

<http://msx.konamiman.com>

konamiman@konamiman.com



Mini-referencia procesador Z80 y BIOS **MSX**



Compilación de Néstor Soriano

1. REGISTROS DEL Z80

Grupo principal

A (acumulador)	F (banderas)
B	C
D	E
H	L

Grupo alternativo

A'	F'
B'	C'
D'	E'
H'	L'

← 8 bits →

I (vector int.)	R (refresco mem.)
IX	
IY	
SP (puntero de la pila)	
PC (contador de programa)	

} Registros de índice

Banderas (registro F)

7	6	5	4	3	2	1	0
S	Z	-	H	-	P/V	N	C

S = Signo P/V = Paridad/Desbordamiento
 Z = Cero N = Resta
 H = Acarreo mitad C = Acarreo

* SECUENCIAS DE ESCAPE PARA USAR CON CHPUT (00A2H)

A continuación se listan las secuencias de escape que pueden ser usadas para conseguir diversos efectos al imprimir texto por pantalla. Para usarlas, primero hay que imprimir el carácter de escape "Esc" (código 27) y a continuación la secuencia deseada. Se han de respetar las mayúsculas y las minúsculas de las secuencias.

Por ejemplo, el siguiente código borra la pantalla, imprime la cadena "Esto es un ejemplo" en las coordenadas 5,10 y a finalmente sitúa el cursor al principio de la pantalla.

```

CHPUT:    equ    #00A2        ;Rutina BIOS de impresión de un caratcer
          LD     HL,TEXTO
BUCLE:    LD     A,(HL)        ;Imprime los caracteres uno a uno
          OR     A
          RET    Z             ;Termina al encontrar un carácter 0
          CALL  CHPUT
          INC   HL
          JR    BUCLE

TEXTO:    db     27, "E"        ;Borra la pantalla
          db     27, "Y", 5+32, 10+32 ;Sitúa cursor en 5,10
          db     "Esto es un ejemplo" ;Cadena
          db     27, "H"        ;Sitúa cursor al principio
          db     0              ;Fin de la cadena
    
```

- **Esc A:** Cursor arriba
- **Esc B:** Cursor abajo
- **Esc C:** Cursor a la derecha
- **Esc D:** Cursor a la izquierda
- **Esc E:** Borra la pantalla y sitúa el cursor al principio de la misma
- **Esc H:** Sitúa el cursor al principio de la pantalla
- **Esc J:** Borra desde el cursor hasta el final de la pantalla
- **Esc K:** Borra desde el cursor hasta el final de la línea
- **Esc L:** Inserta una línea encima de la línea del cursor
- **Esc I:** Borra la línea del cursor
- **Esc M:** Elimina la línea del cursor
- **Esc x 4:** Selecciona cursor de bloque
- **Esc x 5:** Esconde el cursor
- **Esc Y <x>+32 <y>+32:** Posiciona el cursor en las coordenadas <x>,<y>
- **Esc y 4:** Selecciona cursor de línea
- **Esc y 5:** Muestra el cursor

● **EXTROM** (015FH)

Llama a una rutina de la SUB-ROM (BIOS extendida de los MSX2 y superiores)

Entrada: IX = Dirección de la rutina a llamar
Otros registros: depende de la rutina a llamar

Salida: depende de la rutina a llamar

Registros: depende de la rutina a llamar

● **EOL** (0168H)

Borra el texto hasta el final de la línea

Entrada: H = Coordenada X del cursor, L = Coordenada Y del cursor

Salida: -

Registros: Todos

● **MATRIZ DEL TECLADO PARA USAR EN SNSMAT** (0141H)

Fila/Bit	7	6	5	4	3	2	1	0
0	7	6	5	4	3	2	1	0
1	;]	[\	=	-	9	8
2	B	A	ACENT	/	.	,	`	'
3	J	I	H	G	F	E	D	C
4	R	Q	P	O	N	M	L	K
5	Z	Y	X	W	V	U	T	S
6	F3	F2	F1	CODE	CAPS	GRAPH	CTRL	SHIFT
7	RETURN	SELECT	BS	STOP	TAB	ESC	F5	F4
8	→	↓	↑	←	DEL	INS	HOME	SPACE
<i>Teclado numérico:</i>								
9	4	3	2	1	0	/	+	*
10	.	,	-	9	8	7	6	5

2. DESCRIPCION DE LAS BANDERAS

● **S: Signo.** Esta bandera refleja el signo (el MSB) del resultado de determinadas operaciones aritméticas, lógicas, de rotación o desplazamiento, y de algunas operaciones de transferencia de datos entre registros.

● **Z: Cero.** Se activa si el resultado de la operación realizada previamente es cero (cuidado: el valor de la bandera es UNO cuando el resultado de la operación es CERO). También se usa en instrucciones de comparación, para detectar coincidencias.

● **H: Acarreo mitad.** Se activa cuando hay acarreo del bit 3 al 4 en una operación aritmética. Es usada internamente por el Z80 y raramente resulta útil para el programador.

● **P/V: Paridad/Desbordamiento.** Esta bandera tiene dos funciones distintas, dependiendo de la instrucción. Tras una operación lógica o de rotación/desplazamiento, la bandera se activa si el número de bits puestos a 1 del resultado es par. Tras una operación aritmética se activa si se ha producido desbordamiento (el resultado de la operación no cabe en 8 bits).

● **N: Resta.** Se activa tras una resta, se desactiva tras una suma. Al igual que H, normalmente sólo es útil para el propio Z80.

● **C: Acarreo.** Se activa si una operación de suma o resta produce acarreo. También se usa como noveno bit en las instrucciones de rotación y desplazamiento. Puede manipularse directamente con las instrucciones SCF y CCF.

3. FUNCIONAMIENTO DE LA PILA

PUSH rr equivale a:

$(SP-1) \leftarrow high(rr)$
 $(SP-2) \leftarrow low(rr)$
 $SP \leftarrow SP-2$

POP rr equivale a:

$low(rr) \leftarrow (SP)$
 $high(rr) \leftarrow (SP+1)$
 $SP \leftarrow SP+2$

* Ejemplo *

a) Situación inicial

??	#0FFB
??	#0FFC
??	#0FFD
??	#0FFE
??	#0FFF
SP ⇨ ??	#1000

b) Tras ejecutar:
LD HL,#1122
PUSH HL

??	#0FFB
??	#0FFC
??	#0FFD
SP ⇨ #22	#0FFE
#11	#0FFF
??	#1000

c) Tras ejecutar:
LD IX,#AABB
PUSH IX

??	#0FFB
SP ⇨ #BB	#0FFC
#AA	#0FFD
#22	#0FFE
#11	#0FFF
??	#1000

d) Tras ejecutar:
INC SP
POP BC

??	#0FFB
#BB	#0FFC
#AA	#0FFD
#22	#0FFE
SP ⇨ #11	#0FFF
BC = #22AA	??
??	#1000

* PHYDIO (0144H)

Acceso físico al disco (lectura y escritura de sectores)

Entrada: A = Unidad (0=A:, 1=B:, ..., 7=H:)

B = Número de sectores a leer o escribir

C = Identificador del medio *

DE = Primer número de sector a leer o escribir

HL = Dirección del bufer de lectura o escritura en RAM

CY = 1 para escribir sectores, 0 para leer

Salida: CY = 1 en caso de error

B = Número de sectores que han podido ser leídos o escritos

A = Código de error (sólo si CY=1):

0 = Write protected

2 = Not ready

4 = Data error

6 = Seek error

8 = Record not found

10 = Write error

12 = Bad parameter

14 = Out of memory

16 = Other error

Registros: Todos

* El descriptor del medio suele ser #F9 para diskettes 2DD, #F8 para disketes 1DD, #FF para disco RAM, y #F0 para particiones de disco duro. En realidad no es realmente necesario establecer este parámetro, y de hecho en las controladoras IDE y SCSI este valor se interpreta como el tercer byte del número de sector si su séptimo bit es 0, posibilitando así el empleo de números de sector de 23 bits; sin embargo el sistema operativo MSX-DOS está limitado a números de sector de 16 bits.

* KILBUF (0156H)

Borra el búfer del teclado

Entrada: -

Salida: -

Registers: HL

* CALBAS (0159H)

Ejecuta una rutina del intérprete BASIC mediante una llamada interslot

Entrada: IX = Dirección de la rutina a llamar

Otros registros: depende de la rutina a llamar

Salida: depende de la rutina a llamar

Registros: depende de la rutina a llamar

• **DSPFNK** (00CFH)

Activa la exhibición del contenido de las teclas de función en pantalla

Entrada: -

Salida: -

Registros: Todos

• **RSLREG** (0138H)

Lee el contenido del registro selector de slots primario (puerto #A8)

Entrada: -

Salida: A = Valor del registro

Registros: A

• **WSLREG** (013BH)

Escribe en el registro selector de slots primario (puerto #A8)

Entrada: A = Valor a escribir

Salida: -

Registros: -

• **SNSMAT** (0141H)

Comprueba el estado de una tecla determinada. Algunas teclas como SHIFT, CTRL, GRAPH... no pueden ser leídas mediante CHGET o rutinas similares, y la única forma de comprobar si están siendo pulsadas es el uso de esta rutina.

Entrada: A = Fila del teclado a leer (ver matriz del teclado al final de la sección)

Salida: A = Estado de la fila del teclado. El bit correspondiente a cada tecla está a 0 si la tecla está pulsada, en caso contrario está a 1 (ver matriz del teclado al final de la sección)

Registros: AF, C

Ejemplo: Para comprobar si se está pulsando CTRL, llamar a SNSMAT con A=6, y después comprobar si el bit 1 de A está a 0.

4. INSTRUCCIONES DEL Z80

Leyenda para las banderas:

• - Cambia funcionalmente según la operación

. - No cambia

0 - Puesta a 0

1 - Puesta a 1

? - Cambia aleatoriamente

P/V - Significado de la bandera, paridad o desbordamiento

Una casilla de banderas en blanco significa que se repite la casilla de la instrucción anterior

• ADD with Carry - Suma con acarreo

<u>Nemónico</u>	<u>Operación</u>	<u>CZPSNH</u>	<u>Comentarios</u>
ADC A,r	A:=A+r+CY	••V•0•	r - A,B,C,D,E,H,L,IXh,IXl,IYh,IYl
ADC A,(HL) . . .	A:=A+(HL)+CY		n - byte (0..FF)
ADC A,n	A:=A+n+CY		ii - IX,IY
ADC A,(ii+n) . .	A:=A+(ii+n)+CY	••V•0?	rr - BC,DE,HL,SP
ADC HL,rr	HL:=HL+rr+CY		

• ADD - Suma

<u>Nemónico</u>	<u>Operación</u>	<u>CZPSNH</u>	<u>Comentarios</u>
ADD A,r	A:=A+r	••V•0•	
ADD A,(HL) . . .	A:=A+(HL)		
ADD A,n	A:=A+n		
ADD A,(ii+n) . .	A:=A+(ii+n)		
ADD HL,rr	HL:=HL+rr	• . . . 0?	
ADD IX,ry	IX:=IX+px		ry - BC,DE,SP,IY
ADD IY,rx	IY:=IY+py		rx - BC,DE,SP,IX

• AND - Operación lógica "Y"

<u>Nemónico</u>	<u>Operación</u>	<u>CZPSNH</u>	<u>Comentarios</u>
AND r	A:=A and r	0•P•01	
AND (HL)	A:=A and (HL)		
AND n	A:=A and n		
AND (ii+n) . . .	A:=A and (ii+n)		

• **BIT** - Verificación de un bit

<u>Nemónico</u>	<u>Operación</u>	<u>CZPSNH</u>	<u>Comentarios</u>
BIT b,r	Z:=not rb	.●??01	b-num. bit (0..7)
BIT b,(HL) ...	Z:=not (HL)b		xb - bit b de la
BIT b,(ii+n) .	Z:=not (ii+n)b		posicion x

• **CALL** - Llamada a una subrutina

<u>Nemónico</u>	<u>Operación</u>	<u>CZPSNH</u>	<u>Comentarios</u>
CALL nn	PUSH PC;PC:=nn	nn-word (0..FFFF)
CALL cc,nn ...	If cc then CALL nn else continue		cc - C,NC,Z,NZ, M,P,PE,PO

• **Complement Carry Flag** - Complementa el flag de acarreo

<u>Nemónico</u>	<u>Operación</u>	<u>CZPSNH</u>	<u>Comentarios</u>
CCF	CY:=not CY	*...0?	

• **ComPare** - Comparación del acumulador con otro valor

<u>Nemónico</u>	<u>Operación</u>	<u>CZPSNH</u>	<u>Comentarios</u>
CP r	A-r	●●V●1●	CY=1 si A<x
CP (HL)	A-(HL)		Z=1 si A=x,
CP n	A-n		y lo mismo en
CP (ii+n)	A-(ii+n)		CPD(R), CPI(R)

• **ComPare with Increment/Decrement [and Repeat]** -
Comparación con autoincremento/autodecremento [y repetición]

<u>Nemónico</u>	<u>Operación</u>	<u>CZPSNH</u>	<u>Comentarios</u>
CPD	A-(HL);dec HL;dec BC	●●●1●	PV=0 si BC=0
CPDR	Repite CPD hasta Z=1 o BC=0		
CPI	A-(HL);inc HL;dec BC		PV=0 si BC=0
CPIR	Repite CPI hasta Z=1 o BC=0		

• **PINLIN** (00AEH)

Lee caracteres desde el teclado y los introduce en un bufer, hasta que el usuario pulsa ENTER o STOP

Entrada: -

Salida: HL = Dirección del bufer menos uno
CY = 1 si el usuario ha pulsado STOP

Registros: Todos

• **BREAKX** (00B7H)

Comprueba si se está pulsando CTRL+STOP

Entrada: -

Salida: CY=1 si el usuario está pulsando CTRL+STOP

Registros: AF

• **BEEP** (00C0H)

Genera un sonido breve

Entrada: -

Salida: -

Registros: Todos

• **CLS** (00C3H)

Borra la pantalla

Entrada: Z=1

Salida: -

Registros: AF, BC, DE

• **POSIT** (00C6H)

Posiciona el cursor

Entrada: H = Coordenada X, L = Coordenada Y

Salida: -

Registros: AF

• **ERAFNK** (00CCH)

Inhibe la exhibición del contenido de las teclas de función en pantalla

Entrada: -

Salida: -

Registros: Todos

• **CALLF** (0030H)

Realiza una llamada interslot (llama a una rutina almacenada en cualquier slot). Se usa de la siguiente forma:

```
RST   #30   ;o bien CALL #30
DB    x     ;slot de la rutina, como en RDSL T
DW    xx    ;Dirección de la rutina
```

Entrada: depende de la rutina a llamar

Salida: depende de la rutina a llamar

Registros: AF; para los demás, depende de la rutina a llamar

• **CHSNS** (009CH)

Comprueba el estado del bufer del teclado

Entrada: -

Salida: Z=1 si el bufer está vacío, en caso contrario Z=0

Registros: AF

• **CHGET** (009FH)

Lee un carácter del teclado (con espera)

Entrada: -

Salida: A = Código ASCII del carácter introducido

Registros: AF

• **CHPUT** (00A2H)

Imprime un carácter en pantalla.

Entrada: A = Carácter a imprimir

Salida: -

Registros: -

• **LPTOUT** (00A5H)

Envía un carácter a la impresora.

Entrada: A = Carácter a imprimir

Salida: CY=1 en caso de error

Registros: F

• **LPTSTT** (00A8H)

Obtiene el estado de la impresora

Entrada: -

Salida: A=255 y Z=0 si la impresora está lista
A=0 y Z=1 si la impresora no está lista

Registros: AF

• **ComPLEMENT** - Complementa el acumulador (invierte todos los bits)

<u>Nemónico</u>	<u>Operación</u>	<u>CZPSNH</u>	<u>Comentarios</u>
CPL	A:=A xor 25511	

• **Decimal Adjust of Accumulator** -

Ajuste del acumulador tras una operación BCD

<u>Nemónico</u>	<u>Operación</u>	<u>CZPSNH</u>	<u>Comentarios</u>
DAA	Decimal adjust Acc.	••P•••	

• **DECrement** - Resta uno

<u>Nemónico</u>	<u>Operación</u>	<u>CZPSNH</u>	<u>Comentarios</u>
DEC r	r:=r-1	••V•1•	
DEC (HL)	(HL):=(HL)-1		
DEC (i+n) ...	(i+n):=(i+n)-1		
DEC rr	rr:=rr-1	
DEC ii	ii:=ii-1		

• **Disable Interrupts** - Inhabilita las interrupciones

<u>Nemónico</u>	<u>Operación</u>	<u>CZPSNH</u>	<u>Comentarios</u>
DI	IFF:=0	

• **Decrease and Jump if Not Zero** -

Decrementa B y efectúa un salto si no ha llegado a 0

<u>Nemónico</u>	<u>Operación</u>	<u>CZPSNH</u>	<u>Comentarios</u>
DJNZ e	dec B;si B>0 JR e, si B=0 continuar	e - dir. relativa

• **Enable Interrupts** - Habilita las interrupciones

<u>Nemónico</u>	<u>Operación</u>	<u>CZPSNH</u>	<u>Comentarios</u>
EI	IFF:=1	

• **EXchange** - Intercambia registros

<i>Nemónico</i>	<i>Operación</i>	<i>CZPSNH</i>	<i>Comentarios</i>
EX AF,AF'	AF<->AF'	●●●●●	
EX DE,HL	DE<->HL		
EX (SP),HL	(SP)<->HL		
EX (SP),ii	(SP)<->ii		
EXX	BC<->BC';DE<->DE'; HL<->HL'		

• **HALT** - Detiene la CPU hasta recibir una interrupción

<i>Nemónico</i>	<i>Operación</i>	<i>CZPSNH</i>	<i>Comentarios</i>
HALT	Detiene CPU	

• **Interrupt Mode** - Cambia el modo de interrupción del Z80

<i>Nemónico</i>	<i>Operación</i>	<i>CZPSNH</i>	<i>Comentarios</i>
IM 1	Modo interrupción 1	
IM 2	Modo interrupción 2		
IM 3	Modo interrupción 3		

• **INput** - Entrada de datos desde un puerto

<i>Nemónico</i>	<i>Operación</i>	<i>CZPSNH</i>	<i>Comentarios</i>
IN A,(n)	A:=port(n)	
IN r,(C)	r:=port(C)	●P●0●	
IN (C)	sólo establece flags como IN r,(C)	●P●0●	instrucción oculta opcode: ED 70

• **INCrement** - Suma uno

<i>Nemónico</i>	<i>Operación</i>	<i>CZPSNH</i>	<i>Comentarios</i>
INC r	r:=r+1	●V●0●	
INC (HL)	(HL):=(HL)+1		
INC (ii+n)	(ii+n):=(ii+n)+1		
INC rr	rr:=rr+1	
INC ii	ii:=ii+1		

• **RDSLt** (000CH)

Lee un byte del espacio de memoria de un determinado slot. Vuelve con las interrupciones inhibidas.

Entrada: A = Número de slot: %X00SSPP, donde
PP = Número de slot primario (0 a 3)
SS = Número de subslot (0 a 3)
F = 1 si el slot está expandido, 0 si no *
HL = Dirección a leer

Salida: A = Byte leído

Registros: AF, BC, DE

* *Nota:* para saber si el slot X está expandido, mirar el bit 7 del byte almacenado en #FCC1+X. Si dicho bit es 1, el slot está expandido.

• **WRSLt** (0014H)

Escribe un byte en el espacio de memoria de un determinado slot. Vuelve con las interrupciones inhibidas.

Entrada: A = Número de slot (como en RDSLt)

Salida: -

Registros: AF, BC, D

• **CALSLt** (001CH)

Realiza una llamada interslot (llama a una rutina almacenada en cualquier slot)

Entrada: IYh = Slot de la rutina (como en RDSLt)
IX = Dirección de la rutina

Otros registros: depende de la rutina a llamar

Salida: depende de la rutina a llamar

Registros: depende de la rutina a llamar

• **DCOMPR** (0020H)

Compara los contenidos de DE y HL

Entrada: HL, DE = Números a comparar

Salida: Z=1 si HL=DE, CY=1 si HL<DE

Registros: AF

• **ENASLt** (0024H)

Conecta un determinado slot en una página determinada. Vuelve con las interrupciones inhibidas.

Entrada: A = Slot a conectar

H = Página a conectar: 0 para la página 0, #40 para la página 1,
#80 para la página 2, #C0 para la página 3

Salida: -

Registros: Todos

7. REFERENCIA DE LA BIOS DEL MSX

A continuación se detallan algunas rutinas de la BIOS de los ordenadores MSX. Sólo se incluyen las relativas a manejo de slots, lectura del teclado y escritura en pantalla en modo texto; el listado completo de la BIOS se encuentra en el apéndice del *MSX2 Technical Handbook*, que puede ser descargado de <http://msx.konamiman.com>.

Para usar las rutinas de la BIOS:

- Si el slot de la BIOS está conectado en la página 0, basta usar la instrucción CALL:

```
CALL rutina
```

- Si es el slot de la RAM el que está conectado en la página 0, hay que hacer una llamada interslot de la siguiente forma:

```
CALSLT: equ #001C
EXPTBL: equ #FCC1 ;Contiene el slot de la BIOS

LD IY,(EXPTBL-1)
LD IX,rutina
CALL CALSLT ;CALSLT continúa disponible aunque se
;conecte la RAM en página 0
```

- Para conectar la BIOS o la RAM en la página 0 hay que usar la rutina selectora de slots de la siguiente forma:

```
ENASLT: equ #0024
EXPTBL: equ #FCC1 ;Contiene el slot de la BIOS
RAMSLOT: equ #F341 ;Contiene el slot de la RAM

LD A,(EXPTBL) ;o bien RAMSLOT
LD H,0
CALL ENASLT ;ENASLT continúa disponible aunque se
;conecte la RAM en página 0
```

Si se conecta cualquier otro slot en la página 0 hay que desactivar las interrupciones, ya que se pierde el gancho de la dirección #0038. Además en ese caso no están disponibles las llamadas ENASLT ni CALSLT, por lo que hay que actuar directamente sobre los puertos selectores de slot.

- Para averiguar qué slot hay conectado en la página 0 en un momento dado hay que usar rutinas más complejas. Por ahora basta saber que los programas ejecutados en entorno MSX-DOS (ficheros .COM) encontrarán RAM en todas las páginas, mientras que los ejecutados en entorno MSX-BASIC (normalmente ficheros .BIN) encontrarán la BIOS en la página 0, el intérprete de BASIC en la página 1 (mismo slot que la BIOS), y RAM en las páginas 2 y 3.

• INput with Increment/Decrement [and Repeat] -

Lectura de un puerto con autoincremento/autodecremento [y repetición]

<u>Nemónico</u>	<u>Operación</u>	<u>CZPSNH</u>	<u>Comentarios</u>
IND	(HL):=port(C); dec HL;dec B	.●??1?	Z=1 si B=0, si no Z=0
INDR	Repite IND hasta B=0	.1??1?	
INI	(HL):=port(C); inc HL;dec B	.●??1?	Z=1 ai B=0, si no Z=0
INIR	Repite INI hasta B=0	.1??1?	

• JumP/Jump Relative - Salto absoluto/relativo

<u>Nemónico</u>	<u>Operación</u>	<u>CZPSNH</u>	<u>Comentarios</u>
JP nn	PC:=nn	
JP cc,nn	Si cc, JP nn		cc - como CALL
JP (HL)	PC:=HL		
JP (ii)	PC:=ii		
JR e	PC:=PC+e		Salto relativo
JR cond,e	Si cond, JR e		cond - C,NC,Z,NZ

• Load with Increment/Decrement [and Repeat] -

Carga con autoincremento/autodecremento [y repetición]

<u>Nemónico</u>	<u>Operación</u>	<u>CZPSNH</u>	<u>Comentarios</u>
LDD	(DE):=(HL); dec DE,HL,BC	..●.00	PV=0 si BC=0, si no PV=1
LDDR	Repite LDD hasta Z=1 o BC=0	..0.00	
LDI	(DE):=(HL); inc DE,HL;dec BC	..●.00	PV=0 if BC=0, si no PV=1
LDIR	Repite LDI hasta Z=1 o BC=0	..0.00	

• **LoaD** - Carga un registro o una posición de memoria

<u>Nemónico</u>	<u>Operación</u>	<u>CZPSNH</u>	<u>Comentarios</u>
LD r,r	r:=r	
LD r,(HL)	r:=(HL)		
LD r,n	r:=n		
LD r,(ii+n)	r:=(ii+n)		
LD (HL),r	(HL):=r		
LD (ii+n),r	(ii+n):=r		
LD (HL),n	(HL):=n		
LD (ii+n),n	(ii+n):=n		
LD A,(BC)	A:=(BC)		
LD A,(DE)	A:=(DE)		
LD A,(nn)	A:=(nn)		
LD (BC),A	(BC):=A		
LD (DE),A	(DE):=A		
LD (nn),A	(nn):=A		
LD A,I	A:=I	.●●●00	PV=IFF
LD A,R	A:=R		PV=IFF
LD I,A	I:=A	
LD R,A	R:=A		
LD rr,nn	rr:=nn		
LD ii,nn	ii:=nn		
LD HL,(nn)	HL:=(nn)		
LD rr,(nn)	rr:=(nn)		
LD ii,(nn)	ii:=(nn)		
LD (nn),HL	(nn):=HL		
LD (nn),rr	(nn):=rr		
LD (nn),ii	(nn):=ii		
LD SP,HL	SP:=HL		
LD SP,ii	SP:=ii		

• **NEGate** - Niega el acumulador (invierte su signo)

<u>Nemónico</u>	<u>Operación</u>	<u>CZPSNH</u>	<u>Comentarios</u>
NEG	A:=0-A	●●v●1●	

• **No OPeration** - No hace nada

<u>Nemónico</u>	<u>Operación</u>	<u>CZPSNH</u>	<u>Comentarios</u>
NOP	No operation	

• **DEFS** o **DS** (*Define Space*). Repite un determinado byte un número determinado de veces. Ejemplo:

```
DS 5,#34
```

equivale a:

```
DB #34,#34,#34,#34,#34
```

Si se omite el byte a repetir, se asume 0 (DS 4 equivale a DB 0,0,0,0).

• **END**. Finaliza el proceso de ensamblado, ignorando el texto situado a continuación.

• **MACROS**. Es posible definir macros con parámetros. La sintaxis varía ligeramente entre ensambladores; aquí se explica la usada por el ensamblador Compass de MSX.

Para definir la macro se ha de comenzar una línea con el nombre de la misma como si fuera una etiqueta, a continuación la directiva "MACRO" y los nombres de los parámetros, cada uno precedido del símbolo arroba "@" y separados por comas. Es decir:

```
NOMBRE: macro @par1,@par2,...
```

Seguidamente se inserta el cuerpo de la macro (usando los parámetros donde sea necesario), y se cierra la definición con la directiva "ENDM".

Para usar la macro se inserta en el código como si fuera una instrucción más, con los parámetros reales separados por comas.

Ejemplo:

;Definición de la macro

```
COPIA: macro @src,@dst,@len
        ld hl,@src
        ld de,@dst
        ld bc,@len
        ldir
        endm
```

;Uso de la macro

```
COPIA #1000,#2000,#300
```

;Código equivalente ensamblado

```
ld hl,#1000
ld de,#2000
ld bc,#300
ldir
```

6. DIRECTIVAS DEL ENSAMBLADOR

• **ORG**. Indica la dirección de memoria a partir de la cual será ensamblado el listado situado a continuación. Ejemplo:

```
ORG #A000
```

• **EQU**. Define una constante, es decir, un nombre al que se asocia un valor concreto de uno o dos bytes. Cada vez que el ensamblador encuentre ese nombre lo sustituirá por el valor correspondiente. Ejemplo:

```
DATO: EQU #34
DIRECCION: EQU #ABCD

LD HL,DIRECCION
LD A,DATO
LD (HL),A
```

es equivalente a:

```
LD HL,#ABCD
LD A,#34
LD (HL),A
```

• **DEFB** o **DB** (*Define Byte*). Esta directiva sirve para ensamblar bytes sueltos que han de ser interpretados como datos, es decir, que no son instrucciones. Es posible definir con un solo DB varios bytes en formato decimal, hexadecimal o ASCII; se han de separar con comas y serán ensamblados consecutivamente. También se puede insertar texto entre comillas: el ensamblador transformará cada carácter en un byte equivalente a su código ASCII. Ejemplos:

```
DB 1,200,#1A,%1010,'Z'
DB "ABC" ;Equivale a DB 65,66,67
DB "Cadena con salto de linea",13,10
DB 34,"Esto esta entre comillas",34
```

• **DEFW** o **DW** (*Define Word*). Actúa como DB, pero con datos dos bytes. CUIDADO: Los datos de dos bytes definidos con DW son almacenados según el formato estándar del Z80, es decir, el byte bajo antes que el alto ("little endian"). Ejemplo:

```
DW #1234,#5678
```

equivale a:

```
DB #34,#12,#78,#56
```

Cuidado: DW #12 no equivale a DB #12 sino a DB #12,#00.

• **OR** - Operación lógica "O"

<u>Nemónico</u>	<u>Operación</u>	<u>CZPSNH</u>	<u>Comentarios</u>
OR r	A:=A or r	0●P●00	
OR (HL)	A:=A or (HL)		
OR n	A:=A or n		
OR (ii+n)	A:=A or (ii+n)		

• **OUTput** - Escritura de datos a un puerto

<u>Nemónico</u>	<u>Operación</u>	<u>CZPSNH</u>	<u>Comentarios</u>
OUT (n),a	port(n):=A	
OUT (C),r	port(C):=r		

• **OUTput with Increment/Decrement [and Repeat]** -

Escritura de un puerto con autoincremento/autodecremento [y repetición]

<u>Nemónico</u>	<u>Operación</u>	<u>CZPSNH</u>	<u>Comentarios</u>
OUTD	port(C):=(HL); dec HL;dec B	.●??1?	Z=1 si B=0, si no Z=0
OTDR	Repite OUTD hasta B=0	.1??1?	
OUTI	port(C):=(HL); inc HL;dec B	.●??1?	Z=1 si B=0, si no Z=0
OTIR	Repite OUTI hasta B=0	.1??1?	

• **POP/PUSH** - Extracción/Inserción de elementos en la pila

<u>Nemónico</u>	<u>Operación</u>	<u>CZPSNH</u>	<u>Comentarios</u>
POP qq	qq:=(SP);SP:=SP+2	qq - AF,BC,DE,HL,
PUSH qq	SP:=SP-2;(SP):=qq		IX,IY

• **RETurn** - Retorno de subrutina

<u>Nemónico</u>	<u>Operación</u>	<u>CZPSNH</u>	<u>Comentarios</u>
RET	POP PC	
RET CC	Si CC, RET		CC - COMO CALL

- **RETurn from Interrupt/NMI** - Retorno de subrutina de atención a la interrupción normal/no enmascarable

<i>Nemónico</i>	<i>Operación</i>	<i>CZPSNH</i>	<i>Comentarios</i>
RETI	Retorno de int.	
RETN	Retorno de NMI		

- **Rotate Left** - Rotación a la izquierda a través del acarreo

<i>Nemónico</i>	<i>Operación</i>	<i>CZPSNH</i>	<i>Comentarios</i>
RL r		••P•00	
RL (HL)			
RL (ii+n)			
RLA		•...00	Como RL A salvo flags

- **Rotate Left with Copy** - Rotación a la izquierda con copia en el acarreo

<i>Nemónico</i>	<i>Operación</i>	<i>CZPSNH</i>	<i>Comentarios</i>
RLC r		••P•00	
RLC (HL)			
RLC (ii+n) ...			
RLCA		•...00	Como RLC A salvo flags

- **Rotate Left Decimal** - Rotación decimal a la izquierda

<i>Nemónico</i>	<i>Operación</i>	<i>CZPSNH</i>	<i>Comentarios</i>
RLD		•P•00	

- **Rotate Right** - Rotación a la derecha a través del acarreo

<i>Nemónico</i>	<i>Operación</i>	<i>CZPSNH</i>	<i>Comentarios</i>
RR r		••P•00	
RR (HL)			
RR (ii+n)			
RRA		•...00	Como RR A salvo flags

- **Conversión de decimal a binario:**

Seguir el siguiente algoritmo:

```

N:=Número a convertir
P:=0
Repetir
    C:=Cociente de N/2
    R:=Resto de N/2
    Bit de la posición P:=R
    N:=C
    P:=P+1
hasta que C=0
  
```

Ejemplo:

```

205 → 205/2: C=102, R=1 → xxxxxx1
      102/2: C=51, R=0 → xxxxxx01
      51/2: C=25, R=1 → xxxxx101
      25/2: C=12, R=1 → xxxx1101
      12/2: C=6, R=0 → xxx01101
      6/2: C=3, R=0 → xx001101
      3/2: C=1, R=1 → x1001101
      1/2: C=0, R=1 → 11001101
  
```

- **Conversión de decimal a hexadecimal:**

Se usa el mismo algoritmo que en la conversión a binario, pero esta vez dividiendo por 16 en vez de 2, y teniendo en cuenta que hay que convertir cada resto (de 0 a 15) a su dígito hexadecimal equivalente (de 0 a F).

Ejemplo:

```

5070 → 5070/16: C=316, R=14 → xxxE
      316/16: C=19, R=12 → xxCE
      19/16: C=1, R=3 → x3CE
      1/16: C=0, R=1 → 13CE
  
```

Binario	Hexadecimal	Decimal
0001 0000	10	16
0010 0000	20	32
0100 0000	40	64
1000 0000	80	128
1111 1111	FF	255
	100	256
	200	512
	400	1024
	800	2048
	FFF	4095
	1000	4096
	4000	16384
	8000	32768
	C000	49152
	FFFF	65535

• **Conversión de binario a decimal:**

Observar la posición de los unos (de derecha a izquierda, la primera posición es la cero).
 Por cada uno presente sumar 2 elevado a su posición.

Ejemplo:

$$100110 \rightarrow 0 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + 0 \cdot 2^3 + 0 \cdot 2^4 + 1 \cdot 2^5 \rightarrow$$

$$0 \cdot 1 + 1 \cdot 2 + 1 \cdot 4 + 0 \cdot 8 + 0 \cdot 16 + 1 \cdot 32 \rightarrow$$

$$2 + 4 + 32 = 38$$

Para números mayores: $2^6=64$, $2^7=128$, $2^8=256$, $2^9=512$, $2^{10}=1024$

• **Rotate Right with Copy** - Rotación a la derecha con copia en el acarreo

Nemónico	Operación	CZPSNH	Comentarios
RRC r		••P•00	
RRC (HL)			
RRC (ii+n) ...			
RRCA		•...00	Como RRC A salvo flags

• **Rotate Right Decimal** - Rotación decimal a la derecha

Nemónico	Operación	CZPSNH	Comentarios
RRD		••P•00	

• **ReStArt** - Reinicio (llamada corta a subrutina)

Nemónico	Operación	CZPSNH	Comentarios
RST adr	CALL adr	adr-00,08,10,18,20,28,30,38 hex.

• **SubStract with Carry** - Resta con acarreo

Nemónico	Operación	CZPSNH	Comentarios
SBC A,r	A:=A-r-CY	••V•1•	
SBC A,(HL) ...	A:=A-(HL)-CY		
SBC A,n	A:=A-n-CY		
SBC A,(ii+n) .	A:=A-(ii+n)-CY		
SBC HL,rr	HL:=HL-rr-CY	••V•1x	

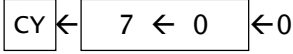
• **Set Carry Flag** - Pone el acarreo a 1

Nemónico	Operación	CZPSNH	Comentarios
SCF	CY:=1	1...00	

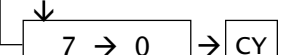
• **SET bit** - Pone a 1 un bit de un registro o posición de memoria

Nemónico	Operación	CZPSNH	Comentarios
SET b,r	rb:=1	
SET b,(HL) ...	(HL)b:=1		
SET b,(ii+n) .	(ii+n)b:=1		

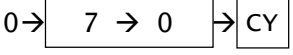
• **Shift Left Arithmetic** - Desplazamiento aritmético a la izquierda

<u>Nemónico</u>	<u>Operación</u>	<u>CZPSNH</u>	<u>Comentarios</u>
SLA r		●●P●00	Usar para multiplicar por 2
SLA (HL)			
SLA (ii+n) ...			

• **Shift Right Arithmetic** - Desplazamiento aritmético a la derecha

<u>Nemónico</u>	<u>Operación</u>	<u>CZPSNH</u>	<u>Comentarios</u>
SRA r		●●P●00	bit 7 se mantiene Usar para dividir por 2
SRA (HL)			
SRA (ii+n) ...			

• **Shift Right Logic** - Desplazamiento lógico a la derecha

<u>Nemónico</u>	<u>Operación</u>	<u>CZPSNH</u>	<u>Comentarios</u>
SRL r		●●P●00	r7:=r6 Usar para dividir por 2
SRL (HL)			
SRL (ii+n) ...			

• **SUBstract** - Resta

<u>Nemónico</u>	<u>Operación</u>	<u>CZPSNH</u>	<u>Comentarios</u>
SUB r	A:=A-r	●●V●0●	
SUB (HL)	A:=A-(HL)		
SUB n	A:=A-n		
SUB (ii+n) ...	A:=A-(ii+n)		

• **XOR** - Operación lógica "O exclusiva"

<u>Nemónico</u>	<u>Operación</u>	<u>CZPSNH</u>	<u>Comentarios</u>
XOR r	A:=A xor r	0●P●00	
XOR (HL)	A:=A xor (HL)		
XOR n	A:=A xor n		
XOR (ii+n) ...	A:=A xor (ii+n)		

5. CONVERSIONES BINARIO/HEXADECIMAL/DECIMAL

<i>Binario</i>	<i>Hexadecimal</i>	<i>Decimal</i>
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

• **Conversión entre binario y hexadecimal para números de más de 4 bits:**

Se puede convertir directamente cada grupo de cuatro bits en su dígito hexadecimal equivalente y viceversa. Para binario a hexadecimal, añadir antes ceros a la izquierda hasta que el número de bits sea múltiplo de cuatro.

Ejemplos:

1010100110 → 0010.1010.0110 → 2A6

3DB → 0011.1011.1011 → 1110111011