

Notas Para el Curso de Introducción a la Ingeniería de Software y Datos

POR JUAN CAMILO MACÍAS

Facultad de Ingeniería y Ciencias Agropecuarias
IUDigital de Antioquia
2024

Índice

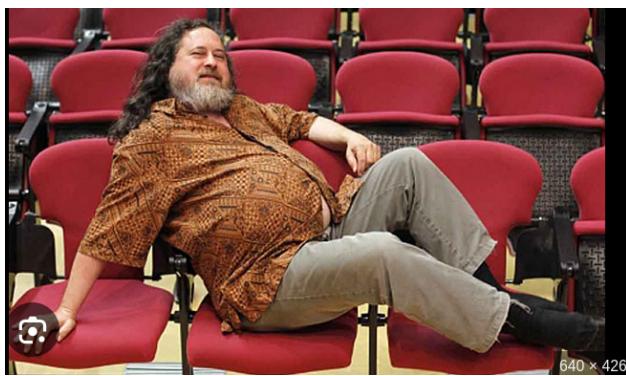
1 Actividad de conocimientos previos	2
2 Hardware	8
1 El ábaco	9
1.1 En base 10	9
1.2 En base 2	11
2 Lógica de las Operaciones matemáticas	11
Reto 1 :	12
Reto 2:	12
Reto 3:	13
Pregunta:	13
Reto 4	13
2.1 Usando el computador para simular las compuertas lógicas	14
2.1.1 Compuerta NOT	14
Reto 1	15
Reto 2	16
2.2 Sumas de 2 bits	16
Reto:	17
Respuesta	17
Reto	17
2.2.1 Construyendo un Sumador de 2 Bits	17
Un paso al frente: Llevado en cuenta lo que sobra	19
Reto:	21
3 Computadores de tubos al vacío	22
3.1 El transistor	26
3 Software	30
1 Lenguajes de programación	35
1.1 Decodificadores Binarios	36

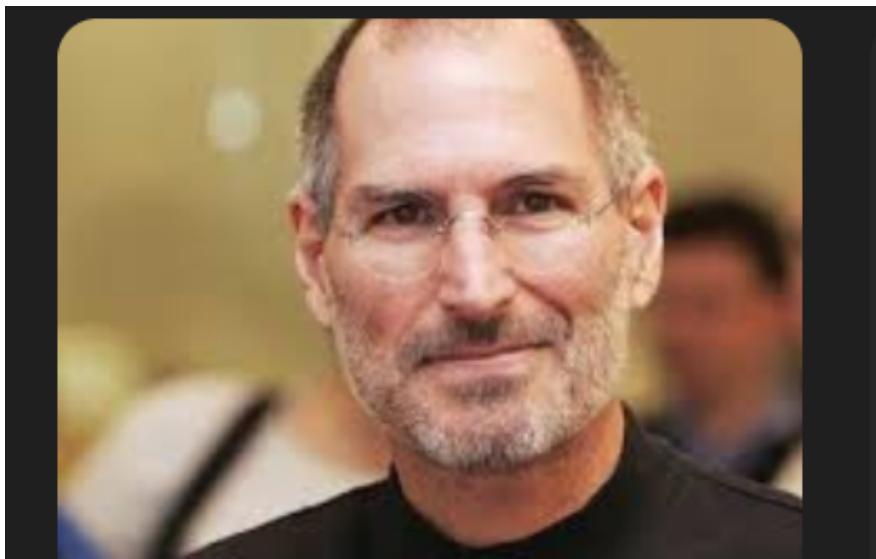
1.2 Breve historia del software	42
2006 – Rust	44
1.3 Git	45
1.3.1 Instalación de Git	45
En Linux:	45
En Windows:	45
Configurar Git	45
Inicializar un repositorio	45
Crear un nuevo repositorio:	45
Inicializar un repositorio	45
Crear un nuevo repositorio:	46
Preparar archivos: <code>git add</code> y la área de “staging”	46
Aregar archivos al área de staging:	46
Guardar cambios: <code>git commit</code>	46
Branches (Ramas) y Cambiar entre ellas	46
Fusionar Branches: <code>git merge</code>	46
1.4 GitHub	47
1.4.1 Estableciendo la conexión	47
Ejecuta el comando para generar la llave:	47
Añadir la llave SSH al agente SSH	47
Iniciar el agente SSH:	47
Añadir tu nueva llave SSH al agente:	47
Añadir la llave SSH a GitHub	47
6. Probar la conexión	49
6.1 Ejecuta este comando para probar la conexión:	49
2 El flujo de trabajo Git Fork	49
2.1 Clonar un repositorio mediante un "fork"	49
2.2 Clonar tu repositorio fork a tu máquina local	49
2.3 Crear una rama para los cambios	50
2.4 Hacer cambios y commits	50
2.5 Subir los cambios a tu fork en GitHub	50
2.6 Crear un "Pull Request" (PR)	50
2.7 Sincronizar tu fork con el repositorio original (upstream)	50
4 Datos	51
1 La internet	51
1.1 Fibra óptica	51
1.2 Cómo los datos son transmitidos	51
1.3 Cómo los datos son encriptados	51
1.3.1 El Algoritmo de Diffie-Hellman	51

1 Actividad de conocimientos previos

Levanta la mano si reconoces estos personajes:













2 Hardware

Antes de comenzar, es fundamental que comprendamos mejor nuestra herramienta de trabajo: el computador. La máquina que tienes a tu disposición, por más modesta que sea, es un dispositivo de altísima tecnología, mucho más poderoso que las computadoras utilizadas para los cálculos de la primera bomba atómica o el proyecto que llevó al hombre a la luna. Los pioneros de la ciencia de la computación y la ingeniería de software no contaron con las computadoras que hoy se pueden adquirir en cualquier tienda. Incluso la más básica de las computadoras actuales habría sido un sueño para ellos.

Sin embargo, las limitaciones de las computadoras a las que los programadores tenían acceso en las décadas de los 70, 80 y 90, exigían un profundo entendimiento del funcionamiento interno de estas máquinas, algo que muchos programadores actuales no poseen.

Hoy en día, aprendemos sobre paradigmas de programación, estructuras de datos y lenguajes de programación, pero a menudo sin comprender completamente por qué fueron diseñados de la manera en que lo fueron. Las herramientas modernas facilitan enormemente nuestro trabajo diario, pero también convierten

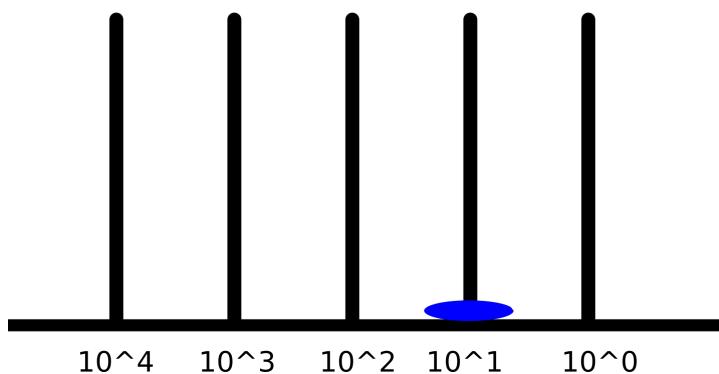
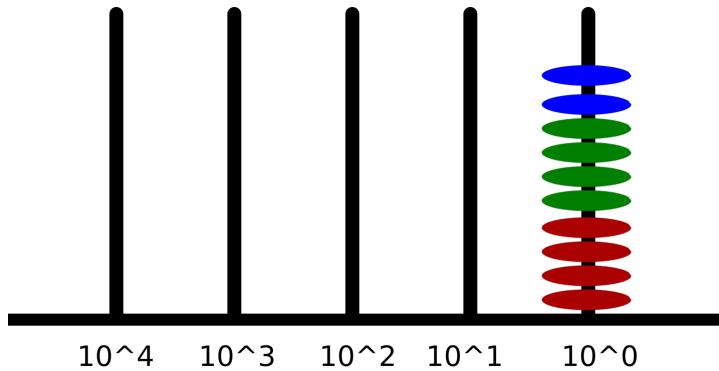
al computador en una caja negra. La capacidad de innovar y proponer nuevos paradigmas o enfoques revolucionarios se encuentra oculta bajo capas y más capas de abstracción, apiladas sobre la más fundamental de todas: el hardware.

Comenzaremos nuestra introducción al mundo de la ingeniería de software y datos desde esta primera capa.

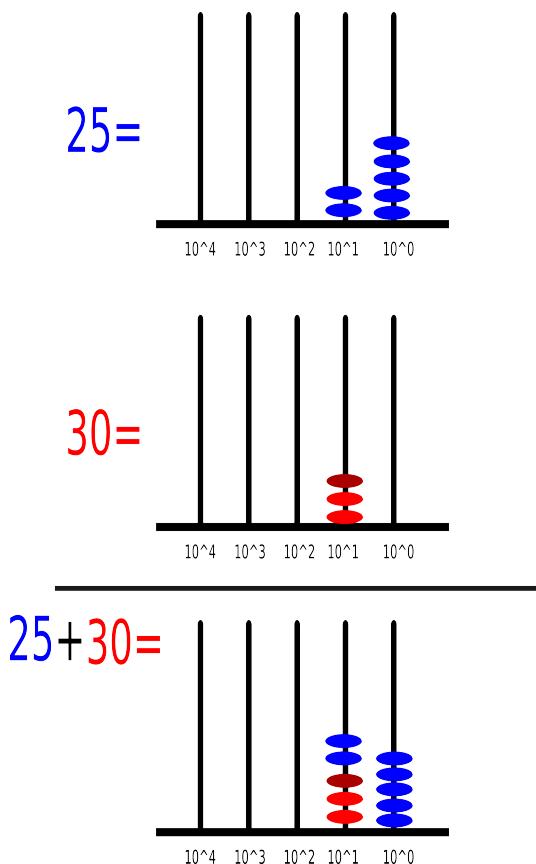
1 El ábaco

1.1 En base 10

Para entender como se hacen cálculos matemáticos con variables recurriremos al ábaco. El ábaco de la figura representamos por ejemplo el número 10:

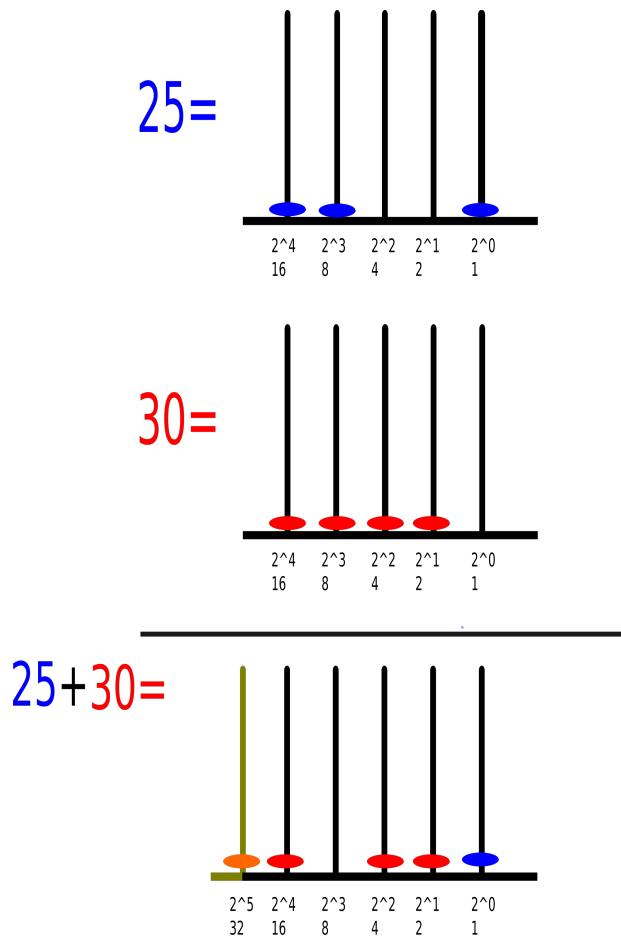


Cada unidad en el asta, representada como $10^0 = 1$, tiene un valor individual de 1. Por lo tanto, las 10 unidades en el primer lugar pueden ser reemplazadas por una sola en el siguiente donde cuenta como $10^1 = 10$. Sucesivamente podemos representar otros números, y su suma, en base 10. Por ejemplo:



1.2 En base 2

Si construimos nuestro computador en base 2, necesitaremos más ‘bits’ para representar los mismos números que usamos en base 10. En el ejemplo anterior, tendríamos que diseñar un computador con un mayor número de ástas.



¿Por qué, entonces, los computadores utilizan base 2 si requieren más memoria? Lo veremos en nuestro próximo encuentro. Por el momento la moraleja es la siguiente:

Podemos representar números en como objetos físicos ocupando lugares con significado abstracto.

2 Lógica de las Operaciones matemáticas

Analicemos los siguientes retos:

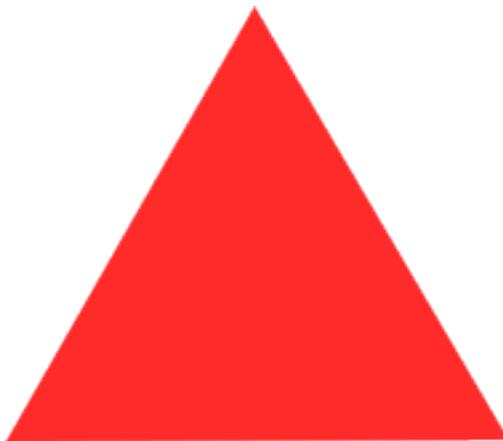


Figura 1.

Reto 1 : Responde VERDADERO o FALSO:

1. ¿El bloque lógico de la figura es rojo AND es triangular?
 - a) VERDADERO
 - b) FALSO
2. ¿El bloque lógico de la figura es rojo AND NO es triangular?
 - a) VERDADERO
 - b) FALSO
3. ¿El bloque lógico de la figura NO es rojo AND es triangular?
 - a) VERDADERO
 - b) FALSO
4. ¿El bloque lógico de la figura NO es rojo AND NO es triangular?
 - a) VERDADERO
 - b) FALSO

Si respondiste las preguntas estás listo para el siguiente reto:

Reto 2: Dadas las expresiones

A = El bloque de la figura es rojo.

B = El bloque de la figura es triangular.

Si asignamos el valor 1 para indicar que A o B son verdaderos, y el valor 0 para indicar que son falsos, completa la siguiente tabla:

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Tabla 1. 1 significa que VERDADERO, 0 significa FALSO.

Reto 3: El circuito contiene 2 interruptores **A** y **B**. Representamos un interruptor abierto por 0 y uno cerrado por 1. Completa la tabla:

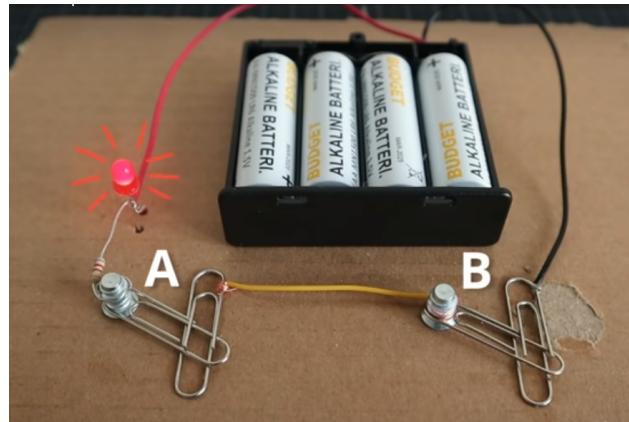


Figura 2. Aquí tenemos un circuito simple con dos entradas A y B. La luz solamente enciende cuando las dos entradas están cerradas. La foto de la izquierda, el circuito pude construirse usando switch de lámparas que venden en las ferreterías. En Yolombó se pueden comprar en «ElectroAlejo» en el parque principal.

A	B	Luz
0	0	0
0	1	0
1	0	0
1	1	1

Tabla 2. Recorda: luz encendida es 1 y apagada es 0.

Pregunta: A qué tabla de verdad se te parece la tabla 2?

Reto 4 Para el circuito de la figura completa la tabla:

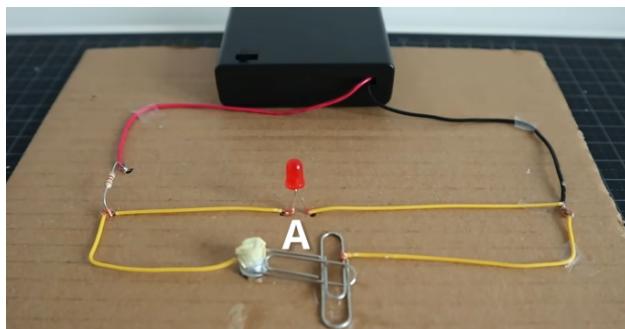


Figura 3.

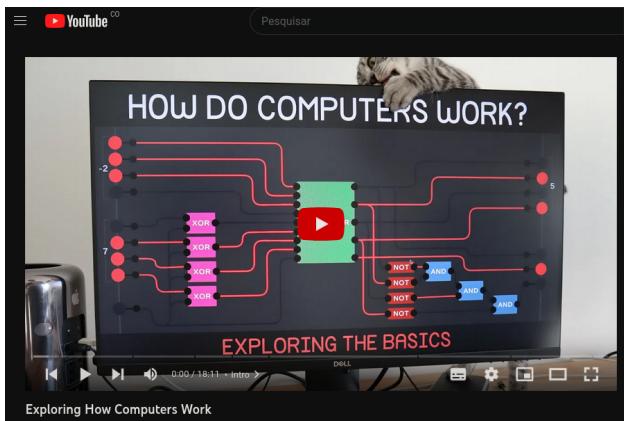
A	Luz
0	1
1	0

Tabla 3.

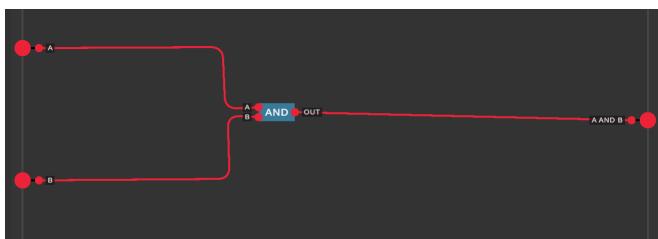
Qué nombre le darías a este operador?

2.1 Usando el computador para simular las compuertas lógicas

Descarga el simulador de compuertas lógicas, <https://github.com/DigitalLogicSimCommunity/Digital-Logic-Sim-CE/releases/tag/v0.39.1>. Puedes explorar el contenido del video de YouTube <https://www.youtube.com/watch?v=QZwneRb-zqA&list=PLn20Vn487hvyG983yv1xMQBe6F7kdDRkK&index=6> para entender como funciona el simulador de compuertas lógicas.

**Figura 4.**

El simulador nos permite experimentar con la compuerta AND

**Figura 5.**

La salida solo está encendida cuando A y B están encendidas.

2.1.1 Compuerta NOT

Otra compuerta interesante es la compuerta NOT, que es simplemente la negación de un valor de verdad. En este caso si A es verdadero su negación NOT A es falso y reciprocamente, si A es falso, NOT A es

verdadero. Entonces la tabla de verdad se vería como:

A	NOT A
1	0
0	1

Tabla 4.

Un circuito que emula este comportamiento es mostrado en la figura:

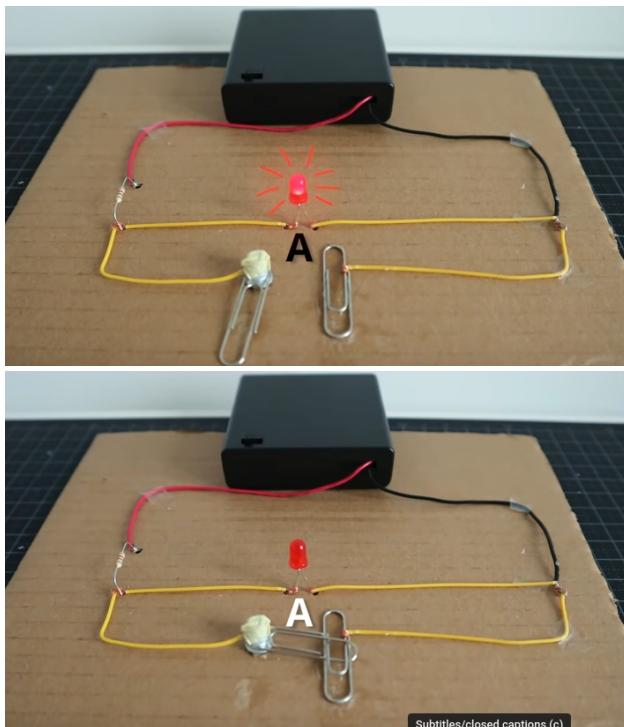


Figura 6.

Reto 1

Usando las compuertas AND and NOT. Construir la compuerta NAND que tiene el comportamiento de la tabla

A	B	Luz
0	0	1
0	1	1
1	0	1
1	1	0

Tabla 5. La luz solamente está apagada cuando las dos compuertas están abiertas. Esta es la tabla de verdad de la conjunción o NAND en Inglés, por ser la negación de AND

Solución:

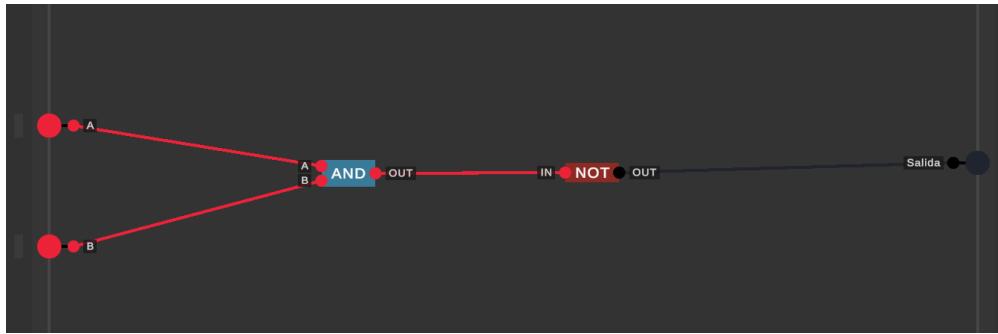
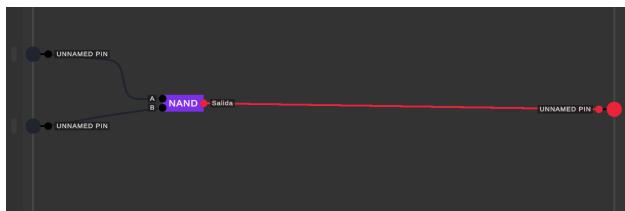


Figura 7.

Este arreglo de compuertas puede resumirse en el circuito integrado NAND



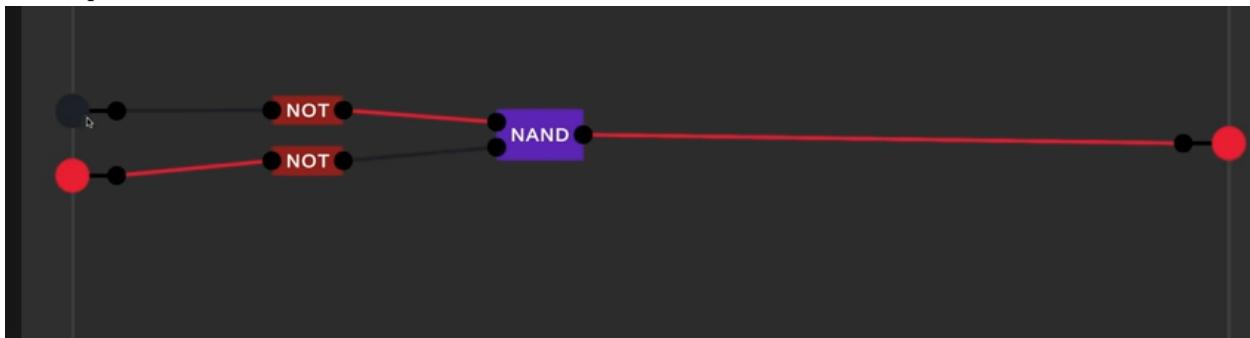
Reto 2

Usando las compuertas AND and NOT. Construir la compuerta OR que tiene el comportamiento de la tabla

A	B	Luz
0	0	0
0	1	1
1	0	1
1	1	1

Tabla 6. La luz solamente está apagada cuando las dos compuertas están cerradas. Esta es la tabla de verdad de la disyunción u OR en Inglés.

Respuesta:



2.2 Sumas de 2 bits

Haciendo uso del ábaco sumamamos $10001 + 10101$. El objetivo de esta actividad es representar números en binario y ver su relación con las compuertas lógicas.

Reto: Despues de lo que descubriste en el ábaco, completa la siguiente tabla con 1 o 0

A	B	Reciduo	Suma
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Tabla 7.

Respuesta

A	B	Reciduo	Suma
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Tabla 8.

Reto : A qué compuerta lígica se te parece el resultado del reciduo?

Respuesta: Es la compuerta AND

2.2.1 Construyendo un Sumador de 2 Bits

Puede verifica que esta es la tabla de verdad de la suma de 2 Bits

A	B	Lo que sobra	Suma
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Tabla 9. Tabla de verdad de una suma de 2 Bits. Cuando las dos entradas son 1, el resultado de la suma es 0, pero sobra 1.

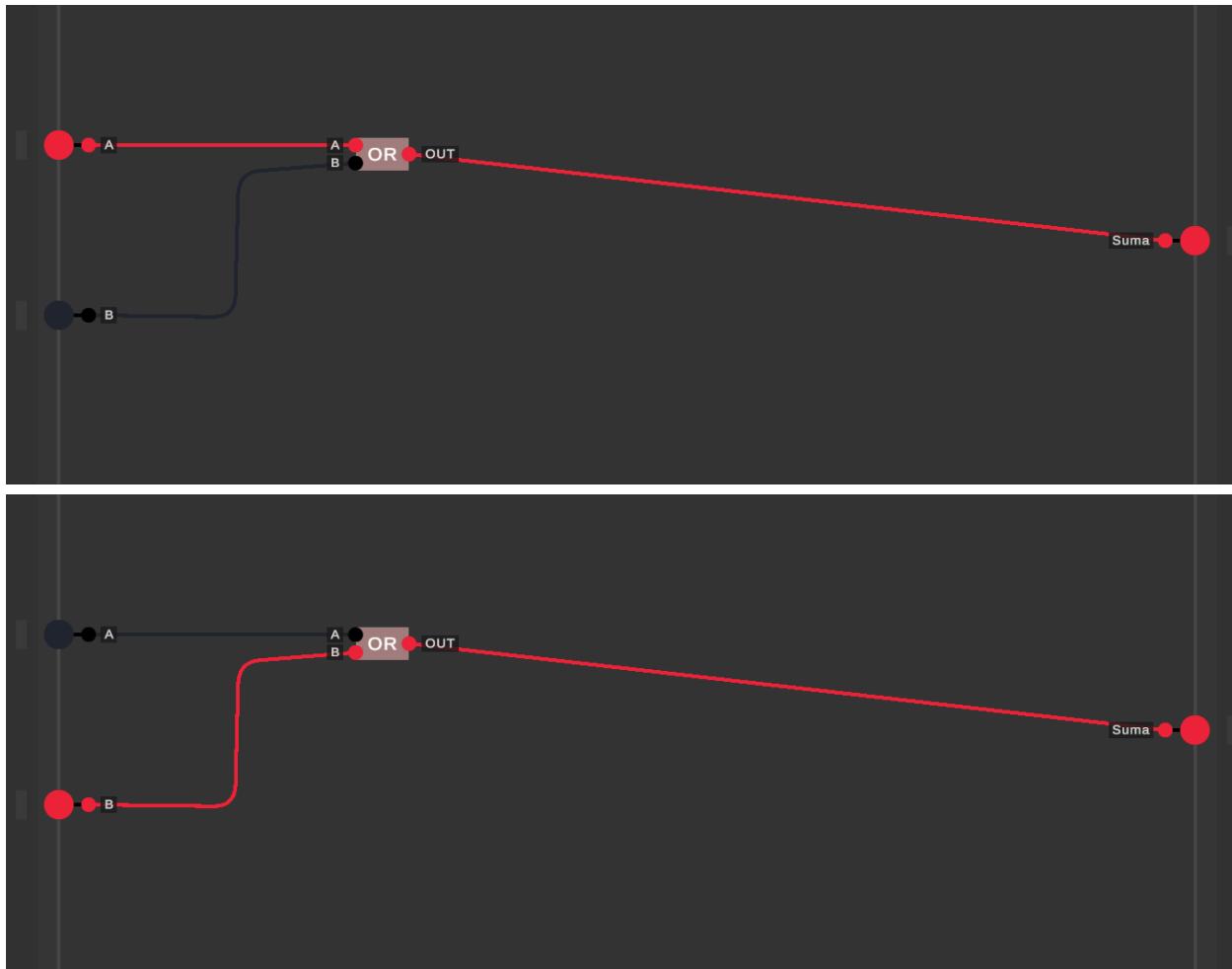
Reto: Observa la columna de lo que sobra. A que operacidor lógico se te parece?

Respuesta: El el operador AND.

Reto: Observa la columna **Suma** a que operador lógico se te parece?

Respuesta: Se parece al operador OR. Sólo que en vez de tener un 1 en la última fila, tiene un cero.

Teneindo en cuenta tu respuestas a los dos retos anteriores, tomemos la compuerta OR como punto de partida

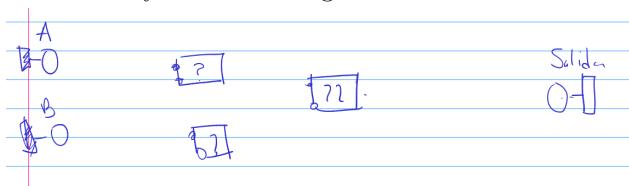


La suma es 1 cuando alguno de los as dos entradas es 1. Pero queremos replicar este comportamiento:

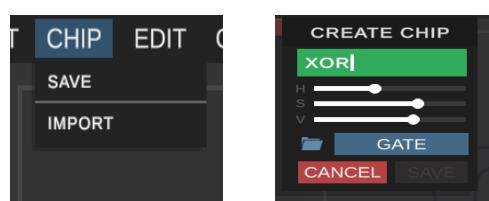
Suma
0
1
1
0

Cómo lo harías?

Reto: Ensayar varias configuraciones en el simulador para lograr el comportamiento de la suma.



De hecho este arreglo de compuertas tiene un nombre especial es llamado el OR exclusivo (XOR). Puedes usar el menú CHIP -> SAVE para resumir esta configuración en un solo chip integrado.



Un paso al frente: Llevado en cuenta lo sobra

Volvamos a la suma, supongamos que queremos sumar dos números binarios. Por ejemplo:

$$\begin{array}{r} 100011 \\ + 000111 \\ \hline \end{array}$$

Sumamos las columnas como lo hacemos desde primero de escuela, solo que en binario. Entonces , la primera columna nos da $1 + 1 = 10$ pongo 0 y llevo el 1:

$$\begin{array}{r}
 & | \rightarrow \text{Lo que sobra} \\
 100011 & \\
 + 000111 & \\
 \hline
 0 \rightarrow \text{Suma.}
 \end{array}$$

En la siguiente columna tenemos $1 + 1 + 1 = 11$, pongo el 1 y llevo 1 y así sucesivamente con las demás columnas

$$\begin{array}{r}
 | | | \rightarrow \text{Lo que sobra} \\
 100011 \\
 + 000111 \\
 \hline
 101010 \rightarrow \text{Suma.}
 \end{array}$$

¡Excelente! ¡Hemos descubierto cómo construir un chip sumador que cumple con los requisitos de la columna ‘Suma’ en la tabla! Ahora el desafío consiste en diseñar un chip que tenga una salida adicional para representar ‘lo que sobra’ y que esta nueva salida se comporte de acuerdo a la columna ‘Lo que sobra’. ¡Manos a la obra!

Lo que sobra	Suma
0	0
0	1
0	1
1	0

Reto: Observa el siguiente chip incompleto



Aquí tienes un emocionante desafío! Si decides aceptarlo, tu misión es utilizar el chip que acabas de crear, junto con otros que ya conoces, para lograr que la salida correspondiente a “lo que sobra” se comporte de acuerdo a la tabla que descubriste para la suma de 2 bits. ¡Demuestra tu ingenio y habilidades para resolver este desafío! ¡Adelante!

A	B	Lo que sobra	Suma
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Tabla 10. Tabla de verdad de una suma de 2 Bits. Cuando las dos entradas son 1, el resultado de la suma es 0, pero sobra 1.

3 Computadores de tubos al vacío

La conexión entre los circuitos eléctricos y la lógica fue establecida por Claude Shannon en 1937, quien se basó en los trabajos de George Boole de 1864 específicamente en *An Investigation of the Laws of Thought on Which are Founded the Mathematical Theories of Logic and Probabilities*. Shannon fue el primero en reconocer que las operaciones booleanas podían representarse mediante circuitos electrónicos.

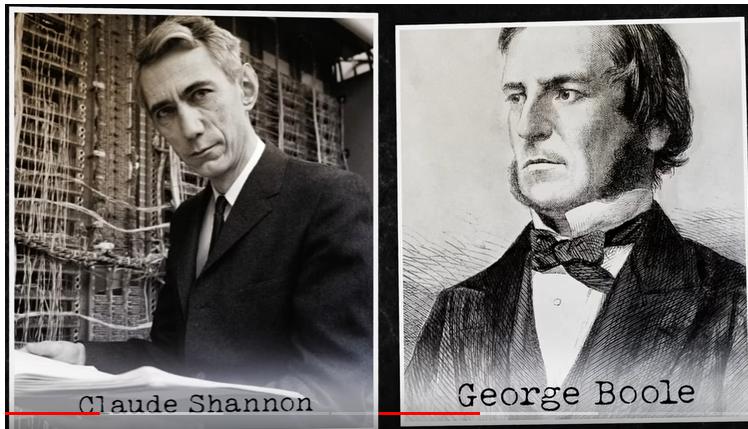


Figura 8.

Shannon observó lo que hemos explorado en las secciones anteriores: es posible reconstruir operaciones lógicas en circuitos mediante la activación y desactivación de interruptores eléctricos dispuestos de manera ingeniosa. Sin embargo, estos interruptores debían ser activados manualmente. Al principio, se intentó automatizar este proceso utilizando interruptores magnéticos, pero con el tiempo surgieron tecnologías más eficientes, como los tubos de vacío y, más recientemente, los transistores. Para introducir lo que hacen los transistores, la figura muestra la compuerta NOT que estudiamos en la clase anterior, donde el interruptor ha sido reemplazado por un pequeño transistor.

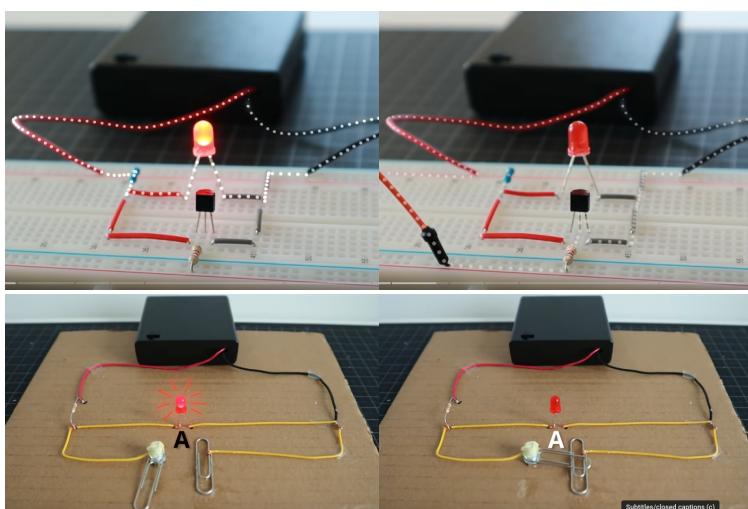


Figura 9.

En el panel derecho de la figura, una pequeña corriente hace que el transistor conduzca energía, impiendiéndole así que la energía fluya a través del bombillo, lo que provoca que este se apague. La física del transistor permite que funcione como un circuito cerrado en presencia de la corriente y abierto en su ausencia. Para comprender mejor su origen y funcionamiento, es útil estudiar cómo evolucionaron a partir de los tubos de vacío.

Los tubos de vacío eran, en esencia, bombillos. Thomas Alva Edison fue el primero en notar que sus bombillos siempre se oscurecían en un lado. Esto se debe a que, cuando el filamento dentro de un tubo de vacío se calienta al aplicar una corriente eléctrica, su temperatura aumenta significativamente. Este filamento, generalmente hecho de tungsteno, alcanza temperaturas tan altas que algunos de sus electrones adquieren suficiente energía para liberarse de los átomos que los mantienen unidos. Este proceso, conocido como **emisión termoiónica**, ocurre porque la agitación térmica proporciona a los electrones la energía necesaria para superar la barrera de potencial que normalmente los mantiene ligados al núcleo del átomo.

Como las bombillas de Edison funcionaban con corriente continua, se generaba un campo eléctrico en el espacio entre el cátodo (negativo) y el ánodo (positivo). Dado que los electrones son partículas cargadas negativamente, son atraídos hacia el ánodo cargado positivamente. Algunos de estos electrones impactaban contra el vidrio del tubo, causando su oscurecimiento gradual.

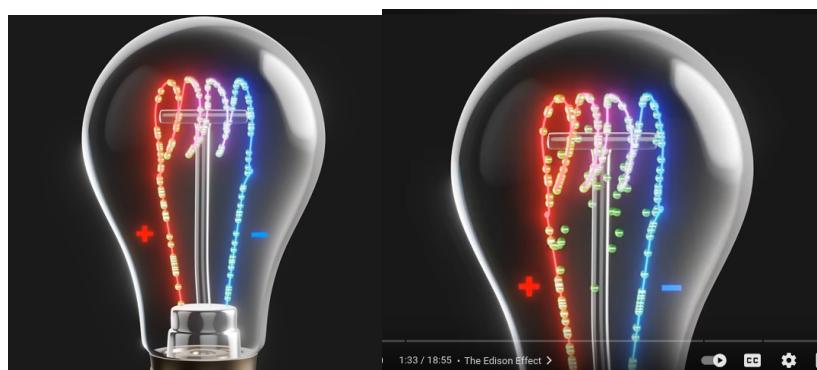


Figura 10. https://www.youtube.com/watch?v=FU_YFpfDqqA

Este flujo de electrones, que ocurre naturalmente en las bombillas de Edison solo en una dirección, puede invertirse si se agrega un pequeño filamento en el centro, como se muestra en la figura.

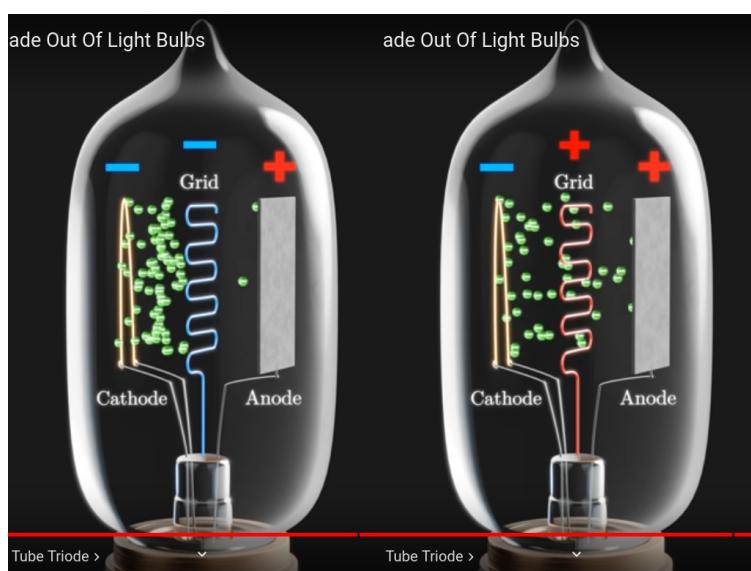


Figura 11.

Este invento, inspirado por el *efecto Edison*, se conoció como el triodo y fue el avance clave que dio origen a la electrónica moderna. La versión original fue patentada en 1904 por John Ambrose Fleming y perfeccionada en 1906 por Lee de Forest. En el triodo, un electrodo en forma de rejilla o espiral se coloca entre el cátodo y el ánodo, de ahí su nombre.

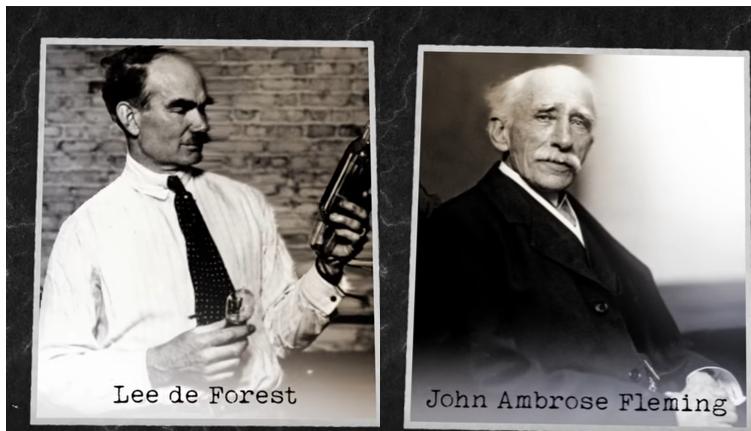


Figura 12.

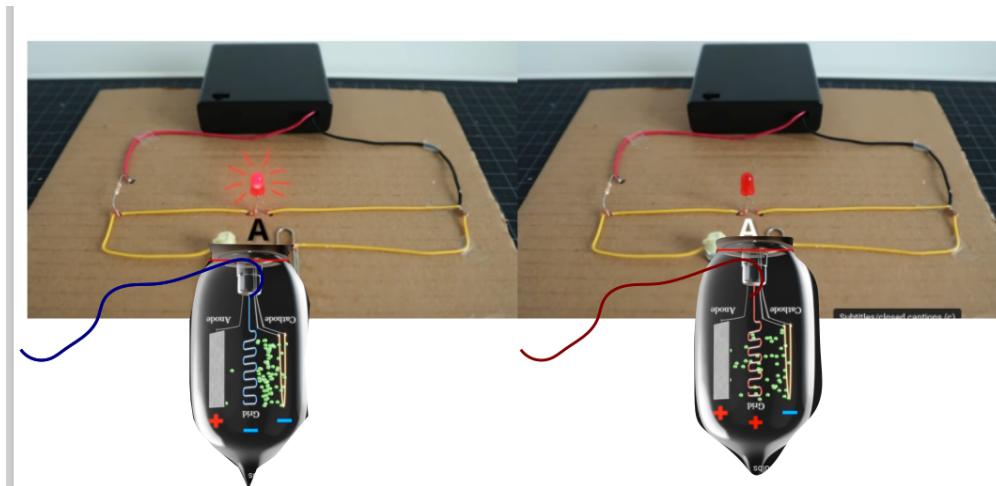
Un potencial puede aplicarse entre el ánodo y el cátodo, pero el número de electrones que fluye entre ellos es controlado por el voltaje en la rejilla central. Si la rejilla se carga negativamente, repele a los electrones hacia el cátodo y cancela la corriente. Si, por el contrario, se carga positivamente, acelera a los electrones hacia el ánodo, permitiendo que un pequeño voltaje en la rejilla controle el flujo de corriente en el circuito, es decir, puede encender o apagar el circuito.

Este aspecto, el de poder controlar el flujo de electrones con un pequeño voltaje, implica que este pequeño voltaje puede estar asociado con las oscilaciones de una onda de radio o la señal de un teléfono. En estos casos, el triodo funciona como un amplificador de la señal. Este avance dio origen a las llamadas de larga distancia por teléfono, así como a radios y televisores, todos basados en tubos de vacío.

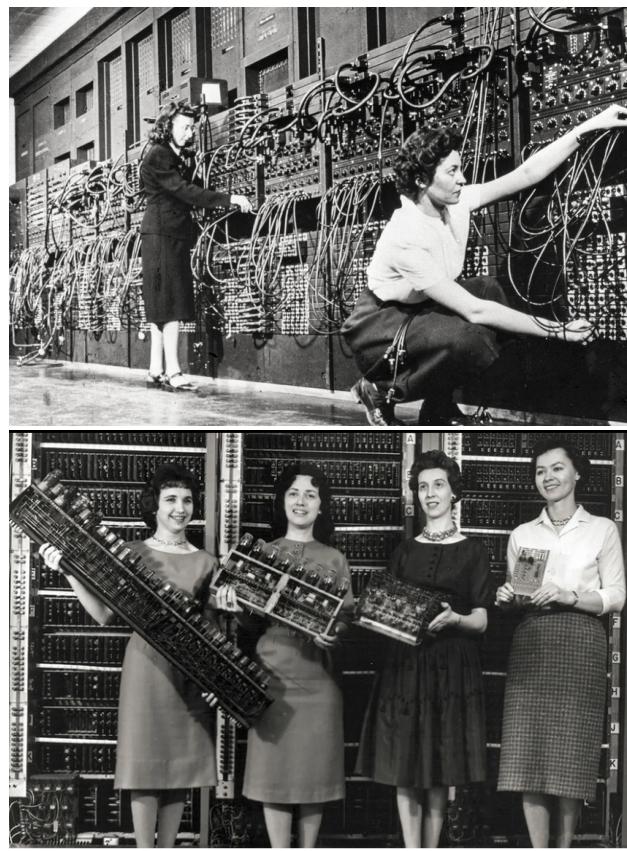


Figura 13.

Lo más interesante del triodo es que también funciona como un interruptor. Recordemos las compuertas NOT en la figura (mencionar figura anterior). El trabajo del interruptor puede ser realizado por un triodo al que se le aplica una pequeña corriente para abrir o cerrar el circuito, como se muestra en la figura.

**Figura 14.**

En otras palabras, ¡el triodo es un interruptor electrónico! Esto abrió la posibilidad de implementar las *Laws of Thought on Which are Founded the Mathematical Theories of Logic*, es decir, de realizar operaciones matemáticas utilizando álgebra booleana y circuitos eléctricos. Así fue como se creó el primer computador electrónico programable, el ENIAC (*Electronic Numerical Integrator And Computer*), que comenzó a operar por primera vez en diciembre de 1945.



Patsy Simmers, holding ENIAC board; Gail Taylor, holding EDVAC board; Milly Beck, holding ORDVAC board; and Norma Stec, holding BRLESC-1 board. | U.S. Army/ARL Technical Library Archives

Figura 15.

<https://www.digitaltrends.com/computing/remembering-eniac-and-the-women-who-programmed-it/>

3.1 El transistor

<https://www.youtube.com/watch?v=7ukDKVHnac4>

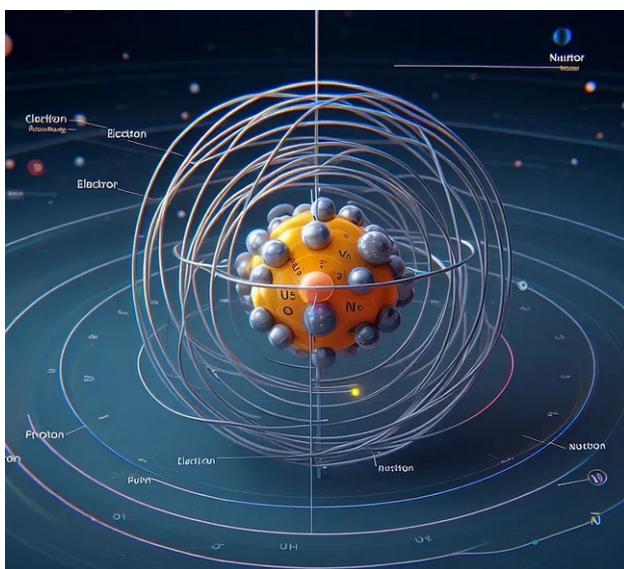
<https://www.youtube.com/watch?v=0wS9aTE2Go4&t=14s>

Es hora de profundizar un poco en la física que permite las herramientas que estamos estudiando:

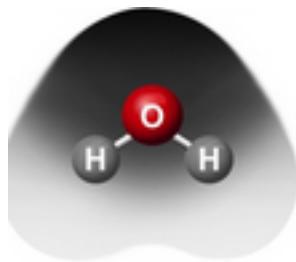
Esto es un átomo



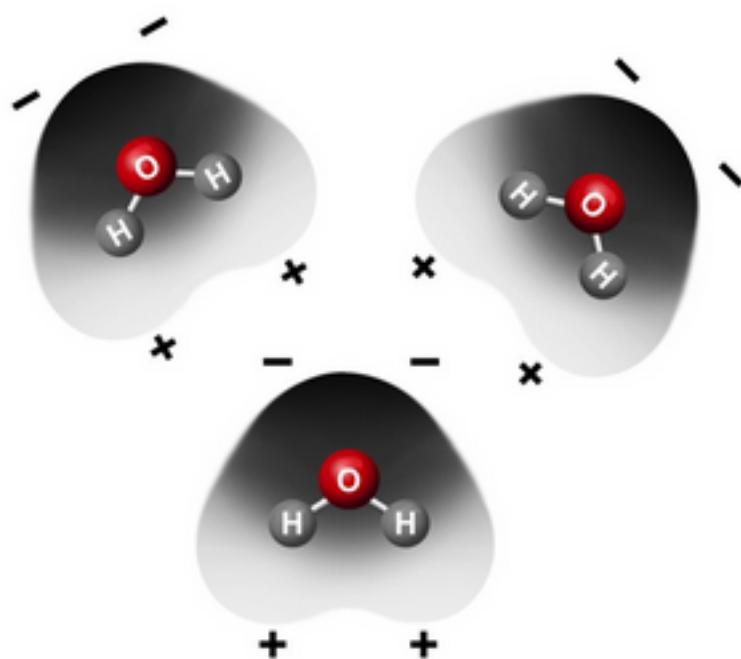
Este es el núcleo



Esta es una molécula de agua

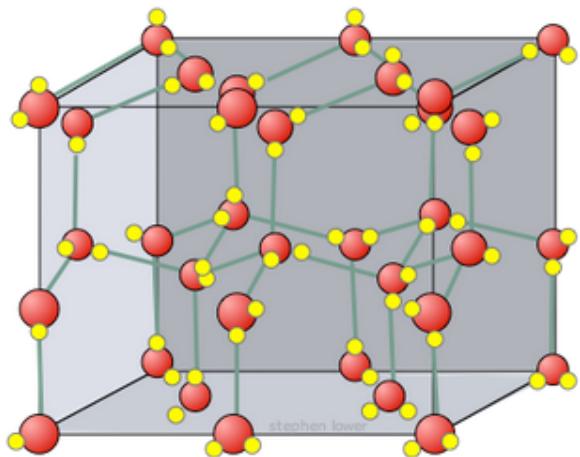


Y esto es agua líquida:



Este es un Pedazo de hielo



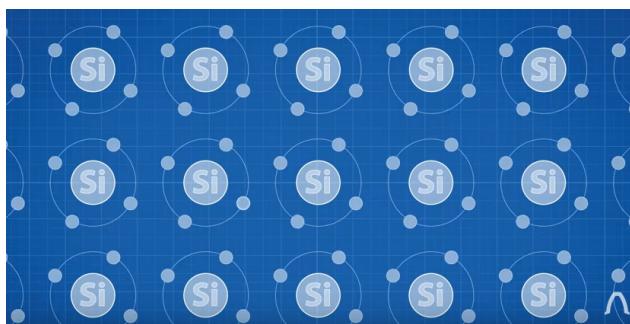


Este es un átomo de Silicio

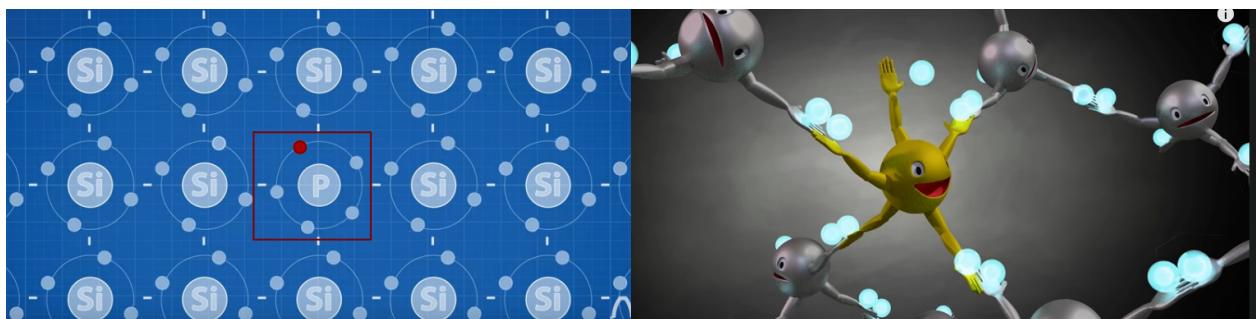


<https://www.youtube.com/watch?v=0wS9aTE2Go4&t=14s>

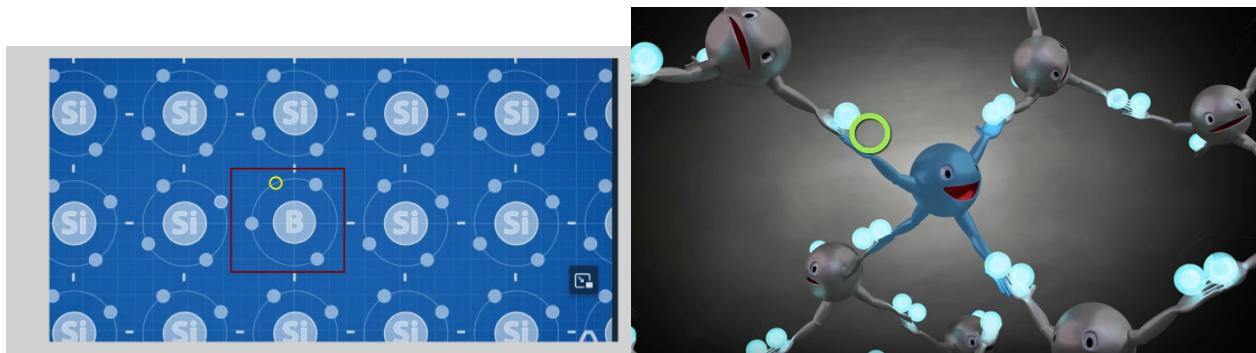
Esto es el un red cristalina de silicio



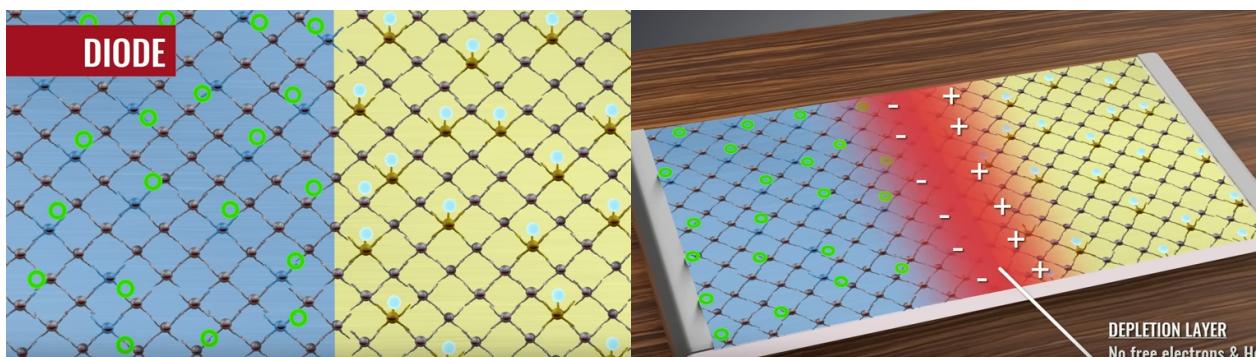
Silicio dopado con Fósforo, que tiene un electrón de valencia más:



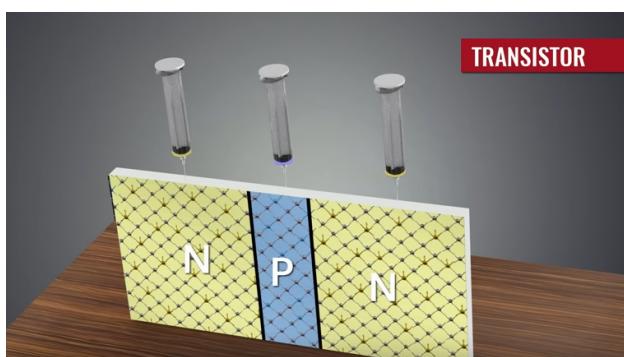
Silicio dopado con Boro, que tiene un electrón menos



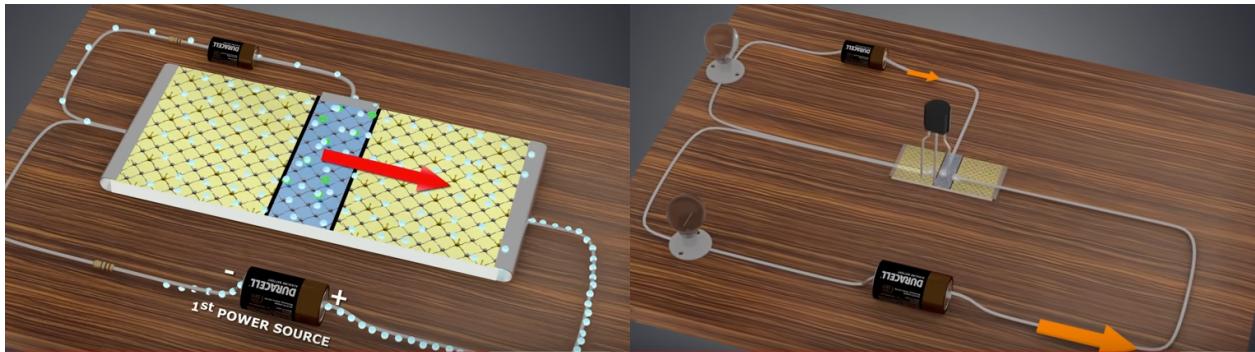
Diodo:



Transistor



Activado



Ahora te invito a que veas este interesante video sobre como los transistores leen código
<https://www.youtube.com/watch?v=HjneAhCy2N4>

3 Software

En la sesión anterior aprendimos que un transistor es básicamente un interruptor. En electrónica se representa por el símbolo

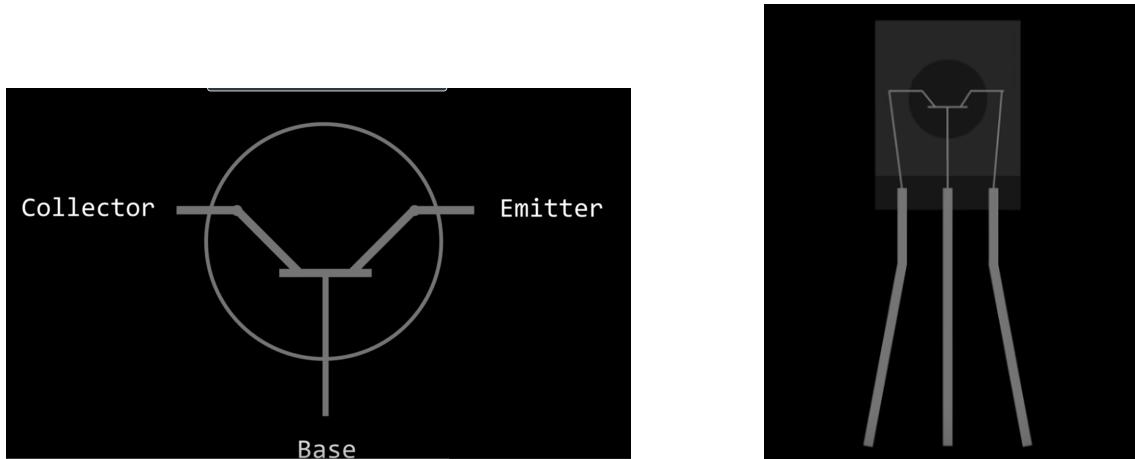


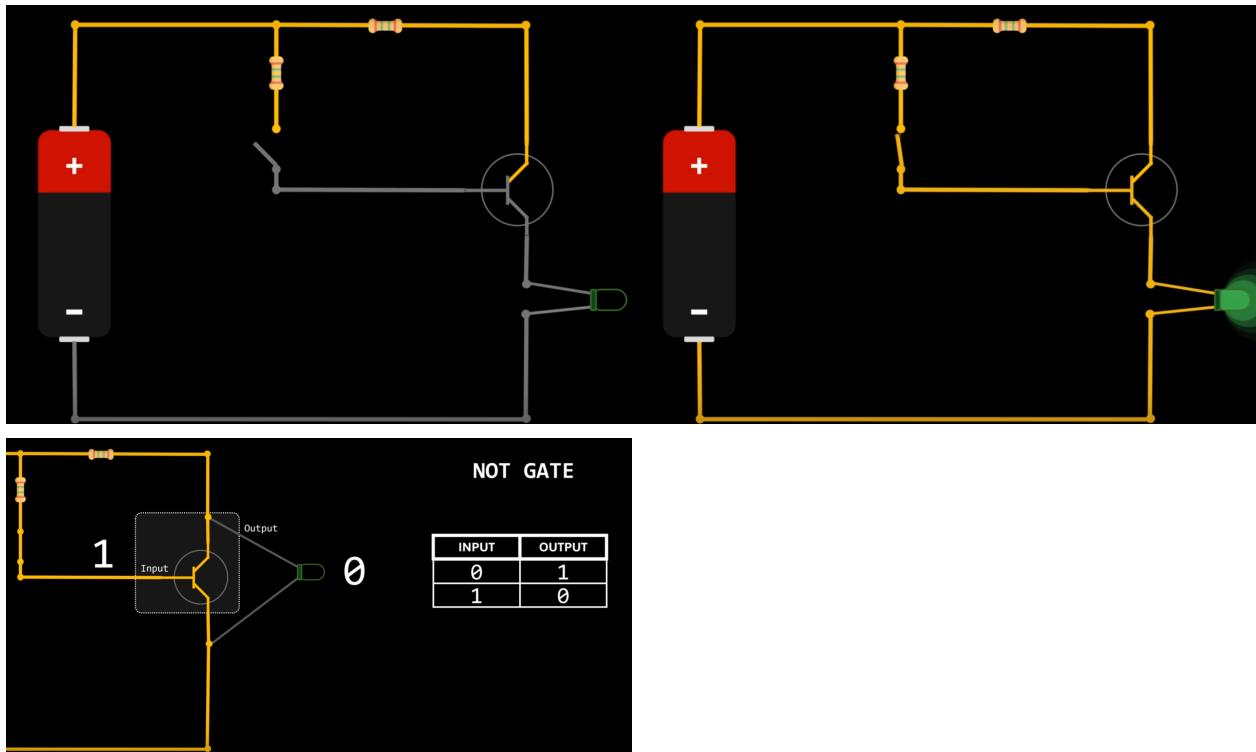
Figura 16.

Cuando no existe corriente en la base el transisor evita el paso de la corriente en el circuito, y una pequeña corriente en la base hace que el interruptor se abra:



Figura 17.

En el caso de la copuerta NOT tenemos el diagrama



La compuesta AND consiste en dos transistores en paralelo:

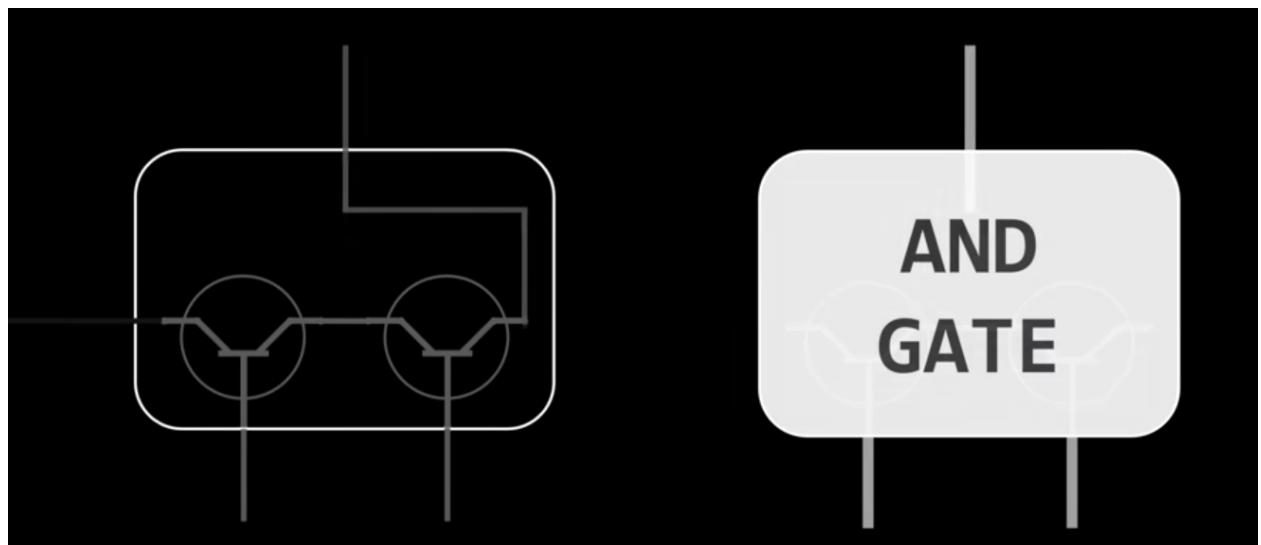
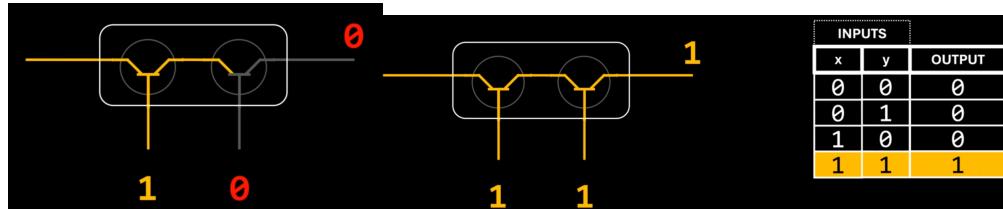


Figura 18.

Si conectamos los transistores en paralelo.

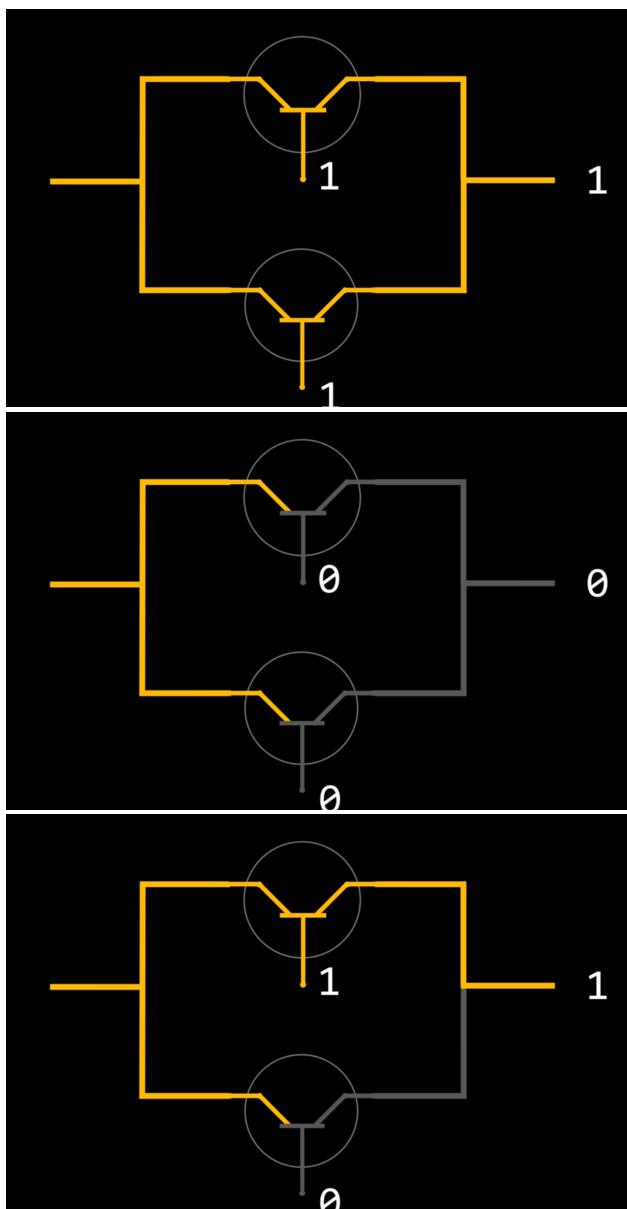


Figura 19.

Lo importante que podemos abstraer los circuitos a una caja. Por ejemplo:

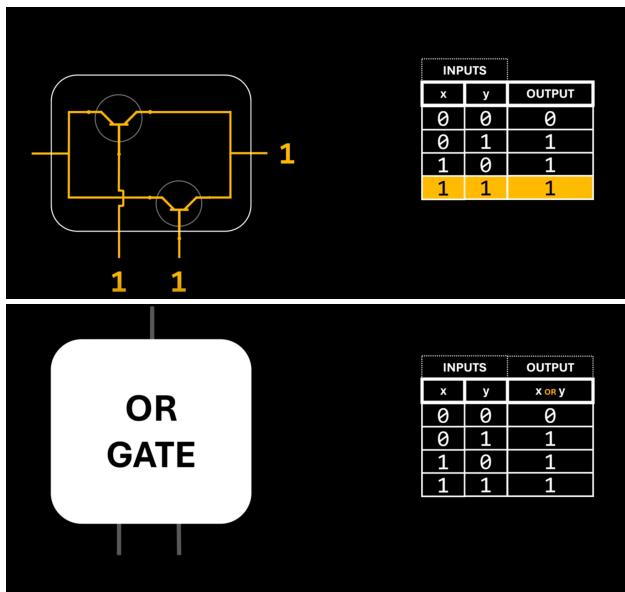


Figura 20.

Estas compuertas lógicas son tan fundamentales que tienen su propio símbolos

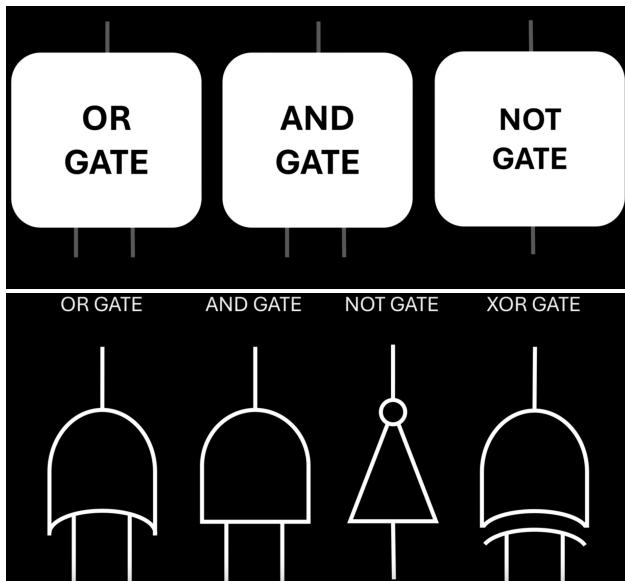


Figura 21.

Tambien vimos que usamos estas copuertas para construir comportamientos mas complejos. Por ejemplo un circuito que implemente la suma de dos bits, llamado un ADDER debe tener el comportamiento

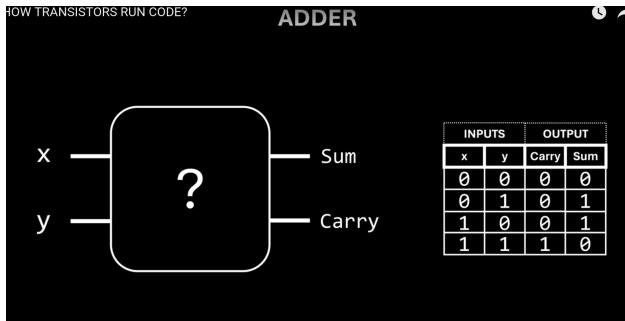


Figura 22.

Y vismo que este comportamiento es implementado por el circuito:

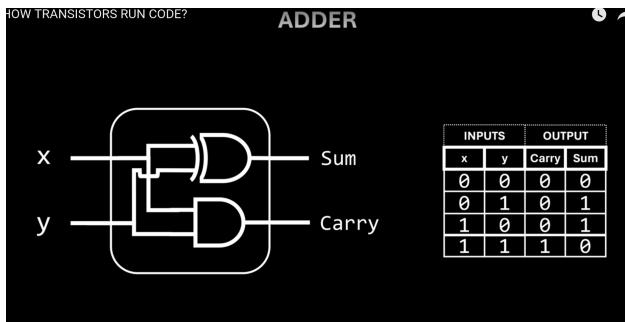


Figura 23.

Un sumadore de 3 bits es llamado un FULL ADDER y es implementado por el circuito

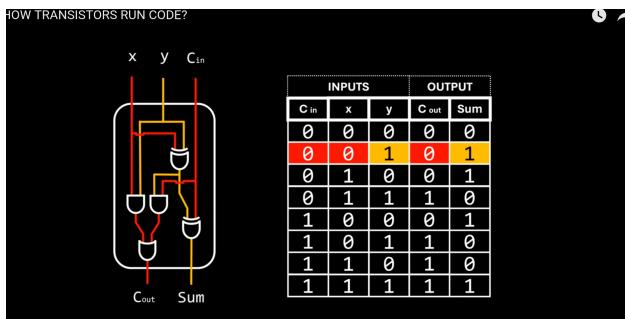


Figura 24.

La figura muestra el proceso de sumar 3 dígitos usando 3 FULL ADDERS

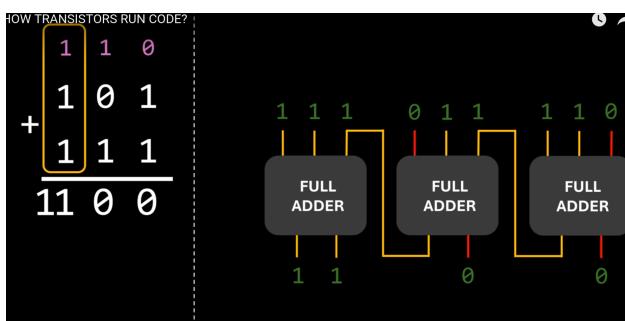


Figura 25.

Mientras que para 8 bits el diagrama necesitaría 8 FULL ADDERS y así sucesivamente

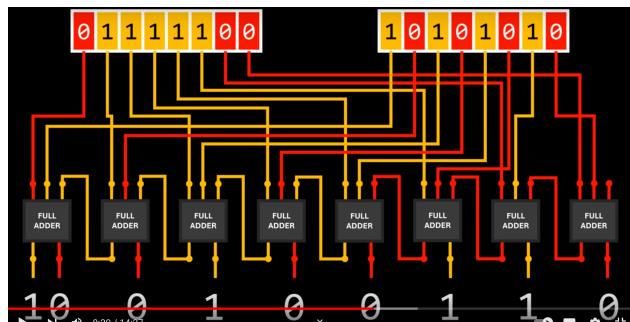


Figura 26.

Y nuevamente este circuito puede ser encapsulado en un solo componente conocido como el 8-BIT ADDER

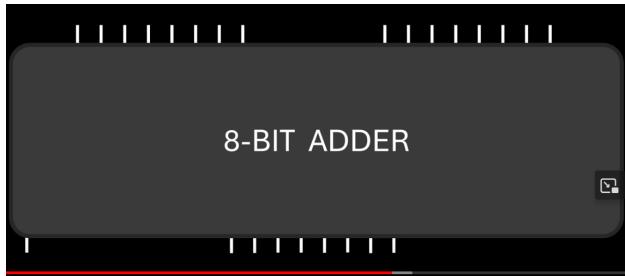


Figura 27.

A partir de estas mismas compuertas lógicas se pueden construir componentes con las instrucciones que entiende una CPU.

1 Lenguajes de programación

La figura muestra una unidad lógica de una CPU y el código en **lenguaje ensamblador** (o **assembly**). El lenguaje ensamblador es un lenguaje de bajo nivel que se utiliza para escribir instrucciones que son directamente ejecutables por la CPU. En este caso:

- LOAD [a] R0: Carga el valor almacenado en la dirección de memoria a en el registro R0.
- LOAD [b] R1: Carga el valor almacenado en la dirección de memoria b en el registro R1.
- ADD R0 R1: Suma los valores de los registros R0 y R1, y usualmente almacena el resultado en uno de los registros, en este caso parece ser R0.
- STORE R0 [a]: Guarda el valor del registro R0 en la dirección de memoria a.

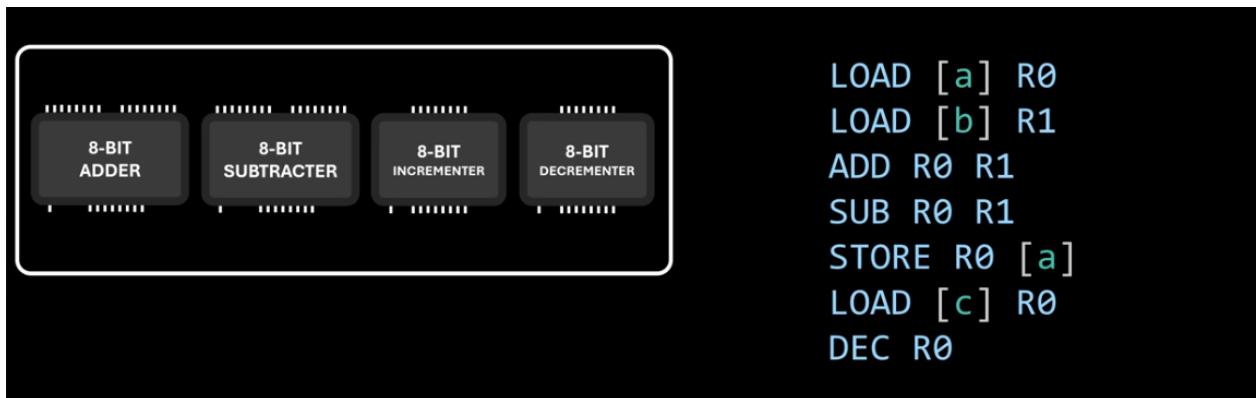


Figura 28.

```

LOAD [a] R0 ;Load value of memory address [a] in Register_0
LOAD [b] R1 ;Load value of memory address [b] in Register_1
ADD R0 R1 ;Add Register_0 with Register_1 and write the result in Register_0
SUB R0 R1 ;Subtract Register_0 with Register_1 and the write result in Register_0
STORE R0 [a] ;Store value in Register_0 in [a] memory address
LOAD [c] R0 ;Load value of [c] in Register_1
DEC R0 ;Decrement value in Register_0

```

Pero quién controla que parte del circuito ejecuta la instrucción?

1.1 Decodificadores Binarios

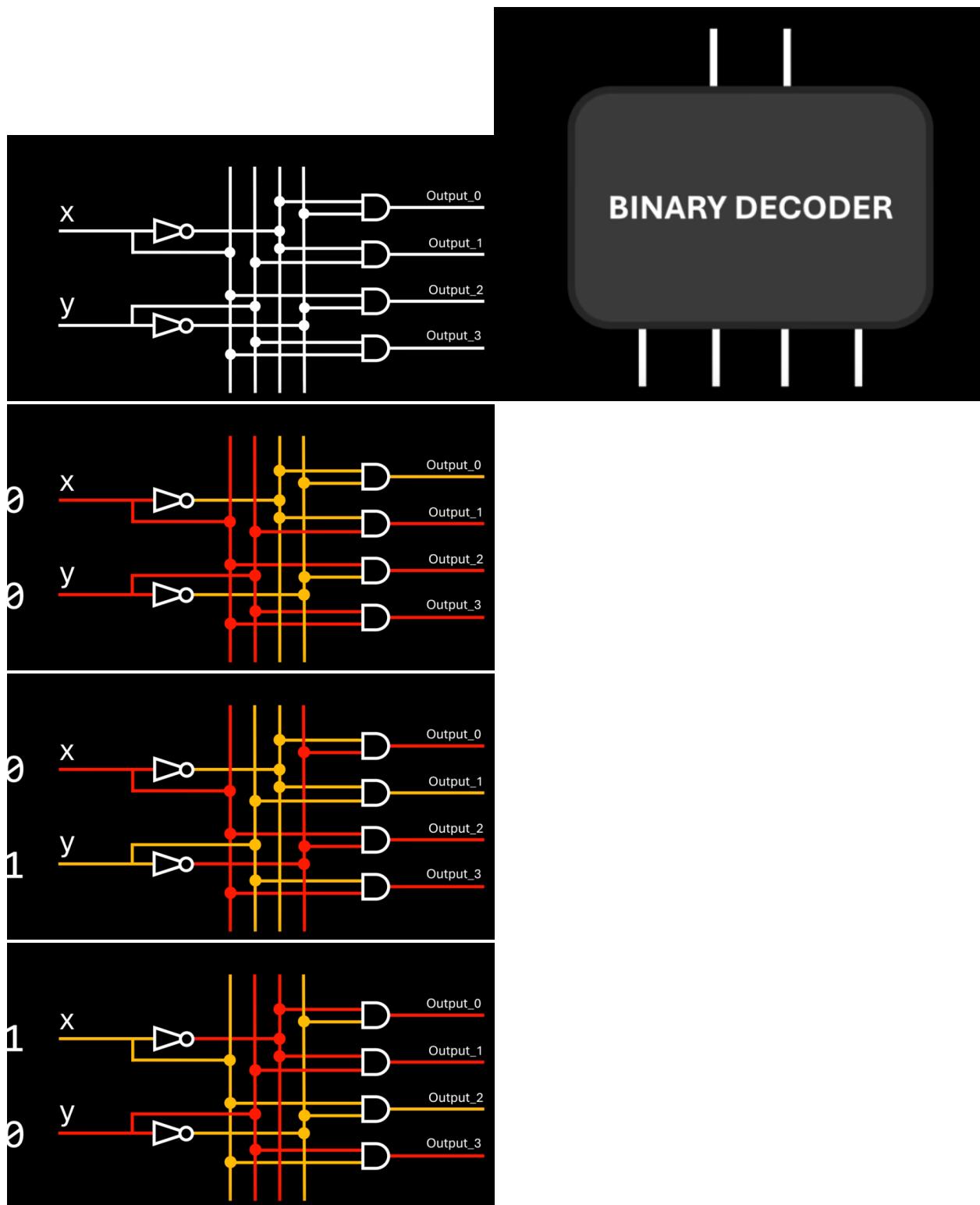


Figura 29.

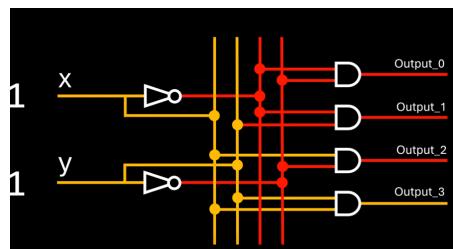


Figura 30.

Un **decodificador binario** (o **binary decoder**) es un circuito digital que convierte una entrada codificada en binario en una única salida activa. En otras palabras, toma un conjunto de bits de entrada y activa una de las muchas posibles salidas en función de esa entrada.

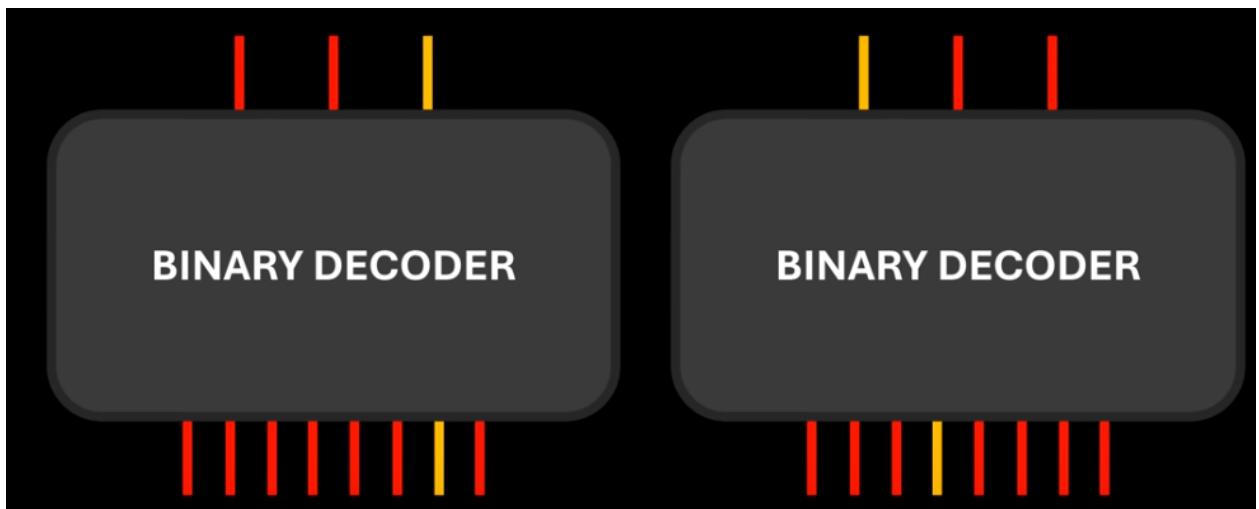


Figura 31.

En general con n inputs, podemos controlar 2^n outputs.

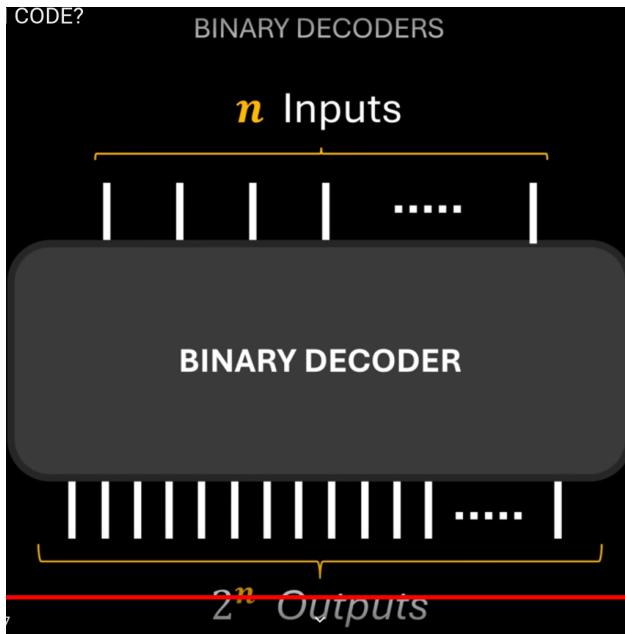


Figura 32.

Esta es la base del software, porque significa que podemos construir circuitos que nos permitan escoger entre múltiples opciones:

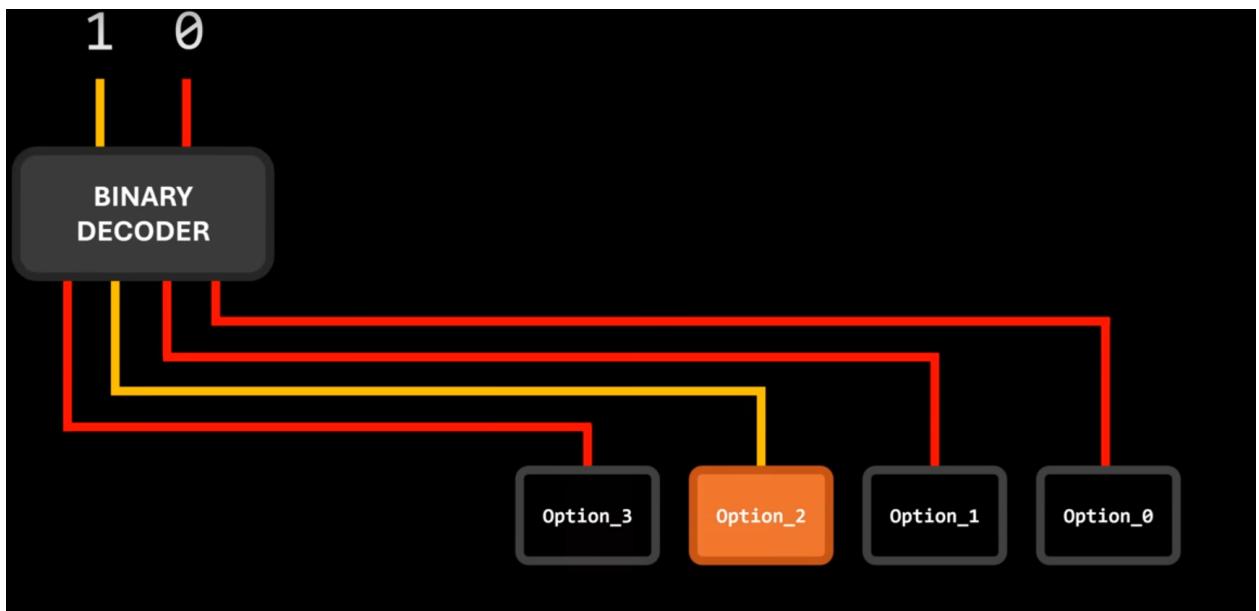
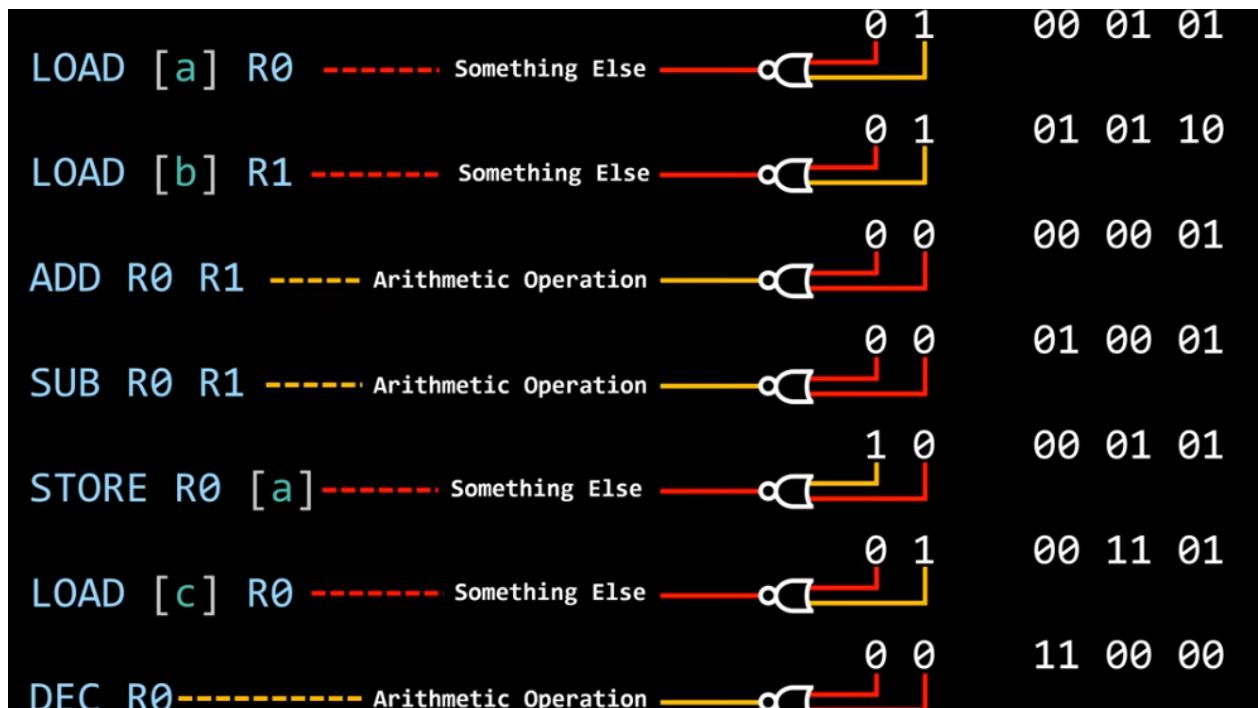


Figura 33.

De hecho el código assembly que acabamos de ver es simplemente una representación más legible de combinaciones de 1 y 0 para que el codificador binario:

Ejemplo:



Concentrémonos en operaciones aritméticas:

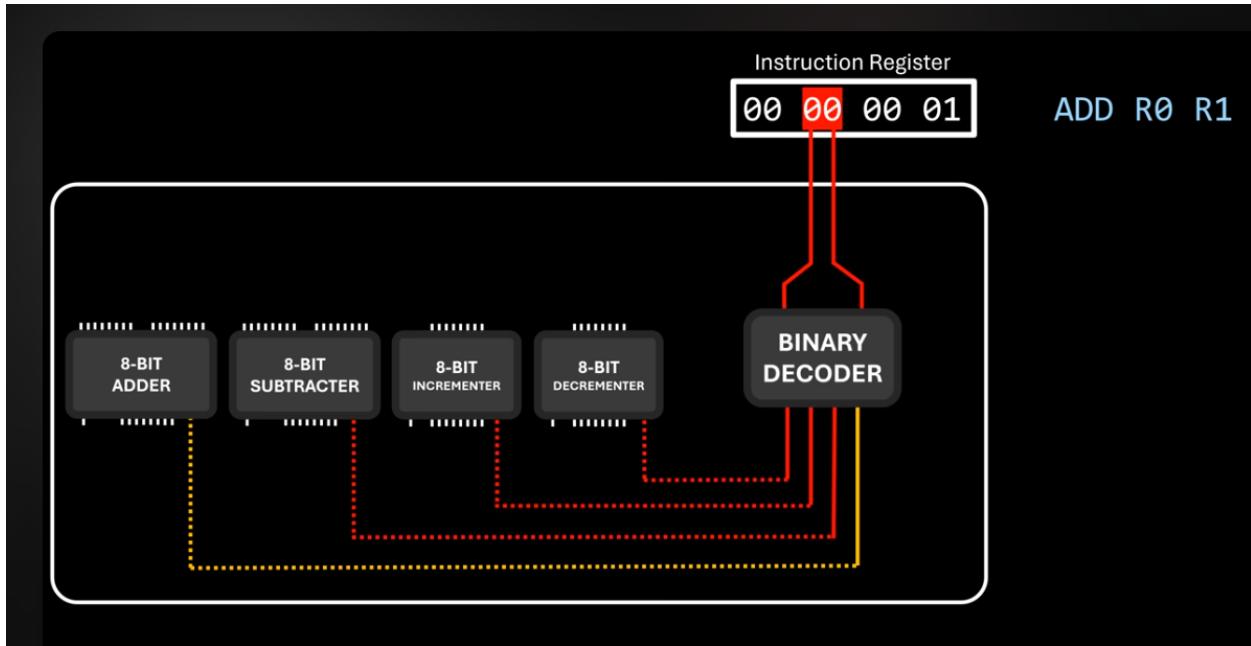
ADD R0 R1	00 00 00 01
SUB R0 R1	00 01 00 01
DEC R0	00 11 00 00

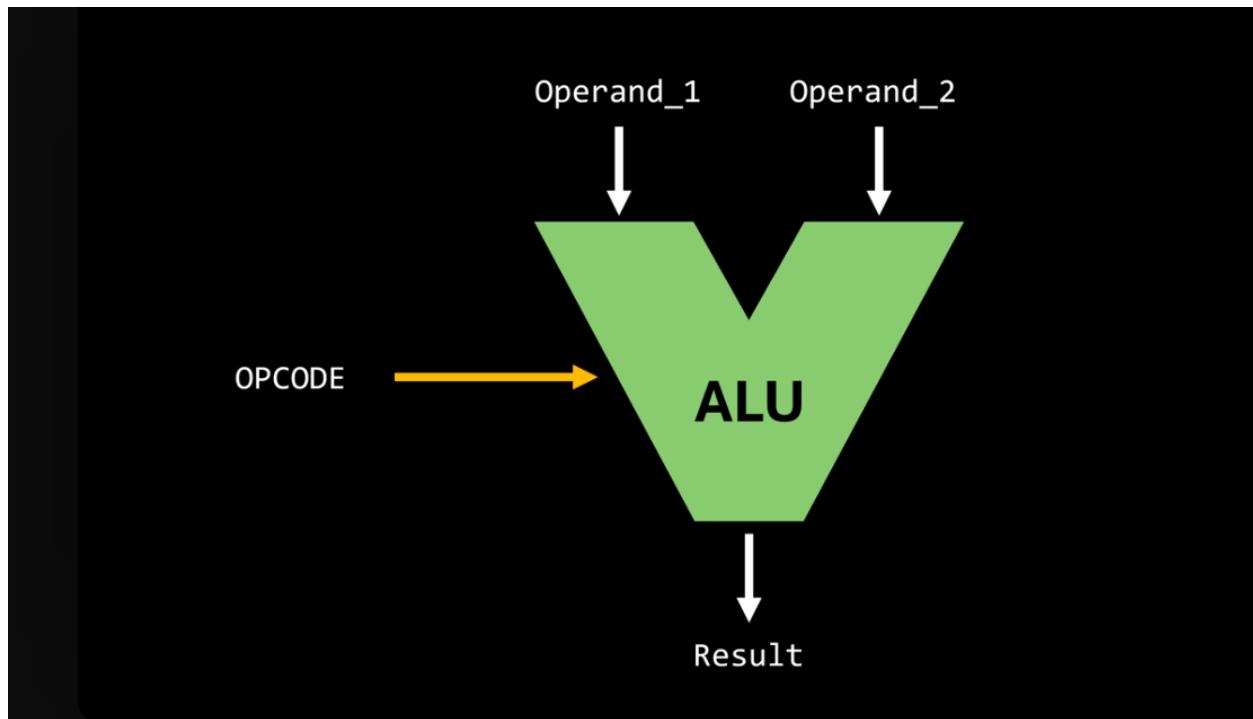
Figura 34.

ADD R0 R1	00	00	00 01
SUB R0 R1	00	01	00 01
DEC R0	00	11	00 00

ADD R0 R1	00	00	00	01
SUB R0 R1	00	01	00	01
DEC R0	00	11	00	00
ADD R0 R1	00	00	00	01
SUB R0 R1	00	01	00	01
DEC R0	00	11	00	00

OPCODE				
ADD R0 R1	00	00	00	01
SUB R0 R1	00	01	00	01
DEC R0	00	11	00	00





1.2 Breve historia del software

Los computadores que utilizaban tubos de vacío marcan el origen de los computadores digitales modernos, con la construcción del ENIAC en 1945. Simultáneamente, el matemático John von Neumann realizó importantes contribuciones que cambiaron la manera de entender cómo los computadores deben ser organizados y construidos. Basándose en los trabajos de Alan Turing, von Neumann mejoró los conceptos de memoria y direcciones en los computadores. Supervisó la construcción del sucesor del ENIAC, el EDVAC, completado en 1950, que se convirtió en el primer computador capaz de almacenar instrucciones y programas en su memoria.



Figura 35.

Este hito marcó el comienzo de la evolución de las ciencias de la computación como un campo de estudio independiente, dejando de ser solo una rama de las matemáticas, la física o la ingeniería pues pasamos de computadores mecánicos, como el ábaco, a computadores digitales basados en tubos de vacío, que podían realizar cálculos en microsegundos y almacenar instrucciones y programas. En 1949, los Laboratorios Bell lograron un avance significativo con la construcción del primer transistor de silicio. Posteriormente, en 1954, se construyó el primer computador basado en transistores, conocido como el **TX-0**. Este cambio de los tubos de vacío a los transistores permitió a los computadores volverse más rápidos, eficientes y pequeños.

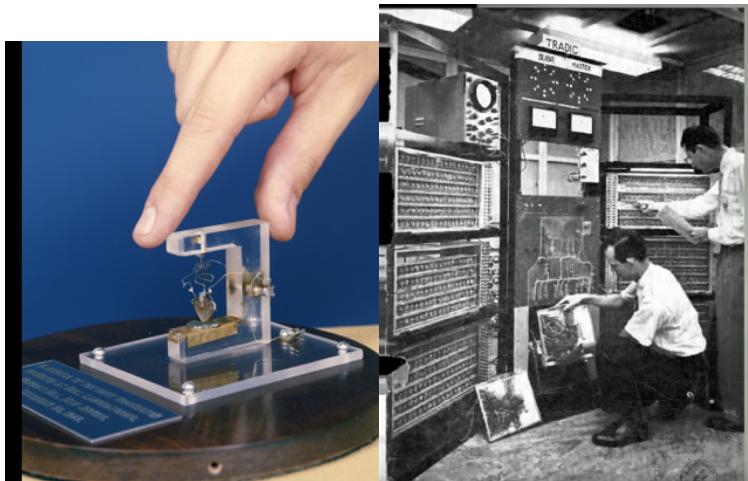


Figura 36.

Es en esta época cuando comenzaron a aparecer los primeros tratados sobre programación de computadores. Entre ellos destaca el trabajo de Donald Knuth, autor de la serie de libros *The Art of Computer Programming*, que formó a las primeras generaciones de programadores modernos. Puedes escuchar su historia aquí.

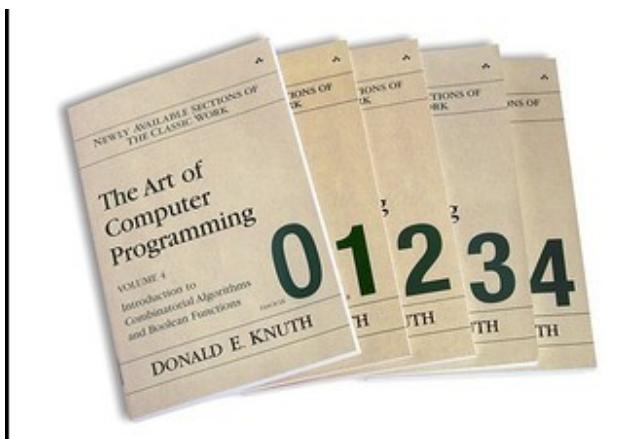


Figura 37.

El lenguaje ensamblador (Assembly) fue el primer lenguaje de programación introducido en 1949, permitiendo a los programadores comunicarse directamente con el hardware de los computadores a través de un código más amigable con el ser humano.

```

; Example of IBM PC assembly language
; Accepts a number in register AX;
; subtracts 32 if it is in the range 97-122;
; otherwise leaves it unchanged.

SUB32 PROC      ; procedure begins here
    CMP AX,97   ; compare AX to 97
    JL DONE     ; if less, jump to DONE
    CMP AX,122   ; compare AX to 122
    JG DONE     ; if greater, jump to DONE
    SUB AX,32   ; subtract 32 from AX
DONE: RET        ; return to main program
SUB32 ENDP      ; procedure ends here

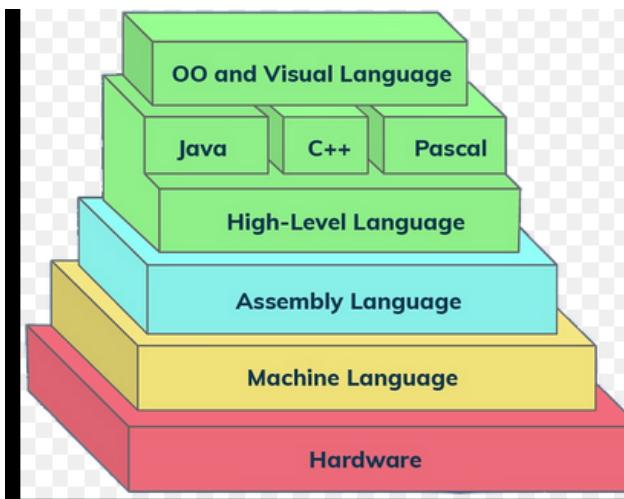
```

Figura 38.

- * **Assembly (1949):** Código cercano al hardware, eficiente para sistemas embebidos.
- * **FORTRAN (1957):** Cálculos científicos y de ingeniería.
- * **LISP (1958):** Pionero en inteligencia artificial y procesamiento simbólico.
- * **COBOL (1959):** Aplicaciones de negocios y sistemas de gestión.
- * **BASIC (1964):** Lenguaje accesible para principiantes.
- * **Pascal (1970):** Educación y desarrollo estructurado.
- * **C (1972):** Desarrollo de sistemas y aplicaciones de alto rendimiento.
- * **Prolog (1972):** Resolución de problemas lógicos y IA.
- * **Ada (1980):** Aplicaciones de misión crítica.
- * **C++ (1983):** Programación orientada a objetos y sistemas complejos.
- * **Perl (1987):** Procesamiento de texto y administración de sistemas.
- * **Python (1991):** Ciencia de datos, IA, desarrollo web.
- * **Java (1995):** Aplicaciones empresariales y móviles.
- * **JavaScript (1995):** Desarrollo web frontend.
- * **C# (2000):** Aplicaciones de escritorio, videojuegos, desarrollo empresarial.

2006 – Rust

Rust fue creado por Mozilla como un lenguaje de sistemas seguro y eficiente, diseñado para prevenir errores de memoria comunes en lenguajes como C y C++, a través de un sistema de gestión de memoria sin recolección de basura.



1.3 Git

Git es un sistema de control de versiones distribuido que permite gestionar el código fuente y colaborar en proyectos de manera eficiente. A través de comandos, Git rastrea los cambios, almacena versiones anteriores y facilita la colaboración en equipo.

1.3.1 Instalación de Git

En Linux:

Para instalar Git en distribuciones basadas en Debian/Ubuntu, ejecuta:

```
sudo apt update
sudo apt install git
```

En Windows:

1. Descarga [Git para Windows](#).
2. Ejecuta el instalador y sigue las instrucciones.
3. Abre “Git Bash” para usar Git en una terminal similar a Linux.

Configurar Git

Una vez instalado, debes configurar tu nombre de usuario y correo electrónico globalmente (estos aparecerán en cada commit).

```
git config --global user.name "Tu Nombre"
git config --global user.email "tuemail@ejemplo.com"
```

Inicializar un repositorio

Un repositorio es donde Git almacena tus archivos y sus versiones.

Crear un nuevo repositorio:

Para inicializar un repositorio de Git en un directorio existente, usa:

Inicializar un repositorio

Un repositorio es donde Git almacena tus archivos y sus versiones.

Crear un nuevo repositorio:

Para inicializar un repositorio de Git en un directorio existente, usa:

```
git init
```

Esto creará un subdirectorio .git en la carpeta actual, donde Git mantendrá el historial.

Preparar archivos: **git add** y la área de “staging”

Git usa una “área de staging” para preparar archivos antes de comprometerlos al historial del repositorio.

Agregar archivos al área de staging:

```
git add archivo.txt
```

O, para agregar todos los archivos modificados:

```
git add .
```

El comando `git add` toma una instantánea de los archivos y los mueve a la zona de “staging”. Estos archivos están listos para ser confirmados (commit), pero no han sido guardados permanentemente en el historial.

Guardar cambios: **git commit**

Una vez que los archivos están en la “staging”, debes confirmarlos con `git commit`. Esto guarda una versión permanente de los cambios en el historial de Git.

```
git commit -m "Mensaje que describe los cambios"
```

Branches (Ramas) y Cambiar entre ellas

Git permite trabajar en diferentes ramas para mantener un desarrollo organizado, facilitando la creación de características nuevas o corrección de errores sin afectar el código principal.

```
git checkout -b nueva-rama
```

Esto crea una nueva rama llamada `nueva-rama` y automáticamente te cambia a esa rama.

Si ya tienes una rama existente, puedes cambiarte a ella:

```
git checkout nombre-de-la-rama
```

Fusionar Branches: **git merge**

Cuando terminas de trabajar en una rama, puedes fusionarla con otra (normalmente con la rama principal `main` o `master`).

Por ejemplo, cambiamos a la rama `main`

```
git checkout main
```

y una vez allí podemos fusionar el contenido de alguna de las otras ramas:

```
git merge nombre-de-la-rama
```

Esto combinará los cambios de la `nombre-de-la-rama` en la rama actual.

Recuerda que puedes usar `git status` para ver el estado actual de tu repositorio, mostrándote qué archivos han sido modificados, cuáles están listos para ser confirmados (en el área de “staging”) y cuáles no se están rastreando aún.

1.4 GitHub

Git, como ya hemos visto, es una herramienta fundamental para el control de versiones de software, permitiendo a los desarrolladores gestionar cambios en sus proyectos de manera local y colaborativa. Sin embargo, cuando trabajamos en equipo o queremos almacenar nuestros proyectos en la nube, **GitHub** entra en escena como la extensión perfecta de Git.

1.4.1 Estableciendo la conexión

Una **llave pública SSH** es un componente de la criptografía de clave pública que se utiliza en el protocolo **SSH** (Secure Shell) para autenticar de manera segura la comunicación entre dos sistemas, generalmente para conectarse a un servidor remoto sin necesidad de introducir una contraseña en cada ocasión. Lo primero que haremos será generar una llave en nuestro computador:

Ejecuta el comando para generar la llave:

```
ssh-keygen -t ed25519 -C "tuemail@ejemplo.com"
```

Este comando genera una nueva llave SSH utilizando el algoritmo `ed25519` (recomendado). Asegúrate de reemplazar `tuemail@ejemplo.com` con el correo electrónico asociado a tu cuenta de GitHub.

Cuando se te pregunte dónde guardar la llave, presiona `Enter` para aceptar la ubicación predeterminada (`~/.ssh/id_ed25519`).

Te pedirá ingresar una frase de seguridad. Puedes dejarlo en blanco si no quieres una contraseña, pero agregar una frase de seguridad es una buena práctica de seguridad.

Añadir la llave SSH al agente SSH

El agente SSH se encarga de gestionar tus llaves SSH, permitiéndote no tener que ingresar la frase de seguridad cada vez que la uses.

Iniciar el agente SSH:

```
eval "$(ssh-agent -s)"
```

Añadir tu nueva llave SSH al agente:

```
ssh-add ~/.ssh/id_ed25519
```

Ahora, necesitas copiar la llave pública para añadirla a tu cuenta de GitHub.

Usa el siguiente comando para copiar el contenido de tu llave pública al portapapeles:

```
cat ~/.ssh/id_ed25519.pub
```

Esto mostrará el contenido de la llave pública en la terminal. Selecciona y copia todo el texto.

Añadir la llave SSH a GitHub

1. Inicia sesión en tu cuenta de GitHub.

2. Dirígete a **Settings** (Configuración) en la parte superior derecha.

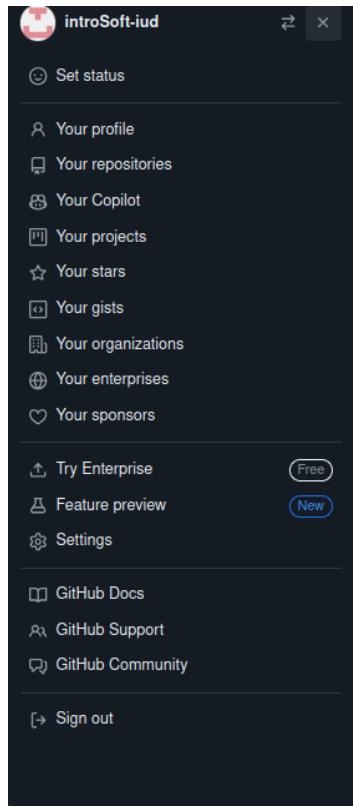


Figura 39.

3. En el menú de la izquierda, selecciona **SSH and GPG keys**.

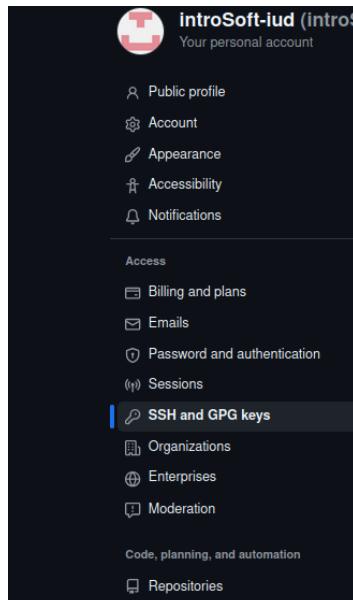


Figura 40.

4. Haz clic en el botón **New SSH Key**.



Figura 41.

5. Pega la llave pública que copiaste en el campo correspondiente y dale un nombre descriptivo (por ejemplo, "computador_vaio_casa")

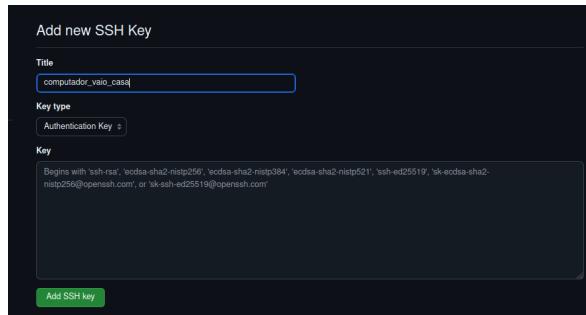


Figura 42.

6. Haz clic en **Add SSH Key**.

6. Probar la conexión

Una vez añadida la llave SSH a tu cuenta de GitHub, puedes verificar que todo está funcionando correctamente.

6.1 Ejecuta este comando para probar la conexión:

```
cat ~/.ssh/id_ed25519.pub
```

```
cat ~/.ssh/id_ed25519.pub
```

2 El flujo de trabajo Git Fork

El flujo de trabajo **Git Fork** es una estrategia comúnmente utilizada cuando varios desarrolladores colaboran en un proyecto de código abierto o en cualquier repositorio al que no tienen acceso directo para hacer cambios. Como lo aplicarás con tu grupo de trabajo para desarrollar la tarea 3, a continuación se explica cómo funciona:

2.1 Clonar un repositorio mediante un "fork"

El flujo de trabajo comienza con la creación de un **fork**. Un *fork* es una copia completa del repositorio que te permite trabajar de forma independiente en tu propia cuenta de GitHub (u otra plataforma Git). Este paso es esencial cuando no tienes permisos para hacer *push* directamente al repositorio principal (generalmente llamado el **repositorio upstream**).

Ejemplo: Si encuentras un proyecto interesante en GitHub pero no tienes permisos para modificarlo, puedes hacer un *fork*. Esto crea una copia del repositorio en tu cuenta.

2.2 Clonar tu repositorio fork a tu máquina local

Una vez que has hecho un *fork* del proyecto en tu cuenta, clonas tu copia del repositorio a tu máquina local para empezar a trabajar en los cambios.

```
git clone https://github.com/tu-usuario/tu-fork-del-proyecto.git
cd tu-fork-del-proyecto
```

2.3 Crear una rama para los cambios

Antes de comenzar a modificar el código, es una buena práctica crear una nueva rama para trabajar en los cambios. Esto ayuda a mantener el repositorio organizado y a evitar modificar directamente la rama principal (*main* o *master*).

```
git checkout -b mi-nueva-rama
```

2.4 Hacer cambios y commits

Realizas los cambios necesarios en tu copia local del repositorio. Después de realizar los cambios, puedes agregar y hacer *commits*.

```
git add .
git commit -m "Descripción de los cambios"
```

2.5 Subir los cambios a tu fork en GitHub

Los cambios se suben a tu repositorio *fork* en GitHub o GitLab (o la plataforma de Git que estés utilizando).

```
git push origin mi-nueva-rama
```

2.6 Crear un "Pull Request" (PR)

Una vez que los cambios están en tu repositorio *fork* en GitHub, puedes solicitar que se integren al repositorio original (*upstream*). Esto se hace mediante un **Pull Request** (PR). En el PR, los mantenedores del proyecto pueden revisar tus cambios y decidir si los aceptan o no.

Pull Request: Es una solicitud para que los mantenedores del repositorio original revisen y fusionen los cambios propuestos desde tu *fork*.

Ejemplo: Si has corregido un error o añadido una nueva funcionalidad al proyecto original, puedes enviar un PR para que esos cambios sean revisados y potencialmente aceptados en el código base.

2.7 Sincronizar tu fork con el repositorio original (*upstream*)

A medida que el repositorio original sigue evolucionando, debes mantener tu *fork* actualizado. Para ello, primero necesitas agregar el repositorio original como un **remoto** (usualmente llamado *upstream*).

```
git remote add upstream https://github.com/usuario-del-proyecto-original/proyecto-original.git
```

Luego, puedes obtener los cambios del repositorio original y fusionarlos con tu copia local:

```
git fetch upstream
git merge upstream/main # o 'master', según la rama que uses
```

4 Datos

1 La internet

<https://www.youtube.com/watch?v=x3c1ih2NJEg>

Cable coaxial

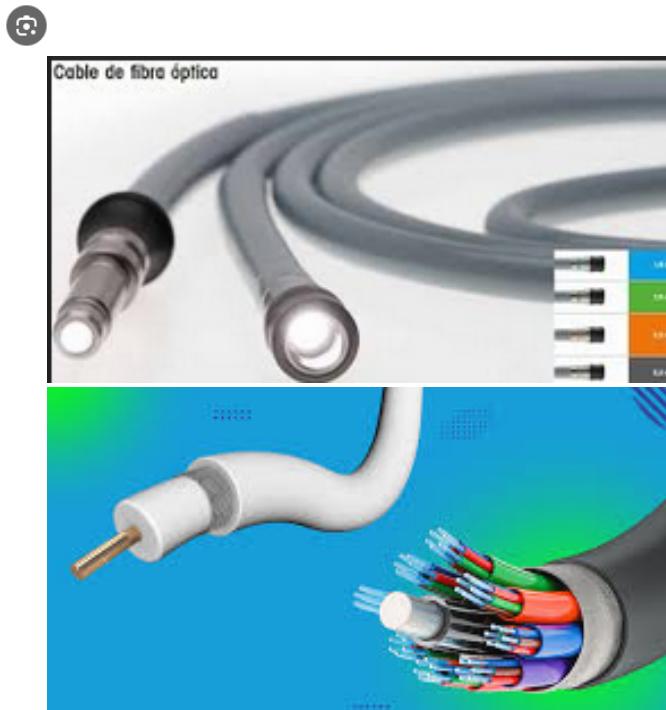


Figura 43.

1.1 Fibra óptica

1.2 Cómo los datos son transmitidos

1.3 Cómo lo datos son encriptados

1.3.1 El Algoritmo de Diffie-Hellman

En 1976 Withfield Hiffie y Martin Hellman desarrollaron las bases del algoritmo de encriptado de mensajes más usado actualmente. El problema puede ser descrito de la siguiente manera:

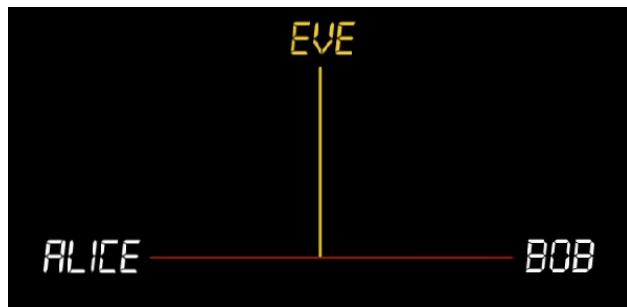


Figura 44.

Alice y Bob quieren compartir mensajes encriptados sin que eve descubra el mensaje. La solución es basada en las llamadas funciones de una sola vía:

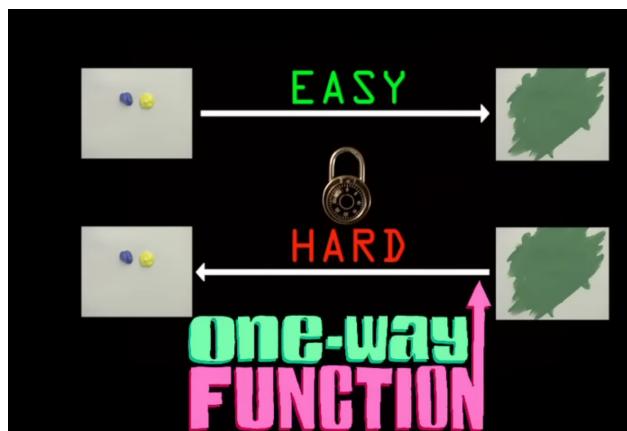


Figura 45.

La solución funciona de la siguiente manera:

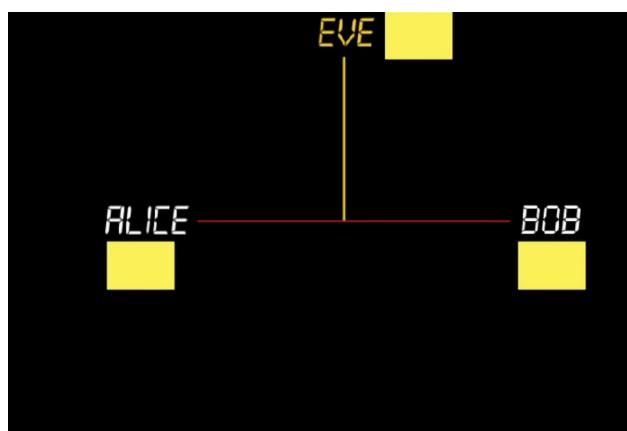
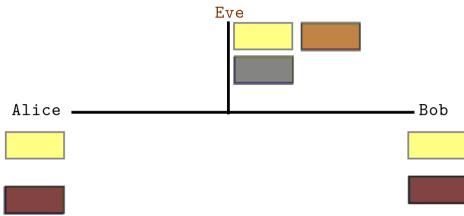


Figura 46.

**Figura 47.**

https://www.youtube.com/watch?v=YEBfamv_-do

1.3.2 Matemáticas del Algoritmo de Diffie-Hellman

La base matemática de este algoritmo de algoritmo de Diffie-Hellman radica en la exponenciación modular y en la dificultad de calcular logaritmos discretos.

Pensemos en el siguiente asertijo: ¿Cuanto vale a en

$$19^a \bmod 23 = 2?$$

¿Cuánto vale b en

$$5^b \bmod 23 = 19?$$

Formalmente el intercambio de claves públicas puede pensarse así?

Imaginemos dos partes, A y B , acuerdan públicamente dos números: una base g (conocida como generador) y un número primo grande p . A partir de ahí, cada parte selecciona un número secreto:

- A elige un número secreto a ,
- B elige un número secreto b .

Ambas partes ahora calculan un valor público para enviar a la otra parte:

$$A_{\text{pub}} = g^a \bmod p$$

$$B_{\text{pub}} = g^b \bmod p$$

Luego, A envía A_{pub} a B , y B envía B_{pub} a A . A partir de estos valores, cada parte puede calcular la clave compartida de la siguiente manera:

$$\text{Clave de } A = (B_{\text{pub}})^a \bmod p = (g^b)^a \bmod p = g^{ba} \bmod p$$

$$\text{Clave de } B = (A_{\text{pub}})^b \bmod p = (g^a)^b \bmod p = g^{ab} \bmod p$$

Es importante notar que $g^{ab} \bmod p = g^{ba} \bmod p$, por lo que ambas partes llegan al mismo valor final, que se convierte en su clave compartida.

Para entender mejor el paso clave, veamos más en detalle el cálculo:

1. A recibe $g^b \bmod p$ de B y lo eleva a la potencia a , resultando en $(g^b)^a \bmod p$.
2. Aplicando propiedades de los exponentes, sabemos que $(g^b)^a = g^{ba}$, lo que nos lleva a $g^{ba} \bmod p$.
3. B hace lo mismo, pero eleva $g^a \bmod p$ a la potencia b , lo que da $(g^a)^b = g^{ab} \bmod p$.

Como $g^{ab} = g^{ba}$, ambos obtienen el mismo resultado.

Ejemplo

Sean:

$$g = 5 \quad (\text{generador}), \quad p = 23 \quad (\text{número primo})$$

A elige el secreto $a = 6$

B elige el secreto $b = 15$

El primer paso es que cada parte calcula su valor público:

$$A_{\text{pub}} = g^a \bmod p = 5^6 \bmod 23 = 8$$

$$B_{\text{pub}} = g^b \bmod p = 5^{15} \bmod 23 = 19$$

A continuación, *A* y *B* intercambian estos valores públicos. Con los valores recibidos, cada parte calcula la clave compartida:

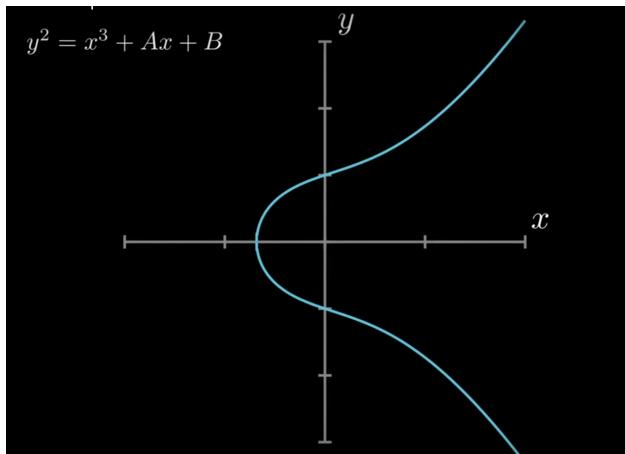
$$\text{Clave de } A = (B_{\text{pub}})^a \bmod p = 19^6 \bmod 23 = 2$$

$$\text{Clave de } B = (A_{\text{pub}})^b \bmod p = 8^{15} \bmod 23 = 2$$

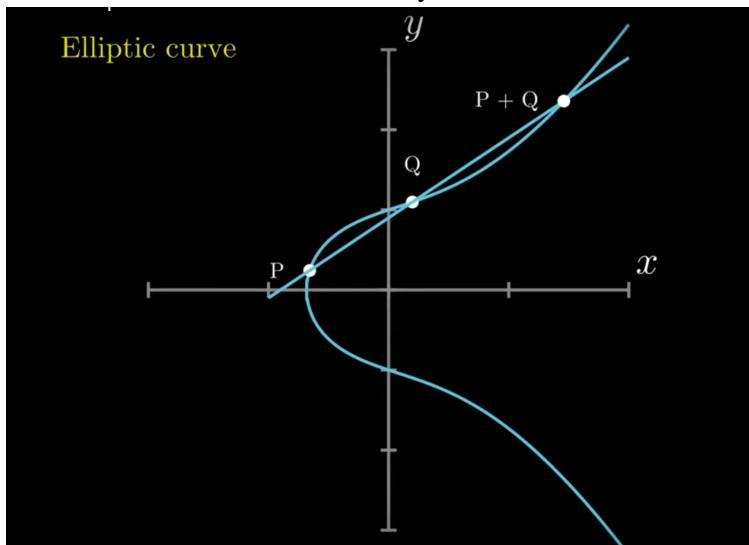
Así, ambas partes obtienen la misma clave compartida, que en este caso es 2.

1.3.3 Encriptación elíptica, blockchain y bitcoin

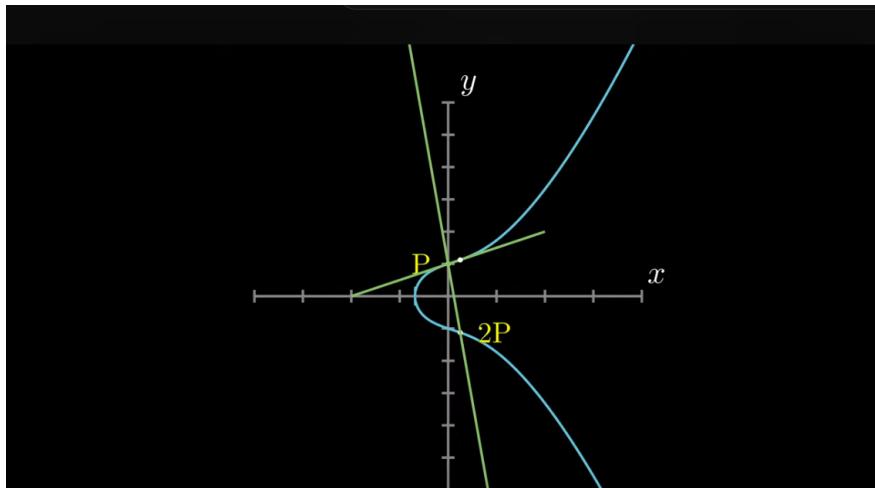
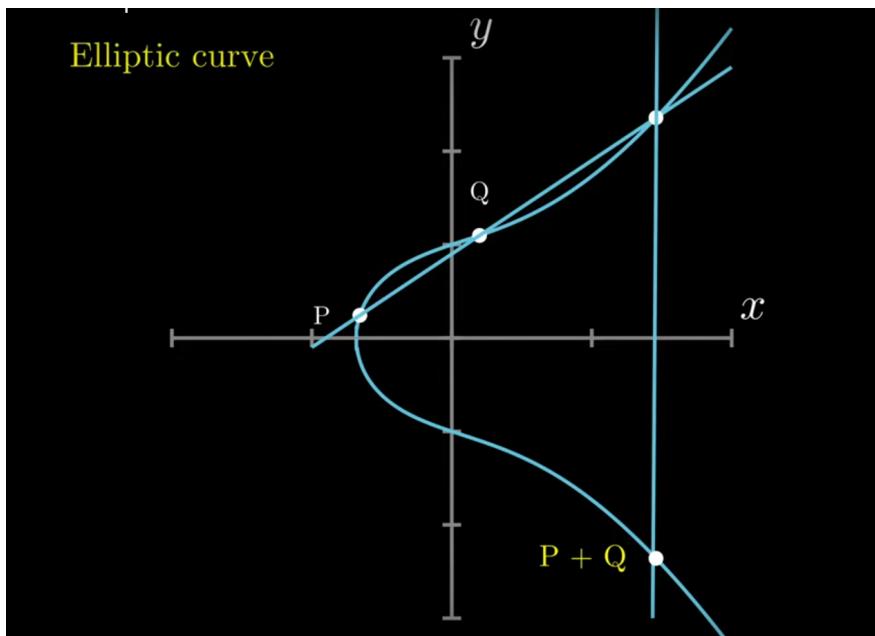
<https://www.youtube.com/watch?v=qCafMW40G7s&t=17s>



Se define D como la suma de $P + Q = D$



Y definimo la suma verdadera como la proyección





Elliptic curve cryptogrpahy

Take $P \in \mathbb{Q}^2$

$$P + P + P + \dots + P = n P = \left(\frac{26862913}{1493284}, \frac{139455877527}{1824793048} \right)$$

Private key	Public key
n	$\left(\frac{26862913}{1493284}, \frac{139455877527}{1824793048} \right)$

En criptografía de curvas elípticas, la suma de un punto consigo mismo se conoce como **duplicación de puntos**. Esta operación es fundamental para la multiplicación escalar de puntos, que es la base de la criptografía de curvas elípticas (ECC).

Para una curva elíptica dada sobre un campo finito, la ecuación general suele tener la forma de una ecuación de Weierstrass:

$$y^2 = x^3 + a x + b$$

donde a y b son constantes que definen la curva, y los puntos (x, y) están en la curva.

En la aritmética de curvas elípticas, para duplicar un punto $P = (x_1, y_1)$ en la curva (es decir, encontrar $P + P = 2P$), se sigue el siguiente procedimiento geométrico:

1. Se toma la línea tangente en el punto P (ya que se está sumando el punto consigo mismo).
2. Se encuentra la intersección de esta línea tangente con la curva.
3. El punto resultante de la intersección se refleja a través del eje x para obtener el resultado de la duplicación del punto.

Sea $P = (x_1, y_1)$ el punto en la curva elíptica que se desea duplicar. La fórmula para calcular $2P = (x_3, y_3)$ es la siguiente: Primero, se calcula la pendiente λ de la línea tangente en el punto P :

$$\lambda = \frac{3x_1^2 + a}{2y_1} \pmod{p}$$

Aquí, a es el coeficiente de la ecuación de la curva elíptica $y^2 = x^3 + ax + b$, y p es el número primo que define el campo finito (si se trabaja sobre un campo finito).

La coordenada x_3 del punto resultante $2P = (x_3, y_3)$ se calcula de la siguiente manera:

$$x_3 = \lambda^2 - 2x_1 \pmod{p}$$

Finalmente, la coordenada y_3 es:

$$y_3 = \lambda(x_1 - x_3) - y_1 \pmod{p}$$

Así, el punto resultante después de duplicar $P = (x_1, y_1)$ es $2P = (x_3, y_3)$.

Ejemplo

Supongamos que tenemos una curva elíptica definida por:

$$y^2 = x^3 + 2x + 3 \quad \text{sobre el campo finito } p = 97$$

Y que el punto $P = (3, 6)$ está en la curva. Queremos calcular $2P$.

Paso 1: Calcular la pendiente λ

$$\lambda = \frac{3(3^2) + 2}{2(6)} \pmod{97} = \frac{3(9) + 2}{12} \pmod{97} = \frac{29}{12} \pmod{97}$$

En lugar de dividir directamente, necesitamos encontrar el inverso modular de $12 \pmod{97}$, que es $12^{-1} \equiv 89 \pmod{97}$. Entonces:

$$\lambda = 29 \times 89 \pmod{97} = 2581 \pmod{97} = 60$$

Paso 2: Calcular la nueva coordenada x_3

$$x_3 = 60^2 - 2(3) \pmod{97} = 3600 - 6 \pmod{97} = 3594 \pmod{97} = 3$$

Paso 3: Calcular la nueva coordenada y_3

$$y_3 = 60(3 - 3) - 6 \pmod{97} = -6 \pmod{97} = 91$$

Por lo tanto, $2P = (3, 91)$.