El lenguaje oculto
del hardware y el software

CÓDIGO

SEGUNDA EDICIÓN

CHARLES PETZOLD

ANAYA MULTIMEDIA

Índice de contenidos

1 Mejores amigos

Tiene 10 años. Su mejor amigo vive al otro lado de la calle. De hecho, sus ventanas quedan una frente a otra. Todas las noches, cuando sus padres les han mandado a la cama a una hora demasiado temprana, todavía necesitan intercambiar pensamientos, observaciones, secretos, cotilleos, chistes y sueños. Nadie puede culparles. Al fin y al cabo, el impulso de comunicarse es uno de los rasgos más humanos.

Mientras las luces de sus dormitorios siguen encendidas, usted y su mejor amigo pueden saludarse con la mano desde la ventana y, usando gestos amplios y un lenguaje corporal rudimentario, transmitir un pensamiento o dos. Pero los intercambios más sofisticados parecen difíciles y, una vez que sus padres han dicho "¡apaga la luz!", se necesitan soluciones más sigilosas.

¿Cómo comunicarse? Si tienen la suerte de tener un teléfono móvil a los 10 años, quizá una llamada secreta o un mensaje silencioso funcione, pero ¿qué pasa si sus padres tienen la costumbre de confiscar teléfonos a la hora de dormir o incluso de quitar la Wi-Fi? Un dormitorio sin comunicación electrónica es una habitación muy aislada.

Lo que sí tienen usted y su mejor amigo, no obstante, son linternas. Todo el mundo sabe que las linternas se inventaron para permitir a los niños leer libros bajo las mantas; las linternas también parecen perfectas para la tarea de comunicarse en la oscuridad. Desde luego, son lo bastante silenciosas y la luz puede dirigirse bien, así que es poco probable que sus suspicaces padres la vean por debajo de la puerta.

¿Se puede lograr que las linternas hablen? Merece la pena probar. Aprendieron a escribir letras y palabras en papel en primaria, así que transferir ese conocimiento a la linterna parece razonable. Lo único que tienen que hacer es ponerse delante de la ventana y dibujar las letras con luz. Para una O, encienden la linterna, trazan un círculo en el aire y la apagan. Para una I, trazan una línea vertical. Pero enseguida descubren que este método es un desastre. Mientras mira la linterna de su amigo haciendo giros y líneas en el aire, se da cuenta de que es demasiado difícil unir los múltiples trazos en su cabeza. Estos giros y barras de luz no son lo bastante precisos.

Quizá una vez vio una película en la que un par de marineros se hacían señales a través del mar con luces parpadeantes. En otra película, un espía movía un espejo para reflejar la luz del sol en una habitación donde había otro espía cautivo. Quizá esa es la solución, así que primero diseñan una técnica simple. Cada letra del alfabeto se corresponde con una serie de parpadeos de la linterna. Una A es 1 parpadeo, una B son 2 parpadeos, una C son 3, y así hasta los 26 parpadeos de la Z. La palabra *BAD* son 2 parpadeos, 1 parpadeo y 4 parpadeos con pequeñas pausas entre las letras para que no se confundan con los 7 parpadeos de una G. Se hace una pausa un poco más larga entre palabras.

Esto parece prometedor. La buena noticia es que ya no tienen que agitar la linterna en el aire; solo hay que apuntar y encender. ¡La mala es que uno de los primeros mensajes que intenta enviar para saber qué tal está su amigo ("*How are you?*") requiere un total de 131 parpadeos! Además, se han olvidado de la puntuación, así que no sabe cuántos parpadeos corresponden al signo de interrogación.

Pero están cerca. Piensan que seguro que alguien habrá tenido que enfrentarse a este problema antes, y tienen toda la razón. Con un paseo a la biblioteca o una búsqueda en Internet, descubren un invento maravilloso conocido como código Morse. Es justo lo que estaban buscando, incluso aunque ahora tengan que volver a aprender a "escribir" todas las letras del alfabeto.

Aquí está la diferencia: en el sistema inventado por ustedes, cada letra del alfabeto es un número determinado de parpadeos, desde 1 parpadeo para la A a 26 parpadeos para la Z. En código Morse, hay dos tipos de parpadeos: cortos y largos. Por supuesto, esto hace que el código Morse sea más complicado, pero, en un uso real, resulta ser mucho más eficiente. La frase "*How are you?*" requiere solo 32 parpadeos (unos cortos y otros largos) en vez de 131, y eso incluye un código para el signo de interrogación.

Cuando se describe el funcionamiento del código Morse, la gente no habla de "parpadeos cortos" y "parpadeos largos", sino de "puntos" y "rayas", porque es una manera conveniente de mostrar los códigos en páginas impresas. En código Morse, cada letra del alfabeto se corresponde con una serie corta de puntos y rayas, como puede ver en la siguiente tabla.

A	•–
B	–•••
C	–•–•
D	–••
E	•
F	••–•
G	––•
H	••••
I	••

J	•–––
K	–•–
L	•–••
M	––
N	–•
O	–––
P	•––•
Q	––•–
R	•–•

S	•••
T	–
U	••–
V	•••–
W	•––
X	–••–
Y	–•––
Z	––••

Aunque el código Morse no tiene nada que ver con ordenadores, familiarizarse con la naturaleza de los códigos es un paso preliminar esencial para conseguir una comprensión profunda de los lenguajes ocultos y las estructuras internas del hardware y el software de un ordenador.

En este libro, la palabra "código" suele significar un sistema para transferir información entre personas, entre personas y ordenadores o dentro de los propios ordenadores.

Un código nos permite comunicarnos. A veces, los códigos son secretos, pero la mayoría no lo son. De hecho, la mayoría de los códigos deben entenderse bien porque son la base de la comunicación humana.

Los sonidos que hacemos con la boca para formar palabras constituyen un código que es inteligible para cualquiera que pueda oír nuestras voces y entender el idioma que hablamos. Llamamos a este código "palabra hablada" o "discurso oral".

Dentro de las comunidades de personas sordas, hay varios lenguajes de signos que emplean las manos y los brazos para formar movimientos y gestos que representan letras individuales de palabras o palabras y conceptos completos. Los dos sistemas más comunes en Norteamérica son el *American Sign Language* (ASL), que se desarrolló a principios del siglo XIX en la American School for the Deaf, y la *Langue des signes Québécoise* (LSQ), que es una variación del lenguaje de signos francés.

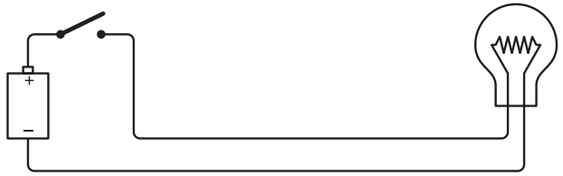
Usamos otro código para las palabras en papel u otros medios, llamado "palabra escrita" o "texto". El texto puede escribirse o teclearse a mano y, después, imprimirse en periódicos, revistas y libros, o mostrarse de forma digital en diversos tipos de dispositivos. En muchos idiomas, existe una fuerte correspondencia entre el discurso oral y el texto. En inglés, por ejemplo, las letras y los grupos de letras se corresponden (más o menos) con sonidos hablados.

Para las personas con una discapacidad visual, la palabra escrita puede sustituirse por Braille, que usa un sistema de puntos en relieve que se corresponden con letras, grupos de letras o palabras completas (hablaremos del Braille con más detalle en el capítulo 3).

Cuando las palabras habladas deben transcribirse a texto con mucha rapidez, la taquigrafía o estenografía resulta útil. En los tribunales o para generar subtítulos cerrados para noticias o retransmisiones deportivas en televisión, los taquígrafos usan un estenógrafo con un teclado simplificado que incorpora sus propios códigos que se corresponden con texto.

Usamos una variedad de códigos diferentes para comunicarnos entre nosotros porque algunos códigos son más convenientes que otros. El código de la palabra hablada no puede almacenarse en papel, así que se usa el código de la palabra escrita. Intercambiar información en silencio a través de cierta distancia en la oscuridad no es posible con un discurso oral o en papel. Por tanto, el código Morse en una alternativa conveniente. Un código es útil si sirve a un propósito al que no puede servir ningún otro código.

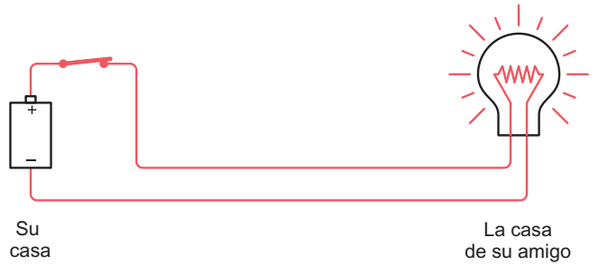
Como veremos, también se utilizan varios tipos de códigos en ordenadores para almacenar y comunicar texto, números, sonidos, música, fotografías y vídeos, además de las instrucciones dentro del propio ordenador. Los ordenadores no pueden tratar con facilidad códigos humanos



A partir de ahora, los circuitos se mostrarán de una forma más simbólica que realista. Aunque voy a enseñar solo una pila, en realidad podría estar usando dos. En este y en futuros diagramas, este será un interruptor apagado (o abierto):

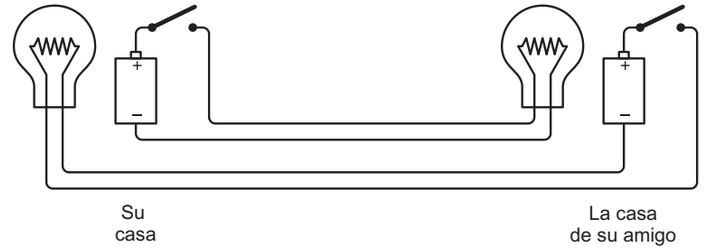
Y este será el interruptor cuando esté encendido (o cerrado):

La linterna de este capítulo funciona de la misma manera que la ilustrada en el capítulo anterior, salvo porque los cables que conectan los componentes ahora son un poco más largos. Cuando cierra el interruptor en su casa, la luz se enciende en la de su amigo:



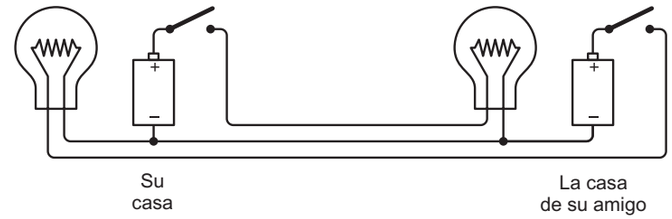
Ahora pueden enviarse mensajes usando código Morse.

Una vez que tengan una linterna que funcione, pueden conectar otra de larga distancia para que su amigo pueda enviarle mensajes a usted:



¡Enhorabuena! Acaban de montar un sistema de telégrafo bidireccional. Observará que se trata de dos circuitos idénticos totalmente independientes entre sí. En teoría, puede estar enviando un mensaje a su amigo mientras su amigo está enviándole un mensaje a usted, aunque podría ser difícil para su cerebro leer y enviar mensajes al mismo tiempo.

Puede que también sean lo bastante listos para darse cuenta de que no necesitan tantos cables que cubran la distancia entre las dos casas. Pueden eliminar uno de los cuatro cables si configuran la conexión de este modo:



En este libro, los cables que se conectan entre sí se simbolizan mediante un puntito en la conexión. Este diagrama tiene dos conexiones así, una debajo de la pila en su casa y otra debajo de la bombilla en la casa de su amigo.

Observe que los terminales negativos de las dos pilas están ahora conectados. Los dos circuitos circulares (pila a interruptor a bombilla a pila) todavía operan de forma independiente, incluso aunque ahora estén unidos.

Este tipo de conexión entre los dos circuitos se denomina común. En este circuito, el común se extiende entre los dos puntos de conexión de los cables, desde el punto donde la bombilla y la pila más a la izquierda se conectan hasta el punto donde la bombilla y la pila más a la derecha se conectan.

La I es un uno. Esto podía derivar de una raya vertical o de un solo dedo levantado. La V, que es posiblemente un símbolo para representar una mano, es un cinco. Dos V forman una X, que representa el diez. La L es cincuenta. La letra C viene de la palabra *centum*, que es cien en latín. La D es quinientos. Por último, la M viene de la palabra latina *mille*, que es mil. Con mil pasos de izquierda a derecha, recorrerá más o menos una milla.

Aunque puede que no estemos de acuerdo, durante mucho tiempo se consideró que los números romanos eran fáciles de sumar y restar, y por eso sobrevivieron tanto tiempo en Europa para la contabilidad. De hecho, para sumar dos números romanos, solo hay que combinar todos los símbolos de ambos números y, después, simplificar el resultado usando unas pocas reglas: cinco I hacen una V, dos V hacen una X, cinco X hacen una L, etcétera.

Pero multiplicar y dividir números romanos es difícil. Muchos otros sistemas numéricos primitivos (como el de los antiguos griegos) tampoco eran adecuados para trabajar con números de una manera sofisticada. Los griegos desarrollaron una geometría extraordinaria que todavía se enseña en los institutos actuales sin apenas cambios, pero no son famosos por su álgebra.

El sistema numérico que usamos hoy en día se conoce como indo-arábigo. Es de origen indio, pero fueron los matemáticos árabes quienes los trajeron a Europa. Es especialmente conocido el matemático persa Muhammed ibn Musa al-Khwarizmi (de cuyo nombre deriva la palabra algoritmo), que escribió un libro sobre álgebra en torno al año 820 e. c. que usaba el sistema hindú para contar. Hay una traducción al latín que data del 1145 e. c., aproximadamente, y que tuvo una gran influencia para acelerar la transición por toda Europa de los números romanos a nuestro sistema indo-arábigo actual.

Los números indo-arábigos se diferencian de los sistemas numéricos anteriores de tres maneras:

- Se dice que el sistema numérico indo-arábigo es posicional, lo que significa que un dígito concreto representa una cantidad diferente dependiendo de dónde se encuentre en el número. ¡Dónde aparece un dígito es, en realidad, mucho más significativo que los dígitos en sí! Tanto 100 como 1.000.000 tienen un solo 1, pero todos sabemos que un millón es mucho mayor que cien.
- Casi todos los sistemas numéricos primitivos tienen algo que el sistema indo-arábigo no tiene, que es un símbolo especial para el número diez. En nuestro sistema numérico, no hay símbolo especial para el diez.
- Por otra parte, a casi todos los sistemas numéricos primitivos les falta algo que el sistema indo-arábigo tiene, y que resulta ser mucho más importante que un símbolo para el diez. Estamos hablando del cero.

Sí, el cero. El humilde cero es, sin duda, una de las invenciones más importantes de la historia de los números y las matemáticas. Soporta la notación posicional porque nos permite ver de inmediato la diferencia entre 25 y 205 y 250. El cero también facilita muchas operaciones matemáticas que resultan incómodas en sistemas no posicionales, en particular la multiplicación y la división.

La estructura completa de los números indo-arábigos se revela en la manera en que los pronunciamos. Tomemos como ejemplo el 4.825. Decimos "cuatro mil ochocientos veinticinco". Eso significa:

cuatro millares
ocho centenas
dos decenas y
cinco.

O podemos escribir los componentes así:

$$4.825 = 4.000 + 800 + 20 + 5$$

O, para descomponerlo todavía más, podemos escribir el número de esta manera:

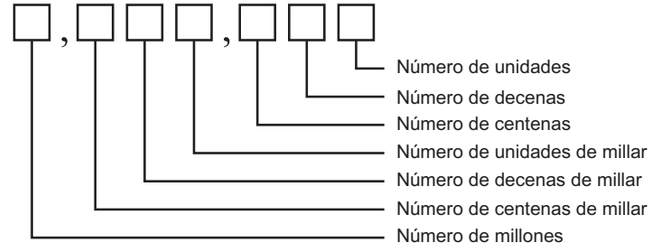
$$\begin{aligned} 4.825 = {} & 4 \times 1.000 + \\ & 8 \times 100 + \\ & 2 \times 10 + \\ & 5 \times 1 \end{aligned}$$

O, usando potencias de diez, el número puede escribirse así:

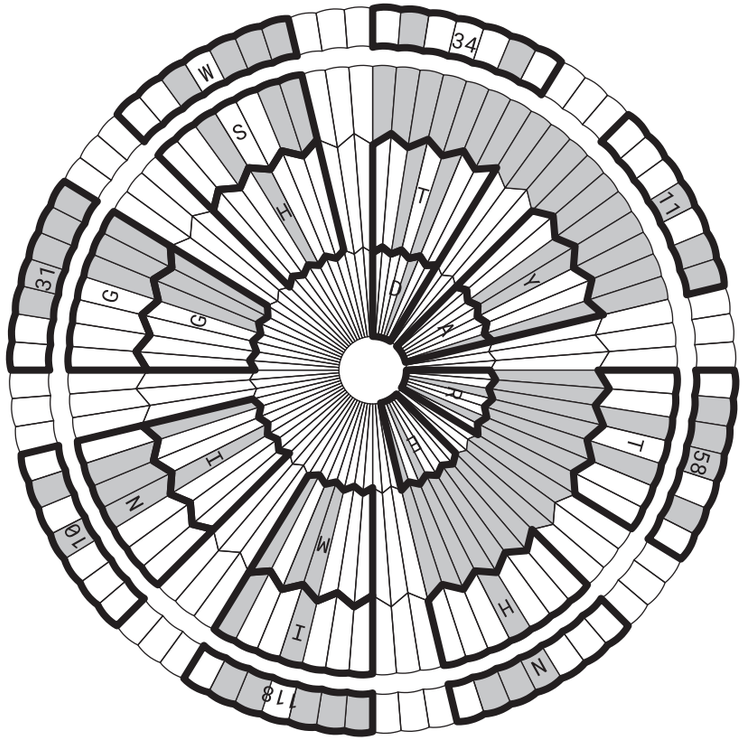
$$\begin{aligned} 4.825 = {} & 4 \times 10^3 + \\ & 8 \times 10^2 + \\ & 2 \times 10^1 + \\ & 5 \times 10^0 \end{aligned}$$

Recuerde que cualquier número elevado a 0 es igual a 1.

Cada posición en un número con múltiples dígitos tiene un significado particular. Los siete cuadros que se muestran aquí nos permiten representar cualquier número del 0 al 9.999.999:



Puesto que cada posición corresponde a una potencia de diez, no se requiere un símbolo especial para el diez, porque este se representa colocando el 1 en una posición diferente y utilizando el 0 como marcador de posición.



Alrededor del círculo exterior hay también algunos números codificados, que revelan la latitud y la longitud del Laboratorio de Propulsión a Chorro: 34°11’58”N 118°10’31”O. Con el simple sistema de codificación usado aquí, no hay nada que distinga letras y números. Los números 10 y 11 que forman parte de las coordenadas geográficas podrían ser las letras J y K. Solo el contexto nos indica que son números.

Quizá la representación visual más común de dígitos binarios sea el omnipresente Código Universal de Producto (*Universal Product Code*, UPC), el símbolo del código de barras que aparece en casi cualquier artículo empaquetado que compremos. El UPC es una de las docenas de códigos de barras utilizados para varios propósitos. Si tiene la versión impresa de este libro, verá en la contraportada otro tipo de código de barras que codifica el ISBN (*International Standard Book Number*, código normalizado internacional para libros).

Aunque el UPC generó cierta paranoia con su introducción, en realidad es una cosita inocente, inventada con el fin de automatizar el cobro y la elaboración del inventario en el comercio minorista, algo que hace bastante bien. Antes del UPC, no era posible que las cajas registradoras de los supermercados generasen un ticket de compra desglosado. Ahora es algo habitual.

A nosotros lo que nos interesa aquí es que el UPC es un código binario, aunque podría no parecerlo a primera vista. Podría ser interesante descodificar el UPC y examinar cómo funciona.

En su forma más común, el UPC es una colección de 30 barras negras verticales de diversas anchuras, divididas por espacios de diversas anchuras, junto con algunos números. Por ejemplo, este es el UPC que aparece en la sopa de fideos con pollo Campbell de 10¾ onzas:

Ese mismo UPC apareció en la primera edición de este libro. ¡No ha cambiado en más de 20 años!

Nos sentimos tentados de tratar de interpretar de forma visual el UPC desde el punto de vista de barras negras y blancas finas, espacios estrechos y anchos y, desde luego, esa es una forma de verlo. Las barras negras en el UPC pueden tener cuatro anchuras diferentes, y las barras más gruesas pueden tener anchuras dos, tres o cuatro veces superiores a las de la barra más fina. De manera similar, los espacios más anchos entre las barras son dos, tres o cuatro veces superiores a la anchura del espacio más estrecho.

Pero otra forma de ver el UPC es como una serie de bits. Tenga en cuenta que el símbolo del código de barras completo no es exactamente lo que "ve" el escáner en la caja registradora. El escáner no intenta interpretar los números impresos en la parte inferior, por ejemplo, porque eso requeriría una técnica informática más sofisticada, conocida como reconocimiento óptico de caracteres (*optical character recognition*, OCR). En vez de eso, el escáner solo verá una tira estrecha de este bloque completo. El UPC es así de grande para proporcionar a los cajeros algo a lo que apuntar con el escáner. La tira que ve el escáner puede representarse así:

Esto parece casi código Morse, ¿verdad? De hecho, la invención original de los códigos de barras que se podían escanear se inspiró en parte en el código Morse.

A medida que el ordenador escanea esta información de izquierda a derecha, asigna un bit 1 a la primera barra negra que encuentra y un bit 0 al siguiente espacio blanco. Los espacios y barras subsiguientes se leen como una serie de 1, 2, 3 o 4 bits consecutivos, dependiendo de la anchura del espacio o de la barra. La correspondencia del código de barras escaneado en bits es simplemente:

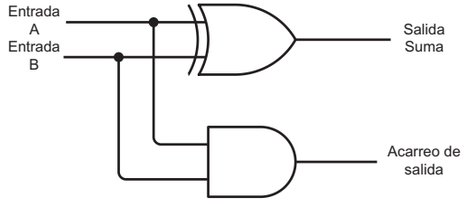
10100011010110001001100100011010001101000110101010111001011001101101100100111011001101000100101

Puede usar las dos siguientes puertas lógicas para obtener estos resultados:

XOR	0	1
0	0	1
1	1	0

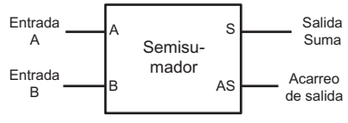
AND	0	1
0	0	0
1	0	1

La suma de dos números binarios la da la salida de una puerta XOR y el bit de acarreo lo da la salida de una puerta AND. Así pues, podemos combinar una puerta AND y una puerta XOR para sumar dos dígitos binarios llamados A y B:



¡Tenga en cuenta que esto es más complejo de lo que parece! La puerta XOR es en realidad una combinación de una puerta OR, una puerta NAND y una puerta AND, y cada una de esas puertas consta de dos relés. Pero es más fácil de entender si se ocultan muchos detalles. Este proceso se denomina a veces "encapsulación": un montaje complejo de algo se oculta en un paquete más simple. En cualquier momento, podemos desenvolver ese paquete si queremos ver los detalles, pero no es necesario.

Aquí tiene otra encapsulación: en vez de dibujar y redibujar una puerta AND y una puerta XOR, podemos representar de manera simple el circuito entero con un cuadro como este, denominado "semisumador":



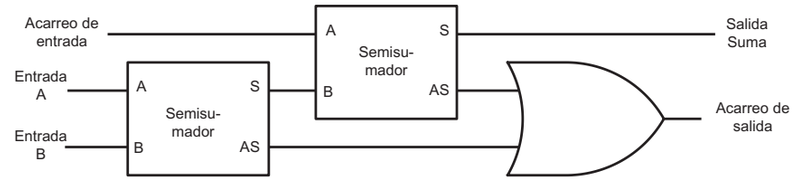
Las etiquetas S y AS significan "Suma" y "Acarreo de salida". A veces, una caja de este tipo se denomina "caja negra". Una combinación particular de entradas tiene como resultado unas salidas particulares, pero la implementación está oculta. No obstante, como sabemos lo que va dentro del semisumador, es más correcto el término "caja transparente".

Esta caja se etiqueta como "semisumador" por una razón. Está claro que suma dos dígitos binarios y nos da un bit de suma y un bit de acarreo. Pero, para números binarios superiores a 1 bit, el semisumador es inadecuado para todo lo que no sea sumar los dos bits menos significativos. No añade un posible bit de acarreo de una suma de 1 bit anterior. Por ejemplo, supongamos que estamos sumando dos números binarios como estos:

```
  1111
+ 1111
 11110
```

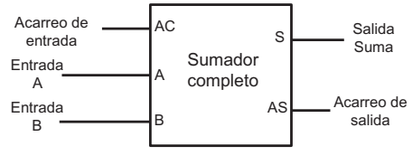
Podemos utilizar el semisumador para la suma de la columna del extremo derecho: 1 más 1 es 0 y nos llevamos 1. Para la segunda columna empezando por la derecha, en realidad necesitamos sumar tres números binarios debido al acarreo. Y eso se aplica a todas las columnas subsiguientes. Cada suma subsiguiente de dos números binarios debe añadir el bit de acarreo de la columna anterior.

Para sumar tres números binarios, necesitamos dos semisumadores y una puerta OR, conectados de esta manera:

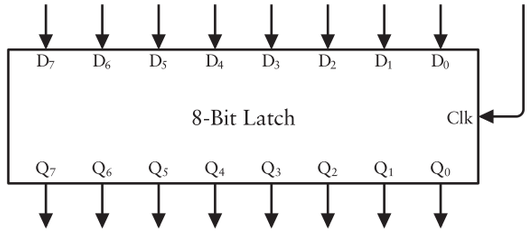


Puede que la razón por la que esto funciona no esté muy clara. Empiece por las entradas A y B al primer semisumador de la izquierda. La salida es una suma y un acarreo. Esa suma debe añadirse al acarreo de la columna anterior, llamado acarreo de entrada. Ese acarreo de entrada y la suma del primer semisumador son entradas al segundo semisumador. La suma del segundo semisumador es la suma final. Los dos acarreos de salida de los semisumadores son entradas a una puerta OR. Puede que piense que aquí hace falta otro semisumador y lo cierto es que eso podría funcionar. Pero, si examina todas las posibilidades, se dará cuenta de que nunca se da el caso de que en ambos acarreos de salida los dos semisumadores sean igual a 1. La puerta OR es suficiente para sumarlos porque es igual que la puerta XOR si nunca sucede que las dos entradas sean 1.

En vez de dibujar y redibujar ese diagrama, podemos llamarlo sumador completo:

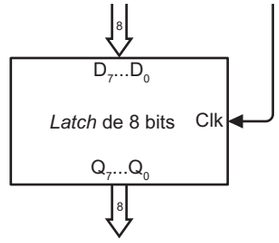


Por ahora, vamos a intentar guardar solo 1 byte de datos. Puede montar ocho biestables de tipo D activados por nivel con todas las entradas Reloj consolidadas como una sola señal. Aquí está el paquete resultante:

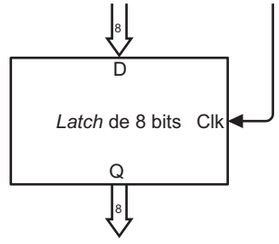


Este *latch* es capaz de guardar un byte entero de una vez. Las ocho entradas de la parte superior se etiquetan de D_0 a D_7, y las ocho salidas de la parte inferior se etiquetan de Q_0 a Q_7. La entrada de la derecha es Reloj. La señal Reloj suele ser 0. Cuando la señal Reloj es 1, el valor de 8 bits completo en las entradas D se transfiere a las salidas Q. Cuando la señal Reloj vuelve a 0, ese valor de 8 bits se queda ahí hasta la siguiente vez que la señal Reloj es 1. Las salidas $\bar{Q}$ de cada *latch* se ignoran.

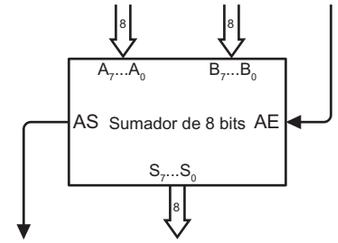
El *latch* de 8 bits también puede dibujarse con las ocho entradas Datos y ocho salidas Q agrupadas en una ruta de datos, como se muestra aquí:



O todavía más simplificado con las entradas etiquetadas simplemente como D y Q:

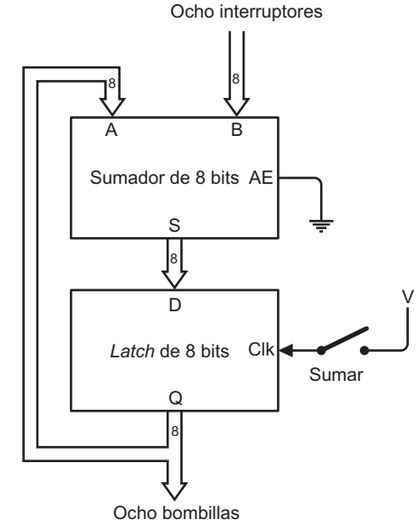


Hacia el final del capítulo 14, también se han reunido y conectado ocho sumadores de 1 bit para sumar bytes enteros:

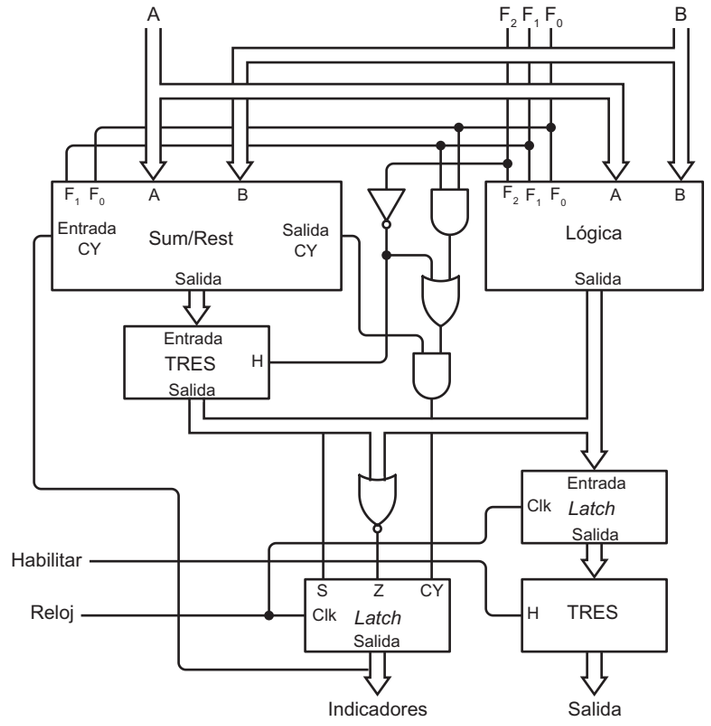


En ese capítulo, las ocho entradas A y las ocho entradas B se conectaban a interruptores, el AE (acarreo de entrada) se conectaba a tierra y las ocho salidas S (Suma) y salidas AS (acarreo de salida) se conectaban a bombillas.

El *latch* y el sumador pueden utilizarse como bloques de construcción modulares para montar circuitos más complejos. Por ejemplo, es posible guardar la salida del sumador de 8 bits en un *latch* de 8 bits. También es posible sustituir una de las filas de ocho interruptores por un *latch* de 8 bits de manera que la salida del *latch* sea una entrada al sumador. Aquí hay algo que combina esos dos conceptos para crear lo que podría llamarse "sumador acumulador" que mantiene un total acumulado de múltiples números:



Observe que el interruptor con la etiqueta Sumar controla la entrada Reloj del *latch*.



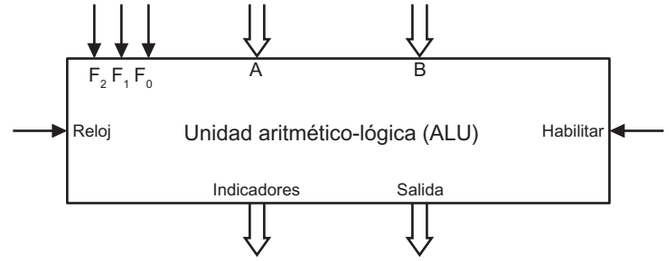
Las dos cajas con la etiqueta TRES son búferes de tres estados. El módulo Lógica habilita una salida solo para las tres combinaciones de F_0, F_1 y F_2 que seleccionan operaciones AND, OR o XOR. El búfer de tres estados en la salida del modelo Sumar/Restar solo se habilita si F_2 es 0, lo cual indica suma o resta.

Hacia la parte inferior, dos *latches* tienen las entradas Clk conectadas a una entrada Reloj en la parte inferior izquierda que se aplica a la ALU entera. Otro búfer de tres estados está controlado por una señal Habilitar en la parte inferior izquierda que también es una entrada a la ALU. El búfer de tres estados en la parte inferior derecha es la salida compuesta de los módulos Sumar/Restar y Lógica.

La mayoría de las puertas lógicas del diagrama están dedicadas al Indicador de acarreo (abreviado como CY). El Indicador de acarreo debería configurarse si la señal F_2 es 0 (indicando una operación de suma o resta) o F_1 y F_0 son 1, lo que indica una operación Comparar.

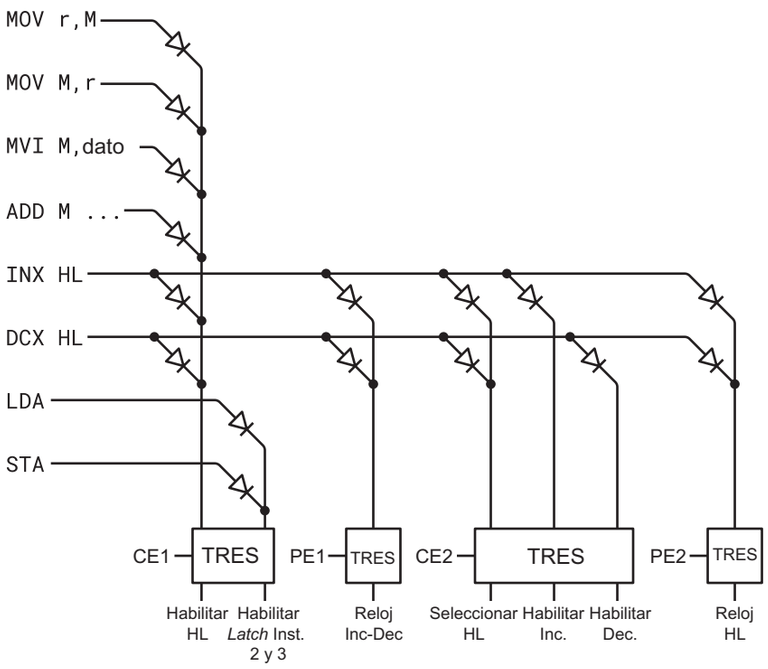
Los tres indicadores son entradas al *latch* en el centro de la parte inferior. Una puerta NOR de ocho entradas determina si el resultado de una operación es todo ceros. Ese es el Indicador de cero (abreviado como Z). El bit alto de la salida de datos es el Indicador de signo (abreviado como S). Aunque solo hay tres indicadores, se tratan como 3 bits de un byte al salir de la ALU. Entonces, el Indicador de acarreo vuelve a subir a la parte superior para proporcionar la entrada Indicador de acarreo de entrada del módulo Sumar/Restar.

El siguiente paso es ocultar toda esta lógica compleja en una caja sencilla:



¡La unidad aritmético-lógica está completa!

Aunque la ALU es un componente importantísimo de la unidad central de procesamiento, la CPU necesita más de una forma de realizar operaciones aritméticas y lógicas con números. Necesita una manera de introducir los números en la ALU y una forma de almacenar los resultados y moverlos. Ese es el siguiente paso.



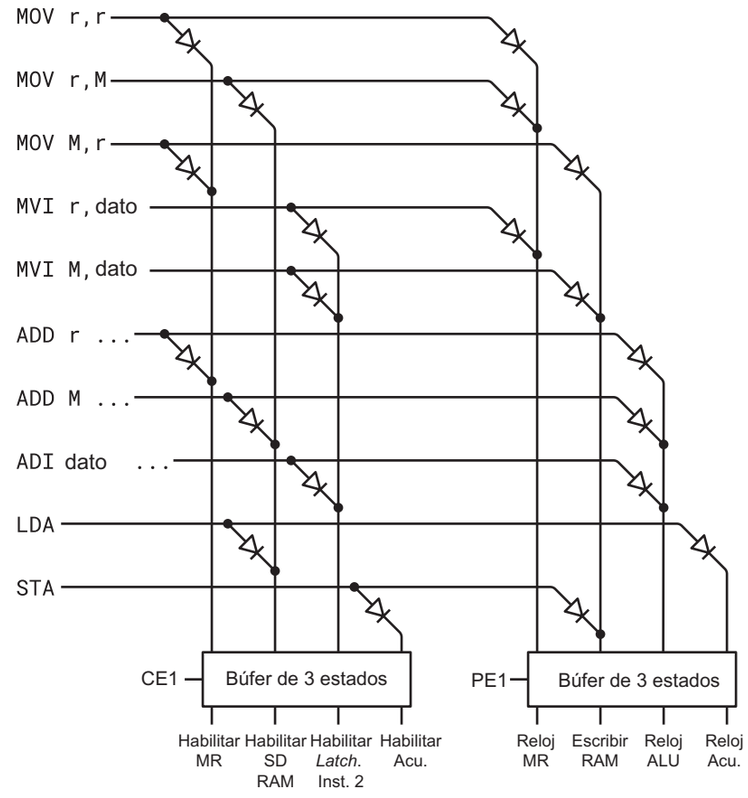
En el caso de las instrucciones `INX` y `DCX`, la señal Pulso Ejecución 1 guarda el valor de los registros HL en el *latch* incrementador-decrementador.

Las instrucciones `INX` y `DCX` son las dos únicas instrucciones que implican al *bus* de direcciones durante el segundo ciclo de ejecución. Estas dos instrucciones hacen que el valor incrementado o decrementado del registro HL esté en el *bus* de direcciones. Entonces, la señal Pulso Ejecución 2 hace que el nuevo valor de HL se guarde en los registros H y L.

La matriz ROM de diodos para el *bus* de datos de 8 bits es un poco más complicada. La he dividido en dos diagramas correspondientes a los dos ciclos de instrucción. En la siguiente figura está la primera instrucción.

Este circuito es la realización de la columna *Bus* de datos de 8 bits de la tabla en la página 362. Los dos búferes de tres estados en la parte inferior se habilitan mediante las señales Ciclo Ejecución 1 y Pulso Ejecución 1. El primer búfer de tres estados controla lo que hay en el *bus* de datos; el segundo búfer de tres estados controla dónde se almacena ese valor.

Los tres tipos de instrucciones `MOV` en la parte superior van seguidas de un destino y una fuente. Estos destinos y fuentes pueden ser cualquiera de los registros o pueden ser memoria direccionada por los registros HL. Cuando la fuente es un registro, la matriz de registro (abreviada como MR en el diagrama) está habilitada en el *bus* de datos; cuando la fuente es memoria, está habilitada la Salida Datos de la RAM. (Tenga en cuenta que la RAM está direccionada por el *bus* de 16 bits durante este tiempo y la matriz ROM de diodos para la dirección establece ese valor en los registros HL). Cuando el destino es un registro, el segundo búfer de tres estados controla la entrada Reloj para la matriz de registro. Cuando el destino es memoria, la señal Escribir RAM guarda ese valor en la memoria.



Para los dos tipos de instrucciones `MVI` ("mover inmediato"), los contenidos del *Latch* de instrucción 2 se habilitan en el *bus* de datos; el valor se almacena en la matriz de registro o se guarda en la memoria.

Todas las instrucciones aritméticas y lógicas se representan en este diagrama mediante las instrucciones `ADD` y `ADI` ("sumar inmediato"). El valor habilitado en el *bus* de datos es la matriz de registro, Salida Datos RAM o *Latch* de instrucción 2, dependiendo de la instrucción. En todos los casos, ese valor se aferra a la unidad aritmético-lógica. Estas instrucciones requieren un trabajo adicional durante el segundo ciclo de ejecución, que veremos enseguida.

C Ó D I G O

1000011 11010011 1000100 1001001 1000111 1001111

SEGUNDA EDICIÓN

LA GUÍA CLÁSICA SOBRE CÓMO FUNCIONAN DE VERDAD LOS ORDENADORES

AMPLIAMENTE ACTUALIZADA

Los ordenadores están en todas partes, de forma más evidente en nuestros portátiles y *smartphones*, pero también en nuestros coches, televisiones, microondas, despertadores, aspiradoras y otros electrodomésticos inteligentes. ¿Alguna vez se ha preguntado qué hay dentro de estos dispositivos que hacen nuestra vida más fácil y, de vez en cuando, más exasperante?

Durante más de 20 años, los lectores han disfrutado de la instructiva historia de Charles Petzold sobre la vida interior de los ordenadores y, ahora, la ha revisado para esta nueva era de la informática. Fácil de entender e ilustrado de manera inteligente, este es el libro que desvela el misterio. Descubrirá lo que las linternas, los gatos negros, los balancines y el paseo de medianoche de Paul Revere pueden enseñarle acerca de la informática y de cómo la ingenuidad humana y nuestra compulsión para comunicarnos han dado forma a todos los dispositivos electrónicos que utilizamos.

Esta nueva edición ampliada explora en mayor profundidad la construcción bit a bit, puerta a puerta, del corazón de un dispositivo inteligente, la unidad de procesamiento central que combina las operaciones básicas más simples para alcanzar los logros más complejos. Además de añadir nuevos capítulos, Petzold ha creado un nuevo sitio web, *CodeHiddenLanguage.com*, que usa gráficos interactivos animados para hacer que los ordenadores sean aún más fáciles de comprender.

Desde el simple tictac de los relojes al zumbido mundial de Internet, *Código* revela la esencia de la revolución digital.

CHARLES PETZOLD lleva 35 años escribiendo sobre programación y ordenadores. Sus libros incluyen más de una docena de tutoriales de programación y *The Annotated Turing: A Guided Tour through Alan Turing's Historic Paper on Computability and the Turing Machine* (Wiley, 2008). Vive en la ciudad de Nueva York con su mujer, la historiadora y novelista Deirdre Sinnott, y dos gatos llamados Honey y Heidi. Su sitio web es *www.charlespetzold.com*.

ANAYA MULTIMEDIA

www.anayamultimedia.es

ISBN 978-84-415-4735-3

2315186