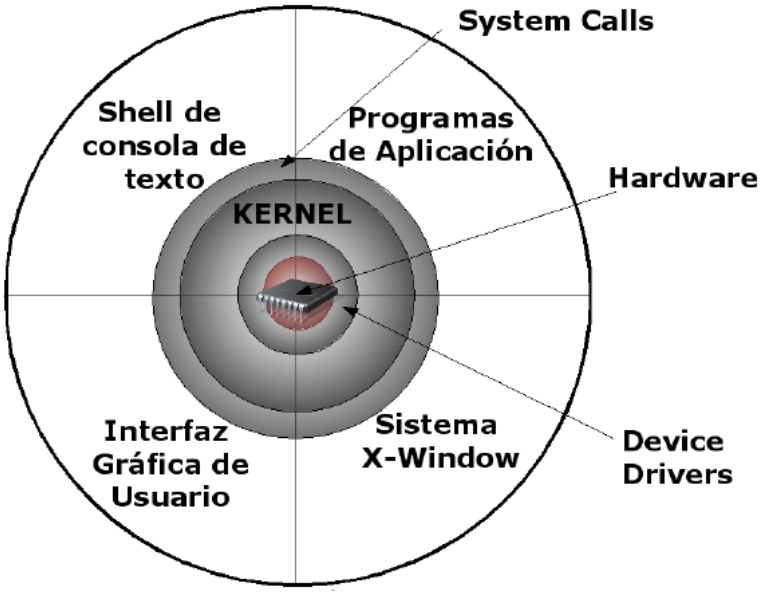
Que es un sistema operativo?: Un sistema operativo es una colección de programas que se encargan de administrar los recursos del computador, proveyendo a los diferentes usuarios que pueden estar logueados al sistema la posibilidad de utilizar los recursos de hardware y software del mismo sin necesidad de conocer detalles sobre ellos y de manera segura para sí y para el resto.

API (Aplication Programming Interface): serie de servicios o funciones que el Sistema Operativo ofrece al programador y a través de éstas acceder a los recursos de hardware del sistema. La implementación de las API dentro del Sistema Operativo se conocen como ***System Calls.***

Kernel: El conjunto de programas que constituyen el Sistema Operativo propiamente dicho y la implementación de las System Calls se denomina kernel (núcleo). (NOTA wiki: Es el principal responsable de facilitar a los distintos programas acceso seguro al harware de la computadora o en forma básica, es el encargado de gestionar recursos. También se encarga de decidir qué programa podrá hacer uso de un dispositivo de hardware y durante cuánto tiempo).

Shell: Es una interfaz de usuario. El Shell es un intérprete de comandos que se encarga de traducir los pedidos del usuario e invocando las System Calls apropiadas los lleva adelante. Esta interfaz se la conoce como modo consola. **EXTRA:** Cada vez que tipeamos un comando el shell lo asume como un ejecutable binario o como un script. Un ejecutable binario es el resultado de la edición y compilación y linkeo de un programa determinado. Un script es un archivo de texto que contiene comandos binarios u otros scripts para ejecutar, y que además puede incluir sentencias de control de flujo.

Device Drivers (Manejadores de Dispositivos): Para acceder al Hardware, el sistema operativo posee una interfaz de muy bajo nivel llamada *Device Drivers*. Estos componentes de software, son parte del kernel y su misión es acceder al hardware de sistema en forma directa. Las aplicaciones NO PUEDEN efectuar este acceso en los sistemas operativos modernos ya que se trata de entornos multiusuario en los que la estabilidad del sistema y de las aplicaciones que se ejecutan dentro de él debe estar garantizada.



File system: Un file system es un conjunto de políticas definidas para la organización de la información en archivos dentro de un medio de almacenamiento, de modo de permitir definir su ubicación dentro de ese medio y posibilitar el acceso a los datos contenidos por éstos de manera simple. El *file system manager* es una pieza de software de un sistema operativo que implementa las políticas definidas para el file system. JEEMPLOS de file system: discos duros, los CD's, DVD's. El file system manager es el responsable de organizar los sectores y pistas de "los discos, cd's...." en archivos, directorios, enlaces, etc, y mantener actualizada la base de datos de los sectores que corresponden a cada archivo, directorio, enlace, etc.

Procesos: Un proceso es (intentando ensayar una definición simplificada) una instancia de ejecución de un programa, EXTRA: Los procesos son programas que se ejecutan en un momento dado. como ejemplo el comando ls. Pueden existir en un momento varias personas ejecutando ls en forma simultánea. Cada instancia de ejecución de ls es un proceso diferente. (con el comando ps se ven los ProcesoS). Cada vez que ejecutamos un ls para ver un directorio el sistema crea un proceso.

Archivos ".h":Estos archivos, contienen definiciones de funciones, macros y variables.

El compilado: El proceso de compilación involucra cuatro etapas sucesivas: preprocesamiento, compilación, ensamblado y enlazado. Para pasar de un programa fuente escrito por un humano a un archivo ejecutable es necesario realizar estas cuatro etapas en forma sucesiva.

1. ***Pre procesamiento:*** En esta etapa se interpretan las directivas al preprocesador. Entre otras cosas, las variables inicializadas con #define son sustituídas en el código por su valor en todos los lugares donde aparece su nombre. El preprocesado puede pedirse haciendo: *gcc -E circulo.c > circulo.pp* y examinarlo haciendo: *more circulo.pp*
2. ***Compilación:*** La compilación transforma el código C en el lenguaje ensamblador propio del procesador de nuestra máquina.
   gcc -S circulo.c realiza las dos primeras etapas creando el archivo circulo.s; examinándolo con *more circulo.s* puede verse el programa en lenguaje ensamblador.
3. ***Ensamblado:*** El ensamblado transforma el programa escrito en lenguaje ensamblador a código objeto, un archivo binario en lenguaje de máquina ejecutable por el procesador. *gcc -c circulo.c* *gcc -c -o circulo.o circulo.c* *gcc -c circulo.c -o circulo.o*Cualquiera de estas lineas de codicos crea el archivo en código objeto circulo.o a partir de circulo.c (Realiza primero los 2 pasos anteriores). Puede verificarse el tipo de archivo usando el comando *$ file circulo.o* ***NOTA:*** En los programas extensos, donde se escriben muchos archivos fuente en código C, es muy frecuente compilar cada archivo fuente por separado, y luego enlazar todos los módulos objeto creados. Estas operaciones se automatizan colocándolas en un archivo llamado makefile, interpretable por el comando make, quien se ocupa de realizar las actualizaciones mínimas necesarias toda vez que se modifica alguna porción de código en cualquiera de

los archivos fuente.

4. ***Enlazado (linker):*** Las funciones de C/C++ incluídas en nuestro código, tal como printf() en el ejemplo, se encuentran ya compiladas y ensambladas en bibliotecas existentes en el sistema. Es preciso incorporar de algún modo el código binario de estas funciones a nuestro ejecutable (los combina o linkea). En esto consiste la etapa de enlace, donde se reúnen uno o más módulos en código objeto con el código existente en las bibliotecas. *$ gcc -o circulo circulo.o* crea el ejecutable circulo a partir del objeto.

Libreria: Una librería de programas no es otra cosa que un archivo que contiene código y datos compilados y funcionales, que serán incorporados a otros programas cuando estos los requieran. Las librerías de código facilitan la reutilización de código evitando tener que re escribirlos cada vez que los necesitamos y a su vez permiten el trabajo en equipo. una librería se compone de:

- Prototipos de las funciones (archivo/s cabecera/s .h)
- Definición de las funciones (archivos/s fuente .c)

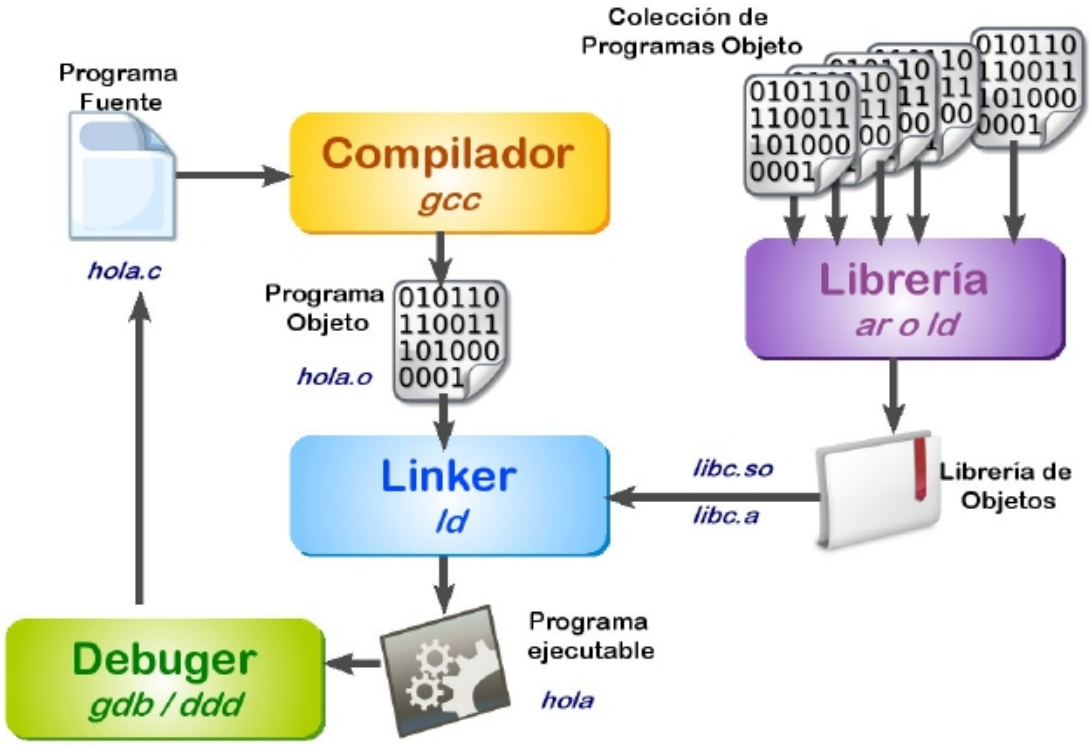
Tipos de libreria:

1. ***Estatica:*** Son simples colecciones de programas objeto agrupados en un único archivo cuyo nombre típicamente finaliza en '.a'. Al compilar un programa que hace uso de código contenido en una librería estática, en el momento de realizar la fase de enlace (link), se copia en nuestro programa el código objeto de la librería, es decir que no es necesario distribuir nuestro programa con la librería ya que ésta se encuentra incrustada de nuestro nuevo programa. Para generar una librería estática utilizamos el utilitario ar. Antes de ello obviamente debemos haber generado el programa objeto, compilando con la opción -c. EJ: $ gcc -c holalib.c Creamos objeto en ese paso..... $ ar rcs libhola.a holalib.o Creamos la libreria. Las librerías en general llevan en su nombre el prefijo “lib”. Por lo tanto el nombre a los efectos del linker es lo que sigue a “lib” y precede a “.a”.
2. ***Compartida (shared):*** Estas librerías no se linkean al programa que llama a funciones empaquetadas en ellas, sino que se resuelven las referencias en el momento de arranque del programa en cuestión. Si en el momento de la carga de nuestro programa, las librerías necesarias no están ya cargadas en memoria, el dynamic loader efectuará la carga simultánea a memoria del programa en sí que va a componer el proceso disparado por el usuario, junto con las librerías que necesita.
3. ***De carga dinamica (DL, por Dynamic Loaded):*** Estas librerías se cargan en momentos diferentes de la carga y ejecución delprograma. Su principal utilidad es la implementación de módulos, o plug-ins, ya que estos elementos de software se cargan cuando se invocan durante la ejecución de un programa. Desde el punto de vista de su formato no tienen diferencias en Linux con respecto a como se construyen librerías compartidas o programas objeto. Sin embargo hay diferencias en el código que se necesita escribir en la aplicación para trabajar con estas librerías. Es decir que el programador de la aplicación debe incluir funciones específicas que hasta ahora en los modelos analizados no se requieren.

Básicamente necesitamos invocar cuatro funciones:

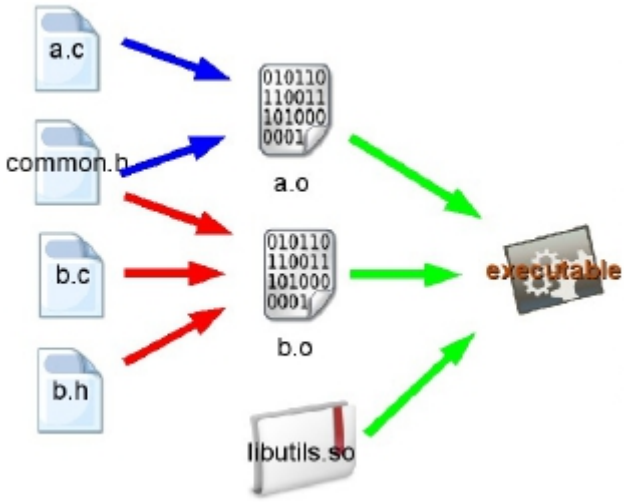
- **dlopen ():** Carga una librería y la prepara para su uso.
- **Dlerror ():** Retorna un string que describe el error generado por las demás funciones de manejo de librerías dinámicas.
- **Dlsym ():** Busca el valor de un símbolo presente en una librería ya abierta con dlopen ().
- **dlclose ():** Cierra la librería abierta con dlopen().

Desarrollo de un programa:



Make: make es una herramienta que permite ejecutar una secuencia de procesos. Utiliza un script, llamada comúnmente makefile. Es capaz de determinar automáticamente cuales pasos de una secuencia deben repetirse debido al cambio en algunos de los archivos involucrados en la construcción de un objeto, o en una operación, y cuales no han registrado cambios desde la última vez de modo que no es necesario repetirlos.

Logica del make (según desarrollo de un programa): Cada programa objeto depende de su correspondiente programa fuente, y headers si los hubiere. Un ejecutable dependerá de los programas objeto involucrados en su proyecto, mas las librerías de uso en el mismo.



Tenemos tres grupos claramente diferenciados por los colores de las flechas. A la hora de armar un makefile necesitamos tener claro el mapa de dependencias.
Una vez armado y en funcionamiento el makefile, si modificamos el archivo b.c, no importa el motivo, make detectará la actualización pero en lugar de repetir todas las compilaciones y linkeos, solo recompilará al archivo b.c, lo cual modificará el objeto b.o, y por lo tanto deberá relinkear la aplicación para llegar a la versión de executable que contenga los cambios hechos en b.c. Lo interesante es que no tocó el resto. Es decir lo que no cambia se deja tal como está. ***Ejemplo Make anterior:***

```
excecutable: a.o b.o
     gcc a.o b.o -o executable -lutils
a.o: a.c
     gcc -c -o a.o a.c
b.o: b.c
     gcc -c -o b.o b.c
clean:
     rm -f ./*.o
     rm -f executable
```

Lo que vemos al inicio de la línea como un nombre terminado con ':' se denomina regla. Si no le especificamos nada y solo tipeamos make a secas, make asumirá que la regla a ejecutar es solo la de la primer línea con esta característica. En nuestro primer ejemplo executable.
Si la regla a continuación del carácter ':' tiene dependencias, éstas deberán corresponder dentro del makefile a otras reglas que se escriban a continuación de la regla dependiente. En nuestro caso excecutable es dependiente de a.o y b.o.
Por lo tanto debe necesariamente existir una regla para a.o y otra para b.o escritas luego de la regla dependiente. Estas líneas están a continuación con sus respectivas dependencias.
Finalmente hemos escrito una regla que permita limpiar todos los archivos generados a partir de los fuentes. Esto puede resultar útil para hacer una recompilación general. Se ejecuta haciendo: “$ make clean” Pasando por alto las demas reglas y ejecutando

solo la regla clean. El formato general de cada regla es:

dependiente: dependencia

comando para generar el dependiente a partir de la dependencia.

Las líneas que contienen comandos empiezan con un tabulador. De otro modo no funciona.

Variables en make: Se colocan al principio del codigo, igualando la constante varaible al comando a reemplazar, y se accede al valor poniendo $ (VARIABLE).

```
CC=gcc
CFLAGS=-c -g
LDFLAGS=-g -lutils
OBJS=a.o b.o
excecutable: $(OBJS)
     $(CC) $(OBJS) -o executable $(LDFLAGS)
a.o: a.c
     $(CC) $(CFLAGS) -o a.o a.c
b.o: b.c
     $(CC) $(CFLAGS) -o b.o b.c
clean:
     rm -f ./*.o
     rm -f executable
```

Unidades basicas:

- BIT[b]: Menor unidad de información, puede valer 0 o 1.
- BYTE[B]: Conjunto de 8 bits.

■

El modelo de Von Newmann: Propone:

- Memoria: Los datos y programas se almacenan en una misma memoria de lectura-escritura.
  - Los contenidos de esta memoria se direccionan indicando su posición sin importar su tipo.
- *Ejecución:* Procesamiento secuencial de instrucciones (salvo que se indique lo contrario).
- *Información:* Todo esto usando Datos binarios.

En la versión actual de las arquitecturas de las computadoras con programa almacenado cumplen mínimamente con las siguientes características:

- *Tres sistemas de harware* (interconectados por Buses):
  - Unidad de Procesamiento Central (CPU).
  - Un sistema de memoria principal.
  - Un sistema de entrada y salida (I/O).
- *Capacidad para realizar el procesamiento secuencialmente.*
- *Un único camino (físico o lógico) entre memoria y CPU* (cuello de botella de la arquitectura Von Newmann).

Unidad de Procesamiento Central (CPU): su misión consiste en coordinar y controlar o realizar todas Las operaciones del sistema. Sus partes principales son las siguientes:

- Unidad de control: implementa la maquina de estados.
- Unidad Aritmético Lógica (ALU).
- La Memoria Central (MC) o Registros (pequeña área de memoria y el contador

de programa).

Sistema de memoria principal: Almacena datos y programas.

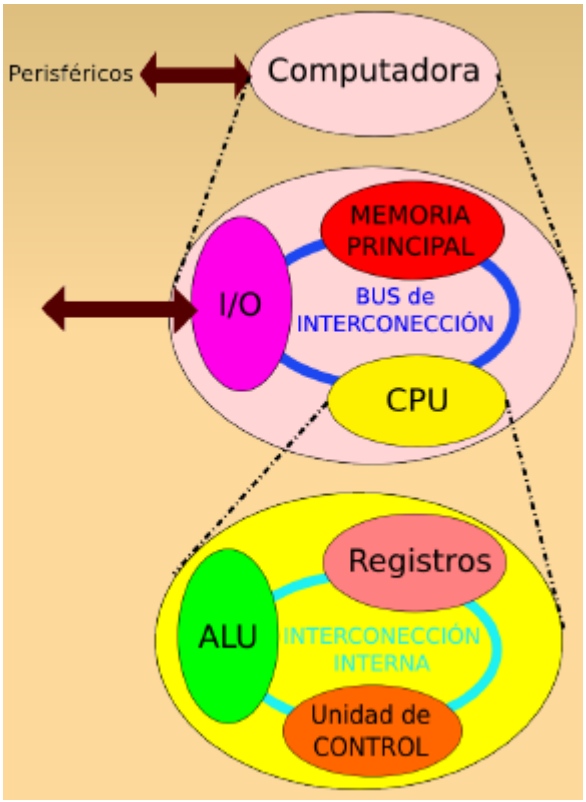
Sistema de entrada y salida: Comunicación con el mundo exterior.

La unidad de procesamiento central el sistema de memoria principal y el sistema de entrada y salida están conectados entre si por un sistema de interconexiones (buses).

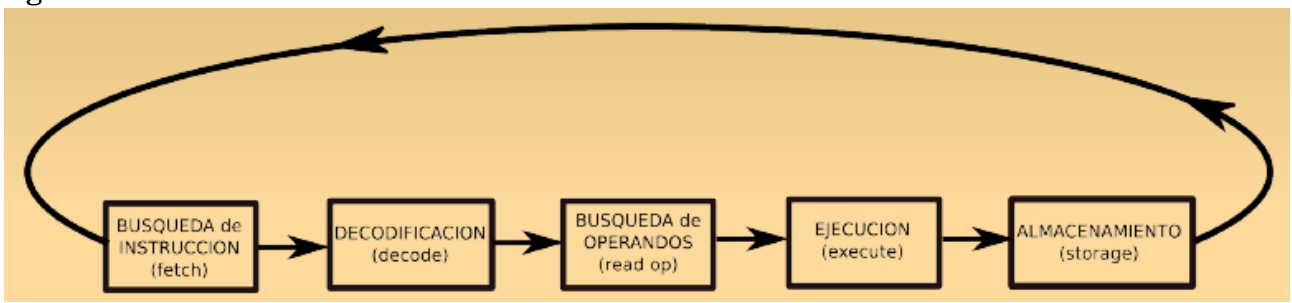
Un típico ciclo completo seria:

1) La unidad de control recupera la "siguiente" instrucción de programa de la memoria principal (utilizando el contador de programa: IP)
2) La instrucción se decodifica.
3) Se toman de memoria los operandos a continuación de la instrucción y se colocan en los registros.
4) La ALU realiza la operación pedida y se coloca el resultado en los registros o en memoria.

VER IMAGEN A CONTINUACION.



Un procesador es una maquina secuencial que esta en un ciclo infinito como el de la figura:



Sistema decimal:

- Necesita 10 símbolos: (0 1 2 3 4 5 6 7 8 9).

- El valor del número viene dado por la suma de cada dígito multiplicado por su "peso".
  - Por ejemplo: $123{,}4 = 1\cdot 10^2 + 2\cdot 10^1 + 3\cdot 10^0 + 4\cdot 10^{-1}$

Sistema binario:

- Necesito 2 símbolos: (0 1).
- El valor del número viene dado por la suma de cada dígito multiplicado por su "peso".
  - Por ejemplo: $10110_2 = 1\cdot 2^4 + 0\cdot 2^3 + 1\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0 = 16+0+4+2+0 = 22_{10}$

Sistema octal:

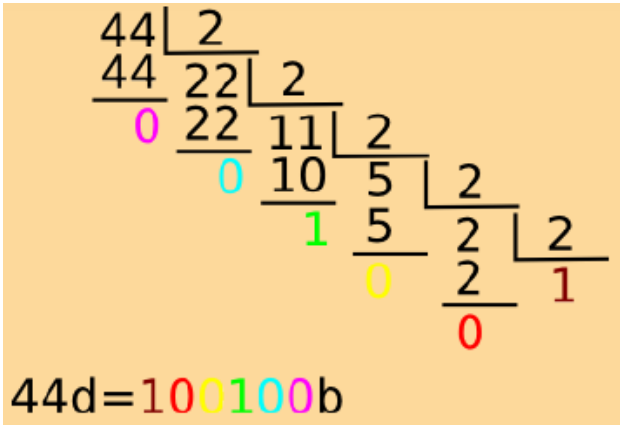
- Necesito 8 símbolos (0 1 2 3 4 5 6 7).
- El valor del número viene dado por la suma de cada dígito multiplicado por su "peso".
  - Por ejemplo: $377_8 = 3\cdot 8^2 + 7\cdot 8^1 + 7\cdot 8^0 = 192+56+7 = 255 \text{ --> } (255 = 4\cdot 8^2 - 1)$

Sistema hexadecimal:

- Necesito 8 símbolos (0 1 2 3 4 5 6 7 8 9 A B C D E F).
- El valor del número viene dado por la suma de cada dígito multiplicado por su "peso".
  - Por ejemplo: $ABC_8 = 10\cdot 16^2 + 11\cdot 16^1 + 12\cdot 16^0 = 2560+176+12 = 2748_{10}$

PASAJES:

$44_{10} = 100100_2$:



Se divide el número a convertir por la base a convertir, hasta que el cociente de un número menor que dicha base. El resultado se compone del último cociente y los restos tomados en sentido inverso a la sucesión de cocientes.

En el caso de los decimales se multiplica por la base y la parte entera es el dígito buscado.

0,375x2=**0**,75 => 0,**0**

0,75x2=**1**,5 => 0,0**1**

0,5x2=**1** => 0,01**1** $2{,}375_{10} = 10{,}011_2$

Binario a octal:

Agrupamos de a 3 bits

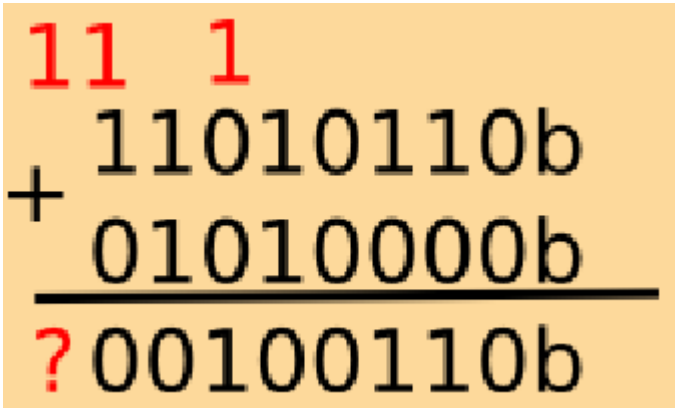
1	1	0	0	1	0	1	1	1	0
1		4			5			6	

Binario a hexa:

Agrupamos de a 4 bits

1	1	0	0	1	0	1	1	1	0
3			2			E			

Si supero el numero maximo de digitos que manejo al sumar tendre problemas, a esto se lo llama “carry”:



- **Carry** Flag (CF) (Bit 0)- Acarreo. Indica con 1 un desborde no signado. Ej: bytes 255 + 1 (el resultado no está en el rango 0...255).
- **Overflow** Flag (OF) (Bit 11)- Desborde. Indica con 1 un desborde signado. Ej: bytes 100 + 50 (resultado no esta en rango de -128...127).

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

Sumar y restar dentro del conjunto de los ENTEROS nos trae un nuevo problema a la hora de hacerlo en una PC: ¿Cómo representar números negativos? Para esto analizamos 4 puntos:

- Signo y Magnitud

- Complemento a 1
- Complemento a 2
- Binario Desplazado

Signo y magnitud: utiliza el bit mas significativo (MSB) para representar el signo y el resto para el modulo. Así, trabajando con 4 bits tenemos que:

- 0010_2 equivale a 2.
- 1010_2 equivale a -2.

Inconvenientes de este metodo:

- Se consumen 2 valores para representar al cero **0**000b=**1**000b (0=-0)
- No se puede usar el mismo HARDWARE que suma números positivos para sumar números negativos. 2+(-1)=1 0010b+1001b=1011b (-3)
- Para realizar una suma primero determinar si los dos números tienen el mismo signo.
  - Mismo signo:sumar parte significativa
  - Distinto signo: restar el mayor del menor y asignar el signo del mayor.

Complemento a 1: Para representar un número negativo se invierte cada bit por su complemento (1 en 0 y viceversa).

$0111_2 = 7_{10}$ Complemento a 1: **1**000b = **-**0111b = **-**7d (Mismo problema con el "0").

Si ocurre un carry al final de la adición/resta, sumarlo al resultado obtenido (end-around carry)

ej:

- 0010b+0100b=0110b (2+4=6)
- 0010b+1110b=^{1}0000b+1=1 (2+(-1)=1)
- 0010b+1100b=1110b=-1 (2+(-3)=-1)
- 1101b+1011b=^{1}1000b+1=-6 (-2+(-4)=-6)
- 0110b+0110b=1100b=-3 (6+6=12 => fuera de escala)
- 1001b+1001b=^{1}0010b+1=3 (-6-6=-12 => fuera de escala)

El resultado esta en complemento a uno, por lo que si la respuesta dio negativa se invierten los bits. Al sumar mismos signos, si me dio como respuesta un signo opuesto entocnes me fui de escala (los ultimos 2 casos).

Complemento a dos: Para representar un número negativo se invierte cada bit por su complemento y se le suma 1 (C2=C1+1).

0111b=7d

0000b=0d

ahora al ser negativo se hace el complemento (teniendo en cuenta C2 = C1 + 1)

1111b=**-**0000b+1=**-**1d

1000b=**-**0111b+1=**-**8d

No requiere ajuste al sumar 2 números de distinto signo. (el carry no se suma, tener

en cuenta nuevamente la suma de signos que de lo que corresppponda como en caso anterior).

- 0010b+0100b=0110b (2+4=6)
- 0010b+1111b=^{1}0001b=1 (2+(-1)=1)
- 0010b+1101b=1111b=-1 (2+(-3)=-1)
- 1110b+1100b=^{1}1010b=-6 (-2+(-4)=-6)
- 0110b+0110b=1100b=-4 (6+6=12 => fuera de escala)
- 1010b+1010b=^{1}0100b=4 (-6-6=-12 => fuera de escala)

Binario desplazado: Se suma al valor signado el valor absoluto de la mitad del rango menos 1. (El valor que quiero representar le sumo: la mitad del valor absoluto signado, este valor absoluto es 2 (por binario) elevado a la cantidad de bits a representar, y a este valor le resto 1). El resultado de la cuenta es el valor (que tengo que pasar a binario) que representa el numero en binario desplazado. Ej:

Resumen para 4 bits

Decimal	Bit de Signo	Complemento a 1	Complemento a 2	Binario desplazado
8				1111
7	0111	0111	0111	1110
6	0110	0110	0110	1101
5	0101	0101	0101	1100
4	0100	0100	0100	1011
3	0011	0011	0011	1010
2	0010	0010	0010	1001
1	0001	0001	0001	1000
(+)0	0000	0000	0000	0111
(-)0	1000	1111		
-1	1001	1110	1111	0110
-2	1010	1101	1110	0101
-3	1011	1100	1101	0100
-4	1100	1011	1100	0011
-5	1101	1010	1011	0010
-6	1110	1001	1010	0001
-7	1111	1000	1001	0000
-8			1000	

Punto flotante: Representa cualquier numero real.

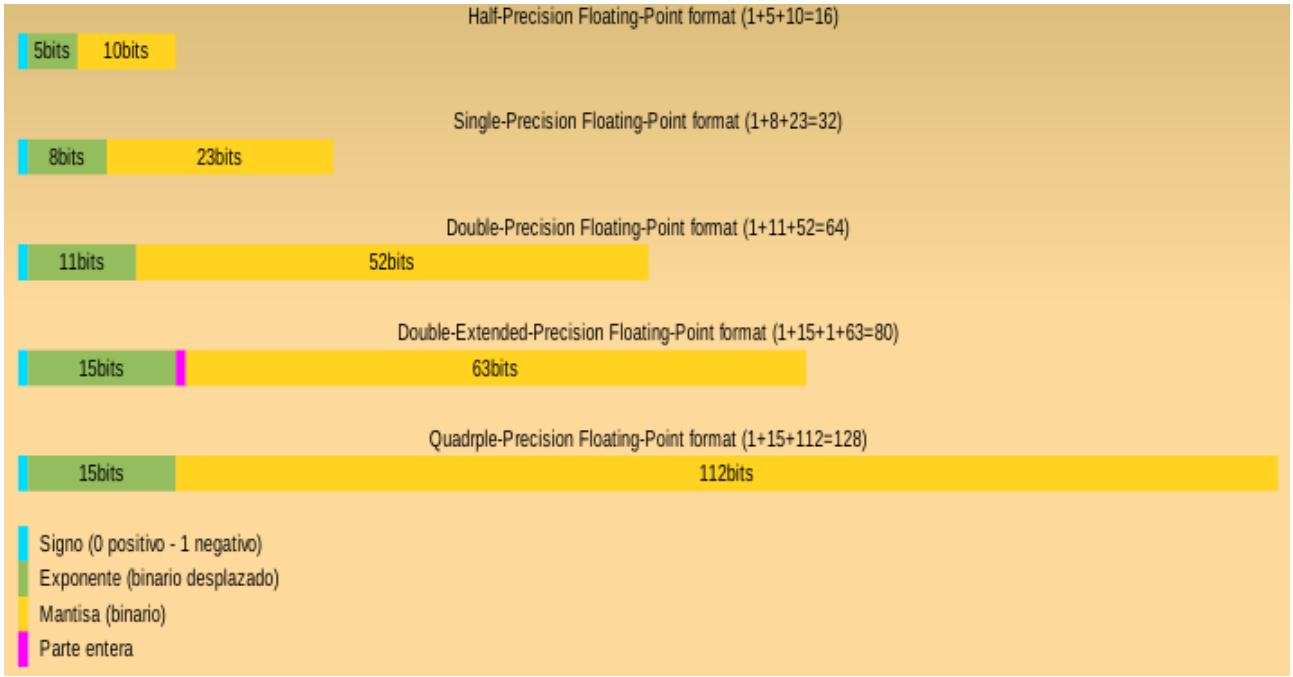
Se representan con los pares ordenados (m,e):

$$(m,e)=m\cdot b^{e}$$

m: mantisa; representa a un número fraccionario.

b: base; toma el valor del sistema de numeración.

e: exponente; un número entero.



Ejemplos

- **Ejemplo wikipedia:**

Codifiquemos el número decimal -118,625 usando el sistema de la IEEE 754.

Necesitamos obtener el signo, el exponente y la fracción.

Dado que es un número negativo, el bit de signo es "1".

Primero, escribimos el número (sin signo) usando notación binaria. Mira el sistema de numeración binario para ver cómo hacer esto. El resultado es 1110110,101.

Ahora, movamos la coma decimal a la izquierda, dejando sólo un 1 a su izquierda.

$1110110{,}101 = 1{,}110110101 \cdot 2^{6}$ Esto es un número en coma flotante normalizado.

El significante es la parte a la derecha de la coma decimal, rellenada con ceros a la derecha hasta que obtengamos todos los 23 bits. Es decir 11011010100000000000000.

El exponente es 6, pero necesitamos convertirlo a binario y desplazarlo (de forma que el exponente más negativo es 0, y todos los exponentes son solamente números binarios no negativos). Para el formato IEEE

754 de 32 bits, el desplazamiento es 127, así es que 6 + 127 = 133. En binario, esto se escribe como 10000101.

Poniendo todo junto:

```
 1     8               23             <-- tamaño en bits
+-+--------+-----------------------+
|S|  Exp   |     Significante      |
|1|10000101|11011010100000000000000|
+-+--------+-----------------------+
31 30    23 22                    0   <-- índice del bit (0 a la derecha)
   desplazado +127
```

- **Ejemplo utenianos:**

Expresar en Base 10 los siguientes números dados en forma de Punto Flotante Precisión Simple.

Y te da el siguiente numero: 35C1F.

Coma flotante de Simple Precision, son 4 bytes.

Por lo que el numero en realidad es 35C1F000

35C1F000 convertido a binario:

00110101110000011111000000000000

Y ahora dividimos en: 1 bit de signo, 8 de exponente, 23 de mantisa

0|01101011|10000011111000000000000

Signo: 0 -> +1
Exponente: 01101011 -> 107 -> le restamos el exceso -> 107 - 127 = -20 -> como son potencias de 2 nos quedamos con -> 2^-20
Mantisa: 10000011111000000000000 -> 2^-1+2^-7+2^-8+2^-9+2^-10+2^-11 = 0.515136719 -> Se le suma 1 -> 1.515136719

Multiplicamos todo:

+1 * 2^-20 * 1.515136719 = 1.44494697 E -6

Expresar en punto flotante simple precisión los números siguientes expresados en base 10.

Y nos da este numero: 165,625.

Tenemos 165,625

Buscamos su signo: +1 -> 0

Transformamos el numero sin el signo a binario

165,625 -> 10100101,101

Exponente:
Desplazamos la coma hasta dejar solo un 1 a la izquierda

1,0100101101 -> 7 lugares -> Mantisa: 0100101101
Exponente: 7 -> Desplazado en 127 -> 127+7 = 134 -> En binario: 10000110

Armamos el numero

0|10000110|0100101101 -> completamos con ceros hasta los 4 bytes

01000011001001011010000000000000 -> En hexa: 4325A000 -> 4325A

#define NOMBRE valor

enum: tipo de dato o variable similar a #define. Esta se realiza por conteo.

enum boolean {NO, SI }; / / NO=0 , SI =1.

enum meses {Enero =1 , Febrero , Marzo , Abril , Mayo , Junio , Julio , Agosto , Setiembre , Octubre , Noviembre , Diciembre }; / / Febrero =2 , Marzo= 3 , . . .

enum escapes {BELL = '\a' , TAB = '\t' , NVLIN = '\n'};

if inline: a = (i<0) ? 0 : 100;