MANUAL DE PROGRAMACIÓN EN ENSAMBLADOR 8086

ALUMNO: SANTIAGO SÁNCHEZ GONZÁLEZ
TITULACIÓN: I.T.I. SISTEMAS
ARQUITECTURA DE COMPUTADORES I

ÍNDICE:

PÁGINA

INTRODUCCIÓN AL ENSAMBLADOR:

El lenguaje ensamblador es el sistema alfanumérico para escribir código máquina mediante expresiones abreviadas (mnemotécnicos).

La compilación es más complicada porque incluye la conversión de operaciones matemáticas complejas, comandos de lenguaje natural o tipos de comandos complejos.

Cada ordenador tiene su propio lenguaje ensamblador, exclusivo de su CPU; un lenguaje de alto nivel (LAN) puede ser compilado en distintas máquinas.

Es usado principalmente porque hay aplicaciones o programas que deben tratar directamente con los registros de la máquina, la memoria, dispositivos de E/S, etc.

PROCESOS PARA LA CREACIÓN DE UN PROGRAMA:

Para la creación de un programa es necesario seguir cinco pasos: Diseño del algoritmo, codificación del mismo, su traducción a lenguaje máquina, la prueba del programa y la depuración. En la etapa de diseño se plantea el problema a resolver y se propone la mejor solución, creando diagramas esquemáticos utilizados para el mejor planteamiento de la solución. La codificación del programa consiste en escribir el programa en algún lenguaje de programación (en este caso en ensamblador 8086), tomando como base la solución propuesta en el paso anterior. La traducción al lenguaje máquina es la creación del programa objeto, esto es, el programa escrito como una secuencia de ceros y unos que pueda ser interpretado por el procesador. La prueba del programa consiste en verificar que el programa funcione sin errores, o sea, que haga lo que tiene que hacer. La última etapa es la eliminación de las fallas detectadas en el programa durante la fase de prueba. La corrección de una falla normalmente requiere la repetición de los pasos comenzando desde el primero o el segundo. Para crear un programa en ensamblador utilizararemos el debugger, que se encuentra en cualquier PC con el sistema operativo MS-DOS, lo cual lo pone al alcance de cualquier usuario que tenga acceso a una máquina con estas características. Debug solo puede crear archivos con extensión .EXE, y por las características de este tipo de programas no pueden ser mayores de 64 kb, además deben comenzar en el desplazamiento, offset, o dirección de memoria 0100Hh dentro del segmento específico.

REGISTROS INTERNOS DE LA UNIDAD CENTRAL DE PROCESO (UCP):

La UCP o CPU tiene 14 registros internos, cada uno de ellos de 16 bits (una palabra). Los bits están enumerados de derecha a izquierda, de tal modo que el bit menos significativo es el bit 0.

Los registros se pueden clasificar de la siguiente forma:

Registros de datos:

AX: Registro acumulador. Es el principal empleado en las operaciones aritméticas.

BX: Registro base. Se usa para indicar un desplazamiento.

CX: Registro contador. Se usa como contador en los bucles.

DX: Registro de datos. También se usa en las operaciones aritméticas.

Estos registros son de uso general y también pueden ser utilizados como registros de 8 bits, para utilizarlos como tales es necesario referirse a ellos como por ejemplo: AH y AL, que son los bytes alto (high) y bajo (low) del registro AX. Esta nomenclatura es aplicable también a los registros BX, CX y DX.

Registros de segmentos:

CS: Registro de segmento de código. Contiene la dirección de las instrucciones del programa.

DS: Registro segmento de datos. Contiene la dirección del área de memoria donde se encuentran los datos del programa.

SS: Registro segmento de pila. Contiene la dirección del segmento de pila. La pila es un espacio de memoria temporal que se usa para almacenar valores de 16 bits (palabras).

ES: Registro segmento extra. Contiene la dirección del segmento extra. Se trata de un segmento de datos adicional que se utiliza para superar la limitación de los 64Kb del segmento de datos y para hacer transferencias de datos entre segmentos.

Registros punteros de pila:

SP: Puntero de la pila. Contiene la dirección relativa al segmento de la pila.

BP: Puntero base. Se utiliza para fijar el puntero de pila y así poder acceder a los elementos de la pila.

Registros índices:

SI: Índice fuente.

DI: Índice destino.

Puntero de instrucciones:

IP: Registro puntero de instrucción o contador de programa (PC). Contiene el desplazamiento de la siguiente instrucción a ejecutar respecto al segmento de código en ejecución. Por lo tanto, la dirección completa de la siguiente instrucción sería CS:IP. La única forma de influir en este registro es de forma indirecta mediante instrucciones de bifurcación.

Registro de banderas (flags):

Cada bandera es un bit y se usa para registrar la información de estado y de control de las operaciones del microprocesador. Hay nueve banderas (los 7 bits restantes no se utilizan):

Banderas de estado: Registran el estado del procesador, normalmente asociado a una comparación o a una instrucción aritmética.

CF: Bandera de acareo.

OF: Bandera de desbordamiento (aritmético).

ZF: Bandera de resultado 0 o comparación igual.

SF: Bandera de resultado o comparación negativa.

PF: Bandera de paridad (número par de bits).

AF: Bandera auxiliar. Indica si hay necesidad de ajuste en las operaciones aritméticas con números BCD.

Banderas de control:

DF: Bandera de dirección. Controla la dirección de las operaciones con cadenas de caracteres incrementando o decrementando automáticamente los registros índices (SI y DI)

IF: Bandera de interrupciones. Indica si están permitidas o no las interrupciones de los dispositivos externos.

TF: Bandera de atrape. Controla la operación de modo paso a paso (usada por el programa DEBUG).

SEGMENTOS Y REGISTROS ASOCIADOS:

La arquitectura de los procesadores x86 obliga al uso de segmentos de memoria para manejar la información, el tamaño de estos segmentos es de 64kb. La razón de ser de estos segmentos es que, considerando que el tamaño máximo de un número que puede manejar el procesador esta dado por una palabra de 16 bits o registro, no sería posible accesar a más de 65536 localidades de memoria utilizando uno solo de estos registros, ahora, si se divide la memoria de la pc en grupos o segmentos, cada uno de 65536 localidades, y utilizamos una dirección en un registro exclusivo para localizar cada segmento, y entonces cada dirección de una casilla específica la formamos con dos registros, nos es posible acceder a una cantidad de 4294967296 bytes de memoria, lo cual es, en la actualidad, más memoria de la que veremos instalada en una PC. Para que el ensamblador pueda manejar los datos es necesario que cada dato o instrucción se encuentren localizados en el área que corresponde a sus respectivos segmentos. El ensamblador accede a esta información tomando en cuenta la localización del segmento, dada por los registros DS, ES, SS y CS, y dentro de dicho registro la dirección del dato específico. Es por ello que cuando creamos un programa empleando el Debug en cada línea que vamos ensamblando aparece algo parecido a lo siguiente:

1CB0:0102 MOV AX,BX

En donde el primer número, 1CB0, corresponde al segmento de memoria que se está utilizando, el segundo se refiere la la dirección dentro de dicho segmento, y a continuación aparecen las instrucciones que se almacenaran a partir de esa dirección.

La forma de indicarle al ensamblador con cuales de los segmentos se va a trabajar es por medio de las directivas .CODE, .DATA y .STACK.

El ensamblador se encarga de ajustar el tamaño de los segmentos tomando como base el número de bytes que necesita cada instrucción que va ensamblando, ya que sería un desperdicio de memoria utilizar los segmentos completos. Por ejemplo, si un programa únicamente necesita 10kb para almacenar los datos, el segmento de datos únicamente será de 10kb y no de los 64kb que puede manejar.

Un programa consta de 4 tipos de segmentos. Cada segmento se direcciona mediante un determinado tipo de registro de segmento:

Segmento código: Cada instrucción se direcciona mediante el registro segmento de código y el registro de desplazamiento IP, CS:IP.

Segmento de datos: Los datos de direccionan mediante el registro de segmento de dato y un registro de desplazamiento (BX, SI o DI), por ejemplo DS:BX.

Segmento de pila: Los datos se direccionan mediante el registro segmento de pila y un registro de desplazamiento (SP o BP), por ejemplo SS:SP.

Segmento extra: Igual que el de datos, sustituyendo el registro de segmento, por ejemplo ES:BX.

SOFTWARE NECESARIO:

Para poder crear un programa se requieren varias herramientas: Primero un editor para crear el programa fuente. Segundo un compilador (Turbo Assembler) que no es mas que un programa que "traduce" el programa fuente a un programa objeto. Y tercero un enlazador (Turbo Linker), que genere el programa ejecutable a partir del programa objeto. El editor puede ser cualquier editor de textos que se tenga a la mano, como compilador utilizaremos el programa Turbo Assembler (TASM), y como enlazador utilizaremos el Turbo Link (TLINK). La extensión usada para que el TASM reconozca

los programas fuente en ensamblador es .ASM, una vez traducido el programa fuente, el TASM crea un archivo con la extensión .OBJ, este archivo contiene un "formato intermedio" del programa, llamado así porque aún no es ejecutable pero tampoco es ya un programa en lenguaje fuente. El enlazador genera, a partir de un archivo .OBJ o la combinación de varios de estos archivos, un programa ejecutable, cuya extensión es .EXE.

LA ESTRUCTURA DEL ENSAMBLADOR:

En el lenguaje ensamblador las líneas de código constan de dos partes, la primera es el nombre de la instrucción que se va a ejecutar y la segunda son los parámetros del comando u operando. Por ejemplo:

ADD ah bh

Aquí "ADD" es el comando a ejecutar (en este caso una adición o suma) y tanto "ah" como "bh" son los parámetros.

El nombre de las instrucciones en este lenguaje puede estar formado de 2 a 6 letras, a estas instrucciones también se les llama nombres mnemónicos o códigos de operación, ya que representan alguna función que habrá de realizar el procesador. Existen algunos comandos que no requieren parámetros para su operación, así como otros que requieren solo un parámetro.

Algunas veces se utilizarán las instrucciones como sigue:

ADD al,[170]

Los corchetes en el segundo parámetro nos indican que vamos a trabajar con el contenido de la casilla de memoria número 170 y no con el valor 170, a esto se le conoce como direccionamiento directo (en la instrucción viene la dirección del objeto).

TÉCNICAS DE CODIFICACIÓN EN ENSAMBLADOR:

En ensamblador los bucle, condicionales, subrutinas y demás elementos se codifica de forma distinta de cómo se hace en otros lenguajes de alto nivel. A continuación describiremos algunas técnicas útiles para codificar las estructuras mas usuales.

BUCLES:

Inicialización de un registro(a modo de contador) al nº de vueltas del bucle
Etiqueta: Primera instrucción del bucle
<resto de instrucciones dentro del bucle>
decrementar el contador
saltar a la etiqueta si el contador es mayor que cero
<instrucciones fuera del bucle>

SALTOS CONDICIONALES:

Instrucción de comparación
Evaluar condición y si se cumple saltar a la etiqueta prefijada
<instrucciones para el caso de no cumplirse la condición>
saltar fuera del condicional

MACROS:

Declaración:
<nombre de la macro> MACRO
<instrucciones de la macro>
ENDM

Llamada:
<nombre de la macro>

SUBRUTINAS:
Declaración:
<nombre la de subrutina> PROC
<instrucciones de la subrutina>
RET
<nombre de la subrutina> ENDP

Llamada:
CALL <nombre de la subrutina>

EJEMPLO PRÁCTICO DE UN PROGRAMA EN ENSAMBLADOR 8086:

A continuación haremos un pequeño programa de ejemplo que nos imprimirá por pantalla la palabra "HOLA", para hacer comentarios se usará el ";" dentro del propio código fuente como vemos a continuación:

```
.MODEL SMALL                ;modelo pequeño de compilación (64k como
                             máximo)
.STACK 100h                 ;Segmento de pila: Pila 256 posiciones
CR EQU 13                   ;Declaración de constantes
LF EQU 10
.DATA                       ;Comienzo del segmento de datos
 TEXTO DB 'HOLA$',CR,LF     ;Reserva de memoria con retorno de carro e
                            ;Inicio de línea, la variable TEXTO es de tipo
                            ;byte,el símbolo $ es el símbolo de fin de cadena
.CODE                       ;Comienzo del segmento de código
 MOV AX, SEG TEXTO          ;Mueve a AX la dirección del primer byte del
                            ;segmento de datos
 MOV DS,AX                  ;Mueve dicha dirección al segmento de datos
 LEA DX,TEXTO               ;Cargamos en DX la dirección efectiva del texto
                            ;que vamos a imprimir por pantalla
 MOV AH,9                   ;función "Escribir texto por pantalla"
 INT 21h                    ;Llamada a DOS por medio de la interrupción
                            ;para ejecutar la función
 MOV AH,4ch                 ;función "Retornar a DOS"
 INT 21h                    ;Interrupción que llama a DOS para terminar el
                            ;programa
END                         ;fin
```

INTERRUPCIONES:

Definición de interrupción:
Una interrupción es una instrucción que detiene la ejecución de un programa para permitir el uso de la UCP a un proceso prioritario. Una vez concluido este último proceso se devuelve el control a la aplicación anterior. Por ejemplo, cuando estamos trabajando con un procesador de palabras y en ese momento llega un aviso de uno de los puertos de comunicaciones, se detiene temporalmente la aplicación que estábamos

utilizando para permitir el uso del procesador al manejo de la información que está
legando en ese momento. Una vez terminada la transferencia de información se reanudan las funciones normales del procesador de palabras. Las interrupciones ocurren muy seguido, sencillamente la interrupción que actualiza la hora del día ocurre aproximadamente 18 veces por segundo. Para lograr administrar todas estas interrupciones, la computadora cuenta con un espacio de memoria, llamado memoria baja, donde se almacenan las direcciones de cierta localidad de memoria donde se encuentran un juego de instrucciones que la UCP ejecutará para después regresar a la aplicación en proceso. La sentencia para usar una interrupción en ensamblador es INT número. En el programa anterior hicimos uso de la interrupción número 21h para llamar a DOS.

A continuación describiremos los 3 tipos de interrupciones:

1) INTERRUPCIONES INTERNAS DE HARDWARE:

Las interrupciones internas son generadas por ciertos eventos que surgen durante la ejecución de un programa. Este tipo de interrupciones son manejadas en su totalidad por el hardware y no es posible modificarlas. Un ejemplo claro de este tipo de interrupciones es la que actualiza el contador del reloj interno de la computadora, el hardware hace el llamado a esta interrupción varias veces durante un segundo para mantener la hora actualizada. Aunque no podemos manejar directamente esta interrupción (no podemos controlar por software las actualizaciones del reloj), es posible utilizar sus efectos en la computadora para nuestro beneficio, por ejemplo para crear un "reloj virtual" actualizado continuamente gracias al contador del reloj interno. Unicamente debemos escribir un programa que lea el valor actual del contador y lo traduzca a un formato entendible para el usuario.

2) INTERRUPCIONES EXTERNAS DE HARDWARE:

Las interrupciones externas las generan los dispositivos periféricos, como pueden ser: teclado, impresoras, tarjetas de comunicaciones, etc. También son generadas por los coprocesadores. No es posible desactivar a las interrupciones externas. Estas interrupciones no son enviadas directamente a la UCP, sino que se mandan a un circuito integrado cuya función es exclusivamente manejar este tipo de interrupciones. El circuito, llamado PIC 8259A, si es controlado por la UCP utilizando para tal control una serie de vías de comunicación llamadas puertos.

3) INTERRUPCIONES DE SOFTWARE:

Las interrupciones de software pueden ser activadas directamente por el ensamblador invocando al número de interrupción deseada con la instrucción INT. El uso de las interrupciones nos ayuda en la creación de programas, utilizándolas nuestros programas son más cortos, es más fácil entenderlos y usualmente tienen un mejor desempeño debido en gran parte a su menor tamaño. Este tipo de interrupciones podemos separarlas en dos categorías: las interrupciones del sistema operativo DOS y las interrupciones del BIOS. La diferencia entre ambas es que las interrupciones del sistema operativo son más fáciles de usar pero también son más lentas ya que estas interrupciones hacen uso del BIOS para lograr su cometido, en cambio las interrupciones del BIOS son mucho más rápidas pero tienen la desventaja que, como son parte del hardware son muy

específicas y pueden variar dependiendo incluso de la marca del fabricante del circuito. La elección del tipo de interrupción a utilizar dependerá únicamente de las características que le quiera dar a su programa: velocidad (utilizando las del BIOS) o portabilidad (utilizando las del DOS).
Interrupción 21H
Propósito: Llamar a diversas funciones del DOS.
Sintaxis:
Int 21H
Nota: Cuando trabajamos en TASM es necesario especificar que el valor que estamos utilizando es hexadecimal. Esta interrupción tiene varias funciones, para acceder a cada una de ellas es necesario que el registro AH se encuentre el número de función que se requiera al momento de llamar a la interrupción.

SALTOS, CICLOS Y PROCEDIEMIENTOS:

Los saltos incondicionales en un programa escrito en lenguaje ensamblador están dados por la instrucción **JMP**, un salto es alterar el flujo de la ejecución de un programa enviando el control a la dirección indicada. Un ciclo, conocido también como iteración o bucle, es la repetición de un proceso un cierto número de veces hasta que alguna condición se cumpla. En estos ciclos se utilizan los bifurcaciones "condicionales" basados en el estado de las banderas. Por ejemplo la instrucción **JNZ** que salta solamente si el resultado de una operación es diferente de cero y la instrucción **JZ** que salta si el resultado de la operación es cero. Por último tenemos los procedimientos o rutinas, que son una serie de pasos que se usarán repetidamente en el programa y en lugar de escribir todo el conjunto de pasos únicamente se les llama por medio de la instrucción **CALL**. Un procedimiento en ensamblador es aquel que inicie con la palabra **PROC** y termine con la palabra **RET** (ya explicado anteriormente). Realmente lo que sucede con el uso de la instrucción call es que se guarda en la pila el registro **IP** y se carga la dirección del procedimiento en el mismo registro, conociendo que IP contiene la localización de la siguiente instrucción que ejecutara la UCP, entonces podemos darnos cuenta que se desvía el flujo del programa hacia la dirección especificada en este registro. Al momento en que se llega a la palabra ret se saca de la pila el valor de IP con lo que se devuelve el control al punto del programa donde se invocó al procedimiento. Es posible llamar a un procedimiento que se encuentre ubicado en otro segmento, para ésto el contenido de **CS** (que nos indica que segmento se está utilizando) es empujado también en la pila.

MOVIENTO DE LOS DATOS:

En todo programa es necesario mover datos en la memoria y en los registros de la UCP; existen diversas formas de hacer esto: puede copiar datos de la memoria a algún registro, de registro a registro, de un registro a una pila, de la pila a un registro, transmitir datos hacia dispositivos externos así como recibir datos de dichos dispositivos. Este movimiento de datos está sujeto a reglas y restricciones. Algunas de ellas son las que se citan a continuación. No es posible mover datos de una localidad de memoria a otra directamente, es necesario primero mover los datos de la localidad origen hacia un registro y luego del registro a la localidad destino. No se puede mover una constante directamente a un registro de segmentos, primero se debe mover a un registro de la UCP. Es posible mover bloques de datos por medio de las instrucciones **movs**, que copia una cadena de bytes o palabras; **movsb** que copia n bytes de una localidad a otra; y **movsw** copia n palabras de una localidad a otra. Las dos últimas instrucciones toman los valores de las direcciones definidas por DS:SI como grupo de datos a mover y ES:DI como nueva localización de los datos. Para mover los datos también existen las estructuras llamadas pilas, en este tipo de estructuras los datos se introducen con la instrucción **PUSH** y se extraen con la instrucción **POP.** En una pila el primer dato introducido es el último que podemos

sacar, esto es, si en nuestro programa utilizamos las instrucciones:
PUSH AX
PUSH BX
PUSH CX
Para devolver los valores correctos a cada registro al momento de sacarlos de la pila es necesario hacerlo en el siguiente orden:
POP CX
POP BX
POP AX
Para la comunicación con dispositivos externos se utilizan el comando **out** para mandar información a un puerto y el comando **in** para leer información recibida desde algun puerto.
La sintaxis del comando out es:
OUT DX,AX
Donde DX contiene el valor del puerto que se utilizará para la comunicación y AX contiene la información que se mandará.
La sintaxis del comando in es:
IN AX,DX
Donde AX es el registro donde se guardará la información que llegue y DX contiene la dirección del puerto por donde llegará la información.

A continuación nos centraremos con la instrucción MOV, ya que es la más empleada para hacer transferencias de datos.

INSTRUCCIÓN MOV:

Propósito: Transferencia de datos entre celdas de memoria, registros y acumulador. Sintaxis:
MOV Destino,Fuente
Donde Destino es el lugar a donde se moverán los datos y fuente es el lugar donde se encuentran dichos datos. Los diferentes movimientos de datos permitidos para esta instrucción son:
Destino: memoria. Fuente: acumulador
Destino: acumulador. Fuente: memoria
Destino: registro de segmento. Fuente: memoria/registro
Destino: memoria/registro. Fuente: registro de segmento
Destino: registro. Fuente: registro
Destino: registro. Fuente: memoria
Destino: memoria. Fuente: registro
Destino: registro. Fuente: dato inmediato
Destino: memoria. Fuente: dato inmediato Ejemplo:
MOV AX,0006h
MOV BX,AX
MOV AH,4Ch
INT 21H
Este pequeño programa mueve el valor 0006H al registro AX, luego mueve el contenido de AX (0006h) al registro BX, por último mueve el valor 4Ch al semiregistro AH (parte alta del registro de datos AX) para terminar la ejecución con la opción 4C de la interrupción 21h.

OPERACIONES LÓGICAS Y ARITMÉTICAS:

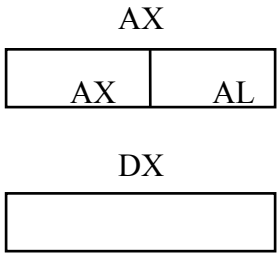
Las instrucciones de las operaciones lógicas son: **AND**, **NOT**, **OR** y **XOR**, éstas trabajan sobre los bits de sus operandos. Para verificar el resultado de operaciones recurrimos a la instrucción **CMP**. Las instrucciones utilizadas para las operaciones algebraicas son: para sumar **ADD**, para restar **SUB**, para

multiplicar **MUL** y para dividir **DIV**.
Casi todas las instrucciones de comparación están basadas en la información contenida en el registro de banderas. Normalmente las banderas de este registro que pueden ser directamente manipuladas por el programador son la bandera de dirección de datos DF, usada para definir las operaciones sobre cadenas. Otra que también puede ser manipulada es la bandera IF por medio de las instrucciones **STI** y **CLI**, para activar y desactivar respectivamente las interrupciones.
A continuación explicaremos brevemente las intrucciones aritméticas y lógicas por separado:

OPERACIONES ARITMÉTICAS:

Las operaciones en aritmética binaria a entera permiten a la CPU realizar cálculos con números enteros positivos y negativos con una representación en complemento a 2.

- **NEG operando:** cambia el signo del operando. Equivaldría al NOT del número y le sumaría 1.
- **ADD destino, fuente:** destino = destino + fuente.
- **ADC destino, fuente:** destino = destino + fuente + carry (acarreo).
- **SUB destino, fuente:** destino = destino - fuente.
- **SBB destino, fuente:** destino = destino - (fuente + acarreo).
- **MUL operando:** multiplica sin considerar el signo. Multiplica el acumulador {AL} o {AX}por el operando fuente. Si el operando fuente es de tipo byte, el resultado se almacena en AX y si es de tipo palabra el resultado se almacena en AX la parte inferior y en DX la palabra superior.
  Si tipo fuente = byte:
  AX = AL * fuente (multiplicación sin signo)
  Si tipo fuente = palabra:
  DX, AX = AX * fuente (multiplicación sin signo)
  Si mitad superior (CF: acarreo) del resultado = 0
  En CC CF = 1
- **IMUL operando:** multiplica considerando el signo.
- **DIV operando:** divide sin considerar el signo, un número contenido en el acumulador entre el operando fuente. El cociente se almacena en el acumulador. El resto se almacena en la extensión del acumulador. Si la extensión de AX será DX (que ocurrirá cuando sea de tipo palabra), la operación y la extensión de AL será AH.



- **IDIV operando:** igual que el DIV pero considerando el signo.
- **CBW:** pasa de byte a palabra el contenido del acumulador.
- **CWD:** pasa de palabra a doble palabra el contenido del acumulador.
- **INC destino:** incrementa el destino.
- **DEC destino:** decrementa el destino.

OPERACIONES LÓGICAS:

Se usan para realizar operaciones a nivel de bits.

- **NOT operando:** cambia los bits 1 por 0 y viceversa y devuelve el resultado en el mismo operando.

```
AL = F2h          AL      1111  0010
NOT AL;     NOT AL      0000  1101 = Odh
```

- **OR destino, fuente:** operación *o lógico inclusivo*. El resultado se almacena en destino.

```
AX = FEDCh=       1111  1110  1101  1100
BX = 1234 h =     0001  0010  0011  0100
      OR AX, BX   1111  1110  1111  1100 = FEFC h
```

- **AND destino, fuente:** la operación *Y lógica* entre 2 operandos, el resultado se deja en destino.

```
AX = FEDC h       1111  1110  1101  1100
BX = 1234 h       0001  0010  0011  0100
ADD AX, BX        0001  0010  0001  0100 = 1214 h
```

- **XOR destino, fuente:** la operación *o lógico exclusiva*; el resultado se deja en destino.

```
AX = FEDC h       1111  1110  1101  1100
BX = 1234 h       0001  0010  0011  0100
XOR AX, BX        1110  1100  1110  1000 = ECE8 h
```

APENDICE 1- INTERRUPCIONES.

DIRECCION (hex)	INTERRUPCION (hex)	FUNCION
0-3	0	Division by zero
4-7	1	Single step trace
8-B	2	Non maskable interrupt
C-F	3	Break point instruction
10-13	4	Overflow
14-17	5	Print screen
18-1F	6,7	Reserved
20-23	8	Timer
24-27	9	Keyboard interrupt
28-37	A,B,C,D	Reserved
38-3B	E	Diskette interrupt
3C-3F	F	Reserved
40-43	10	Video screen I/O
44-47	11	Equipment check
48-4B	12	Memory size check
4C-4F	13	Diskette I/O
50-53	14	Communication I/O
54-57	15	Cassette I/O
58-5B	16	Keyboard input
5C-5F	17	Printer Output
60-63	18	ROM Basic entry code
64-67	19	Bootstrap loader
68-6B	1A	Time of day
6C-6F	1B	Get control on keyboard break
70-73	1C	Get control on timer interrupt
74-77	1D	Pointer to video initialization table
78-7B	1E	Pointer to diskette parameter table
7C-7F	1F	Pointer to table for graphics characters
ASCII 128-255		
80-83	20	DOS program terminate
84-87	21	DOS function call
88-8B	22	DOS terminate address
90-93	24	DOS fatal error vector
94-97	25	DOS absolute disk read
98-9B	26	DOS absolute disk write
9C-9F	27	DOS terminate, fix in storage
A0-FF	28-3F	Reserved for DOS
100-1FF	40-7F	Not used
200-217	80-85	Reserved by BASIC
218-3C3	86-F0	Used by BASIC interpreter
3C4-3FF	F1-FF	Not Used

APÉNDICE 2.- JUEGO DE INSTRUCCIONES DEL 8086.

En la siguiente lista de instrucciones, para la descripción y su sintaxis se recurre a las siguientes abreviaturas:

acum	uno de los acumuladores: AX o AL.
reg	cualquiera de los registros
segreg	uno de los registros de segmento
r/m	uno de los operandos generales: registro, memoria, basado, indexado o basado-indexado
inmed	constante o símbolo de 8 o 16 bits
mem	un operando de memoria: símbolo, etiqueta, variable.
etiqueta	etiqueta de instrucciones.
src	fuente en operaciones de cadena
dest	destino en operaciones de cadena.

8086

AAA	Ajuste ASCII para adición.
AAD	Ajuste ASCII para división.
AAM	Ajuste ASCII para multiplicación.
AAS	Ajuste ASCII para división.
ADC acum, inmed r/m,inmed r/m, reg reg, r/m	Suma con acarreo.
ADD acum,inmed r/m,inmed r/m, reg reg, r/m	Suma.
AND acum,inmed r/m,inmed r/m, reg reg, r/m	Operación AND a nivel bit.
CALL etiqueta r/m	Llamado.
CBW	Convierte byte apalabra.
CLC	Limpia bandera de acarreo.
CLD	Limpia bandera de dirección.
CLI	Limpia bandera de interrupción.
CMC	Complementa bandera de acarreo.
CMP acum,inmed r/m,inmed r/m, reg reg, r/m	Comparación
CMPS src,dest	Comparación de cadenas.
CMPSB	Compara cadenas byte por byte.
CMPSW	Compara cadenas palabra por palabra.
CWD	Convierte palabra a palabra doble.

DAA	Ajuste decimal para adición.
DAS	Ajuste decimal para substracción.
DEC r/m reg	Decremento.
DIV r/m	División.
ESC inmed, r/m	Escape con 6 bits.
HLT	Alto.
IDIV r/m	División entera.
IMUL r/m	Mutiplicación entera.
IN accum,inmed acum, **DX**	Entrada desde puerto.
INC r/m reg	Incremento.
INT3	Interrupción3 codificada como un byte.
INT inmed	Interrupción0-255.
INTO	Interrupción en *overflow*.
IRET	Retorno de interrupción.
JMP etiqueta r/m	Brinco incondicional.
J(condición)etiqueta	Brinca de acuerdo a las condiciones: **A** (arriba), **AE** (arriba o igual), **B** (siguiente), **BE** (siguiente o igual), **C**(acarreo), **CXZ** (**CX** en cero), **E** (igual), **G** (mayor), **GE** (mayor o igual), **L** (menor), **LE** (menor o igual), **NA** (no anterior), **NAE** (no anterior o igual), **NB** (no siguiente), **NBE**(no siguiente o igual), **NC** (no acarreo), **NE** (no igual), **NG**(no mayor), **NGE** (no mayor o igual), **NL** (no menor), **NLE** (no menor o igual), **NO**(no sobreflujo), **NP** (no paridad), **NS** (no signo), **NZ** (no cero), **O**(sobreflujo), **P** (paridad), **PE**(paridad par), **PO** (paridad impar), **S** (signo), **Z** (cero).
LAHF	Carga **AH** con las banderas.
LDS r/m	Carga **DS**.
LEA r/m	Carga la dirección.
LES r/m	Carga **ES**.
LOCK	Cierra bus.
LODS src	Carga cadena.
LODSB	Carga byte de cadena en **AL**.
LODSW	Carga palabra de la cadena en **AX**.
LOOP etiqueta	Ciclo.
LOOPE etiqueta	Ciclo mientras igual.
LOOPNE etiqueta	Ciclo mientras no igual.
LOOPNZ etiqueta	Ciclo mientras no cero.
LOOPZ etiqueta	Ciclo mientras cero.
MOV acum,mem r/m,inmed mem, acum r/m, reg r/m,segreg reg, inmed reg,r/m segreg,r/m	Mueve un valor del segundo al primer operando
MOVS dest, src	Mueve cadena.

MOVSB	Mueve cadena byte por byte.
MOVSW	Mueve cadena palabra por palabra.
MUL r/m	Multiplicación.
NEG r/m	Niega(complemento a 2).
NOP	Operación ociosa.
NOT r/m	Invierte valores de bits (complemento a 1).
OR acum, inmed	Operación OR a nivel de bit.
r/m,inmed	
r/m, reg	
reg,r/m	
OUTDX, accum	Salida por el puerto dado por el primer operando.
inmed, acum	(inmediato de 8 bits)
POP r/m	Recupera valor de la pila.
reg	
segreg	
POPF	Recupera banderas.
PUSH r/m	Guarda valor en la pila.
reg	
segreg	
PUSHF	Guarda banderas.
RCL r/m,**1**	Rotación a la izquierda con acarreo.
r/m,**CL**	
RCR r/m, **1**	Rotación a la derecha con acarreo.
r/m, **CL**	
REP	Repite.
REPE	Repite si igual.
REPNE	Repite si no igual.
REPNZ	Repite si no cero.
REPZ	Repite si cero.
RET [inmed**]**	Regresa después de recuperar bytes de la pila.
ROL r/m,**1**	Rotación a la izquierda.
r/m, **CL**	
ROR r/m,**1**	Rotación a la derecha.
r/m, **CL**	
SAHF	Carga banderas con el valor de **AH**.
SAL r/m, **1**	Desplazamiento aritmético a la izquierda.
r/m, **CL**	
SAR r/m, **1**	Desplazamiento aritmético a la derecha.
r/m, **CL**	
SBB acum, inmed	Substracción con acarreo.
r/m,inmed	
r/m, reg	
reg,r/m	
SCAS dest	Explora cadena.
SCASB	Explora cadena para el byte en **AL**.
SCASW	Explora cadena por la palabra en **AX**.
SHL r/m, **1**	Desplazamiento a la izquierda.
r/m, **CL**	
SHR r/m, **1**	Desplazamiento a la derecha.

r/m, **CL**	
STC	Fija bandera de acarreo.
STD	Fija bandera de dirección.
STI	Fija bandera de interrupción.
STOS dest	Guarda cadena.
STOSB	Guarda byte en **AL** en la cadena.
STOSW	Guarda palabra en **AX** en la cadena.
SUB accum, inmed	Substracción.
r/m,inmed	
r/m, reg	
reg,r/m	
TEST acum, inmed	Comparación.
r/m,inmed	
r/m, reg	
reg,r/m	
WAIT	Aguarda.
XCHG acum, reg	Intercambio.
r/m,inmed	
r/m, reg	
reg,r/m	
XLAT	Traduce.
XOR acum, reg	Operación XOR a nivel bit.
r/m,inmed	
r/m, reg	
reg,r/m	