

ESTRUCTURA DE COMPUTADORES

Universidad Politécnica de Madrid

Curso 2022 – 2023

Juan Diego Sanz Guevara

ÍNDICE

1.	INTRODUCCIÓN.....	2
1.1.	COMPONENTES	2
1.2.	MEMORIA PRINCIPAL.....	3
1.3.	UNIDAD CENTRAL DE PROCESO (CPU)	5
1.4.	UNIDAD DE E/S.....	8
1.5.	SOFTWARE DE SISTEMAS.....	9
1.6.	PARÁMETROS CARACTERÍSTICOS	10
2.	ENSAMBLADOR.....	11
2.1.	LENGUAJE E INSTRUCCIONES MÁQUINA.....	11
2.2.	MODO DE DIRECCIONAMIENTO.....	13
2.3.	JUEGO DE INSTRUCCIONES	15
2.4.	FORMATO DE LAS INSTRUCCIONES	18
2.5.	ARQUITECTURA 88110	18
2.6.	JUEGO DE INSTRUCCIONES 88110.....	20
2.7.	ENSAMBLADOR/CARGADOR.....	20
3.	PROCESADOR.....	22
3.1.	OPERACIONES ELEMENTALES	23
3.2.	ESTRUCTURA DE COMPUTADOR ELEMENTAL Y SUS SEÑALES DE CONTROL.....	24
3.3.	CRONOGRAMAS	25
3.4.	DISEÑO DE LA UNIDAD DE CONTROL.....	26
3.5.	CONTROL DE EXCEPCIONES.....	28
4.	ARITMÉTICA.....	29
4.1.	INTRODUCCIÓN.....	29
4.2.	REPRESENTACIÓN EN COMA FIJA	33
4.2.1.	Binario puro (sin signo):.....	33
4.2.2.	Complemento a 2:.....	33
4.2.3.	Complemento a 1	34
4.2.4.	Signo-magnitud	35
4.2.5.	Exceso a "M":.....	35
4.3.	REPRESENTACIÓN EN COMA FLOTANTE.....	36
5.	PERIFÉRICOS.....	39
5.1.	INTRODUCCIÓN.....	39
5.2.	UNIDADES DE CONTROL DE CINTA MAGNÉTICA.....	39
5.3.	UNIDADES DE DISCO MAGNÉTICO (HDD).....	40
5.4.	DISCO DE ESTADO SÓLIDO (SSD)	44
5.5.	MONITOR LCD	45

1. INTRODUCCIÓN

En esta asignatura (y en arquitectura) aprenderemos el funcionamiento de la máquina y será una introducción a cómo funciona esta máquina a bajo nivel.

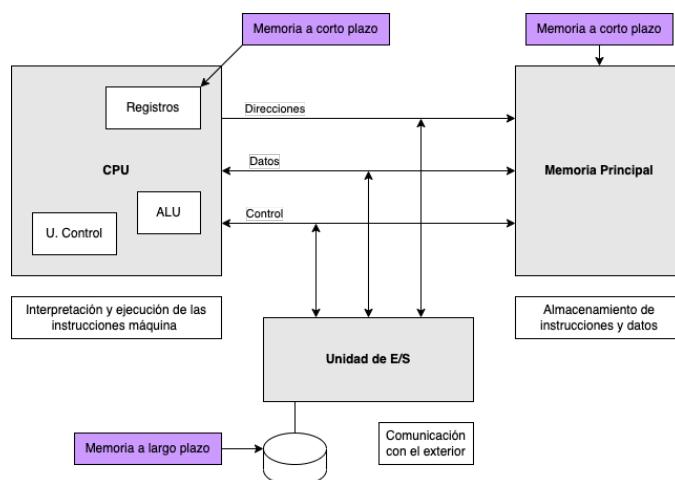
Esta asignatura se basa en una máquina muy básica según la Arquitectura Von Neumann y utilizaremos un microprocesador educativo (Motorola 88110) para poder tener el primer contacto con la programación en ensamblador y ver el funcionamiento más interno de éste

Pero antes, veamos algunas definiciones:

- **Máquina:** Es el hardware en el que se ejecuta TODO.
- **El sistema operativo:** Conjunto de órdenes y programas que controlan los procesos básicos de una computadora y permiten el funcionamiento de otros programas.
- **Compilador:** Es una herramienta de software que traduce el código fuente escrito en un lenguaje de programación de alto nivel a código maquina o programa ejecutable para que pueda ser interpretado directamente por la computadora.
- **Función básica del computador:** Ejecución de instrucciones elementales en las que se especifica:
 - La operación a realizar
 - Los saltos o su localización
 - La localización del resultado.
- **Arquitectura Von Neumann:** Instrucciones y datos deben estar almacenados en una memoria única de lectura/escritura accesible por direcciones. La ejecución es implícitamente secuencial. (Pasa primero por la celda 1, luego por la celda 2, ...).

1.1. COMPONENTES

$$\text{COMPUTADOR} = \text{CPU} + \text{MEMORIA PRINCIPAL} + \text{UNIDAD E/S}$$



- **Memoria principal:** Comúnmente llamada RAM (Random Access Memory). Contiene las instrucciones y los datos que va a manejar el programa.
- **Unidad Central de Proceso (CPU):** Ordena la lectura de instrucciones de la memoria, las interpreta y las ejecuta.

- **Unidad Aritmético-Lógica (ALU):** Realiza las operaciones.
- **Unidad de Control (UC):** Analiza las instrucciones y reparte al resto de componentes.
- **Registros:** Pequeña memoria donde se almacena temporalmente la información.
- **Unidad de Entrada/Salida:** Permite la comunicación del computador con el exterior.
- **Buses:** Interconectan los componentes entre sí.
 - **Direcciones:** Va conectado a los "Registros". Llevan las direcciones desde la CPU hasta la Memoria. La CPU le dice a la Memoria a dónde quiere acceder. Es una dirección unidireccional.
 - **Datos:** Va conectado a la "ALU". Intercambia datos entre la CPU y la Memoria
 - **Control:** Va conectado a la "UC". Transmite señales de control y permite controlar los distintos componentes del computador.
 - Señal de Lectura/Escritura
 - Petición de acceso a información

Para que nos entendamos entre los distintos **dispositivos de almacenamiento**, por tanto, los hay de 3 tipos:

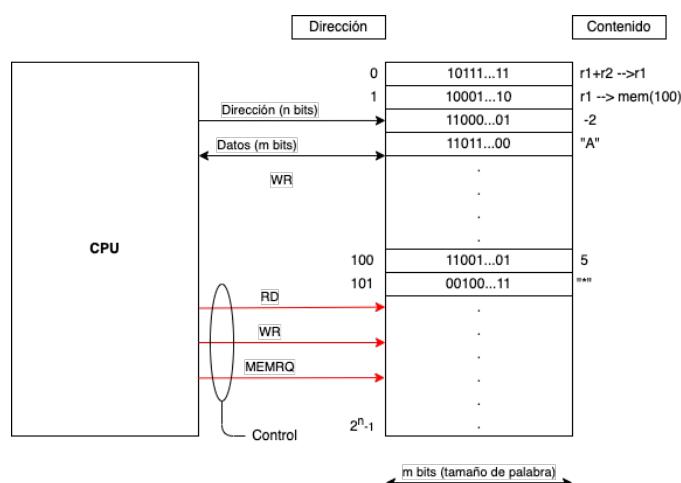
- **Memoria a corto plazo:** Memoria de CPU. Registros.
- **Memoria a medio plazo:** Memoria principal.
- **Memoria a largo plazo:** Disco magnético. (Actualmente se usan discos de estado sólido)

Cuando se apaga el computador, la información de la CPU y la Memoria se borran, pero no del disco magnético.

1.2. MEMORIA PRINCIPAL

Es un conjunto de celdas del mismo tamaño llamadas **palabras** (tamaño privilegiado del ordenador y todo salvo que se diga lo contrario es de una palabra) donde se almacenan tanto **instrucciones** como **datos**, todo en forma de 1's y 0's.. Cada celda tiene **una dirección única que la identifica**. Cada una tiene asignada dirección (0, 1, 2, 3, 4, ..., 2^n-1).

Sólo entra en funcionamiento cuando recibe órdenes de lectura o de escritura por parte de la CPU y cuantas menos operaciones en la Memoria Principal se hagan, más rápida será la CPU.



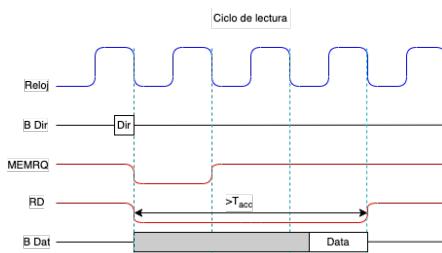
Dichas órdenes viajan por las líneas de control (del bus de control). Dependiendo de lo que queramos hacer en la memoria (si leer o escribir), tendremos dos tipos de ciclos:

- **Ciclo de lectura**

La CPU coloca en el **bus de direcciones** la dirección de memoria cuya información quiere leerse.

La CPU activa (a nivel bajo) las señales de control **MEMRQ** (Memory Request) y **RD** (Petición de lectura).

El tiempo de acceso (T_{acc}) es el tiempo desde que se introduce la dir en el bus de direcciones hasta que la Mp devuelve por el bus de Datos el dato/instrucción solicitada.



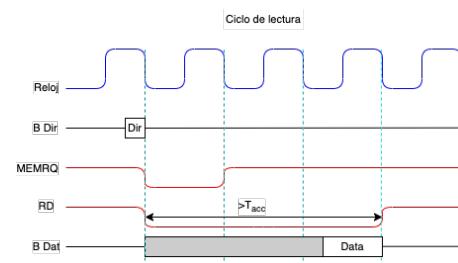
- **Ciclo de escritura**

La CPU coloca en el **bus de direcciones** la dirección igual que en el ciclo de lectura.

La CPU coloca en el **bus de datos** el dato a escribir.

La CPU activa las señales de control MEMRQ y WR.

El tiempo de acceso es en este caso el tiempo desde que se introduce la dirección hasta que la Mp escribe la info enviada.



La memoria se caracteriza por los siguientes parámetros:

- **Capacidad:** Cómo de grande es. Se suele medir en Bytes
- **Tiempo de acceso:** Cómo de rápida es. La CPU debe tener en cuenta este tiempo para hacer operaciones de lectura / escritura.
- **Tamaño de palabra:** Qué cantidad de información se transmite en paralelo entre la CPU y la Mp (número de bits de un dato).

La ordenación del espacio de memoria asignado a un programa se basa en lo siguiente:

El Sistema Operativo (SO) es el componente que se ocupa de gestionar el hardware, gestionar la memoria, repartir el procesador... Además, es el encargado de hacer que lea una instrucción como un dato y viceversa. Asigna un espacio de memoria al programa y ordena la información del programa en la memoria: **código, datos estáticos** (por ejemplo, ya inicializados), **datos dinámicos** (creados y destruidos a lo largo de la ejecución) y **pila** (almacena llamadas a funciones, recibiendo los parámetros). Por último, utiliza métodos para restringir/proTEGER el acceso a determinadas zonas de memoria:

- **Registros frontera:** registros especiales del procesador que marcan la primera y última dirección de un fragmento de memoria. Si la dirección está fuera de un rango determinado, la dirección es incorrecta.
- **Memoria virtual:** no permite ejecutar instrucciones ni operar con datos de la memoria.
- **Arquitectura Harvard:** en lugar de tener una única memoria para datos e instrucciones (Von Neumann), se propone tener una memoria para instrucciones y otra para datos.

1.3. UNIDAD CENTRAL DE PROCESO (CPU)

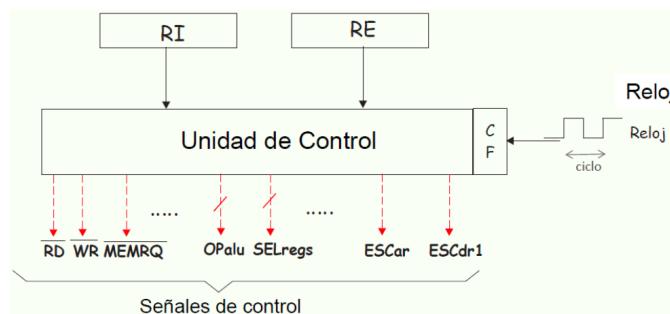
Está constituida por los siguientes componentes interconectados por buses internos:

- **Unidad de Control (CPU):**

Se encarga de **ordenar la lectura** en la Memoria de la siguiente instrucción lanzando un ciclo de acceso a memoria. Una vez leída la instrucción, **la decodifica** y da las **órdenes a los componentes necesarios** para ejecutar la instrucción a través de **señales de control**. Si detecta que le llega una **instrucción**, se **decodifica** y se **ejecuta** y si lo que recibe es un **dato** se realizan las **operaciones pedidas**.

Consulta el **Registro de Instrucción** (RI, instrucción a ejecutar) y el **Registro de Estado** (RE, para saltos condicionales como jumpz) y además el **reloj** marca los instantes de tiempo en los que se mandan las señales de control. El **contador de fase** (CF) se inicializa a 0 cuando comienza una instrucción y se incrementa con el pulso del reloj y además divide en varias etapas las instrucciones (operaciones elementales).

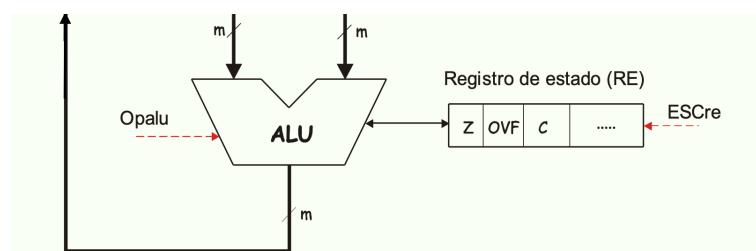
Tiene un **juego de instrucciones básico** con operaciones de **transferencia** (ld, st ...): transfiere la información de un lugar a otro del procesador; **procesamiento** (add, and, ...): manipulan / modifican datos y generan resultados; y **salto** (jump, ...): pasan a ejecutar una instrucción distinta, las hay de dos tipos: **sin retorno**: perdiendo la información desde la cual se salta y **con retorno**: guardando el resultado que me ha dado la función; y **otras instrucciones privilegiadas** ejecutadas solo desde el código del sistema operativo y sólo se pueden ejecutar siendo usuario privilegiado.



- **Unidad Aritmético-Lógica (ALU):**

Es la encargada de **realizar las operaciones de procesamiento** que le ordena la unidad de control sobre los **datos** de longitud de **m bits** (palabras de 32 o 64 bits dependiendo del computador) que se le presentan. La Unidad de Control es la que decide qué operación realizar, qué hacer con el resultado y almacenar las características del resultado.

Además de generar los resultados de las operaciones también expulsa otro tipo de información conocida como **biestables de estado**, pertenecientes al **Registro de estado**. (Biestable Z: 'Zero', Biestable OVF: 'Overflow', Biestable C: 'Carry' (Acarreo)).



En cuanto a los **operandos**, estos pueden venir de los registros de la CPU o de los datos de la Mp y es por esto por lo que debemos siempre fijarnos en el **modelo de ejecución** que emplea nuestra máquina:

- **Registro-Registro:** Ambos operandos deben venir de registros del procesador y que se almacenen también en registros.

$\text{add r1, r2} = \text{r1} + \text{r2} \rightarrow$ Resultado en r1 y r2 no se modifica.

- **Registro-Memoria:** Un operando viene de un registro y otro de memoria y el resultado puede guardarse en registro o memoria

$\text{add r1, /1000} = \text{r1} + \text{Mp}(1000) \rightarrow$ Resultado a r1.

$\text{add /1000, r1} = \text{Mp}(1000) + \text{r1} \rightarrow$ Resultado a Mp(1000).

- **Memoria-Memoria:** Ambos operandos pueden venir de memoria y el resultado se almacena en memoria.

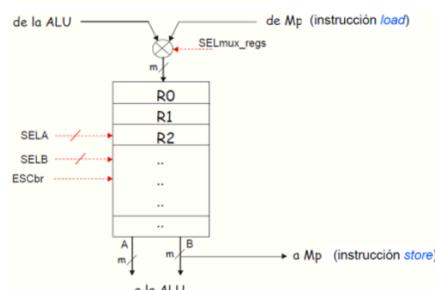
$\text{add /1000, /20} = \text{Mp}(1000) + \text{Mp}(20) \rightarrow$ Resultado a Mp(1000)

El más rápido es el Registro-Registro y el más lento Memoria-Memoria, dado que para este hay muchos tiempos de acceso a la Mp.

- **Registros:**

Como ya hemos dicho, son memoria a corto plazo, que almacena temporalmente instrucciones, datos o direcciones de memoria. Información que será necesaria procesar en un momento del programa. Estos registros pueden ser de tres tipos:

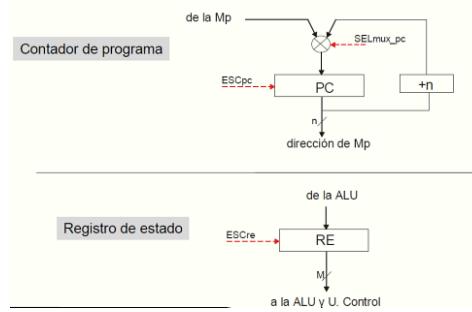
- **De propósito general:** Registros que son especificados explícitamente en las instrucciones
 - **Banco de Registros (BR): Almacenamiento temporal de datos y resultados de operaciones.** Cada registro tiene el tamaño equivalente a una palabra y cada posición tiene un nombre (r0, r1, r2, ...)



- **De propósito específico:** Uso restringido a determinadas instrucciones máquina para labores muy concretas.

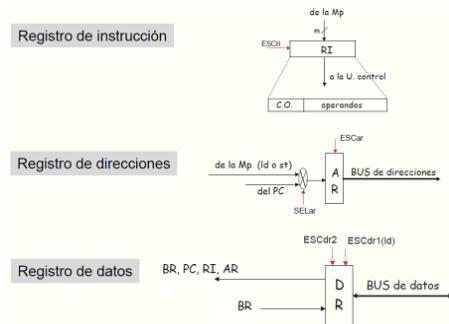
- **Contador de programa (PC):** Contiene la **dirección de memoria donde está la siguiente instrucción a ejecutar**. Cada vez que se ejecuta una instrucción se incrementa en el tamaño de palabra, salvo que sea una instrucción de salto, en cuyo caso se carga la dirección desde la Mp.
- **Registro de estado (PE):** Contiene información acerca de la última operación realizada por la ALU y las características del resultado.
- **Puntero de pila (SP):** No todos los computadores lo tienen. Algunos usan un registro de propósitos general para cumplir su función. Contiene la dirección

de memoria de la pila. Se decrementa cuando se introducen cosas en la pila y se incrementa cuando se sacan.

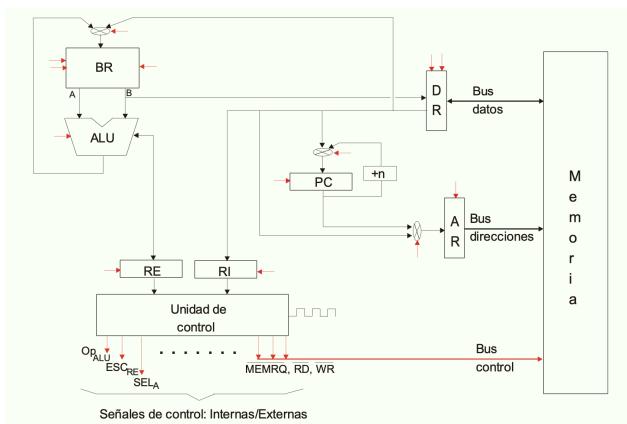


- **Transparentes:** No son visibles para el programador. Uso interno de la CPU para labores muy concretas

- **Registro de instrucción (RI):** Contiene la instrucción que se está ejecutando. Se almacena antes de proceder a la ejecución. La unidad de control accede para saber qué instrucción tiene que ejecutar a continuación. La instrucción cuenta con un código de operación que la identifica y operandos.
- **Registro de direcciones (AR):** Contiene la dirección de memoria a la que se quiere acceder. (Operaciones de lectura o escritura sobre la Mp).
- **Registro de datos (DR):** Contiene la información que lee de memoria o que se quiere escribir en memoria.



Con todo esto, obtenemos un esquema completo de un computador según esta arquitectura:



Las instrucciones, como hemos visto antes, tienen una serie de etapas (operaciones elementales): Estas fases de ejecución son:

1. **Fetch:** Se busca la instrucción en memoria (localizar en memoria los bits que la codifican, identificada con una dirección), se incrementa el contador de programa, PC (para poder avanzar en las direcciones de la Mp) y se introduce su valor en el bus de datos.
2. **Decodificación:** La unidad de control lo analiza y da las órdenes pertinentes.
3. **Ejecución:** La operación se lleva a cabo
 - a. Búsqueda de operandos
 - b. Operación: Activación de la **ALU** y generación del resultado.
 - c. Almacenamiento de los resultados
 - d. Si es una instrucción de salto se carga la dirección en el PC
4. **Finalización o vuelta a la fase de fetch:** Según corresponda. Preparar la siguiente instrucción implica aumentar el PC para que apunte a la siguiente instrucción.

1.4. UNIDAD DE E/S

Comunica los periféricos con el resto del computador. Por ejemplo, para tener información en memoria hay que haberla cargado desde un disco externo. Los periféricos son **muy distintos** entre sí, con distintos tipos de información a transmitir, distintas velocidades, etc. Por eso los periféricos se conectan al computador mediante un controlador específico para cada tipo de periférico conocido como módulo E/S. Ese módulo E/S **oculta las particularidades de cada periférico**, presentando a la CPU una visión homogénea de todos ellos. La **CPU interacciona con el módulo E/S y no con el periférico**. El módulo E/S tiene tres registros:

- **Registro de Datos:** Un "buzón" donde la CPU deja información para el periférico y viceversa, todo ello a través de la conexión del módulo al bus de datos.
- **Registro de Control:** Por donde la CPU envía sus órdenes al periférico (lectura/escritura, apagado/encendido, ...).
- **Registro de Estado:** De aquí obtiene la CPU el estado del periférico (Comprobación de que haya papel en una impresora, de si hay datos disponibles para leer o no, etc).

Problemas de comunicación entre CPU y Módulo de E/S:

- **Selección del dispositivo / periférico (direcccionamiento):** Al igual que la dirección de la Mp estaba organizada por direcciones, cada registro de **cada módulo tiene asignado una dirección**. La dirección se envía a través del bus de direcciones. Cada registro y su dirección asociada se llama "Puerta de E/S". Al salir una dirección o una señal de la CPU le llega a Memoria y a la Unidad de E/S, en algunos procesadores hay instrucciones especiales que diferencian un acceso de E/S de uno a memoria ($\text{in/out} \neq \text{ld/st}$)
- **Modo de realizar la operación de transferencia de E/S:** Dependen del grado de participación de la CPU en la operación de transferencia de E/S (ordenando de mayor a menor grado de participación de la CPU)
 - **Programada: La CPU lo hace todo.** Inicia la operación y la dirige durante todo el proceso. La principal desventaja es que la CPU está permanente ocupada esperando datos si el periférico es más lento. Se hace mediante un pequeño programa que lleva a cabo lo siguiente:
 1. **Iniciar la operación:** Le dice al periférico que quiere realizar una operación de lectura enviando esa información al registro de control.

2. **Comprobar dato:** Mira si el periférico ya tiene un dato disponible para ella en el registro de estado.
3. **Enviar dato:** Si el dato está listo se recoge y se envía a un registro de la CPU y seguidamente, si procede, a memoria.
4. **Finalización o vuelta a recoger datos:** Según corresponda.
 - **Interrupciones:** CPU lee datos y lo envía a Mp.
 1. **CPU:** El módulo E/S avisa a la **CPU** enviándole una **señal de control** cuando tiene un dato listo, de forma que permite a la CPU dedicarse a otras cosas mientras tanto (si el computador tiene n cores o procesadores, podrá ejecutar n programas al mismo tiempo).
 2. **Periférico:** Cuando la CPU recibe la **señal de interrupción** (enviada por el periférico) detiene lo que estuviese haciendo, de forma que no hay sincronización, sino interrupción.
 3. **CPU:** Ejecuta una **rutina de tratamiento de interrupción**. (Rutina privilegiada). Esta rutina hace la operación de transferencia de dato desde el **módulo E/S a la CPU y si procede a memoria**. Una vez ha terminado, la CPU sigue con lo que estaba haciendo.
 - **Acceso directo a memoria (DMA):** La CPU casi no interviene.
 1. **CPU:** El módulo E/S se encarga por sí mismo de transferir la operación a memoria mientras la **CPU está ocupada** en otras cosas.
 2. **Periférico:** Una vez ha terminado la transferencia, el módulo E/S envía una **señal de control** (INT) a la CPU.
 3. **CPU:** La CPU ejecuta la **rutina de tratamiento de interrupciones** (instrucción privilegiada), que se encarga de comprobar que la transferencia ha sido correcta, y vuelve a continuar con lo que estaba haciendo.

1.5. SOFTWARE DE SISTEMAS

- **Compiladores y ensambladores:** traducen el lenguaje de alto nivel usado por el programador al lenguaje máquina.
- **Montadores (linker):** junta los distintos ficheros que forman un programa.
- **Cargadores:** carga el código del programa en memoria.
- **Depuradores:** permite ejecutar el programa paso a paso para ver dónde está el problema.
- **Editores de texto:** ayuda con la escritura de los programas.
- **Sistema operativo:** reparte el uso de la CPU entre los programas, organiza el espacio de memoria y oculta la complejidad de los periféricos al programador.

1.6. PARÁMETROS CARACTERÍSTICOS

- **Ancho de palabra:** Cantidad de información que se transmite por el computador en paralelo (en los buses).
 - 8, 16, 32, 64 bits.
- **Tamaño de la memoria:** Almacenamiento del BR, del Mp, de un periférico, ...
 - Kilo (banco de registros), Mega (Mp), Giga, Tera (discos magnéticos), Peta (discos magnéticos) bytes
- **Frecuencia de reloj:** Señal que gobierna la CPU.
 - Mega, Giga hercios.
- **Duración de las operaciones:** De la ALU en la CPU.
 - mili, micro, nano, pico, femto segundos
- **Capacidad de cómputo (velocidad):** Velocidad con la que la CPU ejecuta instrucciones o realiza operaciones.
 - **MIPS** (millones de instrucciones por segundo), **MFLOPS** (millones de operaciones de coma flotante por segundo), specint, specfp
 - **MIPS:** Medida usada en registros de propósito general.
 - **MFLOPS:** Medida usada en computadores orientados a cálculo científico.
- **Ancho de banda (caudal):** Cantidad de información por segundo capaz de transferir una unidad determinada (hacia o desde memoria, un disco duro, ...).
 - KB/s (KBps), MB/s (MBps), Kb/s (Kbps), Mb/s (Mbps).
- **Ejemplo para el cálculo de MIPS:**
 - $102 \text{ (fetch)} + 1 \text{ (decodificación)} + 202*0,2 \text{ (st)} + 203*0,3 \text{ (ld)} + 1*0,3 \text{ (ALU)} + 102*0,1 \text{ (salto exitoso)} + 1*0,1 \text{ (salto fallido)} = 214,9 \text{ ciclos.}$
 - $0,3 \text{ ns/ciclo} * 214,9 \text{ ciclos/instrucción} = 64,47 \text{ ns por instrucción.}$
 - $1 / (64,47 * 10^{-9}) = 15,5 \text{ MIPS (millones de instrucciones por segundo).}$

2. ENSAMBLADOR

2.1. LENGUAJE E INSTRUCCIONES MÁQUINA

Vamos a explicar un poco de contexto para que no vayamos perdidos:

1 Byte = 8 bits. 8 bits pueden contener 2^8 combinaciones → 0, ..., 255 (decimal)

Máquina de 32 bits = Máquina de 4 Bytes. 32 bits con 2^{32} combinaciones.

1 Byte: 0101 0111. → 0x57 (Hexadecimal, cada 4 bits en un dígito hexadecimal).

1 Byte: 1011 1001 → 0xB9

4 Bytes: 1011 1001 1011 1001 1011 1001 1011 1001 → 0xB9B9B9B9

Hexadecimal: [0, ..., 9, A, ..., F]

Notación:

- 0x indica que lo siguiente vendrá en hexadecimal = 0xB9B9B9B9
- #H lo mismo = #H'FFFFFFF = $2^{32}-1 = 32$ bits a 1. F = 1111 = $2^4 - 1 = 15$

Lenguaje máquina: Es el que es capaz de interpretar y ejecutar directamente el computador. Basado en sistemas de representación binarios y es particular de cada procesador.

Instrucciones: Se representan como cadenas de ceros y unos, realizan **una única y sencilla función** (en general), operan sobre un **número fijo de operandos** y son **autocontenidas**. Es decir, contienen **toda la información necesaria para su ejecución**: **operación** a realizar, **operando** o donde se encuentran, donde dejar el **resultado** y ubicación de la **siguiente instrucción**.

Formato de las instrucciones: Es la manera en la que se codifica cada una de las instrucciones: Instrucción = código de la operación (c.o) + Operandos.

Subrutina: Un trozo de programa dedicado a una tarea específica. Por ejemplo, un número de instrucciones dedicadas a sumar dos operandos. (Lo análogo a una función)

Lenguaje ensamblador: Lenguaje parecido al máquina, pero comprensible.

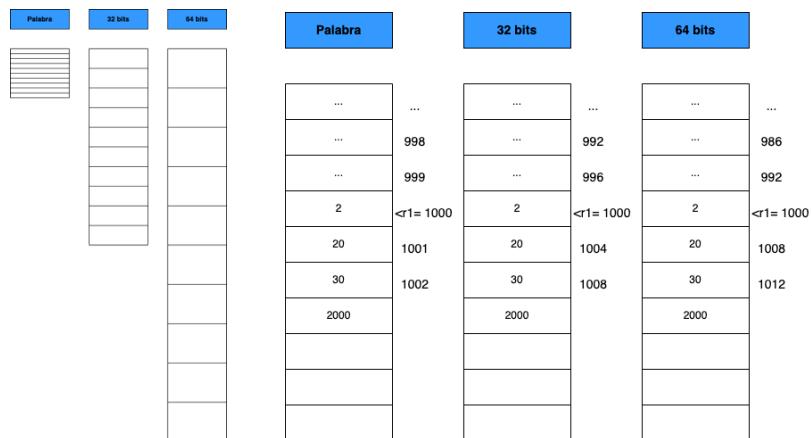
Juego de instrucciones: Es un conjunto de operaciones máquina que es capaz de entender y ejecutar la CPU. Viene definido por las **operaciones** que realiza, los **tipos de datos** que maneja, los **modos de direccionamiento** y los **formatos de las instrucciones**. Todo ello afecta a la velocidad de ejecución.

- **Operaciones:** Transferencia (ld, st, move), procesamiento: (add, sub, and, cmp), control de flujo de ejecución (br, bz, call, ret).
- **Tipos de datos:** Direcciones, números, caracteres, datos lógicos, etc.
- **Tamaños:** Palabra, doble palabra, media palabra, byte.
- **Direccionamiento de memoria:**
 - **A nivel de palabra:** Direcciones de memoria consecutivas corresponden a palabras consecutivas.
 - **A nivel de Byte:** Direcciones de memoria consecutivas corresponden a bytes consecutivos.
 - **Ordenación de los bytes de una palabra en memoria:**
 - **Little Endian (Le):** Byte menos significativo de una palabra en la dirección menor
 - **Big Endian (Be):** Byte menos significativo de una palabra en la dirección mayor

- **Ejemplo:** Número decimal 70960543 (0x043AC59F) almacenado en 32 bits de la dirección 184.

Big endian	184	185	186	187
	04	3A	C5	9F
Little endian	184	185	186	187
	9F	C5	3A	04

- **Direcciones de memoria en bytes vs palabras:** Como lo vimos en su comienzo, denotamos a palabra como el tamaño privilegiado del ordenador, que es 1 Byte = 8 bits. A continuación, mostramos cómo recorremos la memoria dependiendo del tipo de direccionamiento:



Alineamiento de datos: Un dato que ocupa k bytes está alineado en memoria si está almacenado en una dirección múltiplo de k. Muchas arquitecturas solo permiten accesos a datos alineados. Las que permiten acceso a datos no alineados son más lentos e implican varios accesos a memoria.

Ejemplo:

184		10	20	30
188	40			

- La palabra 0x10203040 no está alineada en la dirección 185
- La palabra 0x2030 sí está alineada en la dirección 186

Debemos tener mucho cuidado con los errores de alineamiento

2.2. MODO DE DIRECCIONAMIENTO

Modos de direccionamiento: Forma de especificar los operandos de la instrucción. El **objeto** puede ser un dato, un resultado o una instrucción. Puede estar en la propia instrucción (datos), un registro (datos o resultados) o en memoria (datos, resultados o instrucciones). En cuanto a **la dirección del objeto**, es el identificador que especifica el registro o posición de memoria donde se encuentra.

Existen los siguientes tipos de direccionamiento:

- **Inmediato:** El objeto está contenido en la propia instrucción. Se expresa como #valor. Útil para constantes. Es rápido, ya que no se requieren accesos a memoria, pero el operando tiene un rango limitado.

- **ADD .R1, #4** ; R1 \leftarrow R1 + 4
- **LD .R2, #4** ; R2 \leftarrow R2 + 1

- **Directo:** En la instrucción se especifica la dirección del objeto.

- **Absoluto:** La instrucción contiene la dirección completa del objeto en registro o en memoria.

- **A registro:** Se expresa como **.registro**. Útil para variables utilizadas frecuentemente. El acceso es rápido y se necesitan pocos bits para especificar la dirección, pero hay un número limitado de registros.

- **ADD .R4, .R3** ; R4 \leftarrow R4 + R3

- **A memoria:** Se expresa como **/dirección memoria**. Permite el acceso a un gran espacio de direcciones, pero el acceso es más lento, el código no es reubicable y el número de bits necesario para especificar la dirección es mayor.

- **ADD .R4, ./100** ; R4 \leftarrow Mem(100)
 - **BR /1000**

- **Relativo:** La instrucción contiene un desplazamiento sobre una dirección que suele estar almacenada en un registro. El objeto está en memoria. Permite dar la dirección con un número de bits, ya que el operando contiene un valor entero (desplazamiento) y un registro en el que está la dirección de memoria. Este direccionamiento se adapta mucho para estructuras de datos y parámetros en pila y el código es reubicable pero el rango de direcciones accesibles está limitado por el desplazamiento.

- **A registro base:** Se expresa como **#desp [.registro_base]**, como **[.registro_base, #desp]** o como **[#desp, .registro_base]**.

- **LD .R1, #4[.R7]** ; R1 \leftarrow Mem(R7+4)
 - **LD .R1, [#4, .R7]** ; R1 \leftarrow Mem(R7+4)

- **A contador de programa:** Se expresa como **\$desp**. El registro base es el PC. Se utiliza en instrucciones de salto. El objeto de este direccionamiento es instrucción. Permite alcanzar instrucciones cercanas a las que se está ejecutando. Sirve por ejemplo, para bucles if, then, else...

- **BR \$10** ; PC \leftarrow PC+10

- **A registro índice:** En este caso el registro base se modifica. El tamaño del incremento o decremento es igual al tamaño del objeto direccionado. Es útil para el recorrido de vectores y matrices. Se expresa que es relativo a registro base, pero con un post/pre incremento/decremento sobre el registro base.

Por ejemplo, **#desp[++.registro_base]** o **[.registro_base--, #desp]**.

- Preincremento:
 - **LD .R1, #8[++.R7]** ; R7 \leftarrow R7+4 y R1 \leftarrow Mem(R7+8)
- Predecremento:
 - **LD .R1, #8[--.R7]** ; R7 \leftarrow R7+4 y Mem(R7+8) \leftarrow R1
- Postincremento:
 - **LD .R1, #8[.R7++]** ; R1 \leftarrow Mem(R7+8) y R7 \leftarrow R7+4
- Postdecremento:
 - **SUB .R1, #8[.R7--]** ; R1 \leftarrow Mem(R7+8) y R7 \leftarrow R7-4
- **Indirecto:** Se expresa como **[/dirección memoria]** o **[.registro]**. Se especifica la dirección donde está almacenada la dirección del objeto.
 - **A registro:**
 - **LD .R1, [.R7]** ; R1 \leftarrow Mem(R4)
 - **BR [.R1]** ; PC \leftarrow R1
 - **A memoria:**
 - **LD .R1, [/1000]** ; R1 \leftarrow Mem(Mem(1000))
 - **A registro base:**
 - **LD .R1, [#8[.R4]]** ; R1 \leftarrow Mem(Mem(R4+8))
 - **A registro índice:**
 - **LD .R1 [#8[.R2++]]**; R1 \leftarrow Mem(Mem(R2+8)), R2 \leftarrow R2+4
- **Implícito:** La instrucción no contiene ni la dirección ni el objeto. El operando se supone ubicado en un lugar específico según la instrucción. Por ejemplo, **PUSH .R1** sabe que tiene que mandar el contenido de R1 a la pila.

A modo de **resumen**, entonces:

- **Inmediato:** #Valor
- **Directo**
 - Absoluto
 - A registro..... registro
 - A memoria..... /dirección
 - Relativo
 - A registro base..... #desp[.registro_base]
[.registro_base, #desp]
 - A registro índice
 - A PC \$desp
- **Indirecto** []
- **Implícito**

2.3. JUEGO DE INSTRUCCIONES

Instrucciones de transferencia: Mueve información entre registros y memoria. No se modifican los biestables de estado.

- **Load:** Lleva de memoria a registro.

○ **LD** .R2, #4[.R4] ; R2 \leftarrow MEM(R4+4)

- **Store:** Lleva de registro a memoria

○ **ST** .R2, #4[.R4] ; MEM(R4+4) \leftarrow R2

- **Move:** Lleva de memoria a memoria

○ **MOVE** .R2, .R4 ; R4 \leftarrow R2

○ **MOVE** [.R2], [.R4] ; MEM(R4) \leftarrow MEM(R2)

- **PUSH y POP** meten y sacan de la pila

○ **PUSH** .R1 ; MEM(SP) \leftarrow R1; SP \leftarrow SP-4

○ **POP** .R1 ; SP \leftarrow SP +4; R1 \leftarrow MEM(SP)

Instrucciones aritméticas y de comparación: Todas modifican los biestables de estado. Están restringidas por el modelo de ejecución. Si el modelo de ejecución es registro-registro los operandos solo pueden estar en registros. CMP no modifica ningún operando, así que sacamos su resultado del registro de estado.

- **ADD** .R1, .R2 ; R1 \leftarrow R1 + R2
- **SUB** .R1, .R2 ; R1 \leftarrow R1 - R2
- **MUL** .R1, .R2 ; R1 \leftarrow R1 * R2
- **DIV** .R1, .R2 ; R1 \leftarrow R1 / R2
- **ADDC** .R1, .R2 ; R1 \leftarrow R1 + R2 + C
- **SUBC** .R1, .R2 ; R1 \leftarrow R1 - R2 - C
- **CMP** .R1, .R2 ; R1 = R2 ; No modifica ni R1 ni R2
- **INC** .R1 ; R1 \leftarrow R1 + 1
- **DEC** .R2 ; R2 \leftarrow R1 - 1

La ubicación de los operandos depende del modelo de ejecución del procesador:

- **Registro - Registro:** Menos de 1ns, porque no accede a Mp
- **Registro - Memoria:** 40-50ns, porque accede 1 vez a memoria
- **Memoria - Memoria:** 120ns, porque accede 2 veces a memoria

Definen también el número de direcciones del procesador:

- **3 direcciones:** add .R3, .R1, .R2 ; R3 \leftarrow R1 + R2
- **2 direcciones:** add .R1, R2 ; R1 \leftarrow R1 + R2
- **1 dirección:** add .R1 ; AC \leftarrow AC + R1
- **0 direcciones:** add

Instrucciones lógicas: Realizan la operación bit a bit. Modifican los biestables de estado

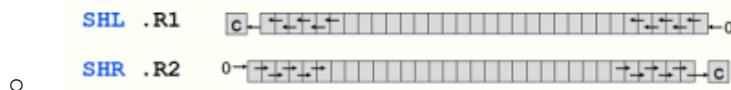
- **AND .R1, .R2** ; $R1 \leftarrow R1 \text{ AND } R2$
- **XOR .R1, #4** ; $R1 \leftarrow R1 \text{ XOR } 4$
- **OR .R1, [.R2]** ; $R1 \leftarrow R1 \text{ OR Mem}(R2)$
- **NOT #8[.R1]** ; $R1 \leftarrow \text{NOT Mem}(R1+8)$

Instrucciones de bit: Realizan operaciones sobre un bit

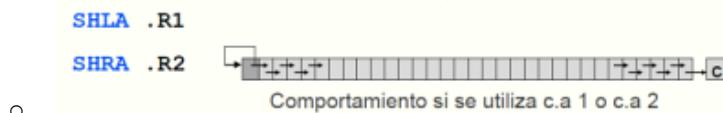
- **CLR.I #3, .R1** ; Pone a cero el bit 3 de R1
- **SET.I #3, .R1** ; Pone a uno el bit 3 de R1
- **TEST.I #3, .R1** ; Biestable Z $\leftarrow \text{NOT}(\text{bit3}(R1))$
 - ; si $\text{bit3}(R1) = 0$, biestable Z $\leftarrow 1$
 - ; si no, biestable Z $\leftarrow 0$

Instrucciones de desplazamiento: Realizan desplazamientos de bits a izquierda o derecha. Equivalen a multiplicar o dividir por 2.

- **Lógico:**



- **Aritmético:**

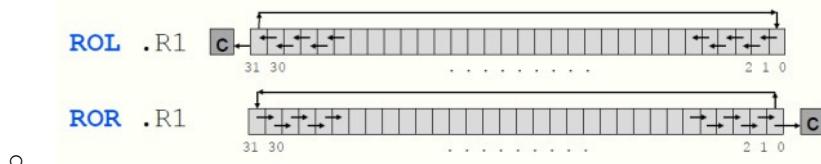


- **Operaciones con bits:**

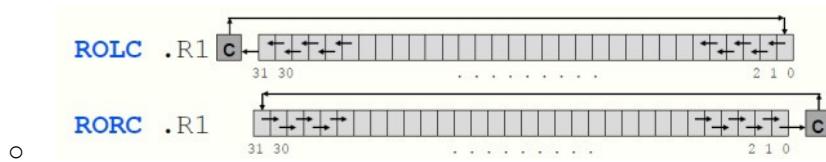
- **Desplazamiento a la izquierda:** Se multiplica el número por su base.
- **Desplazamiento a la derecha:** Se divide el número por su base.

Instrucciones de rotación: Igual que las de desplazamiento, pero lo que sale por un lado entra por el otro.

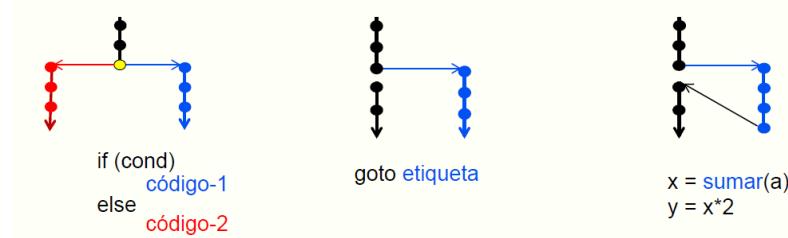
- **Rotación:**



- **Rotación a través del biestable de Acarreo**



Instrucciones de salto o bifurcación: Modifican la secuencia habitual de ejecución (secuencial). No modifican los biestables de estado. Pueden ser condicionales/incondicionales o con/sin retorno



- **Saltos incondicionales:** El salto se realiza siempre. Tienen como operando la dirección de memoria de la siguiente instrucción a ejecutar. Aunque los operandos se escriban como si fuese a leerse en memoria no se lee en memoria. Simplemente se guarda el valor de la dirección de memoria a la que se quiere saltar en el contador de programa.

- **BR** /1000 ; $PC \leftarrow 1000$
- **BR** #4[.R3] ; $PC \leftarrow R3 + 4$
- **BR** \$10 ; $PC \leftarrow PC + 10$
; El PC apunta a la dirección de la siguiente instrucción!

- **Saltos condicionales:** El salto se realiza sólo si cumple la condición. Se sustituye la R de la instrucción del salto incondicional por la letra o letras que correspondan a la condición.

- **BZ** /1000 ; Si biestable(Z) = 1, $PC \leftarrow 1000$
Si no, $PC \leftarrow PC+N$
- **BNC** #4[.R3] ; Si biestable(C) = 0, $PC \leftarrow R3 + 4$
Si no, $PC \leftarrow PC+N$
- **BP** \$10 ; Si biestable(S) = 0, $PC \leftarrow PC + 10$
Si no, $PC \leftarrow PC + N$

- **Saltos con retorno:** Se guarda la dirección de retorno. Se utilizan para implementar saltos a subrutinas o llamadas a funciones. Una vez ejecutado el código al que se salta se debe retornar al siguiente salto.

- En un registro de propósito general
 - **CALL** / 1000 ; $R1 \leftarrow PC, PC \leftarrow 1000$
 - **RET** ; $PC \leftarrow R1$
No permiten llamadas anidadas
- Guardando la dirección de retorno en la pila
 - **CALL** /1000 ; $Mem(SP) \leftarrow PC, SP \leftarrow SP-4,$
; $PC \leftarrow 1000$
 - **RET** ; $SP \leftarrow SP+4, PC \leftarrow Mem(SP)$
Permite llamadas anidadas

2.4. FORMATO DE LAS INSTRUCCIONES

Modo en el que se disponen los bits que identifican cada elemento de una instrucción. Se suele emplear más de un formato. Cada instrucción encaja en un formato predeterminado. Si la longitud es fija o variable, el número de formatos distintos y el si la codificación es regular tienen impacto en la velocidad de la CPU. El código de operación debe estar siempre al principio y el número de bits que ocupa dependerá del número de operaciones distintas que haya en el juego. Ocupa el mismo espacio en todos los formatos de instrucción distintos que se usen.

Paradigmas de Arquitectura de un computador: Responden a la evolución tecnológica en el diseño de procesadores. Existen de dos tipos: **CISC** (Complex Instruction Set Computer) y **RISC** (Reduced Instruction Set Computer): El CISC permitía una gran cantidad de instrucciones hasta el punto de que el lenguaje ensamblador empezaba a asemejarse a un lenguaje de alto nivel. Sin embargo, se observó que el 80% del tiempo de ejecución de los programas lo ocupaban el 20% de las instrucciones. Por tanto, se realizó el cambio a RISC con el objetivo de hacer más rápidos los casos más frecuentes.

Arquitectura	CISC	RISC
Nombre	Complex Instruction Set Computer	Reduced Instruction Set Computer
Modelo de ejecución	R-R, R-M, M-M	R-R (Sólo load y store hacen referencia a memoria). Con load los datos de memoria se cargan en el BR, cuyo acceso es más rápido
Tipos de instrucciones	Complejas	Sencillas
Tiempo de ejecución	Alto	Bajo
Modelo de direccionamiento	Complejos	Sencillos
Formatos de instrucción	Muchos	Pocos y codificación uniforme
Unidad de Control	Compleja	Sencilla
Nº instrucciones / programa	Menos	Más
Ejemplos	ix86, AMD	Sparc, Arm, PowerPC

2.5. ARQUITECTURA 88110

Procesador:

- Arquitectura **RISC**
- Máquina de **3 direcciones**.
 - Ej.: add rD, rS1, rS2 rD \downarrow rS1+rS2.
- Modelo de ejecución **registro-registro**. No jmp/ld/st.
- **Palabra** de 32 bits.

- **Direccionamiento** a nivel de byte.
- **ALU** opera en complemento a 2.
- En el **emulador** ejecutamos en serie, es decir, una instrucción por ciclo.

Banco de registros: hay 32 registros de propósito general de r0 a r31. Son accesibles por pares en operaciones de 64 bits.

- **r0:** cableado a 0 (siempre tiene valor 0). Lo podemos leer, pero no modificarlo.
- **r1:** guarda la dirección de retorno de subrutina
- **PC:** contador de programa. Indica cuál es la siguiente instrucción a ejecutar.
- Registro de estado **PSR** (Processor Status Register):
 - **Bit 12:** overflow en operaciones con enteros. Si OVF=1 no se modifica rD.
 - **Bit 28:** acarreo.
- Registros que reservamos nosotros de forma particular
 - **r30:** puntero de pila. (SP)
 - **r31:** puntero de marco de pila.

Memoria principal:

- **Arquitectura Von Neumann:** Almacena en memoria instrucciones y datos
- **Direccionamiento a nivel de byte** (1 palabra ocupa 4 direcciones de memoria)
- **Bus de direcciones de 32 bits.** La máxima capacidad (teórica) es de 2^{18} bytes = 4 GB. 0x00000000 – 0xFFFFFFFF
- **Capacidad del emulador:** 2^{18} direcciones = 2^{18} bytes = 256 KB. 0x00000000 – 0x0003FFFF

Modos de direccionamiento:

- **Inmediato:** no hace falta poner almohadilla delante del dato. Con signo o sin signo en decimal (sin nada) o en hexadecimal (con 0x).
 - add r1, r2, -13.
 - **Complemento a 2:** Para pasar de binario puro a C2, negamos el numero bit a bit y le sumamos 1:
 - 13 en binario: 0000 1101
 - -13 en binario: 1111 0011
- **Directo a registro:** no hace falta poner puntos delante de los registros.
 - add rD, rS1, rS2.
- **Relativo a registro base:** se expresa sin corchetes ni almohadillas.
 - ld r1, r4, 13/r10 ; r1 \leftarrow MEM(r4+13/r10)
- **Relativo a PC:**
 - et0: br D ; PC \leftarrow et0 + 4*D.
- **Indirecto a registro:** se expresa con paréntesis.
 - jmp (r10).

- **Direccionamiento a campos de bits:** en ciertas instrucciones se pueden seleccionar bits individuales o campos de bits. La forma de expresarlo depende de la instrucción.
- **No tiene absoluto, relativo a registro índice, indirecto a memoria ni relativo a pila.**

2.6. JUEGO DE INSTRUCCIONES 88110

La información relativa al juego de instrucciones de este microprocesador, la encontramos en el PDF "Emulador 88110"

2.7. ENSAMBLADOR/CARGADOR

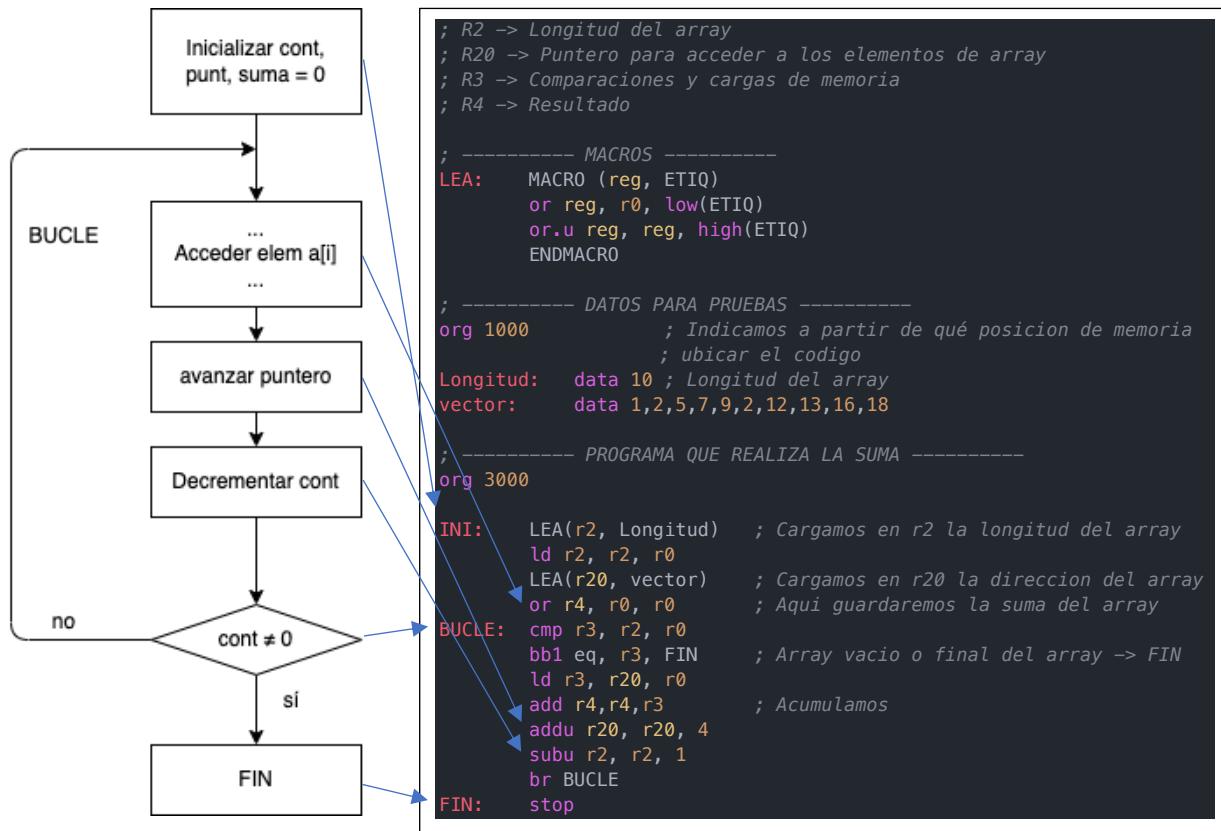
Llamamos **ensamblador** al programa que se encarga de "traducir" un programa escrito en lenguaje ensamblador a lenguaje máquina. El ordenador no puede leer un programa en lenguaje ensamblador, hay que pasarlo a un fichero binario que el ordenador sí podrá leer. Estos programas se basan en una serie de instrucciones que pueden ser o bien instrucciones máquina o una pseudo-instrucción.

- **Pseudoinstrucción:** No genera instrucciones máquina (en memoria) y son instrucciones para el ensamblador. Estas indican al ensamblador cómo debe generar el código máquina.
 - **Org n:** Indica al programa a partir de qué posición de memoria ubicar el código.
 - **Res n:** Indica reservar n bytes en memoria.
 - **Data a, b, c, ...:** Inicializar posiciones en memoria con datos a, b y c.
 - **Data "texto":** Inicializar una posición en memoria con una cadena de bytes texto.
 - **Low (etiqueta o inmediato):** Devuelve los 16 bits menos significativo del dato.
 - **High (etiqueta o inmediato):** Devuelve los 16 bits más significativos del dato.
 - Se divide una palabra de 32 bits en dos palabras de 16.
- **Macroinstrucciones ("macros"):** Conjunto de instrucciones que pueden recibir argumentos y que realizan una tarea concreta. Se usan para encapsular pequeños fragmentos de código que no son lo suficientemente extensos como para ser subrutinas. Su sintaxis es:


```
Nombre_de_macro: MACRO (arg1, arg2, ..., argn)
        Conjunto de instrucciones que componen la macro
      ENDMACRO
```

Algunas de sus características son: Es responsabilidad del llamante liberar los registros que esta utilice, se permiten macros anidadas, no se permiten la definición de etiquetas dentro de una macro.

Vectores: Suma de los elementos N de un vector:



Matrices: Suma de los elementos de cada columna de la matriz MxN y guardar en vector VSUMA

Matrices

Listas

Subrutinas

3. PROCESADOR

Unidad de control: Es la encargada de interpretar las instrucciones del programa y gobernar la ejecución de estas

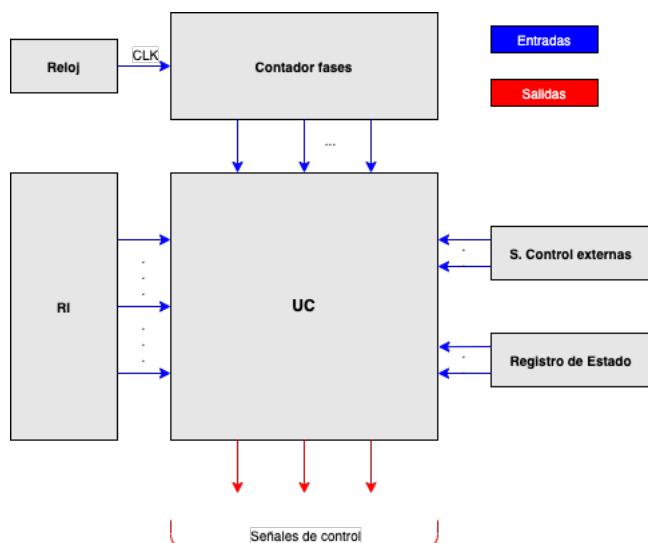
Camino de datos: Conexiones de la CPU por las que se transfieren los datos procedentes de la **memoria principal** o **registros internos**. Debe permitir el conjunto de operaciones básicas que precisa el repertorio de instrucciones.

Funciones de la CPU:

- **Ejecuta instrucciones:**
 - Lectura, decodificación e interpretación de las instrucciones.
 - Generación de órdenes para la ejecución.
 - Secuenciamiento de instrucciones (decidir cuál es la siguiente a ejecutar).
- **Resuelve situaciones anómalas** (desbordamiento, operación no válida, error de paridad, etc).
- **Controla la comunicación con periféricos**

Entradas y salidas de la CPU:

- **Entradas:** Registro de instrucción (CO = Código de Operación), MDs (Modos de Direccionamiento), reloj (registro contador de fases), registros de estado, señales de control externas (E/S, Mp, ...)
- **Salidas:** Todas las señales de control que permiten realizar las instrucciones máquina.

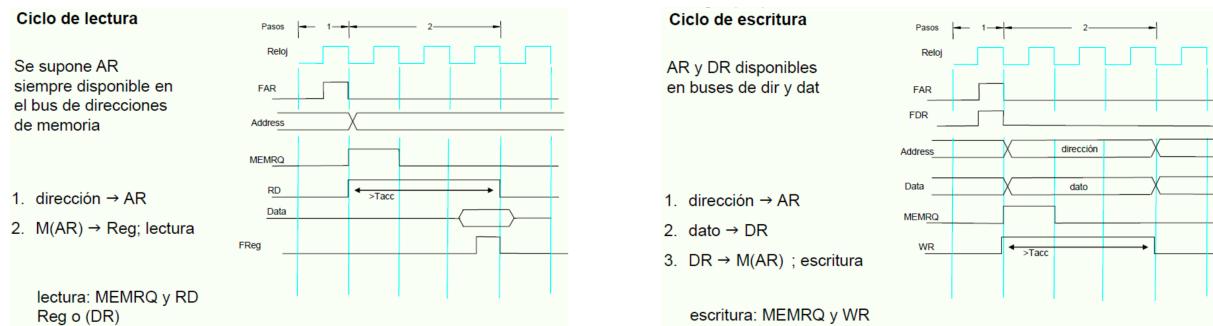


Reloj: Tren de pulsos caracterizado por su periodo (T). Gobierna la máquina: Marca el ritmo de avance de las operaciones. Cuanto más rápido sea el reloj, más rápida será la ejecución de instrucciones. Las señales de control se sincronizan con el reloj. Define el tiempo de cada operación (en Memoria: Tiempo de lectura / escritura y en la ALU: Tiempo de operación)

- **Camino crítico:** Camino de máximo retardo entre un origen y un destino. Depende de los dispositivos que tengan que atravesar las señales. (Si $f = 100\text{Hz}$, $T = 10\text{ns}$. Conviene que se haga en menos de 10ns (Camino Crítico))

Ciclo de lectura y escritura en memoria: Ciclo de BUS

Implica el acceso al exterior a la CPU. Durante el tiempo de acceso a estos datos, solo la CPU puede utilizar el bus de datos. Suelen durar varios ciclos de reloj y suelen durar varios ciclos de reloj.



Rendimiento (ejemplo)

Tiempos de ejecución:

- Acceso a memoria: 8 ns.
- ALU y sumadores: 2 ns.
- Acceso a registros: 1 ns.

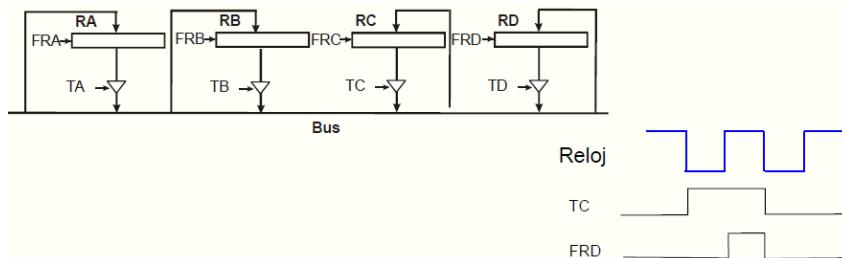
En cada ciclo de reloj se realiza una instrucción. Los ciclos no pueden ser de longitud variable (cada ciclo dura lo mínimo necesario para ejecutar su respectiva instrucción) porque si no, no habría sincronización.

3.1. OPERACIONES ELEMENTALES

Son **operaciones realizables directamente por hardware** en la que la UC divide cada una de las fases de ejecución de una instrucción. Se realizan por la UC mediante **activación de señales de control**. Cada microoperación dura un ciclo de reloj (excepto la Mp). Se pueden simultanear en el tiempo cuidando el orden preestablecido y conflictos por hardware.

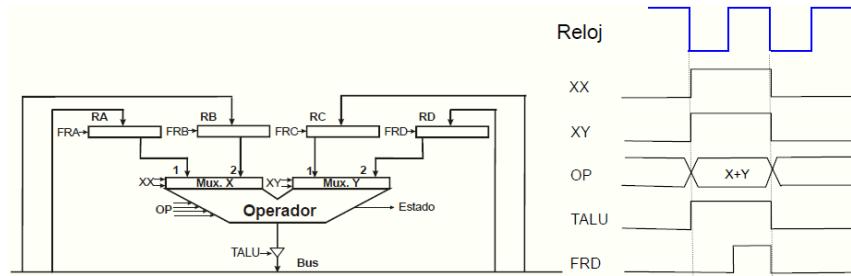
Tipos:

- **De transferencia:** Llevan información de origen a destino. Se establece el camino físico entre salida de origen y entrada de destino. Y se envía la señal al destino para que tome la información. Supongamos como ejemplo que en la siguiente imagen queremos copiar de C a D (C → D).



El **triestado TC** (aisla el registro C del bus de datos) se activa, envía una señal a la **señal de entrada del destino** (FRD) y al acabar el ciclo ya se habrá efectuado la transferencia.

- De proceso:** Llevan información de origen y destino, pero ésta pasa por un operador. Por ejemplo, $A+C \rightarrow D$



XX = Señal de multiplexor que elige el dato A (0 = A, 1 = B).

XY = Señal de multiplexor que elige el dato C (0 = C, 1 = D).

OP = Señal que indica la operación suma

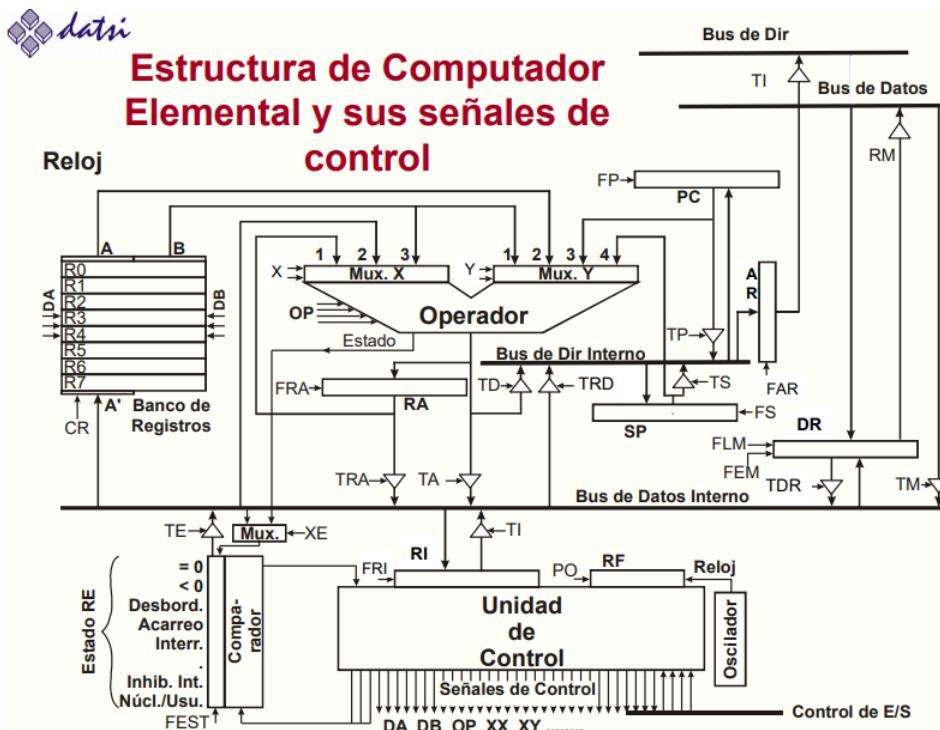
TALU = Señal de triestado que permite la salida de datos de la ALU

FRD = Señal de entrada de datos del destino (registro D)

Reglas de agrupación de operaciones:

- Respetar el orden establecido de algunas acciones. Por ejemplo, antes de leer guardar en el registro de direcciones (AR).
- Evitar conflictos en el mismo recurso físico. Un dispositivo no puede estar en dos estados diferentes al mismo tiempo. Por ejemplo, la memoria no puede leer y escribir al mismo tiempo, la ALU no puede realizar dos operaciones distintas simultáneamente.
- La información debe guardarse en algún sitio. Origen y destino deben poder almacenar información (Registros o Mp).

3.2. ESTRUCTURA DE COMPUTADOR ELEMENTAL Y SUS SEÑALES DE CONTROL



- Todos los registros tienen triestados de salida que no permiten que se vierta información al bus de datos accidentalmente (ruido).
- **Banco de registros:** Tienen dos puertas de entrada (DA y DB), con 3 hilos cada uno de codificar los 8 registros que hay dentro. Existe la opción también de cargar datos desde el bus de datos (A') al registro indicado por la puerta A (gobernado por la señal de control del registro CR).
- **Contador de programa (PC):** Conectado al bus de direcciones interno para llevar la dirección de la siguiente instrucción a ejecutar al **registro de direcciones (AR)**.
- **Puntero de pila (SP):** Conectado a bus de direcciones interno.
- **Lectura de instrucciones y datos:** Cuando se lee una instrucción de memoria, llega al **registro de datos (RD)** y llega al **Registro de Instrucciones (RI)** a través del bus de datos.
- **Registro / contador de fases (RF):** Cuenta los ciclos pasados, gobernado por la señal de reloj
- **Unidad de control:** Activa las señales de entrada del banco de registros (DA, DB), del Operador (ALU) y sus multiplexores (XX, XY),

3.3. CRONOGRAMAS

Consisten en la división de una instrucción en operaciones elementales, organizado por ciclos de reloj.

Por ejemplo: **ADD .R4, .R7**

- **Memoria principal**
 - $04B4 \rightarrow 5447$
 - $04B6 \rightarrow 7B35$
- $H'54 =$ código de suma: 4; 7; $\rightarrow 5447 =$ codificación de la instrucción de suma.
- **Ciclo 1:**
 - PC (04B4) \rightarrow Bus de direcciones \rightarrow AR (se carga con el flanco de bajada)
- **Ciclo 2: (1º ciclo de memoria):**
 - AR (04B4) \rightarrow Mp (se busca el contenido asociado a la dirección).
 - Se activan las señales de CM (ciclo de memoria) y L (lectura).
- **Ciclo 3 (2º ciclo de memoria + preparación siguiente instrucción):**
 - Mp(AR) \rightarrow Bus de datos (5447) \rightarrow RI (La lectura del dato en memoria ha durado dos ciclos de reloj). Se desactiva la señal CM, pero no la señal L.
 - PC + 2 (04B6) \rightarrow Bus de direcciones \rightarrow AR se incrementa en 2 porque las palabras ocupan dos direcciones (la siguiente instrucción está en la siguiente palabra). La operación de incremento de PC en la ALU no modifica los "flags" del Registro de Estado (RE).
 - (Nuevo PC \rightarrow AR sólo puede hacerse mientras no esté en uso la dirección que esté en AR).
- **Ciclo 4 (Decodificación):**
 - RF = 0 (en flanco de bajada). Se desactivará la señal L.
- **Ciclo 5 (ejecución de suma y fetch de siguiente instrucción):**

- $5447 = 54 / 4 / 7$
 - $\rightarrow 54 \rightarrow OP \rightarrow Suma$
 - $\rightarrow 4 \text{ y } 7 \rightarrow DA \text{ y } DB \rightarrow BR(R4 \text{ y } R7)$
 - $\rightarrow R4 = 04F2 \text{ y } R7 = 003C.$
 - $\rightarrow R4 \text{ y } R7 \rightarrow A \text{ y } B \rightarrow ALU \rightarrow RE$ (modifica biestables de estado aritméticos).

- Flanco de bajada: $R4+R7 \rightarrow$ Bus de datos $\rightarrow R4$ (052E); y RF suma fases

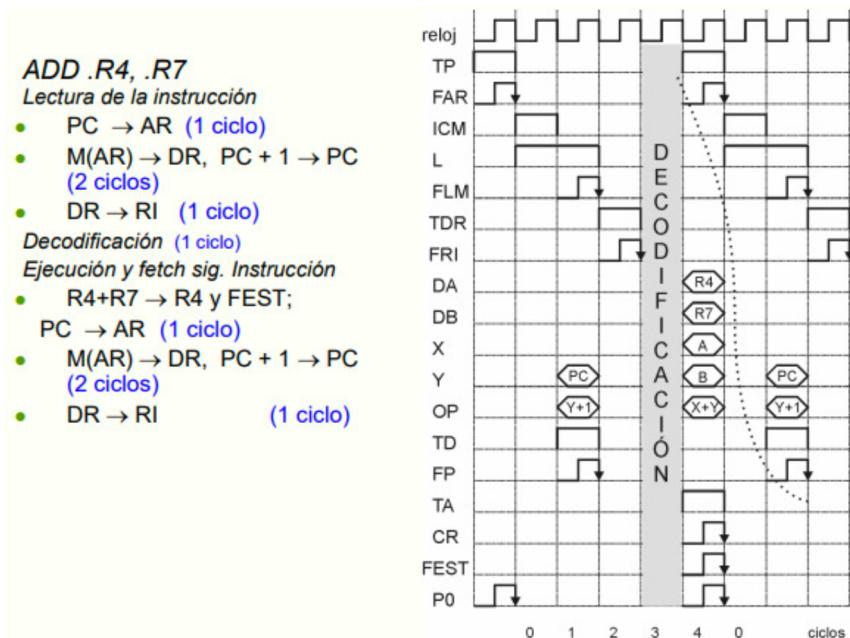
- Ciclo 6:

- Acabada la operación de suma, se lee la siguiente instrucción en memoria. AR (04B6) $\rightarrow Mp$. Se activan las señales CM (ciclo de memoria) y L (lectura).

- Ciclo 7:

- $Mp(AR) \rightarrow$ Bus de datos (7B35) $\rightarrow RI$ (la lectura del dato en memoria ha durado dos ciclos de reloj). Se desactiva la señal CM, pero no la señal L
- $PC + 2$ (04B8) \rightarrow Bus de direcciones $\rightarrow AR$. La operación de incremento de PC en la ALU no modifica los "flags" del Registro de Estado (RE)
- (El contador de fases se incrementa con cada flanco de bajada de la señal de reloj).

El cronograma que indica qué señales están activas a lo largo de las fases contadas por RF



3.4. DISEÑO DE LA UNIDAD DE CONTROL

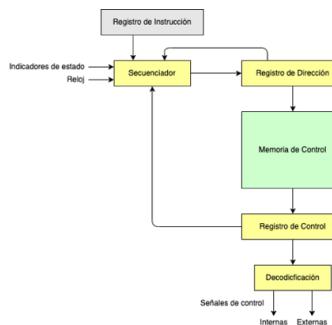
El diseño debe haber definido previamente el repertorio de instrucciones (formatos y direccionamientos) a nivel de cronogramas o de operaciones elementales. La UC funciona como un traductor entre entradas y salidas.

La UC funciona como un **traductor** cuyas señales de control son: **Registro de Instrucción (RI)**, que lleva el Código de Operación CO y Modos de Direccionamiento MD, **reloj** que marca la fase en la que estamos, el **registro de Estado (RE)** que indica el estado de la última operación aritmético-lógica y las **señales externas** que sirven si algún periférico necesita atención por parte del procesador.

Formas de diseño de la UC:

- **UC Cableada:** Utiliza exclusivamente puertas lógicas. Se construye mediante métodos generales del diseño lógico. Emplea una arquitectura RISC, son más rápidas y tienen menos componentes (no tiene secuenciador ni memoria de control, etc), pero su diseño y construcción es complejo y costoso.
- **UC Microprogramada:** Utiliza una **memoria de control** para almacenar el estado de las señales de control en cada periodo de ejecución de cada instrucción. Generar el cronograma de cada instrucción (ejecutar la instrucción) es ir leyendo de esta memoria de control. Usa una arquitectura CISC, son más lentas que las cableadas ya que tienen que realizar operaciones de lectura de una memoria para obtener las señales de control.

- **Partes de la UC micropogramada:**



- **Secuenciador:** Decodifica el CO (Código de Operación) que indica qué instrucción hay en ejecución (ADD, SUB, CALL, ...) De esta información se sabrá a partir de qué microinstrucción de la Memoria de Control (MC) se deberá empezar a trabajar.
- **Registro de Dirección:** Microcontador de programa (microPC), recorre la secuencia de la instrucción.
- **Memoria de Control:** Incluye los valores para todas las señales a generar. Una vez conocida la instrucción de ejecución, escoge una línea de su memoria de control (ristra de 1's y 0's) que envía al Registro de Control.
- **Registro de Control:** Ristras de 0's y 1's que indican qué señales activar o desactivar.
- **Microinstrucción:** Cadena de ceros y unos que representa la activación o no del conjunto de señales de control durante un ciclo de reloj y la posibilidad de decidir condicionalmente qué instrucción se debe ejecutar a continuación. Cada palabra de la memoria de control es una microinstrucción. Tienen el siguiente formato:

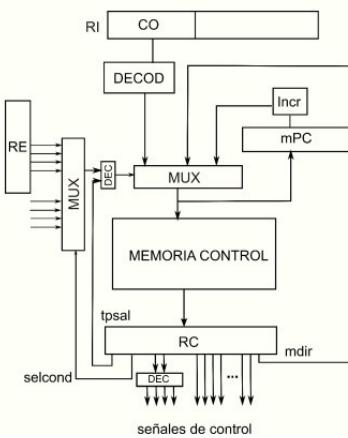
- **Señales de control:** Señales para el encaminamiento de datos.
- **Tipo de secuenciamiento:** Si la next instrucción es la continuación o un salto.
- **Condiciones:** Bits para seleccionar la condición que se dese utilizar para en función de si es cierta o no ejecutar la siguiente microinstrucción o saltar a otra.
- **Siguiente microinstrucción:** Indica la siguiente a ejecutar.

Señales de control	Tipo Sec.	Cond	Siguiente píns

Para **diseñar** el formato de microinstrucción se comienza con la lista completa de señales de control. Se agrupan las señales de control de función similar en

un mismo campo. Se puede aplicar codificación para reducir el tamaño de la microinstrucción y para evitar que ciertas señales se activen al mismo tiempo (evitar errores de acceso a bus, operación de memoria, ...).

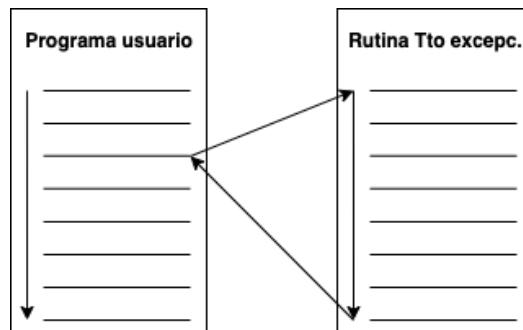
- **Micropograma:** Secuencia de microinstrucciones cuya ejecución permite interpretar una instrucción. Cambiando micropogramas se pueden ejecutar otros repertorios de instrucciones (simular otras arquitecturas), es decir, emular.
- **Memoria de control (MC):** Memoria que almacena micropogramas a partir de los que se obtienen las señales de control necesarias para la ejecución del repertorio de instrucciones. Suelen ser ROM (read only memory).
- **Estructura básica de la UC microprogramada:**



3.5. CONTROL DE EXCEPCIONES

Además de ejecutar la secuencia de instrucciones, la UC debe controlar situaciones excepcionales:

- **Excepción:** Evento no planificado que para la ejecución de un programa. Se salvan el PC y el estado actual en la pila, se pasa a modo SO (modo supervisor), se salta a rutina de SO que maneje excepciones y SO retorna al programa su anterior estado. Las excepciones las causan eventos internos (overflow, error de alineamiento de memoria, fallo de página, ...). Son síncronos a la ejecución del programa (se puede saber cuándo ocurren). Puede que el programa se retome tras la excepción o que se aborde.



- **Interrupción:** Un evento externo inesperado rompe la ejecución del programa. Son asíncronas a la ejecución del programa (pueden ocurrir en cualquier instante). Se atienden entre instrucciones. Mantienen sin ejecutarse un tiempo el programa.

4. ARITMÉTICA

4.1. INTRODUCCIÓN

El computador recibe una serie de **información** que debe procesar para lanzar unos **resultados**. Se trata de los **datos e instrucciones** que vienen definidos por los **símbolos**: letras, números, caracteres... y las **ideas**: operaciones, movimientos, modificaciones...

Además, el computador se ve limitado por la utilización del **sistema binario**. Es finito, así que las **representaciones** de información son **acotadas**. Por el diseño de sus unidades funcionales existen **tamaños privilegiados** (byte, palabra, ...).

Existen varios modos de representación que nos permiten representar de forma gráfica la información. Así, tenemos representaciones alfanuméricas, numéricas, redundantes, gráficas y etiquetadas.

Representaciones alfanuméricas:

El más común es el **ASCII**.

- **ASCII 7 bits:** Es el conjunto de caracteres ASCII original, cada uno se representa utilizando 7 bits, teniendo un rango de 0 (0000000) a 127 (1111111), un total de 128 caracteres (2^7). Con esto podemos representar:
 - **Las 26 letras del alfabeto (mayúsculas y minúsculas):** Desde H'41 (A) hasta H'5A (Z) y desde H'61 (a) hasta H'7A (z)
 - **Los 10 dígitos decimales:** Desde H'30 (0) hasta H'39 (9)
 - **Un conjunto de caracteres especiales (+ , - = ...)**
 - **Un conjunto de caracteres de control (no visibles):** Desde H'00 (NUL) hasta H'1F (US). Excepción H'7F (DEL).

Bastan 7 bits para todos los símbolos ASCII. En principio el 1er bit se deja a 0 (0111111). Se utiliza para representar caracteres especiales según el idioma. En casos extremos se añaden bits adicionales.

- **ASCII 8 bits:** Norma ISO que añade la representación de caracteres no presentes en inglés. Por ejemplo, las de raíz latina (tildes, cedilla, ...). Se le conoce como Extended ASCII y su rango de representación va del 0 al 255 (1111111) en un total de 256 caracteres (2^8). **Ejemplo:**
 - Á (H'C1), É (H'C9), á (H'E1), é (H'E9).
- **UTF-8 (8-bit Unicode Transformation Format):** Se trata de la codificación de un carácter con varios bytes.
 - **Codificación de un byte:** carácter es ASCII de 7 bits.
 - **Codificación de dos bytes:** carácter es de lenguas romances y otras
 - **Ejemplo:** á es H'C3 H'A1 (2 bytes).

á (UTF-8): 1100 0011 (0xC3). 1010 0001 (0xA1).
 - **Codificación de tres bytes:** Carácter es de lenguas asiáticas.
 - **Codificación de cuatro bytes:** otros.

Representaciones numéricas:

Tienen ciertas limitaciones:

- **Número finito de valores representables:** Hay un rango de representación, es decir, un intervalo entre el mayor y el menor número representables.
- **Número finito de bits para la representación:** Hay una resolución, es decir, una diferencia entre dos valores representables consecutivos.
- **Operaciones con resultados no representables:** Habrá desbordamiento, es decir, cuando un resultado esté fuera del rango de representación.

Las representaciones numéricas son sistemas posicionales con base b. (Nº natural > 1).

Ejemplos de cambios de base:

- **Binario a decimal**
 - $(101.11)_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} = 1 \times 4 + 0 \times 2 + 1 \times 1 + 1/2 + 1/4 = (5.75)_{10}$
- **Decimal a binario**
 - $(11.25)_{10} = ???_2$
 - **Parte entera:** Dividimos por la base

$$\begin{array}{r} 11|2 \\ 5|2 \\ 1|2 \\ 0|1|2 \\ 1|1 \end{array}$$

- **Parte decimal:** Multiplicamos por la base

$$\begin{aligned}
 0.25 \times 2 &= 0.5 \\
 0.5 \times 2 &= 1.0 \\
 0.0 \times 2 &= 0.0 \\
 \dots
 \end{aligned}$$

Nos da un resultado de $(1011.010)_2$

- **Hexadecimal a decimal**

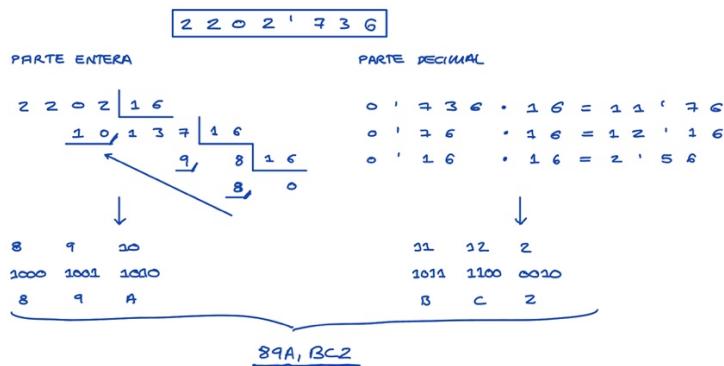
- $(A27.8C)_{16} = 10 \times 16^2 + 2 \times 16^1 + 7 \times 16^0 + 8 \times 16^{-1} + 12 \times 16^{-2} = (2599.546875)_{10}$

- **Decimal a hexadecimal**

- $(2202,736)_{10} = (???)_{16}$

- **Parte entera:** Se divide sucesivamente entre la nueva base (16) y pasa de decimal a hexadecimal los restos (de dcha a izqda)

- **Parte decimal:** La multiplica sucesivamente por la nueva base (16) y pasa de decimal a hexadecimal la parte entera del resultado.

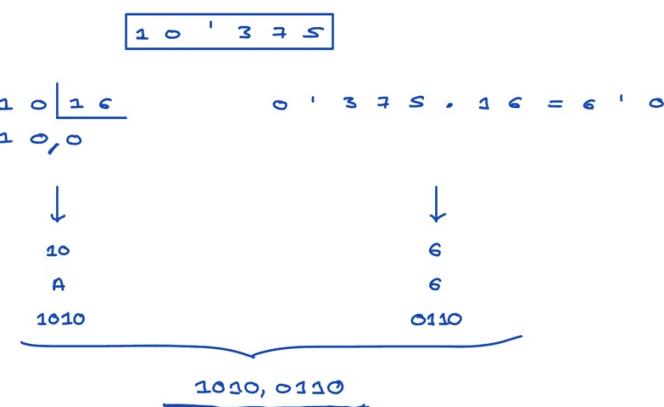


- **Truco para pasar de decimal a binario**

- $(10.375)_{10} = (???)_2$

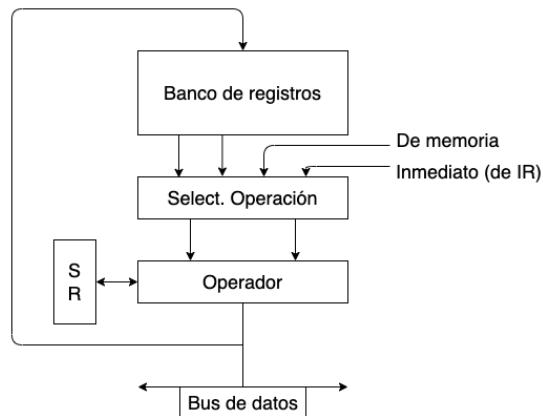
- Para pasar un número a binario, en vez de multiplicar y/o dividir por 2 sucesivamente, se puede realizar un cambio de base intermedio a hexadecimal y convertir cada bit hexadecimal a 4 bits binarios.

- 8 bits entera y decimal: si el número no ocupa "funcionalmente" los 8 bits, se rellena con ceros a izquierda (entera) o derecha (fracción).



Operaciones en la ALU:

- **Componentes de la ALU:**
 - **Operador:** Circuito que realiza una operación.
 - **Registro de estado (SR):** Los flags más usuales son acarreo (C), cero (Z), signo (S), desbordamiento (V), paridad (P), resta (N), acarreo BCD (H).
 - **Estructura de la ALU:** Tiene un modelo de ejecución memoria – memoria.



- **Operaciones lógicas:** NOT, OR, AND, XOR, ... Actúan sobre los operandos bit a bit.
 - $(1001) \text{ XOR } (0101) = 1100$.
- **Desplazamientos:**
 - **Lógicos:** Rellenan los huecos generados con ceros, ya sean a la izq o a la dcha.
 - **Aritméticos:** Realizan la multiplicación por 2 (a la izq) o división por 2 (a la dcha). Dependen de la representación:
 - **Multiplicación en Complemento a 2:** Se rellena el hueco con 0 y hay desbordamiento si cambia de signo. Esto es porque el bit más significativo marca el signo (0 + y 1 -).
 - **División en complemento a 2:** Si es menor que 0 se pone un 1 en el hueco generado.
 - **Concatenados entre registros y con biestables (acarreo).**
 - **Circulares o rotaciones.**
- **Extensión de signo:** Representa un dato de n bits con m bits, tal que m > n. Depende del sistema de representación. En complemento a 2 si el número es negativo se llenan con 1s todos los m-n bits añadidos.
- **Cambio de signo:** Dado un número a se obtiene -a. Depende del sistema de representación.
- **Suma/Resta:** Depende del sistema de representación.

En cada formato de representación, habrá que definir:

1. Representación y valor.
2. Rango y resolución (mínima diferencia entre un nº representable y el siguiente).
3. Cambio de signo ($X \rightarrow -X$)

4. Desplazamientos aritméticos.
5. Extensión de signo ($m > n$, con $n =$ longitud en bits de la representación del número).
6. Suma/Resta (incluido OVF).

4.2. REPRESENTACIÓN EN COMA FIJA

4.2.1. Binario puro (sin signo):

- 1) Representación:** $(X_{n-1}, X_{n-2}, \dots, X_1, X_0)$
- 2) Rango:** $[0, 2^n - 1]$ y **Resolución:** 1
- 3) Cambio de signo:** N/A
- 4) Desplazamiento aritmético:** Desplazamiento lógico
- 5) Extensión de signo:** añadir ceros a la izquierda
- 6) Suma/Resta:**
 - **Suma: $A + B$:** Se suman los números bit a bit normal.
 - **Resta: $A - B$:** $A + [(2^n - 1 - B) + 1] - 2^n$
 - Se suma A y la inversa de B bit a bit ($A + (-B)$). ($-B = 2^n - 1 - B$)
 - Se suma 1 a la inversa de B por el acarreo inicial (*Confirmar pq este 1 no me cuadra)
 - **Desbordamiento (OVF) con CY (biestable de acarreo):**
 - **Carry:** Llevada en una suma.
 - $C_{n-1} = 1$ y S/R = 0 ($CY = 1$, carry = 1).
 - **Borrow:** Llevada en una resta.
 - $C_{n-1} = 0$ y S/R = 1 ($CY = 1$, borrow = 1).
 - Desbordamiento = Cd una suma/resta se sale del rango de representación.

4.2.2. Complemento a 2:

Es posible representar números positivos y negativos

- 1) Representación:** $(X_{n-1}, X_{n-2}, \dots, X_1, X_0)$
- **Bit de signo:** El bit más significativo $\rightarrow X_{n-1} = 0$ (positivo) y 1 (negativo).
 - Si $X_{n-1} = 0 \rightarrow X \geq 0$. Igual que binario puro.
 - Si $X_{n-1} = 1 \rightarrow X < 0$. $Rep(X) = 2^n - |X|$.
 - $Rep(X) + Rep(-X) = 2^n$
- 2) Rango:** $[-2^{n-1}, -1] \cup [0, 2^{n-1} - 1]$.
 - Rango de representación asimétrico.
 - Representación del cero única.
 - **Resolución:** 1
- 3) Cambio de signo:** Aplicar el complemento a 2.
 - **Ejemplo:** $n = 6$; Representar $A = 7 = 000111$; Calcular valor $B = 101110$
 - ($-A$ se representa invirtiendo los bits de A y sumando 1)

- $-A = 2^n - |A| = 1000000 - 000111 + 1 = \textbf{111001}$
- $|B| = 1000000 - 101110 = 010010 = 18 \rightarrow B = \textbf{-18}.$
- $n = 6$. Valor máximo = $011111 = 2^5 - 1 = 31$
- $n = 6$. Valor mínimo = $100000 = 2^5 = -32$

4) Desplazamientos aritméticos

- **Izquierda (x2):** Se rellena el hueco con 0 y hay desbordamiento si cambia de signo.
- **Derecha (/2):** Se rellena con 0 o 1 dependiendo de si es positivo o negativo.

5) Extensión de signo: (Pasar de un número de m bits y n bits sin variar el valor):

- Si $X > 0$:
 - Binario puro → Añadir $n-m$ ceros a la izquierda.
- Si $X < 0$:
 - TBD

6) Suma/Resta:

- **Suma A + B:**

	A	B	A+B	C _{n-1}	OVF
A+ B+	a	b	a+b	0	S _{n-1} =1 C _{n-2} =1
A- B-	2 ⁿ - a	2 ⁿ - b	2 ⁿ +2 ⁿ - (a+b)	1	S _{n-1} =0 C _{n-2} =0
A+ B- (a>=b)	a	2 ⁿ - b	2 ⁿ + (a-b)	1	NO
A+ B- (a<b)	a	2 ⁿ - b	2 ⁿ - (b-a)	0	NO

- **Resta A - B: (A + (-B)):** $A + [2^n - 1 - B] + 1$

4.2.3. Complemento a 1: También es posible representar números positivos y negativos

1) Representación: ($X_{n-1}, X_{n-2}, \dots, X_1, X_0$)

- **Bit de signo:** El bit más significativo → $X_{n-1} = 0$ (positivo) y 1 (negativo).
 - Si $X_{n-1} = 0 \rightarrow X \geq 0$. Igual que binario puro.
 - Si $X_{n-1} = 1 \rightarrow X < 0$. Rep(X) = $2^n - 1 - |X|$.
 - Rep(X) + Rep(-X) = $2^n - 1$

2) Rango: $[-(2^{n-1} - 1), -0] \cup [0, 2^n - 1]$.

- Rango de representación simétrico.
- Representación del cero doble: 000...000 y 111...111.
- **Resolución:** 1

3) Cambio de signo: Aplicar el complemento a 1.

- **Ejemplo:** $n = 6$; Representar $A = -7 = 000111$; Calcular valor $B = 101110$
- ($-A$ se representa invirtiendo los bits de A y sumando 1)
 - $-A = 2^n - |A| = 1000000 - 000111 = \textbf{111000}$
 - $|B| = 111111 - 101110 = 010001 = 17 \rightarrow B = \textbf{-17}$
 - $n = 6$. Valor máximo = $011111 = 2^5 - 1 = 31$

- o $n = 6$. Valor mínimo = $100000 = -011111 = -(2^5 - 1) = -31$

4) Desplazamientos aritméticos

- **Izquierda (x2)**: Se rellena el hueco con 0 y hay desbordamiento si cambia de signo.
- **Derecha (/2)**: Se rellena con 0 o 1 dependiendo de si es positivo o negativo.
- 5) **Extensión de signo**: Se llenan con 0s o con 1s los $m-n$ bits dependiendo de si el número es positivo o negativo (igual que en ca2).

6) Suma/Resta:

- **Suma A + B**:

	A	B	A+B	C _{n-1}	OVF
A+ B+	a	b	a+b	0	S _{n-1} =1 C _{n-2} =1
A- B-	2 ⁿ -1- a	2 ⁿ -1- b	2 ⁿ -1 + 2 ⁿ -1-(a+b)	1	S _{n-1} =0 C _{n-2} =0
A+ B- (a>=b)	a	2 ⁿ -1- b	2 ⁿ -1 +(a-b)	1	NO
A+ B- (a<b)	a	2 ⁿ -1- b	2 ⁿ -1-(b-a)	0	NO

- **Resta A - B: (A + (-B))**: $A + [2^n - 1 - B]$

4.2.4. Signo-magnitud: Se representa el bit de signo por un lado y la magnitud por otro.

1) Representación: ($X_{n-1}, X_{n-2}, \dots, X_1, X_0$)

- **Bit de signo**: El bit más significativo $\rightarrow X_{n-1} = 0$ (positivo) y 1 (negativo).
 - o Si $X_{n-1} = 0 \rightarrow X \geq 0$. Igual que binario puro.
 - o Si $X_{n-1} = 1 \rightarrow X \leq 0$.

2) Rango y resolución igual que en complemento a 1.

- Representación del cero doble: 000...000 y 100...000.

3) Cambio de signo: Cambiar el bit de signo y ya.

- **Ejemplo**: $n = 6$; Representar $A = -7 = 000111$; Calcular valor $B = 101110$
 - o **A = 000111; -A = 100111**
 - o $B = -14 ; -B = 001110$
 - o $n = 6$. Valor máximo = $011111 = 2^5 - 1 = 31$
 - o $n = 6$. Valor mínimo = $111111 = -2^5 - 1 = -31$

4) Desplazamientos aritméticos: (creo que) igual que en complemento a 1.

5) Extensión de signo: Añadir ceros entre el bit de signo y los bits de magnitud

6) Suma/Resta:

- **Suma A + B**: TBD
- **Resta A - B**: $A + (-B)$

4.2.5. Exceso a "M": Binario puro con desfase

TBD

4.3. REPRESENTACIÓN EN COMA FLOTANTE

1) Representación: $V(X) = M \times r^E$ (notación científica)

- **M** = Mantisa o fracción (p bits)
- **r** = base
- **E** = exponente (q bits)
- **Representación:** $(e_{q-1} e_{q-2} e_1 e_0 m_{p-1} m_{p-2} \dots m_1 m_0)$

- **Características:**

- **Mantisa** = coma fija con signo y base r.
- Normalmente r = 2^k (2, 8, 16).
- Exponente = entero y base 2.

- **Distribución de bits:**

- 1 para representar el signo, 7 para el exponente y 8 para la mantisa.
- Entre los bits del exponente y la mantisa va una coma.
- Rango de la mantisa: 8 bits: [00...00, 11...11] $\rightarrow [0, 1 - 2^{-8}]$
 - Mayor número positivo: $0,11\dots11 (+ 0,00\dots01 (2^{-8}) = 1.00000000) = 1 - 2^{-8}$
- Rango del exponente: 7 bits: 2^7 posiciones = 128 $\rightarrow [0, 127] - 64 (M) \rightarrow [-64, 63]$.

2) Rango y resolución:

- Rango: $\pm [2^{-8} * 2^{-64}, (1 - 2^{-8}) * 2^{63}] \cup 0$.
- Resolución: $2^{-8} * 2^E$, donde E es el exponente de la representación.

S	EXPONENTE	MANTISA
1	7	8

- **Ejemplo:**

- $A = H'C63C = 1\ 1000110, 00111100$
 - $A = -,00111100 * 2^6 = -15_{(10)}$
 - Cálculo lento: $M = ,00111100 = 2^{-3} + 2^{-4} + 2^{-5} + 2^{-6}$
 - Cálculo rápido: $M = ,0011\ 1100 = 3 * 16^{-1} + 12 * 16^{-2}$
 - $A = ,01111000 * 2^5 = -15_{(10)} \rightarrow A = H'C578$
 - $A = ,11110000 * 2^4 = -15_{(10)} \rightarrow A = H'C4F0$

- **Normalización:** Como hemos visto, hay varias maneras de representar un valor. Diremos que la mantisa está normalizada si el bit más significativo de la mantisa es un 1.

- **Rango:** $-10000000 = 2^{-1}; -11111111 = 1 - 2^{-8} / \pm [2^{-1} * 2^{-64}; (1-2^{-8}) * 2^{63}] \cup 0$.
- **Resolución:** $2^{-8} * 2^e$; e = exponente de la representación
- **Problemas de normalización:**

- Resultados de operaciones no normalizados, así que deberemos postnormalizarlos.
- El cero no es representable, dado que el 00000000 no es una mantisa normalizada.
- **Con bit implícito:** Si $r = 2$ y mantisa en signo magnitud y normalizada, entonces puede dejarse el bit más significativo como implícito, porque tiene que ser 1 (normalización).

S	EXPONENTE	MANTISA
1	7	1 8

- **Rango:** $-1\ 0000\ 0000 = 2^{-1}; -1\ 1111\ 1111 = 1 - 2^{-9} / \pm [2^{-1} * 2^{-64}; (1-2^{-9}) * 2^{63}]$
- **Resolución:** $2^{-9} * 2^e$; e = exponente de la representación.

3) Suma/Resta: Pasos a seguir:

- Comparar exponentes
- Desplazar mantisa de exponente menor a la derecha
- Sumar/Restar mantisas
- Si el resultado es cero, FIN
- Normalizar mantisa (si redondeo postnormalizar)
- Ajuste del exponente
- Detectar desbordamiento

4) Redondeo: Ejemplo: Se busca **ajustar** una mantisa de 10 bits ($M = -100100\ 1011$) a una mantisa de 6 bits mediante **técnicas de redondeo**:

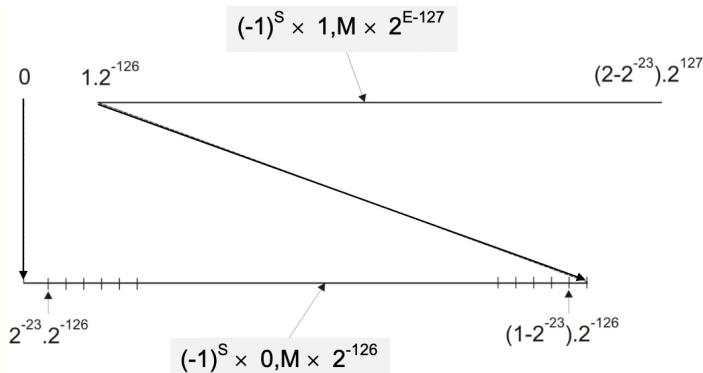
- **Truncamiento:** Suprimir los bits sobrantes
 - **Ejemplo:** $-100100\ 1011 \rightarrow -100100\ 1011 \rightarrow -100100$
 - **Error absoluto:** $< 2^{-n}$ (por defecto). En este caso, $n = 6$.
- **Forzado a 1:** Truncamiento, pero dejando a 1 el bit más significativo
 - **Ejemplo:** $-100100\ 1011 \rightarrow -100100\ 1011 \rightarrow -100100 \rightarrow -100101$
 - **Error absoluto:** $< 2^{-n}$ (por exceso y por defecto). En este caso, $n = 6$.
- **Redondeo al más próximo:** Ajustar a un extremo u otro, sumando 1 al bit $n+1$.
 - **Ejemplo:** $-100100\ 1011 + -000000\ 1000 \rightarrow -100101\ 0011 -100101$
 - **Error absoluto:** $1/2 * 2^{-n}$ (por exceso y por defecto). En este caso, $n = 6$.
- **Redondeo al par más próximo (nearest even):** Ajustar al extremo par.
 - **Ejemplo:** $-100100 < -100100\ 1011 < -100101 \rightarrow -100100\ 1011 \rightarrow -100100$
- **Redondeo a cero:** Ajustar la mantisa al extremo más cercano a 0.
 - **Ejemplo:** $-100100 < -100100\ 1011 < -100101 \rightarrow -100100\ 1011 \rightarrow -100100$
- **Redondeo a $+\infty$:** Ajustar la mantisa al extremo mayor.
 - **Ejemplo:** $-100100 < -100100\ 1011 < -100101 \rightarrow -100100\ 1011 \rightarrow -100101$
- **Redondeo a $-\infty$:** Ajustar la mantisa al extremo menor.
 - **Ejemplo:** $-100100 < -100100\ 1011 < -100101 \rightarrow -100100\ 1011 \rightarrow -100100$

- 5) Post normalización:** Desplazar la mantisa una vez realizado un redondeo por exceso.
- 6) Dígitos de guarda:** Dígitos añadidos a la mantisa para obtener la precisión máxima. En el caso de mantisa en signo-magnitud se necesitarían dos bits de guarda, uno para normalizar el resultado y otro para redondeo.
- 7) Bit retenedor:** Un bit añadido para propagar la llevada (borrow) en la resta. Al realizar los desplazamientos en la mantisa de menor exponente en la operación suma / resta, el bit retenedor se pone a 1 en el momento que pase un 1 y permanece ese valor independientemente de los bits que pasen después.
- 8) Estándar IEEE 754 de coma flotante:**

S	EXPONENTE	MANTISA
1	8	l' 23

- Por norma general:
 - Exponente en exceso $2^{p-1} - 1$
 - Mantisa en signo-magnitud, normalizada, con bit implícito y la coma a la derecha del bit implícito.
- Depende de en qué precisión estemos trabajando, tendremos una distribución con más o menos bits:
 - Simple precisión (32 bits) → $p = 8, q = 23$.
 - Doble precisión (64 bits) → $p = 11, q = 52$.
 - Cuádruple (128 bits) → $p = 15, q = 112$.
- Tenemos los siguientes **tipos**:
 - Exponente $\neq 0$ ni 127 ($e = 00\dots01; 11\dots10$):
 - Números normalizados: $= (1. d_1d_2d_3\dots d_{23}) * 2^{[(e_1e_2e_3\dots e_8) - 127]}$
 - Exponente = 0 ($e = 00\dots00$):
 - Mantisa = 0:
 - ± 0
 - Mantisa $\neq 0$
 - Números denormalizados: $= (0. d_1d_2d_3\dots d_{23}) * 2^{-126}$
 - Exponente = 127 ($e = 111111111$)
 - Mantisa = 0
 - $\pm \infty$
 - Mantisa $\neq 0$
 - NaN
- **Rango:**
 - **Nº normalizados:** $\pm [2^{-126}; (2 - 2^{-23}) * 2^{127}] =$
 - $(\pm[(1.00\dots00) * 2^{[(00\dots01) - 127]}; (1.11\dots11) * 2^{[(11\dots10) - 127]}])$
 - **Nº denormalizados:** $\pm [2^{-23} * 2^{-126}; (1 - 2^{-23}) * 2^{-126}] =$

- $(\pm[(0.00\dots01) \times 2^{-126}, (0.11\dots11) \times 2^{-126}])$



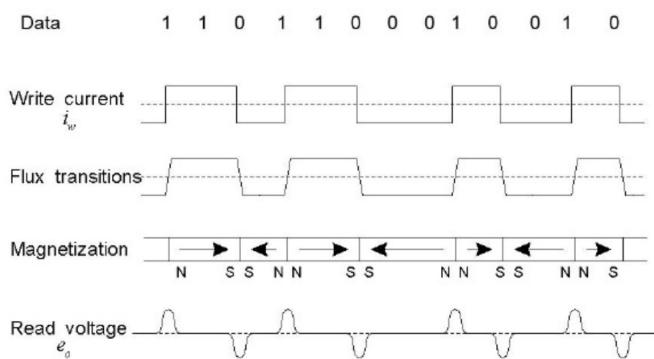
5. PERIFÉRICOS

5.1. INTRODUCCIÓN

La Unidad de E/S gobierna el intercambio de información entre el computador y los periféricos, pues un computador sin periféricos no puede comunicarse con el exterior. Pueden diferenciarse según su modo de funcionamiento, el formato y tamaño de los datos, la velocidad de transferencia y el tiempo de acceso. Los clasificamos según si son usados como herramientas de **almacenamiento** (cintas, discos (duros, de estado sólido, etc)) o **comunicación** (humanos: teclado, ratón, etc; computadores: redes, etc; medio físico: sistemas de control, empotrados, o incluso IoT). Ejemplos de estos pueden ser **domésticos** como el ratón, teclado, impresora, disco duro, etc; e **industriales** como un sensor de temperatura, de presión, motores para la orientación de un objeto, etc.

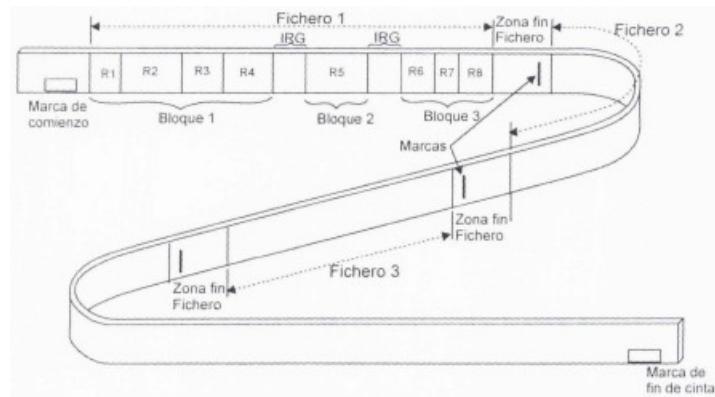
5.2. UNIDADES DE CONTROL DE CINTA MAGNÉTICA

Se usan principalmente para backup. Se divide la cinta en celdas de bit en las que se graba información de bit o información de sincronismo. La densidad de grabación se mide en bits por pulgada (bpi).

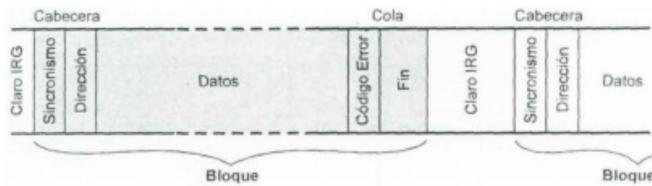


El cabezal graba en 9 pistas simultáneamente, 8 para el byte y 1 adicional de redundancia para detectar errores. Este bit adicional se llama bit de paridad y será 0 o 1 en función de si el byte tiene un número par o impar de 1s, lo que permite saber si es erróneo el dato o no.

Formato de grabación: En la siguiente imagen de ejemplo vemos que se ha guardado la información en 3 bloques, de forma que si se produce un error en un bloque solo hay que volver a leer ese bloque, no la cinta entera. Entre bloque y bloque se deja un inter record gap (IRG) donde no se graba nada, sino que sirve para el rotor de la cinta y volver a arrancarlo.



Además, la organización de los bloques tiene la siguiente estructura:



Donde las secciones de **Sincronismo** y **Fin** son secuencias de 1s y 0s que permiten delimitar bloques de información de un IRG; la **Dirección** es la identificación de los datos vendrán a continuación y **Código detector de error** verifica que los datos anteriores son correctos (leyendo el bit de paridad).

Como en la cinta habrá que incluir la información de sincronismo, de fin, la dirección y el código de detección de error, la capacidad neta (datos) será menor a la capacidad bruta calculada.

Fórmulas:

$$\text{VELOCIDAD DE TRANSFERENCIA (B/s)} = \text{Densidad de grabación lineal (DGL b/mm)} * \text{Velocidad lineal (m/s)}$$

$$\text{CAPACIDAD BRUTA (B)} = \text{Nº pistas} * \text{Longitud cinta (mm)} * \text{DGL (bits/mm)}$$

$$\text{LONGITUD CINTA} = \text{Nº de pistas} * \text{Lontigud pista}$$

$$\text{LONGITUD PISTA} = \frac{\frac{\text{Capacidad Bruta(B)} * 8 (\text{b/B})}{\text{DGL (\text{b/mm})}}}{\text{nº pistas}} \text{ (pistas paralelas)}$$

$$\text{LONGITUD BLOQUE} = \frac{\text{Capacidad bloque (B)} * 8 (\text{b/B})}{\text{DGL (\text{b/mm})}} + \text{IRL (\text{mm})}$$

$$\text{Tiempo de lectura de bloque} = \frac{\text{Longitud bloque (mm)}}{\text{Velocidad lineal (\frac{mm}{s})}} = \frac{\text{capacidad bloque (B)}}{\text{Velocidad de transferencia (B/s)}}$$

$$\text{Nº bloques} = \frac{\text{Longitud cinta} * \text{Nº pistas}}{\text{Longitud bloques}}$$

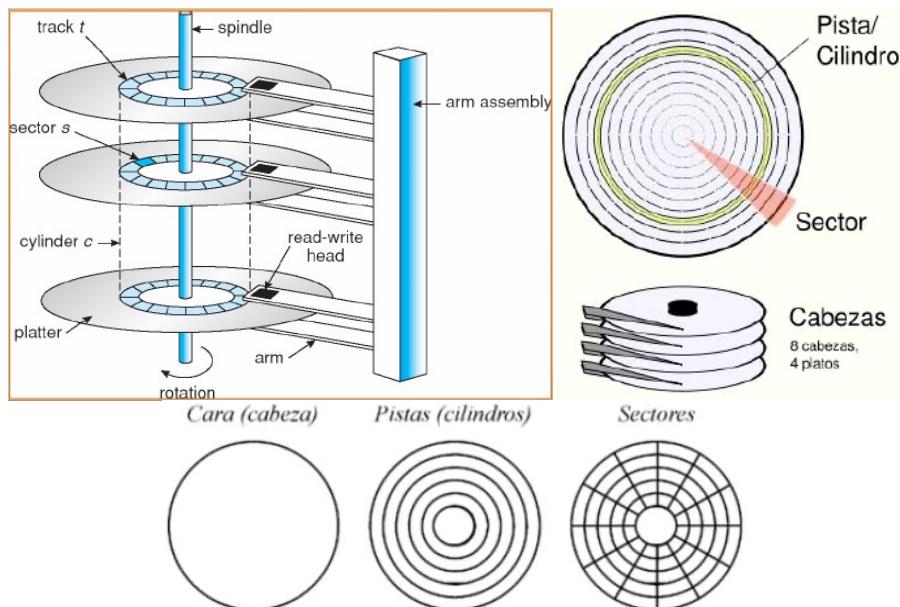
5.3. UNIDADES DE DISCO MAGNÉTICO (HDD)

Son dispositivos de **tipo bloque**, al igual que las cintas magnéticas. (Si se leen o se escriben datos, debe realizarse por bloques (no por sectores de bloque)). Los **bloques reciben el nombre de sectores**. Lo ideal es que los ficheros que ocupen más de un bloque estén en sectores consecutivos (Si un dato ocupa 3 ½ bloques, habrá que escribir en 4 bloques).

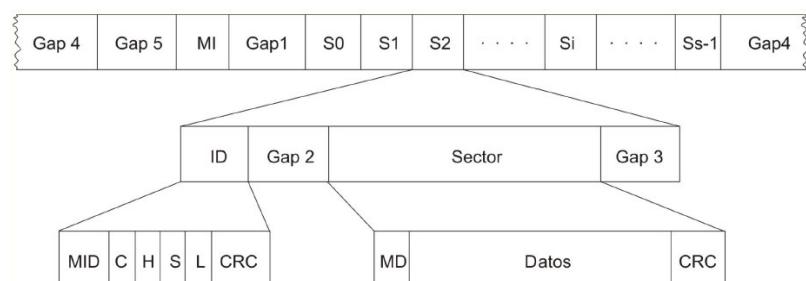
Las coordenadas de la información se componen de **cilindros (C) o pistas, superficies (H) o caras y sectores (S)**. En discos antiguos todas las pistas tenían el mismo número de sectores, pero hoy en día

las pistas más externas tienen un mayor número de sectores, ya que son más largas. Además, contamos con una **cabeza lectora – escritora** (head) por cada superficie y el **acceso a un sector** se realiza mediante la localización de estas coordenadas.

Se trata de dispositivos de acceso semi-directo, pues se mueve el brazo al cilindro de pistas con el dato (directo), se activa la cabeza y se selecciona la superficie y se busca el sector que almacena el dato (no directo: el disco gira y muestra cada dato secuencialmente).



Formato de grabación: La imagen siguiente representa una pista



Si: **Sectores** que están compuestos por **identificador (ID)** o dirección, Gap 2 o **IRG, sector**, y Gap 3.

Los **identificadores** se componen de: Zona de sincronismo (MID), cilindro (C), cabeza (H), sector (S), longitud (L) y código del error (CRC); y los **sectores** de: zona de sincronismo (MD), datos y CRC.

La cabeza lee secuencialmente los sectores de un disco, comprobando el ID de cada sector para comprobar si es el dato que se busca. Cuando el ID coincide con el dato buscado, la cabeza pasa a modo de escritura y se mete en los datos del sector.

Estos discos cuentan con una serie de **parámetros**: La **capacidad** (de 500GB por ej), **velocidad de transferencia** (70 MB/s), el **tiempo de acceso** (5 ms) y **otras** como la **densidad de grabación lineal** (bits/pulgada) y la **densidad de grabación angular** (bits/radián).

Funcionamiento: El motor gira siempre a la misma velocidad de rotación en el mismo sentido. El brazo se mueve hasta el cilindro destino ($t_{busqueda}$). Una vez en el cilindro destino se espera a que el sector buscado pase por debajo de la cabeza lectora (gira hasta el comienzo del sector) ($t_{latencia}$, cuya media es el tiempo de media vuelta). Una vez encontrado, se lee/escribe en el sector y el tiempo de lectura/escritura ($t_{l/e}$) será el de giro del sector ($t_{l/e} = t_{sext}$).

Fórmulas de cálculo de tiempos:

$$\text{TIEMPO ACCESO} = t.\text{búsqueda} + t.\text{latencia}/2$$

$$\text{TIEMPO LATENCIA} = t.\text{vuelta}/2 = (\text{sect destino} - \text{sect origen}) * t.\text{sect} = \text{Capacidad Bruta pista}/v.\text{transfer}$$

$$\text{TIEMPO LATENCIA MAX} = \frac{60 \text{ s/min}}{\text{rpm}}$$

$$\text{TIEMPO BUSQUEDA} = t.\text{posicionamiento} + t.\text{estabilización}$$

$$\text{TIEMPO POSICIONAMIENTO} = \text{mov.cab} * \text{pistas}/2$$

$$\text{TIEMPO SECTOR} = \frac{t.\text{latencia}}{\text{Nº sectores/pista}}$$

$$\text{TIEMPO TRANS} = t.\text{sector} = \frac{\text{tamaño fichero} * \text{bits}}{\text{velocidad que te dan b/s}}$$

$$\begin{aligned} \text{TIEMPO OPERACION} = \\ \text{HDD} \rightarrow t.\text{acceso} + t.\text{sector}. \\ \text{Cinta} \rightarrow t.\text{arranque} + t.\text{transferencia} + t.\text{parada} \end{aligned}$$

- **Tiempo de búsqueda:** El motor no se para, lo que indica que mientras se mueve la cabeza, se habrá avanzado un determinado número de sectores.
- **Tiempo de latencia:** Depende de la velocidad de rotación y del nº del sector. De media, el tiempo será de media vuelta.
- **Tiempo de lectura / escritura:** Igual a tiempos de cabeza en sector, si hay ZBR.

Fórmulas de cálculo de coordenadas CHS (cilindro, superficie y sector):

$$\text{CILINDRO (C)} = \frac{\text{sector abs}}{\text{SC}} (\text{Nº de cilindro})$$

$$r = \text{sect abs mod SC} \rightarrow SA - (\text{SC} * \text{res})$$

$$\text{SUPERFICIE o HEAD (H)} = \frac{r}{\text{SP}} (\text{Nº de sup o cabeza})$$

$$\text{SECTOR (S)} = r \text{ mod SP} (\text{Nº del sector})$$

$$\text{SECTORES/CILINDRO SC} = \text{Nº sup} * \text{SP}$$

$$\text{SP} = \frac{\text{capacidad neta}}{\text{nº sup} * \text{nº cilindro} * \text{nº sect}} \left(\frac{\text{sectores}}{\text{pista}} \right)$$

$$\text{Sector abs (coordenadas)} = C * \text{SC} + H * \text{SP} + S$$

Ejemplo: En t = 0s la cabeza del disco se encuentra al comienzo del sector s1 en la pista p50: ¿en qué instante concluirá la transferencia del sector s20 de la pista p70?

Datos: Velocidad del giro: 3000 rpm = 20 ms/rev

$$\text{Nº de pistas} = 500 \text{ pistas}$$

$$\text{Nº de sectores/pista} = 25 \text{ sect/pista} = t_{\text{sect}} = 0.8 \text{ ms/sect}$$

$$T \text{ de pista a pista consecutiva} = 0.2 \text{ ms/pista}$$

T estabilización al llegar pista destino = 4 ms

Solución:

- **Tiempo medio de acceso**

$$t_{busq} = t_{pos} + t_{est} = 0.2 \text{ ms/pista} * 500/2 \text{ pistas} + 4 \text{ ms} = 54 \text{ ms}$$

$$t_{acc} = t_{busq} + t_{lat} = 54 \text{ ms} + 20 \text{ ms/rev / 2} = 64 \text{ ms}$$

$$t_{l/e} = t_{sect} = 0.8 \text{ ms (por 1 sector)}$$

$$t_{op} = t_{acc} + t_{l/e} = 64 \text{ ms} + 0.8 \text{ ms} = 64.8 \text{ ms}$$

- **Tiempo de acceso en enunciado**

$$t_{busq} = t_{pos} + t_{est} = 0.2 \text{ ms/pista} * 20 \text{ pistas} + 4 \text{ ms} = 8 \text{ ms}$$

Sectores girados desde s₁ en 8 ms: 8 ms / (0.8 ms/sect) = 10 sect

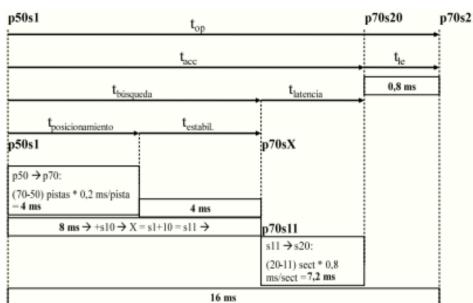
Tras t_{busq} la cabeza se encuentra al comienzo del sector s₁₁.

$$t_{acc} = t_{busq} + t_{lat} = 8 \text{ ms} + 9 \text{ sect} * 0.8 \text{ ms/sect} = 15.2 \text{ ms}$$

$$t_{l/e} = t_{sect} = 0.8 \text{ ms (por 1 sector)}$$

$$t_{op} = t_{acc} + t_{l/e} = 15.2 \text{ ms} + 0.8 \text{ ms} = 16 \text{ ms}$$

Diagrama de solución:



Ejemplo de distribución de sectores lógicos en el disco:

Datos: 2000 cilindros o pistas por disco (C)

8 superficies (H) o caras

200 sectores

Cálculos:

$$200 * 8 = 1600 \text{ sectores por cilindro}$$

$$1600 * 2000 = 3200000 \text{ sectores totales en el disco}$$

Problema: Coordenadas geométricas CHS del sector 1234567:

$$\textbf{Cilindro: } 1234567 / 1600 = 771 \text{ (Resto: 967)}$$

$$\textbf{Superficie: } 967 / 200 = 4 \text{ (Resto: 167)}$$

$$\textbf{Sector: } 167$$

$$\mathbf{C = 771; H = 4; S = 167}$$

Fórmulas:

$$\text{CAPACIDAD NETA POR PISTA (B/pista)} = \text{Nº de sectores por pista} * \text{Nº B por sector}$$

$$\text{VELOCIDAD DE ROTACION (pistas/n)} = \frac{\text{Velocidad de transmisión (B/s)}}{\text{Capacidad bruta (B/pista)}}$$

$$\text{VELOCIDAD DE TRANSFERENCIA} = \frac{\text{Capacidad Bruta por sector (B)}}{\text{tiempo sector}}$$

$$\text{RPM (Revoluciones por Minuto)} = \text{Velocidad rotación (pistas/s)} * 60 \text{ s}$$

$$\text{DENSIDAD DE GRABACIÓN ANGULAR (bits/grado)} = \frac{\text{Capacidad Bruta} * 8 \text{ (b/B)}}{360^\circ}$$

$$\text{DENSIDAD DE GRABACION LINEAL (bits/grado)} = \frac{\text{Capacidad Bruta} * 8 \text{ (b/B)}}{2 * \pi * \text{radio}}$$

$$\text{Nº DE CILINDROS} = \frac{\text{Capacidad Bruta Total (B)}}{\frac{\text{Capacidad bruta pista} * \text{Nº pistas}}{\text{Nº cilindros}}}$$

$$\text{SECTORES POR PISTA} = \frac{\text{Capacidad neta}}{\frac{\text{Nº cilindros} * \text{Nº superficies} * \text{Capacidad neta sector}}{}}$$

$$\text{SECTORES POR CILINDRO} = \frac{\text{Nº de pistas}}{\text{cilindro}} * \frac{\text{Nº de sectores}}{\text{pista}}$$

$$\text{CAPACIDAD BRUTA TOTAL (B)} = \text{Capacidad bruta pista (B)} * \frac{\text{Nº pistas}}{\text{superficie}} * \text{superficies}$$

$$\text{CAPACIDAD BRUTA PISTA (B)} = \text{Vel. transferencia (B/s)} * \text{tiempo latencia}$$

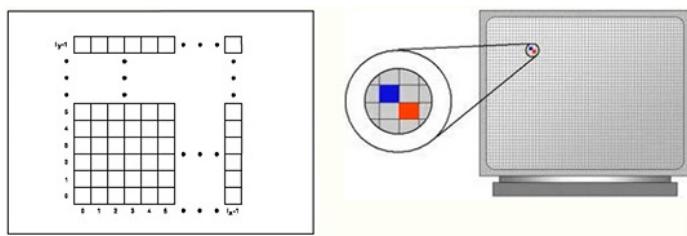
5.4. DISCO DE ESTADO SÓLIDO (SSD)

Unidad de almacenamiento construida con circuitos integrados y elementos de memoria. No es un disco, pero cumple su función de almacenar memoria. Los datos están distribuidos por la memoria, no tienen coordenadas CHS.

Estos discos cuentan con **ventajas** como que **no contienen elementos electro-mecánicos**; tienen **menor tiempo de acceso** y de **transferencia** (las conmutaciones electrónicas son más rápidas que las conmutaciones mecánicas, por ej: SSD: 2GB/s y HDD: 300-400 MB/s); son **más robustos** físicamente, funcionan **silenciosamente** (pues no realizan conmutaciones mecánicas); tienen un **menor consumo** eléctrico (aprox 1/3 que los HDD); tienen un **menor peso** y son **más fiables** (menor tendencia a contener errores).

Sin embargo, también tienen **desventajas** como: su precio: cuestan más que los HDD; **velocidad de acceso en escritura es mayor que la de en lectura**; y tienen un **tiempo de vida más limitado**.

5.5. MONITOR LCD



- **Resolución:** Número de pixeles en línea * Nº de líneas
- **Profundidad de color:** Nº de bits o bytes usados para representar un píxel. Determina el número de colores distintos. Por ejemplo: 1 bit (blanco y negro), 8 bits (2^8 colores), 24 bits (2^{24} colores).
- **Memoria de pantalla (de vídeo, de cuadro, ...):** Matriz almacenada en memoria que representa la imagen que proyecta la pantalla. La resolución y la profundidad de color determinan el tamaño de la matriz.
- **Frecuencia de la pantalla:** Frecuencia con que se proyecta la memoria en pantalla. A mayor frecuencia de pantalla, mayor velocidad de transferencia habrá de la matriz en la memoria de la pantalla a la pantalla del monitor. Por ejemplo: 60 FPS.

*Tamaño de la memoria = Resolución * Profundidad de color (en bytes)*

*Velocidad de transferencia = Tamaño de memoria * Frecuencia de pantalla*

- **Longitud de palabra:** Nº de bits o bytes que se leen en un ciclo de memoria.
- **Tiempo de acceso:** Tiempo que se tarda en leer una palabra.
- **Ancho de banda (velocidad de transferencia):** Velocidad máxima de lectura de la memoria

Fórmulas:

$$TAMAÑO (B) = \text{resolución} * \text{profundidad color (B)}$$

$$VELOCIDAD LECTURA MEMORIA (B/s) = Tamaño (B) * Frecuencia pantalla (Hz)$$

$$ANCHO DE BANDA \left(\frac{B}{s}\right) = \frac{\text{Longitud de palabra (B)}}{\text{Tiempo de acceso (ns)}} * \frac{1}{10^{-9}} = \text{Resolución} * \text{profundidad color (B/pixel)} * \text{frecuencia refresco}$$

$$Nº \text{ DE BLOQUES} = \frac{\text{resolución} * \frac{\text{profundidad color (b)}}{8 (b/B)}}{\text{información neta (B)}}$$

$$TIEMPO DE TRANSFERENCIA (s) = \frac{Nº \text{ bloques} * \text{Inf bruta (B)} * 8 (b/B)}{\text{Velocidad de transferencia (b/s)}}$$

$$FRECUENCIA (Hz) = \frac{\text{Ancho de banda}}{\text{tamaño}}$$

5.6. DISPOSITIVOS DE COMUNICACIÓN