UNIVERSIDAD AUTONOMA DE TAMAULIPAS
LICENCIATURA EN INFORMATICA
MATERIA: **LENGUAJE ENSAMBLADOR**
PROFESOR: **ING. RAMON V. ROQUE HERNÁNDEZ, M.C.**
ramonroque@yahoo.com ramonroque@hotmail.com

Descripción general de la materia:

La materia presenta al alumno los conceptos, fundamentos e instrucciones necesarios para programar en lenguaje ensamblador con nutridos ejemplos.

Contenido:

Unidad I.- **Conceptos e introducción**

- Introducción, Ventajas / Desventajas del lenguaje ensamblador
- Ubicación del Ensamblador dentro de la clasificación de los lenguajes de programación
- Conceptos
- Sistemas Numéricos, conversiones y operaciones
- Representación de datos en la PC

Unidad II.- **Arquitectura de la PC**

- Componentes básicos
  - (Memoria ROM y RAM, Temporizador, Chip PPI, Controlador de interrupciones, Chip DMA)
- Microprocesador
  - (Bus, Unidad de Ejecución, Interfaz de Bus)
- Registros
  - (Uso general, Apuntadores, Indices, Segmentos, Banderas)
- Modos de direccionamiento
  - (a registros, inmediato, directo, indirecto, base)
- La Pila

Unidad III.- **Conjunto de Instrucciones (Lenguaje Ensamblador)**

- Instrucciones para movimiento de datos
- Instrucciones aritméticas
- Instrucciones lógicas
- Instrucciones de transferencia de control

Unidad IV.- **Entorno de programación (DEBUG)**

- Descripción del debug
- Comandos Generales
- Prácticas

Unidad V.- **Técnicas de programación avanzadas**

- Interrupciones
- Archivos

Unidad VI.- **Aplicación de programas**

- Proyectos de aplicación

Forma de Evaluar:

- Exámenes
- Tareas e investigaciones
- Participación
- Exposición de clase
- Prácticas (programas)

Horario de clases:

MARTES y JUEVES
6:00 PM a 8:00 PM

1. INTRODUCCION

1.1 MOTIVACION PARA EL ESTUDIO DEL LENGUAJE ENSAMBLADOR

El lenguaje ensamblador ha sido una herramienta de desarrollo altamente criticada pero a la vez alabada. Sus críticos arguyen la complejidad de programación y su injerencia en el tiempo de desarrollo de sistemas, mientras sus seguidores se desbordan en alabanzas de su flexibilidad así como el sentimiento de poderío y conocimiento que este ofrece a aquellos que se atreven a aventurar mas allá de la superficie de un lenguaje de programación.

Con la creación de lenguajes definidos como "XBASE compatibles" y otros similares, el desarrollo de sistemas se ha facilitado enormemente. Sin embargo, lenguajes tales como COBOL, PASCAL, C, MODULA, BASIC y FORTRAN (entre otros) son herramientas alternas de desarrollo que de alguna forma dan mayor flexibilidad y control, sobre todo a las personas que se encuentran inmersas en las técnicas de programación.

La mayoría de los programadores de hoy en día usan uno de estos lenguajes en su trabajo, pero por mucha flexibilidad que dichos lenguajes ofrezcan es inevitable que en ocasiones tengan severas restricciones para implementar un proceso determinado. Este puede ser un manejador de dispositivos, una rutina cuyo tiempo de ejecución es vital, cuando se desea invocar alguna función interna de DOS o del BIOS, o necesita acceder alguna localidad específica de la memoria y dicho lenguaje no tiene los mecanismos que permitan hacerlo.

Es en estos momentos cuando los programadores recurren al lenguaje ensamblador.

La verdad finalmente, es que si desea efiecientizar, optimizar, afinar o profundizar en el conocimiento interno de la PC, el lenguaje ensamblador es la elección adecuada.

1.2 ¿QUE ES LENGUAJE ENSAMBLADOR?

El ensamblador es una variante (legible para el ser humano) del lenguaje máquina que usan las computadoras para ejecutar programas.
El ensamblador resulta indispensable cuando se desea escribir programas que controlen la entrada-salida de la PC, agregar nuevas interfaces de entrada-salida, escribir rutinas optimizadas de un procedimiento en especial, escribir rutinas que aprovechen y maximicen el uso del hardware, y en general realizar cualquier tarea que no puedan llevar a cabo los demás lenguajes de programación.

1.3 VENTAJAS DEL LENGUAJE ENSAMBLADOR

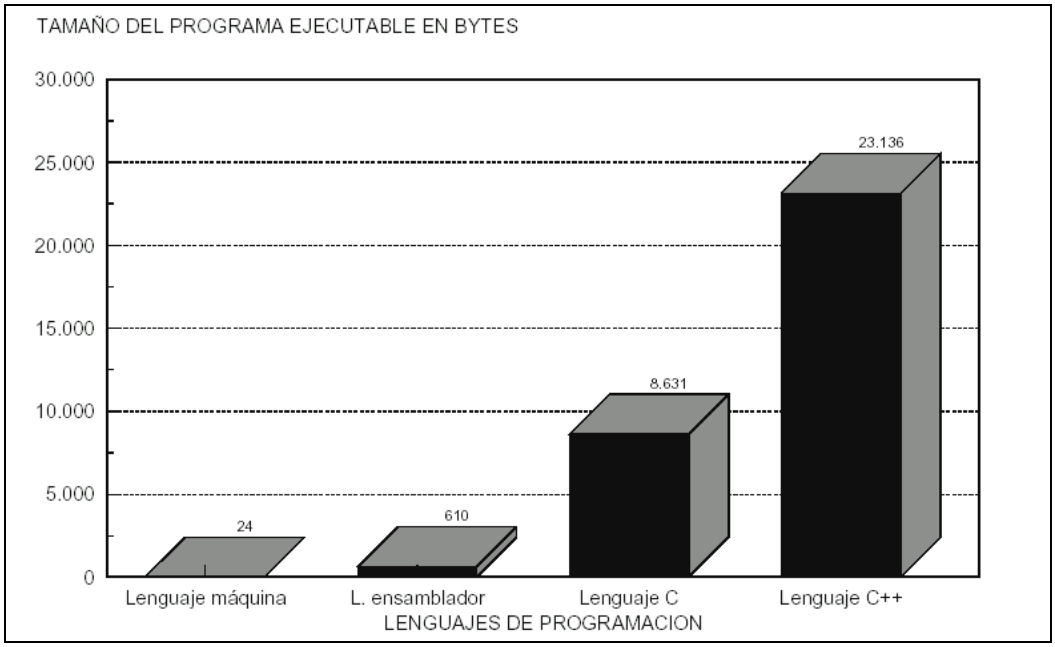
- El ensamblador le brinda la oportunidad de conocer mas a fondo la operación de la PC. Esto le permite implementar software o hardware de una manera mas consciente (comprendiendo como y por qué lo hace en vez de seguir una receta fija.
- El programador conserva el control total de lo que deberá hacer la PC, siempre y cuando ésta sea físicamente capaz de hacerlo.
- Los programas en ensamblador son mas rápidos, mas compactos y tienen mayor capacidad que los creados en otros lenguajes.
- Los programas se pueden optimizar al máximo (tanto en tamaño como en velocidad de ejecución). Contrario al lenguaje ensamblador, la mayoría de lenguajes de alto nivel genera el código de manera predefinida, lo que dificulta la optimización del programa.

  A continuación se presentan dos ejemplos: el primero en una tabla que indica los tamaños de los archivos (fuente, objeto y ejecutable) creados por diferentes compiladores; el segundo

en una gáfica. No se están comparando las velocidades de ejecución, aunque si tomamos en cuenta el tamaño del programa ejecutable (.EXE), es obvio que mientras mas grande sea éste, su ejecución será mas tardada debido al tiempo necesario para cargarlo.
NOTA: A pesar de que ambos ejemplos ilustran el mismo concepto, el Ejemplo 1 es completamente independiente del Ejemplo 2.

LENGUAJE	FUENTE	OBJETO	EJECUTABLE (.EXE)
BASIC	27 bytes	651 bytes	12,814 bytes
CLIPPER	22 bytes	572 bytes	159,178 bytes
ENSAMBLADOR	162 bytes	169 bytes	543 bytes

Ejemplo 1 : Comparación de tamaños de archivos producidos por programas similares en lenguajes diferentes



Ejemplo 2: Comparación de tamaños de archivos producidos por programas similares en lenguajes diferentes

1.4 DESVENTAJAS DEL LENGUAJE ENSAMBLADOR

- Una instrucción mal interpretada o un error de lógica en el programa pueden causar caos (Puede ser necesario reiniciar la PC).
- La insuficiencia de conocimientos sobre el funcionamiento interno de la PC puede causar efectos impredecibles.
- El programador debe adoptar una convención especial respecto al uso de interrupciones.
- El programa puede volverse muy complejo conforme se le agregan rutinas adicionales.

2. CLASIFICACION DE LOS LENGUAJES DE PROGRAMACION

2.1 LENGUAJES DE PROGRAMACION

Las relaciones humanas se llevan a cabo a través del lenguaje. Un lenguaje permite la expresion de ideas y razonamientos y sin él la comunicación sería imposible. Las computadoras solo aceptan y comprenden un lenguaje de bajo nivel, que consiste en largas secuencias de unos y ceros. Estas secuencias son ininteligibles para muchas personas, y además, son específicas para cada computadora, constituyendo el denominado lenguaje máquina. La programación de computadoras se realiza en los llamados lenguajes de programación que posibilitan la comunicación de órdenes a la computadora.

Un lenguaje de programación es una notación para escribir programas, a través de los cuales es posible la comunicación con el hardware, para dar así las ordenes adecuadas para la realización de procesos. Un lenguaje está definido por una gramática (Conjunto de reglas y símbolos para escribir programas).

En la figura 2.1 se presentan distintos niveles de acceso al Hardware mediante programación, teniendo en cuenta que por el único que se accede al hardware directamente es por el lenguaje máquina; por el resto se accede a una máquina virtual que considera el lenguaje del nivel en que se encuentra como su propio lenguaje máquina.

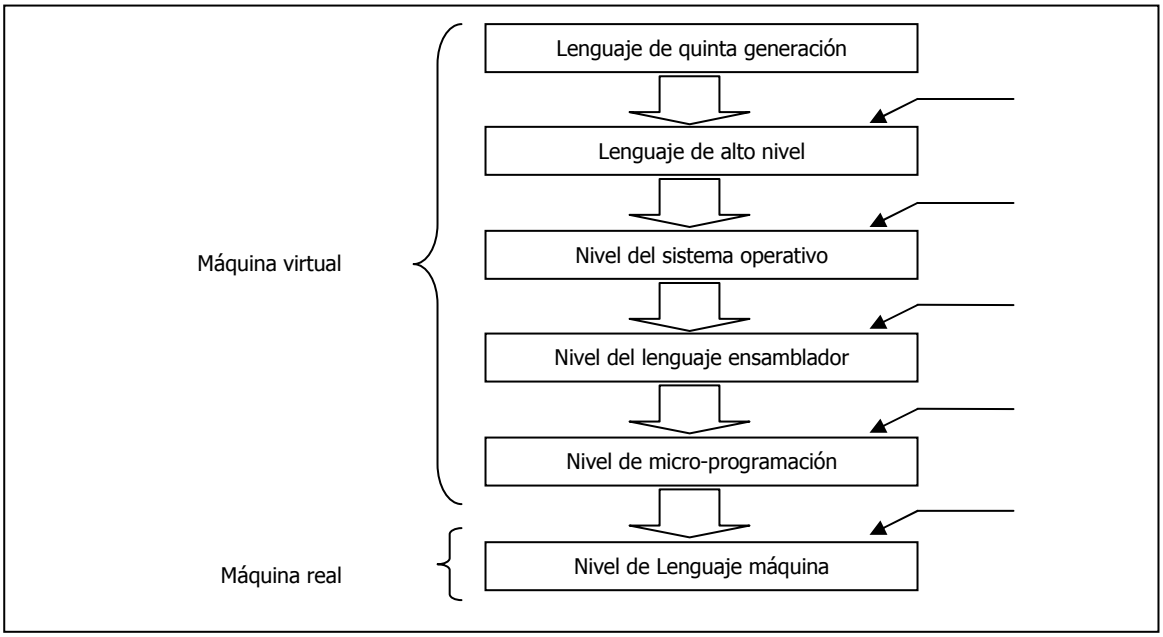


Figura 2.1 Niveles de programación.

La figura 2.2 muestra las diferentes capas del software en relación al hardware. Nótese que la capa que se comunica directamente con el hardware es la del "software del sistema", ya que todo el software de aplicación se ejecuta sobre ella.

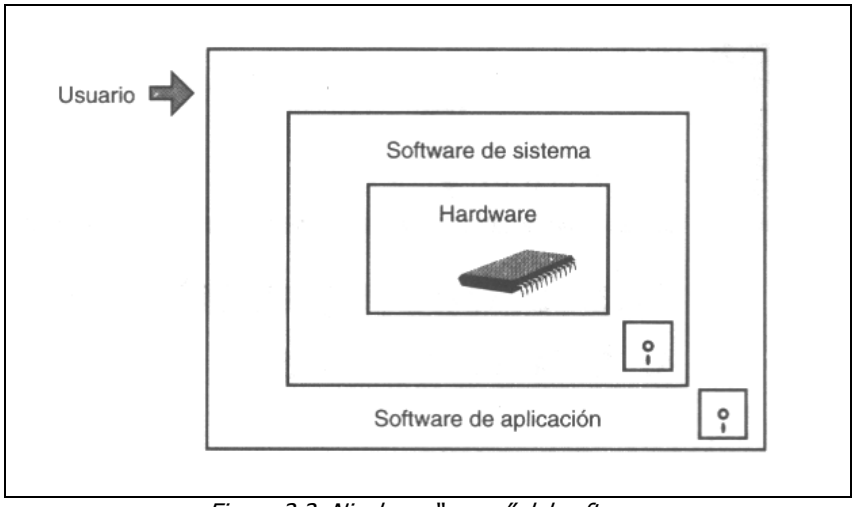


Figura 2.2 Niveles o "capas" del software

2.2 CLASIFICACION DE LOS LENGUAJES DE PROGRAMACION

- De acuerdo a su proximidad al lenguaje máquina o al lenguaje natural (humano) se pueden clasificar en:
  - Bajo nivel (máquina)
  - Intermedios (ensambladores)
  - Alto nivel (evolucionados)

- De acuerdo a la forma de trabajar de los programas y a la filosofía que fueron concebidos se pueden clasificar en:
  - Imperativos (Utilizan instrucciones como unidad de trabajo en los programas)
  - Declarativos (Utilizan descripciones de funciones o expresiones lógicas)
  - Orientados a objetos (Se basan mas en clases. Hay objetos, métodos, eventos, etc.)
  - Orientados al problema (Problemas muy específicos, generan aplicaciones)
  - Lenguajes naturales (Intentan aproximar sus instrucciones al lenguaje humano).

2.3 GENERACIONES DE LOS LENGUAJES DE PROGRAMACION

- Primera Generación.- Lenguajes máquina y ensambladores.
- Segunda Generación.- Primeros lenguajes de alto nivel imperativos (Fortran, Cobol)
- Tercera Generación.- Lenguajes de alto nivel imperativos (Algol, Pascal, PL/I)
- Cuarta Generación.- Orientados a aplicaciones y manejo de bases de datos (SQL)
- Quinta Generación.- Orientados a la inteligencia artificial y procesamiento de lenguaje natural (Lisp, Prolog).
- Generación orientada a objetos.- Con la proliferación de las interfaces gráficas de usuarios en los años 80
- Generación visual.- Exigencia de los usuarios de interfaces amigables en los años 90 (Visual Basic, Delphi)
- Generación internet.- Necesidad de manejar aplicaciones en diferentes plataformas dentro de internet (Java, XML, HTML, VRML, .NET)

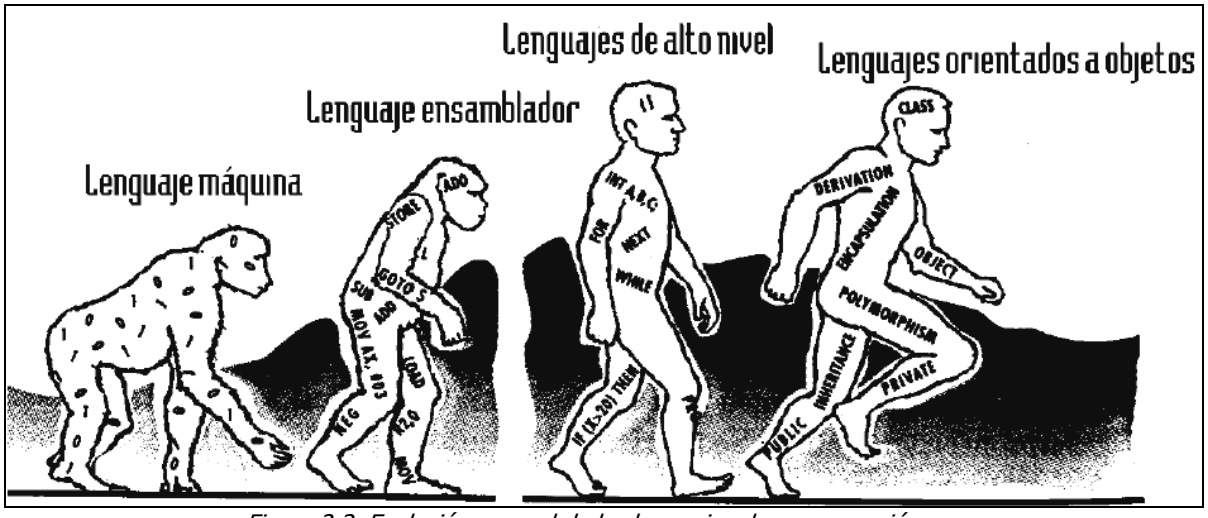


Figura 2.3 Evolución general de los lenguajes de programación

2.4 LENGUAJES DE ALTO NIVEL vs. LENGUAJES DE MAS BAJO NIVEL

Los lenguajes de alto nivel son aquellos que se encuentran en una capa superior a los ensambladores y lenguaje máquina. Algunas de las ventajas de los lenguajes de alto nivel son:

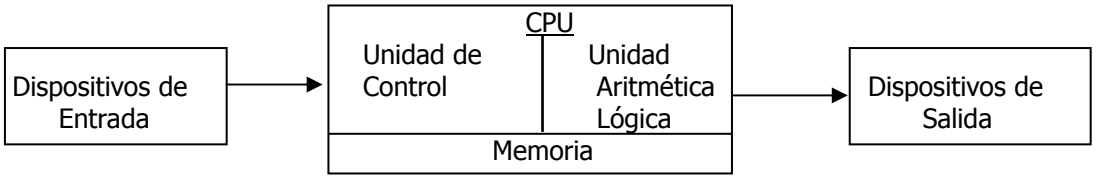
- Más fáciles de aprender, no se necesitan conocimientos hardware de la máquina, son prácticamente independientes de ésta.
- Liberación de ocupaciones rutinarias con referencias a instrucciones simbólicas o numéricas, asignaciones de memoria, etc. El traductor se encarga de ello.
- No se necesita saber la forma en que se colocan los diferentes tipos de datos en la memoria.
- Ofrecen una gran variedad de estructuras de control que facilitan la programación estructurada.
- Se depuran más fácilmente. Tienen construcciones que reducen ciertos tipos de errores de programación.
- Mayor capacidad de creación de estructuras de datos complejas, tanto estáticas como dinámicas. Los Lenguajes Orientados a Objetos permiten la reutilización de código.
- Permiten un diseño modular del código, división del trabajo y mayor rendimiento en desarrollo de aplicaciones.

Algunas desventajas de los lenguajes de alto nivel son:

- Mayor tamaño de los ejecutables.
- Mayores tiempos de compilación y de ejecución.
- No se tiene acceso a ciertas zonas del hardware, sólo posible con los de bajo nivel.

3. CONCEPTOS

3.1 ORGANIZACION FISICA DE UNA COMPUTADORA



Dispositivos de Entrada.- Sirven para introducir datos (información) en la computadora para su proceso. Los datos se leen de los dispositivos de entrada y se almacenan en la memoria interna. Ejemplos: teclado , scanners (digitalizadores de imagen), mouse (ratón), joystick (palancas de juego), lápiz óptico.

Dispositivos de Salida.- Entregan los datos procesados que sirven de información al usuario. Ejemplo: monitor, impresora.

La Unidad Central de Proceso (C.P.U) se divide en:

- **Unidad de Control.-** Coordina las actividades de la computadora y determina cuales operaciones se deben realizar y en que orden.
- **Unidad Aritmético – Lógica.-** Ejecuta las operaciones aritméticas y lógicas, tales como suma, resta, multiplicación, división y comparaciones.

La Memoria de la computadora se divide en:

- **Memoria Central o Interna.-** Según el tipo de acceso puede ser RAM o ROM.
  - La memoria **RAM (Randon Access Memory).-** Recibe el nombre de memoria principal, en ella se almacena información solo mientras la computadora esta encendida. Cuando se apaga o arranca nuevamente la computadora, la información se pierde, por lo que se dice que la memoria RAM es volátil.
  - La memoria **ROM (Read Only Memory).-** Es una memoria estática, la computadora puede leer los datos almacenados en la ROM, pero no puede introducir datos en ella, o cambiar los datos que ahí se encuentran; por lo que se dice que esta memoria es de solo lectura. Los datos de la ROM están grabados en forma permanente y son introducidos por el fabricante de la computadora.
- **Memoria Auxiliar o Externa.-** Es donde se almacenan todos los programas y datos del usuario. Los dispositivos de almacenamiento mas comúnmente utilizados son: disquettes, discos duros, CDs, Flash drives, etc.

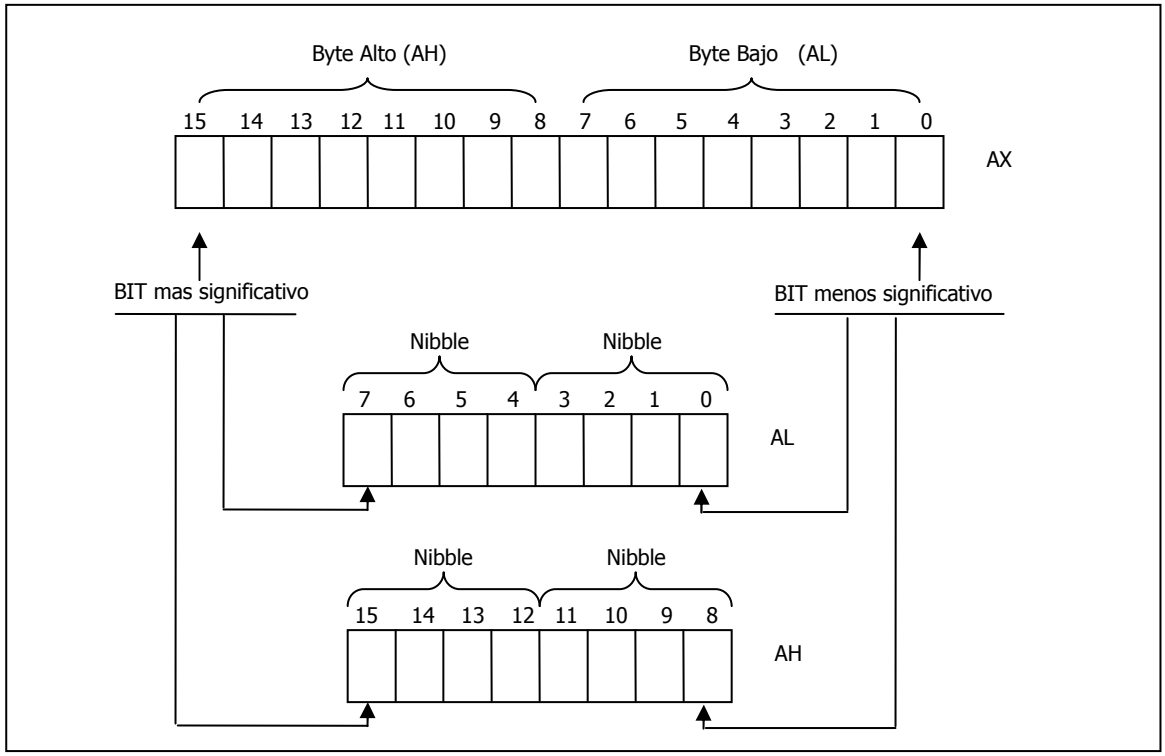
3.2 LOS REGISTROS DEL CPU

Toda la información almacenada dentro del CPU es procesada dentro de celdas llamadas registros. Un registro es un conjunto de 8 o 16 circuitos flip-flops también conocidos como basculadores o biestables, cuyo contenido es manipulado simultáneamente.

Los flip-flops son dispositivos electrónicos capaces de almacenar dos niveles de voltaje: Ya sea un voltaje bajo (típicamente 0 a 0.5 volts) que la PC interpreta como "cero" o "apagado"; o un voltaje alto (típicamente 5 volts) que es interpretado como estado "uno" o "encendido". Cada uno de estos estados es conocido como BIT (contracción de BInary digiT, que significa dígito binario).

A un grupo de 16 bits se le conoce como "word" (palabra) y dicha palabra puede dividirse en dos grupos de 8 bits conocidos como "bytes". Un grupo de 4 bits es conocido como "nibble".

El CPU contiene ciertos registros de 16 bits que también pueden ser utilizados como si fueran de 8 bits. Un ejemplo es el registro AX que tiene 16 bits (o flip-flops) y puede almacenar un número binario de 16 dígitos. Visto de otra forma, el registro AX tiene la capacidad de representar 65,535 números binarios entre 0000 0000 0000 0000 y 1111 1111 1111 1111 es decir, entre 2^0 y 2^{15}. El registro AX también puede utilizarse como dos registros separados de 8 bits cada uno (AH y AL).



4. SISTEMAS NUMERICOS

4.1 DEFINICIONES

SISTEMA NUMERICO.- Es el conjunto de símbolos y reglas que se utilizan para la representación de datos numéricos o cantidades. Ejemplos: Binario, Decimal, Octal, Hexadecimal.

BASE DE UN SISTEMA NUMERICO.- Es el número de símbolos distintos que utiliza el sistema numérico; además es el coeficiente que determina cual es el valor de cada símbolo dependiendo de la posición que ocupe.

Sistema numérico	Base	Símbolos
Binario	2	0, 1
Octal	8	0,1,2,3,4,5,6,7
Decimal	10	0,1,2,3,4,5,6,7,8,9
Hexadecimal	16	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

La equivalencia a decimal de algunos números es:

DECIMAL	BINARIO	OCTAL	HEXADECIMAL
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14

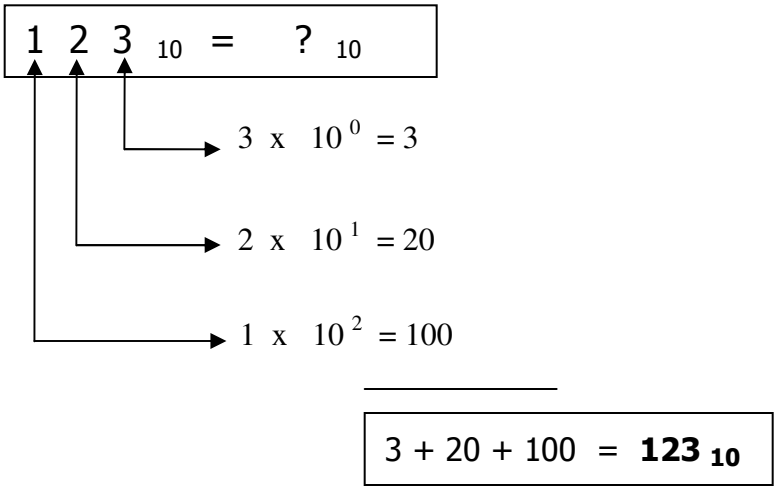
TEOREMA FUNDAMENTAL DE LA NUMERACION.- Relaciona una cantidad expresada en cualquier sistema de numeración con la misma cantidad expresada en el sistema decimal. Se utiliza para convertir de cualquier sistema numérico a decimal.

$$\sum_{n=0}^{\infty} D_n * B^n$$

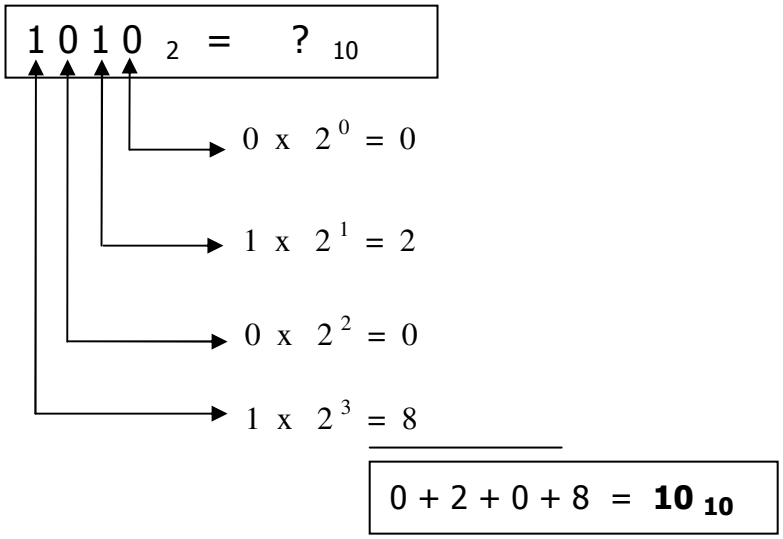
donde: ***D*** es el dígito, ***B*** es la Base y ***n*** es la posición que ocupa en la cantidad.

4.2 EJEMPLOS UTILIZANDO EL TEOREMA FUNDAMENTAL DE LA NUMERACION

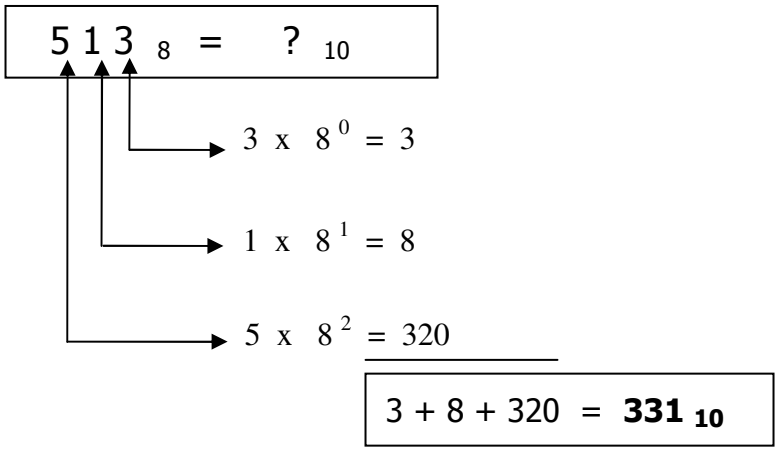
Ejemplo 4.1 Demostrar el valor decimal del número 123_{10}



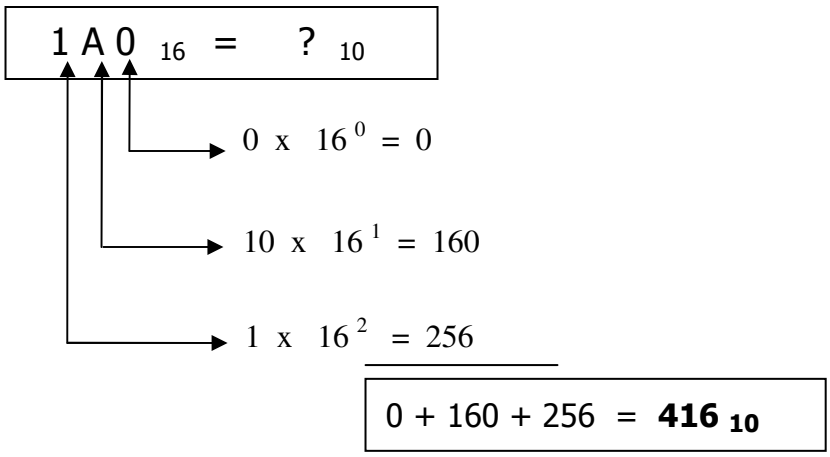
Ejemplo 4.2 Convertir 1010_2 a decimal



Ejemplo 4.3 Convertir 513_8 a decimal



Ejemplo 4.4 Convertir $1A0_{16}$ a decimal

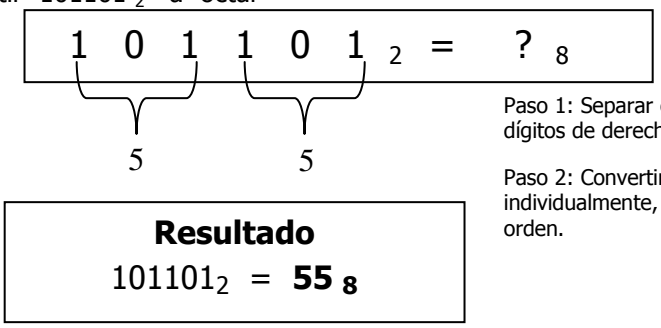


4.3 CONVERSIONES ENTRE OTROS SISTEMAS NUMERICOS

4.3.1 CONVERSION BINARIO A OCTAL

Un número octal tiene la característica que sus dígitos siempre se pueden representar en binario utilizando 3 posiciones. Para realizar la conversión Binario a Octal, separar la cifra en grupos de 3 dígitos de derecha a izquierda, convertir a octal cada grupo individualmente y mantener el orden de ellos.

Ejemplo 4.5 Convertir 101101_2 a octal



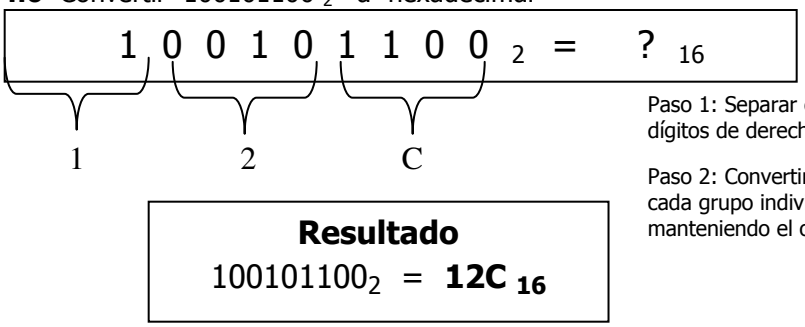
Paso 1: Separar en grupos de 3 dígitos de derecha a izquierda

Paso 2: Convertir a octal cada grupo individualmente, manteniendo el orden.

4.3.2 CONVERSION BINARIO A HEXADECIMAL

Un número hexadecimal tiene la característica que sus dígitos siempre se pueden representan en binario utilizando 4 posiciones. Para realizar la conversión Binario a Hexadecimal, separar la cifra en grupos de 4 dígitos de derecha a izquierda, convertir a hexadecimal cada grupo individualmente y mantener el orden de ellos.

Ejemplo 4.6 Convertir 100101100_2 a hexadecimal



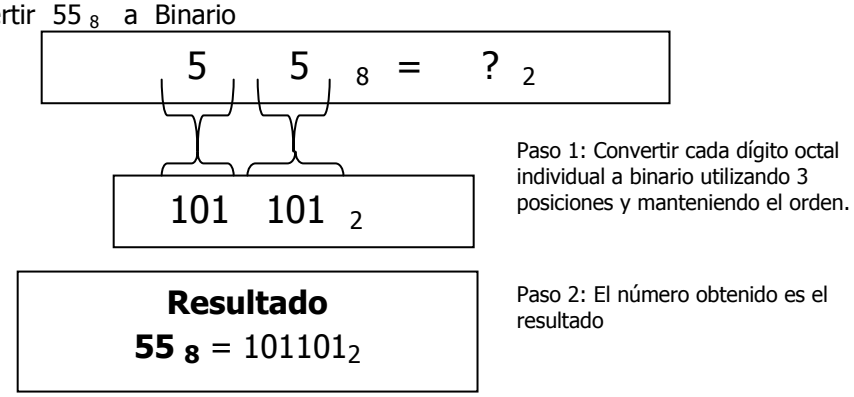
Paso 1: Separar en grupos de 4 dígitos de derecha a izquierda

Paso 2: Convertir a hexadecimal cada grupo individualmente, manteniendo el orden.

4.3.3 CONVERSION OCTAL A BINARIO

Para convertir un número de octal a Binario, se convierte cada dígito octal individual a binario utilizando 3 posiciones y manteniendo el orden de los dígitos.

Ejemplo 4.7 Convertir 55_8 a Binario



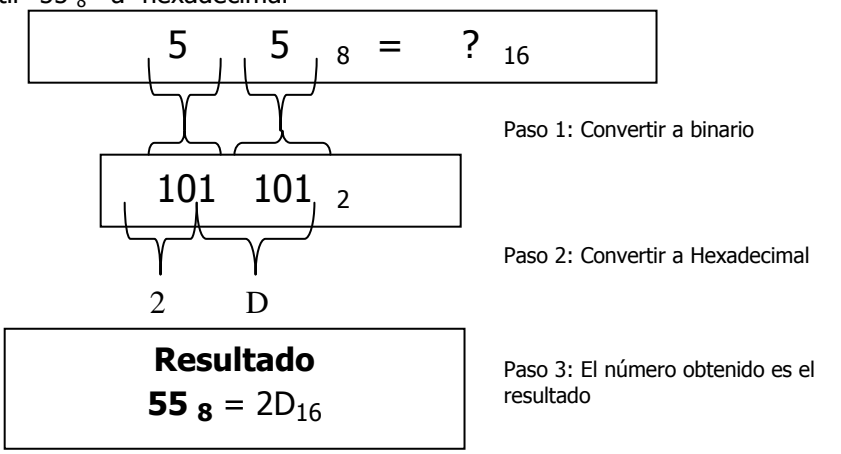
Paso 1: Convertir cada dígito octal individual a binario utilizando 3 posiciones y manteniendo el orden.

Paso 2: El número obtenido es el resultado

4.3.4 CONVERSION OCTAL A HEXADECIMAL

La conversión de octal a hexadecimal se realiza convirtiendo primero de octal a binario y posteriormente de binario a hexadecimal.

Ejemplo 4.8 Convertir 55_8 a hexadecimal



Paso 1: Convertir a binario

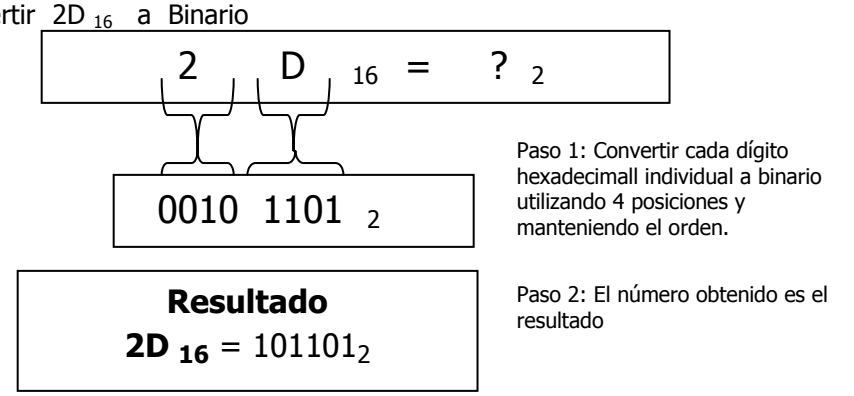
Paso 2: Convertir a Hexadecimal

Paso 3: El número obtenido es el resultado

4.3.5 CONVERSION HEXADECIMAL A BINARIO

Para convertir un número de hexadecimal a Binario, se convierte cada dígito hexadecimal individual a binario utilizando 4 posiciones y manteniendo el orden de los dígitos.

Ejemplo 4.9 Convertir $2D_{16}$ a Binario



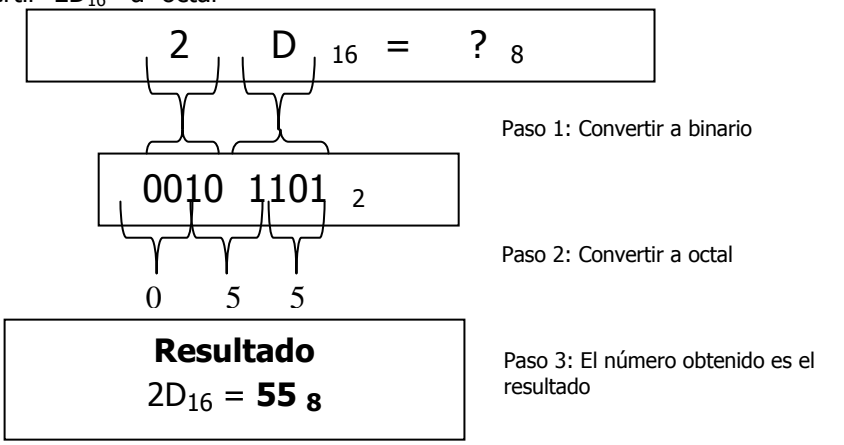
Paso 1: Convertir cada dígito hexadecimall individual a binario utilizando 4 posiciones y manteniendo el orden.

Paso 2: El número obtenido es el resultado

4.3.6 CONVERSION HEXADECIMAL A OCTAL

La conversión de hexadecimal a octal se realiza convirtiendo primero de hexadecimal a binario y posteriormente de binario a octal.

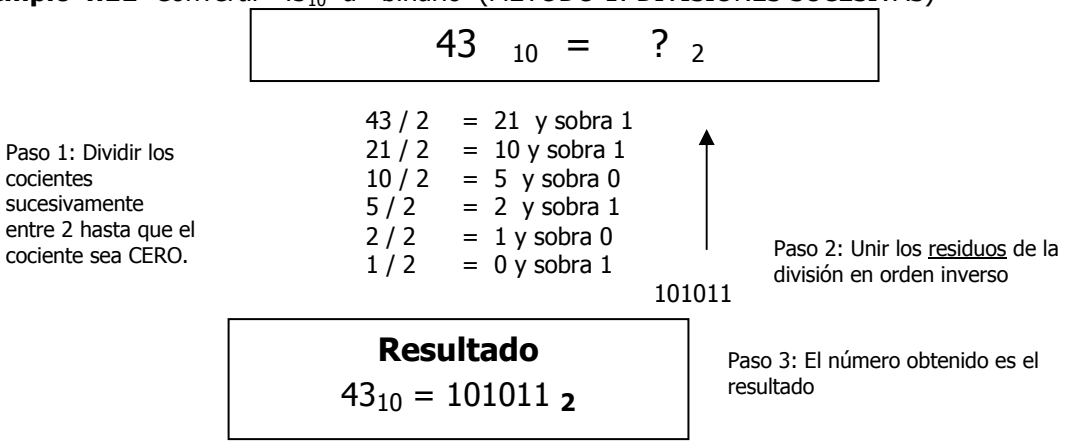
Ejemplo 4.10 Convertir $2D_{16}$ a octal



4.3.7 CONVERSION DECIMAL A BINARIO

Puede realizarse de dos maneras diferentes, llegando al mismo resultado:
En el método de divisiones sucesivas, se realizan divisiones sucesivas de la cantidad entre 2 y al final, los residuos de la división se acomodan desde el último al primero.
En el método de la tabla de posiciones, se obtiene el valor mas grande de la tabla que no exceda el valor original y se anota. Se resta este número del original y se vuelve a buscar en la tabla con el valor obtenido. El procedimiento se repite hasta llegar a cero. Finalmente se suman todas las cantidades binarias anotadas y ese será el resultado.

Ejemplo 4.11 Convertir 43_{10} a binario (METODO 1: DIVISIONES SUCESIVAS)



Ejemplo 4.12 Convertir 43_{10} a binario (METODO 2: USANDO LA TABLA)

n	2^n (Número Decimal)	Numero Binario
0	1	1
1	2	10
2	4	100
3	8	1000
4	16	10000
5	32	100000
6	64	1000000
7	128	10000000

Tabla 4.1 Tabla de posiciones binarias y su equivalente decimal.

$43_{10} = ?_{2}$

Paso 1: Ir tomando de la tabla el mayor número posible hasta completar la cantidad (Sin pasarse)

32	→	100000
8	→	1000
2	→	10
1	→	1
SUMA:		101011

Paso 2: Sumar los números

Resultado

$43_{10} = 101011_{\mathbf{2}}$

Paso 3: El número obtenido es el resultado

4.3.8 CONVERSION DECIMAL A OCTAL

Para realizar la conversión decimal a octal, se puede realizar primero la conversión decimal a binario y posteriormente de binario a octal.

EJEMPLO 4.13 Convertir 43_{10} a octal

$43_{10} = ?_{2}$

Paso 1: Convertir a binario (Ver ejemplos 4.10 y 4.11)

43 / 2 = 21 y sobra 1
21 / 2 = 10 y sobra 1
10 / 2 = 5 y sobra 0
5 / 2 = 2 y sobra 1
2 / 2 = 1 y sobra 0
1 / 2 = 0 y sobra 1

101011

$1\,0\,1\,0\,1\,1_{2} = ?_{8}$

5 3

Paso 2: Convertir de binario a octal

Resultado

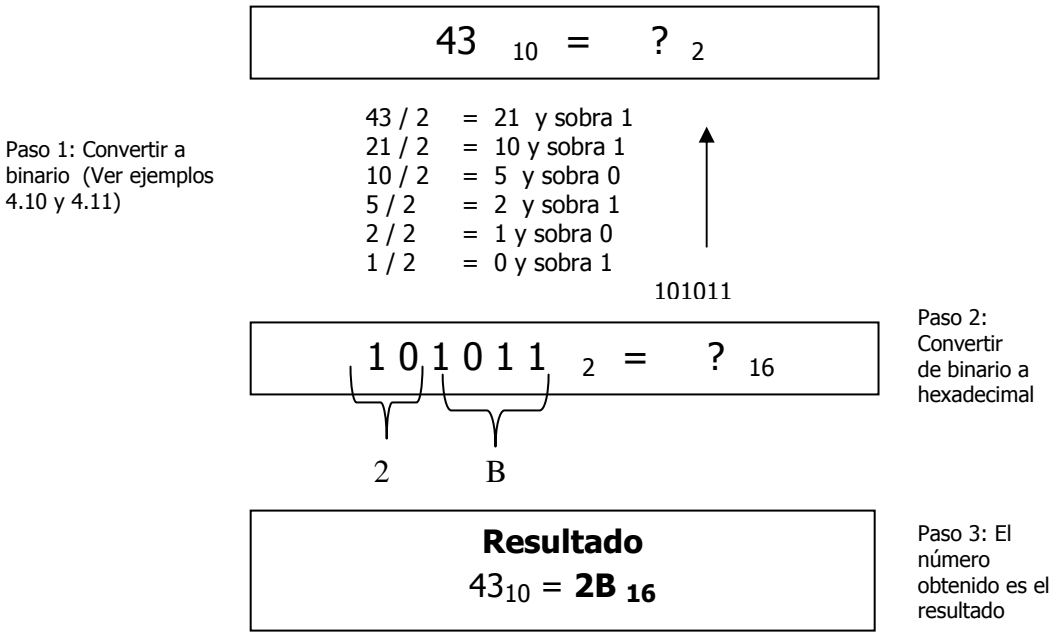
$43_{10} = \mathbf{53}_{\mathbf{8}}$

Paso 3: El número obtenido es el resultado

4.3.9 CONVERSION DECIMAL A HEXADECIMAL

Para realizar la conversión decimal a hexadecimal, se puede realizar primero la conversión decimal a binario y posteriormente de binario a hexadecimal.

EJEMPLO 4.14 Convertir 43_{10} a hexadecimal



4.4 OPERACIONES BINARIAS

4.4.1 SUMA BINARIA

Para realizar la suma binaria se suma dígito por dígito de derecha a izquierda de acuerdo al siguiente criterio:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0 con Carry 1

EJEMPLO 4.15 Sumar los números 101_2 y 11_2

	1	0	1	→	5_{10}
+		1	1	→	+ 3_{10}
1	0	0	0	→	8_{10}

RESULTADO: $101_2 + 11_2 = 1000_2$

4.4.2 RESTA BINARIA

La resta binaria puede realizar sumando al primer número, el complemento a dos del segundo.

El complemento a dos es la representación del número binario negativo; se obtiene primero cambiando todos los unos por ceros y los ceros por unos, y después sumando uno al resultado.

EJEMPLO 4.16 Realizar la resta: $11101_2 - 111_2$

Paso 1: Obtener el complemento a dos del segundo número (el negativo):

0	0	1	1	1	Número original (Con el mismo número de dígitos que el primero)
1	1	0	0	0	Se cambian ceros por unos y unos por ceros
			+	1	Se suma uno
1	1	0	0	1	El número obtenido es el complemento a dos

Paso 2: Se suma el primer número y el complemento a dos obtenido en el paso anterior:

```
      1   1   1   0   1
+     1   1   0   0   1
-------------------------
1     1   0   1   1   0
```

Ese bit se ignora.

Resultado : $11101_2 - 111_2 = 10110_2$

4.4.3 MULTIPLICACION BINARIA

Se realiza de manera similar al sistema decimal: dígito por dígito, de derecha a izquierda pero con los siguientes criterios:

0 * 0 = 0
0 * 1 = 0
1 * 0 = 0
1 * 1 = 1

EJEMPLO 4.17 Realizar la multiplicación $101_2 * 10_2$

```
      101 2
*      10 2
-----------
      000
     101
-----------
     1010 2
```

4.4.4 DIVISION BINARIA

Se realiza de manera similar al sistema decimal.

EJEMPLO 4.18 Realizar la división $100010_2 / 110_2$

```
        101
    ------
110 | 100010
    | -110
    ------
      001010
       -110
      ------
        0100
```

Resultado: 101_2 y sobran 100_2

5. REPRESENTACION DE DATOS EN LA PC

El almacenamiento de datos en la memoria de la PC se realiza siguiendo un orden específico: el byte menos significativo primero y después el mas significativo (ver Figura 4.1).

Existen diferentes maneras de almacenar datos en la PC. Tres de las mas importantes son: ASCII, BCD y Representación de punto flotante.

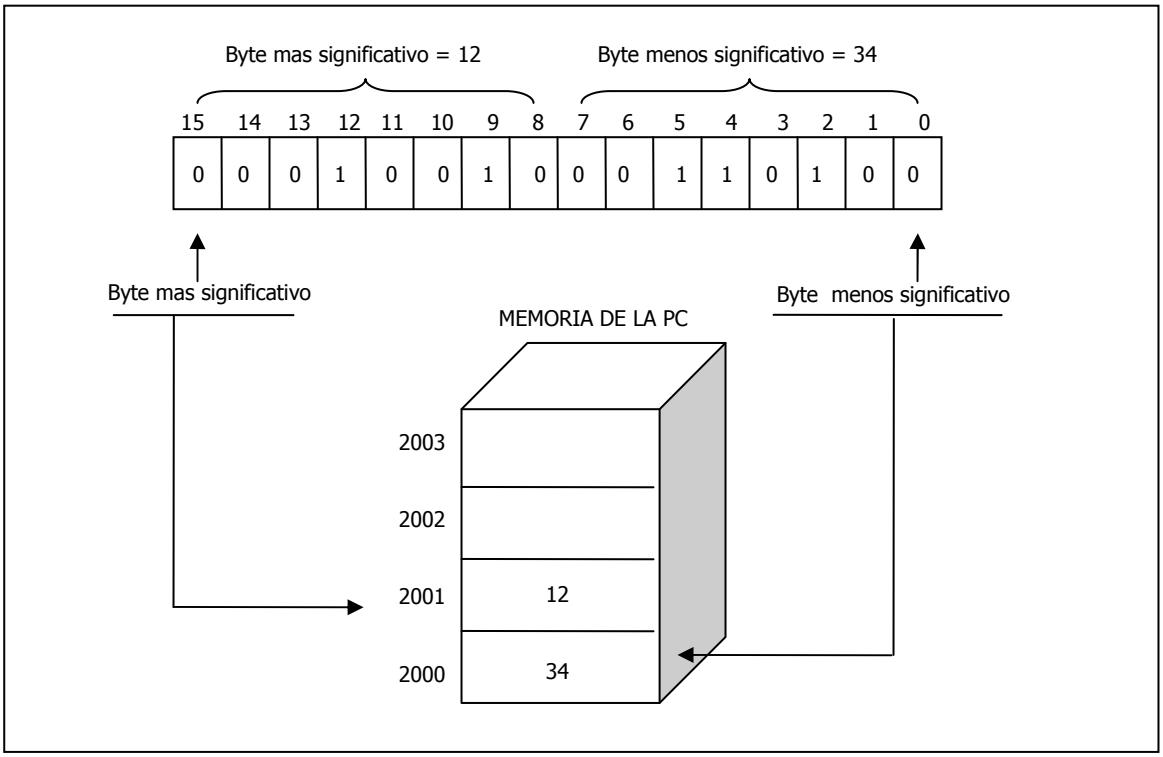


Figura 4.1 Almacenamiento en la memoria de la PC

- **ASCII** (American Standard Code for Information Interchange) asigna un número binario de 7 bits a las letras del alfabeto, a los dígitos y a los símbolos especiales (mantiene el bit 8 en cero).
  Por ejemplo, para almacenar el mensaje "HOLA" en realidad se almacenaría: 48 4F 4C 41 siendo estos números las representaciones hexadecimales de las letras respectivas en ASCII.
  Para almacenar números, se utiliza un byte para cada número; por ejemplo: Al almacenar 678 los datos guardados serían: 36 37 38.
  El formato ASCII se usa principalmente para representar texto; y aunque también se pueden representar números, el manejo de éstos no se realiza en forma eficiente.

- **BCD** (Binary Coded Digital) utiliza grupos de 4 bits para representar cada dígito decimal del 0 al 9, siendo 0 = 0000, 1= 0001, 2 = 0010, …. , 9 = 1001. La notación BCD no usa en forma eficiente la memoria como lo hace la representación binaria pura. Por ejemplo, un byte puede representar en binario un valor de hasta 255 mientras que en BCD solo se puede representar hasta el valor de 99.
  El formato BCD se utiliza para representar números en aplicaciones mercantiles y facilitar a la vez la operación sobre los mismos evitando errores de redondeo.

EJEMPLO 5.1 Representar el número 2567 en BCD

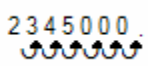
2	5	6	7
0010	0101	0110	0111

Resultado: El número 2567 en BCD es el: 0010 0101 0110 0111

- El **FORMATO DE PUNTO FLOTANTE** es la forma mas flexible de representar números, ya que permite a la computadora manejar números grandes y chicos. Se basa en la notación científica. La idea es representar el número en dos partes: Su mantisa (un número entre uno y diez) y su exponente (que indica donde colocar el punto decimal).

  EJEMPLO 5.2 Representar el número 2345000 con formato de punto flotante

  Resultado: $2.345 * 10^{6}$
  ya que se debe recorrer el punto decimal 6 posiciones a la derecha para obtener el número:

  EJEMPLO 5.3 Representar el número 0.0000345 con formato de punto flotante.

  Resultado: $3.45 * 10^{-5}$

  ya que se debe recorrer el punto decimal 5 posiciones a la izquierda para obtener el número: