h alternadamente, se le indica al ensamblador lo siguiente:

cantidad EQU 10

arreglo DW cantidad DUP (1234h, 4321h)

USO DE ORG

¿Cómo se ve en memoria de datos?

```
ORG 1000h
contador DW 1234h
cantidad DB 0
ORG 2000h
arreglo DB 0A0h, 3 DUP (15)
cadena DB "Un string es un arreglo de bytes."
END
```

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

...

INSTRUCCIONES – USO DEL 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

MSX88 INSTRUCCIONES DE TRANSFERENCIA

1	MOV *dest,fuente*	Copia *fuente* en *dest*	(*dest*)←(*fuente*)
2	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)
2	POPF	Desapila los flags	(flags)←[SP+1:SP]; (SP)←(SP)+2
3	IN *dest,fuente*	Carga el valor en el puerto *fuente* en *dest*	(*dest*)←(*fuente*)
4	OUT *dest,fuente*	Carga en el puerto *dest* el valor en *fuente*	(*dest*)←(*fuente*)

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

MSX88 INSTRUCCIONES ARITMÉTICO - LÓGICAS

1	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*)
5	NEG *dest*	Negativo de *dest*	(*dest*)← CA2(*dest*)
5	INC *dest*	Incrementa *dest*	(*dest*)← (*dest*)+1
5	DEC *dest*	Decrementa *dest*	(*dest*)← (*dest*)-1
1	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*)
5	NOT *dest*	Complemento a 1de *dest*	(*dest*)← CA1(*dest*)

1. *dest/fuente* son: *reg/reg, reg/mem, reg/op.inm, mem/reg, mem/op.inm.*

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 TRANSFERENCIA DE CONTROL

6	CALL *etiqueta*	Llama a subrutina cuyo inicio es *etiqueta*	
6	RET	Retorna de la subrutina	
6	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. *mem* es la dirección de memoria llamada *etiqueta*.