TEC | Tecnológico de Costa Rica

Principios básicos ASM

Lenguaje Ensamblador Intel8086

Contenidos

- Lenguaje ensamblador
- Conceptos introductorios
- Formatos de Instrucción
- Modos de Direccionamiento
- Conjunto de instrucciones básico

Archivos fuente, objeto y ejecutable (I)

- Son necesarias 3 herramientas para escribir programas en lenguaje ensamblador:
  - Un ***editor de textos*** para hacer los archivos fuentes,
  - El ***MASM*** (o alguna otra versión de ensamblador; por ejemplo TASM) que sirve para generar archivos objetos a partir de los archivos fuentes,
  - El ***LINK*** para combinar uno o mas archivos objetos hacia un archivo ejecutable que puede ser ejecutado por DOS.

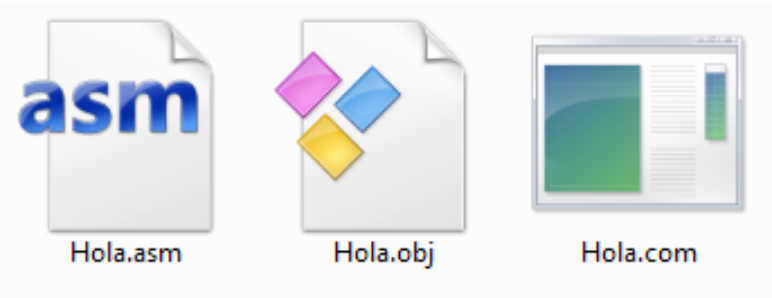
Archivos fuente, objeto y ejecutable (II)

- Después de que se crea un programa fuente en ASM, este debe ser almacenado en un archivo.
- Este es referido como archivo fuente, que es un archivo de texto que contiene enunciados en lenguaje ensamblador, cada uno de estos termina con los caracteres CR y LF (Retorno de carro y salto de línea).
- Generalmente los nombres de los archivos fuentes tienen la extensión ASM.

Archivos fuente, objeto y ejecutable (III)

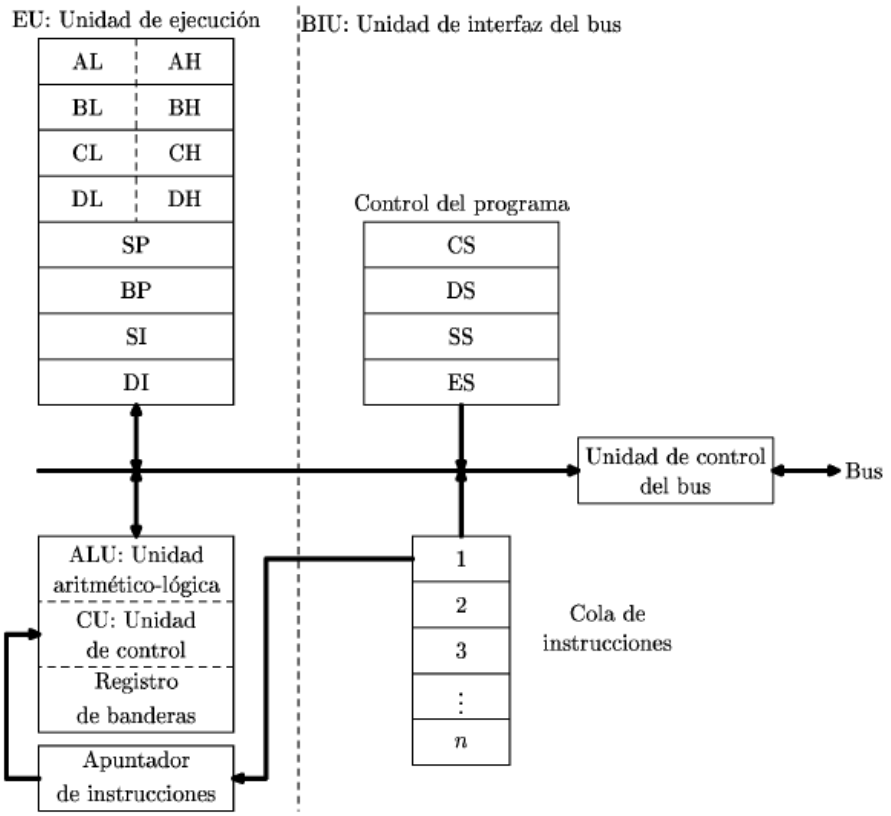
- El resultado de ensamblar un archivo fuente es un archivo binario con el código maquina y las instrucciones para el encadenador **(LINK).**
- Este archivo es llamado archivo objeto y tiene la extensión por defecto OBJ.
- Uno o mas archivos objeto son combinados por el encadenador para formar un programa ejecutable, el cual tiene la extensión por defecto EXE.

.ASM → .OBJ → .EXE



Arquitectura 8086 (I)

- El 8086 se divide en dos unidades lógicas: una **unidad de ejecución** (**EU**) y una **unidad de interfaz del bus** (**BIU**).
- El papel de la EU es ejecutar instrucciones, mientras que la BIU envía instrucciones y datos a la EU.
- La EU posee una unidad aritmético-lógica, una unidad de control y 10 registros.
- Permite ejecutar las instrucciones, realizando todas las operaciones aritméticas, lógicas y de control necesarias.



Arquitectura 8086 (II)

- La BIU tiene tres elementos fundamentales: la unidad de control del bus, la cola de instrucciones y los registros de segmento.
- La BIU controla el bus externo que comunica el procesador con la memoria y los distintos dispositivos de E/S.
- Los registros de segmento controlan el direccionamiento y permiten gestionar hasta 1 MB de memoria principal.
- La BIU accede a la memoria para recuperar las instrucciones que son almacenadas en la cola de instrucciones constituida por 6 bytes (4 bytes para el 8088).
- Mientras la BIU busca las instrucciones, la EU ejecuta las instrucciones que va recogiendo de la cola, es decir, la BIU y la EU trabajan en paralelo.

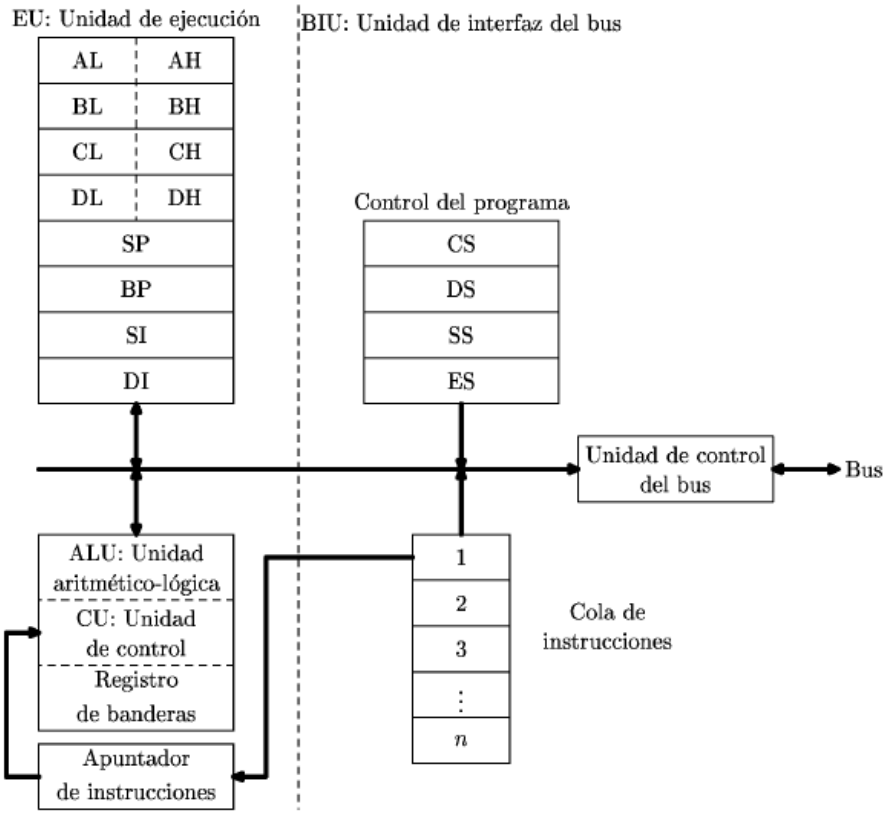


Figura 2. *Arquitectura interna del 8086.*

Registros 8086/88 (I)

- Los registros almacenan posiciones de memoria que van a sufrir repetidas manipulaciones, ya que los accesos a memoria son mucho más lentos que los accesos a los registros.
- El 8086 dispone de 14 registros de 16 bits que se emplean para controlar la ejecución de instrucciones, direccionar la memoria y proporcionar capacidad aritmética y lógica.
- Cada registro puede almacenar datos o direcciones de memoria.
- Los registros son direccionables por medio de un nombre.

Registros 8086/88 (II)

- Registros de propósito general o de datos,
- Registros de segmento,
- Registro apuntador de instrucciones (IP),
- Registros apuntadores (SP y BP),
- Registros índice (SI y DI) y
- Registro de banderas, FLAGS o registro de estado (FL).

Registros 8086/88 (III)

AX
BX
CX
DX

Registros de propósito general o de datos

SP
BP
SI
DI

Registros punteros y Registros índice

CS
DS
SS
ES

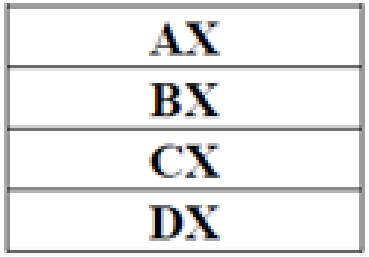
Registros de segmento

IP
FLAGS o FL

Registro puntero de instrucciones; y Registro de banderas, FLAGS o de estado (FL)

Registros de propósito general (I)

- Se utilizan **para cálculo y almacenamiento de propósito general**.
- Los programas leen datos de memoria y los dejan en estos registros, ejecutan operaciones sobre ellos, y guardan los resultados en memoria.
- Hay cuatro registros de propósito general que, aparte de ser usados a voluntad por el programador, tienen fines específicos



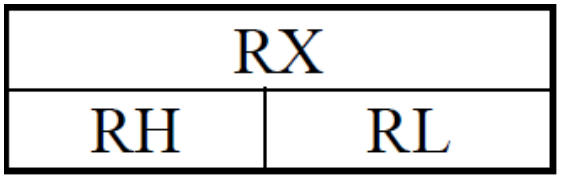
Registros de propósito general o de datos

Registros de propósito general (II)

Registro AX	Este registro es el acumulador principal, implicado en gran parte de las operaciones de aritméticas y de E/S.
Registro BX	Recibe el nombre de registro base ya que es el único registro de propósito general que se usa como un índice en el direccionamiento indexado. Se suele utilizar para cálculos aritméticos.
Registro CX	El CX es conocido como registro contador ya que puede contener un valor para controlar el número de veces que se repite una cierta operación.
Registro DX	Se conoce como registro de datos. Algunas operaciones de E/S requieren su uso, y las operaciones de multiplicación y división con cifras grandes suponen que el DX y el AX trabajando juntos.

Registros de propósito general (III)

- Los registros de propósito general se pueden direccionar como una palabra o como un byte.
- El byte de la izquierda es la parte alta y el byte de la derecha es la parte baja:



- Siguiendo esta nomenclatura, es posible referirse a cada uno de los dos bytes, byte de orden alto o más significativo y byte de orden bajo o menos significativo, de cada uno de estos registros.
  - Por ejemplo: AH es el byte más significativo del registro AX, mientras que AL es el byte menos significativo.

Registros de Segmento

- Los registros de segmento son registros de 16 bits que constituyen la implementación física de la arquitectura segmentada del 8086

Registro CS	Registro Segmento de Código. Establece el área de memoria dónde está el programa durante su ejecución.
Registro DS	Registro Segmento de Datos. Especifica la zona donde los programas leen y escriben sus datos.
Registro SS	Registro Segmento de Pila. Permite la colocación en memoria de una pila, para almacenamiento temporal de direcciones y datos.
Registro ES	Registro Segmento Extra. Se suele utilizar en algunas operaciones con cadenas de caracteres para direccionar la memoria.

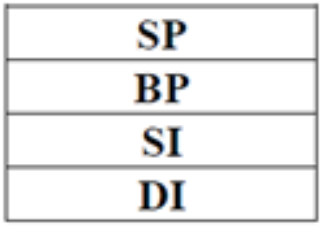
Registro Apuntador de Instrucciones (IP)

- Se trata de un registro de 16 bits que contiene el desplazamiento de la dirección de la siguiente instrucción que se ejecutará.
- Está asociado con el registro CS en el sentido de que IP indica el desplazamiento de la siguiente instrucción a ejecutar dentro del segmento de código determinado por CS

Dirección del segmento de código en CS:	25A40H
Desplazamiento dentro del segmento de código en IP:	+ 0412H
Dirección de la siguiente instrucción a ejecutar:	25E52H

Registros Apuntadores (SP y BP)

- Los registros apuntadores están asociados al registro de segmento SS y permiten acceder a los datos almacenados en la pila

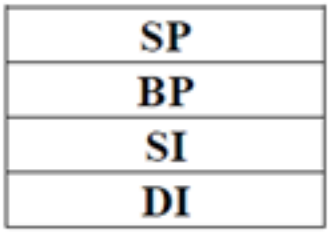


Registros punteros y Registros índice

Registro SP	Proporciona un valor de desplazamiento que se refiere a la palabra actual que está siendo procesada en la pila.
Registro BP	Facilita la referencia a los parámetros de las rutinas, los cuales son datos y direcciones transmitidos vía la pila.

Registros Índice (SI y DI)

- Los registros índice se utilizan fundamentalmente en operaciones con cadenas y para direccionamiento indexado



Registros punteros y Registros índice

Registro SI	Registro índice fuente requerido en algunas operaciones con cadenas de caracteres. Este registro está asociado con el registro DS.
Registro DI	Registro índice destino requerido también en determinadas operaciones con cadenas de caracteres. Está asociado al registro DS o ES.

Registro de Banderas (FLAGS) (I)

- Es un registro de 16 bits, pero sólo se utilizan nueve de ellos. Sirven para indicar el estado actual de la máquina y el resultado del procesamiento.
- La mayor parte de las instrucciones de comparación y aritméticas modifican este registro.
- Algunas instrucciones pueden realizar pruebas sobre este registro para determinar la acción siguiente.

Registro de Banderas (FLAGS) (II)

- Los bits 0, 2, 4, 6, 7 y 11 son indicadores de condición que reflejan los resultados de las operaciones del programa; los bits 8 al 10 son indicadores de control que, modificados por el programador, sirven para controlar ciertos modos de procesamiento, y el resto no se utilizan.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	**O**	**D**	**I**	**T**	**S**	**Z**	-	**A**	-	**P**	-	**C**

Registro de Banderas (FLAGS) (III)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	**O**	**D**	**I**	**T**	**S**	**Z**	-	**A**	-	**P**	-	**C**

OF	Bit de Overflow o desbordamiento. Indica desbordamiento de un bit de orden alto (más a la izquierda), después de una operación aritmética.
DF	Bit de Dirección. Designa la dirección, creciente (0) o decreciente (1), en operaciones con cadenas de caracteres.
IF	Bit de Interrupción. Indica que una interrupción externa, como la entrada desde el teclado, sea procesada o ignorada.
TF	Bit de Trap o Desvío. Procesa o ignora la interrupción interna de *trace* (procesamiento paso a paso).
SF	Bit de Signo. Indica el valor del bit más significativo del registro después de una operación aritmética o de desplazamiento.
ZF	Bit Cero. Se pone a 1 si una operación produce 0 como resultado.
AF	Bit de Carry Auxiliar. Se pone a 1 si una operación aritmética produce un acarreo del bit 3 al 4. Se usa para aritmética especializada (ajuste BCD).
PF	Bit de Paridad. Se activa si el resultado de una operación tiene paridad par.
CF	Bit de Acarreo. Contiene el acarreo de una operación aritmética o de desplazamiento de bits.

Ejemplo

```
; To produce a .COM file, .model must be tiny, also you must add /tiny to the linker command line

.model tiny
.data
      ; El string debe terminar con $
      msg   db "Este es un ejecutable DOS de 16 bits (.COM)",13,10,"Hola mundo!",13,10,"$"
.code
    .startup
    mov   dx,offset msg ; Toma la dirección de nuestro mensaje en el DX
    mov   ah,9          ; La función 09h en AH significa "Escriba un string en la salida estándar"
    int   21h           ; LLama la interrupción de DOS (DOS function call)

    mov   ah,0          ; LLama la un función del bios "GET KEYSTROKE"
    int   16h
    .exit
end
```

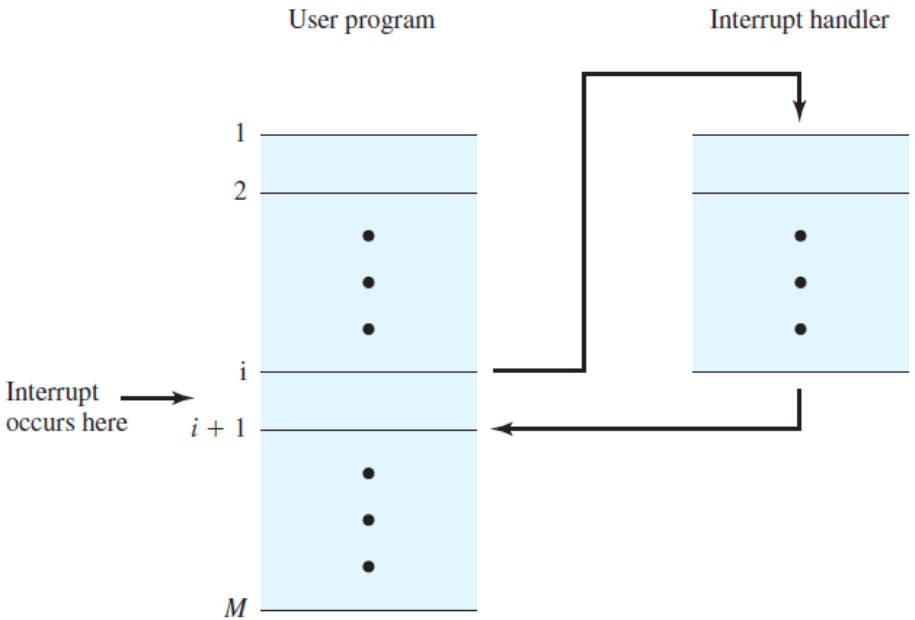
¿Porqué se utiliza el modelo tiny?

¿Importa que se almacene en dx?

Registro DX	Se conoce como registro de datos. Algunas operaciones de E/S requieren su uso, y las operaciones de multiplicación y división con cifras grandes suponen que el DX y el AX trabajando juntos.

Interrupciones

- Una interrupción es una situación especial que suspende la ejecución de un programa de modo que el sistema pueda realizar una acción para tratarla.
- Tal situación se da, por ejemplo, cuando un periférico requiere la atención del procesador para realizar una operación de E/S.



Tomado de: Stallings, W. Operating Systems, 6.a ed., Prentice Hall, 2008. Pag. 18

Instrucciones de transferencia de datos (MOV) (I)

Operación	**MOV: Mover datos**
Descripción	Transfiere datos entre dos registros o entre un registro y memoria, y permite llevar datos inmediatos a un registro o a memoria.
Banderas	No las afecta.
Formato	MOV {registro/memoria},{registro/memoria/inmediato}
Ejemplo	MOV AX, 54AFH

- Su misión es intercambiar la información entre los registros y las posiciones de memoria.
- Las operaciones de este tipo más relevantes son:

MOV**{**REGISTRO/MEMORIA**}**, **{**REGISTRO/MEMORIA/INMEDIATO**}**

Instrucciones de transferencia de datos (MOV) (II)

Operación	**MOV: Mover datos**
Descripción	Transfiere datos entre dos registros o entre un registro y memoria, y permite llevar datos inmediatos a un registro o a memoria.
Banderas	No las afecta.
Formato	MOV {registro/memoria},{registro/memoria/inmediato}
Ejemplo	MOV AX, 54AFH

```
; To produce a .COM file, .model must be tiny, also you must add /tiny to the linker command line

.model tiny
.data
      ; El string debe terminar con $
      msg   db "Este es un ejecutable DOS de 16 bits (.COM)",13,10,"Hola mundo!",13,10,"$"
.code
    .startup
    mov   dx,offset msg ; Toma la dirección de nuestro mensaje en el DX
    mov   ah,9          ; La función 09h en AH significa "Escriba un string en la salida estándar"
    int   21h           ; LLama la interrupción de DOS (DOS function call)

    mov   ah,0          ; LLama la un función del bios "GET KEYSTROKE"
    int   16h
    .exit
end
```

Instrucciones de transferencia de datos (MOV) (III)

- Las operaciones **MOV** no permitidas son de memoria a memoria, inmediato a registro de segmento y de registro de segmento a registro de segmento. Para estas operaciones es necesario utilizar más de una instrucción.
- No pueden utilizarse registros de segmento como operandos, ni tampoco dos direcciones de memoria.

Operación	**XCHG: Intercambiar datos**
Descripción	Intercambia datos entre dos registros o entre un registro y memoria.
Banderas	No las afecta.
Formato	XCHG {registro/memoria},{registro/memoria}
Ejemplo	XCHG AL,AH

Instrucciones de transferencia de datos (PUSH)

Operación	**PUSH: Guardar en la pila**
Descripción	Guarda en la pila una palabra para su uso posterior. SP apunta al tope de la pila; PUSH decrementa SP en 2 y transfiere la palabra a SS:SP.
Banderas	No las afecta.
Formato	PUSH {registro/memoria/inmediato(sólo 286 o posteriores)}
Ejemplo	PUSH DX

```
...
add al,bl    ; los sumanos
add al,48    ; obtenemos el equivalente ASCII
push ax      ; respaldamos en la pila
...
```

Instrucciones de transferencia de datos (POP)

Operación	**POP: Sacar una palabra de la pila**
Descripción	Saca de la pila una palabra previamente guardada y la envía a un destino especificada. SP apunta al tope de la pila; POP la transfiere al destino especificado e incrementa SP en 2.
Banderas	No las afecta.
Formato	POP {registro(excepto CS, se debe usar RET)/memoria}
Ejemplo	POP BX

```
pop ax              ; recuperamos de la pila
mov ah,02h
mov dl,al
int   21h           ; Call the DOS interrupt (DOS function call)
```

Instrucciones Aritméticas (I)

- Sirven para llevar a cabo operaciones aritméticas manipulando los registros y las posiciones de memoria: suma, resta, multiplicación, división, incremento, decremento y versiones con acarreo.

Instrucciones Aritméticas (II)

Operación	**ADD: Sumar números binarios**
Descripción	Suma números binarios desde la memoria, registro o inmediato a un registro, o suma números en un registro o inmediato a memoria. Los valores pueden ser un byte o una palabra.
Banderas	AF, CF, OF, PF, SF y ZF.
Formato	ADD {registro/memoria},{registro/memoria/inmediato}
Ejemplo	ADD DL,AL

Operación	**ADC: Sumar con acarreo**
Descripción	Por lo común es usado en suma de múltiples palabras binarias para acarrear un bit en el siguiente paso de la operación. ADC suma el contenido de la bandera CF al primer operando y después suma el segundo operando al primero, al igual que ADD.
Banderas	AF, CF, OF, PF, SF y ZF.
Formato	ADC {registro/memoria},{registro/memoria, inmediato}
Ejemplo	ADC AX,CX

Instrucciones Aritméticas (III)

Operación	**SUB: Restar números binarios**
Descripción	Resta números binarios en un registro, memoria o inmediato de un registro, o resta valores en un registro o inmediato de memoria.
Banderas	AF, CF, OF, PF, SF y ZF.
Formato	SUB {registro/memoria},{registro/memoria/inmediato}
Ejemplo	SUB AL,CL

Operación	**SBB: Restar con acarreo**
Descripción	Normalmente, se usa esta operación en la resta binaria de múltiples palabras para acarrear el bit uno de desbordamiento al siguiente paso de la aritmética. SBB resta primero el contenido de la bandera CF del primer operando y después el segundo operando del primero, de manera similar a SUB.
Banderas	AF, CF, OF, PF, SF y ZF.
Formato	SBB {registro/memoria},{registro/memoria/inmediato}
Ejemplo	SBB AX,CX

Instrucciones Aritméticas (IV)

Operación	**DEC: Disminuye en uno**
Descripción	Disminuye 1 de un byte o una palabra en un registro o memoria.
Banderas	AF, OF, PF, SF y ZF.
Formato	DEC {registro/memoria}
Ejemplo	DEC DL

Operación	**INC: Incrementa en uno**
Descripción	Incrementa en uno un byte o una palabra en un registro o memoria.
Banderas	AF, OF, PF, SF y ZF.
Formato	INC {registro/memoria}
Ejemplo	INC [1B15h]

Instrucciones Aritméticas (V)

Operación	**MUL: Multiplicar sin signo**
Descripción	Multiplica dos operandos sin signo.
Banderas	CF y OF (AF, PF, SF y ZF quedan indefinidas). CF = OF = 1 ⇒ AH ≠ 0 Ó DX ≠ 0.
Formato	MUL {registro/memoria} (Ver tabla)
Ejemplo	--

Operando 1	Operando 2	Producto	Ejemplo
AL	R/M 8 bits	AX	MUL BL
AX	R/M 16 bits	DX:AX	MUL BX

Instrucciones Aritméticas (VI)

Operación	**IMUL: Multiplicar con signo (enteros)**
Descripción	Multiplica dos operandos con signo. IMUL trata el bit de más a la izquierda como el signo
Banderas	CF y OF (AF, PF, SF y ZF quedan indefinidas). CF = OF = 1 ⇒ AH ≠ 0 Ó DX ≠ 0.
Formato	IMUL {registro/memoria} (Ver tabla)
Ejemplo	--

Operando 1	Operando 2	Producto	Ejemplo
AL	R/M 8 bits	AX	IMUL BL
AX	R/M 16 bits	DX:AX	IMUL BX

Instrucciones Aritméticas (VII)

Operación	**DIV: Dividir sin signo**
Descripción	Divide un dividendo sin signo entre un divisor sin signo. La división entre cero provoca una interrupción de división entre cero.
Banderas	(AF, CF, OF, PF, SF y ZF quedan indefinidas)
Formato	DIV {registro/memoria} (Ver tabla)
Ejemplo	--

Dividendo	Divisor	Cociente	Resto	Ejemplo
AX	R/M 8 bits	AL	AH	DIV BL
DX:AX	R/M 16 bits	AX	DX	DIV CX

Instrucciones Aritméticas (VIII)

Operación	**IDIV: Dividir con signo**
Descripción	Divide un dividendo con signo entre un divisor con signo. La división entre cero provoca una interrupción de división entre cero. IDIV trata el bit de la izquierda como el signo.
Banderas	(AF, CF, OF, PF, SF y ZF quedan indefinidas)
Formato	IDIV {registro/memoria}
Ejemplo	--

Dividendo	Divisor	Cociente	Resto	Ejemplo
AX	R/M 8 bits	AL	AH	IDIV BL
DX:AX	R/M 16 bits	AX	DX	IDIV CX

Instrucciones Aritméticas (IX)

Operación	**NEG: Niega**
Descripción	Invierte un número binario de positivo a negativo y viceversa. NEG trabaja realizando el complemento a dos.
Banderas	AF, CF, OF, PF, SF y ZF.
Formato	NEG {registro/memoria}
Ejemplo	NEG AL

Ejemplo

- Vamos a realizar algunos ejemplos

Directivas de Ensamblador (I)

- El lenguaje ensamblador permite usar diferentes enunciados que sirven para controlar la forma en que un programa se ensambla y lista.
- Estos enunciados reciben el nombre de *directivas*. Se caracterizan porque sólo tienen influencia durante el proceso de ensamblado, pero no generan código ejecutable alguno.

Directivas de Ensamblador (II)

Directiva	**PAGE**
Descripción	Determina al comienzo del programa el número máximo de líneas para listar en una página, así como el número de columnas. El valor por defecto es cincuenta líneas y ochenta columnas.
Formato	PAGE [longitud] [, ancho]
Ejemplo	PAGE 60, 100

Directiva	**TITLE**
Descripción	Se emplea para hacer que aparezca un título para el programa en la línea dos de cada página del listado.
Formato	TITLE texto
Ejemplo	TITLE PASM Programa en ensamblador

Directivas de Ensamblador (III)

Directiva	**.MODEL (nótese ".")**
Descripción	Especifica el modelo de memoria utilizado Lenguaje puede ser C, Pascal, Assembler, FORTRAN, etc.
Formato	.MODEL modelo[,lenguaje]
Ejemplo	.MODEL Compact

MODELO	**CARACTERÍSTICAS**
TINY	Datos y código cogen en un solo segmento de 64K. Todas las direcciones tanto de datos como de procedimientos son NEAR (sólo especifican un desplazamiento dentro del segmento).
SMALL	Un segmento de datos y otro de código. Todas las direcciones son NEAR, aunque hay un segmento distinto para datos y otro para código.
COMPACT	Múltiples segmentos de datos y un único segmento de código. Las direcciones de datos son FAR -especifican un par (*Segmento,Desplazamiento*)- y las direcciones de código (procedimientos) son NEAR. Este modelo y el anterior son los que normalmente se utilizan.
LARGE	Múltiples segmentos de datos y de código. Tanto los datos como los procedimientos tienen direcciones FAR.
HUGE	Múltiples segmentos de datos y de código. Los segmentos pueden pasar 64K, pero haciendo operaciones especiales de normalización de direcciones. Estas normalizaciones son realizadas por compiladores de C.

Directivas de Ensamblador (IV)

Directiva	**.DATA (nótese ".")**
Descripción	Las declaraciones siguientes se insertarán en el segmento de datos. Se continúa donde la anterior directiva .DATA terminó.
Formato	.DATA

Directiva	**.CODE (nótese ".")**
Descripción	Las declaraciones siguientes se insertarán en el segmento de código. Se continúa donde la anterior directiva .CODE terminó.
Formato	.CODE

Directiva	**.STACK (nótese ".")**
Descripción	Las declaraciones siguientes se insertarán en el segmento de pila. Se continúa donde la anterior directiva .STACK terminó.
Formato	.STACK

Directivas de Ensamblador (V)

Directiva	**END**
Descripción	Finaliza todo el programa.
Formato	OPERACIÓN OPERANDO END [dir_inicial] ; Generalmente etiqueta del PROC principal.

Directivas para definición de datos (I)

- El ensamblador permite definir elementos para datos de diferentes longitudes de acuerdo con un conjunto de directivas específicas para ello.
- El formato general es el siguiente

[nombre]	Dn	Contador_Repeticiones DUP (expresión)

Donde Dn es una de las directivas de la siguiente tabla:

DIRECTIVA	DESCRIPCIÓN
DB	Definir un byte. Sirve además para definir cadenas de caracteres.
DW	Definir una palabra (2 bytes).
DD	Definir una palabra doble (4 bytes).
DF	Definir una palabra larga (6 bytes).
DQ	Definir una palabra cuádruple (8 bytes).
DT	Definir diez bytes (10 bytes).

Directivas para definición de datos (II)

EJEMPLO			COMENTARIO
DATO1	DB	?	No se inicializa.
DATO2	DB	25	Constante decimal.
DATO3	DB	10101011B	Constante binaria.
DATO4	DB	1BH	Constante hexadecimal.
DATO5	DB	1,2,3,4,5,6,7,8,9,10	Diez bytes inicializados.
DATO6	DB	5 DUP(?)	Cinco bytes no inicializados.
DATO7	DB	5 DUP(14)	Cinco bytes incializados a 14.
DATO8	DB	'Cadena de caracteres'	Cadena de caracteres.
DATO9	DW	0FFF0H	Constante hexadecimal.
DATO10	DW	10,12,14,16,18,20	Seis palabras inicializadas.
DATO11	DD	?	No se inicializa.
DATO12	DD	14,49	Dos palabras dobles inicializadas.

Control de Flujo (I)

- En ensamblador hay un conjunto de instrucciones para el control de flujo de un programa, como en cualquier otro lenguaje.
- El más común es el salto incondicional, por ejemplo:

```
jmp Label
```

- Esta instrucción lo que hace es indicar a través de una etiqueta (Alabel) el cambio del control de flujo del programa

```
jmp Label
…
…
Label:
```

Control de Flujo (II)

- En caso de que un programa esté sujeto a una condición, este salto es del tipo condicional a través de la instrucción correspondiente.
- Primeramente comparamos y luego usamos la instrucción de salto que corresponda.
- Ejemplo

```
cmp ax,3 ; AX = 3?
je correcto ;si
```

Condicionales (I)

Instrucción			Condición
JZ	Jump if Zero	salta si cero	ZF=1
JNZ	Jump if Not Zero	salta si no cero	ZF=0
JC	Jump if Carry	salta si acarreo	CF=1
JNC	Jump if Not Carry	salta si no acarreo	CF=0
JO	Jump if Overflow	salta si overflow	OF=1
JNO	Jump if Not Overflow	salta si no overflow	OF=0
JS	Jump if Sign	salta si signo	SF=1
JNO	Jump if Not Sign	salta si no signo	SF=0
JP/JPE	Jump if Parity (Parity Even)	salta si paridad (Paridad Par)	PF=1
JNP/JPO	Jump if Not Parity (Parity Odd)	salta si no paridad (Paridad Par)	PF=0

Condicionales (II)

- Para un salto condicionado por comparación y no directamente por el estado de los flags, hacemos una comparación CMP A, B.

Instrucción			Condición
JA	Jump if Above	salta si por encima	A>B (sin signo)
JAE	Jump if Above or Equal	salta si por encima o igual	A>=B (sin signo)
JB	Jump if Below	salta si por debajo	A<B (sin signo)
JBE	Jump if Below or Equal	salta si por debajo o igual	A<=B (sin signo)
JE	Jump if Equal	salta si igual	A=B
JG	Jump if Greater	salta si mayor	A>B (con signo)
JGE	Jump if Greater or Equal	salta si mayor o igual	A>=B (con signo)
JL	Jump if Less	salta si menor	A<B (con signo)
JLE	Jump if Less or Equal	salta si menor o igual	A<=B (con signo)

CMP: comparación de un valor

```
CMP registro o variable, valor
jxx destino
```

Ejemplo:

```
cmp al,'Y'        ; compara el valor en al con Y
je  IGUAL         ; si es igual entonces salta a IGUAL
```

Bucles (loops) (I)

- Una forma de optimizar el uso de instrucciones de salto anidadas con instrucciones del tipo JMP, es por medio de una instrucción de tipo bucle.
- Esto es, usando la instrucción LOOP y colocando el valor del número de veces que se tiene que hacer el bucle en el registro CX.
- En cada pasada del bucle se ve decrementado el valor de CX (CX-1).

Bucles (loops) (II)

```
mov cx,100        ; bucle de 100 iteraciones
Label:
.
.
.
Loop Label:       ; decrementa CX y salta a la posición Label
```

El equivalente sin usar la instrucción de LOOP es:

```
mov cx,100        ; bucle de 100 iteraciones

Label:
dec cx            ; CX = CX-1
jnz Label         ; continua hasta que el valor de CX=0
```

Macros (I)

- Al igual que una macro utilizada cualquier lenguaje de programación, se trata de una serie de instrucciones que se almacenan para que se puedan ejecutar de forma secuencial mediante una sola llamada u orden de ejecución.
- Dicho de otra forma, una macroinstrucción es una instrucción compleja, formada por otras instrucciones más sencillas.
- Esto permite la automatización de tareas repetitivas.

Macros (II)

- Una definición de macro aparece antes de que cualquier definición de segmento.

```
INICIAREGS    MACRO          ;Define macro
   MOV AX,  @data      ; Cuerpo de
   MOV DS,   AX        ; la definición
   MOV ES,    AX       ; de la macro
ENDM                       ; Fin de la macro
```

Macros (III)

- Una definición de macro aparece antes de que cualquier definición de segmento.

```
DESPLEGAR_MSG   MACRO   MENSAJE  ; Argumento mudo
        MOV   AH,   09H
        LEA    DX,   MENSAJE
        INT     21H
ENDM
```

Procedimientos (I)

- En ensamblador un procedimiento es equivalente a una función en C o Pascal. Es también un mecanismo sencillo de encapsular partes de código. Para definir un procedimiento se siguen los siguientes pasos:

```
PROC AProcedure
.
. ; código
.
ret
ENDP AProcedure
```

- La llamada para ejecutar este procedimiento es:

```
call Aprocedure
```

Procedimientos (II)

- Existen tres formas:
  - Por Registro,
  - Por memoria o
  - Por pila (stack)

Direccionamiento (I)

- **Modos de Direccionamiento.** se denominan modos de direccionamiento a aquellos algoritmos empleados por el procesador para calcular las direcciones de las instrucciones y datos.
- Las técnicas más comunes
  - ***Direccionamiento de registro:*** cuando ambos operandos son un registro:
    Ej: mov bx,ax
  - ***Direccionamiento Inmediato:*** cuando el operando origen es una constante.
    Ej: mov ax,500
  - ***Direccionamiento directo:*** cuando el operando es una dirección de memoria. Esta puede ser especificada con su valor entre corchetes [], o bien mediante una variable previamente definida.
    Ej: mov bx,[1000]
    mov ax,TABLA

Direccionamiento (II)

- ***Otras técnicas de direcciomiento:***
  - ***Direccionamiento indirecto mediante registro:*** la dirección del operando se encuentra almacenada en un registro.
    Ej: mov ax,[bx]
  - ***Direccionamiento por registro base:*** el operando está en memoria en una posición apuntada por el registro BX o BP, al que se le añade un desplazamiento.
    Ej: mov ax,[bp+2]
  - ***Direccionamiento indexado:*** cuando la dirección del operando es obtenida por la suma de un desplazamiento más un índice (DI,SI).
    Ej: mov ax, [TABLA+DI]

Bibliografía

- Ensamblador del 8086/88, apuntes por Juan Fernández Peinador, Dpto. Ingeniería y Tecnología de Computadores Facultad de Informática - Universidad de Murcia Febrero de 1998

Muchas gracias