INGENIERÍA COMPUTACIONAL II

CONJUNTO DE INSTRUCCIONES

Introducción.

Uno de los aspectos más interesantes y más estudiados dentro del diseño de una computadora es el diseño del **Conjunto de Instrucciones**. Este diseño es uno de los más complejos, ya que afecta varios aspectos de una computadora. El Conjunto de Instrucciones define muchas de las funciones ejecutadas por el CPU, por lo tanto, los requerimientos del programador deben de considerarse en el diseño.

Dentro de las características a considerar en el diseño encontramos:

- **Repertorio de Operaciones:** Cuántas y cuáles operaciones se deben proporcionar. Qué tan complejas deben ser.
- **Tipo de datos:** Se refiere a los diferentes tipos de datos que manejan las operaciones.
- **Formato de la instrucción:** Longitud de la instrucción, número de direccionamientos, tamaño de los campos, etc.
- **Registros:** Número de registros del CPU que pueden referenciarse por las instrucciones, y su uso.
- **Direccionamiento:** El modo o modos en que la dirección de un operando se especifica.

El 8088 tiene la capacidad de sumar, restar, multiplicar, dividir, y comparar dos valores. Puede transferir datos de un lugar a otro, puede recibir datos de dispositivos externos y enviar datos a dispositivos externos. Además, el orden en que ejecuta estas funciones puede modificarse.

Existen seis grupos de códigos de operación en el conjunto del 8088. Estos son:

1. Transferencia de datos
2. Aritméticos
3. Manipulación de bits
4. Control de programa
5. Manipulación de strings
6. Control de CPU

Los mnemónicos de cada grupo son los siguientes:

1. Instrucciones de Transferencia de Datos

MOV	Mover
PUSH, POP	Operaciones del Stack
XCHG	Intercambiar
IN, OUT	Puertos de Entrada / Salida

2. Instrucciones Aritméticas

ADD	Sumar
INC	Incrementar
SUB	Restar
DEC	Decrementar
NEG	Negar
CMP	Comparar
MUL	Multiplicar
DIV	Dividir

3. Instrucciones Lógicas

NOT	Complementar
AND	And
OR	Or Inclusivo
XOR	OR Exclusivo
TEST	Examinar bits
SHL, SHR	Corrimiento izquierda / derecha
ROL, ROR	Rotación izquierda / derecha

4. Instrucciones de Manipulación de Strings

MOVS	Mover strings
CMPS	Comparar strings
SCAS	Recorrer string
LODS	Leer del string
STOS	Guardar en el string

5. Instrucciones de Transferencia de Control

CALL	Ligar a una subrutina
RET	Regresar de una subrutina
JMP	Salto
JZ, JNZ ...	Saltos Condicionales
LOOP	Ciclo
LOOPNE, ...	Ciclos Condicionales
INT	Interrupción

IRET Regreso de la interrupción

6. Instrucciones de Control del Procesador

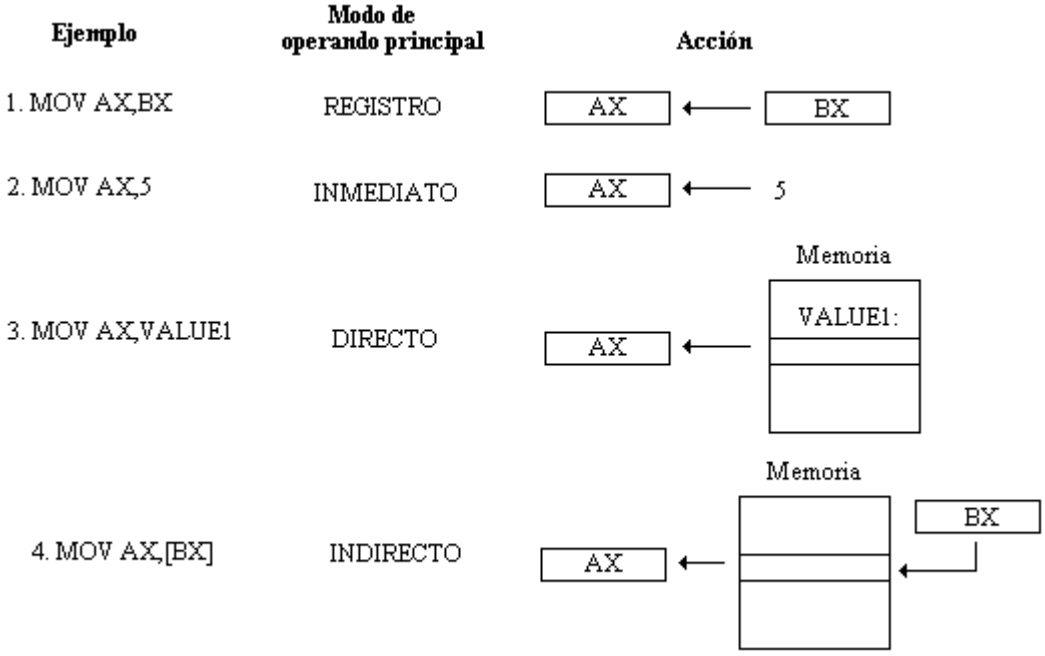
CLC, STC ... Inicializar, Inicializar banderas
HLT Detener Procesador

Instrucciones de Transferencia de datos

Las **Instrucciones de Transferencia de Datos** nos permiten *mover* datos entre registros y memoria. El mover datos puede parecer una instrucción sencilla, sin embargo se complica porque contamos con diferentes *modos* de direccionamiento de datos.

Modos de direccionamiento de Datos

figura 1.



En la figura 1 podemos ver una serie de instrucciones, cada una de las cuales selecciona una palabra de datos y la mueve al registro AX. Estas instrucciones ilustran **algunos** modos de direccionamiento del 8088.

La instrucción MOV

MOV es una instrucción de propósito general que se utiliza para transferir datos entre registros y memoria. La sintáxis en ensamblador es la siguiente:

MOV destino,fuente

Donde *destino* es un **registro** o una **localidad de memoria**.

Y *fuente* es el **valor de un registro**, el **valor en una dirección** (operando de memoria) o un **valor especificado en el código de operación** (un valor inmediato).

El operando fuente se copia en el destino; es decir, el valor de éste último se actualiza con el nuevo valor y el fuente permanece sin cambio.

La instrucción MOV permite mover cualquier operando de memoria y cualquier registro excepto en los siguientes casos:

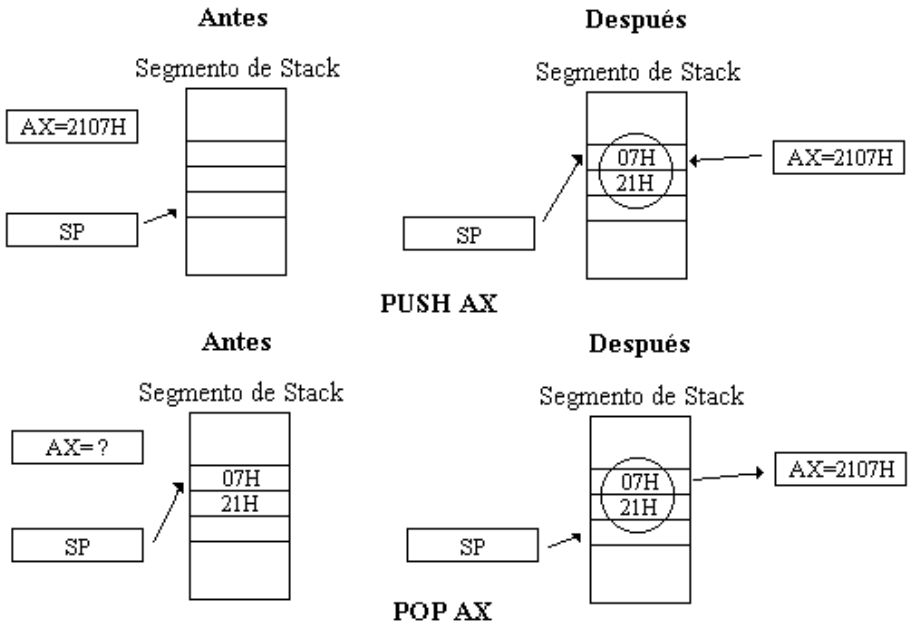
- Mover un operando de memoria a otro operando de memoria
- Mover un operando fuente de 16 bits a un operando destino de 8 bits
- Mover un operando fuente de 8 bits a un operando destino de 16 bits
- Un valor inmediato a un registro de segmento
- Cualquier valor (registro, memoria, inmediato) a los registros CS y PC (IP).

Las instrucciones PUSH y POP

Un tipo especial de Instrucciones de Transferencia de Datos son las *operaciones del stack*: PUSH y POP, que implementan una estructura LIFO (Last-in, First-out) en memoria, conocida como *stack*. Estas operaciones *guardan* y *recuperan* 16 bits de datos del stack.

En la figura 2 se ilustra la operación de estas instrucciones.

figura 2.



El stack siempre lo encontramos en el *segmento de stack* y, el registro denominado SS siempre se utiliza durante las referencias al stack; no se permite el encimamiento con otro

segmento. El registro de *apuntador al stack (SP)* se utiliza implícitamente en todas las operaciones al stack. La instrucción PUSH decrementa el contenido del SP en 2 posiciones y después guarda el operando en la dirección de memoria especificada por SP. La instrucción POP busca los datos en la dirección de memoria especificada por el SP, los guarda en el operando, y después incrementa el contenido de SP en dos posiciones. Cabe notar que las instrucciones PUSH y POP siempre transfieren una **palabra** (2 bytes) de datos, en contraste con la instrucción MOV que puede manipular un byte o una palabra.

Las instrucciones PUSH y POP proporcionan un mecanismo de almacenamiento que puede utilizarse para almacenar diferentes datos y después volverlos a obtener en el orden inverso en el que se guardaron. Esta característica hace que el stack se utilice como un almacenamiento temporal ideal cuando no deseamos utilizar una localidad específica de memoria. Por ejemplo, supongamos que estamos escribiendo un programa y parte de él utiliza y altera el valor del registro CX. Si necesitamos utilizar este valor después, podemos almacenarlo en el stack de la siguiente forma:

```
PUSH        CX          ;Guardar valor en el stack
...
...
...                     ;Sección del programa que altera el registro CX
...
...
POP         CX          ;Obtener el valor original
```

Los operandos de las instrucciones PUSH y POP pueden direccionarse utilizando cualquier modo de direccionamiento de datos (algunos de ellos están en la figura 1). Además, el contenido del *registro de banderas* puede guardarse y volverse a obtener en el stack con las instrucciones **PUSHF** y **POPF** (Estas son las únicas instrucciones que permiten guardar y recuperar el valor de las banderas. No se permite moverlas directamente a otro registro. Por lo tanto, PUSHF y POPF no tienen operandos).

Debido a su naturaleza LIFO, el stack se utiliza por las instrucciones de subrutinas: CALL y RETurn, así como por el mecanismo de *interrupciones* del 8088.

La única instrucción que no está permitido realizar es: POP CS.

Instrucciones de transferencia de Entrada/Salida.

Las instrucciones IN y OUT se utilizan para accesar los puertos de entrada/salida (e/s) del 8088. Un puerto de e/s es una ruta entre el microprocesador y el mundo exterior. Los dispositivos externos como las terminales, impresoras y drives se comunican con el microprocesador por medio de puertos de e/s.

El 8088 tiene 65,536 (64k) puertos de e/s y, por lo tanto, necesita una palabra de 16-bits para poder direccionarlos. Cuando queremos comunicarnos con un puerto de e/s, tenemos

que especificar una dirección de 16-bits para seleccionarlo. Las direcciones van desde 0 hasta FFFFH.

Tanto un byte de 8-bits como una palabra de 16-bits puede transferirse a través de un puerto de e/s. En el 8088, la manipulación de datos (enviar o recibir) se realiza siempre en el registro acumulador AX. En los casos en que sólo se desee manipular bytes, se utiliza la parte baja del acumulador, AL. La dirección del puerto de e/s se especifica ya sea en el contenido del registro **DX** o en forma **inmediata**. Si utilizamos la forma **inmediata** pueden accesarse sólo de la dirección **0** a la **FFH** (los primeros 256 puertos de e/s). A continuación veremos algunos ejemplos: (Note que para las instrucciones IN el puerto de e/s es el operando *fuente* y el acumulador es el operando *destino*; para las instrucciones OUT es a la inversa).

```
IN      AL,2FH      ;Recibir un byte del puerto 2FH
OUT     5,AL        :Enviar un byte al puerto 5
MOV     DX,3FCH
IN      AX,DX       ;Recibir una palabra del puerto 3FCH
```

La instrucción XCHG

Esta instrucción *intercambia* el contenido de sus dos operandos. El segundo operando *siempre* debe ser un registro. El primer operando puede accesarse utilizando cualquiera de los modos de direccionamiento de datos (excepto el inmediato). Ejemplo:

```
MOV     AX,5        ; AX = 5
MOV     BX,10       ; BX = 10
XCHG    AX,BX       ; AX = 10, BX = 5
```

Instrucciones Aritméticas y el Registro de Banderas

La instrucciones aritméticas se utilizan para ejecutar operaciones matemáticas. Las instrucciones ADD, SUB y CMP cuentan con dos operandos. El primer operando *siempre* funciona como el *destino* del resultado. La instrucción:

```
ADD   AX,BX
```

Sumará el contenido de AX con el contenido de BX y el resultado se colocará en AX. En forma similar:

```
SUB   AX,BX
```

Restará el contenido de BX del contenido de AX y el resultado se colocará en AX. La instrucción de *comparación* (CMP) realiza la misma función que la instrucción SUB, sólo que ningún operando es actualizado, lo que modifica son algunas banderas.

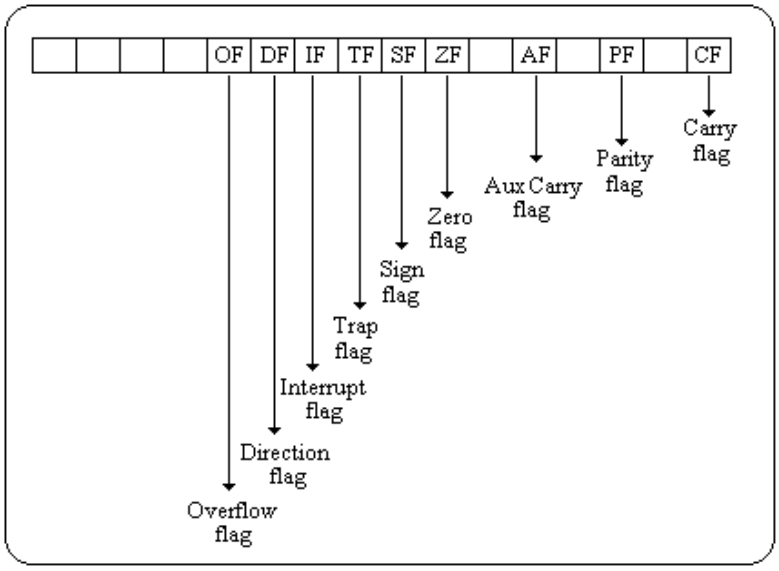
Las instrucciones INC, DEC y NEG utilizan solamente un operando. Éste puede direccionarse utilizando cualquiera de los modos de direccionamiento excepto el *inmediato*. INC incrementa en **uno** al operando. DEC decrementa en **uno** al operando y NEG obtiene el **complemento a dos** del operando.

Todas las instrucciones aritméticas, además, actualizan varios bits en el registro de banderas para indicar el estado del resultado. Las instrucciones de *transferencia condicional* pueden utilizarse para examinar estos bits y cambiar el flujo del programa. Casi todas las decisiones tomadas por el 8088 dependen de los valores del registro de banderas.

Registro de Banderas

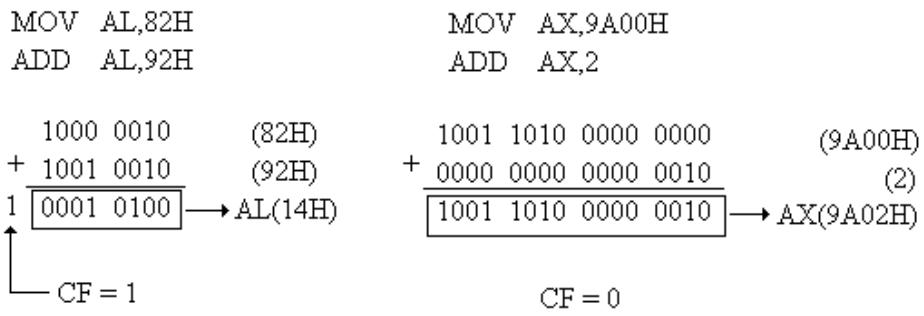
La figura 3 muestra detalladamente el registro de banderas de 16-bits del 8088.

figura 3.



Nueve de los bits de este registro tienen un uso especial y reciben cierto nombre, el resto **no** se utiliza. El bit de *acarreo* se encuentra en el bit 0 del registro de banderas. Después de una operación aritmética, la bandera de acarreo (CF) se pondrá a uno (1) si existe un acarreo de salida del bit más significativo del resultado; se pondrá a cero (0) si no existe acarreo de salida. Esto se ilustra en la figura 4.

figura 4.



De forma similar, la bandera de cero (ZF), se pondrá a uno (1) si el resultado de la operación es cero; de lo contrario tendrá un valor de 0. La bandera de signo (SF), se pondrá a uno (1) si el resultado es negativo: de lo contrario tendrá un valor de 0. La bandera de overflow (OF) se pondrá a uno (1) si un resultado **con signo** no puede respresentarse en el operando destino.

Las instrucciones MUL y DIV

Las instrucciones de multiplicación y división constituyen funciones avanzadas del microprocesador y son menos flexibles para utilizar que el resto de las instrucciones aritméticas. El *destino* es siempre el acumulador (AX) y sólo se codifica el operando *fuente* en la instrucción. Es importante saber que cuando multiplicamos byte por byte, el resultado puede ser tan grande como una palabra (2 bytes). Así, en una multiplicación de bytes, el operando *fuente* (un byte) se multiplica por el contenido de AL y el resultado se guarda en AX. En el caso de una multiplicación de palabras, el operando *fuente* (una palabra) se multiplica por el contenido de AX. Si el resultado ocupa hasta 32 bits, la parte menos significativa se guarda en AX y la más significativa en DX.

En cambio, en una división de bytes, el numerador (AX), es dividido por un operando *fuente* de un byte. El resultado se coloca en AL y el residuo en AH. Una operación de división de una palabra asume un numerador de 32-bits (la parte alta en DX y la baja en AX). Este numerador es dividido por el operando *fuente* de una palabra. El resultado se guarda en AX y el residuo en DX.

A continuación se ilustran ejemplos del uso de MUL y DIV:

```
DIV   BL        ; 8-bit.    AL = int(AX/BL),     AH=residuo
DIV   BX        ; 16-bit.   AX = int(DX:AX/BX), DX=residuo
MUL   BL        ; 8-bit.    AX = AL*BL
MUL   BX        ; 16-bit.   DX:AX = AX*BX
```

Nota: El operando *fuente* no puede ser un valor *inmediato*.

Instrucciones Lógicas

Las instrucciones aritméticas asumen siempre que sus operandos representan información numérica. En contraste, las instrucciones *lógicas* asumen que sus operandos son simples *cadenas de bits*.

La instrucción NOT

Esta instrucción simplemente invierte cada bit de un operando, cambiando los ceros por unos y viceversa.

Las instrucciones AND, OR, XOR

Cada una utiliza dos operandos. El resultado de una instrucción AND coloca bits con valor de uno (1) sólo si las dos posiciones de los operandos son, *ambas*, unos. Esta instrucción se utiliza para *enmascarar* o para forzar a *cero* algunos bits seleccionados del operando *destino*. El resultado de una instrucción OR coloca bits con valor de uno (1) donde *alguna* de las posiciones de los operandos tenga un valor de uno. Esta instrucción se utiliza para forzar a *uno* algunos bits seleccionados del operando *destino*. El resultado de una operación XOR coloca bits con valor de uno (1) en aquellas posiciones donde *alguna pero no ambas* posiciones de los operandos tengan un valor de uno. Esta instrucción se utiliza para *invertir* algunos bits seleccionados del operando *destino*.

A continuación, veremos ejemplos del uso de estas instrucciones:

AND Coloca en **cero** los bits del operando **destino**
donde existe un bit con valor de cero en el operando **fuente**

```
MOV AL,0BFH
AND AL,0FCH

     1011 1111      (BFH)
AND  1111 1100      (FCH)
-------------------
     1011 1100  --> AL (BCH)
```

OR Coloca en **uno** los bits del operando **destino**
donde existe un bit con valor de uno en el operando **fuente**

```
MOV AL,43H
OR  AL,20H

     0100 0011      (43H)
OR   0010 0000      (20H)
-------------------
     0110 0011  --> AL (63H)
```

XOR Invierte los bits en el operando **destino**
donde exista un bit con valor de uno en el operando **fuente**

```
MOV  AL,1
XOR  AL,3

     0000 0001      (1)
XOR  0000 0011      (3)
     ---------
     0000 0010 → AL (2)
```

La instrucción TEST

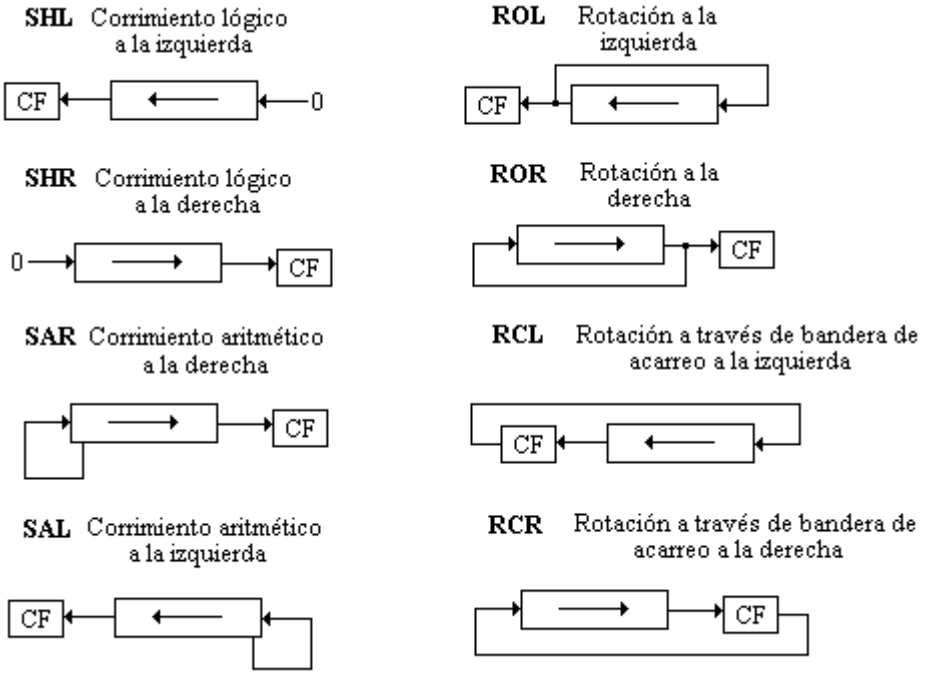
Ejecuta la misma función que la instrucción AND, excepto que el operando *destino* no se modifica. Se utiliza para verificar bits puestos a uno en un byte o palabra.

Todas las instrucciones lógicas vistas anteriormente colocan a *cero* las banderas de *acarreo* y *overflow*. Actualizan, además la bandera de *cero* para indicar cuándo el resultado es cero.

Las instrucciones SHL, SHR, ROL, ROR (shift y rotate)

Las instrucciones de corrimiento (shift) y rotación (rotate) mueven bits lateralmente (de izquierda a derecha o viceversa) dentro de sus operandos. La figura 5 ilustra lo anterior:

figura 5.



Como ejemplo, asumamos que el registro AL contiene el valor de 3. La instrucción SHL (*shift left logical*) tendrá el siguiente resultado:

AL antes = 03H = 0000 0011
Ejecutar SHL AL,1
AL después = 06H = 0000 0110

Cada bit en el operando destino se recorrió una posición a la izquierda. Un bit con valor de cero (0) se agregó a la derecha para llenar el bit en la posición *menos significativa*. Además, el bit *más significativo* fue recorrido hacia afuera del operando y se colocó en la *bandera de acarreo (CF)*. En el ejemplo anterior CF = 0.

La instrucción SHR (*shift right logical*) funciona igual que la instrucción SHL, excepto que los bits se mueven en la dirección opuesta. Estas instrucciones pueden utilizarse para desarrollar *multiplicaciones* y *divisiones* en potencias de dos. Si recorremos los bits de un valor numérico hacia la izquierda, multiplicamos ese valor por dos. Si recorremos los bits de un valor numérico hacia la derecha, dividimos el valor entre dos. Si el valor numérico es signado (representado en complemento a dos, por ejemplo), entonces la división debe ejecutarse utilizando la instrucción SAR (*shift right arithmetic*). Esta instrucción difiere de SHR en que retiene el bit más significativo (signo) en el operando destino.

Las instrucciones de corrimiento y rotación de la figura 5 requieren de un segundo operando que especifica *cuántas veces* se va a efectuar la operación. El valor de este segundo operando puede darse de forma inmediata o por medio del contenido del registro CL. Ejemplo:

```
MOV   AL,1
MOV   CL,3
SHL    AL,CL
```

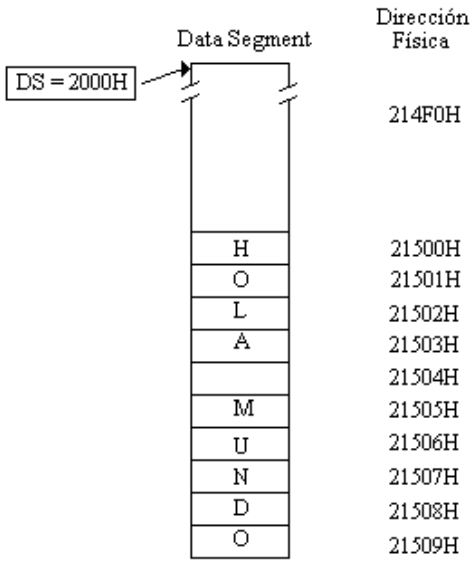
Que dará como resultado AL = 8.

Las instrucciones lógicas pueden operar con bytes o palabras y utilizar la mayor parte de los modos de direccionamiento de datos.

Instrucciones de Manipulación de Strings

Las instrucciones de manipulación de strings proporcionan al programador una gran capacidad de manipulación de cadenas de datos almacenadas en memoria. En la figura 6 se muestra como el string "HOLA MUNDO" puede almacenarse en memoria. Este string ocupa 10 bytes y se almacena en el *segmento de datos* empezando en la dirección de desplazamiento 1500H.

figura 6.



Las instrucciones básicas de strings (MOVS, CMPS, SCAS, LODS y STOS) pueden procesar o un byte o una palabra a la vez. Cuando utilizamos estas instrucciones, se debe colocar una B o una W al final del mnemónico para indicar el tamaño de dato deseado. Así, la instrucción **MOVSB** moverá los datos de un string un byte a la vez, y la instrucción **MOVSW** moverá los datos de un string una palabra (2 bytes) a la vez. Los operandos de las instrucciones de strings están ***implícitos***, de tal forma que no es necesario especificarlos cuando escribimos una instrucción. Los strings *fuente* se direccionan por el registro SI, y los strings *destino* se direccionan por el registro DI. Siempre se asume que el string destino se encuentra en el *segmento extra* y que el string fuente, generalmente, se encuentra en el *segmento de datos*, a menos que se indique una superposición con otro segmento.

En la figura 7 se ilustra el movimiento de un string de un segmento a otro. Los registros DS y SI definen en inicio del string fuente. Los registros ES y DI definen el lugar a donde se moverá el string. Al ejecutar la instrucción MOVSB el byte apuntado por SI del string fuente se copiará en el byte apuntado por DI en el string destino. SI y DI se actualizan automáticamente para apuntar el siguiente **byte**. La instrucción MOVSW actualiza SI y DI para apuntar a la siguiente **palabra**. Esta actualización de SI y DI hacia adelante se logra colocando en la bandera de dirección un valor de cero (0), por lo que la copia se realiza de forma "forward".

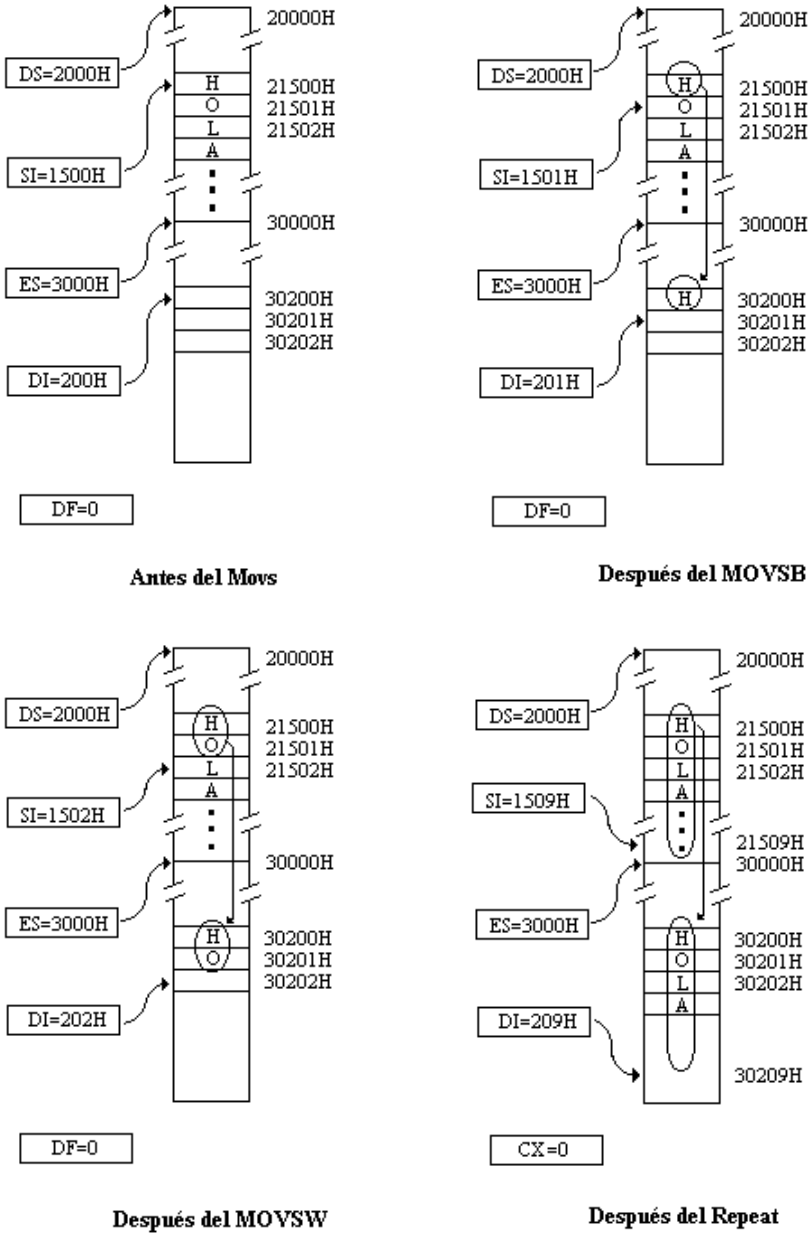
Esta facilidad de actualización hace particulamente fácil manejar grandes cadenas de caracteres (strings) repitiendo la instrucción de manipulación de strings varias veces. Por ejemplo, en CX colocamos el número de veces que desea ejecutarse la instrucción y posteriormente se hace referencia a una instrucción de repetición como se ilustra a continuación:

```
MOV  CX,10
REP  MOVSB
```

Nota: cada vez que se realiza la instrucción de manipulación de strings, el valor de CX se decrementa en uno. Cuando CX llega a cero, el procesador continúa con la siguiente instrucción. Así, CX se utiliza para especificar la longitud de los strings que se van a procesar.

Si la bandera de dirección DF tiene un valor de uno (1) entonces la copia se hace de forma "backward", es decir, SI y DI se actualizan decrementándose en un byte o en una palabra, según sea el caso.

figura 7.

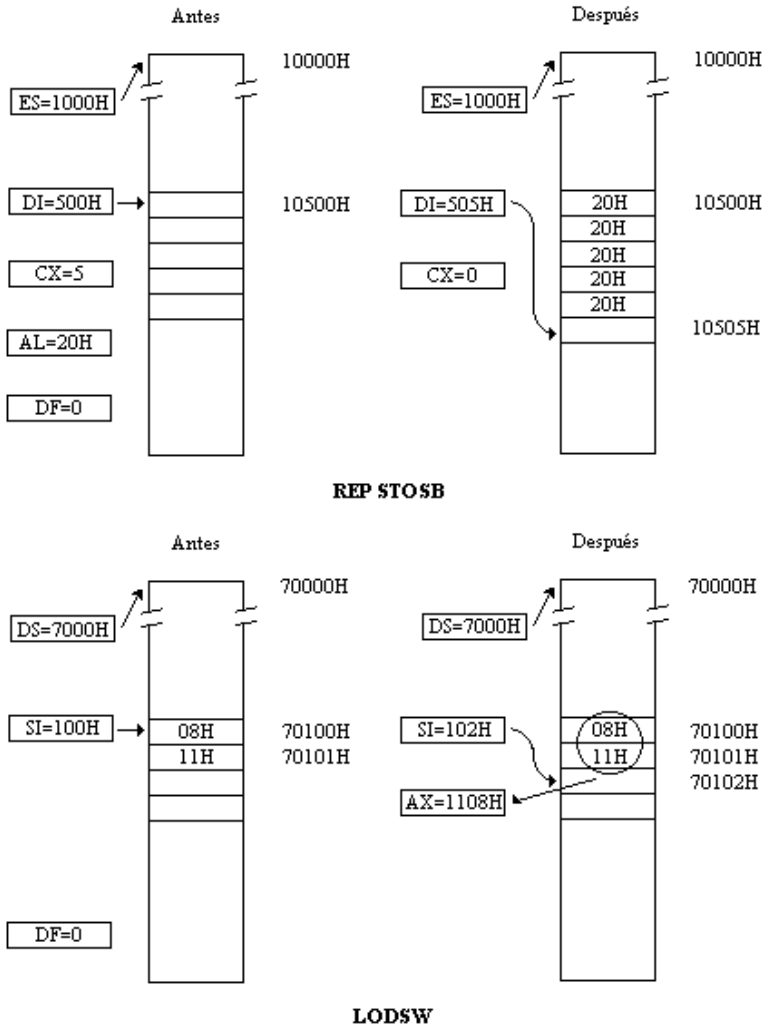


Las instrucciones **STOSB** y **STOSW** (store string) guardan el contenido del acumulador (AL en operaciones con bytes y AX en operaciones con palabras) en el string destino y posteriormente actualizan el registro DI. No se hace referencia alguna al registro SI, por lo que NO es modificado. Las instrucciones REP STOSB y REP STOSW son útiles para inicializar bloques de memoria a un valor constante (El string destino siempre está en el segmento extra).

Las instrucciones **LODSB** y **LODSW** buscan un byte o una palabra del *string fuente* y la colocan en AL o AX según sea el caso; posteriormente se actualiza SI. No se hace referencia alguna al registro DI, por lo que NO es modificado. Estas instrucciones son útiles para programas de traducción de lenguajes (parsers) cuando es necesario examinar cada elemento de un string, uno a la vez, en el orden en el que se encuentran.

La figura 8 ilustra el funcionamiento de STOS y LODS:

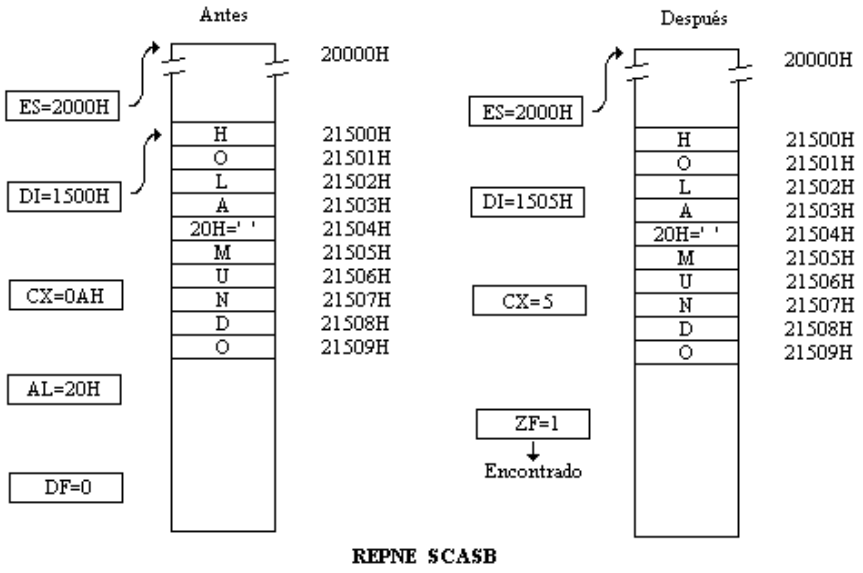
figura 8.



Las instrucciones **SCASB** y **SCASW** restan el byte o palabra del string destino del contenido de AL o AX y actualizan varias banderas para indicar el resultado de la comparación. Sólo se alteran las banderas y el registro DI. Estas instrucciones se pueden utilizar para buscar un determinado caracter dentro de un string. Para complementar estas instrucciones tenemos los prefijos REPE (repeat while equal) y REPNE (repeat while not equal). Estos prefijos hacen que la instrucción se repita el número de veces especificado en CX. Además, pueden detener la función de repetición cuando la bandera de cero, ZF, indica cuando la condición (REPE o REPNE) se ha alcanzado.

La figura 9 muestra cómo la repetición de la instrucción SCASB puede utilizarse para buscar un caracter específico dentro de un string. Generalmente actualizamos los registros ES y DI para apuntar al string donde se va a realizar la búsqueda. El caracter a buscar se coloca en el registro AL y la longitud del string en el registro CX. Después se ejecuta la instrucción REPNE SCASB. Si la instrucción termina con ZF = 0, entonces NO se encontró el caracter dentro del string. Se lo contrario, el registro DI apuntará a la posición inmediata posterior, dentro del string, donde se encontró el caracter, y CX tendrá el número de caracteres restantes del string.

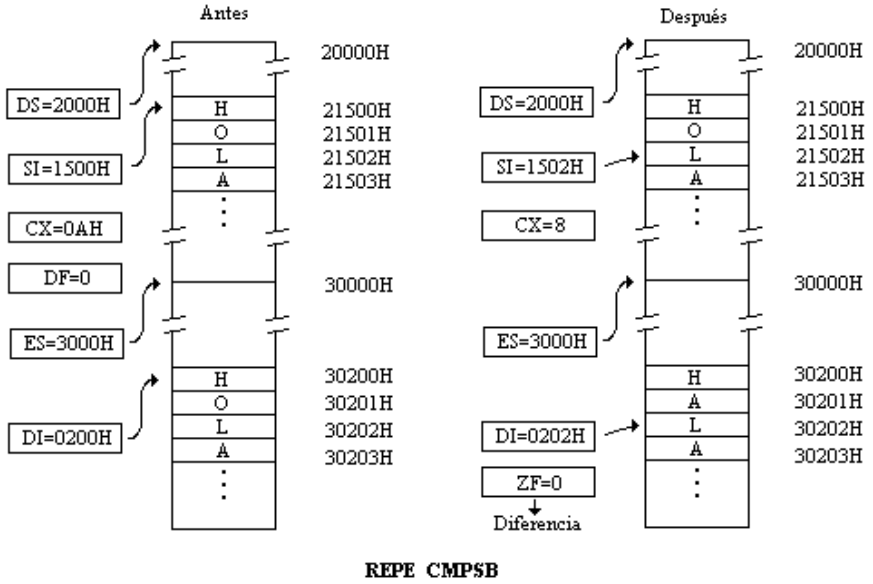
figura 9.



REPNE SCASB

Las instrucciones **CMPSB** y **CMPSW** restan el string *destino* del string *fuente* y actualizan algunas banderas para indicar el resultado de la comparación. Los operandos no se alteran, solamente las banderas y los registros SI y DI. Estas instrucciones también pueden utilizarse con el prefijo REPE para comparar dos strings hasta que se encuentre una *diferencia*, o con el prefijo REPNE para comparar dos strings hasta que se encuentre una *coincidencia*. La figura 10 muestra el uso de la instrucción CMPSB.

figura 10.



REPE CMPSB

Instrucciones de Transferencia de Control

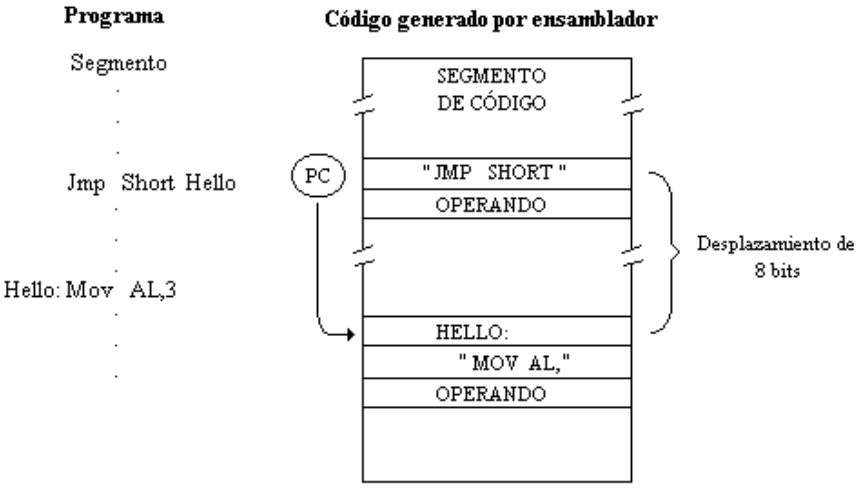
Estas instrucciones nos permiten alterar el flujo de la ejecución del programa. Como se ha mencionado, el PC contiene la dirección de la siguiente instrucción del programa que se va a ejecutar. En condiciones normales, el flujo del programa se desarrolla secuencialmente de una instrucción a la siguiente (incrementando el PC en cada ejecución de instrucción).

Existen dos formas de alterar el flujo del programa: alterando el contenido del PC (esto provoca que el programa continúe ejecutándose dentro del mismo segmento de código, pero en una dirección diferente [*transferencia intrasegmento*]) o alterando los registros PC y el CS simultáneamente (esto permite cambiar la ejecución del programa a cualquier dirección en un diferente segmento de código [*transferencia intersegmento*]).

La instrucción de salto incondicional (JMP)

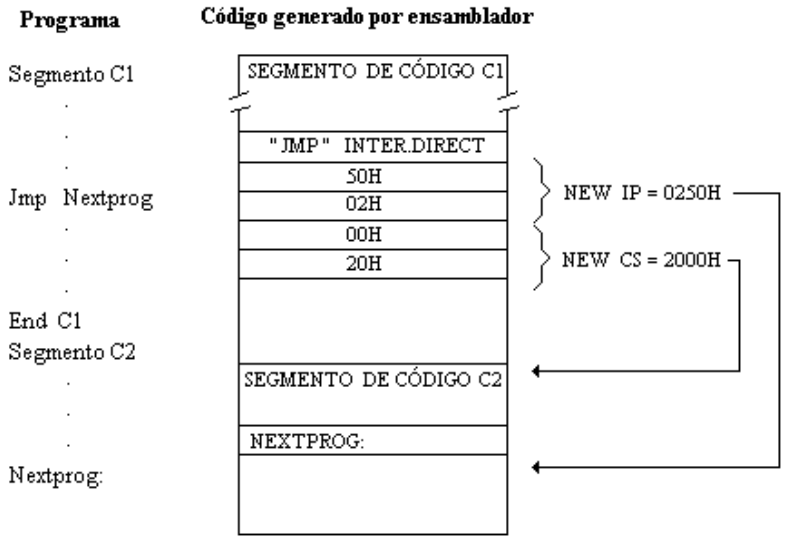
Esta instrucción provoca que el flujo del programa cambie sin considerar condición alguna. Consta de un sólo operando que especifica exáctamente dónde debe de continuar la ejecución del programa. Sin embargo, esta instrucción puede complicarse debido a los diferentes *modos de direccionamiento*. Existen dos modos de direccionamiento para el salto incondicional (JMP): *directo* e *indirecto*. El modo *directo* especifica el destino del salto dentro de la misma instrucción (etiqueta). El ensamblador calcula la diferencia entre la instrucción JMP y la dirección donde se encuentra la etiqueta. Este *desplazamiento* se utiliza como el operando de la instrucción JMP. La figura 11 nos muestra un salto *corto* (intrasegmento) en el que el operando de *desplazamiento* es de un byte. Como necesitamos saltar hacia adelante o hacia atrás, el desplazamiento se considera como un valor signado (complemento a dos). De esta forma y con un sólo byte, el salto del JMP puede estar a -128 o +127 posiciones de él.

figura 11.



Para realizar saltos mayores, se utiliza un JMP con un operando de 16 bits. Esto permite accesar cualquier instrucción dentro de los 64k del segmento de código.
Para accesar directamente una instrucción *fuera* del actual segmento de código, el ensamblador puede generar un JMP intersegmento. La figura 12 ilustra lo anterior. Este modo incluye un operando de cuatro bytes que especifica tanto el nuevo valor del CS (code segment) y una dirección dentro de este nuevo segmento (PC). Aquí no se utilizan desplazamientos sino direcciones *exactas*.

figura 12.



Instrucciones de salto Condicionales

Estas instrucciones nos permiten verificar ciertas condiciones antes de realizar un salto. Sólo se transfiere el control a la nueva dirección si y sólo si se cumplen las condiciones deseadas, de lo contrario el flujo del programa continúa de forma secuencial.

Las condiciones que se verifican se encuentran en el *registro de banderas* (cuyo contenido depende del resultado de operaciones aritméticas o lógicas). En la tabla siguiente se encuentran todos los saltos condicionales del 8088:

Instrucciones de Salto Condicionales

Mnemónico	Condición requerida	Interpretación (salta si...)
JE o JZ	ZF = 1	"igual" o "cero"
JL o JNGE	(SF xor OF) = 1	"menor" o "no mayor o igual"
JLE o JNG	((SF xor OF) or ZF) = 1	"menor o igual" o "no mayor"
JB o JNAE o JC	CF = 1	"abajo" o "no arriba o igual" o "acarreo"
JBE o JNA	(CF or ZF) = 1	"abajo o igual" o "no arriba"
JP o JPE	PF = 1	"paridad" o "paridad par"
JO	OF = 1	"overflow"
JS	SF = 1	"signo"
JNE o JNZ	ZF = 0	"no igual" o "no cero"

Instrucciones de Salto Condicionales

Mnemónico	Condición requerida	Interpretación (salta si...)
JNL o JGE	(SF xor OF) = 0	"no menor" o "mayor o igual"
JNLE o JG	((SF xor OF) or ZF) = 0	"no menor o igual" o "mayor"
JNB o JAE o JNC	CF = 0	"no abajo" o "arriba o igual" o "no acarreo"
JNBE o JA	(CF or ZF) = 0	"no abajo o igual" o "arriba"
JNP o JPO	PF = 0	"no paridad" o "paridad non"
JNO	OF = 0	"no overflow"
JNS	SF = 0	"no signo"
JCXZ	CX = 0	"registro CX = 0"

Instrucciones de Control de Iteraciones

Se utilizan cuando es necesario ejecutar una secuencia de instrucciones un número repetido de veces. En forma similar a las instrucciones de saltos condicionales, requieren de un sólo operando que especifique la dirección del salto (etiqueta). En la siguiente tabla se ilustran estas instrucciones:

Instrucciones de Control de Iteraciones

Mnemónico	Acción
LOOP	CX = CX - 1, Si CX <> 0 salta, continúa
LOOPE (Loop While Equal) LOOPZ (Loop While Zero)	CX = CX - 1, Si CX <> 0 y ZF = 1 salta, continúa
LOOPNE (Loop While Not Equal) LOOPNZ (Loop While Not Zero)	CX = CX - 1, Si CX <> 0 y ZF = 0 salta, continúa

La instrucción LOOP: En esta instrucción, el registro CX contiene un valor de conteo del ciclo. Su contenido se decrementa en uno y, *si el resultado es diferente de cero*, se transfiere el control a la dirección del salto (etiqueta). Si el resultado es cero, el programa continúa secuencialmente. Por ejemplo, supongamos que tenemos dos arreglos, cada uno con 10 elementos de un byte. Deseamos sumar cada elemento del primer arreglo con el correspondiente elemento del segundo arreglo. Utilizaremos el registro SI como un apuntador a los arreglos. La parte principal de nuestro ciclo buscará un elemento del primer arreglo, lo sumará el segundo arreglo y actualizará el apuntador SI. Esta operación se realizará 10 veces:

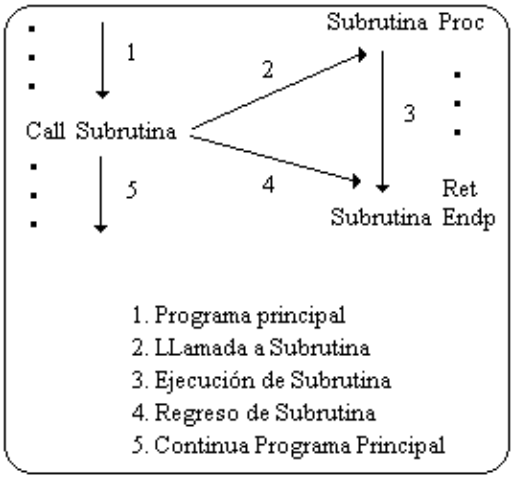
```
        MOV  SI,0
        MOV  CX,10
AGAIN:  MOV  AL,[SI+ARRAY1]
        ADD  [SI+ARRAY2],AL
        INC  SI
        LOOP AGAIN
```

La instrucción LOOPE funciona igual que la instrucción LOOP, excepto que permite terminar el ciclo antes de que CX sea igual a cero *si la bandera de ZF está en cero*. La instrucción LOOPNE permite terminar el ciclo antes si la *bandera ZF está en uno*.

Subrutinas

Se utilizan cuando necesitamos ejecutar una secuencia de instrucciones varias veces en diferentes puntos del programa. Para evitar teclear estas líneas en cada punto del programa, se colocan en una *subrutina*. Esta subrutina se ***manda llamar*** en cada punto donde se necesita. El control del programa se transfiere a la subrutina, se ejecutan las instrucciones que se encuentran dentro de ella y, cuando se ha terminado la ejecución, la subrutina ***regresa*** el control del programa al punto donde inicialmente fue llamada. La figura 13 ilustra lo anterior:

figura 13.

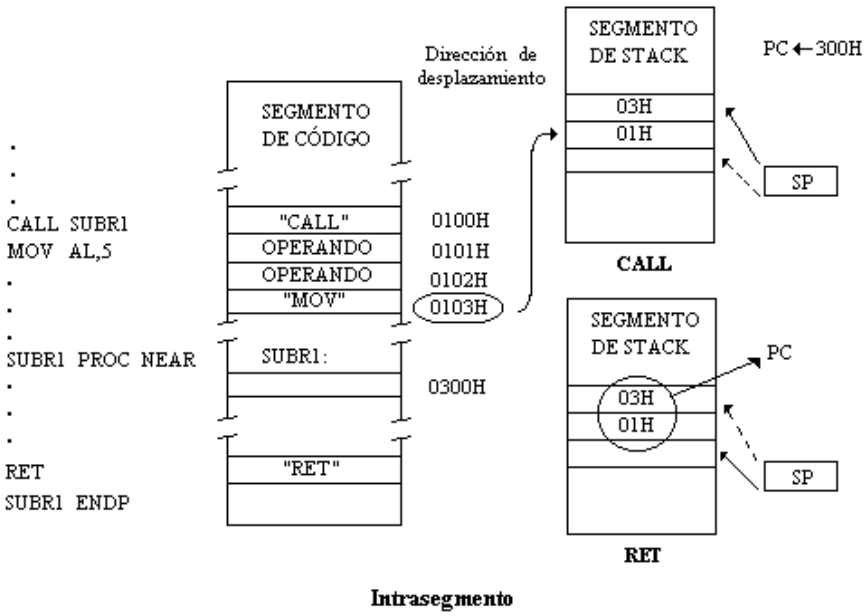


El 8088 implementa este mecanismo con las instrucciones **CALL** y **RET**. Ambas instrucciones pueden realizarse intrasegmento o intersegmento (figura 14). Cuando se ejecuta la subrutina, la **localidad donde se debe de regresar el control** se guarda en el *stack*. Esta información se saca del stack (POP) para efectuar el regreso (RET).

Las llamadas *intrasegmento* guardan sólo la dirección de *desplazamiento* de la localidad de regreso. Al realizar la instrucción RET, esta dirección se saca del stack y se coloca dentro del PC.

Las llamadas *intersegmento* guarda la dirección *física* del registro de segmento de código (CS) y la dirección de *desplazamiento* de la localidad de regreso. Al realizar la instrucción RET, se saca la dirección del PC y posteriormente la dirección del CS.

figura 14.



Interrupciones

Para llevar cuenta de los sucesos que vienen del exterior, el microprocesador debe ser capáz de identificarlos y tomar las acciones correspondientes. Por ejemplo, para determinar cuándo se está utilizando el teclado, éste manda una señal al microprocesador cada vez que se oprime una tecla. El microprocesador decodifica esta señal y, una vez determinado que la señal proviene del teclado, lee el caracter del puerto adecuado.

La señal de la que se habla se conoce como **interrupción**, y el *mecanismo de interrupción* del microprocesador es responsable de atenderla. Es importante comprender que las interrupciones pueden ocurrir en cualquier momento. Por ello, el mecanismo de interrupción debe de ser capáz de:

1. *Deshabilitar* al microprocesador para que pueda atender a la señal de interrupción.
2. Cuando se recibe una señal de interrupción, el microprocesador debe de *guardar* información de lo que estaba ejecutando en ese momento para poder continuar una vez que se ha atendido la interrupción.

El primer punto se cumple utilizando la **bandera de interrupción** del 8088 (figura 3). Si la bandera de interrupción IF está en cero (IF=0), el microprocesador ignorará cualquier señal de interrupción. Sólo cuando IF está en uno (IF=1) el microprocesador responderá a las llamadas de interrupción.

El segundo punto del mecanismo de interrupción se cumple al tratar a las interrupciones como si fueran **llamadas a subrutinas**. Cuando se recibe una interrupción, se guarda en el stack el **registro de banderas** (16 bits). Las banderas de interrupción (IF) y trampa (TF) se colocan en cero (para evitar recibir una nueva interrupción mientras no se termine de atender a la primera). Después, el registro CS se guarda en el stack. Finalmente, la dirección de

desplazamiento de la siguiente instrucción a realizar (contenida dentro del PC) se guarda dentro del stack.

Una vez que se ha salvado el estado actual del procesador, el mecanismo de interrupción transfiere el control a un *programa especial en una dirección específica*. Este programa llamado, **rutina de servicio de interrupción**, es responsable de manejar el suceso externo. Cuando termina su tarea, regresa el control a la tarea que se encontraba realizando el microprocesador antes de la interrupción (sacando del stack los registros PC, CS y las banderas). Esto se logra con la instrucción **IRET** (interrupt return).

Nota: Si la rutina de servicio altera cualquier otro registro, guarda en el stack su valor y lo restaura después de terminar su tarea, de tal forma que se conserva la integridad del programa que recibió la interrupción.

Instrucciones de Control del Procesador

Instrucciones de Control del Procesador

Mnemónico	Acción
CLD (Clear Carry Flag)	CF = 1
CLD (Clear Direction Flag)	DF = 0
CLI (Clear Interrupt Flag)	IF = 0
CMC (Complement Carry Flag)	CF = NOT(CF)
STC (Set Carry Flag)	CF = 1
STD (Set Direction Flag)	DF = 1
STI (Set Interrupt Flag)	IF = 1
HLT (Halt)	Stop

Estas instrucciones nos permiten manipular las banderas de acarreo (CF), dirección (DF) e interrupción (IF) directamente.

La instrucción HLT provoca que el procesador se detenga y no ejecute más instrucciones.