# UNIVERSIDAD AUTONOMA DE BAJA CALIFORNIA


## Organización de computadoras y lenguaje ensamblador

**Practica 6.** Introducción a los registros del procesador 8086

**Alumno:** Caudillo Sánchez Diego

**Matricula:** 1249199

**Grupo:** 551

**Docente:** Dr. Mauricio Alonso Sánchez

**Objetivo:** Familiarizarse con el uso de DEBUG para identificar los registros, modos de direccionamiento del procesador 8086, así como hacer uso de sus instrucciones básicas.

**Materiales:** debug.exe

**Teoría:** Hacer una reseña sobre:

- La arquitectura interna del procesador 8086
- Describir que es el programa debug.exe

Para un procesador 8086, definir lo siguiente:

- Estructura de un programa
- Modelos de memoria (tiny, small, etc.)
- Definición de segmentos
- Declaración de variables
- Declaración de constantes
- Modos de direccionamiento

Describir las instrucciones para:

- Movimientos
- Aritmética
- Lógica
- Manipulación de bits

**Desarrollo:**

PARTE 1.

Identificar dentro del programa debug.exe: los grupos de registros, y las banderas.

Actividad para validar el desarrollo de esta parte:

1. Mostrar una captura de pantalla donde se muestre cada grupo de registros, y banderas claramente identificados.

PARTE 2.

Mediante el programa de debug.exe: ejemplificar, ejecutar y analizar cada uno de los modos de direccionamientos disponibles en el procesador 8086.

Actividad para validar el desarrollo de esta parte:

1. Por cada modo de direccionamiento agregar capturas de pantalla ejemplificando la identificación de los datos a direccionar, su ejecución, y finalmente como es que este afecta los registros.

PARTE 3.

Mediante el programa de debug.exe: ejemplificar, ejecutar y analizar cada una de las instrucciones disponibles en el procesador 8086:

De movimiento de datos:
PUSH, POP, PUSHF, POPF, XCHG, XLAT, IN, y OUT

Instrucciones aritméticas:
ADD, ADC, INC, SUB, SBB, DEC, MUL, IMUL, DIV, IDIV, CMP, CBW, y CWD

Instrucciones lógicas y de manipulación de bits:
AND, OR, XOR, NOT, TEST, SHL, SAL, SHR, SAR, ROL, RCL, ROR, y RCR

Actividad para validar el desarrollo de esta parte:

1. Por cada instrucción agregar capturas de pantalla ejemplificando la identificación de los datos a direccionar, su ejecución, y finalmente como es que este afecta los registros.

**Arquitectura del 8086**

El 8086 fue diseñado para trabajar con lenguajes de alto nivel, disponiendo de un soporte hardware con el que los programas escritos en dichos lenguajes ocupan un pequeño espacio de código y pueden ejecutarse a gran velocidad. Esta concepción, orientada al uso de compiladores, se materializa en un conjunto de facilidades y recursos, y en unas instrucciones entre las que cabe destacar las que permiten efectuar operaciones aritméticas de multiplicar y dividir, con y sin signo; las que manejan cadenas de caracteres, etc.

El 8086 dispone de instrucciones especiales para el tratamiento de cadenas de caracteres.

- Los registros del 8086 tienen una misión específica, por lo que se podría decir que cada uno de ellos tiene su propia personalidad, aunque varios comparten tareas comunes.
- El encapsulado del 8086 está formado por 40 patillas, simplificando así el hardware, aunque por contra, es necesario la multiplexación del bus de datos con el de direcciones.
- El 8086 dispone de un conjunto de registros, denominados 'cola de instrucciones', en el cual se van almacenando de forma anticipada los códigos de las instrucciones, consiguiendo que este aumente su velocidad de trabajo.
- Las 20 líneas del bus de direcciones sólo permiten direccionar una memoria de 1 Megabyte.
- El 8086 requiere una señal de reloj exterior, siendo 5 y 8 Mhz las frecuencias típicas de funcionamiento.
- El 8086 dispone de una arquitectura "pipe line", es decir, que la CPU puede seguir leyendo instrucciones en los tiempos en que el bus no se utiliza.

Este microprocesador está dividido en dos sub-procesadores. Por un lado está la "Unidad de Ejecución" (EU) encargada de ejecutar las instrucciones, la cual posee una ALU (unidad aritmético-lógica) con un registro de estado con varios *flags* asociados y un conjunto de registros de trabajo, y por otro está la "Unidad de Interfaz de bus" (BIU)encargada de la búsqueda de las instrucciones, ubicarlas en la cola de instrucciones antes de su ejecución y facilitar el direccionamiento de la memoria, es decir, encargada de acceder a datos e instrucciones del mundo exterior.

El 8086 contiene 14 registros de 16 bits, de los cuales, unos pertenecen a la EU, que normalmente se suelen usar para direccionamiento, y otros pertenecen a la BIU.

Los registros del 8086 podrían clasificarse en tres grupos de acuerdo con sus funciones. El grupo de datos, que es esencialmente el conjunto de registros aritméticos; el grupo de apuntadores, que incluye los registros base e índices y también el contador de programa y el puntero de pila; y el grupo de registros de segmento, que es un conjunto de registros base de propósito especial.

El grupo de registros de datos o registros generales son registros de 16 bits, pudiéndose usar cada uno de ellos como dos registros de 8 bits. Aun siendo registros de uso general tiene asignadas unas operaciones específicas. Así, por ejemplo, el AX es el acumulador de 16 bits y usándolo a veces

provoca que el ensamblador produzca un lenguaje máquina codificado en muy pocos octetos. Se emplea en multiplicaciones, divisiones, entradas/salidas, etc.; el registro BX, se utiliza como registro base para el direccionamiento de memoria; el registro CX, se utiliza como contador y almacenaje de datos y el registro DX, se utiliza para almacenar datos de 16 bits. Puede pensarse que es una extensión del registro AX para multiplicaciones y divisiones con 16 bits. Otra de sus funciones específicas es para almacenar la dirección de E/S durante algunas operaciones de E/S.

Aparte de todos estos registros, el 8086 consta de un registro de estado de estado de 16 bits, aunque algunos de ellos no se utilizan. Cada uno de los bits se denomina indicador o *flag*, que generalmente, se modifican por las operaciones lógicas y aritméticas.

Estos indicadores son:

- SF (indicador de signo).
- ZF (indicador de cero).
- PF (indicador de paridad).
- CF (indicador de acarreo).
- AF (indicador de acarreo auxiliar).
- OF (indicador de desbordamiento).
- DF (indicador de dirección).
- IF (indicador de interrupción).
- TF (indicador de trap).

**Debug**

Tradicionalmente, todas las computadoras y sistemas operativos han incluido una función de mantenimiento, utilizada para determinar si un programa está funcionando correctamente. [Citación necesitada] Tim Paterson escribió la depuración para cumplir con este propósito en QDOS. Cuando Paterson comenzó a trabajar para Microsoft a principios de la década de 1980, trajo el programa con él. La depuración fue parte de DOS 1.00 y se ha incluido en MS-DOS y Microsoft Windows. La depuración de DOS tiene varias limitaciones:

Solo puede acceder a registros de 16 bits y no a registros extendidos de 32 bits. Cuando se usa el subcomando "n" para nombrar archivos, el nombre de archivo se almacena desde el desplazamiento DS: 5D a DS: 67 (el área Bloque de control de archivo de prefijo de segmento de programa), lo que significa que el programa solo puede guardar archivos en formato de nombre de archivo FAT.

El paquete DEBUG mejorado incluye una versión "DEBUGX" de clonación de 32 bits que también admite programas DPMI de 32 bits. Andreas "Japheth" Grech, el autor del extensor HX DOS, desarrolló versiones DEBUG mejoradas 0.98 ... 1.25, y el antiguo desarrollador de PC DOS Vernon Brooks agregó versiones 1.26 ... 1.32.

Los sistemas operativos Intel ISIS-II, DEC TOPS-10 y TOPS-20, THEOS / OASIS, Stratus OpenVOS, PC-MOS, y AROS también proporciona un comando de depuración.

**Estructura interna del 8086**

El 8086 usa un esquema llamado segmentación, para acceder correctamente a un megabyte completo de memoria, con referencias de direcciones de sólo 16 bits, y todo esto gracias a la utilización de registros de segmento que dividen esencialmente el espacio de memoria en segmentos de 64K de longitud, que pueden estar separados entre sí, adyacentes o superpuestos, y que comienzan en una dirección divisible por 16.La forma en que se completan los 20 bits del bus de direcciones, disponiendo en la CPU, solamente, registros de 16 bits, se consigue de la siguiente manera:

Se parte del contenido de uno de los registros de segmento, que actúan como base. Después, se multiplica por 16 el contenido del registro de segmento, lo que, en binario, significa añadirle 4 ceros a la derecha y convertirlo en una magnitud de 20 bits. Finalmente, se suma un desplazamiento al resultado de la multiplicación anterior. Abreviadamente, la fórmula para calcular una dirección de memoria es: Dirección Física = 16 * (registro de segmento) + desplazamiento.

Las direcciones completas de las instrucciones y de las posiciones de la pila se forman sumando el contenido de los registros IP y SP con el segmento de código (CS) y el segmento de pila (SS) respectivamente.

La dirección de un dato puede formarse mediante la suma de los contenidos de los registros BX ó BP, los contenidos de SI ó DI, y un desplazamiento. El resultado de este cálculo se denomina dirección efectiva (EA) ó desplazamiento de segmento. La dirección definitiva del dato, sin embargo, se determina mediante la EA y el registro de segmento apropiado, el segmento de datos (DS), el segmento extra (ES) ó el segmento de pila (SS).

El uso de los diferentes segmentos significa que hay áreas de trabajo separadas para el programa, la pila y los datos. Cada área de trabajo tiene un tamaño máximo de 64K bytes y un mínimo de 0. Dado que hay cuatro registros de segmento, uno de programa (CS), uno de pila (SS), uno de datos (DS) y uno extra (ES) el área de trabajo puede llegar hasta 256K.

El programador puede determinar la posición de estos segmentos, cargando el apropiado registro de segmentación de 16 bits con la dirección de segmento apropiada. Esto se realiza normalmente al inicio del programa, pero se puede fácilmente hacer en cualquier momento mientras al programa se ejecuta, cambiando dinámicamente la dirección de estos segmentos.

Cambiando el desplazamiento, el programador puede acceder a cualquier punto de segmento. Se puede pensar en el segmento como una ampliación de memoria que forma un área de trabajo. El desplazamiento es la única parte de la dirección que aparece normalmente en los programas en lenguaje ensamblador durante las referencias a direcciones del área de trabajo.

**Modelos de memoria**

**Tiny:** se combina datos y código en el mismo segmento, debe ser menor que 64k. este modelo permite reara chivos .COM el cual se origina en la localidad 100h.

**Small:** contiene dos segmentos separados

- Code menor o igual a 64K.
- Data menor o igual a 64k.

**Médium:** contiene un segmento de datos y cualquier numero de segmentos de código.

**Compact:** contiende un segmento de código y cualquier numero de segmentos de datos.

**Large**: permite cualquier número de segmentos de datos y código.

**Huge:** igual que Large, pero los segmentos de datos pueden tener más de 64K.

**Definición de segmentos**

**Segmento físico**
Un segmento físico sólo puede comenzar en direcciones de memoria pares divisibles por 16, incluyendo la dirección 0. A estas direcciones se las denomina "párrafos" (paragraph). Se puede reconocer un párrafo ya que su dirección hexadecimal siempre termina con 0, como lo es en 10000h o 2EA70h. Los procesadores 8086/286 utilizan segmentos de un tamaño de 64.

**Segmentos lógicos**
Los segmentos lógicos contienen los tres componentes de un programa: código, datos y pila. Los registros de segmento CS, DS y SS contienen las direcciones de los segmentos de memoria físicos en donde residen los segmentos lógicos.

Se pueden definir segmentos de dos formas distintas: con directivas simplificadas o con directivas completas (también se pueden usar los dos tipos en el mismo programa).

Las directivas simplificadas guardan muchos de los detalles de la definición del segmento, mientras que las directivas completas requieren una sintaxis más compleja, pero brindan un control más completo respecto de la forma en que el ensamblador genera los segmentos. Si se usan directivas completas en la definición de los segmentos, se debe escribir el código necesario para manejar todas las tareas que se hacían en forma automática con las directivas simplificadas.

**Declaración de variables**

**Modos de direccionamiento**

La forma en que se especifica un operando se denomina modo de direccionamiento, es decir, es un conjunto de reglas que especifican la localización (posición) de un dato usado durante la ejecución de una instrucción.

El 8086 tiene 25 modos de direccionamiento o reglas para localizar un operando de una instrucción.

Los modos de direccionamiento más frecuentes son los que calculan la dirección del operando mediante la suma de la dirección base de un registro segmento, multiplicado por 16 y el valor de un desplazamiento.

La gran variedad de direccionamientos proviene de las muchas formas en que se puede determinar el desplazamiento.

En general, los modos de direccionamiento del 8086 se dividen en dos grandes grupos:

1. Modos de direccionamiento de la memoria de programa.
2. Modos de direccionamiento de la memoria de datos.

En la búsqueda de una instrucción, su dirección se obtiene sumando el desplazamiento, contenido en el IP, al valor del registro de segmento CS, multiplicado por 16.

Normalmente al terminar la ejecución de una instrucción, el IP se incrementa, con lo que se pasa a direccionar la siguiente instrucción. Las instrucciones de salto y salto a subrutinas pueden modificar el contenido de IP de tres formas diferentes:

- Por direccionamiento relativo. Al contenido del IP se suma, de forma inmediata, un desplazamiento de 8 a 16 bits, con signo, proporcionado por la misma instrucción.
- Por direccionamiento directo. Se carga, en el IP, una nueva dirección presente en la instrucción.
- Por direccionamiento indirecto. El dato, obtenido por cualquiera de las formas de direccionado de la memoria de datos, es interpretado por las instrucciones de salto, como la dirección a la que se debe saltar.

## Movimientos

Las instrucciones de movimiento copian el elemento de datos al que hace referencia de su segundo operando (es decir, el contenido del registro, el contenido de la memoria o un valor constante) en la ubicación a la que hace referencia de su primer operando (es decir, un registro o una memoria). Mientras que los movimientos de registro a registro son posibles, los movimientos de memoria a memoria directa no lo son. En los casos en que se desean transferencias de memoria, primero se debe cargar el contenido de la memoria de origen en un registro, luego se puede almacenar en la dirección de la memoria de destino.

## Aritmética

**INC:** La instrucción INC se usa para incrementar un operando en uno. Funciona en un solo operando que puede estar en un registro o en la memoria.

**DEC:** La instrucción DEC se usa para decrementar un operando en uno. Funciona en un solo operando que puede estar en un registro o en la memoria.

**ADD/SUB:** Las instrucciones ADD y SUB se utilizan para realizar la suma / resta simple de datos binarios en bytes, palabras y tamaño de doble palabra, es decir, para sumar o restar operandos de 8 bits, 16 bits o 32 bits, respectivamente.

**MUL/IMUL:** Hay dos instrucciones para multiplicar datos binarios. La instrucción MUL (Multiplicar) maneja datos sin firmar y la IMUL (Multiplicación de enteros) maneja datos firmados. Ambas instrucciones afectan al indicador de transporte y desbordamiento.

**DIV/IDIV:** La operación de división genera dos elementos: un cociente y un resto. En caso de multiplicación, el desbordamiento no se produce porque los registros de doble longitud se utilizan para conservar el producto. Sin embargo, en caso de división, puede ocurrir un desbordamiento. El procesador genera una interrupción si se produce un desbordamiento.

La instrucción DIV (división) se usa para datos sin firmar y la IDIV (división entera) se usa para datos firmados.

### Lógica

**AND:** La instrucción AND se utiliza para admitir expresiones lógicas realizando una operación AND a nivel de bits. La operación AND a nivel de bits devuelve 1, si los bits coincidentes de ambos operandos son 1, de lo contrario devuelve 0.

**OR:** La instrucción OR se usa para admitir expresiones lógicas realizando una operación OR a nivel de bits. El operador OR a nivel de bit devuelve 1, si los bits coincidentes de uno o ambos operandos son uno. Devuelve 0, si ambos bits son cero.

**XOR:** La instrucción XOR implementa la operación XOR a nivel de bits. La operación XOR establece el bit resultante en 1, si y solo si los bits de los operandos son diferentes. Si los bits de los operandos son iguales (ambos 0 o ambos 1), el bit resultante se borra a 0.

**TEST:** La instrucción TEST funciona igual que la operación AND, pero a diferencia de la instrucción AND, no cambia el primer operando. Por lo tanto, si necesitamos verificar si un número en un registro es par o impar, también podemos hacerlo usando la instrucción TEST sin cambiar el número original.

**NOT:** La instrucción NO implementa la operación NO bit a bit. El operador NO invierte los bits en un operando. El operando podría estar en un registro o en la memoria.

## Desarrollo

PARTE 1.

```
-r
AX=0000  BX=0000  CX=0000  DX=0000  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=0100   NV UP EI PL NZ NA PO NC
073F:0100 0000          ADD     [BX+SI],AL                      DS:0000=CD
-
```

PARTE 2.

**Direccionamiento a Registro**

```
-a
073F:0100 mov ax,18f5
073F:0103 mov bx,0a45
073F:0106 mov bx,ax
073F:0108
-t

AX=18F5  BX=0000  CX=0000  DX=0000  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=0103   NV UP EI PL NZ NA PO NC
073F:0103 BB450A        MOV     BX,0A45
-t

AX=18F5  BX=0A45  CX=0000  DX=0000  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=0106   NV UP EI PL NZ NA PO NC
073F:0106 89C3          MOV     BX,AX
-t

AX=18F5  BX=18F5  CX=0000  DX=0000  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=0108   NV UP EI PL NZ NA PO NC
073F:0108 0000          ADD     [BX+SI],AL                      DS:18F5=00
```

**Direccionamiento Inmediato**

```
-a
073F:0108 mov cx,89b5
073F:010B
-t

AX=18F5  BX=18F5  CX=89B5  DX=0000  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=010B   NV UP EI PL NZ NA PO NC
073F:010B 0000          ADD     [BX+SI],AL                      DS:18F5=00
```

**Direccionamiento Directo**

```
-a
073F:010B mov byte ptr[1234], al
073F:010E mov dl, [1234]
073F:0112
-t

AX=18F5  BX=18F5  CX=89B5  DX=0000  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=010E   NV UP EI PL NZ NA PO NC
073F:010E 8A163412      MOV     DL,[1234]                       DS:1234=F5
-t

AX=18F5  BX=18F5  CX=89B5  DX=00F5  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=0112   NV UP EI PL NZ NA PO NC
073F:0112 46            INC     SI
-
```

**Direccionamiento de Registro Indirecto**

```
-a
073F:0112 mov word ptr[BX], FFFF
073F:0116 mov ax, [bx]
073F:0118
-t

AX=18F5  BX=18F5  CX=89B5  DX=00F5  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=0116   NV UP EI PL NZ NA PO NC
073F:0116 8B07          MOV     AX,[BX]                         DS:18F5=FFFF
-t

AX=FFFF  BX=18F5  CX=89B5  DX=00F5  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=0118   NV UP EI PL NZ NA PO NC
073F:0118 B210          MOV     DL,10
-
```

**Direccionamiento Base mas Índice**

```
-a
073F:0118 mov si,7
073F:011B mov dx, [bx+si]
073F:011D
-t

AX=FFFF  BX=18F5  CX=89B5  DX=00F5  SP=00FD  BP=0000  SI=0007  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=011B   NV UP EI PL NZ NA PO NC
073F:011B 8B10          MOV     DX,[BX+SI]                      DS:18FC=0000
-t

AX=FFFF  BX=18F5  CX=89B5  DX=0000  SP=00FD  BP=0000  SI=0007  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=011D   NV UP EI PL NZ NA PO NC
073F:011D 07            POP     ES
```

**Direccionamiento Relativo a Registro**

```
-a
073F:011D mov al, [bx+8]
073F:0120
-t

AX=FF00  BX=18F5  CX=89B5  DX=0000  SP=00FD  BP=0000  SI=0007  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=0120   NV UP EI PL NZ NA PO NC
073F:0120 0000          ADD     [BX+SI],AL                      DS:18FC=00
```

**Direccionamiento Relativo a Base más Índice**

```
-a
073F:0128 mov ax, [bx+si-4]
073F:012B
-t

AX=0000  BX=18F5  CX=89B5  DX=0000  SP=00FD  BP=0000  SI=0007  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=012B   NV UP EI PL NZ NA PO NC
073F:012B 0000          ADD     [BX+SI],AL                      DS:18FC=00
```

PARTE 3.

```
-a
073F:012B push bx
073F:012C pop ax
073F:012D pushf
073F:012E popf dx
              ^ Error
073F:012F popf
073F:0130 xchg ax, cx
073F:0132 xlat
073F:0133 in ax,dx
073F:0134 out 05,al
073F:0136
```

Estado inicial de los registros: Movimiento de datos

```
AX=0000  BX=18F5  CX=89B5  DX=0000  SP=00FD  BP=0000  SI=0007  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=012B   NV UP EI PL NZ NA PO NC
073F:012B 0000          ADD     [BX+SI],AL                          DS:18FC=00
```

**PUSH BX:** se inserta el valor de BX a la pila. SP decrementa 2 unidades. Las banderas no se modifican.

```
AX=0000  BX=18F5  CX=89B5  DX=0000  SP=00FB  BP=0000  SI=0007  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=012C   NV UP EI PL NZ NA PO NC
073F:012C 58            POP     AX
```

**POP AX**: se  retira el valor de la pila y ahora se puede notar que ahora los registros AX y BX tienen el mismo valor. El registro SP se aumenta en 2. Las banderas no se modifican

```
AX=18F5  BX=18F5  CX=89B5  DX=0000  SP=00FD  BP=0000  SI=0007  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=012D   NV UP EI PL NZ NA PO NC
073F:012D 9C            PUSHF
```

**PUSHF:** Se inserta los valores del registro de banderas en la pila.

```
AX=18F5  BX=18F5  CX=89B5  DX=0000  SP=00FB  BP=0000  SI=0007  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=012E   NV UP EI PL NZ NA PO NC
073F:012E 9D            POPF
```

**POPF:** se retiran los valores de las banderas que se encuentran en la pila. El valor de SP se incrementa en tres. No se modifican las  banderas.

```
AX=18F5  BX=18F5  CX=89B5  DX=0000  SP=00FF  BP=0000  SI=0007  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=0130   NV UP DI PL NZ NA PO NC
073F:0130 87C1          XCHG    AX,CX
```

**XCHG AX, CX:** Se intercambian los valores de los registros AX y CX. No se modifican las banderas.

```
AX=89B5  BX=18F5  CX=18F5  DX=0000  SP=00FF  BP=0000  SI=0007  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=0132   NV UP DI PL NZ NA PO NC
073F:0132 D7            XLAT
```

**XLAT:** En AL se pasa el valor de la dirección que apunta [BX+AL], lo cual como todo el valor de memoria se inicializan en 0, en AL se guarda un 00. Los valores de las banderas no se modifican.

```
AX=8900  BX=18F5  CX=18F5  DX=0000  SP=00FF  BP=0000  SI=0007  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=0133   NV UP DI PL NZ NA PO NC
073F:0133 ED            IN      AX,DX
```

**IN AX, DX:** El valor que se encuentra en DX se copia en AX. Los valores de las banderas no se modifican.

```
AX=0000  BX=18F5  CX=18F5  DX=0000  SP=00FF  BP=0000  SI=0007  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=0134   NV UP DI PL NZ NA PO NC
073F:0134 E605          OUT     05,AL
```

**OUT 05, AL:** En la dirección 05 se copia el valor de AL. Esta instrucción sirve para enviar datos mediante un numero de bus.

```
AX=0000  BX=18F5  CX=18F5  DX=0000  SP=00FF  BP=0000  SI=0007  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=0136   NV UP DI PL NZ NA PO NC
073F:0136 0000          ADD     [BX+SI],AL                      DS:18FC=00
```

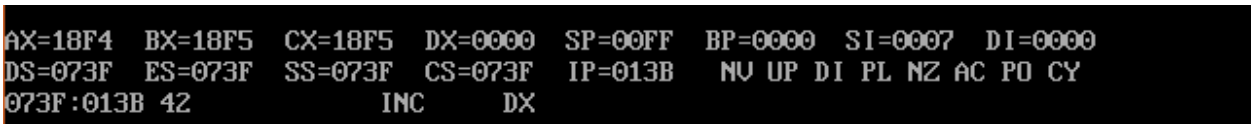
Estado inicial de los registros: Instrucciones de datos

```
073F:0136 add ax, ffff
073F:0139 adc ax, bx
073F:013B inc dx
073F:013C sub bx, dx
073F:013E sbb bx, bx
073F:0140 dec ax
073F:0141 mul bl
073F:0143 imul cl
073F:0145 div bx
073F:0147 idiv dx
073F:0149
```

```
AX=0000  BX=18F5  CX=18F5  DX=0000  SP=00FF  BP=0000  SI=0007  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=0136   NV UP DI PL NZ NA PO NC
073F:0136 0000          ADD     [BX+SI],AL                      DS:18FC=00
```

**ADD AX, FFFF**: la instrucción realiza una suma en el registro AX, sumándole FFFF.

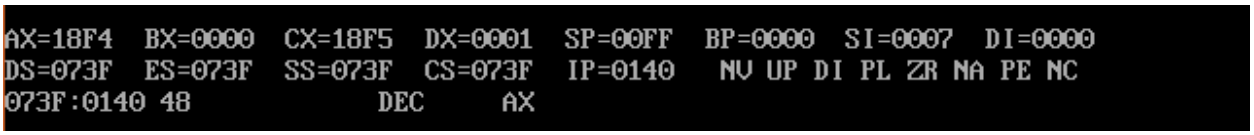

**ADC AX, BX:** es similar a la suma solo que la instrucción después de haber sumado BX a AX, también se le suma el valor de la bandera de acarreo. Las banderas que se modifican son las de acarreo, paridad, acarreo auxiliar y signo.



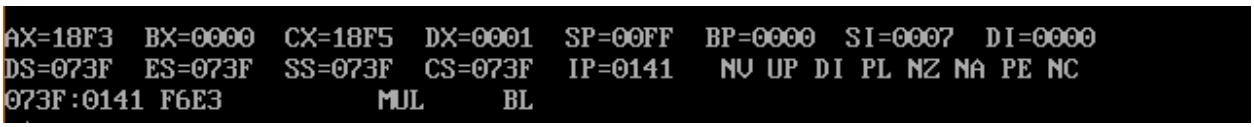

**INC DX:** suma un 1 al registro DX, en este caso no se modifican las banderas porque no ocurre un overflow o algún acarreo, pero si puede cambiarlo en su debido momento.


**SUB BX, DX:** se resta DX al registro BX. Se modifica la bandera de acarreo.


**SBB BX, DX:** es una resta, solamente que al final se le resta el valor de la bandera de acarreo.

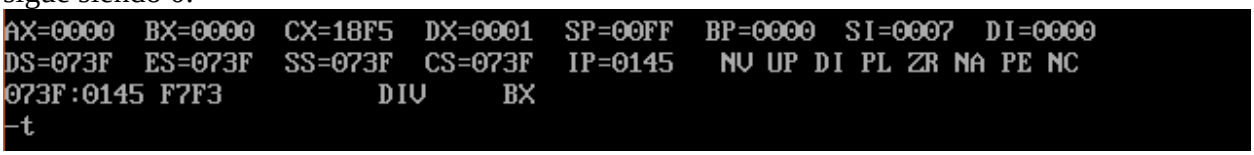



**DEC AX:** le resta un 1 al registro AX.

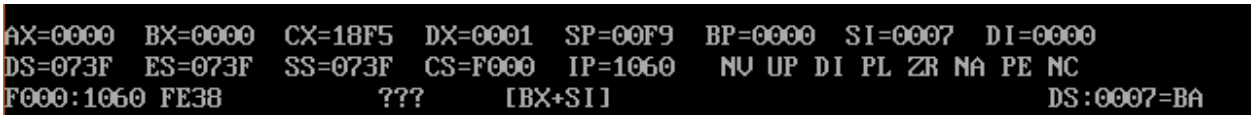



**MUL BL:** multiplica el valor de AL con el valor de BL. El resultado de guarda en AX. Y como el resultado es 0 porque en el registro BL es 0, la bandera de zero se activa.

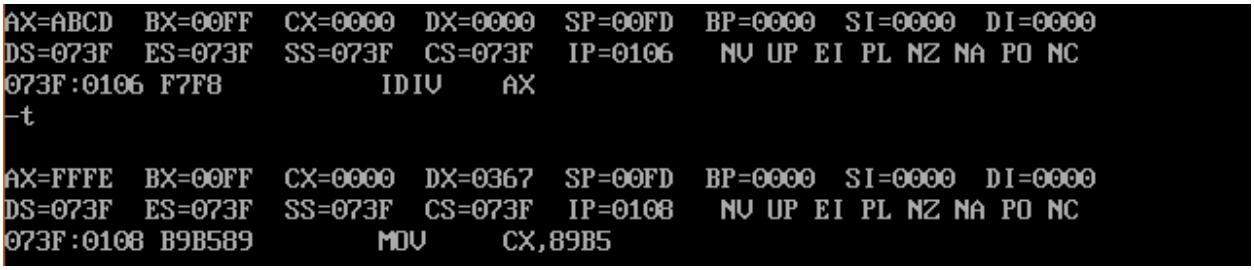
**IMUL CL:** realiza una multiplicación con signo. En este caso se multiplica el registro CL con AL, el resultado se almacena en AX. La bandera Zero se queda encendida porque el resultado sigue siendo 0.



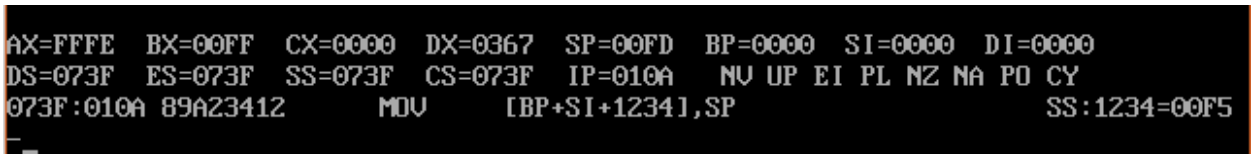

**DIV BX:** realiza una división entre los registros BX y AX. El resultado de almacena en AX-DX. En AX se almacena el cociente y en DX el residuo.




**IDIV BX:** realiza una división con signo, al igual que la instrucción DIV el resultado de almacena en AX-DX.




**CMP BX, AX:** compara dos registros. En realidad, la instrucción realiza una resta entre amabas por los que la bandera de acarreo se modifica.




**CBW:** Convierte un dato de tamaño byte (8 bits) a una de tamaño palabra (16 bits), el signo se conserva de AL y se extiende a AH

Antes de la instrucción CWB

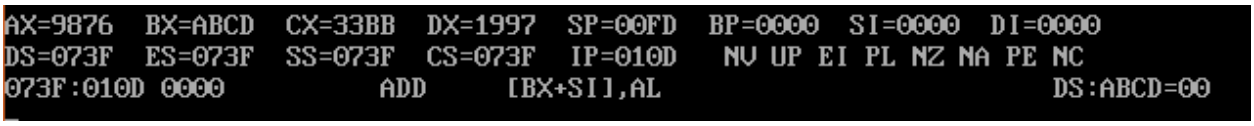

Después de la instrucción CWB

**CWD:** la instrucción convierte un dato de tamaño Word (16 bits) en un de double Word(32 bits) haciendo la extensión de signo en DX. Se puede observar en las siguientes dos imágenes como cambia el valor de DX.

Antes de la instrucción CWD

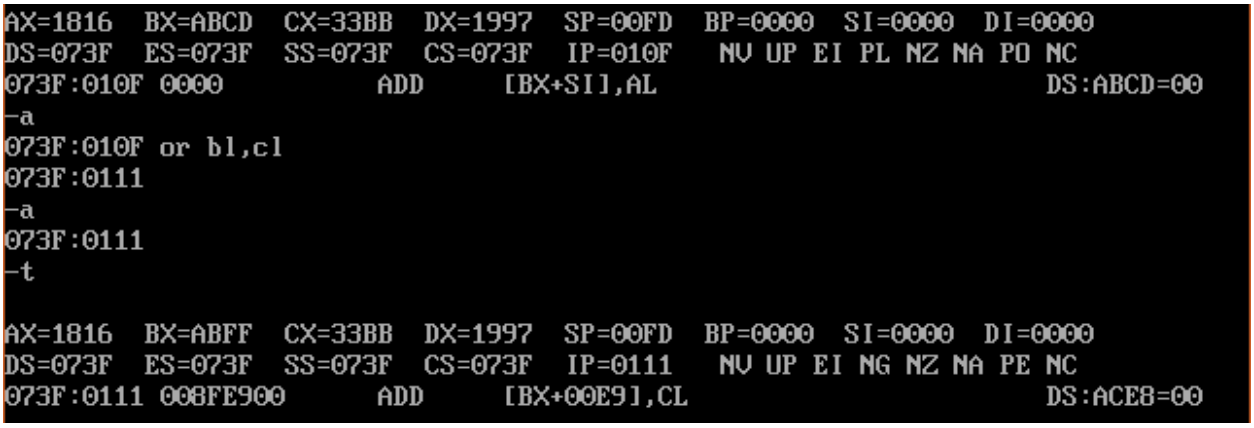

Después de la instrucción CWD


Instrucciones lógicas y de manipulación de bits.
**AND AX, BX:** realiza una operación AND entre los registros AX y BX, el resultado de almacena en el primer operando. La bandera de paridad se modifica pasando de encendida a apagada.





**OR BL, CL:** realiza la operación OR entre los registros BL y CL. La bandera de paridad se modifica y ahora pasa a estar encendida al igual que la bandera de signo que igual se enciende.



**XOR AL, AB:** la instrucción realiza una operación XOR en el registro AL, la su valor se altera.

```
AX=1816  BX=ABFF  CX=33BB  DX=1997  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=0111   NV UP EI NG NZ NA PE NC
073F:0111 008FE900      ADD     [BX+00E9],CL                      DS:ACE8=00
-a
073F:0111 xor al,ab
073F:0113
-t

AX=18BD  BX=ABFF  CX=33BB  DX=1997  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=0113   NV UP EI NG NZ NA PE NC
073F:0113 E900F0        JMP     F116
```

**NOT BP:** la instrucción se realiza una operación NOT la cual invierte todos los bits de un registro en este caso los de BP.

```
073F:0113 mov bp, ffff
073F:0116 not bp
073F:0118
-t

AX=18BD  BX=ABFF  CX=33BB  DX=1997  SP=00FD  BP=FFFF  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=0116   NV UP EI NG NZ NA PE NC
073F:0116 F7D5          NOT     BP
-t

AX=18BD  BX=ABFF  CX=33BB  DX=1997  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=0118   NV UP EI NG NZ NA PE NC
073F:0118 0400          ADD     AL,00
```

**TEST AX, BX:** la instrucción test realiza una operación AND entre los dos registros, pero no modifica su valor. Lo único que se modifica son las banderas. En este caso paso de ser a negativo a positivo.

```
AX=18BD  BX=ABFF  CX=33BB  DX=1997  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=0118   NV UP EI NG NZ NA PE NC
073F:0118 0400          ADD     AL,00
-a
073F:0118 test ax, bx
073F:011A
-t

AX=18BD  BX=ABFF  CX=33BB  DX=1997  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=011A   NV UP EI PL NZ NA PE NC
073F:011A 8700          XCHG    AX,[BX+SI]                        DS:ABFF=0000
```

**SHL DX, CL:** la instrucción realiza un desplazamiento a la izquierda. El numero de desplazamiento se encuentra en CL. La bandera de acarreo se activa

```
073F:0122 mov cl,2
073F:0124 mov dx, ffff
073F:0127 shl dx,cl
073F:0129
-t

AX=18BD  BX=ABFF  CX=0002  DX=1997  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=011D   NV UP EI PL NZ NA PE NC
073F:011D BAFFFF        MOV     DX,FFFF
-t

AX=18BD  BX=ABFF  CX=0002  DX=FFFF  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=0120   NV UP EI PL NZ NA PE NC
073F:0120 D3E2          SHL     DX,CL
-t

AX=18BD  BX=ABFF  CX=0002  DX=FFFC  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=0122   NV UP EI NG NZ AC PE CY
073F:0122 B102          MOV     CL,02
```

**SHR AX, CL:** la instrucción realiza un desplazamiento a la derecha de 3 veces. La bandera se acarreo es activada, la bandera se zero se activa.

```
AX=7854  BX=FCFC  CX=FA03  DX=1B9F  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=0115   NV UP EI PL ZR AC PE NC
073F:0115 00B200B2      ADD     [BP+SI+B200],DH                 SS:B200=00
-a
073F:0115 shr ax, cl
073F:0117
-t

AX=0F0A  BX=FCFC  CX=FA03  DX=1B9F  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=0117   NV UP EI PL NZ AC PE CY
073F:0117 00B21499      ADD     [BP+SI+9914],DH                 SS:9914=00
```

**ROL BX, CL:** la instrucción realiza un corrimiento a la izquierda. El numero de rotaciones esta dado por el registro CL, el cual tiene valor de 2. No se modificó ninguna bandera.

```
AX=0F0A  BX=FCFC  CX=FA02  DX=1B9F  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=0119   NV UP EI PL NZ AC PE CY
073F:0119 1499          ADC     AL,99
-a
073F:0119 rol bx, cl
073F:011B
-t

AX=0F0A  BX=F3F3  CX=FA02  DX=1B9F  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=011B   NV UP EI PL NZ AC PE CY
073F:011B 002E072E      ADD     [2E07],CH                       DS:2E07=00
```

**RCL DX, CL:** la instrucción realiza una rotación a la izquierda con acarreo. El bit que sale de DX se inserta en la bandera de acarreo, y el bit de la bandera de acarreo se inserta en la parte menos significativa de DX.

```
AX=C3F5  BX=F3F3  CX=FA02  DX=1B9F  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=0121   NV UP EI PL NZ AC PE CY
073F:0121 E2B1          LOOP    00D4
-a
073F:0121      rcl dx, cl
073F:0123
-t

AX=C3F5  BX=F3F3  CX=FA02  DX=6E7E  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=0123   NV UP EI PL NZ AC PE NC
073F:0123 02BAFFFF      ADD     BH,[BP+SI+FFFF]                  SS:FFFF=00
```

**ROR AX, CL:** Realiza una rotación hacia la derecha. El bit de signo se conserva.

```
AX=0FD7  BX=F3F3  CX=FA02  DX=1B9F  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=011F   NV UP EI PL NZ AC PE CY
073F:011F 07            POP     ES
-a
073F:011F ror ax,cl
073F:0121
-t

AX=C3F5  BX=F3F3  CX=FA02  DX=1B9F  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=0121   NV UP EI PL NZ AC PE CY
073F:0121 E2B1          LOOP    00D4
```

**RCR AL, 1:** la instrucción realiza una rotación a la derecha con acarreo. No se modificó ni una bandera.

```
AX=0FAF  BX=F3F3  CX=FA02  DX=1B9F  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=011D   NV UP EI PL NZ AC PE CY
073F:011D 07            POP     ES
-a
073F:011D rcr al, 1
073F:011F
-t

AX=0FD7  BX=F3F3  CX=FA02  DX=1B9F  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=011F   NV UP EI PL NZ AC PE CY
073F:011F 07            POP     ES
```

## Conclusión

Con la realización de la siguiente práctica, se solidificaron las bases de las instrucciones del procesador 8086 así como los modos de direccionamiento que estos tienen. Una de las complicaciones que se tuvo al concluir esta práctica, fue el identificar que pasaba en la instrucción y porque modificaba ciertas banderas. Pero gracias a esto ya se tiene mejor conocimiento de lo que realmente hace cada instrucción del procesador.

## Bibliografía

http://www.eeeguide.com/8086-shift-instructions/
https://www.felixcloutier.com/x86/xlat:xlatb
http://www.c-jump.com/CIS77/MLabs/M11arithmetic/M11_0070_imul_example.htm
http://www.c-jump.com/CIS77/MLabs/M11arithmetic/M11_0110_cbw_cwd_cdq.htm
https://c9x.me/x86/html/file_module_x86_id_27.html