Unidad II: Programación básica

2.1 Ensamblador (y ligador) a utilizar

Aunque todos los ensambladores realizan básicamente las mismas tareas, podemos clasificarlos de acuerdo a características.

Así podemos clasificarlos en:

Ensambladores Cruzados (Cross-Assembler).

Se denominan así los ensambladores que se utilizan en una computadora que posee un procesador diferente al que tendrán las computadoras donde va a ejecutarse el programa objeto producido.

El empleo de este tipo de traductores permite aprovechar el soporte de medios físicos (discos, impresoras, pantallas, etc.), y de programación que ofrecen las máquinas potentes para desarrollar programas que luego los van a ejecutar sistemas muy especializados en determinados tipos de tareas.

Ensambladores Residentes.

Son aquellos que permanecen en la memoria principal de la computadora y cargan, para su ejecución, al programa objeto producido. Este tipo de ensamblador tiene la ventaja de que se puede comprobar inmediatamente el programa sin necesidad de transportarlo de un lugar a otro, como se hacía en cross-assembler, y sin necesidad de programas simuladores.

Sin embargo, puede presentar problemas de espacio de memoria, ya que el traductor ocupa espacio que no puede ser utilizado por el programador. Asimismo, también ocupará memoria el programa fuente y el programa objeto. Esto obliga a tener un espacio de memoria relativamente amplio. Es el indicado para desarrollos de pequeños sistemas de control y sencillos automatismo empleando microprocesadores.

La ventaja de estos ensambladores es que permiten ejecutar inmediatamente el programa; la desventaja es que deben mantenerse en la memoria principal tanto el ensamblador como el programa fuente y el programa objeto.

Macroensambladores.

Son ensambladores que permiten el uso de macroinstrucciones (macros). Debido a su potencia, normalmente son programas robustos que no permanecen en memoria una vez generado el programa objeto. Puede variar la complejidad de los mismos, dependiendo de las posibilidades de definición y manipulación de las macroinstrucciones, pero normalmente son programas bastantes complejos, por lo que suelen ser ensambladores residentes.

Microensambladores.

Generalmente, los procesadores utilizados en las computadoras tienen un repertorio fijo de instrucciones, es decir, que el intérprete de las mismas interpretaba de igual forma un determinado código de operación.

El programa que indica al intérprete de instrucciones de la UCP cómo debe actuar se denomina microprograma. El programa que ayuda a realizar esta microprograma se llama microensamblador. Existen procesadores que permiten la modificación de sus microprogramas, para lo cual se utilizan microensambladores.

Ensambladores de una fase.

Estos ensambladores leen una línea del programa fuente y la traducen directamente para producir una instrucción en lenguaje máquina o la ejecuta si se trata de una pseudoinstrucción. También va construyendo la tabla de símbolos a medida que van apareciendo las definiciones de variables, etiquetas, etc.

Debido a su forma de traducción, estos ensambladores obligan a definir los símbolos antes de ser empleados para que, cuando aparezca una referencia a un determinado símbolo en una instrucción, se conozca la dirección de dicho símbolo y se pueda traducir de forma correcta. Estos ensambladores son sencillos, baratos y ocupan poco espacio, pero tiene el inconveniente indicado.

Ensambladores de dos fases.

Los ensambladores de dos fases se denominan así debido a que realizan la traducción en dos etapas. En la primera fase, leen el programa fuente y construyen una tabla de símbolos; de esta manera, en la segunda fase, vuelven a leer el programa fuente y pueden ir traduciendo totalmente, puesto que conocen la totalidad de los símbolos utilizados y las posiciones que se les ha asignado. Estos ensambladores son los más utilizados en la actualidad.

2.2 Ciclos numéricos

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 accesar 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 accesa 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.

2.3 Captura básica de cadenas

2.4 Comparación y prueba

La instrucción CMP pro lo común es utilizada para comparar dos campos de datos, uno de los cuales están contenidos en un registro. El formato general para CMP es:| [etiqueta:] | CMP | {registro/memoria}, {registro/memoria/inmediato} |

Observe que la operación compara el primer operando con el segundo; por ejemplo, el valor del primer operando es mayor que, igual o menor que el valor del segundo operando?

La instrucción CMPS compara el contenido de una localidad de memoria (direccionada por DS:SI). Dependiendo de la bandera de dirección, CMPS

incrementa o disminuye también losregistros SI y DI en 1 para bytes, en 2 para palabras y en 4 para palabras dobles. La operación establece las banderas AF, CF, OF, PF, SF y ZF.

Cuando se combinan con un prefijo REP y una longitud en el CX, de manera sucesiva CMPS puede comparar cadenas de cualquier longitud.

Pero observe que CMPS proporciona una comparación alfanumérica, esto es, una comparación de acuerdo a con los valores ASCII. Considere la comparación de dos cadenas que contienen JEAN y JOAN

Algunas derivaciones de CMPS son las siguientes:

- CMPSB. Compara bytes
- CMPSD. Compara palabras dobles
- CMPSW. Compara palabras

2.5 Saltos

La mayoría de los programas constan de varios ciclos en los que una serie de pasos se repite hasta alcanzar un requisito específico y varias pruebas para determinar qué acción se realiza de entre varias posibles.

Una instrucción usada comúnmente para la transferencia de control es la instrucción JMP

(jump, salto, bifurcación). Un salto es incondicional, ya que la operación transfiere el control bajo cualquier circunstancia. También JMP vacía el resultado de la instrucción previamente procesada; por lo que, un programa con muchas operaciones de salto puede perder velocidad de procesamiento.

La instrucción LOOP, requiere un valor inicial en el registro CX. En cada iteración, LOOP de forma automática disminuye 1 de CX. Si el valor en el CX es cero, el control pasa a la instrucción que sigue; si el valor en el CX no es cero, el control pasa a la dirección del operando. La distancia debe ser un salto corto, desde -128 hasta +127 bytes. Para una operación que exceda este límite, el ensamblador envía un mensaje como "salto relativo fuera de rango".

2.6 Ciclos condicionales

Sintaxis:

LOOP etiqueta

La instrucción loop decrementa CX en 1, y transfiere el flujo del programa a la etiqueta dada como operando si CX es diferente a 1.

Instrucción LOOPE

Propósito: Generar un ciclo en el programa considerando el estado de ZF

Sintaxis:

LOOPE etiqueta

Esta instrucción decrementa CX en 1. Si CX es diferente a cero y ZF es igual a 1, entonces el flujo del programa se transfiere a la etiqueta indicada como operando.

Instrucción LOOPNE

Propósito: Generar un ciclo en el programa, considerando el estado de ZF

Sintaxis:

LOOPNE etiqueta

Esta instrucción decrementa en uno a CX y transfiere el flujo del programa solo si ZF es diferente a 0.

2.7 Incremento y decremento

Son las instrucciones más básicas a la hora de hacer operaciones con registros: INC incrementa el valor de un registro, o de cualquier posición en memoria, en una unidad, y DEC lo decrementa.

Instrucción INC

INC AX

Incrementa en uno el valor de AX

IN WORD PTR

Incrementa la palabra situada en CS.

Instrucción DEC

DEC AX

Decremento AX, le resta uno.

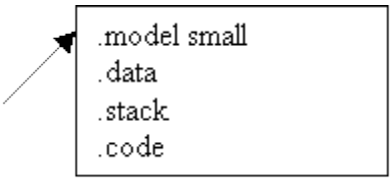
DEC WORD PTR

Decrementa la palabra situada en CS.

2.8 Captura de cadenas con formato

Formato de instrucciones para lenguaje ensamblador:

Directivas de segmento



Nombre_procedimiento PROC

Instrucción operando **destino**

, operando fuente

Nombre_procedimiento ENDP

END

; comentarios

MOV

Transfiere datos entre celdas de memoria y registros.

Sintaxis: **MOV Destino,Fuente**

Ejemplo:

```
MOV AX,0006h
MOV DX,AX
```

MOVS (MOVSB) (MOVSW)

Mueve cadenas de bytes o palabras desde la fuente, direccionada por SI, hasta el destino direccionado por DI.

Sintaxis: **MOVS**

Este comando no necesita parametros ya que toma como dirección fuente el contenido del registro SI y como destino el contenido de DI.

Ejemplo:

```
MOV SI, OFFSET VARIABLE1
MOV DI, OFFSET VARIABLE2
MOVS
```

Primero se inicializan los valores de SI y DI con las direcciones de las variables VARIABLE1 y VARIABLE2 respectivamente, despues al ejecutar MOVS se copia el contenido de VARIABLE1 a VARIABLE2.

Los comandos MOVSB y MOVSW se utilizan de la misma forma que MOVS, el primero mueve un byte y el segundo una palabra.

LODS (LODSB) (LODSW)

Carga cadenas de un byte o palabra al acumulador.

Sintaxis: **LODS**

Toma la cadena que se encuentre en la dirección especificada por SI, la carga al registro AL (o AX) y suma o resta 1 (segun el estado de DF) a SI si la transferencia es de bytes o 2 si la transferencia es de palabras.

Ejemplo:

```
MOV SI, OFFSET VARABLE1
LODS
```

La primer linea carga la dirección de VARIABLE1 en SI y la segunda linea lleva el contenido de esa localidad al registro AL.

Los comandos LODSB y LODSW se utilizan de la misma forma, el primero carga un byte y el segundo una palabra (utiliza el registro completo AX).

LAHF

Transfiere al registro AH el contenido de las banderas

Sintaxis: **LAHF**

Se utiliza para verificar el estado de las banderas durante la ejecución de un programa.

Las banderas quedan en el siguiente orden dentro del registro:

SF ZF __ AF __ PF __ CF

LEA

Carga la dirección del operando fuente.

Sintaxis: **LEA destino, fuente**

El operando fuente debe estar ubicado en memoria, y se coloca su desplazamiento en el registro índice o apuntador especificado en destino.

Ejemplo:

MOV SI, OFFSET VAR1

Que es equivalente a:

LEA SI, VAR1

POP

Recupera un dato de la pila

Sintaxis: **POP destino**

Transfiere el último valor almacenado en la pila al operando destino y despues incrementa en dos el registro SP.

Este incremento se debe a que la pila va creciendo desde la dirección mas alta de memoria del segmento hacia la mas baja, y la pila solo trabaja con palabras (2 bytes), entonces al incrementar en dos el registro SP realmente se le esta restando dos al tamaño real de la pila.

POPF

Extrae las banderas almacenadas en la pila.

Sintaxis: **POPF**

Transfiere bits de la palabra almacenada en la parte superior de la pila hacia el registro de banderas.

La forma de transferencia es la siguiente:

BIT BANDERA 0 CF ___ 2 PF ___ 4 AF ___ 6 ZF 7 SF 8 TF 9 IF 10 DF 11 OF

Estas localizaciones son las mismas para el comando PUSHF

Una vez hecha la transferencia se incrementa en 2 el registro SP disminuyendo así el tamaño de la pila.

PUSH

Coloca una palabra en la pila.

Sintaxis: **PUSH fuente**

La instrucción PUSH decrementa en dos el valor de SP y luego transfiere el contenido del operando fuente a la nueva dirección resultante en el registro recién modificado.

El decremento en la dirección se debe a que al agregar valores a la pila ésta crece de la dirección mayor a la dirección menor del segmento, por lo tanto al restarle 2 al valor del registro SP lo que hacemos es aumentar el tamaño de la pila en dos bytes, que es la única cantidad de información que puede manejar la pila en cada entrada y salida de datos.

PUSHF

Coloca el valor de las banderas en la pila

Sintaxis: **PUSHF**

Decrementa en 2 el valor del registro SP y luego se transfiere el contenido del registro de banderas a la pila, en la dirección indicada por SP.

Las banderas quedan almacenadas en memoria en los mismos bits indicados en el comando POPF

2.9 Instrucciones aritméticas

ADC

Adición con acarreo.

Sintaxis: **ADC destino, fuente**

Lleva a cabo la suma de dos operandos y suma uno al resultado en caso de que la bandera CF esté activada, esto es, en caso de que exista acarreo.

El resultado se guarda en el operando destino.

ADD

Adición de los operandos.

Sintaxis: **ADD destino, fuente**

Suma los dos operandos y guarda el resultado en el operando destino.

DIV

División sin signo

Sintaxis: **DIV fuente**

El divisor puede ser un byte o palabra y es el operando que se le da a la instrucción.

Si el divisor es de 8 bits se toma como dividendo el registro de 16 bits AX y si el divisor es de 16 bits se tomara como dividendo el registro par DX:AX, tomando como palabra alta DX y como baja AX.

Si el divisor fué un byte el cociente se almacena en el registro AL y el residuo en AH, si fué una palabra el cociente se guarda en AX y el residuo en DX.

IDIV

División con signo

Sintaxis: **IDIV fuente**

Consiste basicamente en lo mismo que la instrucción DIV, solo que esta última realiza la operación con signo.

MUL

Multiplicación sin signo

Sintaxis: **MUL fuente**

El ensamblador asume que el multiplicando sera del mismo tamaño que el del multiplicador, por lo tanto multiplica el valor almacenado en el registro que se le da como operando por el que se encuentre contenido en AH si el multiplicador es de 8 bits o por AX si el multiplicador es de 16 bits.

Cuando se realiza una multiplicación con valores de 8 bits el resultado se almacena en el registro AX y cuando la multiplicación es con valores de 16 bits el resultado se almacena en el registro par DX:AX.

IMUL

Multiplicación de dos enteros con signo.

Sintaxis: **IMUL fuente**

Este comando hace lo mismo que el anterior, solo que si toma en cuenta los signos de las cantidades que se multiplican.

Los resultados se guardan en los mismos registros que en la instrucción MUL.

SBB

Sbstracción con acarreo

Sintaxis: **SBB destino, fuente**

Esta instrucción resta los operandos y resta uno al resultado si CF está activada. El operando fuente siempre se resta del destino.

Este tipo de substracción se utiliza cuando se trabaja con cantidades de 32 bits.

SUB

Substracción

Sintaxis: **SUB destino, fuente**

Resta el operando fuente del destino.

2.10 Manipulación de la pila

-ROT	(a b c – c a b)	Rota hacia atrás.
-2ROT	(ab cd ef – ef ab cd)	Rota hacia atrás.
NIP	(a b – b)	Quita a de la pila.
OUTK	(... n -- ..)	Elimina el elemento n.
TUCK	(a b -- b a b)	Inserta una copia de b.

2?DUP	(ab – ab ab)	Duplica si ab <> 0.
2DROP	(ab --)	Elimina 2 de encima.
2DUP	(ab – ab ab)	Duplica los elementos.
2NIP	(ab cd – cd)	Elimina elementos.
2OUTK	(::: n -- ::)	Elimina el elemento n
2OVER	(ab cd – ab cd ab)	Inserta una copia de ab.
2PICK	(:: n -- :::)	Copia el elemento n encima de la pila.
2ROLL	(::: n -- :::)	Quita el elemento n y lo deja arriba de la pila.
2ROT	(ab cd ef – cd ef ab)	Rota los elementos
2TUCK	(ab cd – cd ab cd)	Inserta una copia de cd.
2SWAP	(ab cd – cd ab)	Rota los elementos.

2.11 Obtención de cadena con representación decimal

En este modo, los datos son proporcionados directamente como parte de la instrucción.

Ejemplo:

Mov AX,34h ;Copia en AX el número 34h hexadecimal

Mov CX,10 ;Copia en CX el número 10 en decimal

2.12 Instrucciones lógicas

AND

Realiza la conjunción de los operandos bit por bit.

Sintaxis: **AND destino, fuente**

Con esta instrucción se lleva a cabo la operación "y" lógica de los dos operandos:

Fuente	Destino	Resultado en operando destino
1	1	1
1	0	0
0	1	0
0	0	0

NEG

Genera el complemento a 2

Sintaxis: **NEG destino**

Genera el complemento a 2 del operando destino y lo almacena en este mismo operando.

Ejemplo, si AX guarda el valor de –2 (FFFE), entonces:

NEG AX

Dejaría como resultado en AX el valor 0002.

NOT

Lleva a cabo la negación bit por bit del operando destino.

Sintaxis: **NOT destino**

El resultado se guarda en el mismo operando destino.

OR

OR inclusivo lógico

Sintaxis: **OR destino, fuente**

La instrucción OR lleva a cabo, bit por bit, la disyunción inclusiva lógica de los dos operandos:

Fuente	Destino	Resultado en operando destino
1	1	1
1	0	1
0	1	1
0	0	0

TEST

Compara logicamente los operandos

Sintaxis: **TEST destino, fuente**

Realiza una conjunción, bit por bit, de los operandos, pero a diferencia de AND esta instrucción no coloca el resultado en el operando destino, solo tiene efecto sobre el estado de las banderas.

XOR

OR exclusivo

Sintaxis: **XOR destino, fuente**

Su función es efectuar bit por bit la disyunción exclusiva lógica de los dos operandos.

Fuente	Destino	Resultado en operando destino
1	1	0
1	0	1
0	1	1
0	0	0

2.13 Desplazamiento y rotación

Las instrucciones de corrimiento, que son parte de la capacidad lógica de la computadora, pueden realizar las siguientes acciones:

1. Hacer referencia a un registro o dirección de memoria.

2. Recorre bits a la izquierda o a la derecha.

3. Recorre hasta 8 bits en un byte, 16 bits en una palabra y 32 bits en una palabra doble.

4. Corrimiento lógico (sin signo) o aritmético (con signo).

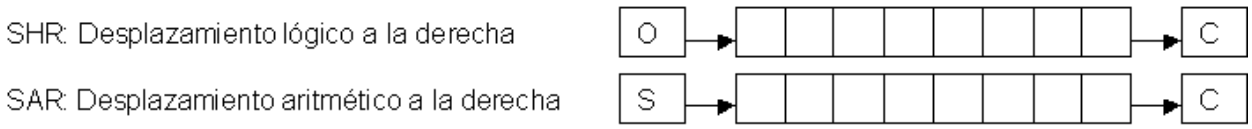
El segundo operando contiene el valor del corrimiento, que es una constante (un valor inmediato) o una referencia al registro CL. Para los procesadores 8088/8086, la constante inmediata solo puede ser 1; un valor de corrimiento mayor que 1 debe estar contenido en el registro CL. Procesadores posteriores permiten constantes de corrimiento inmediato hasta 31.

El formato general para el corrimiento es

| [etiqueta:] | Corrim. | {registro/memoria}, {CL/inmediato} |

DESPLAZAMIENTO O CORRIMIENTO DE BITS HACIA LA DERECHA.

Los corrimientos hacia la derecha (SHR y SAR) mueven los bits hacia la derecha en el registro designado. El bit recorrido fuera del registro mete la bandera de acarreo. Las instrucciones de corrimiento a la derecha estipulan datos lógicos (sin signo) o aritméticos (con signo):



Las siguientes instrucciones relacionadas ilustran SHR y datos con signo:

INSTRUCCION **COMENTARIO**

MOV CL, 03

MOV AL, 10110111B ; AL = 10110111

SHR AL, 01 ; AL = 01011011 Un corrimiento a la derecha

SHR AL, CL ; AL = 00001011 Tres corrimientos adicionales a la derecha

El primer SHR desplaza el contenido de AL un bit hacia la derecha. El bit de mas a la derecha es enviado a la bandera de acarreo, y el bit de mas a la izquierda se

llena con un cero. El segundo SHR desplaza tres bits más a AL. La bandera de acarreo contiene de manera sucesiva 1, 1 y 0; además, tres bits 0 son colocados a la izquierda del AL.

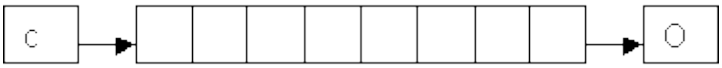
SAR se difiere de SHR en un punto importante: **SAR utiliza el bit de signo** para llenar el bit vacante de más a la izquierda. De esta manera, los valores positivos y negativos retienen sus signos. Las siguientes instrucciones relacionadas ilustran SAR y datos con signo en los que el signo es un bit 1:

En especial, los corrimientos a la derecha son útiles para (dividir entre 2) obtener mitades de valores y son mucho más rápidas que utilizar una operación de división. Al terminar una operación de corrimiento, puede utilizar la instrucción JC (Salta si hay acarreo) para examinar el bit desplazado a la bandera de acarreo.

DESPLAZAMIENTO O CORRIMIENTO DE BITS A LA IZQUIERDA.

Los corrimientos hacia la izquierda (SHL y SAL) mueven los bits a la izquierda, en el registro designado. SHL y SAL son idénticos en su operación. El bit desplazado fuera del registro ingresa a la bandera de acarreo. Las instrucciones de corrimiento hacia la izquierda estipulan datos lógicos (sin signo) y aritméticos (con signo):

SHL: Desplazamiento lógico a la izquierda SAL: Desplazamiento aritmético a la izquierda



Las siguientes instrucciones relacionadas ilustran SHL para datos sin signo:

INSTRUCCION	**COMENTARIO**
MOV CL, 03	
MOV AL, 10110111B	; AL = 10110111
SHL AL, 01	; AL = 01101110 Un corrimiento a la izquierda
SHL AL, CL	; AL = 01110000 Tres corrimientos mas

El primer SHL desplaza el contenido de AL un bit hacia la izquierda. El bit de más a la izquierda ahora se encuentra en la bandera de acarreo, y el último bit de la derecha del AL se llena con cero. El segundo SHL desplaza tres bits más a AL. La bandera de acarreo contiene en forma sucesiva 0, 1 y 1, y se llena con tres ceros a la derecha del AL.

Los corrimientos a la izquierda llenan con cero el bit de mas a la derecha. Como resultado de esto, SHL y SAL don idénticos. Los corrimientos a la izquierda en especial son útiles para duplicar valores y son mucho más rápidos que usar una operación de multiplicación.

Al terminar una operación de corrimiento, puede utilizar la instrucción JC (Salta si hay acarreo) para examinar el bit que ingreso a la bandera de acarreo.

ROTACION DE BITS (DESPLAZAMIENTO CIRCULAR)

Las instrucciones de rotación, que son parte de la capacidad lógica de la computadora, pueden realizar las siguientes acciones:

1. Hacer referencia a un byte o a una palabra.

2. Hacer referencia a un registro o a memoria.

3. Realizar rotación a la derecha o a la izquierda. El bit que es desplazado fuera llena
el espacio vacante en la memoria o registro y también se copia en la bandera de acarreo.

4. Realizar rotación hasta 8 bits en un byte, 16 bits en una palabra y 32 bits en una palabra doble.

5. Realizar rotación lógica (sin signo) o aritmética (con signo).

El segundo operando contiene un valor de rotación, el cual es una constante (un valor inmediato) o una referencia al registro CL. Para los procesadores 8088/8086, la constante inmediata solo puede ser 1; un valor de rotación mayor que 1 debe estar contenido en el registro CL. Procesadores posteriores permiten constantes inmediatas hasta el 31. El formato general para la rotación es:

[etiqueta:]	Rotación	{registro/memoria}, {CL/inmediato}

ROTACIÓN A LA DERECHA DE BITS

Las rotaciones a la derecha (ROR y RCR) desplazan a la derecha los bits en el registro designado. Las instrucciones de rotación a la derecha estipulan datos lógicos (sin signo) o aritméticos (con signo):

ROR: Rotacion logica a la derecha

RCR: Rotacion a la derecha con acarreo

Las siguientes instrucciones relacionadas ilustran ROR:

INSTRUCCION	**COMENTARIO**
MOV CL, 03	
MOV BH, 10110111B	; BH = 10110111
ROR BH, 01	; BH = 11011011 Una rotación a la derecha
ROR BH, CL	; BH = 00001011 Tres rotaciones a la derecha

El primer ROR desplaza el bit de más a la derecha del BH a la posición vacante de más a la izquierda. La segunda y tercera operaciones ROR realizan la rotación de los tres bits de mas a la derecha.

RCR provoca que la bandera de acarreo participe en la rotación. Cada bit que se desplaza fuera de la derecha se mueve al CF y el bit del CF se mueve a la posición vacante de la izquierda.

ROTACIÓN A LA IZQUIERDA DE BITS

Las rotaciones a la izquierda (ROL y RCL) desplazan a la izquierda los bits del registro designado. Las instrucciones de rotación a la izquierda estipulan datos lógicos (sin signo) y aritméticos (con signo):

ROL: Rotacion logica a la izquierda

RCL: Rotacion a la izquierda con acarreo

Las siguientes instrucciones relacionadas ilustran ROL:

INSTRUCCION	**COMENTARIO**
MOV CL, 03	
MOV BL, 10110111B	; BL = 10110111
SHR BL, 01	; BL = 11011011 Una rotación a la izquierda
SHR BL, CL	; BL = 00001011 Tres rotaciones a la izquierda

El primer ROL desplaza el bit de mas a la izquierda del BL a la posición vacante de mas a la derecha. La segunda y tercera operaciones ROL realizan la rotación de los tres bits de mas a la izquierda.

De manera similar a RCR, RCL también provoca que la bandera de acarreo participe en la rotación. Cada bit que se desplaza fuera por la izquierda se mueve al CF, y el bit del CF se mueve a la posición vacante de la derecha.

Puede usar la instrucción JC (salta si hay acarreo) para comprobar el bit rotado hacia la CF en el extremo de una operación de rotación.

Tomado de: Tutorial de Lenguaje Ensamblador

2.14 Obtención de una cadena con la representación hexadecimal

La conversión entre numeración binaria y hexadecimal es sencilla. Lo primero que se hace para una conversión de un número binario a hexadecimal es dividirlo en grupos de 4 bits, empezando de derecha a izquierda. En caso de que el último grupo (el que quede más a la izquierda) sea menor de 4 bits se rellenan los faltantes con ceros.

Tomando como ejemplo el número binario 101011 lo dividimos en grupos de 4 bits y nos queda:

10; 1011

Rellenando con ceros el último grupo (el de la izquierda):

0010; 1011

después tomamos cada grupo como un número independiente y consideramos su valor en decimal:

0010 = 2; 1011 = 11

Pero como no podemos representar este número hexadecimal como 211 porqué sería un error, tenemos que sustituir todos los valores mayores a 9 por su respectiva representación en hexadecimal, con lo que obtenemos:

2BH (Donde la H representa la base hexadecimal)

Para convertir un número de hexadecimal a binario solo es necesario invertir estos pasos: se toma el primer dígito hexadecimal y se convierte a binario, y luego el segundo, y así sucesivamente hasta completar el número.

2.15 Captura y almacenamiento de datos numéricos

Esta representación esta basada en la notación científica, esto es, representar un número en dos partes: su mantisa y su exponente.

Poniendo como ejemplo el número 1234000, podemos representarlo como 1.123*10^6, en esta última notación el exponente nos indica el número de espacios que hay que mover el espacio hacia la derecha para obtener el resultado original.

En caso de que el exponente fuera negativo nos estaría indicando el número de espacios que hay que recorrer el punto decimal hacia la izquierda para obtener el original.

Proceso de 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 útilizados para el mejor planteamiento de la solución.

La codificación del programa consiste en escribir el programa en algún lenguaje de programación; en este caso específico en ensamblador, 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.

2.16 Operaciones básicas sobre archivos de disco

Servicios de la interrupción 16h para manejo del teclado.

Función 00h. Lee un carácter. Esta función maneja las teclas del teclado de 83 teclas, pero no acepta entrada de las teclas adicionales en el teclado ampliado de 101 teclas. Retorna en al el carácter, ah el código de rastreo si al=0 es una tecla de función extendida.

Función 01h. Determina si un carácter esta presente.

Función 02h. Regresa el estado actual de las teclas shift.

Función 10h. Lectura de un carácter del teclado.

Función 11h. Determina si esta presente un carácter.

MOVS. Mueve un byte, palabra o palabra doble desde una localidad en memoria direccionada por SI a otra localidad direccionada por DI.

LODS. Carga desde una localidad de memoria direccionada por SI un byte en AL, una palabra en AX o una palabra doble en EAX.

STOS. Almacena el contenido de los registros AL, AX, o EAX en la memoria direccionada por SI.

CMPS. Compara localidades de memoria de un byte, palabra o palabra doble direccionadas por SI, DI.

SCAS. Compara el contenido de AL, AX o EAX con el contenido de una localidad de memoria direccionada por SI.