

TEMA I. PRINCIPIOS, IMPLEMENTACIÓN Y RENDIMIENTOS.

La máquina de Turing, niveles de descripción de un computador, algoritmos, complejidad, implementación física y medición de rendimientos.

Bibliografía: Rafael Rico López “Principios, implementación y rendimiento” Apuntes de Arquitectura de Computadores; Andrew S. Tanenbaum “Structured Computer Organization”.

Conceptos básicos

COMPUTADOR: Es una máquina destinada a procesar información que está codificada, es decir, información que está traducida en un lenguaje que entiende la máquina. Existen distintos tipos de computadores, por ejemplo: el computador de electrónica digital, que son los ordenadores de hoy en día; los computadores cuánticos, los cuales son capaces de descifrar claves con más facilidad que un ordenador ordinario; entre otros.

PROCESAR: Es la toma de decisiones que se aplica sobre la información que se está analizando. Por ejemplo, Leibniz en 1670 diseñó una máquina de cálculo (calculadora) capaz de realizar operaciones aritméticas, pero en un principio quiso que pudiera tomar la decisión de si una proposición matemática era verdadera o falsa. Por ejemplo:

$$\left. \begin{array}{ll} \forall x \in \mathbb{R} \rightarrow x^2 \geq 0 & \text{Verdadero} \\ \forall x \in \mathbb{R} \rightarrow x^3 \geq 0 & \text{Falso} \end{array} \right\} \text{Tomar decisiones}$$

AUTÓMATA: Es una máquina de cálculo abstracta que realiza cálculos (procesa la información) sobre una entrada para producir una salida.

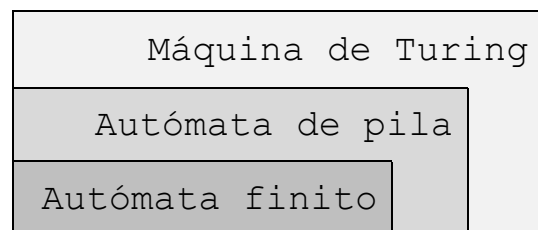
Teoría de autómatas

La teoría de autómatas es una rama de la teoría de la computación que estudia las máquinas abstractas (autómatas) y los problemas que éstas son capaces de resolver.

- **AUTÓMATA FINITO:** Es una máquina que realiza transiciones de un estado a otro dependiendo de la entrada que reciba. Solo se puede introducir un único símbolo del alfabeto que entienda la máquina. Por ejemplo $\{0,1\}$.

- **AUTÓMATA DE PILA:** Es un autómata finito que tiene memoria, organizada como una pila, por lo que puede recibir como entrada una cadena de símbolos que entienda la máquina.

- **MÁQUINA DE TURING:** Es un autómata que engloba a los otros dos (es un autómata finito y de pila) y que puede bifurcar, es decir, puede decidir.



Máquina de Turing (MT)

La **MÁQUINA DE TURING** es un autómata finito con una cinta infinita (indeterminada) compuesta por cuadros (*frames*) en los que una cabeza lectoescritora puede leer o escribir símbolos y desplaza la cinta haciendo uso de una función de transición (δ). Además, tiene dos estados de detención, que son el de aceptación y el de rechazo. No es una máquina física, sino una abstracción, es decir, un modelo teórico de una máquina capaz de resolver un problema, en donde los frames representan los datos, la cinta representa la memoria, y la función de transición a un programa (a un algoritmo).

Problema de la parada

El problema de la parada (**halting problem**) consiste en resolver si una máquina de Turing terminará una tarea en un número finito de pasos cuando se le suministra una entrada concreta. Alan Turing demostró que este problema es indecidible. Si una máquina de Turing se detiene tras realizar una serie de operaciones cuando se le suministra una entrada, esta máquina representa un algoritmo que resuelve un problema computacional. Y, al contrario, un problema no computacional es aquél que no cuenta con un algoritmo que lo solucione, por lo que la máquina nunca se detendría ya que nunca llega a una conclusión.

En definitiva, este teorema sirve para introducir los significados de computabilidad y algoritmo.

COMPUTABILIDAD: Algo que puede ser calculado o resuelto por una máquina o algoritmo.

ALGORITMO: Serie de pasos (receta) con la que se puede resolver un problema computacional.

Resumen de las diferencias entre los problemas decidibles e indecidibles:

Tipo de problema	¿Es computacional?	¿Una MT se detendrá al recibir este problema?	¿Tiene algoritmo que lo solucione?
Decidible	Sí	Sí	Sí
Indecidible	No	No	No

Tipos de máquina de Turing

Una **MÁQUINA DE TURING UNIVERSAL** es la que engloba todas las máquinas de Turing que resuelven problemas específicos. Un ejemplo de una máquina de Turing universal es un ordenador. Cabe destacar que todo lo que puede hacer una máquina de Turing se puede hacer con un lenguaje de programación moderno y viceversa.

Una **MÁQUINA DE TURING DETERMINISTA (DTM)** sigue una única secuencia de pasos para cada entrada, produciendo siempre el mismo resultado. En cambio, una **MÁQUINA DE TURING NO DETERMINISTA (NTM)** puede explorar múltiples caminos simultáneamente, permitiendo varias transiciones posibles desde un mismo estado y símbolo. Aunque ambas tienen la misma capacidad de cómputo, las NTM pueden resolver ciertos problemas más eficientemente en teoría.

Tipos de problemas computacionales por la tarea

Los problemas computacionales son aquellos que cuentan con una máquina de Turing o algoritmo para resolverlo. Según la tarea solicitada se pueden distinguir distintos tipos de problemas:

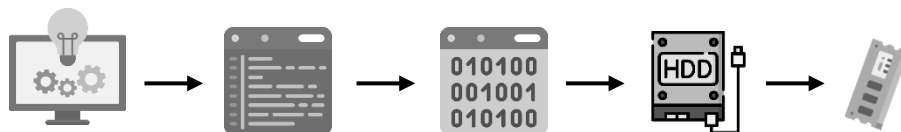
- **DE DECISIÓN:** La solución es un "sí" o un "no".
- **DE RECuento:** La solución es el número de soluciones que cumplen una condición.
- **DE BÚSQUEDA:** La solución debe cumplir una condición.
- **DE OPTIMIZACIÓN:** La solución debe ser la mejor de entre todas las posibles.

Proceso de computación

El proceso de computación son las distintas fases involucradas en la resolución de un problema computacional. El éxito de este depende de la adecuada sintonización de las distintas fases.

* Existencia del problema computacional.

1. **Planteamiento del algoritmo.**
2. **Escribir el código fuente (lenguajes, codificación, estructura de datos).**
3. **Compilar el código (traducir el código fuente a lenguaje máquina).**
4. **Obtención de la imagen binaria (programa) en el disco duro [MT estática].**
5. **Ejecutar la imagen binaria (ejecutarla en la memoria RAM) [MT dinámica].**



La única diferencia existente entre cada etapa del proceso de resolución es la forma de expresión del algoritmo.

Arquitectura de computadores

La arquitectura de computadores es el área que se encarga del diseño, mantenimiento y evaluación de los computadores. Engloba los principios de las ciencias de la computación y de la ingeniería eléctrica. Esto es, en otras palabras, el software y hardware de un computador.

Ciclo de diseño de un computador

A la hora de diseñar un nuevo computador hay que tomar decisiones acerca de una serie de rasgos básicos que lo componen. Estos rasgos básicos son el tipo de problema computacional que puede tratar, el tamaño de palabra que maneja, la tecnología y el repertorio de instrucciones que usa.

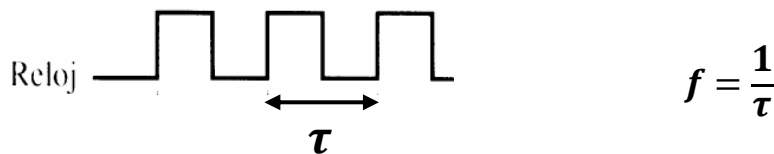
Arquitectura de computadores			
Tipo de problema	Tamaño de palabra	Tecnología	Repertorio de instrucciones

Tamaño de palabra

Se denomina **palabra** al conjunto de bits que son manejados de manera unitaria por un computador. El **tamaño de la palabra** es el número de bits contenidos en ella y fija el tamaño de los registros y de los operadores. Cuando se realiza una operación, como una suma, esta tarda un tiempo llamado retardo, que depende del tamaño de palabra. A mayor tamaño de palabra, mayor retardo, porque el circuito debe procesar más bits.

Tecnología del procesador

Un factor que determina la tecnología de un computador es la señal de reloj del procesador, el cual es una onda periódica que marca el ritmo al que el procesador ejecuta las instrucciones de una operación. Cada ciclo de esta señal tiene una duración llamada periodo (τ), que determina cuánto tiempo tiene el hardware para completar la instrucción. La inversa de este periodo equivale a la frecuencia (f) del reloj. Cuanto mayor sea la frecuencia del reloj, el periodo de cada ciclo será menor, y esto se traduce en que los ciclos de reloj van a un mayor ritmo.



Repertorio de instrucciones

El término repertorio de instrucciones se refiere a la especificación de las operaciones que puede ejecutar un procesador, los modos de acceso a los datos y cómo se codifica toda esta información. En definitiva, es el conjunto de instrucciones con el que trabaja un procesador.

Estructura de computadores (niveles de descripción)

La estructura de computadores es el área que estudia los diferentes niveles de descripción de un computador:

Niveles lógicos software	{	Aplicación	
		Sistema operativo	
		Repertorio de instrucciones	Tema 4. Repertorio de instrucciones.
Niveles físicos hardware	{	Organización	Temas 5 y 6. Unidad de control y memoria.
		Sistema	Tema 3. Diseño de operadores (ALU).
		Físico	Tema 1. Implementación.

En esta asignatura se estudian los cuatro primeros niveles, los cuales incluyen la capa física (implementación de circuitos), la capa de sistema (diseño de operadores lógicos y aritméticos), la organización (tipos de flujo de instrucciones) y el repertorio de instrucciones (conjuntos y especificaciones de las instrucciones).

Organización de computadores (capa de organización)

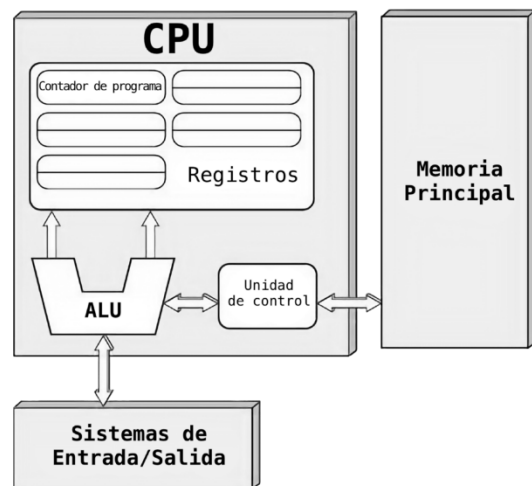
La organización de un computador es el bloque que procesa el repertorio de instrucciones usando todos los componentes del sistema. El bloque que gestiona el flujo de instrucciones se conoce como unidad de control, y el bloque que gestiona el flujo de datos se conoce como unidad aritmético lógica (ALU). El conjunto de ambos se suele denominar unidad central de proceso (CPU). Finalmente, los recursos de almacenamiento, sean de instrucciones o datos, constituyen el bloque de memoria. Dependiendo de cuál de los dos flujos guíe el orden de procesamiento, tenemos dos tipos de máquinas totalmente diferentes:

- MÁQUINAS DE FLUJO DE CONTROL
- MÁQUINAS DE FLUJO DE DATOS

Flujo de control

Las máquinas de flujo de control ejecutan las instrucciones una tras otra en un determinado orden guiadas mediante el contador de programa. Dispone de una memoria única en donde se almacenan tanto las instrucciones como los datos.

Este es el tipo de flujo que usa la arquitectura de Von Neumann, el cual es la base del funcionamiento de los computadores de hoy en día. También es la base de la programación en bajo nivel (código máquina y ensamblador), la cual evolucionaría a los lenguajes de alto nivel.



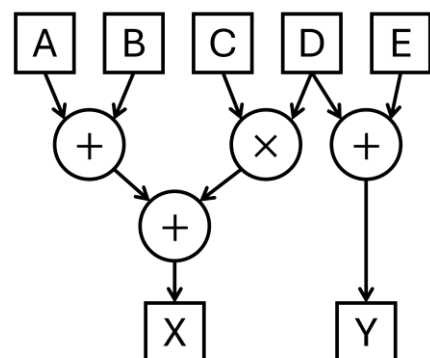
Desventajas: Dispone de un bus único para instrucciones y datos, lo que podría causar retrasos significativos. Además, su ejecución secuencial limita el aprovechamiento de la concurrencia.

Flujo de datos

Las máquinas de flujo de datos ejecutan las operaciones tan pronto como sus datos estén disponibles. No dispone de un contador de programa, por lo que no se sigue una secuencia de instrucciones fija.

Este tipo de flujo se puede representar como grafo, ya que las operaciones e instrucciones se pueden ver como nodos, y las dependencias de datos son aristas.

Fuente: [Wikipedia Arquitectura de flujo de datos](https://es.wikipedia.org/wiki/Arquitectura_de_flujo_de_datos)



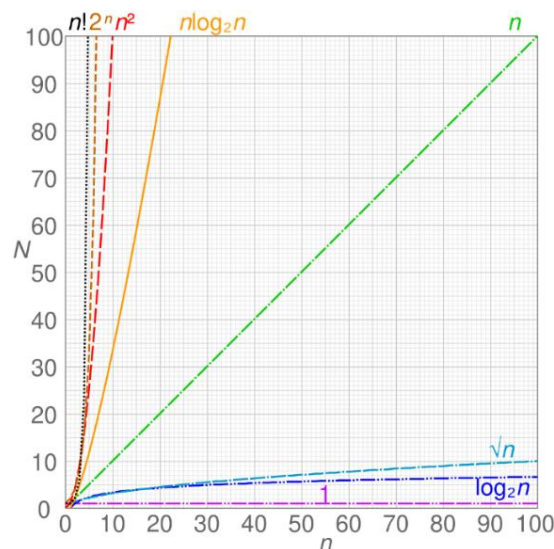
Desventajas: Es más difícil de diseñar, simular y depurar que las arquitecturas de flujo de control. Además, hay poca infraestructura de desarrollo (lenguajes, compiladores, hardware) ampliamente disponible para este tipo de flujo.

Concepto de complejidad

Previamente se mencionó que un algoritmo es complejo ya que requiere de una secuencia de pasos. La complejidad se refiere al uso de recursos requeridos para realizar una cierta tarea.

Los recursos pueden ser: **el tiempo**, medido como la cantidad de pasos de control empleados en alcanzar el final del algoritmo; y **el espacio**, medido como la cantidad de memoria utilizada. Otro tipo de recurso que se puede medir es el **consumo energético**.

La notación de Landau $O(n)$, que se lee como ***O grande de n***, es una forma de expresar el orden (cuánto crecen) el tiempo o el espacio necesarios para ejecutar un algoritmo a medida que aumenta el tamaño de la entrada (a la cual solemos llamar n).



Este diagrama muestra cómo crece el número de operaciones necesarias según la complejidad del algoritmo y el tamaño de la entrada.

Ejemplos del orden de complejidad temporal (retardo) de algunos algoritmos computacionales:

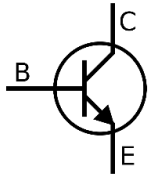
- Un **bucle simple** que va de 0 a $n \rightarrow O(n)$
- Un **bucle anidado** (dentro de otro) que va de 0 a $n \rightarrow O(n^2)$
- Una **búsqueda binaria**, que divide el problema por 2 cada vez $\rightarrow O(\log_2 n)$

Tipos de problemas computacionales por la complejidad

- **P (POLINÓMICO)**: Su orden de complejidad es polinómica o inferior. Se solucionan mediante un algoritmo (una receta).
- **NP (NO POLINÓMICO)**: Su orden de complejidad está entre exponencial y polinómica. Se solucionan a base de un algoritmo más la experiencia.
- **NP-COMPLETO**: Su orden de complejidad es exponencial o superior. Se solucionan mediante fuerza bruta, es decir, probando distintos métodos hasta que funcione.

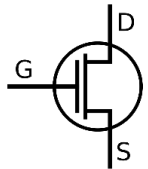
Implementación

La implementación (capa física de un computador) consiste en el diseño de familias lógicas usando transistores, también llamados semiconductores. Los semiconductores más usados a la hora de implementar son los transistores bipolares y los transistores de efecto campo.



Transistor bipolar BJT

Funciona como un interruptor. Si la base B recibe corriente, entonces conecta el colector C con emisor E. Estos tipos de transistores requieren resistencias.

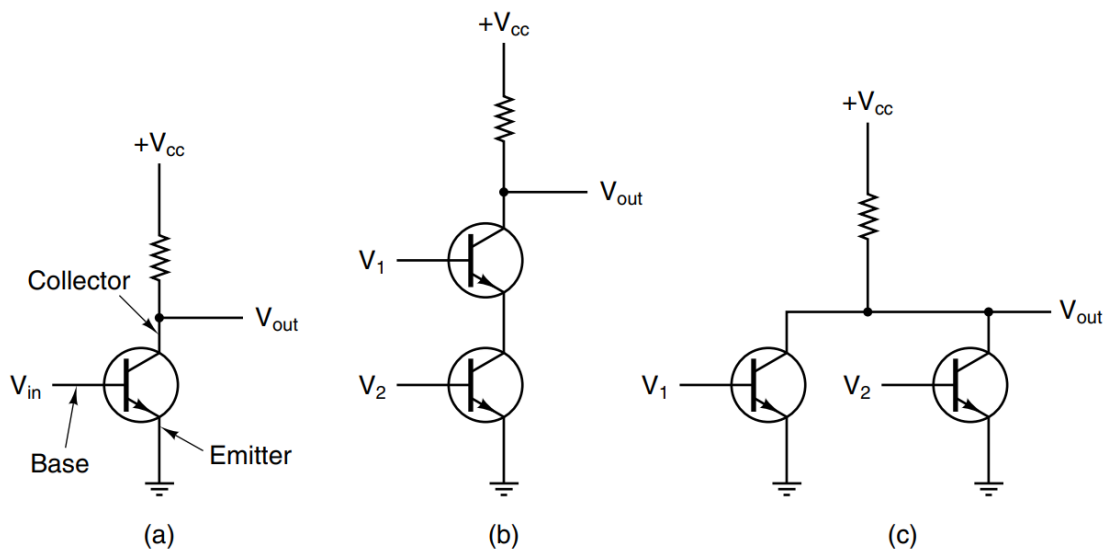


Transistor de efecto campo MOSFET

Ocupan menos área de semiconductor (son más pequeños que los BJT), son más rápidos, consumen menos potencia y no requieren resistencias.

Familia lógica RTL

La lógica RTL utiliza transistores bipolares BJT. Se caracteriza por consumir mucha energía y su gran retardo de conmutación. La implementación de esta familia lógica es la siguiente:



(a) puerta NOT RTL. (b) puerta NAND RTL. (c) Puerta NOR RTL.

Familia lógica TTL

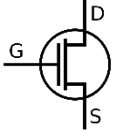
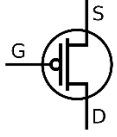
La lógica transistor-transistor utiliza transistores bipolares BJT. Tiene una elevada velocidad de propagación, pero adolece de un gran consumo energético ya que utiliza mucha potencia todo el rato.

Familia lógica ECL

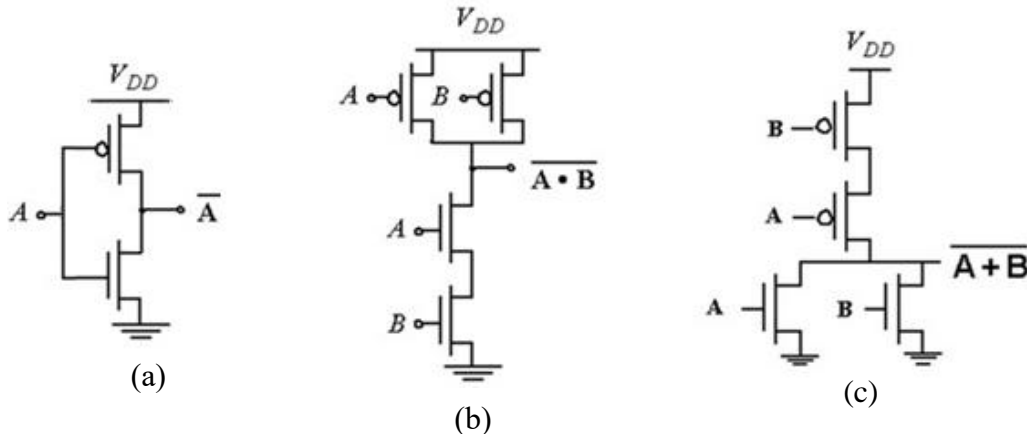
De esta familia solo conviene saber que utiliza transistores bipolares BJT y que es la más rápida de todas las familias lógicas, pero a la vez la que más energía consume de todas.

Familia lógica CMOS

La familia lógica CMOS combina transistores MOSFET NMOS y PMOS. Esta familia lógica es más rápida que la RTL, es más pequeña y es la que menos consume de todas.

<p>NMOS</p> <p>Conecta S con D si G recibe corriente.</p>		<p>PMOS</p> <p>Conecta S con D si G no recibe corriente.</p>	
--	---	---	---

La implementación de esta familia lógica es la siguiente:



(a) puerta NOT CMOS. (b) puerta NAND CMOS. (c) Puerta NOR CMOS.

Costes en la implementación

La implementación de un circuito lógico usando una cierta familia lógica tiene un cierto coste. Esto se refiere al área que necesita y el retardo con el que se propaga la electricidad.

Área (A): número de transistores y resistencias necesarias para implementar el circuito.

Retardo (R): número de bloques de conmutación por los cuales se propaga la electricidad (solo se tiene en cuenta el camino crítico, es decir, el más largo). Un bloque de conmutación es una de las puertas básicas que se puede implementar con esa familia (NOT, NAND, NOR)*.

$$A = (\text{nº transistores} + \text{nº resistencias}) \times a_{LF}$$

$$R = (\text{nº bloques de conmutación}) \times r_{LF}$$

LF (Logic Family) hace referencia a la familia lógica con la que se implementa.

Por esta razón, las puertas AND y OR tienen un retardo de $2r_{LF}$ en las lógicas RTL, TTL y CMOS, ya que requieren 2 bloques de conmutación.

$$AND(x, y) = NOT(NAND(x, y)) \quad OR(x, y) = NOT(NOR(x, y))$$

$$\text{Ejemplo: Retardo AND RTL} = r_{NAND\ RTL} + r_{NOT\ RTL} = r_{RTL} + r_{RTL} = 2r_{RTL}$$

*En la lógica ECL se pueden implementar (la AND y OR) en un solo bloque de conmutación.

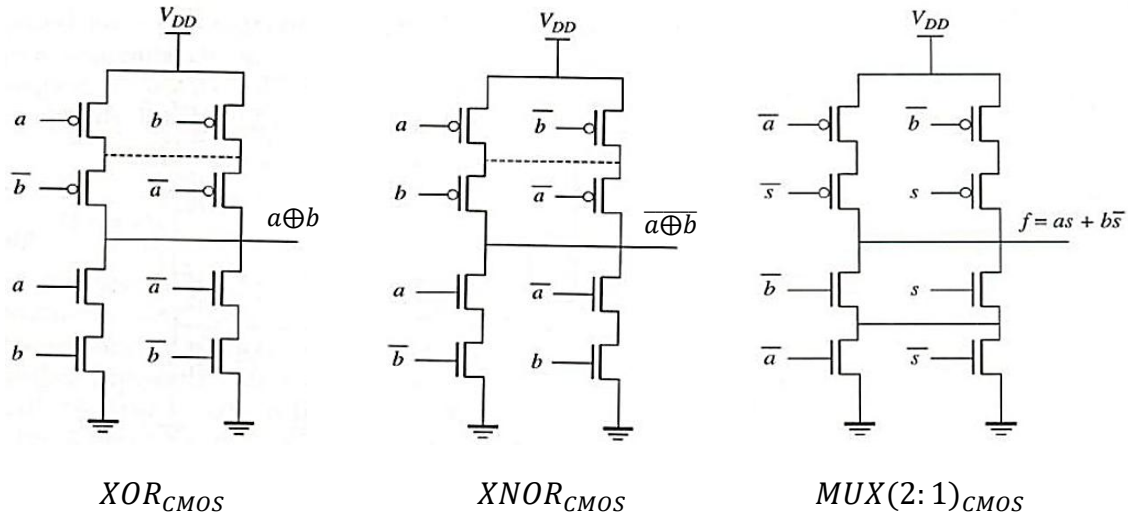
Implementación de puertas XOR, XNOR y multiplexores

La puerta XOR se puede implementar usando la fórmula: $XOR = x \oplus y = \bar{x}y + x\bar{y}$

Y para implementar un multiplexor de 2 a 1 se tiene: $MUX(2:1) = xc + y\bar{c}$

c es la entrada de la señal de control del multiplexor.

Su implementación en las lógicas RTL y TTL tiene un retardo de $R = 3r_{LF}$. Sin embargo, en la lógica CMOS se pueden implementar usando los siguientes bloques de conmutación:



Teniendo en cuenta los inversores NOT, el retardo de cada implantación es de $R = 2r_{CMOS}$.

El problema del *fan-in*

El *fan-in* hace referencia al número de entradas que puede llegar a soportar una puerta y viene limitado por las características eléctricas de cada familia (por lo general solo soportan hasta 3). Por lo tanto, habrá que buscar métodos para implementar circuitos que reciban un gran número de entradas. Por ejemplo, si a partir de puertas OR de 2 entradas queremos crear una OR de n entradas podemos usar una configuración en cascada (serie) o usar una en paralelo.

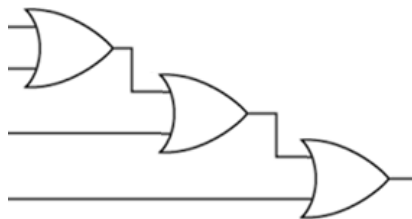


Figura 1

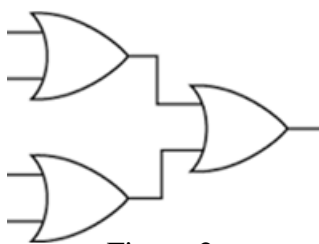


Figura 2

Ambas implementaciones requieren $n - 1$ puertas, por lo que el coste en área crece siguiendo un orden de complejidad de $A = O(n)$. En cambio, el retardo varía dependiendo de cómo coloquemos las puertas. La implementación **por cascada** (figura 1) presenta una longitud proporcional a n , por lo tanto, sigue un orden de $R = O(n)$.

Pero, la implementación **en paralelo** (figura 2) sigue un orden de complejidad de $R = O(\log_2 n)$, esto es debido a que disminuye la longitud del camino crítico (en este caso son divisiones de 2).

En general, la opción paralela, con puertas de *fan-in* f (con f número de entradas), ofrece una complejidad temporal $O(\log_f n)$, la cual es mejor que la de la opción en cascada.

Tiempo de ejecución

El tiempo de ejecución de una tarea computacional es igual al número de ciclos de reloj requeridos para realizar la tarea multiplicado por el periodo del reloj del procesador usado.

$$t = \text{ciclos} \cdot \tau$$

No obstante, el tiempo de ejecución real (tiempo de pared o *wall time*) de una tarea computacional sería la suma del tiempo de cómputo de la tarea en el procesador, más el tiempo de cómputo por el sistema operativo, más el tiempo que está en espera la tarea.

$$\begin{aligned} t_w &= t_{CPU} + t_{SYS} + t_{espera} \\ t_w &= (\text{ciclos}_{CPU} + \text{ciclos}_{SYS} + \text{ciclos}_{espera}) \cdot \tau \end{aligned}$$

Recuento de instrucciones

Un programa (que también se puede ver como un o varios algoritmos) está formado por una serie de instrucciones del repertorio del procesador que debe de ejecutarlos.

El r_i (**recuento de la instrucción i**) es el número de número de veces que se ejecuta una instrucción del mismo tipo. Por ejemplo, si un programa realiza las siguientes instrucciones: 100 sumas, 50 cargas, 20 saltos; los distintos recuentos r_i serían los siguientes:

$$r_{\text{sumas}} = 100, \quad r_{\text{cargas}} = 50, \quad r_{\text{saltos}} = 20$$

Por otra parte, el r_p (**recuento de un programa**) es el sumatorio de la cantidad de veces que se usa cada una de las n instrucciones del repertorio en ese programa. Es decir, es la suma de todos los tipos de instrucciones ejecutadas en el programa.

$$r_p = \sum_i^n r_i$$

Ciclos por instrucción

Cada tipo de instrucción necesita una cierta cantidad de ciclos de reloj para ejecutarse. Esto depende del tipo de operación y del hardware.

El **CPI (Ciclos Por Instrucción)** es una medida que sirve para determinar el número de ciclos que necesita una instrucción. Si sabemos los CPI y recuentos de las n instrucciones que realiza un tarea computacional, podemos determinar entonces el número de ciclos que necesita:

$$\text{ciclos} = \sum_i^n \text{CPI}_i \cdot r_i$$

También se puede hallar el CPI promedio del repertorio de instrucciones (con n instrucciones) de un procesador si se conocen los CPI de cada instrucción del repertorio.

$$\text{CPI}_{\text{set}} = \frac{1}{n} \cdot \sum_i^n \text{CPI}_i$$

De forma análoga, para un programa singular, podemos calcular su CPI promedio si conocemos con exactitud la composición de su mezcla de instrucciones.

$$CPI_p = \frac{1}{r_p} \cdot \sum_i^n CPI_i \cdot r_i = \frac{ciclos_p}{r_p}$$

Tiempo de programa (medición del rendimiento)

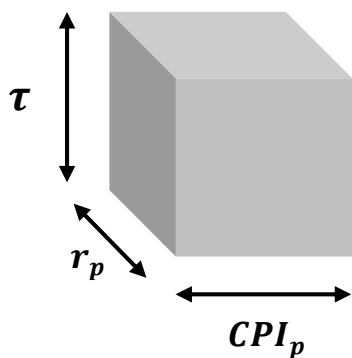
El rendimiento de un programa es inversamente proporcional al tiempo de ejecución del programa, ya que esta medida muestra el tiempo consumido para el procesamiento de una tarea computacional, materializada en un programa. El tiempo de programa es el siguiente:

$$t_p = CPI_p \cdot r_p \cdot \tau = ciclos_p \cdot \tau$$

Sabiendo el tiempo de un programa podemos determinar su rendimiento y productividad:

$$\text{Rendimiento} \propto \frac{1}{t_p} \quad \text{Productividad} = \frac{\text{num procesos}}{t_p}$$

Interpretación tridimensional del tiempo de programa



Siendo el tiempo de programa el producto de tres términos, podemos interpretarlo como el volumen de un prisma, tal y como vemos en la figura.

Los términos recuento r_p y ciclos por instrucción CPI_p , en la base, son debidos a decisiones de diseño, es decir, dependen de la **arquitectura**.

El término correspondiente al periodo de reloj τ , la altura, depende de la **tecnología**.

Millones de instrucciones por segundo

Contabilizar los millones de instrucciones por segundo (MIPS) es una forma de medir la potencia de cómputo de los procesadores, que consiste en dar la cantidad en millones de instrucciones ejecutadas durante un segundo. Formalmente para hallarlo tenemos:

$$MIPS = \frac{1}{CPI} \cdot f \cdot 10^{-6} \quad (\text{si } f \text{ está expresada en Hz})$$

Si queremos determinar el valor promedio de MIPS de un programa, para ello debemos de aplicar la formula anterior usando el CPI promedio del programa. Y para hallar el MIPS de pico, es decir, el valor óptimo de MIPS, se toma la instrucción con menor CPI. Resumen:

$$MIPS_p = \frac{1}{CPI_p} \cdot f \cdot 10^{-6} \quad MIPS_{\text{de pico}} = \frac{1}{CPI_{\text{más pequeño}}} \cdot f \cdot 10^{-6}$$

Coeficiente de mejora

Para poder valorar cuánto de buena es una posible mejora en el rendimiento de un proceso, es decir, cuánto se puede disminuir el tiempo de ejecución, se puede hacer uso de la **ley de Amdahl (Speed-Up)**, un coeficiente que nos indica si el tiempo de ejecución de un proceso mejora o no. Teniendo que el tiempo con mejora es:

$$t_{\text{con mejora}} = t_{\text{sin mejora}} \cdot ((1 - f_m) + \frac{f_m}{a_m})$$

donde f_m es la fracción de tiempo que utiliza la mejora y a_m el factor de mejora alcanzado, para determinar el coeficiente de mejora (*Speed-Up*) usamos la siguiente fórmula:

$$S = \frac{t_{\text{sin mejora}}}{t_{\text{con mejora}}} = \frac{1}{(1 - f_m) + \frac{f_m}{a_m}}$$

- Si $S > 1$, entonces significa que el tiempo mejora
- Si $S < 1$, entonces significa que el tiempo empeora
- Si $S = 1$, entonces significa que el tiempo no mejora, es el mismo

Recordatorio

El periodo es la duración de un ciclo de reloj, y la inversa de este valor es la frecuencia. Para realizar la conversión del periodo (medido en segundos) a la frecuencia (medido en hercios) hay que tener en cuenta lo siguiente.

Periodo	Frecuencia	Resumen
Si está medido en segundos...	... su inversa son los hercios	$s^{-1} = \text{Hz}$, $\text{Hz}^{-1} = s$
Si está medido en microsegundos	... su inversa son los megahercios	$\mu s^{-1} = \text{MHz}$, $\text{MHz}^{-1} = \mu s$
Si está medido en nanosegundos	... su inversa son los gigahercios	$ns^{-1} = \text{GHz}$, $\text{GHz}^{-1} = ns$

Conversiones de tamaño en el periodo:

Expresado en...	1 segundo	1 microsegundo	1 nanosegundo
... segundos	1s	$10^{-6}s$	$10^{-9}s$
... microsegundos	$10^6\mu s$	1 μs	$10^{-3}\mu s$
... nanosegundos	10^9ns	10^3ns	1ns

Conversiones de tamaño en la frecuencia:

Expresado en...	1 hercio	1 megahercios	1 gigahercio
... hercios	1Hz	10^6Hz	10^9Hz
... megahercios	10^{-6}MHz	1MHz	10^3MHz
... gigahercios	10^{-9}GHz	10^{-3}GHz	1GHz