1. Instrucciones de Transferencia de Datos

MOV dest, src ;[1]Copia el contenido del operando fuente (src) en el destino (dest).
Operación: dest **<-** src
Las posibilidades son:

1.MOV reg, {reg/mem/inmed}
2.MOV mem, {reg/inmed}
3.MOV {reg16/mem16}, {CS/DS/ES/SS}
4.MOV {DS/ES/SS}, {reg16/mem16}

PUSH src ; Pone el valor en el tope del stack.
Operación: SP **<-** SP -2, [SP+1:SP] **<-** src donde src = {reg16/mem16/CS/DS/ES/SS} (el puntero SP debe decrementarse en 2 valores, pero el dato queda almacenado en 1 valor más respecto a ese valor del SP, dentro del segmento de pila)

POP dest ; Retira el valor del tope del stack poniéndolo en el lugar indicado.
Operación: dest <- [SP+1:SP], SP <- SP + 2 donde dest = {reg16/mem16/DS/ES/SS}. (aplica la misma consideración anterior respecto al dato y su posición respecto al SP)

XCHG reg, {reg/mem} ; Intercambia ambos valores.

IN {AL/AX}, {DX/inmed (1 byte)} ; Pone en el acumulador el valor hallado en el puerto (port) indicado.

OUT {DX/inmed (1 byte)}, {AL/AX} ; Pone en el puerto (port) indicado el valor contenido en el acumulador.

XLAT ; Realiza una operación de traducción de un código de un byte a otro código de un byte mediante una tabla.
Operación: AL **<-** [BX+AL]

LEA reg, mem ; Almacena la dirección efectiva del operando de memoria en un registro.
Operación: reg **<-** dirección mem

LDS reg, mem32 ; Operación: reg **<-** [mem], DS **<-** [mem+2]
LES reg, mem32 ; Operación: reg **<-** [mem], ES **<-** [mem+2]

LAHF ; Copia en el registro AH la imagen de los ocho bits menos significativos del registro de indicadores.
Operación: AH **<-** SF:ZF:X:AF:X:PF:X:CF

SAHF ; Almacena en los ocho bits menos significativos del registro de indicadores el valor del registro AH.
Operación: SF:ZF:X:AF:X:PF:X:CF **<-** AH

[1] En todo este documento se usa **;** para denotar un comentario, tal y como debe hacerse en un programa real. En este caso para comentar el comando y su función.

PUSHF ; Almacena los flags en la pila.
Operación: SP <- SP **–** 2, [SP+1 :SP] **<-** Flags.

POPF ; Pone en los flags el valor que hay en la pila.
Operación: Flags <- [SP+1:SP], SP <- SP + 2

2. Instrucciones Aritméticas (Afecta las banderas AF, CF, OF, PF, SF, ZF)

ADD dest,src ; Operación: dest <- dest + src.
ADC dest,src ; Operación: dest <- dest + src + CF.
SUB dest,src ; Operación: dest <- dest - src.
SBB dest,src ; Operación: dest <- dest - src - CF.
CMP dest,src ; Operación: dest - src (sólo afecta flags).
INC dest ; Operación: dest <- dest + 1 (no afecta CF).
DEC dest ; Operación: dest <- dest - 1 (no afecta CF).
NEG dest ; Operación: dest <- - dest; donde dest = {reg/mem} y src = {reg/mem/inmed} no pudiendo ambos operandos estar en memoria.

DAA ; Corrige el resultado de una suma de dos valores BCD empaquetados en el registro AL (debe estar inmediatamente después de una instrucción ADD o ADC). OF es indefinido después de la operación.
DAS ; Igual que DAA pero para resta (debe estar inmediatamente después de una instrucción SUB o SBB).

AAA ; Lo mismo que DAA para números BCD desempaquetados.
AAS ; Lo mismo que DAS para números BCD desempaquetados.
AAD ; Convierte AH:AL en BCD desempaquetado a AL en binario.
Operación: AL <- AH * 0Ah **+** AL, AH **<-** 0. Afecta PF, SF, ZF, mientras que AF, CF y OF quedan indefinidos.

AAM ; Convierte AL en binario a AH:AL en BCD desempaquetado.
Operación: AH <- AL / 0Ah. AL <- AL mod 0Ah. Afecta PF, SE, ZF, mientras que AF, CF y OF quedan indefinidos.

MUL {reg8/rnem8} ; Realiza una multiplicación con operandos no signados de 8 por 8 bits.
Operación: AX <- AL * {reg8/mem8}. CF=OF=0 si AH =0, CF=OF=1 en caso contrario. AF, PF, SF, ZF quedan indefinidos.
MUL {reg16/mem16} ; Realiza una multiplicación con operandos no signados de 16 por 16 bits Operación: DX:AX <- AX *{reg16/mem16}. CF=OF=0 si DX=0, CF=OF=1 en caso contrario. AF, PF, SF, ZF quedan indefinidos.

IMUL {reg8/mem8} ; Realiza una multiplicación con operandos con signo de 8 por 8 bits.
Operación: AX <- AL * {reg8/mem8} realizando la multiplicación con signo. CF= OF=0 si el resultado entra en un byte, en caso contrario valdrán 1. AF, PF, SF, ZF quedan indefinidos.
IMUL {reg16/mem16} ; Realiza una multiplicación con operandos con signo de 16 por 16 bits. Operación: DX:AX <- AX * {reg16/mem16} realizando la multiplicación con signo. CF=OF = 0 si el resultado entra en dos bytes, en caso contrario valdrán 1. AF, PF, SF, ZF quedan indefinidos.

```
CBW         ; Extiende el signo de AL en AX. No se afectan los flags.
CWD         ; Extiende el signo de AX en DX:AX. No se afectan flags.
```

3. Instrucciones Lógicas (Afectan las banderas AF, CF, OF, PF, SE, ZF)

```
AND dest, src           ; Operación: dest <- dest and src.
TEST dest ,src          ; Operación: dest and src     Sólo afecta flags.
OR dest, src            ; Operación: dest <- dest or src.
XOR dest, src           ; Operación: dest <- dest xor src.
```

Las cuatro instrucciones anteriores ponen CF = OF = 0, AF queda indefinido y PF, SF y ZF dependen del resultado.

```
NOT dest            ; Operación: dest <- Complemento a 1 de dest. No afecta flags.
SHL/SAL dest, {1/CL}  ; Realiza un desplazamiento lógico o aritmético a la izquierda.
SHR dest, {1/CL}      ; Realiza un desplazamiento lógico a la derecha.
SAR dest, {1/CL}      ; Realiza un desplazamiento aritmético a la derecha.
ROL dest, {1/CL}      ; Realiza una rotación hacia la izquierda.
ROR dest, {1/CL}      ; Realiza una rotación hacia la derecha.
RCL dest, {1/CL}      ; Realiza una rotación hacia la izquierda usando el CF.
RCR dest, {1/CL}      ; Realiza una rotación hacia la derecha usando el CF.
```

En las siete instrucciones anteriores la cantidad de veces que se rota o desplaza puede ser un bit o la cantidad de bits indicado en CL.

4. Instrucciones de Manipulación de Cadenas

```
MOVSB               ; Copiar un byte de la cadena fuente al destino.
Operación:
1. ES:[Dl] <- DS:[SI] (un byte)
2. DI <- DI± 1
3. SI <-SI ±1

MOVSW               ; Copiar dos bytes de la cadena fuente al destino.
Operación:
1. ES:[DI] <- DS:[Slj (dos bytes)
2. DI <- DI±2
3. SI <- SI±2

LODSB               ; Poner en el acumulador un byte de la cadena fuente.
Operación:
1. AL <- DS:[SI] (un byte)
2. SI <- SI±1

LODSW               ; Poner en el acumulador dos bytes de la cadena fuente.
Operación:
1. AX <- DS:[SI] (dos bytes)
```

STOSB ; Almacenar en la cadena destino un byte del acumulador.
Operación:
1. ES:[DI] <- AL (un byte)
2. DI <- DI±1

STOSW ; Almacenar en la cadena destino dos bytes del acumulador.
Operación:
1. ES:[DI] <- AX (dos bytes)
2. DI <- DI±2

CMPSB ; Comparar un byte de la cadena fuente con el destino.
Operación:
1. DS:[SI] - ES:[DI] (Un byte, afecta sólo los flags)
2. DI <- DI±1
3. SI <- SI±1

CMPSW ; Comparar dos bytes de la cadena fuente con el destino.
Operación:
1. DS:[SI] - ES:[DI] (Dos bytes, afecta sólo los flags)
2. DI <- DI±2
3. SI <- SI±2

SCASB ; Comparar un byte del acumulador con la cadena destino.
Operación:
1. AL - ES:[DI] (Un byte, afecta sólo los flags)
2. DI <- DI±1

SCASW ; Comparar dos bytes del acumulador con la cadena destino.
Operación:
1. AX - ES:[DI] (Dos bytes, afecta sólo los flags)
2.DI <- DI±2

En todos los casos el signo + se toma si el indicador DF vale cero. Si vale 1 hay que tomar el signo (-).
Prefijo para las instrucciones MOVSB, MOVSW, LODSB, LODSW, STOSB y STOSW:
REP: Repetir la instrucción CX veces.
Prefijos para las instrucciones CMPSB, CMPSW, SCASB, SCASW:
REPZ/REPE: Repetir mientras que sean iguales hasta un máximo de CX veces.
REPNZ/REPNE: Repetir mientras que sean diferentes hasta un máximo de CX veces.

5. Instrucciones de Transferencia de Control (No afectan flags)

CALL label ; Ir al procedimiento cuyo inicio es label.
RET ; Retorno de procedimiento.
RET inmed ; Retorno de procedimiento y SP <- SP + inmed.

Variaciones de la instrucción de retorno:

RETN [inmed] ; En el mismo segmento de código.
RETF [inmed] ; En otro segmento de código.

6. Instrucciones de Transferencia de Saltos

Saltos condicionales aritméticos (usar después de CMP):
Aritmética signada (con números positivos, negativos y cero)

```
JL etiqueta/JNGE etiqueta    ; Saltar a etiqueta si es menor.
JLE etiqueta/JNG etiqueta    ; Saltar a etiqueta si es menor o igual.
JE etiqueta                  ; Saltar a etiqueta si es igual.
JNE etiqueta                 ; Saltar a etiqueta si es distinto.
JGE etiqueta/JNL etiqueta    ; Saltar a etiqueta si es mayor o igual.
JG etiqueta/JNLE etiqueta    ; Saltar a etiqueta si es mayor.
```

Aritmética sin signo (con números positivos y cero)

```
JB etiqueta/JNAE etiqueta    ; Saltar a etiqueta si es menor.
JBE etiqueta/JNA etiqueta    ; Saltar a etiqueta si es menor o igual.
JE etiqueta                  ; Saltar a etiqueta si es igual.
JNE etiqueta                 ; Saltar a etiqueta si es distinto
JAE etiqueta/JNB etiqueta    ; Saltar a etiqueta si es mayor o igual.
JA etiqueta/JNBE etiqueta    ; Saltar a etiqueta si es mayor.
```

Saltos condicionales según el valor de los indicadores:

```
JC label                     ; Saltar si hubo arrastre/préstamo (CF = 1).
JNC label                    ; Saltar si no hubo arrastre/préstamo (CF = 0).
JZ label                     ; Saltar si el resultado es cero (ZF = 1).
JNZ label                    ; Saltar si el resultado no es cero (ZF =0).
JS label                     ; Saltar si el signo es negativo (SF = 1).
JNS label                    ; Saltar si el signo es positivo (SF = 0).
JP/JPE label                 ; Saltar si la paridad es par (PF = 1).
JNP/JPO label                ; Saltar si la paridad es impar (PF = 0).
```

Saltos condicionales que usan el **registro CX como contador:**

```
LOOP label                   ; Operación: CX <- CX-1. Saltar a label si CX<>0.
LOOPZ/LOOPE label            ; Operación: CX <- CX-1. Saltar a label si CX <> 0 y ZF=1.
LOOPNZ/LOOPNE label          ; Operación: CX <- CX-1. Saltar a label si CX <> 0 y ZF=0.
JCXZ label                   ; Operación salta a label si CX = 0.
```

7. Interrupciones

```
INT número                   ; Salva los flags en la pila, hace TF=IF=0 y ejecuta la
interrupción con el número indicado.
INTO                         ; Interrupción condicional. Si OF = 1, hace INT 4.
IRET                         ; Retorno de interrupción. Restaura los indicadores del stack.
```

8. Instrucciones de control del procesador

```
CLC                  ; CF <- 0.
STO                  ; CF <-1.
CMC                  ; CF <- 1 - CF.
```

```
NOP             ; No hace nada.
CLD             ; DF <- 0 (Dirección ascendente).
STD             ; DF <- 1 (Dirección descendente).
CLI             ; lE <- 0 (Deshabilita interrupciones enmascarables).
STI             ; lE <- 1 (Habilita interrupciones enmascarables).
HLT             ; Detiene la ejecución del procesador hasta que llegue una interrupción
externa.
WAIT            ; Detiene la ejecución del procesador hasta que se active el pin TEST
del mismo.
LOCK            ; Prefijo de instrucción que activa el pin LOCK del procesador.
```

9. Operadores

Operadores aritméticos: +, - , * , / MOD (resto de la división).
Operadores lógicos: AND, OR, XOR, NOT, SHR, SHL.
Para los dos últimos operadores, el operando derecho indica la cantidad de bits a desplazar hacia la derecha (para SHR) o izquierda (para SHL) el operando izquierdo.
Operadores relacionales: Valen cero si son falsos y 65535 si son verdaderos.
EQ: Igual a.
NE: Distinto de.
LT: Menor que.
GT: Mayor que.
LE: Menor o igual a.
GE: Mayor o igual a.

Operadores analíticos: Descomponen operandos que representan direcciones de memoria en sus componentes.
SEG memory-operand: Retorna el valor del segmento.
OFFSET memory-operand: Retorna el valor del offset.
TYPE memory-operand: Retorna un valor que representa el tipo de operando: BYTE = 1, WORD = 2, DWORD =4 (para direcciones de datos) y NEAR= -1 y FAR= -2 (para direcciones de instrucciones).

LENGHT memory-operand: Se aplica solamente a direcciones de datos. Retorna un valor numérico para el número de unidades (bytes, words o dwords) asociados con el operando. Si el operando es una cadena retorna el valor 1.
Ejemplo: Dada la directiva PALABRAS DW 50 DUP (0), el valor de LENGHT PALABRAS es 50, mientras que dada la directiva CADENA DB “cadena” el valor de LENGHT CADENA es 1.
SIZE memory-operand: LENGHT memory-operand ***** TYPE memory-operand.

Operadores sintéticos: Componen operandos de direcciones de memoria a partir de sus componentes.
type PTR memory-operand: Compone un operando de memoria que tiene el mismo segmento y offset que el especificado en el operando derecho pero con el tipo (BYTE, WORD, DWORD, NEAR o FAR) especificado en el operando izquierdo.
THIS type: Compone un operando de memoria con el tipo especificado que tiene el segmento y offset que la próxima ubicación a ensamblar.

Operadores de macros: Son operadores que se utilizan en las definiciones de macros. Hay cinco: &,<>, !, % y ;

&parámetro ; reemplaza el parámetro con el valor actual del argumento.

<texto> ; trata una serie de caracteres como una sola cadena. Se utiliza cuando el texto incluye comas, espacios u otros símbolos especiales.

!carácter ; trata el carácter que sigue al operador! como un carácter en vez de un símbolo o separador.

%texto ; trata el texto que sigue a continuación del operador % como una expresión. El ensamblador calcula el valor de la expresión y reemplaza el texto por dicho valor.

sentencia ; comentario ; Permite definir comentarios que aparecerán en la definición de la macro pero no cada vez que éste se invoque en el listado fuente que genera el ensamblador.

10. Directivas

Definición de símbolos

EQU ; Define nombres simbólicos que representan valores u otros valores simbólicos. Las dos formas son:
nombre EQU expresión
nuevo_nombre EQU viejo_nombre

Una vez definido un nombre mediante EQU, no se puede volver a definir.
= ; Es similar a EQU pero permite que el símbolo se pueda redefinir. Sólo admite la forma:
nombre = expresión.

Definición de datos

Ubica memoria para un ítem de datos y opcionalmente asocia un nombre simbólico con esa dirección de memoria y/o genera e valor inicial para ese ítem.

[nombre] DB valor_inicial [, valor_inicial...]
donde valor_inicial puede ser una cadena o una expresión numérica cuyo resultado esté entre -255 y 255.

[nombre] DW valor_inicial [, valor_iniçial...]
donde valor_inicial puede ser una expresión numérica cuyo resultado esté entre -65535 y 65535 o un operando de memoria en cuyo caso se almacenará el offset del mismo.

[nombre] DD valor_inicial [, valor_inicial...]
donde valor_inicial puede ser una constante cuyo valor esté entre -4294967295 y 4294967295, una expresión numérica cuyo valor absoluto no supere 65535, o bien un operando de memoria en cuyo caso se almacenarán el offset y el segmento del mismo (en ese orden). Si se desea que no haya valor inicial, deberá utilizarse el símbolo ?. Otra forma de expresar el valor inicial es:

cuenta DUP (valor_inicial [,valor_inicial...])
donde cuenta es la cantidad de veces que debe repetirse lo que está entre paréntesis.

Definición de segmentos
Organizan el programa para utilizar los segmentos de memoria del microprocesador 8088. Estos son SEGMENT, ENDS, DOSSEG, ASSUME, GROUP.

nombre_segm
SEGMENT [alineación][combinación]['clase']
sentencias
nombre_segm
ENDS

Alineación ; define el rango de direcciones de memoria para el cual puede elegirse el inicio del segmento. Hay cinco posibles:
1. BYTE : El segmento comienza en el siguiente byte.
2.WORD : El segmento comienza en la siguiente dirección par.
3.DWORD : Comienza en la siguiente dirección múltiplo de 4.
4. PARA : Comienza en la siguiente dirección múltiplo de 16.
5. PAGE : Comienza en la siguiente dirección múltiplo de 256.
Si no se indica la alineación ésta será PARA.

Combinación: define cómo combinar segmentos que tengan el mismo nombre. Hay cinco posibles:
1. PUBLIC: Concatena todos los segmentos que tienen el mismo nombre para formar un sólo segmento. Todas las direcciones de datos e instrucciones se representan la distancia entre el inicio del segmento y la dirección correspondiente. La longitud del segmento formado será la suma de las longitudes de los segmentos con el mismo nombre.
2. STACK: Es similar a PUBLIC. La diferencia consiste que, al comenzar la ejecución del programa, el registro SS apuntará a este segmento y SP se inicializará con la longitud en bytes de este segmento. 3 COMMON: Pone el inicio de todos los segmentos teniendo el mismo nombre en la misma dirección de memoria. La longitud del segmento será la del segmento más largo.
4. MEMORY: Es igual a PUBLIC.
5. AT dirección_de_segmento: Hace que todas las etiquetas y direcciones de variables tengan el segmento especificado por la expresión contenida en dirección_de_segmento. Este segmento no puede contener código o datos con valores iniciales. Todos los símbolos que forman la expresión dirección_de_segmento deben conocerse en el primer paso de ensamblado.
Si no se indica combinación, el segmento no se combinará con otros del mismo nombre (combinación “privada”).

Clase: Es una forma de asociar segmentos con diferentes nombres, pero con propósitos similares. Sirve también para identificar el segmento de código. Debe estar encerrado entre comillas simples. El linker pone los segmentos que tengan la misma clase uno a continuación de otro, si bien siguen siendo segmentos diferentes. Además supone que los segmentos de código tienen clase CODE o un nombre con el sufijo CODE.

DOSSEG: Esta directiva especifica que los segmentos deben ordenarse según la convención de DOS. Esta es la convención usada por los compiladores de lenguajes de alto nivel.

GROUP: Sirve para definir grupos de segmentos. Un grupo es una colección de segmentos asociados con la misma dirección inicial. De esta manera, aunque los datos estén en diferentes segmentos, todos pueden accederse mediante el mismo registro de segmento. Los segmentos de un grupo no necesitan ser contiguos.
Sintaxis: nombre_grupo GROUP segmento [, segmento...]

ASSUME: Sirve para indicar al ensamblador qué registro de segmento corresponde con un segmento determinado. Cuando el ensamblador necesita referenciar una dirección debe saber en qué registro de segmento lo apunta.

Sintaxis: ASSUME reg_segm:nombre [, reg_segm:nombre...]
donde el nombre puede ser de segmento o de grupo, una expresión utilizando el operador SEG o la palabra NOTHING, que cancela la selección de registro de segmento hecha con un ASSUME anterior.

Control del ensamblador
ORG expresión: El offset del código o datos a continuación será la indicada por la expresión. Todos los símbolos que forman la expresión deben conocerse en el primer paso de ensamblado.

EVEN: Hace que la próxima instrucción o dato se ensamble en la siguiente posición par.
END [etiqueta]: Debe ser la última sentencia del código fuente. La etiqueta indica dónde debe comenzar la ejecución del programa. Si el programa se compone de varios módulos, sólo el módulo que contiene la dirección de arranque del programa debe contener la directiva END etiqueta. Los demás módulos deberán terminar con la directiva END (sin etiqueta).

Definición de procedimientos
Los procedimientos son secciones de código que se pueden llamar para su ejecución desde distintas partes del programa.

etiqueta PROC {NEAR/FAR}
sentencias
etiqueta
ENDP

Macros: La macros asignan un nombre simbólico a uñ bloque de sentencias fuente. Luego se puede usar dicho nombre para representar esas sentencias. Opcionalmente se pueden definir parámetros para representar argumentos para la macro.

Definición de macros

nombre_macro
MACRO
[parámetro [,parámetro...]]
[LOCAL
nombre_local[,nombreJocal...]
sentencias
ENDM

Los parámetros son opcionales. Si existen, entonces también aparecerán en algunas de las sentencias en la definición de la macro. Al invocar la macro mediante:
nombre_macro [argumento [,argumento..]] se ensamblarán las sentencias indicadas en la macro teniendo en cuenta que cada lugar donde aparezca un parámetro se reemplazará por el argumento correspondiente.
El nombre_local de la directiva LOCAL es un nombre simbólico temporario que será reemplazado por un único nombre simbólico (de la forma ??número) cuando la macro se invoque.
Todas las etiquetas dentro de la macro deberán estar indicadas en la directiva LOCAL para que el ensamblador no genere un error indicando que un símbolo está definido varias veces.
La directiva EXITM (usada dentro de la definición de la macro) sirve para que no se ensamblen más sentencias de a macro (se usa dentro de bloques condicionales).
PURGE nombre_macro [,nombre_macro]: Borra las macros indicadas de la memoria para poder utilizar ese espacio para otros símbolos.

Definición de bloques
Son tres: REPT, IRP e RPC. Como en el caso de la directiva MACRO, se puede incluir las sentencias LOCAL y EXITM y deben terminarse con la directiva ENDM.
REPT expresión
sentencias
ENDM
La expresión debe poder ser evaluada en el primer paso del ensamblado y el resultado deberá estar entre O y 65535. Esta expresión indica la cantidad de veces que debe repetirse el bloque.
IRP parámetro, <argumento [,argumento..]
sentencias
ENDM

El parámetro se reemplaza por el primer argumento y se ensamblan las sentencias dentro del bloque. Luego el parámetro se reemplaza por el segundo argumento y se ensamblan las sentencias y así sucesivamente hasta agotar los argumentos.

IRPC parámetro, cadena
sentencias
EN DM

Es similar a IRP con la diferencia que el parámetro se reemplaza por cada carácter de la cadena. Si ésta contiene comas, espacios u otros caracteres especiales deberá encerrarse

con paréntesis angulares (<).

Procesador: Indican el tipo de procesador y coprocesador en el que se va a ejecutar el programa. Los de procesador son: .8086, .186, .286, .386, .486 y .586 para instrucciones en modo real, .286P, .386P, .486P y .586P para instrucciones privilegiadas, .8087, .287 y .387 para coprocesadores. Deben ubicarse al principio del código fuente. Habilitan las instrucciones correspondientes al procesador y coprocesador indicado. Sin estas directivas, sólo se pueden ensamblar instrucciones del 8086 y 8087.

Transferencias externas al módulo

Sirve para poder particionar un programa en varios archivos fuentes o módulos. Son imprescindibles si se hace un programa en alto nivel con procedimientos en assembler. Hay tres: PUBLIC, EXTRN e INCLUDE.

PUBLIC nombre[. nombre...]: Estos nombres simbólicos se escriben en el archivo objeto. Durante una sesión con el linker, los símbolos en diferentes módulos pero con los mismos nombres tendrán la misma dirección.

EXTRN nombre:tipo [,nombre:tipo...]: Define una variable externa con el nombre y tipo (NEAR, FAR, BYTE, WORD, DWORD o ABS (número constante especificado con la directiva EQU o =)) especificado. El tipo debe ser el mismo que el del ítem indicado con la directiva PUBLIC en otro módulo.

INCLUDE nombre_de_archivo: Ensambla las sentencias indicadas en dicho archivo. Permite definir los segmentos sin necesidad de utilizar las directivas de segmentos que aparecen más arriba.

Segmentos simplificados

.MODEL modelo: Debe estar ubicada antes de otra directiva de segmento. El modelo puede ser uno de los siguientes:

1. TINY: Los datos y el código juntos ócupan menos de 64 KB por lo que entran en el mismo segmento. Se utiliza para programas .COM. Algdnos ensambladores no soportan este modelo.
2. SMALL: Los datos caben en un segniento de 64 KB y el código cabe en otro segmento de 64 KB. Por lo tanto todo el código y los datos se pueden acceder como NEAR.
3. MEDIUM: Los datos entran en un sólo segmento de 64 KB, pero el código puede ser mayor de 64 KB. Por lo tanto, código es FAR, mientras que los datos se acceden como NEAR.
4. COMPACT: Todo el código entra en un segmento de 64 KB, pero los datos no (pero no pueden haber matrices de más de 64 KB). Por o tanto, código es NEAR, mientras que los datos se acceden como
FAR.
5. LARGE: Tanto el código como los datos pueden ocupar más de 64 KB (pero no pueden haber matrices de más de 64 KB), por lo que ambos se acceden como FAR.
6. HUGE: Tanto el código como los datos pueden ocupar más de 64 KB (y las matrices también), por lo que ambos se acceden como FAR y los punteros a los elementos de las matrices también son FAR.

.STACK [size] Define el segmento de pila de la longitud especificada.

.CODE [name] Define el segmento de código.

DATA : Define un segmento de datos NEAR con valores iniciales.
DATA? : Define un segmento de datos NEAR sin valores iniciales.
FARDATA [name] : Define un segmento de datos FAR con valores iniciales.
FARDATA? [name] : Define un segmento de datos FAR sin valores iniciales.
CONST : Define un segmento de datos constantes.

Los siguientes símbolos están definidos cuando se usan las directivas anteriores:

@curseg : Tiene el nombre del segmento que se está ensamblando.
@filename : Representa el nombre del archivo fuente (sin la extensión)
@codesize : Vale 0 para los modelos SMALL y COMPACT (código NEAR), y vale 1 para los modelos MEDIUM, LARGE y HUGE (código FAR).
@datasize : Vale 0 para los modelos SMALL y MEDIUM (datos NEAR), vale 1 para los modelos COMPACT y LARGE (datos FAR) y vale 2 para el modelo HUGE (punteros a matrices FAR).
@code : Nombre del segmento definido con la directiva .CODE.
@data : Nombre del segmento definido con las directicas .DATA, .DATA?, CONST y .STACK (los cuatro están en el mismo segmento).
@fardata : Nombre del segmento definido con la directiva .FARDATA.
@fardata? : Nombre del segmento definido con la directiva .FARDATA?

RESUMEN DEL CONJUNTO DE INSTRUCCIONES DE LOS PROCESADORES DE LA FAMILIA INTEL 80x86

INSTRUCCIONES DE TRANSFERENCIA DE DATOS

IN	Lectura de datos de puertos de entrada
OUT	Escritura de datos a puertos de salida
MOV	Copia de datos de un origen a un destino
LAHF	Copia el contenido del registro de banderas al registro AH
SAHF	Copia el contenido del registro AH registro de banderas
PUSH	Inserción de datos en la pila
POP	Copia datos de la pila al operando especificado y remueve los datos de la pila
PUSHF	Copia el contenido del registro de banderas en la pila
POPF	Restaura el contenido del registro de banderas con valores de la pila
XCHG	Intercambia el valor de dos operandos
LEA	Carga una dirección efectiva de memoria en el operando indicado
LDS	Carga el DS en el operando indicado
LES	Carga el ES en el operando indicado

INSTRUCCIONES PARA MANIPULACION DE HILERAS DE CARACTERES (MODIFICAN EL DESTINO CON BASE EN EL TAMAÑO Y EL BIT DE DIRECCION DEL REGISTRO DE BANDERAS)

MOVSB	Mueve bytes de una cadena de caracteres
MOVSW	Mueve dos bytes (word) de cadena de caracteres
CMPSB	Compara bytes de cadena de caracteres
CMPSW	Compara dos bytes (word) de cadena de caracteres
SCASB	Busca un byte en una cadena de caracteres
SCASW	Busca dos bytes (word) en una cadena de caracteres
LODSB	Carga un byte de una cadena de caracteres en el registro AL
LODSW	Carga dos bytes (Word) de una cadena de caracteres en el registro Ax
STOSB	Almacena un byte del contenido del registro AL en una cadena de caracteres
STOSW	Almacena dos bytes (word) del contenido del registro AX en una cadena de caracteres

INSTRUCCIONES ARITMÉTICAS

IDIV	División sin signo
ADD	Suma
AAM	Ajuste ASCII luego de multiplicación
ADC	Suma con acarreo (carry)
AAD	Ajuste ASCII antes de división
SBW	Extender signo de un byte a word
CWD	Extender signo de word a double
INC	Incremento
SUB	Resta
SBB	Resta con acarreo (carry)
AAS	Byte ASCII luego de resta
DAS	Ajuste ASCII luego de BCD
DEC	Decremento
NEG	Negación
CMP	Comparación
MUL	Multiplicación con signo
IMUL	Multiplicación sin signo
DIV	División con signo

MANIPULACIÓN DE BITS

AND	AND lóqico bit por bit
OR	OR lógico bit por bit
XOR	OR exclusive lógico bit por bit
NOT	Inversión de bits
TEST	Verificación de bits, AND pero no afecta operandos
SHL	Corrimiento de bits sin signo a la izquierda
SAL	Corrimiento de bits con signo a la izquierda
SHR	Corrimiento de bits sin signo a la derecha
SAR	Corrimiento de bits con signo a la derecha
ROL	Rotación de bits a la izquierda
ROR	Rotación de bits a la izquierda
RCL	Rotación de bits a la izquierda incluyendo el bit de carry
RCR	Rotación de bits a la derecha incluyendo el bit de carry

CONTROL DE PROGRAMAS

CALL	Llamada a subrutina o procedimientos (variantes NEAR y FAR)
RET	Retorno de una subrutina o procedimiento
1 NT	Interrupción de software
INTO	Interrupción en desbordamiento
IRET	Retorno de una interrupción
LOOP	Ciclos de porciones de código según valor CX > 0
LOOPE or LOOPZ	Ciclos de porciones de código según valor CX > 0 y bandera cero activada (ZF=1)
LOOPNE or LOOPNZ	Ciclos de porciones de código según valor CX> 0 y bandera cero desactivada (ZF=0)
JMP	Salto incondicional a una posición identificada con una etiqueta
JCXZ	Salto condicional si CX igual a cero
JG or JNLE	Salto condicional si resultado es mayor, o no menor o igual
JGE or JNL	Salto condicional si resultado es mayor o igual, o no menor que
JE or JZ	Salto condicional si resultado es igual
JLE or JNG	Salto condicional si resultado es menor o igual que, o no mayor que
JL or JNGE	salto condicional si resultado es menor que, o no mayor o igual que
JA or JNBE	Salto condicional si resultado es si!Iperior o no inferior o igual que
JAE or JNB	Salto condicional si resultado es superior o igual, o no inferior
JBE or JNA	Salto condicional si resultado es iñferior o igual, o no superior
JB or JNAE	Salto condicional si resultado es inferior, o no superior o igual
JC	Salto condicional si carry está activado (CF=1)
JNC	Salto condicional si carry está desactivado (CF=0)
JO	Salto condicional si hay overflow (OF=1)
JNO	Salto condicional si no hay overflów (OF=0)
JP or JPE	Salto condicional si hay paridad de bits (PF=1) (par)
JNP or JPO	Salto condicional si no hay paridad de bits (PF=0) (impar)
JS	Salto condicional si es negativo (SF=1)
JNS	Salto condicional si es positivo (SF=0)

MISCELÁNEOS

STD	Autodecrementar el puntero en instrucciones de cadenas de caracteres
CLD	Autoincrementar el puntero en instrucciones de cadenas de caracteres
STC	Cargar el bit de carry (lo pone en 1)
CLC	Limpiar el bit de carry (lo pone en 0)
CMC	Complementar el bit de carry (cambia su valor)
STI	Cargar el bit de interrupciones, habilita las interrupciones (lo pone en 1)
CLI	Limpiar el bit de interrupciones, deshabilita las interrupciones (lo pone en 0)
HLT	Esperar hasta una interrupción
WAIT	Esperar por ITEST para continuar
ESC	Dar e! control del bus a! coprocesador
LOCK	Bloqueo del bus, previene el uso del bus en la siguiente instrucción
NOP	No operación (no realizar acción)