Unidad 1

Nicolás Gómez Morgado

Arquitectura de Computadores

29 de julio de 2024

Índice

1. Conceptos a tener en cuenta

- **Informática:** Ciencia de la computación. Conjunto de conocimientos científicos y técnicos que hacen posible el tratamiento automático de la información por medio de computadoras.
- **Computadora:** Maquina electronica, analógica o digital, que recibe, procesa y almacena información.
- **Computación:** Se refiere a un proceso matemático que genera una forma de información.
- **Voltaje:** Potencial de fuerza que permite transferencia de corrientes eléctricas.
- **Programación:** Indicar a la computadora que hacer.
- **Programa:** Conjunto de instrucciones que le indican a la computadora que hacer con el fin de resolver un problema.
- **Arquitectura de computadores:** Todo dispositivo que nos permita manejar información, es decir, que pueda realizar operaciones matemáticas.
- **Mapas de Karnaugh:** Son una herramienta que nos permite simplificar funciones booleanas. Se utilizan para simplificar funciones booleanas de hasta 4 variables. Si se quiere comprobar que 2 circuitos son iguales sin las expresiones booleanas, se debe generar la tabla de verdad de ambos circuitos y compararlos.
- **Bit:** Unidad básica de información, puede ser 0 o 1.
- **Runtime:** Tiempo de atención a eventos/instrucciones en nanosegundos.
- **Buses:** Los que se encargan de la comunicación haciendo todo el recorrido.

2. Principios Técnicos

Von Neumann **no definió** la arquitectura de computadores, sino que agregó un concepto básico que transformó la computación: la memoria y la CPU se encuentran en el mismo lugar, lo que permite que la CPU pueda acceder a la memoria de manera directa.

Turing demostró que se pueden crear máquinas decodificadoras. Esto dio paso a la computación de hoy en día y a la transferencia de datos a través de la red.

Los **bits** son los que determinan la velocidad/calidad de un computador, mientras que los **bytes** son los que determinan la capacidad de almacenamiento de un computador.

Ley de Moore

En 1965, Gordon Moore gráfico los datos sobre el crecimiento en el rendimiento de la CPU de las computadoras, con lo cual predijo que cada nuevo chip doblaba la capacidad de su predecesor de hace 2 años (aumento exponencial).

El procesador trabaja todo en lo que se llama registro. Este es como un vector de bits y su tamaño (32 o 64) determinará cuántas operaciones complejas puede realizar.

2.1. Bytes

¿Qué es un byte y por qué tiene 8 bits?
Un byte es una **unidad básica de información** que consiste en 8 bits. La razón por la cual un byte tiene 8 bits se debe a que **256 es el número de combinaciones estandarizadas posibles** con 8 bits, lo que permite representar el **total de caracteres** en un conjunto de caracteres. El rango de caracteres representables va desde 0 hasta 255, lo que abarca la mayoría de los caracteres utilizados en diferentes sistemas de escritura. En caso de que se necesite representar números positivos y negativos la mitad de las combinaciones se utilizan para los números negativos y la otra mitad para los números positivos (127 y 127 implicando un 0 y un -0).

¿El programa que voy a utilizar cabe completamente en la RAM?
Típicamente no. El proceso que voy a describir se conoce como segmentación de código. En términos estrictos, se le llama paginación. El uso de la paginación se suele dar cuando tratamos con programas de gran tamaño. La paginación implica que el sistema acceda al **disco duro** para buscar las instrucciones que se ejecutarán en la **RAM**. Esto se realiza en trozos, siendo la cantidad de trozos igual a la cantidad en que se dividió el programa. La máquina no toma decisiones al respecto, simplemente asigna espacios iguales en la RAM para los programas en ejecución, independientemente de si uno ocupa más o menos memoria. Es decir, no se borra, simplemente se **sobrescribe**.

¿Cuál es más eficiente, tener la máquina ejecutando un solo programa o varios programas?
En términos de eficiencia, es preferible tener menos programas en ejecución, ya que **cuantos más programas se ejecuten simultáneamente, más se divide la RAM**, asignando espacios iguales de memoria sin importar si un programa está siendo utilizado

activamente o no. Se recomienda utilizar solo lo necesario para evitar esta fragmentación.

¿Cuántos bytes ocupa un número entero en C?
En C, un número entero (int) generalmente ocupa 4 bytes (anteriormente ocupaba 2 bytes). Esto permite un total de 2^{32} combinaciones de números, con la mitad menos 1 para los valores positivos y el mismo valor para los valores negativos, con un cero tanto positivo como negativo.

Ejercicio

Ejercicio 1:
¿Cuántos bit hay en 4 Tb?

$$1 \text{ byte} = 8 \text{ bits}$$
$$1 \text{ kilobyte} = 1024 \text{ bytes} = 2^{10} \text{ bytes}$$
$$1 \text{ Megabyte} = 1024 \text{ Kb} = 2^{20} \text{ bytes}$$
$$1 \text{ Gigabyte} = 1024 \text{ Mb} = 2^{30} \text{ bytes}$$
$$1 \text{ Terabyte} = 1024 \text{ Gb} = 2^{40} \text{ bytes}$$
$$4 \text{ Tb} = 4 \times 2^{40} \text{ bytes} = 2^2 \times 2^{40} \text{ bytes} = 2^{42} \text{ bytes.}$$
$$4 \text{ Tb} = 2^{42} \text{ bytes} = 2^{42} \times 8 \text{ bits} = 2^{42} \times 2^3 \text{ bits} = 2^{45} \textbf{ bits.}$$

¿Qué consume más en un disco duro?
Lo que más consumiría serían los **byte de dirección** puesto que al pasar los 256 datos posibles se usa 1 byte más para direcciones aumentando las posibilidades.

Ejercicio

Ejercicio 2:
¿Cuantos bytes de dirección hay en 8 Tb de datos?

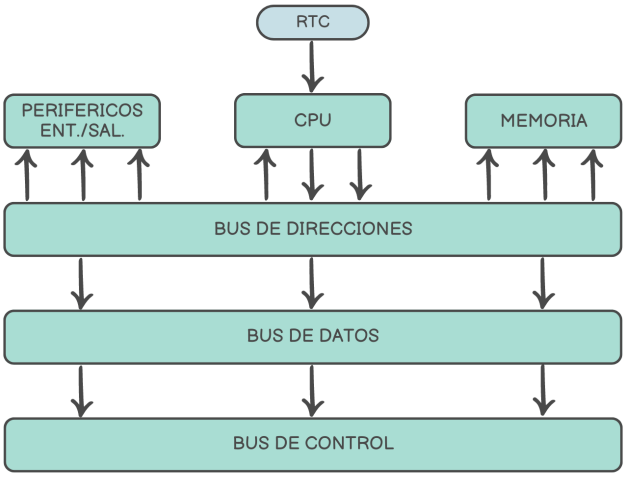
Solución:
$$8 \text{ Tb} = 8 \times 2^{40} = 2^3 \times 2^{40} = 2^{43} \text{ byte.}$$
Para:
$$2^8 \rightarrow 1 \textit{ byte} \times \text{ dato}$$
$$2^{16} \rightarrow 2 \textit{ byte} \times \text{ dato}$$
$$\vdots$$
$$2^{40} \rightarrow 5 \textit{ byte} \times \text{ dato}$$
$$2^{48} \rightarrow 6 \textit{ byte} \times \text{ dato}$$

Para este ejemplo 1 byte de datos esta acompañado de 6 bytes de dirección.

Si un disco duro o respaldo indica que tiene 128 GB de almacenamiento, ¿se refiere esa cifra a bytes de datos y direcciones?
Normalmente, cuando se menciona una capacidad de almacenamiento, no se incluyen explícitamente las capacidades de direccionamiento. Por lo tanto, se puede concluir que la cifra de 128 GB se refiere únicamente a la capacidad de almacenamiento de datos, sin tener en cuenta la capacidad de direccionamiento.

3. Arquitectura de una computadora

Arquitectura de una computadora según Von Neumann:



3.1. Conexiones y buses

- **Bus de direcciones:** Permite que el CPU y los periféricos accedan a direcciones de memoria.
- **Bus de datos:** Canal de comunicación entre el CPU, periféricos y la memoria.
- **Bus de control:** Ordena las operaciones de lectura y escritura de la memoria.

Rutas de Conexión: Caminos entre los circuitos integrados que permiten la comunicación entre los componentes de la computadora.

3.2. Dispositivos y controladores

- **Adaptador Gráfico:** Permite la conexión de la computadora con un monitor. Presenta textos y gráficos en la pantalla.
- **Controlador de disco:** Permite la conexión de la computadora con un disco duro. Administra el disco duro.
- **Interconexión de Módulos Digitales:** Permite la conexión de la computadora con otros dispositivos. Conexiones entre componentes digitales.

4. Unidad central del sistema

4.1. Placa principal

Tiene como principal función conectar todos los componentes de la computadora.

Componentes clave (*Tarjeta Madre Tipo Pentium ATX*)**:**

- **ROM-BIOS:** Programas de arranque y chequeo de dispositivos.
- **Ranuras ISA, PCI:** Para tarjetas controladoras.
- **Puertos Seriales (COM 1, COM 2):** Para periféricos seriales.
- **Puerto Paralelo (LPT1):** Para impresoras y otros periféricos.
- **Puertos USB:** Para dispositivos modernos.
- **Conectores de Energía (PWR AT, PWR ATX)**
- **Conectores IDE 1, IDE 2:** Para discos duros y CD-ROMs.
- **Conector FDC:** Para unidad de disquete.
- **SLOT o Socket del Microprocesador:** Instalación del CPU.
- **Ranuras para Módulos de Memoria RAM**
- **Conectores para HD LED, Speaker y Reset Button**
  - HD LED: Indicador color rojo de actividad del disco duro ubicado en la parte delantera de la case.
  - Speaker (SPK): Conexión del cable del speaker el cual emite los bips.
  - Reset button (RST): Cable del botón del reset que se encuentra en la parte delantera de la case y sirve para reiniciar la máquina

4.2. Procesador

El procesador tiene 4 módulos funcionales:

- **CPU:** Ejecuta ordenes. Con FPU poseen propia unidad de control.
- **FPU:** Unidad de puntos flotantes (decimales).
- **MMU:** Unidad de administración de memoria. Se encarga del proceso de administración y traslado desde el disco duro a la RAM. También se encarga del proceso de paginación.
- **Cache interna:** Cache propia del procesador. Permite acceder a los datos mas eficientemente que la RAM.

Ademas de estos módulos, el procesador se compone de otros elementos, los cuales son:

- **Unidad de control (CU)**
- **Unidad aritmético-lógica (ALU)**
- **Registros**

4.2.1. Unidad de control

La unidad de control es la encargada de coordinar las operaciones del sistema informático. Se encarga de:

- Acceso
- Lectura
- Escritura de memoria
- Interpretación de instrucciones
- Ejecución de tareas

4.2.2. Unidad aritmético-lógica

Encargada de realizar cálculos matemáticos y lógicas.

4.2.3. Registros

La unidad de control es la base para la máquina. Los registros son un **medio de ayuda a la unidad de control y la aritmética**, permiten **almacenar información temporalmente** para la manipulación de los datos por parte de la CPU. Existe un registro en particular que se llama **acumulador**, que almacena todos los resultados de operaciones matemáticas. Los registros son **lo más importante** que tiene la máquina después del procesador.

Observaciones

- La computadora promedio tiene 64 registros.
- *"Todo esta y se manipula en registros" .- Juan Carlos Parra Márquez.*

4.3. Disco Duro

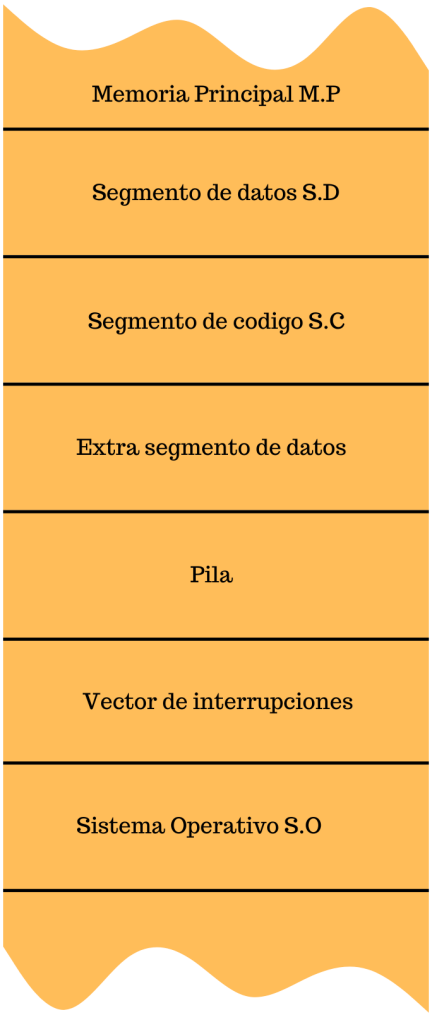
El disco duro es el encargado de almacenar la información de la computadora. Se compone de un conjunto de discos magnéticos que giran a gran velocidad.

- **Tipos de Interfaces:** IDE, SCSI.
- **Funciones:** Almacenamiento no volátil, formateo y particionado.

4.4. Memoria principal

Es la zona de trabajo donde la computadora va a almacenar temporalmente las órdenes a ejecutar y los datos que deberán manipular esas órdenes. Ademas de esta memoria existen otros tipos de memorias:

4.4.1. RAM (Volatil)



La imagen anterior representa a un modulo RAM, la cual en termino estricto es homogénea, pero viene subdividido en sectores:

- **Segmento de datos (S.D):** Se encarga del almacenamiento de datos, su tamaño es proporcional al tamaño de la memoria RAM.
- **Segmento de código (S.C):** Encargado de almacenar el programa (instrucciones), que vienen del disco de almacenamiento (disco duro).

- **Extra segmento de datos:** Un espacio extra en caso de que el tamaño en la S.D no sea el suficiente.
- **Pila:** Determina el orden de ejecución de transferencias de control. Almacena direcciones de memoria de donde ir al terminar cada instrucción (transferencia de control en ciclos anidados) y datos de los parámetros de la instrucción (parámetros de ejecución).
- **Vector de interrupciones:** Vector dado por circuitos dentro de la RAM. Este se encarga de almacenar todos los microprogramas que manejan los dispositivos internos de la computadora. En el vector de interrupciones se manejan todos los periféricos. Trabaja en conjunto con el **Runtime** para saber que se esta usando.
- **Sistema operativo S.O:** Es el encargado de manejar los recursos de la computadora.

Las variables pueden ser dinámicas o estáticas. Se tiende a usar dinámicas en la RAM para ahorrar espacio economizar recursos. Todos los datos se trabajan en SD y ExtraSD se pueden pedir estáticos o dinámicas(punteros). Los punteros pueden apuntar a una celda establecida o darle una celda propia con maloc.

4.4.2. ROM (No volatil)

La memoria ROM es una memoria de solo lectura, es decir, no se puede escribir en ella. Se utiliza para almacenar el programa de arranque de la computadora,

4.4.3. Memoria caché

La memoria caché es una memoria de acceso rápido que se encuentra entre la CPU y la memoria principal. Su función es almacenar los datos que se utilizan con mayor frecuencia,

- **Ubicación:** Parte de la tarjeta madre y del procesador.
- **Uso:** Acceso rápido a la información por el procesador.

4.5. Tarjetas de expansion interna

Las tarjetas de expansión están diseñadas para actividades específicas como controlar la salida de video, gráficos y comunicaciones. Las principales tarjetas de expansión son:

- **Tarjetas de video:** Controlan la salida de video.
- **Tarjetas de entrada y salida de datos:** Controlan la entrada y salida de datos.
- **Tarjetas de comunicaciones:** Controlan la comunicación entre la computadora y otros dispositivos.

Tarjetas Controladoras de Comunicaciones
Estas tarjetas permiten la conexión de una computadora con otras formando una red informática.
Tipos de redes y tarjetas:

- Red de area local: Tarjeta de red **LAN** (NIC: Network Interface Card).
- Red de area extensa: Tarjeta de red **WAN**. Modem.

4.6. Fuentes de alimentación

Las fuentes de alimentación proporcionan la energía eléctrica que necesita la computadora para funcionar. Esa energía se estabiliza para impedir que la computadora se vea afectada por oscilaciones bruscas en el suministro de las compañías eléctricas.

4.6.1. Valores de Voltaje

Típicamente se representan los 2 valores discretos de voltaje utilizados con las letras L y H, donde L es **LOW** y H es **HIGH**.

	Alto/HIGH	**Bajo/LOW**
Rangos de valores voltaje salida	[4 , 5.5]	[-0.5 , 1]

Entra en el certamen según el profesor.

El procesador ejecuta el programa almacenado en el segmento de código, el cual se encuentra en la RAM utilizando los registros.

4.7. Reloj

El reloj de una computadora desempeña **dos funciones principales**. En primer lugar, **sincroniza diversas operaciones** que se realizan en diferentes componentes del sistema. En segundo lugar, sirve para **mostrar la hora actual**. La función principal es la primera, ya que ayuda a obtener el tiempo de ejecución (runtime) de las operaciones.

Reloj del sistema: Un pulso electrónico se utiliza para sincronizar el procesamiento en orden de nanosegundos, lo que permite medir el tiempo de ejecución (runtime) en MHz. Un megahercio (1 MHz) equivale a un millón de ciclos por segundo.

Observaciones

- La frecuencia se refiere a cuantos instantes de atención a eventos/instrucciones se pueden realizar en un segundo.
- ROM es lo que sirve para arrancar y contiene el set-up.
- El lenguaje ensamblador le saca la máxima velocidad a la maquina.
- A lo que se refieren cuando dicen la maquina es de 2.4GHz es a la velocidad de su reloj, la cual es de 2.4 mil millones de ciclos por segundo.

5. Lenguaje Maquina

Los códigos hexadecimales pueden representar instrucciones, registros de la CPU, direcciones de memoria o datos.

Ejemplo

Código	Operación	Descripción
A0	2F	Acceder a la celda de memoria 2F
3E	01	Copiar el registro 1 de la ALU
A0	30	Acceder a la celda de memoria 30
3E	02	Copiar en el registro 2 de la ALU
1D	-	Sumar
B3	31	Guardar el resultado de la celda de memoria 31

5.1. Lenguaje Ensamblador

El lenguaje ensamblador es un lenguaje de programación de bajo nivel que se utiliza para escribir programas que se ejecutan directamente en la CPU. Se utilizan codificación nemotécnica para facilitar la programación y tener una mayor legibilidad.

READ	2F
REG	01
READ	30
REG	02
ADD	-
WRITE	31

Código fuente (lenguaje ensamblador)
↓
Programa ensamblador
↓
Código máquina (lenguaje máquina)

Este tipo de lenguaje se considera un **lenguaje de nivel medio**.

5.2. Lenguajes de alto nivel

Algunos lenguajes de alto nivel son:

- **Ada**
- **ALGOL**
- **BASIC**
- **C**
- **C++**
- **C#**
- **COBOL**
- **Fortran**
- **Java**
- **Pascal**
- **Python**
- **Ruby**
- **Modula**
- **Simula**

6. Algebra de Boole

George Boole introdujo el álgebra de Boole en 1854. Este sistema matemático se basa en la lógica y la teoría de conjuntos, y utiliza variables binarias (0 y 1) junto con compuertas lógicas para representar y manipular proposiciones lógicas. El álgebra de Boole es fundamental en la teoría de circuitos digitales y la informática.

6.1. Compuertas lógicas

Las compuertas lógicas son dispositivos electrónicos que realizan operaciones lógicas sobre una o más entradas binarias y generan una salida binaria. Las compuertas lógicas son:

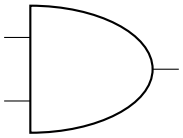
- **AND**
- **OR**
- **NOT**
- **NAND**
- **NOR**
- **XOR**
- **XNOR**

6.2. Compuertas lógicas básicas

6.2.1. Compuerta AND

La compuerta AND produce una salida de 1 solo si todas sus entradas son 1. La tabla de verdad de una compuerta AND es:

A	B	A AND B = A · B = Z
0	0	0
0	1	0
1	0	0
1	1	1



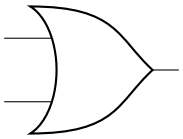
Observación

Recordar que en un circuito representativo de AND en serie el voltaje total es la suma de los voltajes de las compuertas.-

6.2.2. Compuerta OR

La compuerta OR produce una salida de 1 si al menos una de sus entradas es 1. La tabla de verdad de una compuerta OR es:

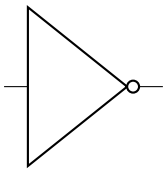
A	B	A OR B = A + B = Z
0	0	0
0	1	1
1	0	1
1	1	1



6.2.3. Compuerta NOT

La compuerta NOT produce una salida de 1 si su entrada es 0 y viceversa. La tabla de verdad de una compuerta NOT es:

A	NOT A $= \overline{A} =$ Z
0	1
1	0



6.3. Leyes del Algebra de Boole

Identidades Básicas

1. $X + 0 = X$
2. $X + 1 = 1$
3. $X + X = X$
4. $X + \overline{X} = 1$
5. $\overline{\overline{X}} = X$
6. $X \cdot 0 = 0$
7. $X \cdot 1 = X$
8. $X \cdot X = X$
9. $X \cdot \overline{X} = 0$

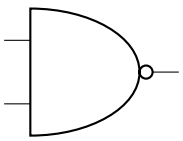
1. $X + Y = Y + X$	5. $XY = YX$	Conmutativa
2. $X + (Y + Z) = (X + Y) + Z$	6. $X(YZ) = (XY)Z$	Asociativa
3. $X(Y + Z) = XY + XZ$	7. $X + YZ = (X + Y)(X + Z)$	Distributiva
4. $\overline{X + Y} = \overline{X} \cdot \overline{Y}$	8. $\overline{X \cdot Z} = \overline{X} + \overline{Z}$	De Morgan

6.4. Compuertas lógicas avanzadas

6.4.1. Compuerta NAND

La compuerta NAND produce una salida de 0 si todas sus entradas son 1. La tabla de verdad de una compuerta NAND es:

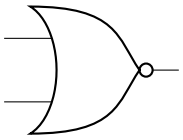
A	B	A NAND B $= \overline{A \cdot B} =$ Z
0	0	1
0	1	1
1	0	1
1	1	0



6.4.2. Compuerta NOR

La compuerta NOR produce una salida de 1 si todas sus entradas son 0. La tabla de verdad de una compuerta NOR es:

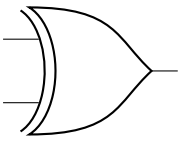
A	B	A NOR B = $\overline{A+B}$ = Z
0	0	1
0	1	0
1	0	0
1	1	0



6.4.3. Compuerta XOR

La compuerta XOR produce una salida de 1 si sus entradas son diferentes. La tabla de verdad de una compuerta XOR es:

A	B	A XOR B = Z
0	0	0
0	1	1
1	0	1
1	1	0



7. Simplificación de funciones lógicas (Mapas de Karnaugh)

Los mapas de Karnaugh son una herramienta gráfica que se utiliza para simplificar funciones lógicas. Los mapas de Karnaugh son especialmente útiles para simplificar funciones lógicas con hasta 4 variables.

Pasos para simplificar funciones lógicas con mapas de Karnaugh:

1. **Identificar las variables:** Identificar las variables de la función lógica.
2. **Crear el mapa de Karnaugh:** Crear un mapa de Karnaugh con las variables identificadas.
3. **Completar el mapa de Karnaugh:** Completar el mapa de Karnaugh con los valores de la función lógica.
4. **Simplificar la función lógica:** Utilizar el mapa de Karnaugh para simplificar la función lógica.

Ejemplo

Para la tabla de verdad de la función lógica $F(A, B, C)$:

A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Por lo que a simple vista podemos concluir la siguiente función lógica:

- $F = \overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\overline{C} + A\overline{B}C$

Por lo tanto la construcción de un mapa de Karnaugh para la función lógica $F(A, B, C)$ es:

$A\backslash BC$	00	01	11	10
0	1	1	1	0
1	0	1	0	0

Lo cual al reducir los grupos de 1's obtenemos la función lógica simplificada tenemos que para:

- El grupo azul-amarillo representa la combinación $\overline{A}\overline{B}$
- El grupo rojo-amarillo representa la combinación $\overline{B}C$
- El grupo verde-amarillo representa la combinación $\overline{A}C$

Por lo que la función lógica simplificada es:

$$F = \overline{A}\overline{B} + \overline{B}C + \overline{A}C$$

8. Sistemas numéricos digitales

8.1. Abstracción digital

- La mayoría de las variables son **continuas**.
  - Voltaje de un cable.
  - Frecuencia de un oscilador.
  - Posición de un objeto.
- La abstracción digital considera un **conjunto discreto** de valores (0,1,2,...).

8.2. Disciplina digital

- Dos valores discretos:
  - 1's y 0's.
  - 1 → Verdadero/True, Alto/High.
  - 0 → Falso/False, Bajo/Low.
- **1 y 0**: Niveles de voltaje, engranajes giratorios, niveles de fluidos, etc.
- Los circuitos de digitales usan niveles de **voltaje** para representar 1's y 0's.
- **Bit:** Dígito binario, unidad más pequeña de información.

8.3. Sistemas de numeración

El sistema numérico decimal es el más común, pero existen otros sistemas de numeración que se utilizan en la informática y la electrónica digital. Los sistemas de numeración más comunes son:

- **Binario:** Base 2.
- **Octal:** Base 8.
- **Hexadecimal:** Base 16.
- **Decimal:** Base 10.

8.3.1. Rangos y valores

A través del sistema binario y decimal existen diferentes rangos de valores que se pueden representar, los cuales podemos calcular de la siguiente manera:

- Números decimales de n dígitos:
  - Valores distintos con n dígitos: 10^n
  - Rango de valores con n dígitos: 0 a $10^n - 1$
- Números binarios de n bits:
  - Valores distintos con n bits: 2^n
  - Rango de valores con n bits: 0 a $2^n - 1$

8.3.2. Conversión de sistemas de numeración

Conversión binaria a decimal:

- **Ejemplo:** Convertir el número binario 1011 a decimal.
  - $1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 8 + 0 + 2 + 1 = 11$

Conversión decimal a binario:

- **Ejemplo:** Convertir el número decimal 13 a binario.
  - $13 \div 2 = 6$ con residuo 1
  - $6 \div 2 = 3$ con residuo 0
  - $3 \div 2 = 1$ con residuo 1
  - $1 \div 2 = 0$ con residuo 1

  Por lo tanto, el número decimal 13 en binario es 1101.

Conversión binaria a hexadecimal:

- **Ejemplo:** Convertir el número binario 101101 a hexadecimal.
  - $101101 = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 32 + 0 + 8 + 4 + 0 + 1 = 45$
  - $45 = 2 \cdot 16^1 + 13 \cdot 16^0 = 2D$

Conversión hexadecimal a binario:

- **Ejemplo:** Convertir el número hexadecimal 2D a binario.
  - $2D = 2 \cdot 16^1 + 13 \cdot 16^0 = 45$
  - $45 = 32 + 8 + 4 + 1 = 101101$

Conversión decimal a hexadecimal:

- **Ejemplo:** Convertir el número decimal 45 a hexadecimal.
  - $45 = 2 \cdot 16^1 + 13 \cdot 16^0 = 2D$

Conversión hexadecimal a decimal:

- **Ejemplo:** Convertir el número hexadecimal 2D a decimal.
  - $2D = 2 \cdot 16^1 + 13 \cdot 16^0 = 45$

Conversión octal a decimal:

- **Ejemplo:** Convertir el número octal 45 a decimal.
  - $45 = 4 \cdot 8^1 + 5 \cdot 8^0 = 37$

Conversión decimal a octal:

- **Ejemplo:** Convertir el número decimal 182 a octal.
  - $182 \div 8 = 22$ con residuo 6
  - $22 \div 8 = 2$ con residuo 6
  - $2 \div 8 = 0$ con residuo 2

  Por lo tanto, el número decimal 182 en octal es 266.

8.4. Bits, Bytes, Nibbles,...

Los **bits** son la unidad mas pequeña de información, los cuales se agrupan, dando a conocer un rango de importancia dependiendo de la ubicación de los bits. Por ejemplo:

$$\textit{Most significant bit}[\text{MSB}]\rightarrow 1\,10101\,0 \leftarrow \textit{Least significant bit}[\text{LSB}]$$

A la agrupación de estos bits se les conoce como **Bytes**, los cuales son de 8 bits. A su vez, los bytes se pueden agrupar en **Nibbles**, los cuales son de 4 bits.Por lo tanto, un byte es igual a 2 nibbles.

Byte	1	0	1	0	1	0	1	0
Nibble 1	1	0	1	0	-	-	-	-
Nibble 2	-	-	-	-	1	0	1	0

Potencias de 2 importantes

- 2^{10} = 1 kilobyte (KB) = 1024 bytes.
- 2^{20} = 1 megabyte (MB) $\approx$ 1 millón de bytes.
- 2^{30} = 1 gigabyte (GB) $\approx$ 1 billón de bytes.
- 2^{40} = 1 terabyte (TB) $\approx$ 1 trillón de bytes.

8.5. Decimal (con decimales) a binario

- **Ejemplo:** Convertir el número decimal 0.6875 a binario.
  - 0,6875 × 2 = 1,375 con parte entera 1 y parte decimal 0,375
  - 0,375 × 2 = 0,75 con parte entera 0 y parte decimal 0,75
  - 0,75 × 2 = 1,5 con parte entera 1 y parte decimal 0,5
  - 0,5 × 2 = 1 con parte entera 1 y parte decimal 0

  Por lo tanto $(0.6875)_{10} = (0.1011)_2$
- **Ejemplo:** Convertir el número decimal 0.1 a binario.
  - 0,1 × 2 = 0,2 con parte entera 0 y parte decimal 0,2
  - 0,2 × 2 = 0,4 con parte entera 0 y parte decimal 0,4
  - 0,4 × 2 = 0,8 con parte entera 0 y parte decimal 0,8
  - 0,8 × 2 = 1,6 con parte entera 1 y parte decimal 0,6
  - 0,6 × 2 = 1,2 con parte entera 1 y parte decimal 0,2
  - 0,2 × 2 = 0,4 con parte entera 0 y parte decimal 0,4
  - 0,4 × 2 = 0,8 con parte entera 0 y parte decimal 0,8
  - 0,8 × 2 = 1,6 con parte entera 1 y parte decimal 0,6
  - 0,6 × 2 = 1,2 con parte entera 1 y parte decimal 0,2

- $0{,}2 \times 2 = 0{,}4$ con parte entera 0 y parte decimal 0,4
- $0{,}4 \times 2 = 0{,}8$ con parte entera 0 y parte decimal 0,8
- $0{,}8 \times 2 = 1{,}6$ con parte entera 1 y parte decimal 0,6 Por lo tanto $(0.1)_{10} = (0.0001100110011...)_2$

- **Ejemplo:** Convertir el número decimal 25.248 a binario.
  - Parte entera: 25
    - $25 \div 2 = 12$ con residuo 1
    - $12 \div 2 = 6$ con residuo 0
    - $6 \div 2 = 3$ con residuo 0
    - $3 \div 2 = 1$ con residuo 1
    - $1 \div 2 = 0$ con residuo 1
    - Por lo tanto, la parte entera de 25 en binario es 11001.
  - Parte decimal: 0.248
    - $0{,}248 \times 2 = 0{,}496$ con parte entera 0 y parte decimal 0,496
    - $0{,}496 \times 2 = 0{,}992$ con parte entera 0 y parte decimal 0,992
    - $0{,}992 \times 2 = 1{,}984$ con parte entera 1 y parte decimal 0,984
    - $0{,}984 \times 2 = 1{,}968$ con parte entera 1 y parte decimal 0,968
    - $0{,}968 \times 2 = 1{,}936$ con parte entera 1 y parte decimal 0,936
    - $0{,}936 \times 2 = 1{,}872$ con parte entera 1 y parte decimal 0,872
    - $0{,}872 \times 2 = 1{,}744$ con parte entera 1 y parte decimal 0,744
    - $0{,}744 \times 2 = 1{,}488$ con parte entera 1 y parte decimal 0,488
    - $0{,}488 \times 2 = 0{,}976$ con parte entera 0 y parte decimal 0,976
    - $0{,}976 \times 2 = 1{,}952$ con parte entera 1 y parte decimal 0,952 Por lo tanto, la parte decimal de 0.248 en binario es 0.001111101...

  Por lo tanto, el número decimal 25.248 en binario es 11001.001111101...

- **Ejemplo:** Convertir el número decimal 0.513 a octal de 3 dígitos.
  - $0{,}513 \times 8 = 4{,}104$ con parte entera 4 y parte decimal 0,104
  - $0{,}104 \times 8 = 0{,}832$ con parte entera 0 y parte decimal 0,832
  - $0{,}832 \times 8 = 6{,}656$ con parte entera 6 y parte decimal 0,656
  - $0{,}656 \times 8 = 5{,}248$ con parte entera 5 y parte decimal 0,248 Por lo tanto, el número decimal 0.513 en octal es $(0{,}4065)_{10} \approx (0{,}407)_8$.

8.6. Números con signo

- **Signo y magnitud:** El bit más significativo (el mas a la izquierda) es el bit de signo (0 = positivo, 1 = negativo).
  - **Ejemplo:** 6 y -6 en signo y magnitud.
    - $6 = 0110$
    - $-6 = 1110$
- **Rango de números con signo y magnitud:** $[-(2^{n-1} - 1)\ ,\ 2^{n-1} - 1]$.

8.6.1. Complemento a 1 y Complemento a 2

- **Complemento a 1:** Invertir todos los bits de un número.
- **Complemento a 2:** Complemento a 1 + 1.

Ejemplo

Invertir el signo de 12_{10} con complemento 2.

- Transformar 12 a binario:
  - $12 \div 2 = 6$ con residuo 0
  - $6 \div 2 = 3$ con residuo 0
  - $3 \div 2 = 1$ con residuo 1
  - $1 \div 2 = 0$ con residuo 1
  - Por lo tanto, 12 en binario es 1100.
- Aplicar complemento a 1:
  - Invertir los bits de 1100: 0011.
- Aplicar complemento a 2:
  - Sumar 1 al complemento a 1: 0011 + 1 = 0100.

  Por lo tanto, -12 en binario es 0100.

8.6.2. Operaciones con complementos

Si realizamos cálculos de sumas o restas con complementos, debemos tener en cuenta que:

- Si el resultado es mayor al rango de bits, se produce un **desbordamiento**.
- Si el resultado es negativo, se produce un **subdesbordamiento**.

Ejemplo

Sumar 5 y -3 en binario con complemento a 2.

- 5 en binario: 0101.
- 3 en binario: 0011.
- Obtener el complemento a 2 de -3:
  - 0011 → Complemento a 1: 1100 → Complemento a 2: 1101.
- Sumar 5 y -3:
  - 0101 + 1101 = 10010.

  Observamos un desbordamiento, ya que el resultado es 10 y solo tenemos 4 bits. Por lo que el resultado a tomar es 0010 = 2.

Sumar 4 y -4 en binario con complemento a 2.

- 4 en binario: 0100.
- Obtener el complemento a 2 de -4:
  - 0100 → Complemento a 1: 1011 → Complemento a 2: 1100.
- Sumar 4 y -4:
  - 0100 + 1100 = 10000.

  Observamos un desbordamiento, ya que el resultado es 100 y solo tenemos 4 bits. Por lo que el resultado a tomar es 0000 = 0.

8.6.3. Multiplicacion

Ejemplo

Ejemplo: Multiplicar 101_2 y 11_2.

$$\begin{array}{r} 101 \\ \times\ \ 11 \\ \hline 101 \\ +101 \\ \hline 1111 \end{array}$$

Por lo tanto 101_2 (5_{10}) $\times$ 11_2 (3_{10}) $=$ 1111_2 (15_{10}).

Ejemplo: Multiplicar 534_8 y 24_8.

- $534_8 \times 4$:
  - $4_8 \times 4_8 = 16_{10} = 2 \times 8_{10} + 0 = 20_8$.
  - $4_8 \times 3_8 + 2 = 14_{10} = 1 \times 8_{10} + 6 = 16_8$.
  - $4_8 \times 5_8 + 1 = 21_{10} = 2 \times 8_{10} + 5 = 25_8$.
- $534_8 \times 2_8$:
  - $2_8 \times 4_8 = 8_{10} = 1 \times 8_{10} + 0 = 10_8$.
  - $2_8 \times 3_8 + 1 = 7_{10} = 0 \times 8_{10} + 7 = 7_8$.
  - $2_8 \times 5_8 + 0 = 10_{10} = 1 \times 8_{10} + 2 = 12_8$.

$$\begin{array}{r} 534 \\ \times\ \ 24 \\ \hline 2560 \\ +1270 \\ \hline 15460 \end{array}$$

8.7. Código ASCII

El código ASCII (American Standard Code for Information Interchange) es un código de caracteres que se utiliza para representar texto en dispositivos electrónicos. El código ASCII asigna un número a cada carácter, como letras, números, signos de puntuación y otros símbolos. Por ejemplo: **Tabla de caracteres y símbolos ASCII:**

	$B_7B_6B_5$							
$B_4B_3B_2B_1$	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	‘	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	’	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	¯
1111	SI	US	/	?	O	_	o	DEL

Caracteres de control ASCII:

NULL	Null	DLE	Data Link Escape
SOH	Start of Header (inicio cabeceo)	DC1	Device Control 1
STX	Start of Text (inicio texto)	DC2	Device Control 2
ETX	End of Text	DC3	Device Control 3
EOT	End of Transmission	DC4	Device Control 4
ENQ	Petición	NAK	No Acknowledge (Ackn. negativa)
ACK	Acknowledge (Confirmación)	SYN	Synchronous Idle (Espera sincrónica)
BEL	Bell (timbre)	ETB	End of Transmission Block
BS	Backspace (retroceso)	CAN	Cancel
HT	Horizontal Tab (tabulador horiz.)	EM	End of Medium
LF	Line Feed (avance de línea)	SUB	Substitute
VT	Vertical Tab (tabulador vertical)	ESC	Escape
FF	Form Feed (avance de página)	FS	File Separator
CR	Carriage Return (retorno de carro)	GS	Group Separator
SO	Shift Out	RS	Record Separator
SI	Shift In	US	Unit Separator
SP	Space	DEL	Delete

9. Assembler

9.1. Introducción

Lenguaje ensamblador (Assembly): Instrucciones definidas en un formato legible por humanos que se traducen a instrucciones de máquina.
Lenguaje maquina: Formato binario comprensible por la CPU.

Arquitectura MIPS (Microprocessor without Interlocked Pipeline Stages): Arquitectura de computadora RISC (Reduced Instruction Set Computer) desarrollada por MIPS Technologies.

9.1.1. Principios de diseño de una arquitectura

1. Simplicidad favorece la regularidad.
   - Formato de instrucción consistente
   - El mismo numero de Operandos (dos de origen y uno de destino)
   - Facilita la codificación y el manejo en el hardware
2. Hacer que el caso común sea rápido.
   - MIPS incluye solo instrucciones simples y que son comúnmente usadas.
   - El hardware para decodificar y ejecutar las instrucciones debe ser simple, pequeño y rápido
   - Las instrucciones mas complejas (que son las menos comunes) son realizadas con múltiples instrucciones simples
   - MIPS es un ***reduced instruction set computer*** (RISC), con un pequeño numero de instrucciones simples
   - Otras arquitecturas, tales como x86 de Intel, son ***complex instruction set computers*** (CISC)
3. Mientras más pequeño, más rápido.
   - MIPS incluye solo un pequeño numero de registros
4. Buen diseño demanda buen compromiso.

9.2. Instrucciones en lenguaje ensamblador

9.2.1. Suma

Código de ejemplo en C y ensamblador MIPS

Código en C

```
1    a + b = c;
2
```

Código ensamblador MIPS

```
1    add a, b, c
2
```

Donde:

- **add:** Instrucción que suma dos registros.
- **a, b, c:** Registros donde se almacenan los valores.
- **b,c:** Operandos de entrada.
- **a:** Operando de salida.

9.2.2. Resta

Código de ejemplo en C y ensamblador MIPS

Código en C

```
1     a - b = c;
2
```

Código ensamblador MIPS

```
1     sub a, b, c
2
```

Donde:

- **sub:** Instrucción que resta dos registros.
- **a, b, c:** Registros donde se almacenan los valores.
- **b,c:** Operandos de entrada.
- **a:** Operando de salida.

9.2.3. Multiples instrucciones

Código de ejemplo en C y ensamblador MIPS

Código en C

```
1     a = b + c - d;
2
```

Código ensamblador MIPS

```
1     add t, b, c # t = b + c
2     sub a, t, d # a = t - d
3
```

Un código de varias operaciones es manejado por varias instrucciones, como se observa en el ejemplo anterior.

9.3. Operandos

Los Operandos tienen como ubicaciones físicas en el computador las siguientes:

- **Registros:** Ubicaciones de almacenamiento en la CPU.
- **Memoria:** Ubicaciones de almacenamiento en la memoria principal.
- **Constantes:** Valores constantes.

9.3.1. Registros

- Los registros son mas rápidos que la memoria.
- Se dice que MIPS es una "arquitectura de 32-bit" porque opera con datos de 32 bits.
- **Registros MIPS:** 32 registros de 32 bits.
  - $0: Siempre tiene el valor 0.
  - $31: Se utiliza para almacenar la dirección de retorno de una subrutina.
- **Registros de uso general:** $1 - $25.
- **Registros de punto flotante:** $f0 - $f31.

9.4. Fases de ejecución de una instrucción

1. **Ciclo fetch(lectura de la instrucción):** Desde que parte una instrucción hasta que comienza la siguiente.(UNA sola instrucción). La dirección de la instrucción ejecutarse se encuentra en el segmento de código.

   Memory address register (MAR) → Lectura → (MBR) → Program counter (PC) → RI.

   - **Memory address register (MAR)**: Lugar donde se guarda la dirección de memoria de la instrucción que se esta ejecutando, enviada por el PC.
   - **Memory buffer register (MBR)**: Lugar donde se guarda la instrucción efectiva que se esta ejecutando enviada por memoria.
   - **Registro de instrucciones (RI)**: Registro donde se ejecuta la instrucción efectiva enviada por el MBR.

   Recordar

   Runtime: Subdividir tiempos de atención en el orden de los nanosegundos. En una instrucción cíclica va a entrar la **pila**.

2. **Decodificación**
3. **Ejecución de la instrucción**
4. **Volver al ciclo fetch**