UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Arqueología informática: emulación de una calculadora mecánica tipo Leibniz con Arduino

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Víctor Zaragozá Calabuig

Tutor: Xavier Molero Prieto

Curso 2016-2017

"Agradecimientos a Lucía por ayudarme a corregir el presente trabajo y sus ánimos para continuar pero, sobre todo a mi padre, por su ayuda con el montaje."

Resum

Mitjançant el desenvolupament d'aquest treball plantegem la implementació completa de la calculadora mecànica de Leibniz emprant com a plataforma a Arduino, formada per un microcontrolador i el seu entorn de desenvolupament. A través d'aquesta plataforma crearem una emulació propera a la calculadora original tot emprant tecnologia d'avui en dia, a més de donar-li l'habilitat de realitzar l'emulació en diferents bases (binaria, octal, etc.).

Durant el transcurs d'aquest projecte es farà una investigació prèvia per coneixer més sobre Leibniz, tant la seva vida com la pròpia calculadora. D'aquesta última s'abordarà el mecanisme emprat i les diferents operacions que realitza per, posteriorment, plasmar-les en el nostre sistema. A més, es parlarà dels predecessors a aquesta màquina i del context històric en el qual Leibniz va desenvolupar el seu dispositiu.

El sistema construït s'usarà amb fins educatius al Museu d'Informàtica de la Universitat politècnica de València, per tal de mostrar als visitants el funcionament que tenia l'original.

Paraules clau: calculadora mecànica Leibniz, cilindre escalonat, Gottfried Wilhelm Leibniz, Arduino, emulació, operacions bàsiques

Resumen

Mediante el desarrollo de este trabajo planteamos la implementación completa de la calculadora mecánica de Leibniz empleando como plataforma a Arduino, formada por un microcontrolador y su entorno de desarrollo. A través de esta plataforma crearemos una emulación cercana a la calculadora original empleando tecnología de hoy en día además de darle la habilidad de realizar la emulación en distintas bases (binaria, octal, etc.).

Durante el transcurso de este proyecto se realizará una investigación previa para conocer más acerca de Leibniz, tanto su vida como la propia calculadora. De ella se abordará el mecanismo empleado y las distintas operaciones que realiza para, posteriormente, plasmarlas en nuestro sistema. Además, se hablará de los predecesores a esta máquina y el contexto histórico en el cual Leibniz desarrolló la suya.

El sistema construido se usará con fines educativos en el Museo de Informática de la Universitat Politècnica de València, con el fin de mostrar a los visitantes el funcionamiento que tenía la original.

Palabras clave: calculadora mecánica Leibniz, cilindro escalonado, Gottfried Wilhelm Leibniz, Arduino, emulación, operaciones básicas

Abstract

Through the development of this work we propose the complete implementation of the Leibniz mechanical calculator using Arduino as a platform, consisting of a microcontroller and its development environment. Through this platform we will create an emulation close to the original calculator using today's technology

as well as giving it the ability to perform emulation on different bases (binary, octal, etc.).

During the course of this project there will be a previous research to know more about Leibniz, both his life and the calculator itself. It will address the mechanism used and the different operations that it performs to later translate them into our system. In addition, we will talk about the predecessors to this machine and the historical context in which Leibniz developed his calculator.

The built system will be used for educational purposes in the Museum of Infomatics of the Universitat Politècnica de València, in order to show visitors the operation of the original one.

Key words: Leibniz Mechanical calculator, stepped reckoner, Gottfried Wilhelm Leibniz, Arduino, emulation, basic operations

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

En este capítulo se exponen las razones por la cuales nos hemos decantado por este proyecto, así como los objetivos que pretendemos alcanzar. También, incluiremos una breve descripción de los capítulos y apartados que van a componer la memoria del trabajo realizado además del uso que se le ha dado a las diferentes fuentes bibliográficas.

1.1 Motivación

La motivación principal que nos ha llevado a elegir este proyecto sobre todos los demás ha sido el poder trabajar con Arduino [1], pues desde siempre nos ha apasionado la electrónica y la informática. Además, las matemáticas también nos han atraído desde bien pequeño y este trabajo se centraba en una calculadora.

Desde la infancia hemos visto la construcción de varios aviones de aeromodelismo así como los circuitos y componentes que lo formaban y le daban la habilidad de volar y nos ha gustado ver como, a partir de plástico y materiales que por si solos no sirven para mucho, se llegaba a la construcción de un conjunto capaz de volar. Llegando a la universidad se vio la oportunidad de aprender algo nuevo, la programación, mejorando lo aprendido de electrónica a lo largo de estos años y dotándolo de un nuevo carácter y enfoque, pues no solo se trataba de construir el *hardware* sino que además se le daba vida con el *software*.

Cuando vimos el trabajo propuesto sobre una calculadora mecánica sabíamos que era para nosotros. Juntaba todo lo que nos había apasionado durante toda la vida y poder hacerlo realidad era incluso mejor.

Otras motivaciones serían las de demostrar que con Arduino es posible hacer pequeños y no tan pequeños proyectos electrónicos mediante el uso de un *software* y poder reconstruir una de las primeras calculadoras mecánicas sobre las que se asientan los ordenadores actuales mostrando al mundo que la tecnología que tenemos ahora no ha surgido de la nada sino que lleva un proceso de desarrollo largo.

[1] Página web disponible en http://www.arduino.cc

Por otro lado, tenemos el Museo de Informática [2] de la Escola Tècnica Superior d'Enginyeria Informàtica de la Universitat Politècnica de València, la actividad del cual es mostrar a los jóvenes el recorrido de la informática durante los últimos años y sería una buena incorporación una herramienta capaz de emular la calculadora de Leibniz y que ellos mismos pudiesen ver y sentir cómo funcionaba.

1.2 Objetivos

Como objetivo general, en este trabajo se pretende aprender más a fondo a programar usando Arduino y la completa integración del código con la componente electrónica, y para ello, se pretende construir una herramienta capaz de emular el funcionamiento de la calculadora de Leibniz.

Aparte de la programación en dicha plataforma, el objetivo principal de este proyecto es conocer mejor el funcionamiento de la calculadora junto a sus características y aprender más sobre el marco y contexto histórico en el cual se construyó además de la vida de su creador, su conocimiento, y que le llevó a construirla, pues los computadores de hoy en día deben gran parte de lo que son a dicha máquina y las investigaciones y desarrollos del su autor.

1.3 Estructura de la memoria

En este apartado vamos a recorrer los distintos capítulos que componen la memoria explicando brevemente el contenido de cada uno de ellos a continuación:

- **Capítulo 1:** En este capítulo se expone la principal motivación que ha llevado al desarrollo de este proyecto, los objetivos propuestos, la estructuración de la memoria y comentarios sobre la bibliografía.
- **Capítulo 2:** En este capítulo se va a hablar del contexto histórico con relación a las distintas calculadoras o modos de cálculo predecesores a la calculadora Leibniz además de hacer un inciso en la sociedad del siglo XVII para posteriormente hablar del autor al que debe su nombre la calculadora Leibniz.
- **Capítulo 3:** En este capítulo se pretende abordar y explicar todo acerca de la propia calculadora, tanto sus orígenes como su mecanismo interno haciendo hincapié en las distintas operaciones que era capaz de realizar.
- **Capítulo 4:** En este capítulo se pretende hablar sobre Arduino, tanto a nivel *hardware* como a nivel *software*, dando comparaciones entre distintos modelos.
- **Capítulo 5:** En este capítulo se va a abordar el desarrollo de la emulación de la calculadora Leibniz usando Arduino desde su montaje hasta su funcionamiento para terminar realizando algunas operaciones con ella.

[2] Página web disponible en http://museo.inf.upv.es

- **Capítulo 6:** En este último capítulo hablaremos sobre los problemas encontrados a lo largo del trabajo así como las soluciones que hemos encontrado. También comprobaremos que se han realizado todos los objetivos propuestos y propondremos algunos trabajos futuros con relación a este.

1.4 Notas sobre la bibliografía

En este apartado vamos a hablar sobre la bibliografía empleada para la realización de este trabajo, describiendo brevemente su uso y enfocando la relación con cada apartado del trabajo. En cuanto a las imágenes, la mayoría han sido extraídas de Internet aunque algunas han sido creadas pues no se han podido encontrar.

- Con el fin de situarnos en el contexto histórico de la época en la que se inventó la calculadora, hacemos uso de las referencias bibliográficas [1, 3, 4, 5, 7, 8, 12, 16, 18] para conocer más a fondo los distintos dispositivos y herramientas que se hacían servir antes de la invención de la calculadora de Leibniz y cómo funcionaban.
- Para adentrarnos más en el contexto histórico y la sociedad del siglo XVII se han empleado las referencias bibliográficas [13, 14] mediante los cuales se ha extraído la información relacionada a dicho siglo y las distintas características que hacían de él un siglo muy importante para el avance científico.
- Para aprender más sobre la vida del autor, se han empleado las referencias bibliográficas [9, 11] de las cuales se ha extraído toda la información relacionada con el trabajo de Leibniz, su vida y los avances en los distintos campos científicos que han favorecido el avance de la sociedad.
- En cuanto al funcionamiento de la propia calculadora, hemos empleado la referencias bibliográficas [15, 18] para entender mejor cómo funcionaba y de qué era capaz dicho invento, desde sus inicios hasta el refinamiento final del mismo.
- Para la realización de la máquina emuladora se han empleado las referencias bibliográficas [2, 10, 17] de las cuales se ha extraído la información relacionada con el lenguaje que usa nuestro Arduino además de las especificaciones técnicas de los distintos modelos.
- Para entender mejor el funcionamiento de las distintas operaciones matemáticas que se van a usar a lo largo de este trabajo, se ha empleado la referencia bibliográfica [6] de la cual se ha aprendido la esencia de los cálculo tanto de la máquina original como para desarrollar la emulación.

CAPÍTULO 2
Contexto histórico

El presente capítulo tiene como propósito dar a conocer un poco de la historia de la informática que precede a la creación de la calculadora de Leibniz. Se realizará también una visita al contexto histórico durante el cual se desarrolló dicha calculadora y la vida del propio autor, dando a conocer sus estudios y conocimientos que poseía y que le llevaron a la invención y construcción de su máquina.

2.1 Primeras calculadoras

La necesidad del ser humano de facilitar las cosas del día a día junto con su inteligencia e ingenio, le han llevado a la invención de distintos métodos e instrumentos que se encargarían de ahorrar tiempo y esfuerzo por parte del usuario en tareas que podrían resultar tediosas e incluso repetitivas como puede ser el cálculo. Por tanto, en este capítulo nos vamos a centrar en las distintas herramientas creadas a lo largo de la historia que han facilitado al ser humano el cálculo, empezando por el ábaco hasta la Pascalina.

2.1.1. El ábaco

Aunque se considere el primer instrumento de cálculo capaz de hacer operaciones aritméticas tales como las sumas, restas o multiplicaciones, su origen sigue siendo incierto. Los expertos datan su invención en la época mesopotámica, entre los años 2700 y 2300 a.C., en Sumeria, una región del antiguo medio oriente que formaba parte de la antigua Mesopotamia. Pese a su incertidumbre, se sabe que los sumerios fueron los que inventaron el sistema sexagesimal, el cual usamos en los relojes, y posiblemente el ábaco. A través del tiempo, empezaron a aparecer nuevas versiones de este invento en regiones como China, Japón, Roma, Rusia, India, Korea, etc., todas ellas con el mismo funcionamiento pero distinto diseño.

Para comprender mejor el porqué de la invención del ábaco, vamos a remontarnos al sistema que usaban anteriormente y que, poco a poco fue mejorando y difundiéndose, el cual se basaba en el uso de guijarros, pequeñas piedras redondeadas, como se ve en la Figura 2.1, que se usaban en la prehistoria para llevar las cuentas. Una práctica frecuente de su uso durante la antigüedad es la de contar

Figura 2.1: Guijarros que podemos encontrar en una playa o río y que eran usados antes del ábaco

los guerreros fallecidos en batalla, donde, antes de partir, cada guerrero colocaba un guijarro en una urna, y al volver, cada uno de ellos sacaba uno, quedando así, la cantidad de soldados fallecidos o perdidos en combate.

Como se puede ver, el cálculo con guijarros era un poco engorroso, pues cuando se trataban de grandes cálculos, eran necesaria una gran cantidad de ellos. Por tanto, decidieron cambiar de sistema, del anterior sistema sexadecimal al decimal, con base diez, con lo que ahora, diez guijarros podían ser sustituidos por otro más grande y así sucesivamente para centenas, millares, decenas de millares, etc. Este paso de sistema se debe a la gran cantidad de guijarros necesaria para representar el siguiente orden de magnitud en el sistema sexagesimal. De esta manera, se disminuía en gran medida la cantidad de guijarros necesarios para realizar operaciones más grandes respecto al uso anterior, pudiendo usar piedras más grandes y con mayor frecuencia para los distintos ordenes de magnitud, reduciendo enormemente la cantidad necesaria para realizar operaciones y su almacenaje si era necesario.

El uso de guijarros pasó a llamarse cálculo por el nombre en latín que recibían estos, «calculus» para el singular y «calculi» para el plural. De esta forma, y hasta la actualidad, la palabra cálculo y el uso de los guijarros están completamente relacionadas al surgir una del uso de la otra.

Con el nuevo sistema en base diez, y el uso de tablas de madera o arcilla que empezaron a aparecer con la invención de la escritura para plasmar datos, ideas o pensamientos, empezó la invención del ábaco, surgiendo en primer lugar en la antigua Sumeria, aunque como se ha comentado anteriormente, es solo una aproximación.

Originalmente, el ábaco sumerio estaba formado por una tabla de madera o arcilla, o podía estar dibujado sobre la arena. Esta tabla tenía columnas consecutivas, aumentando por cada columna el orden de magnitud del sistema numérico, base diez, con lo que tenían columnas para las decenas, centenas, etc. Solamente se usaba para sumas y restas pues el invento era demasiado sencillo y hacer operaciones de mayor volumen como las multiplicaciones y divisiones resultaba

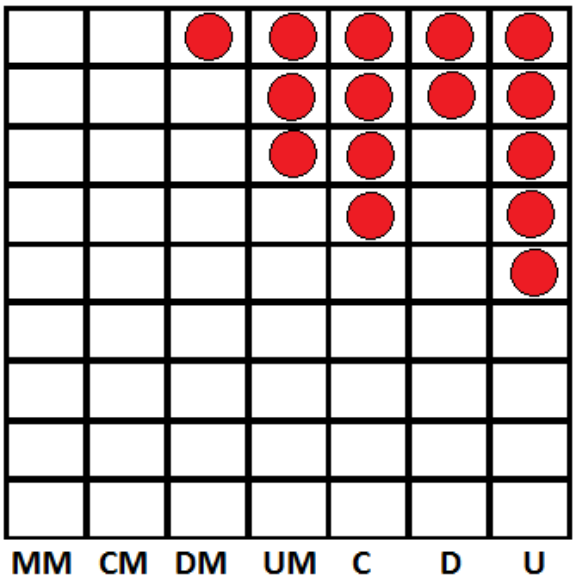


Figura 2.2: Ábaco sumerio representación del número 13 425)

costoso. Se puede ver una representación de este ábaco en la Figura 2.2, con un valor plasmado de 13 425.

Como se hemos visto anteriormente, el ábaco fue extendiéndose a otras regiones. Podemos ver el ábaco chino en la Figura 2.3 en el que se usaban varillas para mantener los guijarros ordenados. El inca a la izquierda en la Figura 2.4 también denominado quipu, se añadían nudos para representar las distintas cifras. El romano a la derecha de la Figura 2.4 el cual contaba con una caja con hendiduras por donde se moverían las piedras dándole mayor rigidez. Todos ellos tenían el mismo funcionamiento que el sumerio pero con un diseño diferente.

Aunque actualmente el ábaco pueda parecer un juguete para niños, cabe destacar que, para la época en la que se inventó, fue muy útil y facilitó en gran medida el desarrollo del comercio al disponer de una herramienta capaz de simplificar los cálculos en los trueques que se realizasen.

Figura 2.3: Ábaco chino

Figura 2.4: Ábaco inca o quipu (izquierda) y ábaco romano (derecha)

2.1.2. Las varillas de Napier

También conocidas como ábaco neperiano, deben su nombre a su creador e inventor John Napier (1550 – 1617), el cual las creó para el cálculo de productos y cocientes de números. Su diseño se publicó en una obra impresa en Edimburgo en 1617 titulada *Rhabdologiæ*, cuyo nombre es el que se le da en griego a este ábaco.

Usando el método de Napier y sus varillas, el cálculo de productos y cocientes se reduce a operaciones de sumas y restas respectivamente, algo que va a ser arrastrado durante muchos años en las posteriores máquinas de cálculo mecánicas, tales como la calculadora tipo Leibniz en la que se centra este trabajo y de la que se hablará en sucesivos capítulos.

Para llevar a cabo las operaciones aritméticas, era necesario un tablero sobre el cual se colocaban las distintas varillas. Este tablero estaba formado por una superficie plana o tabla, con rebordes que dejaban un espacio en el interior donde se colocarían las distintas varillas neperianas para realizar las operaciones de multiplicación, division, raíces cuadradas e incluso cúbicas. El reborde izquierdo estaba dividido en nueve casillas en las que estaban escritos los dígitos del 1 al 9, los cuales, más adelante, veremos para qué se usaban.

En cuanto a las varillas, estaban formadas por tiras de madera, metal o cartón grueso. La cara frontal estaba dividida en diez cuadrados, todos ellos a su vez divididos en dos triángulos, excepto el superior, en el que se mostraba el número de la correspondiente tabla de multiplicar y que representaba a la varilla. Los cuadrados inferiores mostraban los valores consecutivos, es decir, los duplos, triplos, cuádruplos, etc., de la propia tabla de multiplicar.

El juego completo lo componían nueve varillas, numeradas del 1 al 9, como se puede ver en la Figura 2.5, donde además aparece la varilla del 0, que en este caso es irrelevante y no aporta nada a las operaciones. Se observa también que las

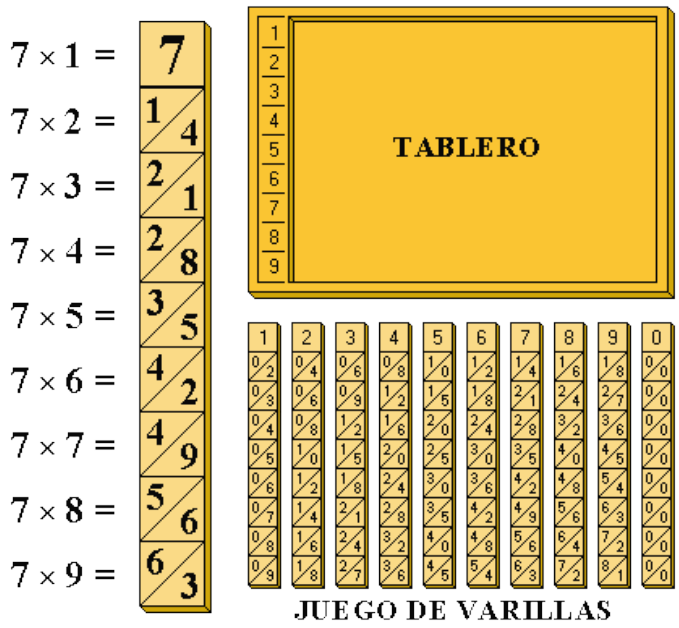


Figura 2.5: Representación de las varillas de Napier y su tablero

varillas no son más que las tablas de multiplicar como se ha comentado anteriormente, pero colocadas de forma ingeniosa para facilitar su uso en los cálculos.

En la realización de los cálculos, por lo general, eran necesarios varios juegos de varillas pues en las operaciones normalmente un mismo dígito aparece varias veces. Para reducir el número de varillas y por tanto su precio, las originales constaban de cuatro caras en las que lados opuestos sumaban 9 centrándonos solamente en el número superior, pudiéndose ver esta característica en la Figura 2.6, formada por el conjunto de todas las varillas con sus correspondientes dígitos. Esta disposición permitía el uso de números de hasta diez cifras siempre y cuando un dígito no se repitiese más de cuatro veces en el mismo número.

Basándonos en las varillas descritas anteriormente, si quisiéramos hacer una multiplicación como por ejemplo 483 256 por 8, tendríamos que colocar las varillas como aparecen en la Figura 2.7. De esta manera, extraemos la fila que representa al 8 en el reborde izquierdo, el número por el cual vamos a multiplicar y hacemos una suma en diagonal como se ha mostrado en la figura, recordando que puede haber acarreo, dando como resultado el valor 3 866 048.

Para hacer multiplicaciones de más cifras, simplemente anotamos los resultados de los productos parciales, entendiendo estas como las multiplicaciones de cada uno de los dígitos como se ha visto con anterioridad, y sumamos en escalera como hemos aprendido de pequeños con los resultados de cada una de estas operaciones.

En cambio, si queremos hacer una división, como por ejemplo 46 785 399 dividido entre 96 431, se colocarían las varillas de igual manera que con las multiplicaciones, a excepción de que se posicionan en base al divisor y no al dividendo, en este caso, 96 431. Además, tendríamos que calcular cada fila como se ha hecho anteriormente con las multiplicaciones para ahorrarnos los tanteos intermedios que eran necesarios cuando hacíamos las divisiones a mano. Al obtener estos valores, simplemente escogemos el que es inmediatamente inferior al valor del resto

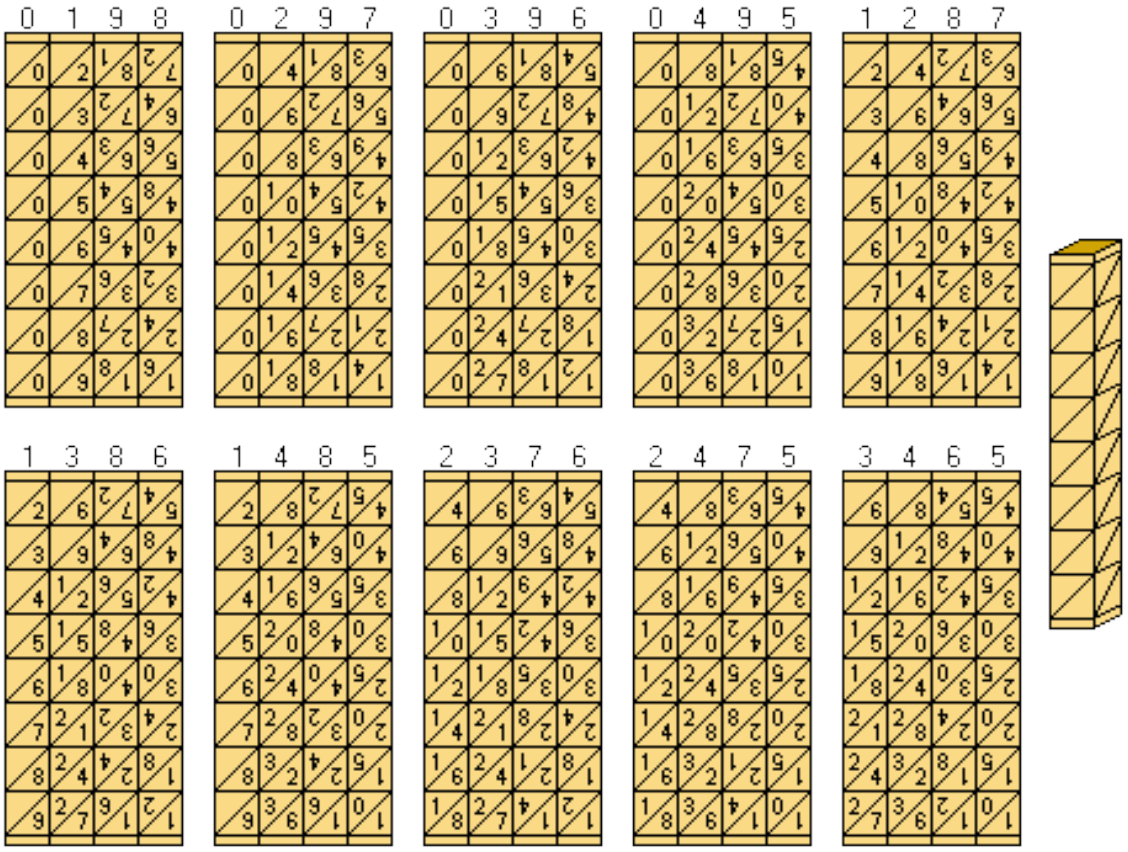


Figura 2.6: Representación de las varillas de Napier originales

actual, pero solamente la parte izquierda superior al propio divisor como se observa en la Figura 2.8. El primer cálculo se aplica sobre 467 853, pues el valor es la parte izquierda del dividendo inmediatamente superior al propio divisor. Una vez tenemos la parte donde vamos a aplicar la resta, buscamos el valor inmediatamente inferior a éste, dando lugar al 385 724, ubicado en la cuarta fila y por tanto obteniendo el primer dígito del resultado, el 4. De igual manera, se aplica con el resto de la operación, hasta que el resto sea menor que el divisor.

2.1.3. El reloj calculante

Calculadora inventada y diseñada por Wilhelm Schickard, centrada en los calculos matemáticos según cita en una carta escrita para su amigo Johannes Kepler, el 20 de septiembre de 1623, en la que describe su máquina a la que bautiza con el nombre de «reloj calculante»incluyendo varios bocetos del mismo.

Un fragmento de la carta contando los detalles de la máquina decía así: «... *Te haré en otra ocasión un diseño más cuidadoso de la máquina aritmética; en resumidas cuentas, mira lo siguiente: aaa son los botones de los cilindros verticales que llevan las cifras de la tabla de multiplicación, que aparecen a la voluntad en las ventanas de las correderas bbb. Los discos ddd son solidarios con ruedas dentadas interiores, de diez dientes, engranadas entre sí de manera que, si la rueda de la derecha da diez vueltas su vecina de la izquierda sólo da una; y que si la primera de la derecha da cien vueltas la tercera de la izquierda da una, y así sucesivamente. Todas ellas giran en el mismo sentido por lo que es necesaria una rueda de reenvío del mismo tamaño engranando permanentemente con su*

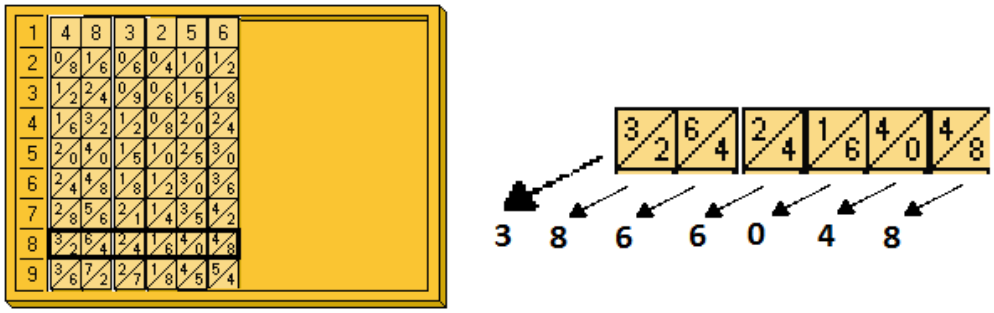


Figura 2.7: Ejemplo de multiplicación con las varillas de Napier

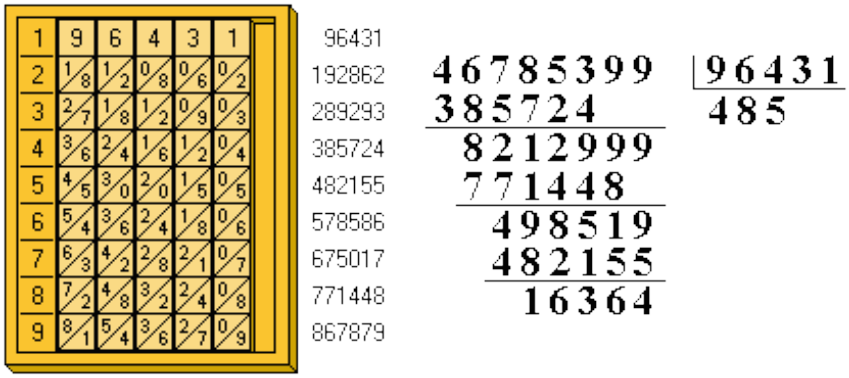


Figura 2.8: Ejemplo de división con las varillas de Napier

vecina de la izquierda, aunque no con la de la derecha, lo que requiere un cuidado especial en la fabricación. Las cifras marcadas en cada una de las ruedas se leen en las aberturas ccc de la plancha central. Finalmente, sobre el zócalo se encuentran los botones eee que sirven para inscribir en las aberturas fff las cifras que se hayan de anotar en el curso de las operaciones. Sería muy prolijo completar esta rápida descripción que se comprendería mejor con la práctica. Te había hecho fabricar un ejemplar de esta máquina por J. Pfister, que vive aquí; pero ha sido destruido hace tres días junto con algunas de mis pertenencias... en un incendio nocturno» [1].

Uno de los bocetos originales que incluía en la carta puede verse en la Figura 2.9, donde se aprecia el diseño exterior y parte de los engranajes que componían la calculadora. En el dibujo de la parte izquierda, se pueden observar las letras que usa Schickard como referencias en la carta para describir las distintas partes de la calculadora.

Dada la fecha de la carta, se demuestra que es la primera calculadora mecánica de la historia de la informática pues, anteriormente se pensaba que era la Pascalina, la cual se inventó posteriormente y no tuvo influencias por el «reloj calculante»por hayarse perdido durante tres siglos. Esta pérdida se debe a que la calculadora no se llegó a construir entera y se perdió en un incendio, cosa que comenta en la cita de la carta anterior, hasta que en el siglo XX, más concretamente en 1960, el barón Bruno Von Freytag-Löringhoff construyó la calculadora basán-

[1] Extraído del artículo [8] el cual ha sido traducido del alemán al español para su mejor interpretación

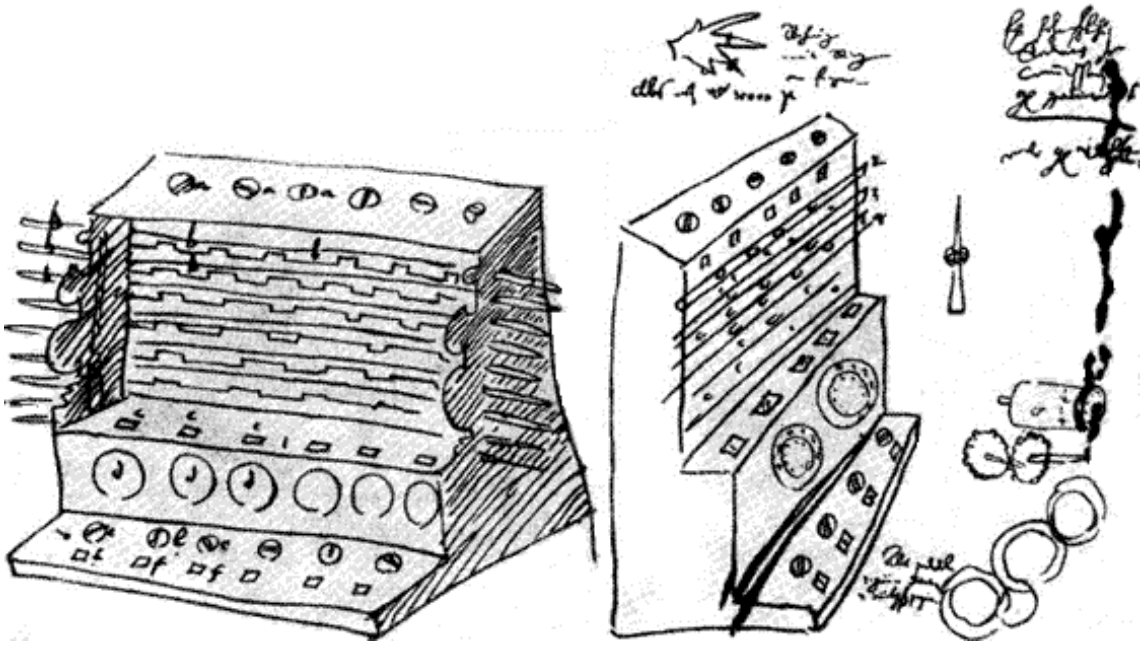


Figura 2.9: Dibujos originales del reloj calculante

dose en los dibujos del autor y las indicaciones de la carta enviada a Kepler. En la Figura 2.10 se puede ver dicho diseño, el cual funciona perfectamente.

Esta calculadora era capaz de realizar operaciones básicas tales como sumas, restas, multiplicaciones y divisiones de hasta seis dígitos y contaba con una campana que sonaba cuando se producían errores de desbordamiento.

Tal y como se enuncia en la carta, la máquina funcionaba con engranajes, los cuales estaban conectados a los que tenían inmediatamente a la izquierda, de tal manera que si una rueda daba diez vueltas, la que estaba a su izquierda daría una, produciéndose de esta manera, el conocido efecto de acarreo. Esta estructura se aplicaba a todas las ruedas, a excepción de la que se encontraba más a la izquierda, la cual, una vez llegaba a la cifra de 9 y se intentaba incrementar su valor, hacía sonar la campana indicando que se había producido el desbordamiento, o lo que es lo mismo, habíamos sobrepasado el valor máximo que aceptaba la calculadora.

La calculadora estaba dividida en tres módulos que eran independientes entre sí, de tal forma que la parte superior se encargaba de hacer las multiplicaciones y divisiones con un mecanismo y diseño similar al de las varillas de Napier comentadas en la sección anterior, las sumas y restas se llevaban a cabo en la parte central, haciendo uso de las ruedas numeradas del 0 al 9. En cuanto a la parte inferior, servía para almacenar datos de operaciones anteriores, para su posterior uso o como resultado final, y no contaba con engranajes entre las distintas cifras. Todo esto se puede apreciar mejor en la imagen de la Figura 2.10.

2.1.4. La Pascalina

La Pascalina o calculadora de Pascal, fue inventada y construida por el filósofo y matemático Blaise Pascal (1623-1662) en el año 1642. Originalmente fue

Figura 2.10: Reconstrucción del reloj calculante por el barón Bruno Von Freytag-Löringhoff

llamada «máquina aritmética», pero posteriormente le cambió el nombre a «rueda Pascalina», de tal manera que tenía el nombre del propio autor. Con el paso del tiempo se quedó con el nombre de «Pascalina».

La idea de esta máquina surgió a la edad de 19 años, cuando su padre acababa de ser nombrado superintendente de la Alta Normandía por el cardenal Richelieu. Dicha idea fue concebida con la necesidad de ayudar a su padre, ya que en su nuevo puesto de trabajo iba a necesitar realizar muchos cálculos para restaurar el orden de los ingresos fiscales del lugar trabajando como cobrador de impuestos. El invento era capaz de hacer sumas y restas de forma sencilla, mientras que las multiplicaciones y divisiones se hacían a partir de sumas y restas repetidas, respectivamente.

El diseño completo ocupaba algo menos que una caja de zapatos como se puede ver en la Figura 2.11. En su interior se encontraban ruedas dentadas conectadas entre sí, formando una cadena de transmisión al igual que ocurría con el reloj calculante de Schickard para las sumas y restas, aunque no se pudo basar en su diseño por la pérdida comentada anteriormente. En el exterior, tenemos las ruedas de entrada mediante las cuales introducimos las cifras y en la parte superior, los registros que contienen el valor de las operaciones y una barra que podemos desplazar. Esta barra tiene dos posiciones, la de suma y la de resta, siendo la posición de suma cuando la barra está en la parte superior y la de resta, la parte inferior.

Las ruedas usaban el sistema decimal, por lo tanto, estaban compuestas por diez pasos o posiciones distintas y numeradas del 0 al 9, diseño que se usará en máquinas posteriores basadas en la Pascalina. El diseño original de la calculadora contaba con ocho de estas ruedas, seis de las cuales servían para representar los números enteros y se encontraban dispuestas a la izquierda, y dos ruedas encargadas de la parte decimal, las dos de la derecha.

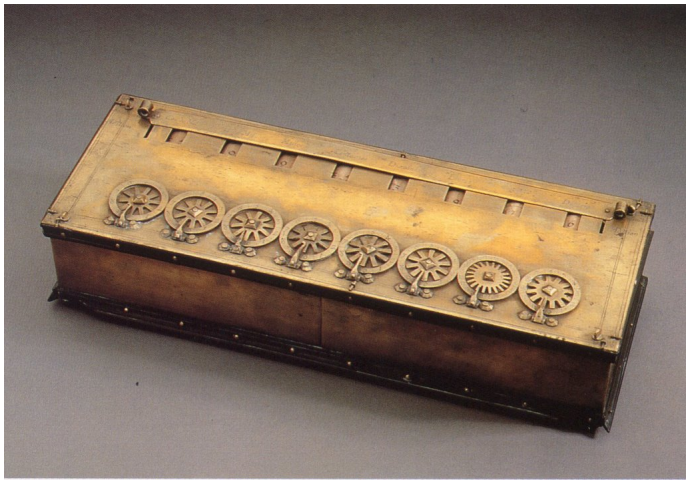


Figura 2.11: La Pascalina inventada y construida por Blaise Pascal en 1642

Para realizar las operaciones de suma y resta, se empleaban las ruedas de entrada, las cuales solo giraban en un sentido por lo que las restas debían de hacerse mediante sumas en «complemento a 9». En cuanto a las multiplicaciones y divisiones, se aplicaban sumas o restas tantas veces como el valor por el que se quisiese multiplicar o dividir, lo cual era una forma tediosa de realizar estas operaciones por la cantidad de giros necesarios. Esto se solucionaría con la calculadora de Leibniz, la cual tiene un capítulo más adelante.

El mecanismo de acarreo en sumas funcionaba de forma similar al reloj calculante de Schickard. En este caso, cuando una rueda alcanzaba el valor de 9 y se le quería incrementar en 1, la rueda pasaba al valor siguiente, el 0, mientras que se avanzaba una posición mediante un gancho a la rueda inmediatamente a su izquierda. Dicho mecanismo se puede observar en la imagen de la Figura 2.12, donde se aprecia el interior de la calculadora así como sus engranajes en un boceto muy detallado.

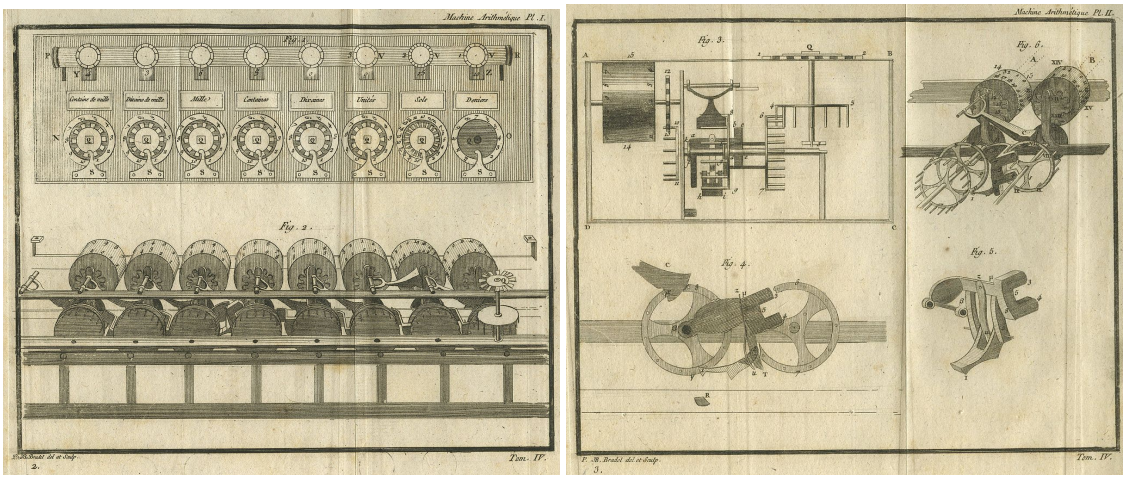


Figura 2.12: Boceto detallado del mecanismo interno de la Pascalina

En cambio, si queríamos hacer una resta, necesitábamos usar el método del «complemento a 9» comentado anteriormente y para ello, debíamos desplazar la barra hacia el centro de la calculadora colocando por tanto, la barra en posición de resta. De esta manera, los registros del resultado quedarían tapados y se

mostraría los del «complemento a 9». En estos nuevos registros, introduciríamos el complementario del valor del cual vamos a substraer. Por ejemplo, queremos restar 12 538 de 37 823, por lo que tendríamos que introducir el complementario de 37 823 que es 62 176 en las ruedas de entrada, mostrándose el valor original, 37 823 en los registros complementarios. Una vez tengamos este valor, simplemente tendríamos que sumar el valor que queremos restar, en este caso 12 538 y se mostraría el resultado en los registros complementarios.

A lo largo de los años, Pascal construyó alrededor de una cincuentena de versiones de la Pascalina con la intención de conseguir una calculadora que realmente le satisfaciera. Aunque, a pesar de la calidad técnica del invento y del prestigio que esta conllevo a su autor, no fue usada en las oficinas reales ni tuvo la aceptación por los compañeros de su padre. Esto se debe a que los contables, por rutina o por miedo a ser desbancados por la máquina, prefirieron rechazarla. Por otra parte, los empresarios no vieron beneficio alguno en la compra de una máquina tan costosa de hacer, pues se construían a mano, mientras que el trabajo manual era mucho más barato.

2.2 La sociedad del siglo XVII

Con el siglo XVII vino la era de la revolución científica, posiblemente el cambio de orientación más importante de la historia de la ciencia. Los estudiosos y eruditos empezaron a preguntarse el porqué de las cosas, por qué ocurren determinados acontecimientos.

El método desarrollado en esta época para afrontar estos nuevos pilares de la ciencia, llamado «nuevo método», consistía en investigar la naturaleza de las cosas mediante el uso de los propios sentidos, como la vista o el olfato, y expresar dichas observaciones mediante el uso de un lenguaje matemático exacto.

La importancia del razonamiento especulativo que se había mostrado efectivo hasta ahora, empezaba a perder terreno frente a la experimentación y el método hipotético-deductivo, el método científico por excelencia para este siglo. De este modo, la interpretación de los fenómenos que ocurrían en el mundo, observados desde una parte más mecanicista, junto a las bases matemáticas que empezaban a aflorar, consiguieron imponerse frente al estándar vigente.

Italia, lugar que había sido hasta ahora la cuna de la ciencia durante más de un siglo, había dado paso a una expasión y desplazamiento científico a otras regiones geográficas. Algunos de los países que empezaban a realizar aportaciones significativas al desarrollo científico fueron los germánicos y anglosajones, de los cuales han surgido grandes científicos conocidos mundialmente en la actualidad y que han aportado un gran avance para llegar a la tecnología que usamos hoy en día.

Todo esto conllevó la aparición de grandes comunidades científicas alrededor del globo. Comunidades compuestas por la agrupación de varias personas de diversos lugares del mundo y culturas que compartían intereses científicos y por los enigmas que empezaban a aparecer al observar el mundo que los rodeaba.

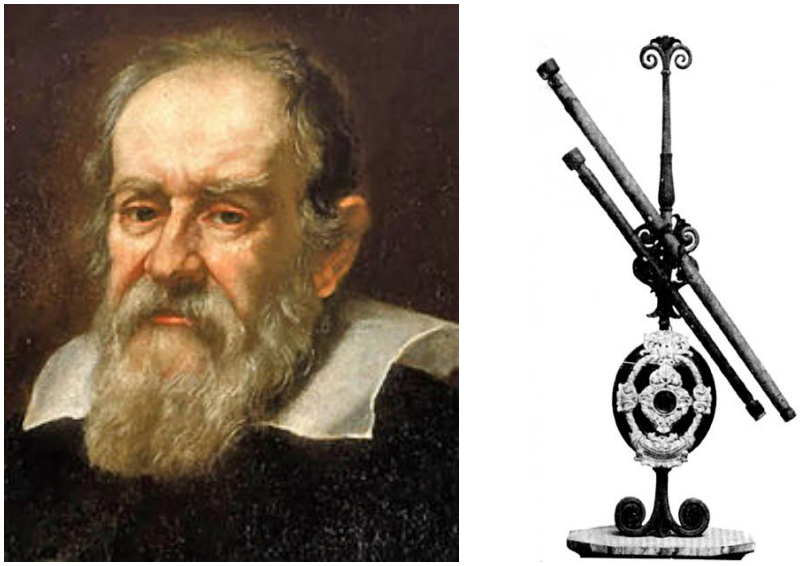


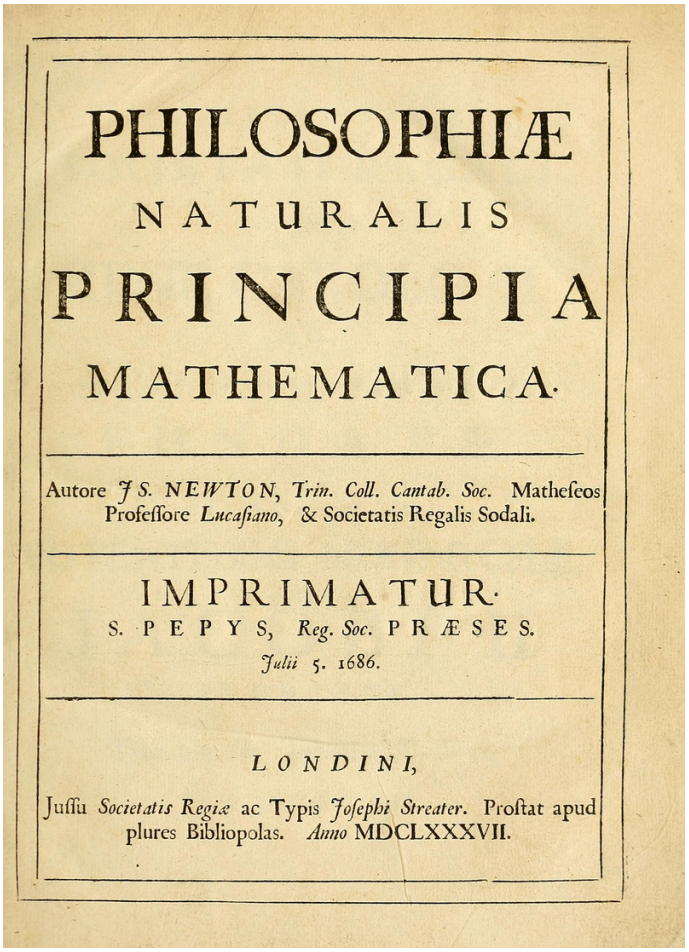
Figura 2.13: Retrato de Galileo Galilei y su telescopio

Todo este espíritu de ataque a lo tradicional para realizar el cambio científico en la época, fue propiciado por Galileo Galilei, el cual podemos ver en el retrato de la Figura 2.13 junto uno de sus inventos, el telescopio, el cual usó para observar la luna y el universo, creando varios dibujos de las distintas fases lunares. A causa de su atrevimiento al contradecir las bases científicas creadas por Aristóteles, fue condenado a cadena perpetua a causa de la denuncia del cardenal Belarmino por considerarse una herejía y se mandó quemar todas las copias de su libro llamado *Diálogo sobre los dos máximos sistemas del mundo*. Más adelante, Isaac Newton siguió al camino de Galileo con el método científico, permitiendo la generalización de la mecánica clásica, la cual se basaba en el estudio de las leyes del comportamiento sobre los cuerpos físicos macroscópicos que se encuentran en reposo y a unas velocidades pequeñas comparadas con la velocidad de la luz.

Aunque en la actualidad separamos el método científico de la religión, en aquel entonces se prefería no enfrentarse directamente a la iglesia y, por tanto, buscar en sus estudios justificaciones para soportar las creencias de que todo lo que existe se debe a la creación por parte de un ser o ente superior y sumamente poderoso. Esto se puede observar en la publicación que hizo Isaac Newton *Philosophiæ naturalis principia mathematica*, portada que se puede ver en la Figura 2.14 y que significa «Principios matemáticos de la filosofía natural», quien años después confeso que mientras avanzaba en sus investigaciones pensaba en este tipo de justificaciones.

Por otra parte, empezó a surgir una línea de pensamiento aportada por Descartes, el racionalismo cartesiano, el cual apoyaba el método matemático y la visión mecanicista del universo, pero desconfiaba de los sentidos y no creía en la experimentación. Algunos pensadores siguieron la línea del racionalismo, como Spinoza, Locke y Leibniz, impulsando relevantemente el desarrollo de algunas ramas del conocimiento.

La segunda mitad del siglo XVII se caracterizaba por un abandono gradual de las doctrinas y pensamientos impuestos por Aristóteles, consideradas ya viejas


PHILOSOPHIÆ
NATURALIS
PRINCIPIA
MATHEMATICA.

Autore *J. S. NEWTON*, *Trin. Coll. Cantab. Soc.* Matheſeos Profeſſore *Lucaſiano*, & Societatis Regalis Sodali.

IMPRIMATUR.
S. PEPYS, *Reg. Soc.* PRÆSES.
Julii 5. 1686.

LONDINI,
Juſſu *Societatis Regiæ* ac Typis *Joſephi Streater*. Proſtat apud plures Bibliopolas. *Anno* MDCLXXXVII.

Figura 2.14: Portada del libro *Philosophiæ naturalis principia mathematica*

mentalidades, creándose una ciencia experimental que viviría durante el resto del siglo y continuaría durante el siglo XVIII.

Como se ha visto, la sociedad veía los inventos, estudios y conocimientos nuevos como una amenaza a los pensamientos que se habían asentado hasta entonces, especialmente la Iglesia, condenándose como herejía algunos de ellos y tomando medidas al respecto, como se ha visto con Galileo. Un esquema que resume los grandes cambios científico de la época se puede ver en la Figura 2.15 [2].

2.3 Leibniz, inventor de la rueda escalonada

Gottfried Wilhelm Freiherr von Leibniz, el cual podemos ver en el retrato de la Figura 2.16, nació el 21 de junio de 1646 en Leipzig, una ciudad alemana al noroeste de Sajonia. En la Figura 2.17 se puede ver la localización en la actualidad. Sus padres fueron Friedrich Leibniz y su tercera esposa Catharina Schmuck-Leibniz, y tuvo dos hermanos consanguíneos, un chico y una chica, por parte de padre, con cada una de las anteriores esposas, y una hermana con la tercera mujer, Anna Catharina, dos años más joven que él.

[2] Mapa conceptual obtenido de: https://sociedadytecnologiag2.wordpress.com/2011/03/19/revolucion-cientifica/

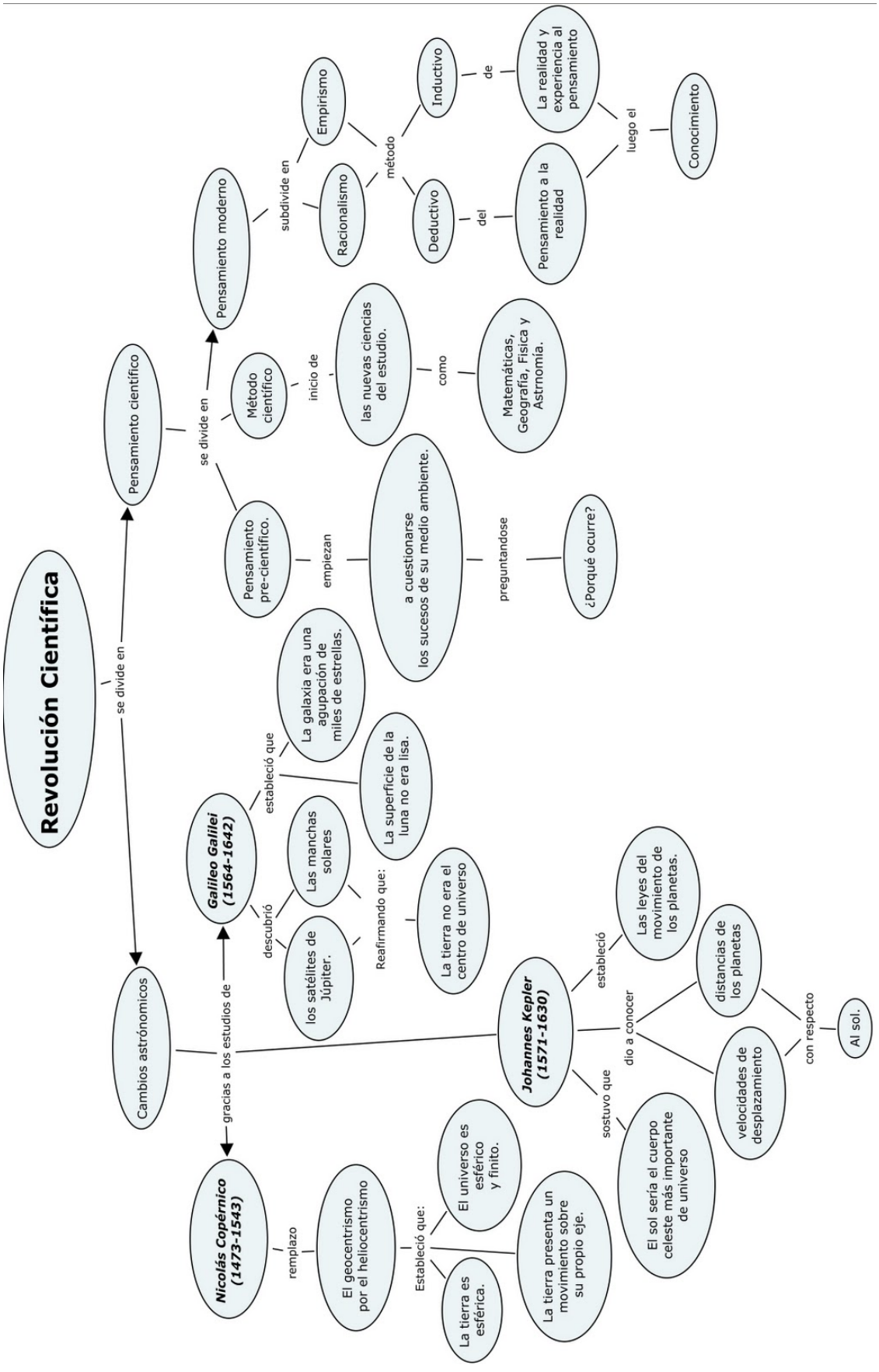


Figura 2.15: Mapa conceptual de los grandes cambios científicos originados en el siglo XVII

Figura 2.16: Retrato de Leibniz (1646 - 1716)

Su padre, aunque era notario, daba clase como filósofo moral y vicepresidente de la facultad de filosofía en la Universidad de Leipzig, mientras que su madre, Catharina, era hija de un famoso abogado, la cual fue muy bien educada e inteligente. Sin embargo, Friedrich Leibniz murió cuando Leibniz tan solo tenía seis años y fue cuidado por su madre. La religión y la moralidad que aprendió fue la que obtuvo de su madre, la cual jugó un papel muy importante en su vida y filosofía.

A una temprada edad, con 7 años, Leibniz fue enviado a la popular Nikolaischule (Escuela Nicolai), la cual se puede ver en la fotografía de la Figura 2.18 tomada en la actualidad. En esta escuela empezó a manifestarse su insaciable deseo de aprendizaje y conocimiento, volviéndose autodidacta. De esta forma, a la edad de 12 años, había aprendido latín por sí solo, el cual usará de forma concurrente durante toda su vida, y empezó a aprender griego, motivado por leer los libros de su padre. Mientras avanzaba en la escuela, se le enseñó la lógica de Aristóteles y teoría del conocimiento, los cuales no parecieron satisfacerle y empezó a desarrollar sus propias ideas y como mejorar las de Aristóteles. Más adelante, Leibniz recalcaría que en esta época de su vida, intentaba encontrar un orden en las verdades lógicas y, aunque no lo sabía en ese momento, estas ideas son las que están detrás de algunas de las pruebas matemáticas.

Una vez aprendió latín y griego, además de sus estudios en la escuela, se dedicó a leer los libros de su padre, centrándose en libros de metafísica y libros de teología, tanto de escritores católicos como protestantes.

En 1661, a los 14 años, entró a la universidad de su padre. Puede parecer hoy en día que era una edad extraordinariamente temprana para acceder a la universidad. Aunque es cierto que era bastante joven, hubo otras personas que entraron a esta temprana edad. Allí estudió filosofía, la cual le fue bien enseñada, y mate-

Figura 2.17: Ciudad de nacimiento de Leibniz ubicada en la Alemania actual

Figura 2.18: Escuela Nicolai a la cual fue enviado Leibniz a la edad de 7 años

máticas, las cuales no se le enseñaron tan bien. Durante estos dos años de grado, también aprendió retórica, la cual es el arte del discurso, latín, griego y hebreo. Se graduó con su título de grado en 1663, con su tesis *De Principio Individui*, el principio de lo individual, la cual enfatizaba en el valor existencial del individuo. Podemos ver la portada de dicha tesis en la Figura 2.19 [3].

En 1663, con la tesis terminada, se fue a Jena, donde, a través del profesor de matemáticas y filósofo Erhard Weigel, Leibniz empezó a comprender la importancia del método matemático para probar cosas tales como la lógica y la filosofía. Dicho profesor pensaba que los números eran el concepto fundamental del universo y que a través de ellos se podía explicar cualquier cosa. Estas ideas hicieron mella en Leibniz.

[3]Imagen extraída de: http://digital.slub-dresden.de/werkansicht/dlf/5292/1/

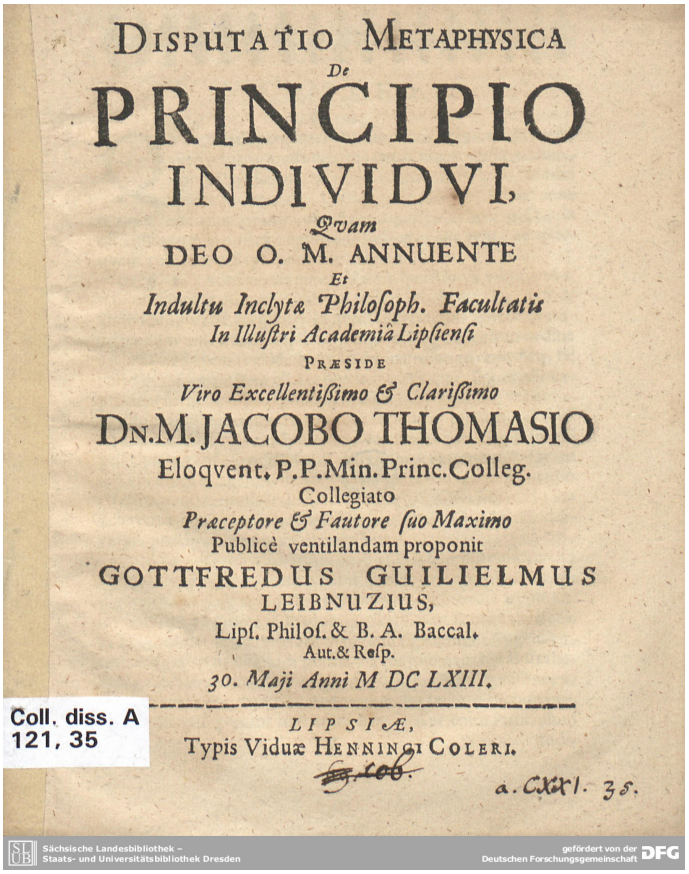

DISPUTATIO METAPHYSICA
De
PRINCIPIO
INDIVIDVI,
Quam
DEO O. M. ANNUENTE
Et
Indultu Inclytæ Philoſoph. Facultatis
In Illuſtri Academiâ Lipſienſi
PRÆSIDE
Viro Excellentiſsimo & Clariſsimo
DN. M. JACOBO THOMASIO
Eloqvent. P.P. Min. Princ. Colleg.
Collegiato
Præceptore & Fautore ſuo Maximo
Publicè ventilandam proponit
GOTTFREDUS GUILIELMUS
LEIBNUZIUS,
Lipſ. Philoſ. & B. A. Baccal.
Aut. & Reſp.
30. Maji Anni M DC LXIII.
LIPSIÆ,
Typis Viduæ HENNINGI COLERI.
Coll. diss. A
121, 35
a. CXXI. 35.
Sächsische Landesbibliothek – Staats- und Universitätsbibliothek Dresden
gefördert von der Deutschen Forschungsgemeinschaft DFG

Figura 2.19: Portada de la tesis de Leibniz *De Principio Individui*

En octubre de 1663, Leibniz había vuelto a su ciudad natal, donde empezó sus estudios de doctorado en leyes, donde, al terminar, fue homenajeado con su Grado en Máster de filosofía después de realizar su conferencia, la cual combinaba temas filosóficos y estudios de leyes con ideas matemáticas que aprendió de Wiegel. Lamentablemente, pocos días después de su discurso, su madre falleció, la cual se había encargado de él durante toda su vida y a la cual apreciaba mucho.

Después de obtener el título en leyes, Leibniz escribió un trabajo que fue publicado en 1666 como *Dissertatio de arte combinatoria*, Discurso sobre el arte de la combinatoria, cuya portada se puede aprecia en la Figura 2.20. En dicho trabajo, intentaba reducir todo el razonamiento y descubrimientos a elementos básicos como los números, letras, sonidos y colores.

A pesar de su reputación, se le rechazó su doctorado en leyes, la razón es desconocida aunque se presupone que es porque era demasiado joven y se le pedía que esperase un año. Este retraso no pudo esperarlo y se fue a la Universidad de Altdorf, donde recibió su doctorado en leyes en 1667.

Durante los próximos años, Leibinz tuvo que viajar mucho, de trabajo en trabajo, e intentó llevar varios proyectos a cabo, todos ellos políticos, científicos o literarios. También continuó su carrera de leyes, donde uno de sus trabajos era mejorar el código de leyes civil romano. Otras de sus metas en la vida era la de comparar todos los conocimientos humanos. En estos años, emprendió el estudio del movimiento que, al igual que Kepler, dijo que el movimiento dependía de la acción de un espíritu.

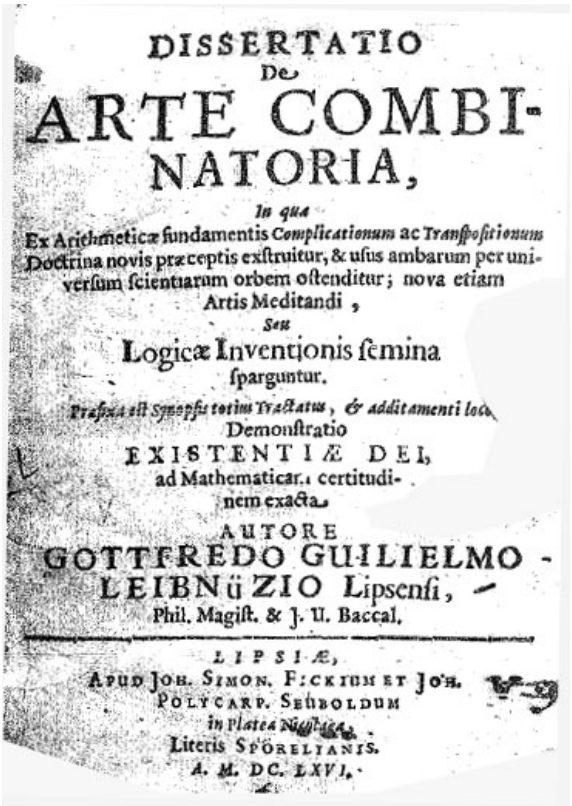

DISSERTATIO
De
ARTE COMBI-
NATORIA,
In qua
Ex Arithmeticæ fundamentis Complicationum ac Transpositionum
Doctrina novis præceptis exstruitur, & usus ambarum per uni-
versum scientiarum orbem ostenditur; nova etiam
Artis Meditandi,
Seu
Logicæ Inventionis semina
sparguntur.
Præfixa est Synopsis totius Tractatus, & additamenti loco
Demonstratio
EXISTENTIÆ DEI,
ad Mathematicam certitudi-
nem exacta.
AUTORE
GOTTFREDO GUILIELMO
LEIBNÜZIO Lipsensi,
Phil. Magist. & J. U. Baccal.
LIPSIÆ,
Apud Joh. Simon. Fickium et Joh.
Polycarp. Sehboldum
in Platea Nicolaitana.
Literis Sporelianis.
A. M. DC. LXVI.

Figura 2.20: Portada del trabajo de Leibniz *Dissertatio de arte combinatoria*

Con todo esto, Leibniz consiguió gran variedad de contactos científicos, pero no estaba satisfecho y deseaba visitar París para conseguir más. Entonces, empezó la construcción de una máquina de cálculo, la cual explicaremos en el siguiente capítulo con todo detalle, y que pensó que sería de gran interés por la sociedad. En 1672, consiguió viajar a París, donde su tarea principal era obtener contactos en el gobierno francés, pero, mientras esperaba para tal oportunidad, se dedicó a conocer a matemáticos y filósofos allí, en especial a Arnauld y Malebranch.

En 1673, visitó la Royal Society para mostrar su máquina inacabada, donde, después de algunos comentarios que no estaban muy a favor de su máquina, se dio cuenta de que no tenía tantos conocimientos de matemáticas como le hubiese gustado, así que, decidió doblar sus esfuerzos en la calculadora y prometió terminarla.

El 19 de abril de ese mismo año, la Royal Society lo aceptó como uno de ellos. Allá dentro, conoció a Huygens, el cual le dio una lista con trabajos realizados por Pascal, Descartes, etc. Con esto, Leibniz se centró en el estudio de la geometría infinitesimal, derivando en el descubrimiento por su parte del cálculo infinitesimal, aunque su nomenclatura por aquel entonces era un poco pobre, rompiendo la promesa de acabar su calculadora y perdiendo los favores de la Royal Society.

Más adelante, en el 1675, escribió un manuscrito usando la nomenclatura $\int f(x)dx$ por primera vez. Por otoño del año siguiente, había descubierto $d(x^n) = nx^{n-1}dx$ tanto para n entera como fraccionaria. Dicha nomenclatura se puede ver en la Figura 2.21, donde se muestra la función de Euler-Lagrange usando dicha nomenclatura.

$$\delta S = \int_{t_1}^{t_2} \left(\frac{\partial L}{\partial x^a} - \frac{d}{dt} \frac{\partial L}{\partial \dot{x}^a} \right) dt$$

Figura 2.21: Nomenclatura de Leibniz en la ecuación Euler-Lagrange

Cuando Newton se enteró de esto, le escribió una carta a Leibniz, la cual tardó un tiempo en llegarle. La carta mostraba varios de los resultados de los cálculos hechos por Newton, pero no sus métodos. Leibniz le respondió, pero como Newton no sabía que había tardado en llegarle, pensó que había tenido todo este tiempo para pensar la respuesta. Esta carta tuvo consecuencias en Leibniz, quien se apresuró en publicar sus descubrimientos y métodos.

Una segunda carta de Newton llegó a Leibniz escrita el 24 de octubre de 1676, la cual se demoró más que la anterior, llegando en junio del año siguiente. Esta carta, aunque estaba en tono educado, acusaba a Leibniz de haberle robado sus métodos así que, en la respuesta, Leibniz le dio alguunos detalles de los principios del cálculo diferencial que había usado incluyendo la regla para derivar funciones de funciones, la «regla de la cadena».

Esto indica que, aunque los dos descubrieron lo mismo, fue por separado y sin ayuda el uno del otro. Aunque Newton también fue descubridor del cálculo infinitesimal, en la actualidad se usa la nomenclatura y métodos de Leibniz, por estar mucho más depurados. Sin embargo, en los países anglosajones, prefirieron usar el método de Newton, rechazando a Leibniz, aunque sus métodos fuesen muy superiores.

A partir de finales de 1676, Leibniz permaneció en Hanover hasta su fallecimiento. Allí empezó otra gran variedad de proyectos, entre los cuales se encuentra la invención de un sistema para drenar agua de las minas usando el poder del agua y del viento. Sin embargo, este proyecto fracasó hacia 1684, aunque Leibniz había conseguido grandes mejoras en el ámbito de la geología a través de la observación, siendo el primero que lo hacía. Durante este tiempo con el proyecto formuló la hipótesis de que la Tierra había estado en un principio fundida.

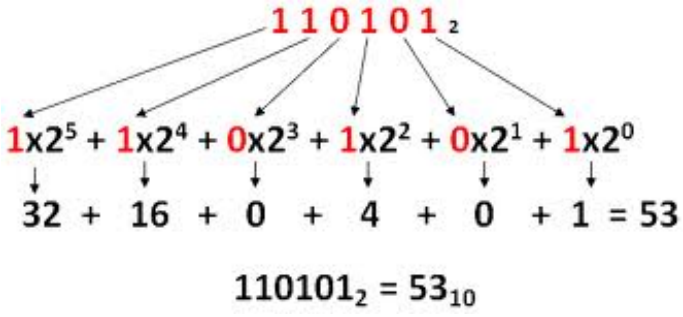


Figura 2.22: Transformación de binario a decimal del número 53 usando el método desarrollado por Leibniz

Otro de los grandes logros de Leibniz en las matemáticas fue el desarrollo del sistema binario, el cual empleamos hoy en día en las distintas máquinas electrónicas como pueden ser los ordenadores o *smartphones*, y que se convirtió en una

Figura 2.23: Manuscrito del desarrollo del sistema binario por Leibniz

de las mayores aportaciones de Leibniz, tanto a la informática como a las matemáticas. Perfeccionó este método por el año 1679, pero no publicó nada hasta el año 1701. En la Figura 2.22 [4] podemos ver una transformación de un número en binario a decimal, usando el método que perfeccionó, el cual aparece en el manuscrito de la Figura 2.23. También desarrolló un trabajo centrado en el cálculo con determinantes en ecuaciones lineales, el cual nunca mostró durante el tiempo que duró su vida, aunque desarrolló diferentes notaciones y aproximaciones para ver cuál resultaba más útil. Un escrito de enero de 1684 contiene resultados y notaciones bastante buenas para el uso de determinantes.

En 1684 publicó sus resultados sobre el cálculo diferencial en una revista creada en Leipzig dos años antes, llamada *Acta Eruditorum* y que podemos ver dicha portada en la Figura 2.24. El escrito contenía la familiar notación de la d que usamos hoy en día, así como reglas para el cálculo de derivadas de potencias, productos y cocientes. A pesar de esto, el escrito no contenía pruebas y Bernoulli prefirió llamarlo enigma antes que explicación.

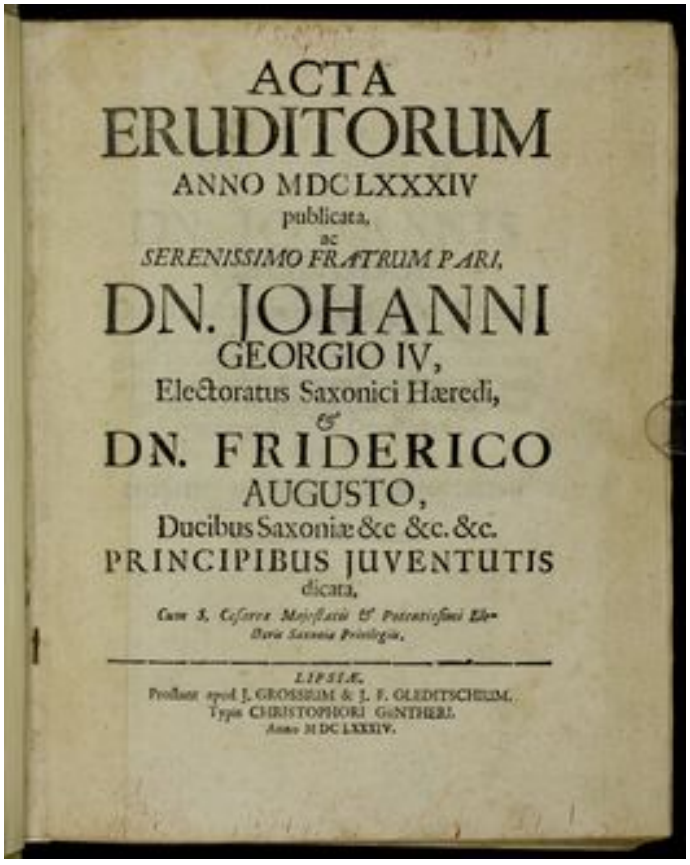

ACTA
ERUDITORUM
ANNO MDCLXXXIV
publicata,
ac
SERENISSIMO FRATRUM PARI,
DN. JOHANNI
GEORGIO IV,
Electoratus Saxonici Hæredi,
&
DN. FRIDERICO
AUGUSTO,
Ducibus Saxoniæ &c &c. &c.
PRINCIPIBUS JUVENTUTIS
dicata,
Cum S. Cæsareæ Majestatis & Potentissimi Electoris Saxoniæ Privilegio.
LIPSIÆ,
Prostant apud J. GROSSIUM & J. F. GLEDITSCHIUM,
Typis CHRISTOPHORI GUNTHERI.
Anno MDCLXXXIV.

Figura 2.24: Portada de la revista *Acta Eruditorum* de 1684

Más adelante, dos años después de la publicación del cálculo diferencial, en 1686, Leibniz publicó en la misma revista el cálculo de integrales, con la notación mostrada anteriormente de $\int$ y que aparecía por primera vez impresa. Por otra parte, un año después, Newton consiguió que le publicaran su trabajo, *Principia*, pues el mismo año de la publicación de Leibniz, se lo rechazaron. Dicho trabajo explicaba su propio método del cálculo de derivadas y que llevaba escrito desde

[4] Imagen obtenida de http://www.areatecnologia.com/sistema-binario.htm

1671. Esto desembocó en otra disputa entre Leibniz y Newton que explicaremos más adelante.

Los siguientes años, Leibniz gastó gran parte de su energía en promover las sociedades científicas, involucrándose en la creación de varias academias en Berlín, Viena, Dresden y San Petersburgo. Gracias a sus esfuerzos e insistencias, consiguió convertirse en el primer presidente de la Brandenburg Society of Sciences o Sociedad de Ciencias de Bradenburgo justo después de su fundación. Sin embargo, la academia no tuvo mucho éxito, lo que llevó a la creación de la academia de Berlín algunos años después.

Para su desgracia, sus intentos de promover la creación de otras academias fueron un fracaso, y fue nombrado director de una academia en Viena en 1712, aunque murió antes de que fuese creada y lo mismo ocurrió con otra academia en San Petersburgo.

Leibniz tuvo correspondencia con muchos de los eruditos de Europa de aquellos años, entre los cuales destaca el matemático Grandi, con el cual habló de los resultados obtenidos de sustituir $x = 1$ en $1/(1+x) = 1 - x + x^2 - x^3 + \ldots$. También con Varignon, con el cual habló sobre su teorema, y discutió con Bernoulli sobre logaritmos y números negativos.

También publicó algun trabajo sobre filosofía, en especial *Théodicée*, en el cual se discute el problema del mal en un mundo creado por un dios bueno donde el universo debe de ser imperfecto porque de otra manera, no sería distinto de Dios, instando que el universo es el mejor que puede haber siendo imperfecto.

La disputa pendiente entre Newton y Leibniz surge cuando Leibniz lee el escrito redactado por Keill en 1711, en el cual se le acusaba de plagio. Leibniz pidió que se retractaran de lo dicho, pues no hubo plagio, alegando de que no había escuchado hablar del cálculo derivativo hasta leer el trabajo de Wallis, pues en 1671 ya estaba completo el trabajo de Newton sobre este tema. A pesar de esto, Keill le contestó a Leibniz que las cartas que había enviado Newton decían lo contrario, que cómo si no había conseguido sacar esos principios de derivación si no habían sido de su trabajo. Sin embargo, todo esto no llegó a buen puerto y ganó Newton, escribiendo él mismo el informe. Este informe no fue visto por Leibniz hasta que Bernoulli le envió una carta en la que le comentaba que había encontrado un error en el trabajo de Newton en el método de derivación. Leibniz decidió escribir un panfleto anónimo llamado *Charta volans* en el que, el fallo de Newton en la comprensión de segundas derivadas o superiores, es usado como prueba de que Leibniz tenía razón. Se puede ver el panfleto entero en la Figura 2.25 donde expone lo dicho anteriormente.

Dicha discusión continuó con la respuesta de Keill al panfleto, aunque Leibniz decidió no continuar discutiendo con él. Sin embargo, cuando Newton decidió escribirle directamente, Leibniz contestó con una detallada descripción del cálculo diferencial que había usado y que, ahora, no eran simples detalles de su método.

Desde este momento hasta el día de su muerte, 14 de noviembre de 1716 en Hanover, Leibniz tuvo correspondencia por carta con Samuel Clarke, un defensor de Newton.

2.3.1. Blaise Pascal

Blaise Pascal (1623-1662), al cual podemos ver en el retrato de la Figura 2.26, fue un matemático, físico, filósofo cristiano y escritor francés. Gran parte de sus contribuciones al mundo de las matemáticas y de la historia natural incluyen el diseño y construcción de máquinas mecánicas para el cálculo, contribuciones a la teoría de la probabilidad, indagaciones sobre los fluidos así como la simplificación de conceptos como pueden ser la presión y el vacío. En los últimos años de su vida, dejó de lado las matemáticas y la física para centrarse en la filosofía y la teología.

Figura 2.26: Retrato de Blaise Pascal

Hablamos de Pascal pues fue una gran influencia para Leibniz, sobre todo en el campo de las calculadoras mecánicas, el cual, como veremos más adelante, basó gran parte de su diseño en la máquina creada por Pascal, la Pascalina, que ya se ha discutido en una sección precedente.

2.3.2. Isaac Newton

Isaac Newton (1642-1727), al cual podemos ver en el retrato de la Figura 2.27, dedicó gran parte de su vida a la física, filosofía, teología, alquimia, matemáticas y también fue inventor. Es autor del *Philosophiæ naturalis principia mathematica* como hemos comentado anteriormente, en el que se describe la ley de la gravitación universal y se establecen las bases de la mecánica clásica a través de sus propias leyes. Pero no solo se centran sus trabajos en la mecánica clásica, sino que además, sus otros descubrimientos científicos tratan sobre la naturaleza de la luz y la óptica y el desarrollo del cálculo matemático, el cual, como hemos visto anteriormente, le llevó a una disputa con Leibniz.

Entre sus mayores hallazgos se encuentra el descubrimiento de que la luz de color proveniente de la luz blanca cuando cruza un prisma, viene dada por la pro-

pia luz y no por el prisma como se pensaba hasta entonces, el cual solo se encarga de separar los distintos colores del haz de luz. Además, argumentó la posibilidad de que la luz que percibimos estuviese compuesta por partículas. Desarrolló la ley de la convección térmica, la cual habla sobre la tasa de enfriamiento de los objetos expuestos al aire.

Una de los principales descubrimiento que hacen que Newton sea considerado el mayor científico de todos los tiempos fue la demostración de que las leyes naturales que gobiernan el movimiento de los objetos en la Tierra son las misma que las que rigen los movimientos de los cuerpos celestes.

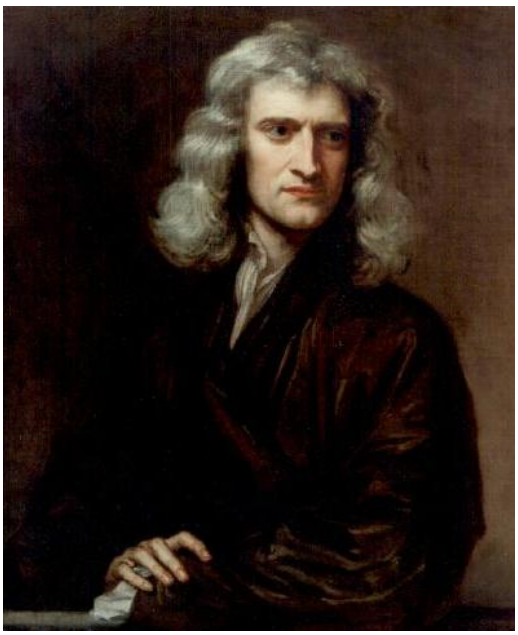


Figura 2.27: Retrato de Isaac Newton

El motivo por el cual tiene una sección propia en este trabajo es porque, junto a Leibniz, es el descubridor y desarrollador del cálculo integral y diferencial, los cuales usó para formular sus leyes físicas.

2.3.3. René Descartes

René Descartes (1596-1650), al cual podemos ver en el retrato de la Figura 2.28, fue un gran filósofo, matemático y físico francés. Se le considera como el padre de la geometría analítica y de la filosofía moderna, así como uno de los precursores destacados de la revolución científica.

En el campo de la filosofía, en el que destacaba, hizo famoso el principio *cogito ergo sum*, el cual significa «pienso, luego existo» y se convirtió en un elemento esencial del racionalismo cartesiano.

Figura 2.28: Retrato de René Descartes

Como ocurre con las anteriores figuras científicas, Descartes aportó a Leibniz el racionalismo cartesiano, el cual llevó consigo toda su vida filosófica, convirtiéndose en uno de los grandes pensadores filosóficos del siglo XVII.

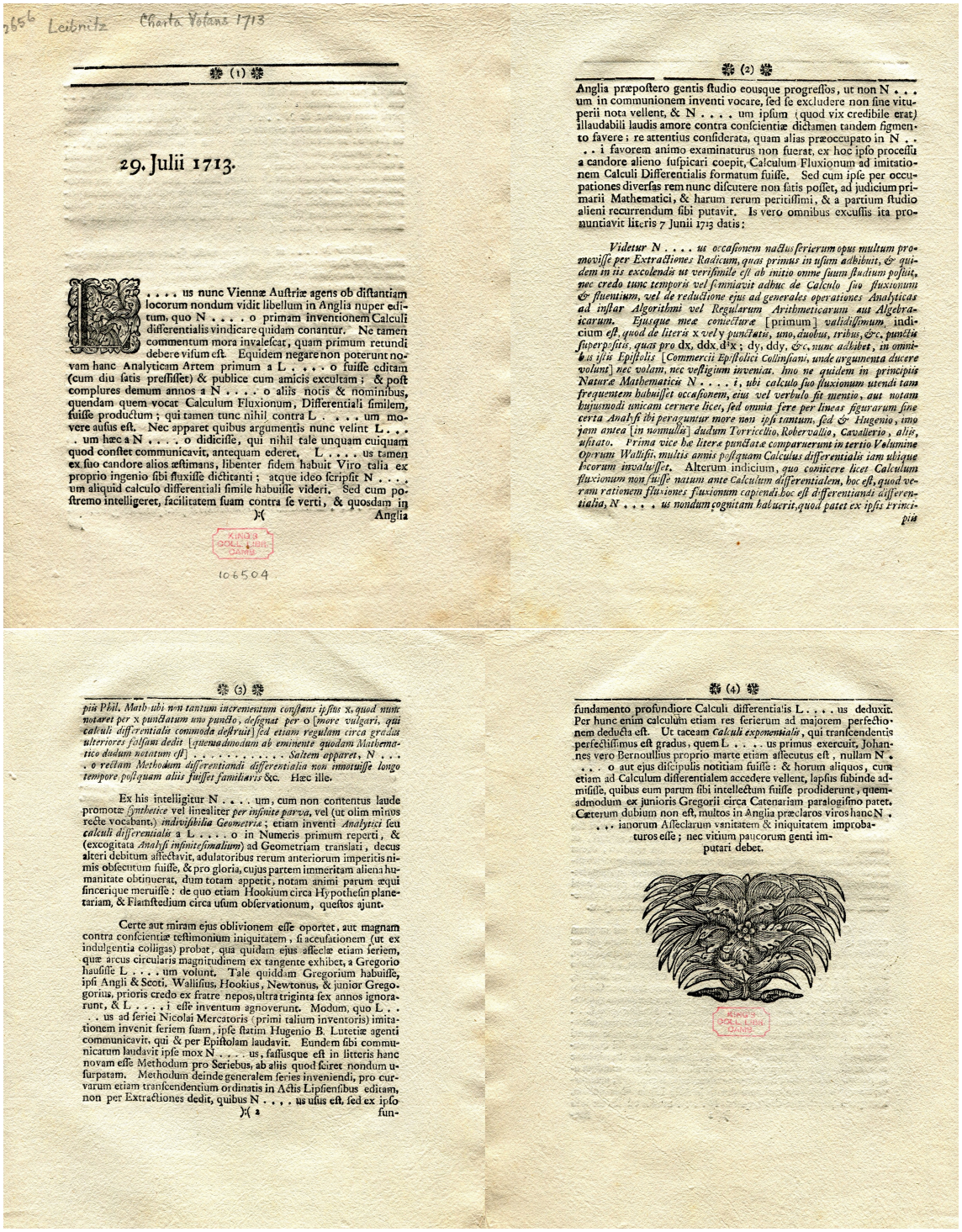


Figura 2.25: Panfleto *Charta volans* completo escrito por Leibniz

CAPÍTULO 3
Mecanismo y funcionamiento de la calculadora Leibniz

El gran matemático Gottfreid Leibniz, inventor de esta calculadora, fue uno de los primeros hombres que soñó con una máquina con pensamiento lógico. De esta forma, Leibniz intentó combinar los principios de la aritmética con los de la lógica imaginando la su dispositivo como algo más que una calculadora, una máquina lógica y pensante. Esto le llevó a idear un sistema que llevaría al cómputo a un escalón que permitiría su cálculo de una forma mucho más sencilla. A este sistema le llamó sistema binario, pues solamente usaba dos dígitos diferentes para representar cualquier número imaginable.

En uno de sus tratados, el cual podemos apreciar en la Figura 3.1, describe cómo una calculadora es capaz incluso de trabajar con este sistema, el binario. Esta máquina sería capaz de funcionar sin ruedas ni cilindros, solamente bolas, agujeros, palos y canales por los cuales serían transportadas las bolas. Según describe en el escrito, la máquina sería capaz de funcionar abriendo y cerrando los agujeros, donde un agujero abierto representaría un 1 y uno cerrado un 0. A través de los agujeros abiertos, se dejarían caer las bolas sobre unos raíles.

Leibniz soñaba con inventar una máquina universal para resolver problemas además de un lenguaje universal para poder discutir sobre problemas de metafísica u otras ramas de la misma manera que se resuelven los problemas matemáticos, sin la necesidad de aprender más que un solo lenguaje.

Afortunadamente para nosotros aunque no para él, sus impresionantes ideas y proyectos se han podido llevar a cabo, pero no por él, sino por Norbert Wiener, creador de la cibernética, después de dos siglos y medio.

Una vez dicho esto y los sueños que tenía Leibniz que no pudo llevar a cabo, vamos a empezar a hablar sobre la calculadora que sí consiguió construir y en la que se basa este proyecto.

3.1 Primeros diseños

Como hemos visto, Leibniz siempre había tenido el sueño de construir una máquina pensar por sí sola, pero es alrededor de 1670 o 1671, cuando le surgió la idea de construir una máquina de cálculo al ver un podómetro, el cual sirve para

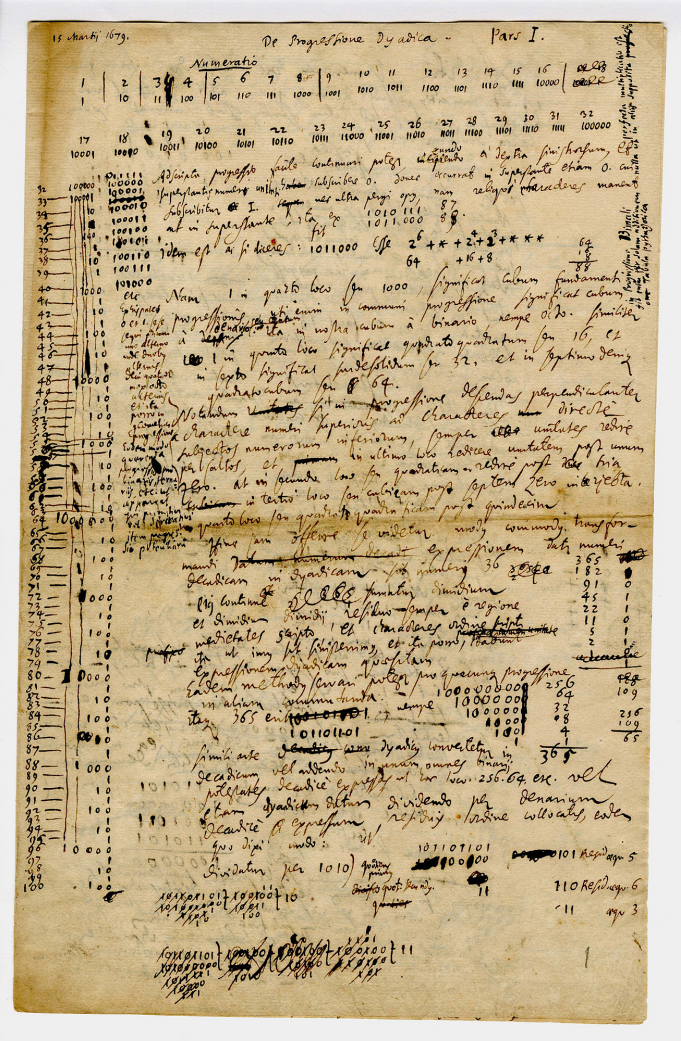


Figura 3.1: El tratado *De progressione Dyadica* en el cual Leibniz describe el cálculo binario de una máquina

calcular la cantidad de pasos que da una persona a través del movimiento de sus caderas al andar. Sin embargo, no fue hasta el año 1672, cuando viajó a París y tuvo la suerte de tener acceso a dos de los escritos que no se habían publicado de dos grandes filósofos, Descartes y Pascal. Es entonces, cuando leyendo el escrito de Pascal sobre la calculadora mecánica que desarrolló, la Pascalina, decidió que podía mejorarla para que no solamente fuese capaz de hacer sumas y restas, sino que además pudiese multiplicar y dividir.

3.1.1. Primer mecanismo

Al principio, pensó en usar un mecanismo similar al que usó Pascal en su calculadora, pero rápidamente se dio cuenta de que para realizar multiplicaciones y divisiones era necesario un mecanismo completamente diferente al que se había usado. Este nuevo mecanismo haría posible que el multiplicando y el dividendo solo se introdujesen una vez y que por medio de una manivela apareciese el resultado, a diferencia del mecanismo empleado por Pascal como hemos visto en el capítulo anterior.

Intentando encontrar una buena manera de solucionar el problema mecánico enunciado anteriormente y antes de inventar su famosa rueda escalonada, hizo varios proyectos. Uno de los bocetos que empleó en un proyecto se puede ver en la Figura 3.2, el cual se data allá por el año 1685.

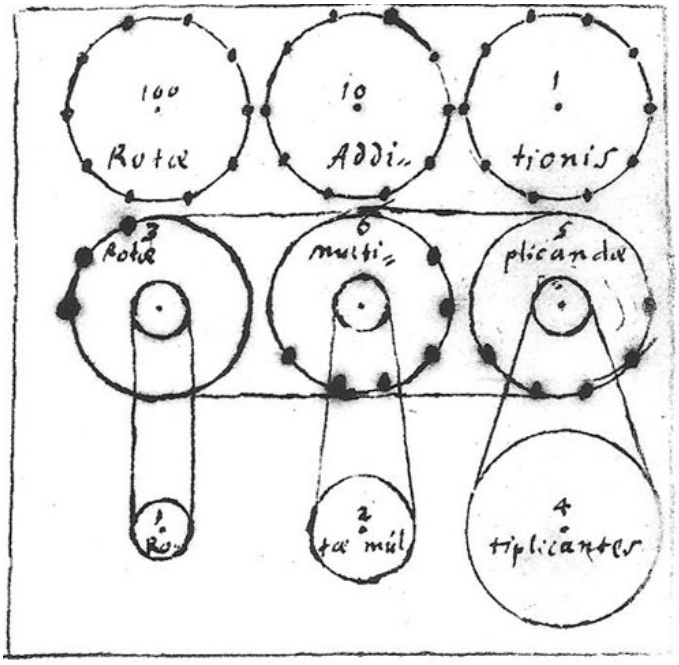


Figura 3.2: Uno de los primeros bocetos del mecanismo de la calculadora de Leibniz anterior a la rueda escalonada

En el boceto de la Figura 3.2, se puede apreciar un mecanismo de entrada de datos, las ruedas o círculos de la parte inferior, en lo cuales se puede ver escrito *Rota multiplicantes*, donde se entiende que deberían de introducirse los multiplicadores. También se observa un mecanismo de cálculo, en el cual se ve escrito *Rota multiplicando* y se encuentra en las ruedas centrales del boceto y donde deberían introducirse los multiplicandos. Para finalizar, se ve el mecanismo de resultado, donde se aprecia escrito *Rota Additionis* y que corresponden a los círculos superiores, donde aparecería el resultado de la multiplicación. El movimiento se transmitía de las ruedas de entrada a las ruedas de cálculo a través de cadenas. Este mecanismo aún no se basaba en la rueda escalonada sino en una especie de molinillo.

Este mecanismo de molinillo se puede ver en la Figura 3.3 y es descrito en otro manuscrito. Este en particular es un mecanismo anticuado, que en 1709 fue reinventado por Giovanni Poleni y mejorado posteriormente por Braun, Baldwin y Odhner, del cual hablaremos más adelante. En el boceto se puede ver escrito *Dens mobile d'une rour de Multiplication* o lo que es lo mismo, «dientes móviles de una rueda del multiplicador».

Como se puede ver, esto es un prototipo y los primero diseños de la calculadora estaban basados en dicho mecanismo de molinillo, aunque nunca se llegaron a construir porque, en 1671, inventó un nuevo mecanismo llamado rueda escalonada que usaría para los siguientes diseños y del cual hablaremos más adelante en este capítulo.

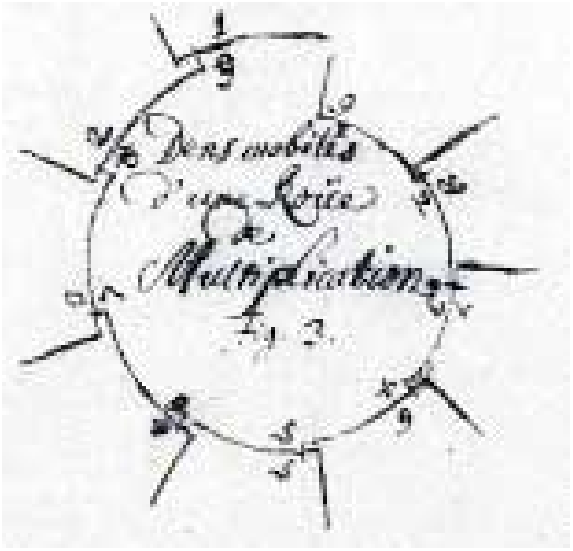


Figura 3.3: Mecanismo de molinillo

3.1.2. Uso de la rueda escalonada

Con todo esto en mente, empezó a construir su primer prototipo de la máquina calculante, basada ya, en la rueda escalonada, pero pronto se tuvo que enfrentar a los mismos problemas que experimentó Pascal con la Pascalina. Estos problemas eran bastante recurrentes en el desarrollo de máquinas mecánicas, pues en la época se hacían a mano y era muy complicado conseguir unos mecanismos que funcionasen de manera fina y fluida.

El primer prototipo que construyó era de madera y solo contaba con dos dígitos, al cual decidió llamarle *Instrumentum Arithmeticum*. Este prototipo fue construido pronto y decidió, a finales del 1672 y principios de 1673, mostrarlo a sus compañeros de la Academia Francesa de Ciencias y al ministro de finanzas Jean-Baptiste Colbert.

En enero de 1673, fue enviado a Londres con una misión diplomática, la cual aprovechó para llevar su calculadora y mostrarla a la Royal Society. Como se ha comentado en el capítulo anterior, la calculadora no tuvo éxito y prometió mejorarla después de volver a París.

En una carta del 26 de marzo de 1673 dirigida a Johann Friedrich, describe el propósito de su máquina aritmética, indicandoe que es para hacer los cálculos fáciles, rápidos y que fuesen de confianza, pues el cálculo a mano conlleva errores. También citó que teóricamente era capaz de calcular números tan grandes como se quisiese, siempre y cuando el tamaño de la calculadora lo permitiese.

De vuelta en París, Leibniz alquiló los servicios de un habilidoso mecánico, Oliver, el cual era un buen artesano que era lo que necesitaba para perfeccionar el mecanismo de la calculadora, pues él no era muy bueno con los trabajos manuales. Oliver terminó el primer prototipo hecho con latón, el cual parece ser que fue el primer dispositivo fiable que estuvo disponible hasta 1685, momento en el que despareció y no aún no se ha encontrado. El segundo prototipo de la máquina fue hecho entre 1686 y 1694.

El primer prototipo fue presentado en 1675 en la Academia Francesa de Ciencias y fue muy bien aceptado, tanto por Antoine Arnauld como por Christian Huygens, lo cuales eran grandes prominencias en dicha academia. Leibniz estaba tan contento de su invento que inmediatamente informó a una de sus correspondencias por carta, Thomas Burnett, de que había conseguido construir una máquina aritmética mucho mejor que la de Pascal, pues era capaz de realizar multiplicaciones y divisiones.

Más adelante, se sabe que Leibniz pidió la construcción de nuevas calculadoras. Aunque no se sabe el número exacto, se presupone que construyó alrededor de diez por la cantidad de dinero que invirtió en dichas construcciones. En la Figura 3.4 podemos ver una de las primeras calculadoras que Leibniz pidió hacer. Una de las primeras máquinas construidas, probablemente la tercera, creada entre 1690 y el 1720, fue guardada en el ático de un edificio de la Univeridad de Göttingen en algún momento de los años 1770, donde fue olvidada. Dicha calculadora permaneció perdida hasta 1879, cuando un grupo de gente se la encontró escondida en una esquina cuando intentaban tapar unas goteras del tejado.

Figura 3.4: Una de las primeras calculadoras construidas de Leibniz

Entre 1894 y 1896, Arthur Burkhardt restauró la calculadora encontrada en el ático y la entregó al *Niedersächsische Landesbibliothek*, actual Biblioteca Gottfried Wilhelm Leibniz, y que ha permanecido allí durante todo este tiempo. En la actualidad existen dos de estas máquinas, y varias réplicas, una de las cuales se puede ver en la fotografía de la Figura 3.5.

3.2 Mecanismo de la calculadora

Como se ha venido hablando durante este capítulo, la gran revolución que le dio el renombre a la calculadora de Leibniz fue la invención el la rueda escalonada o *Stepped Reckoner* en inglés. En esta sección vamos a hablar de ella y del mecanismo completo de la calculadora, haciendo hincapié en las distintas operaciones que es capaz de hacer y cómo las hace.

Para empezar hablaremos del mecanismo de la rueda escalonada, el cual podemos ver en la Figura 3.6, dando detalles de cada una de las partes que lo com-

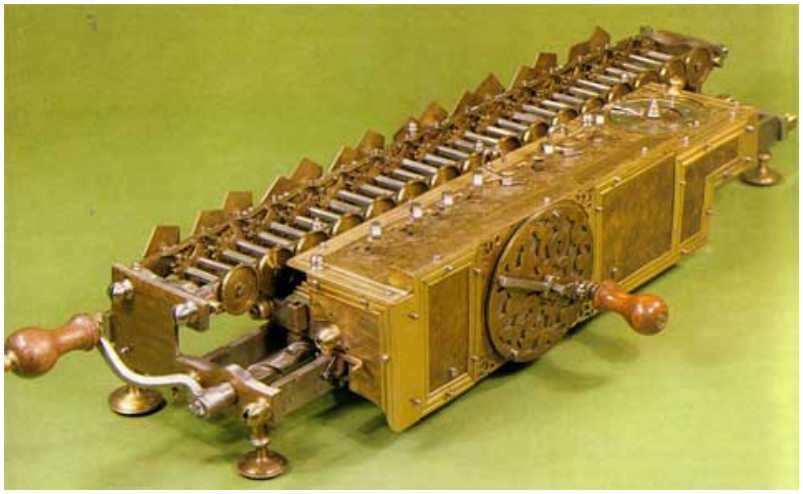


Figura 3.5: Réplica de la calculadora encontrada en el ático de la Univeridad de Göttingen que se encuentra en la Biblioteca Gottfried Wilhelm Leibniz

ponen así como de su funcionamiento. Como podemos ver, los distintos componentes que forman el mecanismo de la rueda escalonada están marcados con diferentes letras. Para guiarnos en la explicación, usaremos dichas letras para referirnos a cada una de estas partes. Empezando por la rueda numerada que vemos con la letra *D*, podemos decir que es la rueda encargada de la entrada, es decir, el usuario va a ver esta rueda y al girarla para seleccionar el dígito, accionará el engranaje marcado con la letra *E*, encargado de transformar el giro de *D* en un movimiento lineal. Dicho engranaje, al girar, moverá la varilla metálica con el nombre de *M*, la cual está unida a la rueda dentada que en este caso es *S*. Al girar *D*, todo el mecanismo nombrado anteriormente hace que *S* se desplace hacia delante o hacia atrás, dependiendo del sentido de giro que le demos a *D*.

Con esto ya tenemos explicado el mecanismo de entrada y como se transforma un tipo de movimiento en otro permitiendo el desplazamiento interno de la rueda escalonada.

La rueda escalonada está compuesta por un cilindro, al cual le sobresalen una especie de escalones, de ahí su nombre, un total de nueve. Estos escalones son de diferente longitud. En la Figura 3.7 se aprecian mejor las distintas longitudes de los escalones. Los escalones y surcos se utilizan para que, al girar la manivela principal, el giro se transmita al engranaje *F* dependiendo del valor de entrada de *D*. De tal manera, si introducimos un 0, el mecanismo estaría lo más al frente posible, haciendo que *F* no tocase ningún escalón y por tanto, no transmita el movimiento a la rueda *R* donde están dibujados los distintos dígitos que se muestran a través de *P*, y por tanto, no se incremente o decremente el valor de esa posición. En cambio, si introducimos un 9, el mecanismo estaría lo más alejado posible, haciendo que *F* cruce por todos los escalones y por tanto incremente o decremente el valor mostrado en *P* en 9.

La entrada tiene ocho de estos mecanismos como se puede ver en la Figura 3.8, donde en la parte frontal se aprecia la manivela principal a la que Leibniz bautizó como *Magna Rota*, y es la que se encarga de realizar las distintas operaciones de suma o resta, dependiendo del sentido de giro. Justo encima están los nombrados mecanismos de entrada y se pueden apreciar ocho, como se ha comentado

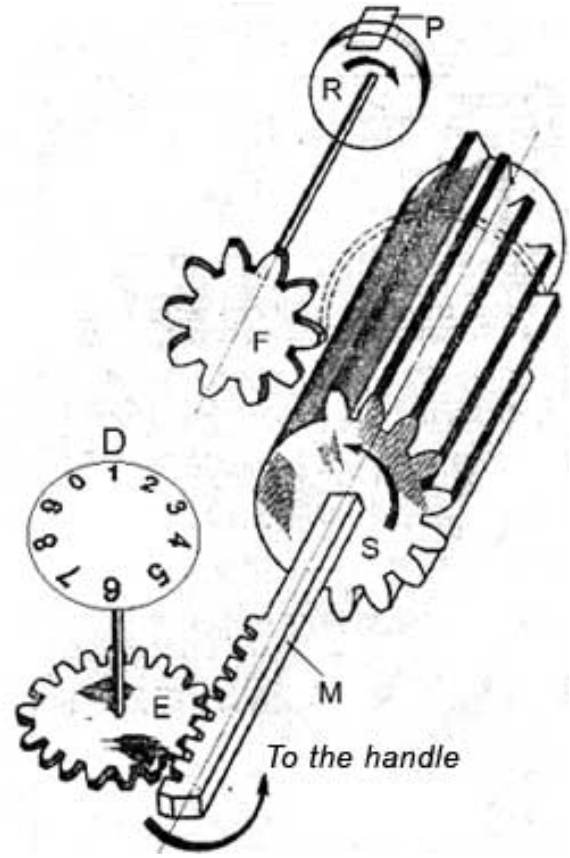


Figura 3.6: Mecanismo del *Stepped Reckoner*

anteriormente. Cada uno de estos mecanismos de la rueda escalonada, harán un giro cuando se accione la manivela, y dependiendo de la posición en la que se encuentre el cilindro escalonado, se incrementarán o disminuirán tantas unidades como escalones hagan girar el engranaje *F*.

En cuanto a la salida, podemos apreciar que estaba compuesta por dieciséis dígitos, lo que permitía multiplicaciones de números de ocho cifras por otro número de ocho cifras sin que hubiese desbordamiento. El mecanismo completo de la calculadora estaba compuesto por una gran cantidad de engranajes de diferentes tamaños y formas. Dicho mecanismo se puede ver en la Figura 3.9, de cual vamos explicar su funcionamiento a continuación, partiendo de lo ya explicado anteriormente del cilindro escalonado.

Uno de los mayores problemas que tenía el mecanismo de la Figura 3.9, a parte de que estaba construido de forma manual, era que el mecanismo encargado de los acarreos no era totalmente automático, al menos en las calculadoras que han sobrevivido hasta la actualidad. Como se ha indicado previamente, Leibniz creó varias de ellas y se piensa que solventaría este problema en las más nuevas, aunque no se sabe con total seguridad pues no han sobrevivido hasta la actualidad o aún no han sido encontradas.

Nos vamos a centrar en explicar el funcionamiento de este conjunto de engranajes que componen el movimiento de dos dígitos consecutivos y que, por tanto, incluye el acarreo. Fijémonos en la Figura 3.9, que al igual que ocurría con la anterior, viene marcada con números para que podamos identificar a qué nos estamos refiriendo cuando hablemos de los distintos componentes. Una vez dicho

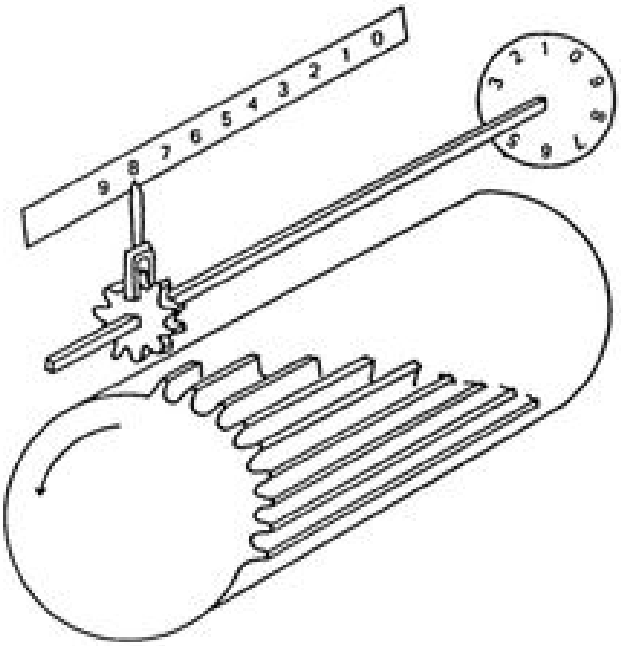


Figura 3.7: Cilindro escalonado

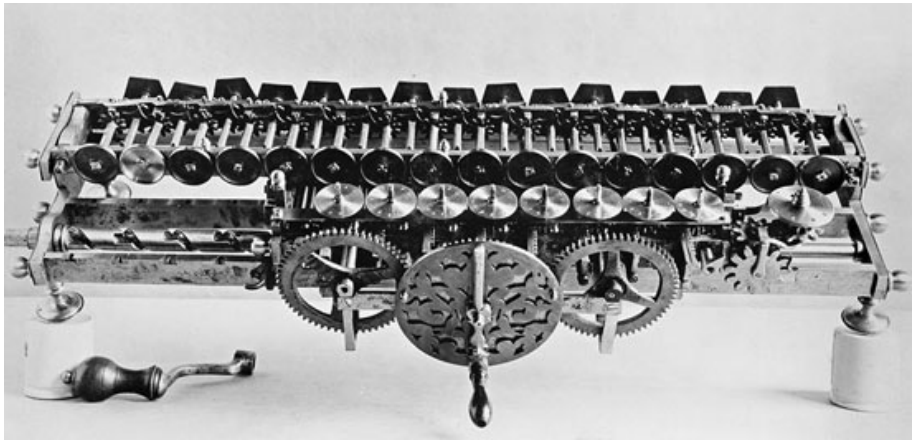


Figura 3.8: Calculadora destapada

esto, los cilindros escalonados de los que hemos hablado antes, vienen marcados en este caso con el número *6*, para que podamos centrarnos en las cosas que tiene alrededor. El acarreo no era totalmente automático y estaba formado por los engranajes *10*, *11*, *12*, *13* y *14*.

Cuando se debía producir un acarreo, la barra numerada con un *7*, la cual se puede ver a la derecha de la imagen, accionaba el engranaje con forma de estrella *8*, haciendo girar la barra donde se encuentra la rueda *11*, también con forma de estrella, que haría girar una posición al *10*. En el eje donde se encuentra este *10*, se observa otra barra numerada con un *12*, la cual se encuentra hacia el centro de la figura. El movimiento del *10* hace moverse el *12*, moviendo a su vez una posición la rueda estrellada *11* que se encuentra en el siguiente dígito, incrementado dicho valor en 1 en la rueda *13* y mostrándose en el *15*. Así se produce el acarreo, y como puede parecer un poco complicado a primera vista, en la imagen de la Figura 3.10

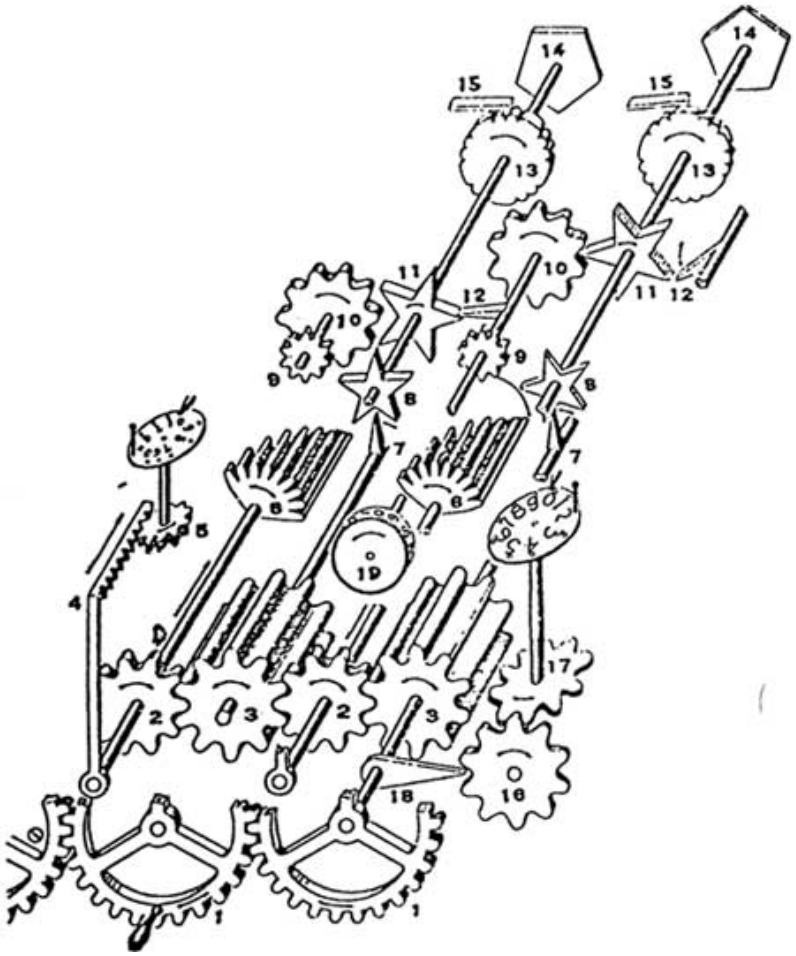


Figura 3.9: Mecanismo completo de la calculadora de Leibniz para dos dígitos consecutivos

se muestra el recorrido que se ha tomado así como las direcciones de giro de las distintas ruedas.

Sin embargo, llegados a este punto, no se pueden producir más acarreos para esta posición. Esto lo podemos ejemplificar de la siguiente manera, pero puede producirse con cualquier valor. Si el dígito al cual se le hace un acarreo está en un valor de 9 y, a través del acarreo, pasa al valor 0, dicho dígito no podría volver a sufrir un acarreo. Para solventar este problema, Leibniz colocó unos discos en forma de pentágono que se pueden ver tanto en la Figura 3.9 como en la Figura 3.10, los cuales vienen numerados con un *14*.

Este pentágono tenía dos posibles posiciones. La primera de ellas es con un lado en la parte alta, de tal forma que si mirásemos desde arriba de la máquina, no veríamos nada. Por otro lado, tenemos el segundo caso, en la que el pentágono se queda con un vétice en la parte superior, el cual sobresaldría por encima de la máquina y podría verse. El primero de los casos es indicativo de que se ha producido correctamente el acarreo, es decir, ha habido un acarreo automático y no se precisa la intervención del usuario. Sin embargo, para el segundo caso, se precisa la intervención humana. Esta intervención consiste en producir el acarreo de forma manual, girando con el dedo manualmente dicho pentágono en la dirección correcta, dependiendo de si es suma o resta, hasta que se deje de percibir

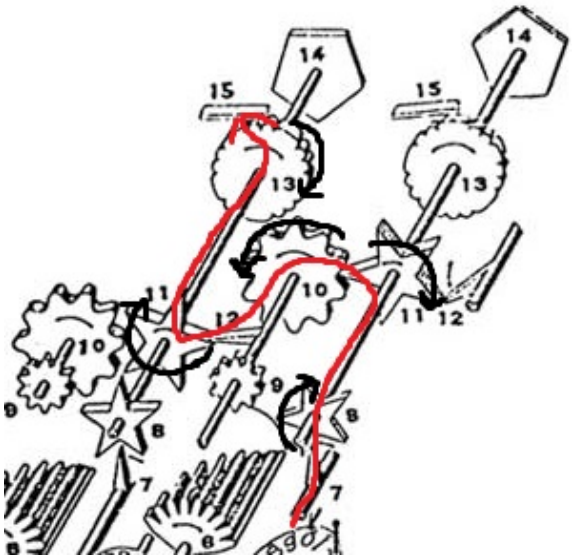


Figura 3.10: Camino recorrido en el acarreo

que sobresale. En la Figura 3.11, podemos apreciar mejor ambas posiciones, siendo la primera de ellas la que corresponde al pentágono izquierdo y la segunda al derecho.

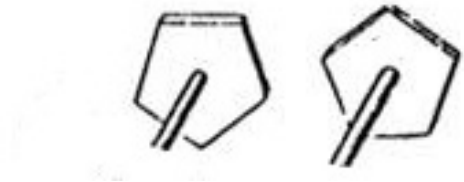


Figura 3.11: Posibles posiciones de los pentágonos del mecanismo de acarreo, izquierda cuando hay acarreo automático y derecha cuando hay acarreo manual

En cuanto a la *Magna rota* o manivela principal, su mecanismo se puede ver mejor en la Figura 3.9, donde está señalizada con un *1*, conectando directamente con los engranajes *2* y *3*, los cuales se encargaban de transmitir el movimiento a los distintos cilindros escalonados. Así pues, con el giro de una sola manivela, podíamos hacer girar todas los mecanismos de entrada.

Viendo el boceto, podemos apreciar otro mecanismo separado de los demás, el que se encuentra a la derecha. Este mecanismo era el encargado de la *Rota magnuscula* o «rueda mayúscula», la cual era necesaria tanto para multiplicaciones como para divisiones. Estaba formado por los engranajes *16* y *17*, que solo se encargaban de transmitir el movimiento, y la púa situada en el eje de la rueda derecha de la *Magna rota* con un *18*, la cual hacía girar al *16* una posición cada vuelta completa de la manivela principal.

Hasta este punto, tenemos explicado el mecanismo interno de la calculadora, tanto la parte del famoso cilindro escalonado como la parte encargada del acarreo.

Ahora bien, no solo se compone del mecanismo interno, sino que la parte externa también tiene sus partes móviles y las vamos a explicar a continuación. Para tener una visión más clara de lo que vamos a hablar, usaremos el boceto de la Figura 3.12.

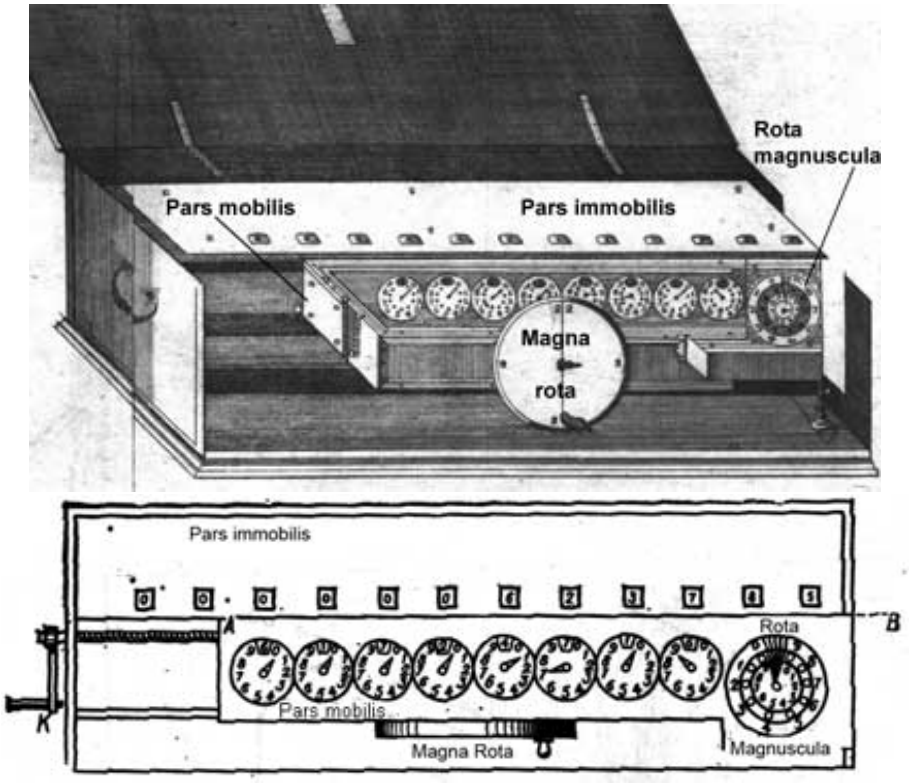


Figura 3.12: Boceto del exterior la calculadora hecho por Leupold y vista superior de la misma

En el boceto nombrado anteriormente, podemos ver que la calculadora está compuesta por dos partes. Por un lado, tenemos la parte superior a la que Leibniz llamó *Pars immobilis*, y se encuentra fija durante cualquier operación. Por otro lado, la parte inferior llamada *Pars mobilis* y que como su nombre indica, es la parte móvil de la máquina.

En la *Pars mobilis* se encuentran las ocho ruedas de entrada, así como el mecanismo compuesto por los cilindros escalonados vistos anteriormente. También tenemos la *Magna rota*, la cual está conectada a las distintas cifras o posiciones de cilindros escalonados. Por otro lado, tenemos a la parte derecha de la *Pars mobilis* la *Rota magnuscula*, necesaria para multiplicaciones y divisiones como hemos dicho con anterioridad. Esta parte completa se puede mover en dos direcciones, tanto a la derecha como a la izquierda, desplazándose a través de unos raíles mediante el uso de una manivela situada a la izquierda, pudiéndose apreciar mejor en la imagen inferior de la Figura 3.12 con el nombre *K*. El movimiento se producía por el efecto «tornillo» al girar *K*.

La *Pars immobilis* estaba compuesta simplemente por el mecanismo de acarreo, el cual es la parte superior de la Figura 3.9, a partir del cilindro escalonado con el número *6*, y las distintas endiduras para mostrar los números y los distintos pentágonos del acarreo. Cabe destacar que estos bocetos son simplificaciones de la máquina original y que por tanto, solo tienen doce dígitos en la salida en vez de dieciséis como la original.

La parte externa del mecanismo de la *Rota magnuscula* estaba compuesta por tres partes. Tenía dos anillos inmóviles, más concretamente el exterior y el que se encontraba más al interior. El exterior tenía escritos los dígitos en negro mientras

que el otro en rojo. Los que estaban en negro se usaban para las multiplicaciones y en rojo para las divisiones. Ambos anillos estaban numerados con las cifras del 0 al 9, el interior en sentido horario y el exterior en sentido anti-horario. En cuanto a la tercera parte, el anillo central que estaba entre los dos anteriores, estaba perforado, con diez agujeros. Estos servían para introducir una varilla y, del mismo modo que funcionaba un reloj antiguo con dial, servía de tope cuando dicha varilla alcanzaba la posición superior del anillo, habiendo finalizado la operación.

Hasta aquí la explicación del mecanismo, tanto interno como externo de la calculadora. Ahora pasaremos a explicar las distintas operaciones aritméticas que era capaz de hacer además de cómo se hacían.

3.3 Sumas y restas

Para hacer una suma era necesario primeramente introducir en las ruedas de entrada los diferentes dígitos que componían uno de los dos números a sumar y posteriormente, girar la manivela principal en el sentido de suma. De esta manera, tendríamos en los registros de salida el valor de dicho número. Como solamente hemos introducido el valos sin hacer ningún cálculo, no se ha producido ningún acarreo.

Ahora que ya tenemos el primer valor, introducimos el siguiente número en las ruedas de entrada y accionamos la manivela principal en el sentido de suma. Si se produce acarreo automático tendremos nuestro resultado en la *Pars immobilis* en los registros de salida. En caso contrario, deberemos mover los pentágonos que sobresalgan, comentados anteriormente, en el sentido de la suma. De esta manera ya tendremos nuestro resultado de sumar dos valores en el registro de salida.

De igual manera que hemos sumado dos números, podemos sumar tres, cuatro, etc., tantos como queramos, siempre teniendo el ojo bien abierto y accionando el acarreo manual siempre que sea necesario o de lo contrario, nuestro resultado sería erroneo.

Para la resta, viene a ser un procedimiento similar. En este caso, una vez tengamos en la salida el valor del que queremos restar, introducimos el otro número y accionamos la manivela en el sentido contrario al de la suma. El resultado puede verse directamente en la salida si no hay que accionar el acarreo manual, el cual se hace de igual forma que en la suma pero girando los pentágonos en sentido contrario.

Vamos a proceder a continuación con un ejemplo de una suma e iremos mostrando los distintos pasos mediante el uso de imágenes. Para este ejemplo usaremos una simplificación de la calculadora con tan solo cuatro registros de entrada y seis de salida.

En este caso, hemos decidido sumar 489 con 7 123. Primero decidimos cuál de los dos valores deseamos introducir primero. Para este ejemplo introduciremos primero el pequeño, 489, y giraremos la manivela para situarlo en el registro de salida como se puede ver en la imagen superior izquierda de la Figura 3.13.

Ahora que ya tenemos el primer valor, introducimos el segundo, 7 123, y accionamos la manivela en el sentido de suma, aplicando el movimiento a los distintos

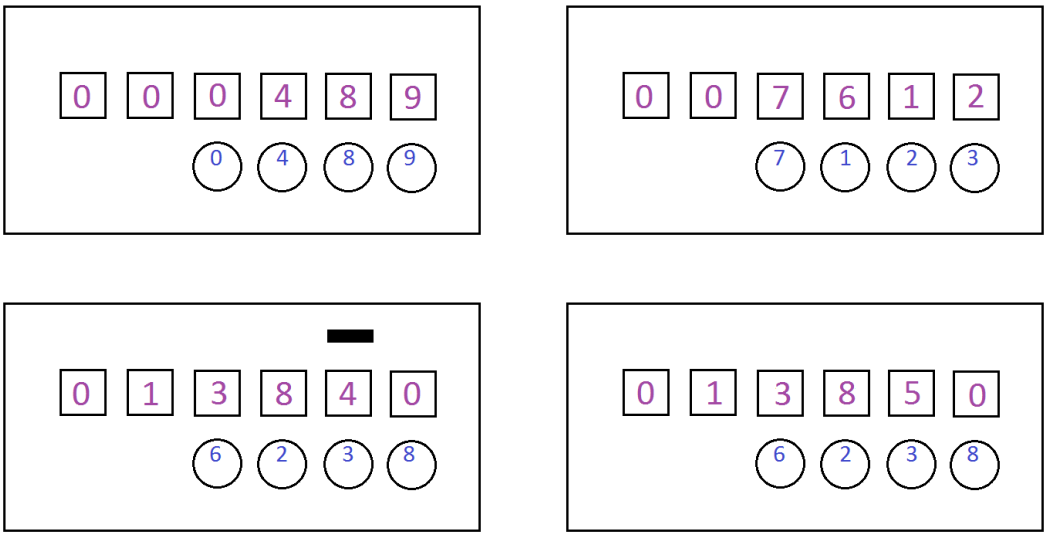


Figura 3.13: Suma con la calculadora

engranajes y dando como resultado 7 612, que se puede apreciar en la Figura 3.13 en la imagen superior derecha. En este caso, todos los acarreos se han producido de forma automática, así que vamos a sumar otro número más para ver qué ocurre.

Sumaremos 6 238 al valor actual que tenemos en el registro de salida. Aplicaremos la misma metodología que hemos aplicado con anterioridad, dando como resultado la imagen inferior izquierda de la Figura 3.13. Como se puede ver, el resultado es incorrecto, pues esperábamos 13 850 y hemos obtenido 13 840, aunque se puede ver que en el segundo dígito empezando por la derecha, las decenas, tenemos señalizado que el acarreo debe hacerse manual. Lo corregimos moviendo el pentágono y tenemos como resultado final 13 850, como se puede ver en la imagen inferior derecha de la Figura 3.13, y con ello hemos terminado la suma.

3.4 Multiplicaciones

Para hacer las multiplicaciones, deberemos de entender muy bien el concepto de «multiplicación escalonada», el cual consiste en multiplicar dígito a dígito, y luego sumar cada uno de los resultados, desplazando correctamente una posición a la izquierda cada uno de ellos conforme sean decenas, centenas, etc. Este uso lo hemos podido ver en el capítulo anterior con las varillas de Napier.

Una vez entendamos este concepto, que es bastante sencillo y que venimos usando desde siempre para realizar estas operaciones a mano, podemos empezar con la explicación de cómo hacer las multiplicaciones con esta calculadora.

A diferencia de la Pascalina, la calculadora de Leibniz era capaz de realizar multiplicaciones como se comentó anteriormente. Para ello, necesitaríamos introducir uno de los dos valores que queremos multiplicar, ya sea el multiplicando o el multiplicador, pues, al igual que con las sumas, el orden con el que realicemos esta operación no altera el resultado. Una vez introducido, procederemos

a la introducción del segundo valor. En este caso es diferente a la suma, donde introducíamos el valor completo en la entrada. Aquí haremos uso de la *Rota magnuscula* de la que hemos hablado anteriormente.

En el anillo central de la *Rota magnuscula* introduciremos la varilla atendiendo a los valores del anillo exterior en negro y al valor de las unidades del segundo número a multiplicar. Colocaremos por tanto la varilla en el agujero correspondiente al valor de las unidades. Una vez hecho esto, giraremos la manivela principal en el sentido de la suma hasta que la varilla que acabamos de colocar haga tope en la parte superior del anillo, todo esto mientras tenemos en cuenta los acarreos manuales. De esta forma acabamos de multiplicar el primer número (multiplicando) por el valor de las unidades del segundo (multiplicador) mediante sumas sucesivas al igual que la Pascalina, y podremos observar este resultado en los registros de salida. La diferencia respecto a la Pascalina aparece cuando el segundo número tiene más de una cifra. Para este caso, deberemos hacer uso de la manivela izquierda, encargada del desplazamiendo de la parte móvil, y desplazar dicha parte una posición a la izquierda. Una vez tengamos colocada la parte móvil en la siguiente posición, colocaremos la varilla en la *Rota magnuscula*, esta vez fijándonos en la siguiente cifra del segundo número y accionando la manivela principal en el mismo sentido que anteriormente hasta que haga tope la varilla. Si tiene más dígitos, se prodedería de la misma forma hasta que tengamos la multiplicación completa.

Esta metodología que hemos empleado es la que se puede ver al usar la «multiplicación escalonada» que hemos aprendido de pequeños, y que a diferencia de la Pascalina, permitía hacer el cálculo de multiplicaciones con una cantidad mucho inferior de movimientos.

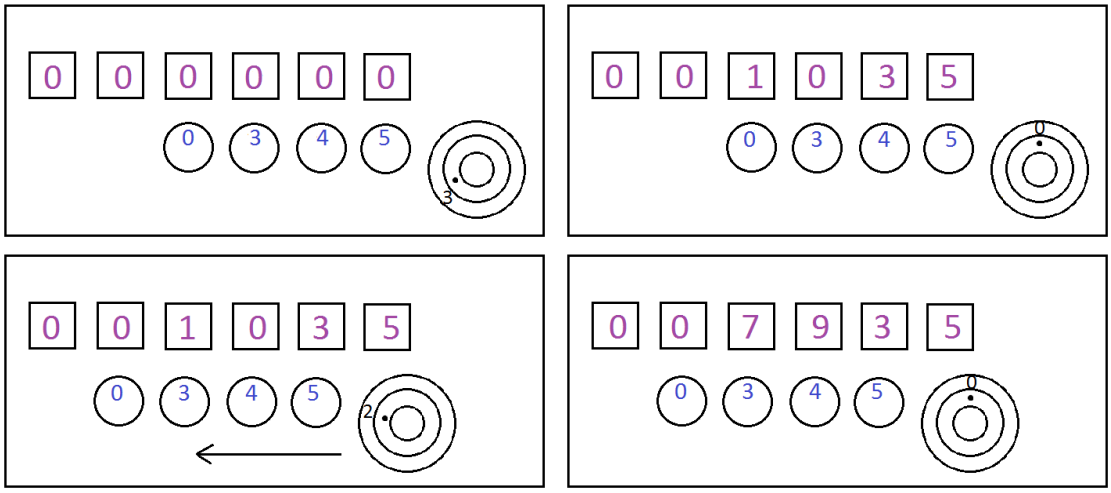


Figura 3.14: Multiplicación con la calculadora

A continuación veremos un ejemplo de una multiplicación usando el método explicado anteriormente, simplificando como con la suma los distintos dibujos que veremos. Procederemos por tanto, a multiplicar, por ejemplo, 345 por 23. Introducimos cualquiera de los valores anteriores, preferiblemente el de más dígitos pues nos ahorrará en número de giros. Colocamos por tanto, 345 en la entrada y la varilla en la posición correspondiente marcada por las unidades del segundo número, 3. Esto se puede ver en la imagen superior izquierda de la Figura 3.14.

Entonces procederemos a girar la manivela principal en el sentido de suma hasta que la varilla nos indique que hemos finalizado haciendo tope, teniendo en cuenta los posibles acarreos manuales intermedios. Veremos, por tanto, un resultado en los registros de salida que está lejos de ser el correcto, pues aún nos falta otro dígito. Podemos ver dicha etapa en la imagen superior derecha de la Figura 3.14 donde tenemos de resultado actual 1 035, que viene a ser lo mismo que haber multiplicado 345 por 3.

Seguimos con el siguiente dígito del segundo número, en este caso es un 2, por lo que colocamos la varilla en la posición 2 y desplazamos toda la parte móvil hacia la izquierda usando la manivela para tal uso. Podemos ver el resultado de estas acciones en la imagen inferior izquierda de la Figura 3.14. Continuaremos con cautela al girar la manivela principal como hemos hecho hasta ahora, teniendo en cuenta los posibles acarreos manuales que requieren de nuestra intervención, y obtendremos por fin, el resultado de multiplicar 345 por 23, como se observa en la imagen inferior derecha de la Figura 3.14, quedando como resultado final 7 935.

3.5 Divisiones

Para poder avanzar y explicar cómo se hacen las divisiones con esta calculadora, tiene que quedar claro el concepto de división más básico de todos, la cual consiste en restas sucesivas al igual que las multiplicaciones eran sumas sucesivas. Otra parte importante que debemos de saber antes de proceder, es cómo funcionan las divisiones hechas en un papel, las que hemos aprendido en la escuela y que muchos han olvidado por el continuado uso de calculadoras.

Dando por sentado que sabemos todo esto, vamos a proceder a explicar el funcionamiento de la calculadora frente a divisiones. Primero de todo, se tiene que destacar que no se ha colocado en el mismo apartado que las multiplicaciones pues a diferencia de las sumas y restas, no se hacen de forma similar.

Para las divisiones, primero tenemos que colocar el dividendo en la entrada, y desplazar hacia la izquierda tantas posiciones como decimales queramos. Una vez hecho esto, giramos la manivela principal en el sentido de suma para tener el valor en el registro de salida, de donde empezaremos a substraer con restas sucesivas y por tanto dividir. Hecho esto, introduciremos en la entrada el valor del divisor, moviendo la parte móvil de tal manera que los dígitos de mayor peso en ambos valores coincidan en posición. De esta manera ya tendríamos preparada la calculadora para empezar a dividir.

Como se ha comentado anteriormente, la división se efectúa mediante restas sucesivas y por tanto, es lo que vamos a hacer a continuación. De la situación anterior, empezamos a restar hasta que veamos que el dígito más significativo del dividendo, en el registro de salida, sea menor que el del divisor, como cuando hacemos las operaciones en papel. Una vez ocurra esto, en la *Rota magnuscula* tendremos el valor del primer dígito de nuestro resultado.

Procederemos a avanzar hacia la derecha una posición la parte móvil y a ver si podemos extraer de aquí; en caso de que podamos, procederemos con la extracción y en caso contrario, avanzaremos una posición más, indicando que el

siguiente dígito del resultado es un 0. Este proceso se repite hasta que la *Pars mobilis* llegue al extremo derecho, dando por finalizada la división. En cualquier caso, hay que tener en cuenta qué parte de los dígitos resultantes es la entera y cuál la fraccionaria.

Llegados a este punto, ya sabemos cómo funciona la división y vamos a ver un ejemplo para clarificar las cosas. Como en los casos anteriores, los ejemplos van a ser simplificaciones de la máquina real, pues es innecesario dibujar tantos registros como en la original si solo se van a aprovechar unos pocos.

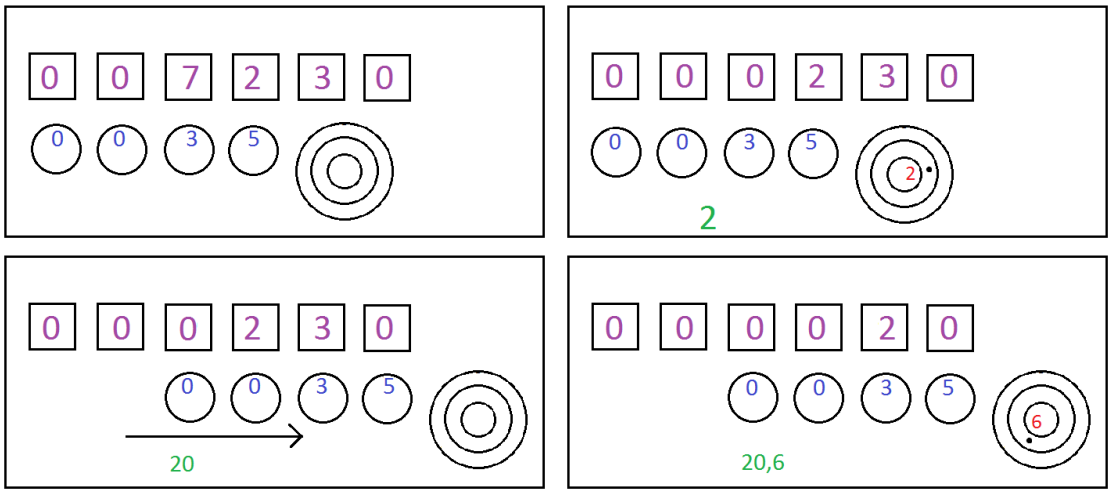


Figura 3.15: División con la calculadora

En este ejemplo, vamos a dividir 723 entre 35, y extraeremos un decimal. Por tanto, vamos a empezar introduciendo el valor del dividendo, 723, en la entrada y moviendo una posición a la izquierda la parte móvil. Accionamos la manivela para que se muestre el valor en la salida (7 230 por el desplazamiento para obtener un decimal) y colocamos el divisor, 35, para que coincidan los dígitos más significativos de ambos, dando lugar a la imagen superior izquierda de la Figura 3.15.

Una vez tengamos la colocación inicial, procederemos a la substracción sucesiva del valor de entrada sobre la salida. Pararemos este proceso cuando la cifra más significativa del dividendo sea inferior a la del divisor como se ha explicado antes, y rescatamos el primer dígito de nuestro cociente de la *Rota magnuscula*. Esto da como resultado la imagen superior derecha de la Figura 3.15 en la cual se observa lo anteriormente comentado.

Desplazaremos la *Pars mobilis* a la derecha una posición y veremos cómo el dígito más significativo del divisor es mayor que el del dividendo, por tanto restamos cero veces en esta posición y avanzamos. Vemos que ahora sí que se puede hacer la resta como se aprecia en la imagen inferior izquierda de la Figura 3.15. Ahora mismo hay que tener cuidado, pues ya hemos sobrepasado la cantidad de dígitos del dividendo y, por tanto, nos encontramos que los valores siguientes van a ser la parte decimal. Realizamos la resta hasta que no se pueda más, quedando la imagen inferior derecha de la Figura 3.15, donde se puede ver el resultado final con un decimal. Dicho resultado es 20,6 y como podemos observar, nos ha quedado un 20 del cual no podemos seguir substrayendo pues hemos alcanzado el borde derecho con la parte móvil.

CAPÍTULO 4
La plataforma Arduino

En este capítulo vamos a hablar sobre la plataforma de desarrollo Arduino, un microcontrolador suficientemente potente para la construcción de diferentes proyectos, desde algunos muy sencillos hasta otros no tanto. Hablaremos tanto del *software* como del *hardware*, comparando distintos modelos.

4.1 Introducción

Arduino es un plataforma compuesta por componentes electrónicos con licencia de código abierto (*Open-source*), es decir, una persona es capaz de crear su propio prototipo Arduino a través de los planos de los distintos componentes así como distribuirlo. Tanto el *hardware* empleado como el *software* vienen bajo dicha licencia y ambos son fáciles de usar y flexibles para que cualquiera sea capaz de crear sus propios proyectos, desde el principiante más inexperto hasta el profesional más avanzado.

Todos los Arduino vienen con un microcontrolador que depende del modelo del mismo, aunque todos se programan de la misma manera mediante el uso del lenguaje de programación propio de Arduino, el *Arduino Programming Language*, el cual está basado en Wiring, también bajo licencia *Open-source*. Para programar el microcontrolador es necesario un entorno de desarrollo proporcionado por la compañía de Arduino y que es totalmente gratuito y se puede descargar a través de su página web [1]. Dicho entorno de desarrollo es el *Arduino Development Environment*, basado en Processing, el cual está también bajo la misma licencia que los anteriores.

Cualquier proyecto desarrollado en Arduino puede ser autónomo, es decir, trabajar por sí solo sin la intervención de otra máquina o del propio usuario, o se puede comunicar con otro *software* instalado en el ordenador con el que trabajará de forma conjunta y en el que el propio usuario puede intervenir en la ejecución. Algunos de los *sofware* más famosos para ordenador que trabajan con Arduino son: Processing, nombrado anteriormente y que permite la creación de interfaces gráficas con botones o cajas de texto, Flash, MaxMSP o Jubito, entre otros.

[1] Página web de descarga del *software* para Arduino en https://www.arduino.cc/en/Main/Software

Como se ha comentado anteriormente, todo lo relacionado con Arduino viene con licencia de código abierto, por que que un usuario puede ensamblar las placas a mano o comprarlas preensambladas además de implementar sus propias versiones de código. Los diseños de los componentes *hardware* de los distintos modelos de placa están disponibles con esta licencia con lo que el usuario es libre de adaptarlos a sus necesidades. En la Figura 4.1 podemos ver el diseño de la placa del microcontrolador del Arduino UNO Rev.3 [2].

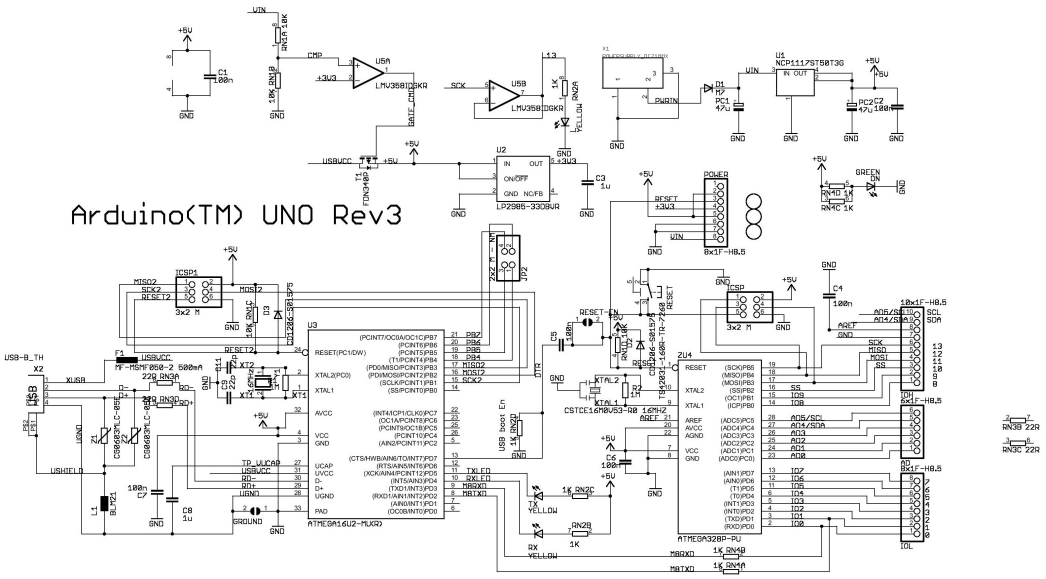


Figura 4.1: Diseño de la placa del microcontrolador del Arduino UNO Rev.3

En cuanto a los microcontroladores, existe gran variedad de ellos así como plataformas sobre las que trabajar, todas ellas con funcionalidades similares a Arduino y con el encapsulamiento del engorroso código del microcontrolador mediante el uso de paquetes sencillos de usar. Al igual que estos microcontroladores, Arduino también simplifica este trabajo pero además ofrece ventajas frente a los demás microcontroladores y plataformas. Las ventajas que proporciona se encuentran en el ámbito del aprendizaje, tanto para profesores como para estudiantes que estén interesados en entrar en el mundo de la programación de microcontroladores. Estas ventajas se enumeran a continuación:

- **Relativamente barato:** El precio de una placa Arduino es relativamente más barato que otras de las ofertas del mercado. La versión más barata puede ser ensamblada a mano e incluso comprando el módulo preensamblado nos cuesta menos de 50 €.

- ***Software* ampliable:** Puesto que el código de Arduino está publicado bajo licencia *Open-source* como se ha indicado con anterioridad, existen grandes cantidades de librerías y API[3] desarrolladas por terceros, las cuales se pueden incorporar a nuestro proyecto si es necesario. El principal uso de estas

[2] Diseño estraído de https://static5.arrow.com/pdfs/2013/10/14/2/49/31/594/arduin_/manual/9arduino_uno_rev3-schematic.pdf

[3] API (inglés): Interfaz de Programación de Aplicaciones

API es la posibilidad de conectar placas de expansión compatibles con Arduino o cualquier componente que tenga su propio chip.

- ***Hardware* ampliable:** Arduino se centra en los microcontoladores ATMEGA328 y ATMEGA168 entre otros, los cuales han sido desarrollados por Atmel. Los distintos planos de estos módulos están bajo licencia *Creative Commons*, lo que permite que cualquier persona sea capaz de crear su propia versión del microcontrolador, ampliando su funcionalidad o incluso mejorándolo, reduciendo el coste de adquisición y para los más inexpertos conlleva el aprendizaje de cómo funciona.
- **IDE o entorno de desarrollo sencillo:** El IDE de Arduino es simple para que puedan empezar los principiantes a programar sus propios proyectos pero lo suficientemente completo para que programadores más experimentados puedan aprovechar lo máximo de Arduino.
- **Multiplataforma:** El *software* de Arduino es capaz de ejecutarse en diferentes tipos de máquinas, ya sean con sistema operativo Windows, GNU/Linux o Macintosh OSX. Gran parte de los microcontroladores solo ofrecen soporte para Windows.
- **Simplicidad:** La plataforma Arduino se encarga de simplificar al máximo la interacción del usuario con el microcontrolador, creando funciones específicas para las diferentes acciones que es capaz de hacer, como puede ser la extracción de la hora o el tiempo de ejecución del código.
- **Gran cantidad de componentes:** Una de las principales razones por las que Arduino es de los microcontroladores más usados es la gran cantidad de componentes compatibles, ya sean sensores para captar el exterior o interactuar con el entorno.
- **Enorme repertorio de modelos:** Al estar desarrollado bajo licencia *Opensource*, diseñadores experimentados en circuitos son capaces de crear diferentes modelos con sus propias características, enfocando más en el uso de Arduino para determinadas tareas.

4.2 Hardware

En esta sección vamos a hablar un poco más del *hardware* de Arduino. Como se ha comentado anteriormente, Arduino está formado por una gran variedad de placas diferentes, cada una con su propio microcontrolador, como pueden ser el Arduino Uno y el Arduino Mega, mostrados en la Figura 4.2, el Due, Leonardo, Diecimilla, Nano, etc. Para ver un poco cómo es cada uno, hemos creado la Tabla 4.1 donde se van a comparar las diferentes características de las versiones más utilizadas actualmente.

Dado que en este proyecto vamos a usar Arduino Mega 2560 para la construcción de la calculadora, tenemos que detallar un poco más sus características. En este caso, este modelo de Arduino viene con las características mostradas en la Tabla 4.1, pero además viene con 4 puertos UART (*Universal Asynchronous*

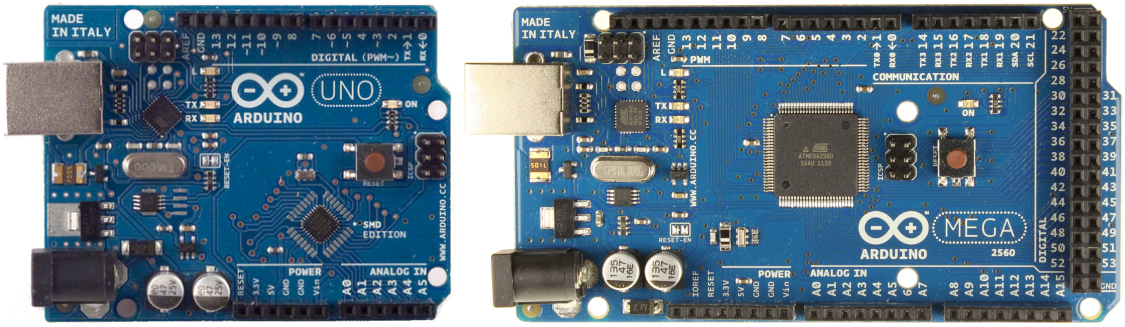


Figura 4.2: Arduino Uno Rev.3 (izquierda) y Arduino Mega 2560 Rev.3 (derecha)

Receiver-Transmitter) de comunicación serie, un cabezal ICSP (*In-Circuit Serial Programming*) para reprogramar el *bootloader*, un puerto USB, un botón de *reset* y un conector de alimentación externa. Todo esto es lo necesario para mantener al microcontrolador, simplemente necesitaremos una fuente de alimentación, ya sea conectando directamente al PC mediante el puerto USB o usando una batería. En la Figura 4.3 podemos ver un esquema general del Arduino Mega 2560 con los elementos nombrados previamente. Además es compatible con la mayoría de *shields* (placas que se conectan directamente sobre el Arduino dándole características extra) compatibles con Arduino Uno, Diecimilla o Duemilanove.

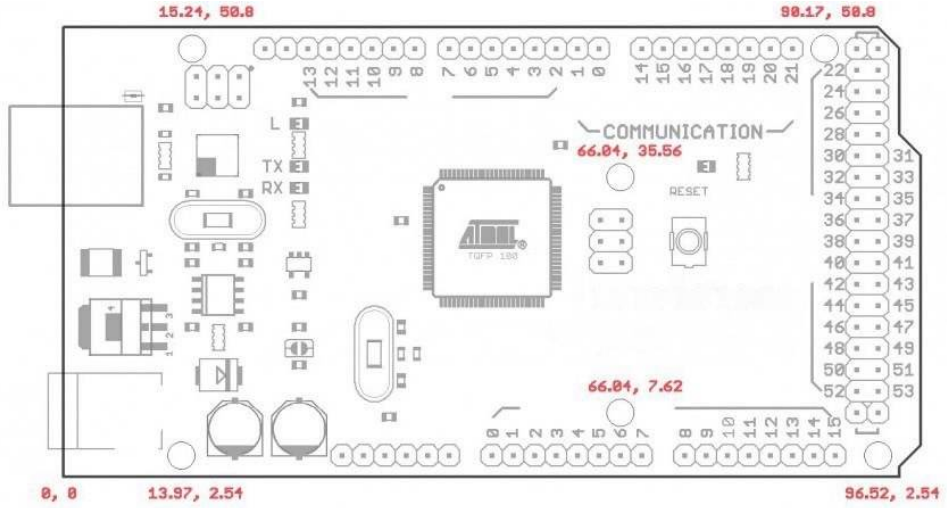


Figura 4.3: Esquema del Arduino Mega 2560 Rev.3

4.3 Software

En este apartado vamos a hablar brevemente sobre el entorno de desarrollo de Arduino y aprenderemos a configurarlo para que funcione con nuestro modelo.

Primero de todo hay que descargar el *software* a nuestro ordenador, el cual esta accesible en la página web de Arduino. Una vez descargado habrá que instalarlo y dependiendo de la versión del sistema operativo que estemos usando se hará de una forma u otra.

	Arduino Uno	Arduino Mega 2560	Arduino Nano	Arduino Due
Microcontrolador	ATmega328P	ATmega2560	ATmega168 o ATmega328	AT91SAM3X8E
Voltaje Operativo	5 V	5 V	5 V	3,3 V
Voltaje de entrada (recomendado)	7-12 V	7-12 V	7-12 V	7-12 V
Voltaje de entrada (límites)	6-20 V	6-20 V	6-20V	6-16 V
Pines E/S digitales	14 (6 de salida PWM)	54 (15 de salida PWM)	14 (6 de salida PWM)	54 (12 de salida PWM)
Pines de entrada analógica	6	16	8	12
Corriente DC por Pin E/S	20 mA	40 mA	40 mA	130 mA
Corriente DC por Pin 3,3 V	50 mA	50 mA	50 mA	800 mA
Memoria Flash	32 KB (0,5 KB para el bootloader)	256 KB (8 KB para el bootloader)	16 KB (ATmega168) o 32 KB (ATmega368) (2 KB para el bootloader)	512 KB
SRAM	2 KB	8 KB	1 KB (ATmega168) o 2 KB (ATmega368)	96 KB (dos bancos: 64 KB y 32 KB)
EEPROM	1 KB	4 KB	512 B (ATmega168) o 1 KB (ATmega368)	
Velocidad de reloj	16 MHz	16 MHz	16 MHz	84 MHz

Tabla 4.1: Características técnicas de distintos Arduino

Cuando abramos el programa por primera vez, nos mostrará una pantalla como se ve en la Figura 4.4. En este momento podemos conectar nuestro Arduino mediante el cable USB que suele venir con la compra del mismo y que sirve tanto alimentar la placa como para enviarle el código que tiene que ejecutar.

Una vez conectada la placa, lo normal es que aparezca bajo el desplegable *Herramientas -> Puerto*, nuestro Arduino a través del puerto COM3. Esto se puede ver en la Figura 4.5 donde, además, aparece la placa que hemos conectado y el tipo de procesador. Estos dos últimos aspectos los suele detectar automáticamente aunque siempre hay que mirar que se corresponde con nuestra placa. En caso de que no se detecte, siempre podemos elegir manualmente nuestra placa y procesador a través de los desplegables que corresponden a cada categoría.

Con esto ya tenemos configurado el entorno de desarrollo para que funcione con nuestro Arduino, aunque se puede dar el caso de que necesitemos agregar alguna librería externa como va a ocurrir en este proyecto. Para ello tendremos que descargar la librería que queramos añadir en el menú *Programa -> Incluir Librería -> Añadir Librería ZIP* tal y como vemos en la Figura 4.6. En caso de querer desinstalarla, simplemente vamos a la ruta de nuestro ordenador (Windows) `\\Usuario\Documentos\Arduino\libraries\` y allí eliminamos la carpeta de la librería que queramos desinstalar.

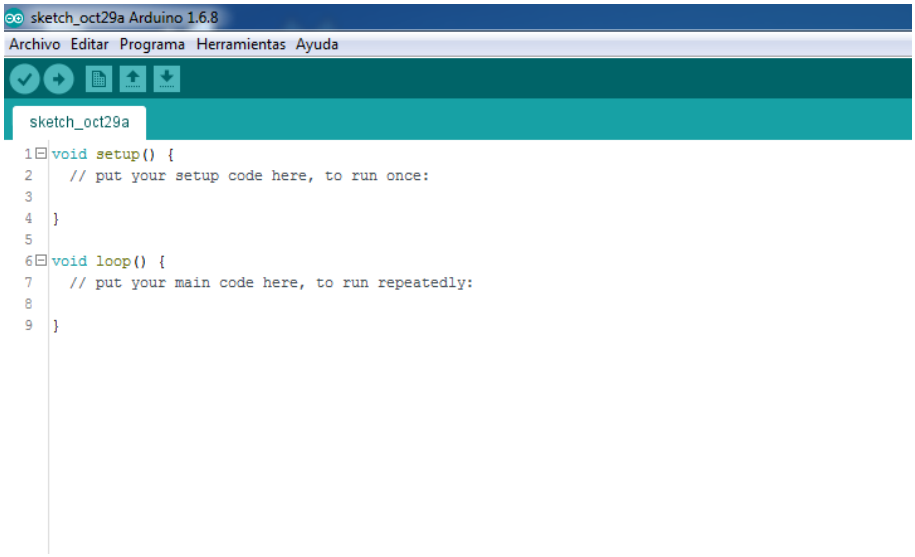


Figura 4.4: Ventana principal del entorno de desarrollo de Arduino

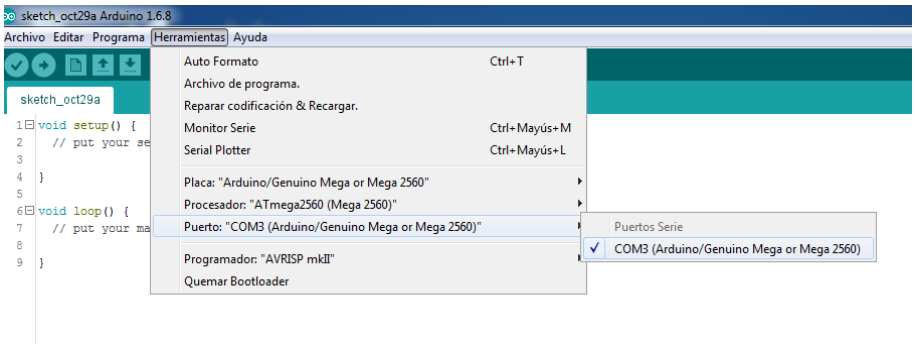


Figura 4.5: Configuración inicial del IDE para empezar a programar

En cuanto al lenguaje de programación usado es similar a Java o C++. En la Figura 4.4 podemos ver dos funciones o métodos creados por defecto y las cuales son necesarias para poder ejecutar el código. La función `setup()` se ejecuta una sola vez al principio de la ejecución y se suele usar para declarar las distintas variables que va a usar nuestro programa. En cambio, la función `loop()` se va a ejecutar indefinidamente hasta que, o bien surja algún error de ejecución o hasta que le cortemos la alimentación o le demos al `reset` de nuestra placa. Dentro de esta última función es donde vamos a colocar todo nuestro código, el cual se sitúa convenientemente a través de llamadas a otras funciones que crearemos nosotros y que hay que posicionar siempre antes del *loop()*. Una vez tengamos escrito nuestro código hay que darle al botón de la flecha que aparece arriba a la izquierda, el cual se encargará de compilar nuestro código y enviarlo al Arduino, avisando si surge algún error ya sea en la compilación o en el envío.

Arduino cuenta a su vez con una API[4] en la que describen las distintas funciones ya implementadas y de las que podemos hacer uso en cualquier momento. Además, si tenemos algún problema relacionado con Arduino, podemos visitar sus foros[5] donde casi siempre encontraremos la solución o en caso contrario, describir nuestro problema y alguien nos contestará.

[4]Enlace a la API `https://www.arduino.cc/en/Reference/HomePage`
[5]Enlace a los foros `http://forum.arduino.cc/`

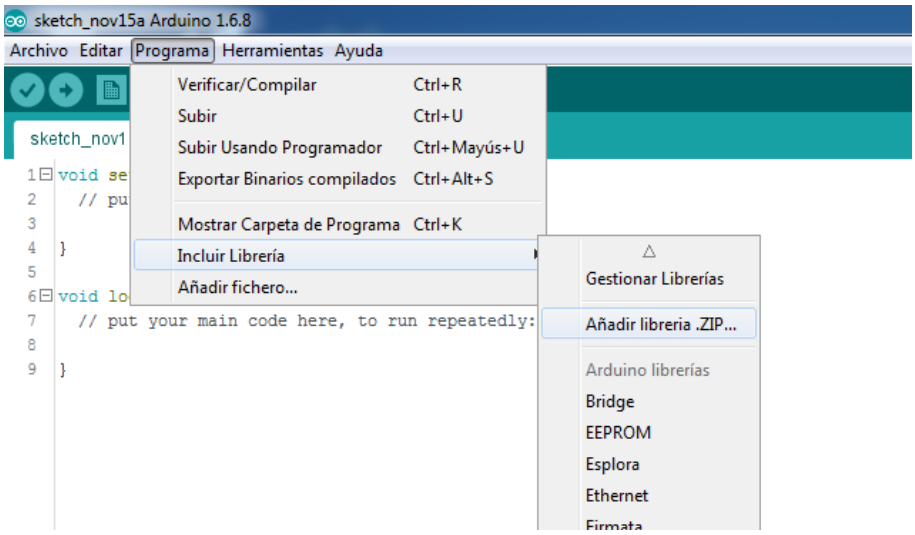


Figura 4.6: Añadir nuevas librerías

CAPÍTULO 5
Diseño y montaje de la calculadora

En este capítulo vamos a hablar acerca de los elementos que van a formar parte de nuestra calculadora, entre los que destaca Arduino Mega 2560, cuyas características técnicas se pueden ver en el capítulo previo, que va a ser el encargado de procesar toda la infomación proveniente de los demás elementos. Entre ellos se encuentran los potenciómetros, LED, botones, etc. Además se hablará sobre el programa desarrollado, haciendo especial énfasis en las funiones encargadas de las sumas y restas.

5.1 Presupuesto

En esta sección haremos un análisis detallado del presupuesto de los distintos componentes que van a componer nuestra máquina y los cuales vamos a necesitar para su construcción. Nos centraremos solamente en los componentes electrónicos.

Por una parte vamos a realizar un presupuesto basándonos en los precios de la tienda de electrónica más cercana, Electrónica Burriana, ciudad de Valencia, en el que podríamos conseguir los materiales de forma inmediata o con un plazo de un par de semanas. Por otra parte, basaremos otro presupuesto en la compra de los mismos componentes directamente a China a través de varias páginas web encargadas de su distribución, con lo que el plazo de llegada oscilaría entre el mes y medio hasta los dos meses y medio.

Como podemos observar en las Tabla 5.1 y Tabla 5.2, precisaremos de componentes tales como un Arduino Mega 2560 Rev.3, varios potenciómetros, botones y LED así como sus resistencias, entre otros componentes. Cabe destacar, que en el presupuesto de China, al venir de fuera y no ser una tienda física como Electrónica Burriana, la cantidad de unidades que venden son mayores y siempre vienen en packs de un número variable de unidades, por lo que algunos componentes pueden parecer más caros. Además, faltan añadir los gastos de envío pues según el vendedor y la cantidad de componentes que compremos, habrá que pagar más o menos por ellos.

Por otro lado, si queremos añadir al presupuesto el gasto de la mano de obra y las horas de trabajo, alrededor de ciento veinte horas, podríamos establecer un incremento en el presupuesto basándonos en la relación precio por hora de la

Presupuesto Burriana				
Componente	Precio Unidad	Uds.	Precio	+21 % IVA
Arduino Mega 2560 Rev.3	34,00 €	1	34,00 €	41,14 €
Potenciómetro lineal	1,33 €	9	11,97 €	14,49 €
Resistencia 330 Ω	0,03 €	17	0,51 €	0,62 €
Resistencia 10 kΩ	0,03 €	8	0,24 €	0,29 €
LED Amarillo 5 mm	0,10 €	15	1,50 €	1,82 €
LED Rojo 5 mm	0,10 €	1	0,10 €	0,12 €
LED Verde 5 mm	0,10 €	1	0,10 €	0,12 €
Adaptador I2C	3,00 €	1	3,00 €	3,63 €
Visualizador numérico 7 segmentos	0,60 €	1	0,60 €	0,73 €
Cables Arduino (40 u)	3,21 €	2	6,42 €	7,77 €
Botón pulsador	0,23 €	8	1,84 €	2,23 €
Visualizador LCD 2004 (20x4)	13,00 €	1	13,00 €	15,73 €
Protoboard 1660	20,60 €	1	20,60 €	24,93 €
		TOTAL	93,88 €	113,62 €

Tabla 5.1: Presupuesto de Electrónica Burriana (Valencia)

Presupuesto China			
Componente	Precio Unidad	Uds.	Precio
Arduino Mega 2560 Rev.3	8.43 €	1	8,43 €
Potenciómetro lineal (10 u)	2,12 €	1	2,12 €
Resistencia 330 Ω (100 u)	1,13 €	1	1,13 €
Resistencia 10 kΩ (100 u)	0,70 €	1	0,70 €
LED Colores 5 mm (100 u)	1,50 €	1	1,50 €
Adaptador I2C	0,78 €	1	0,78 €
Visualizador numérico 7 segmentos (10 u)	1,39 €	1	1,39 €
Cables Arduino (40 u)	0,80 €	2	1,60 €
Botón pulsador (100 u)	1,39 €	1	1,39 €
Visualizador LCD 2004 (20x4)	2,99 €	1	2,99 €
Protoboard 1660	11,93 €	1	11,93 €
		TOTAL	33,96 €

Tabla 5.2: Presupuesto de China (vuelo)

mano de obra realizada. Un programador cobra 10€ la hora con lo que ascendería a un total de 1 200€ más.

Para finalizar, cabe destacar que los costes asociados a la compra del material han sido asumidos por mí y que posteriormente, el Museo de Informática de la Universitat Politècnica de València, ha querido correr con los gastos anteriores. Dado la normativa de la universidad, la factura de compra y los objetos comprados en cuestión deben estar disponibles casi simultáneamente, por lo que la compra a China ha sido descartada y se centra todo en el material proporcionado por Electrónica Burriana asumiendo, por tanto, dichos costes.

5.2 Entrada: montaje

En esta sección vamos a hablar sobre la parte *hardware* que compone toda la entrada de nuestra máquina, es decir, todos los componentes que forman parte del mecanismo que puede emplear el usuario para introducir información a nuestra calculadora.

Para construir la parte de la entrada vamos a necesitar los materiales que se ven el la Figura 5.1 donde se muestra un componente de cada. Cabe destacar que vamos a necesitar varios de cada uno de ellos pero que por simplicidad se ha decidido colocar solo uno. Podemos ver que vamos a necesitar potenciómetros, botones pulsadores y sus correspondientes resistencias y unos cuantos cables.

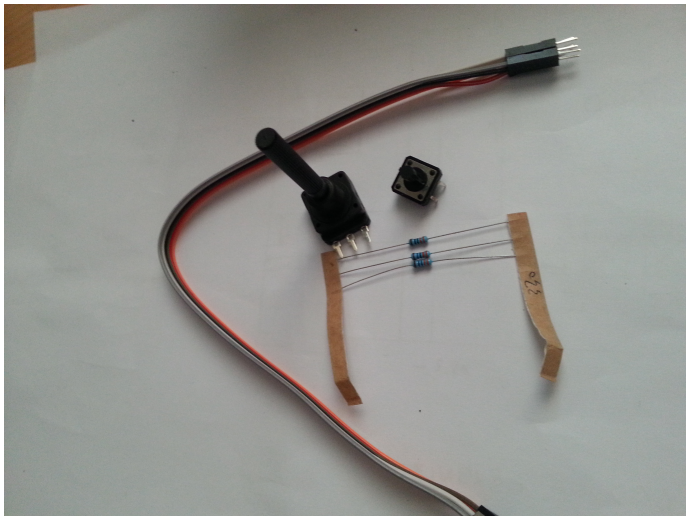


Figura 5.1: Materiales necesarios para el montaje de la entrada

Para empezar, montaremos la parte encargada de la introducción de los números usando ocho potenciómetros lineales de 10 kΩ, un cable para cada uno de ellos el cual conectaremos a los pines de la entrada y un par de cables más para conectar la masa y 5 V a puntos comunes. Una vez tengamos a mano estos materiales, vamos a proceder a realizar las conexiones y soldaduras. Nos vamos a centrar en el esquema eléctrico de la Figura 5.2 donde se conecta un potenciómetro al Arduino que podemos extender para el conjunto de los ocho que poseemos. Como vemos en la imagen, el potenciómetro tiene tres conexiones, de las cuales dos vamos a conectar a puntos comunes, 5 V y masa, mientras que el tercero va

a ir por separado en cada uno de los potenciómetros. Por tanto, vamos a tener todos los puntos de conexión izquierdos a masa y los derechos a 5 V, mientras que el central irá conectado a cada uno de los pines del arduino.

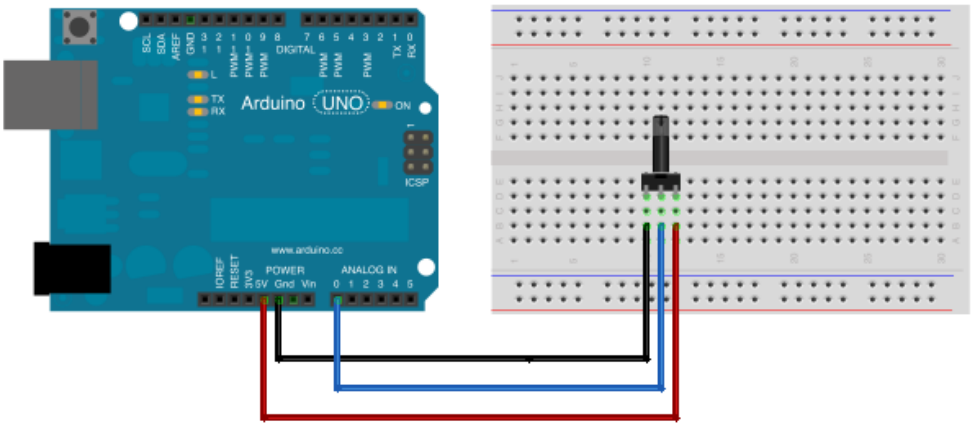


Figura 5.2: Esquema de conexiones para potenciómetros

Para el montaje de esta parte hemos usado un listón de madera que sirve de soporte para los ocho potenciómetros, los cuales van pegados sobre él y entre ellos creando una estructura rígida. Posteriormente se han soldado las distintas conexiones nombradas anteriormente. Para esta parte se ha usado un cable común para la masa y otro para los 5 V a los cuales se les ha quitado el plástico protector en partes determinadas para soldar cada uno de los puntos de conexión de los ocho potenciómetros. Finalmente, para las conexiones individuales había dos opciones posibles: cortar un lado de los cables macho-macho o macho-hembra de tal forma que quedase un macho para conectar al Arduino y soldar el otro al punto de conexión o poner soldadura en dicho punto y usar cables macho-hembra de tal forma que con la soldadura no se saliese y posteriormente pegarlos para aumentar la seguridad y rigidez. Nos hemos decantado por la segunda opción la cual nos permitía mantener intactos los cables. El montaje de esta parte lo podemos ver en la Figura 5.3 donde se aprecia todo lo previamente comentado.

Vamos a continuar con la otra parte de la entrada, en este caso, los botones. Para ello vamos a necesitar otros ocho botones de tipo pulsador. Tienen que ser de este tipo pues queremos que el Arduino detecte solamente cuándo se mantiene pulsado el botón y al soltarlo deje de detectarlo. En esta parte también vamos a usar un par de cables así como de una resistencia por botón de 10 kΩ. El montaje es sencillo si entendemos las conexiones internas del propio pulsador. Nuestros pulsadores tienen cuatro conexiones, las cuales están conectadas dos a dos. Mirando el botón desde arriba y con las patas arriba y abajo, las dos patas más a la izquierda están conectadas y las de la derecha igual. Para realizar el montaje vamos a emplear el esquema de la Figura 5.4 donde se usa la resistencia a modo de *pull down*, es decir, el pin donde conectemos el botón estará permanetemente conectado a masa y el pulsador deja pasar 5 V cuando se presione.

Igual que ocurría con los potenciómetros, vamos a conectar todos los puntos de 5 V a un punto común y los de masa a otro. Cabe destacar que el cable de masa estará conectado a la resistencia que a su vez está conectada a la pata co-

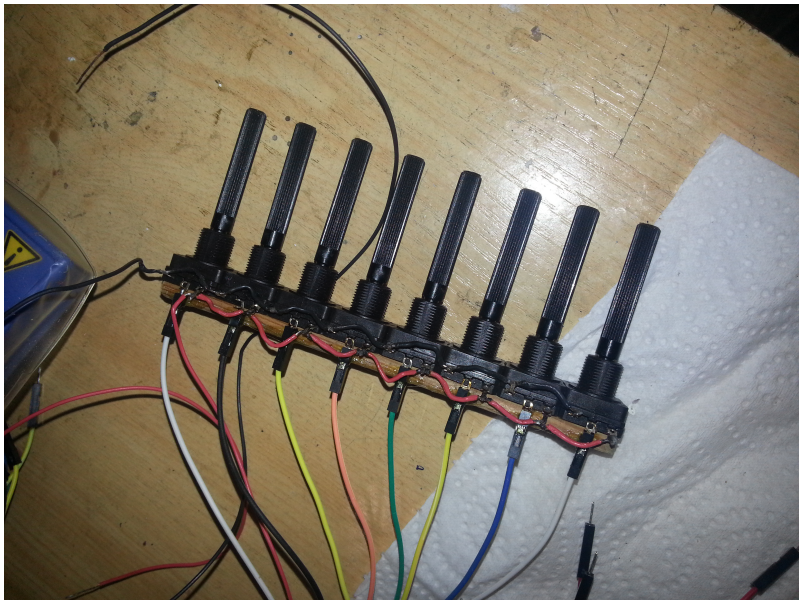


Figura 5.3: Montaje de los ocho potenciómetros

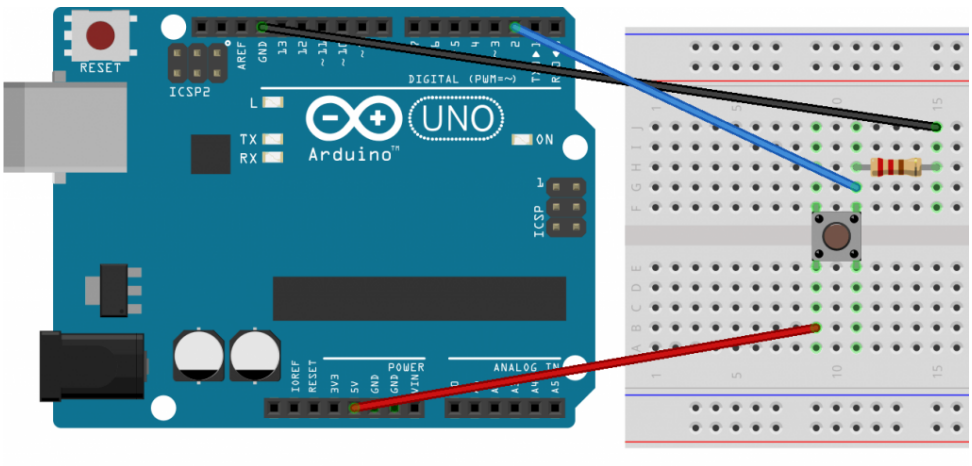


Figura 5.4: Esquema de conexiones para botones pulsadores

rrespondiente del puslador. En esta explicación solo se van a emplear cuatro de los ocho botones. De los cuatro restantes, tres van a ir unidos de la misma forma que los anteriores cuatro y el último irá individual. Esto es así para poder hacer el montaje final que veremos en otro apartado.

Para finalizar, soldaremos la pata opuesta a la que está conectada la resistencia (son el mismo punto) con otro cable individual que se conectará directamente al Arduino y que será el encargado de indicarle cuándo se ha pulsado el botón. En este caso, el cable sí que ha tenido que ser cortado y soldado a la propia pata del puslador por razones técnicas. El diseño final para estos cuatro botones lo podemos ver en la Figura 5.5 donde ademas se incluye un listón de madera como soporte para unir dichos botones.

Una vez tenemos el diseño completo de los botones y potenciómetros, nos quedaría conectar cada uno de los cables individuales a los pines del Arduino correspondientes. Para el caso de los potenciómetros, los pines del Arduino van desde el *A0* hasta el *A7*, siendo el potenciómetro más a la derecha (dígito menos

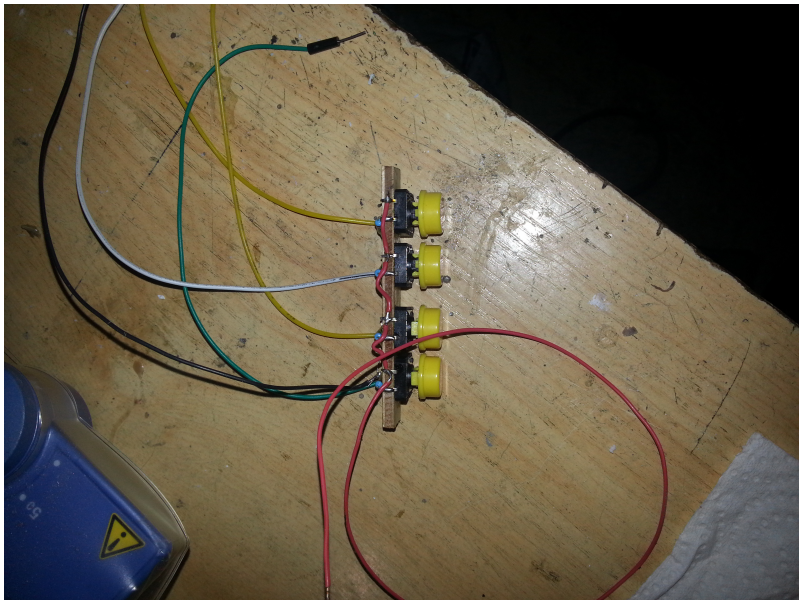


Figura 5.5: Montaje de los cuatro botones

significativo) el que va conectado al *A7* y el izquierdo (dígito más significativo) al *A0*. En cuanto a los botones, dado que cada uno ejerce una función determinada, deberemos conectarlos a los pines correspondientes. En cualquier caso, los pines van desde el *2* hasta el *9* siendo este el orden: *reset* de la salida, desplazamiento izquierda, desplazamiento derecha, suma, resta, acarreo manual, incrementar valor del visualizador de siete segmentos y *reset* del contador.

Llegados a este punto, tenemos creada la parte de la entrada, tanto los potenciómetros para la entrada de los números como de los botones para realizar las distintas acciones.

5.3 Entrada: funcionamiento

En esta sección se va a explicar el funcionamiento de la parte *software* de nuestra entrada, explicando las funciones que se han creado así como las que venían ya implementadas.

Para empezar, la entrada principal de información viene compuesta por ocho potenciómetros, cada uno asignado con un valor de unidades, es decir, el que se encuentra más a la derecha corresponde a las unidades y una posición a la izquierda el de las decenas. Así con cada uno de los ocho. En este caso se ha empleado el sistema decimal para la entrada pero se puede cambiar a cualquier otro sistema siempre y cuando la base sea inferior a diez.

En el bucle principal de nuestro programa se hace una llamada a una función encargada de leer cada una de las entradas anteriores y almacenar cada uno de estos valores en un vector de ocho posiciones útiles. Decimos posiciones útiles pues se han creado algunas extra por si acaso nos saliésemos del vector.

Como podemos ver en el código 5.3, usamos la función `map`. Dicha función necesita cinco argumentos: el primero es el valor que queremos transformar, el

segundo y tercer argumento son el rango de valores que puede tomar el primer argumento, el cuarto y quinto son el rango de valores al cual se quiere transformar. También vemos el uso de la función definida por Arduino `analogRead` la cual se encarga de leer de forma analógica (valores entre 0 y 1 023) del pin que le pasemos como argumento.

```
1    map(analogRead(pinEntrada[posicionEntrada]), 0, 1023, 0, 9);
```

En esta parte nos encontramos un problema. Las entradas analógicas, las cuales leemos mediante la función `analogRead(pin)`, toman valores entre 0 y 1 023, ambos inclusive. Hasta aquí todo correcto. El problema viene al poner el rango de la salida, pues nosotros indicamos que queremos transformar en el rango de 0 al 9 y la propia función asigna el valor 1 023 de la entrada al valor 9, de tal forma que solo obtendremos un 9 si la entrada es 1 023. Esto no es lo que queremos pues nos deja el 9 diferente de los demás dígitos que tienen un rango en el que aparecen.

La solución la podemos ver en el código 5.3, donde se ha decidido colocar el rango de entrada entre 0 y 1 024 y la salida entre 0 y 10. De esta forma, el valor 10 solo aparece cuando en la entrada se lee un 1 024 y como nunca se va a dar el caso, no aparecerá el 10. Cada una de las cifras del 0 al 9 tendrá su propio rango de aparición del mismo tamaño.

```
1    map(analogRead(pinEntrada[posicionEntrada]), 0, 1024, 0, 10);
```

En el código 5.1 podemos ver la función completa donde se recorre el vector de entrada y se le asigna a cada posición un valor entre el 0 y el 9 dependiendo de las lecturas que obtenga el Arduino de cada uno de los potenciómetros. En este caso, el 10 que aparece en el código 5.3 se ha sustituido por una variable que almacena la base del sistema que estamos usando, es decir, por defecto valdrá 10 pero se puede cambiar a un 2 si queremos usar binario o a un 8 si queremos octal. Esta variable va a aparecer de forma recurrente a lo largo del código.

```
1    void leerValoresEntrada()
     {
3        //Recorremos el vector de entradas
         for (int posicionEntrada = 0;
5            posicionEntrada < numeroEntradas;
             posicionEntrada++)
7        {
             entrada[numeroEntradas - posicionEntrada - 1] =
9                map(analogRead(pinEntrada[posicionEntrada]),
                 0, 1024, 0, baseSistema);
11       }
     }
```

Código 5.1: Lectura de potenciómetros

Ahora vamos a pasar a la otra parte de la entrada que corresponde con los botones. Cada uno de estos botones tendrá una función asociada que se lanzará cuando se pulse dicho botón.

Igual que sucedía con anterioridad con los potenciómetros, la llamada a la función encargada de controlar qué botones se han pulsado es llamada en cada iteración del bucle principal. Esta función simplemente controla los botones pulsados y redirecciona el flujo del programa a cada una de las funciones asociadas mediante el uso de `if else`.

La lectura de cada botón se hace al principio de esta función mediante las líneas de código que podemos ver en 5.2. Como se ve en el código nombrado previamente, hay un total de ocho botones con un nombre significativo que representa la función que tienen asignada. Cada uno de los botones se lee mediante la función de Arduino `digitalRead` la cual obtiene los valores digitales (0 o 1) del pin de entrada que le pasemos como argumento, igual que sucedía anteriormente con los potenciómetros.

```
/*Obtenemos los valores de los botones (pulsado o no-pulsado)*******
*/
pulsadoBotonSuma = digitalRead(botonSuma);
pulsadoBotonResta = digitalRead(botonResta);
pulsadoBotonReset = digitalRead(botonReset);
pulsadoBotonDesplazamientoDerecha = digitalRead(
botonDesplazamientoDerecha);
pulsadoBotonDesplazamientoIzquierda =
    digitalRead(botonDesplazamientoIzquierda);
pulsadoBotonAcarreoManual = digitalRead(botonAcarreoManual);
pulsadoBotonDisplaySieteSegmentos = digitalRead(
botonDisplaySieteSegmentos);
pulsadoBotonResetContador = digitalRead(botonResetContador);
```

Código 5.2: Lectura de botones

Una vez hemos ejecutado el código 5.2 tendremos almacenado en las variables `pulsado***`, donde `***` representa el nombre del botón, si el botón estaba pulsado cuando se ha hecho la lectura o por el contrario no lo estaba. A partir de esta información haremos las distintas llamadas a las demás funciones. Como podemos ver en el código ejemplo 5.3, simplemente comprobamos que se ha pulsado el bóton de reinicio del contador y, en caso afirmativo, llamamos a la función encargada de reiniciar el contador. Hay que tener en cuenta que los `if~else` son `true` si el valor que lee es diferente de 0 y `false` cuando es 0 y puesto que las variables almacenan valores digitales, se pueden emplear directamente sin hacer una comprobación extra.

```
if (pulsadoBotonResetContador)
{
    reiniciarContador();
}
```

Código 5.3: Ejemplo de redireccionamiento

Llegados a este punto, ya tenemos completada la lectura de los botones y redirigido el código a cada una de las funciones encargadas de los cálculos intermedios.

5.4 Salida: montaje

En este apartado se va a explicar todo lo relacionado con el montaje de la salida, tanto de los visualizadores como de los distintos LED.

Podemos ver los materiales que vamos a necesitar a lo largo de esta apartado en la Figura 5.6. La cantidad de los materiales difiere de los requeridos por simplicidad en la imagen. Podemos ver que se van a necesitar un visualizador LCD y de siete segmentos así como algunos LED con sus correspondientes resistencias de 330 Ω y algunos cables para las conexiones. Cabe destacar que también necesitaremos un adaptador I2C que no se ve en la imagen por estar ya conectado al propio visualizador LCD.

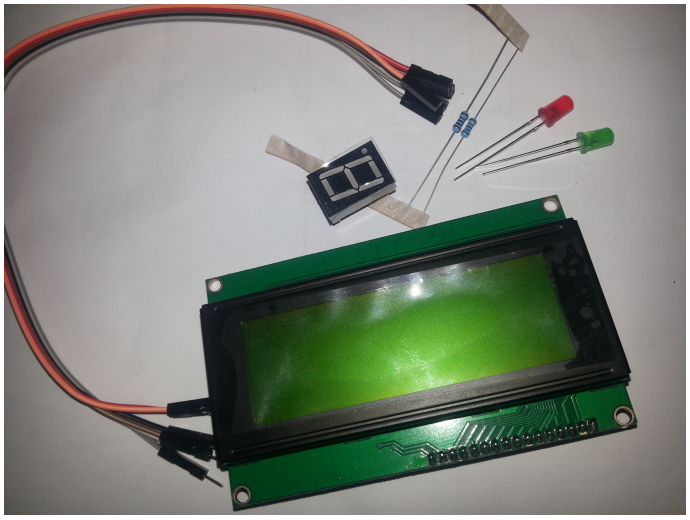


Figura 5.6: Materiales necesarios para la construcción de la salida

Empezaremos, por tanto, con la parte más sencilla e importante, el visualizador LCD donde se van a mostrar los resultados de nuestras operaciones. Sin él sería imposible la realización de la emulación. Necesitaremos para su montaje cuatro cables, el visualizador y el adaptador I2C.

Lo primero de todo es conectar nustro adaptador I2C al visualizador. Para ello simplemente colocaremos las patas del adaptador en los agujeros correspondientes del visualizador, en la parte superior del mismo, dejando los cuatro pines que usaremos del adaptador a la izquierda visto desde arriba del visualizador. Soldaremos cada una de las patas para que no se mueva.

Las conexiones de las cuatro patas laterales tiene el nombre de *SDA*, *SCL*, *VCC* y *GND*. Las dos primeras las conectaremos directamente a los pines del Arduino con su mismo nombre. Por otra parte, los dos restantes se conectarán a la masa común para el caso de *GND* y a 5 V para el caso de *VCC*. En cualquier caso, se ha empleado un cable macho-hembra para las conexiones, usando la parte de la hembra en el adaptador y el macho para conectar con el Arduino a excepción de los puntos comunes donde se ha cortado el cable para poder soldarlo. Podemos ver dicho montaje en la Figura 5.7 donde se aprecia la parte trasera del visualizador con el adaptador I2C. En el adaptador I2C hay un potenciómetro para regular el contraste de la pantalla y diferenciar mejor los carácteres del fondo.

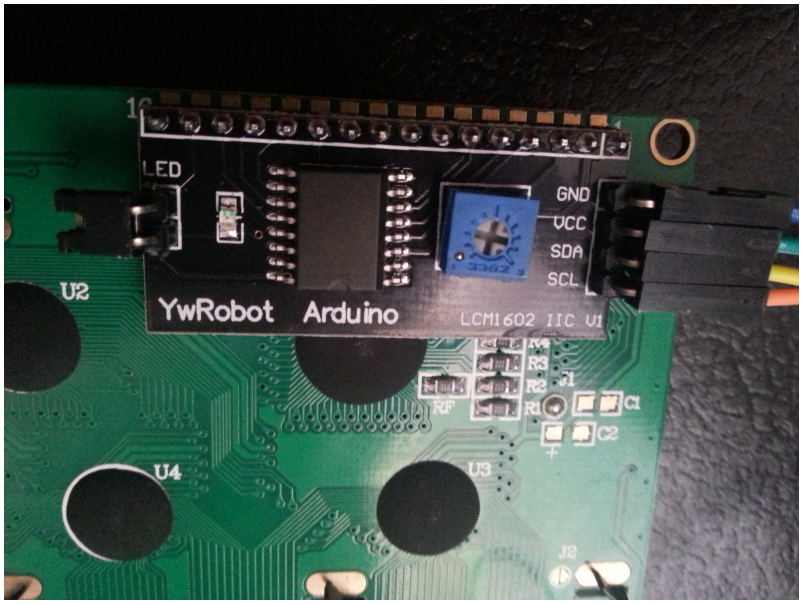


Figura 5.7: Montaje del visualizador LCD

Seguiremos ahora con los LED de acarreo, un total de quince, de color amarillo y sus respectivas resistencias, una para cada uno. Para el montaje de esta parte nos basaremos en el esquema de la Figura 5.8, donde se conecta la resistencia a la pata positiva del LED y ésta a su pin de control. En este caso vamos a ver las conexiones de los LED de acarreo aunque el LED de multiplicación (verde) sigue el mismo esquema. El LED de desbordamiento (rojo) es diferente pues se conecta al pin 13 del Arduino que lleva su propia resistencia interna y no es necesario soldarle una.

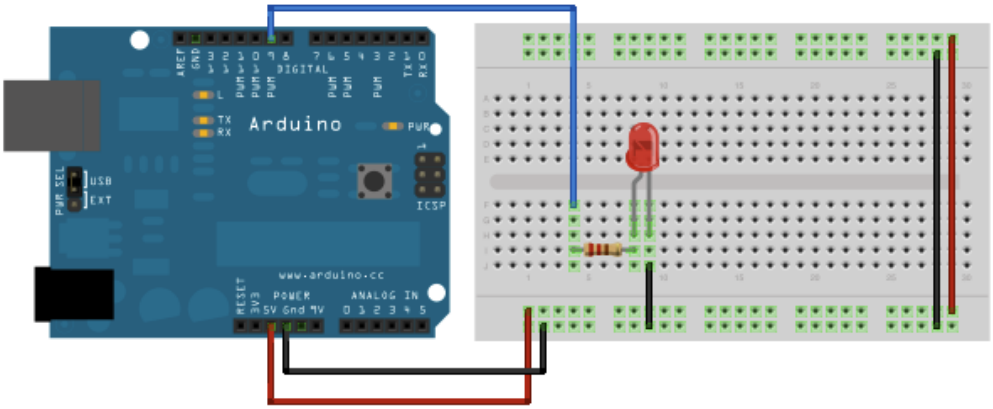


Figura 5.8: Esquema de conexión de un LED

Para reducir el número de cables y el espacio se ha decidido crear una placa con circuito donde efectuar las distintas soldaduras y conexiones en los LED de acarreo. Suponemos el esquema previo y soldamos a la placa los distintos LED y sus respectivas resistencias, usando un cable para masa común como hemos hecho con los demás montajes. Al finalizar todo esto, tenemos un diseño reducido de conexiones de quince LED con sus respectivas resistencias que podemos ver en la Figura 5.9, donde aparece tanto la placa con circuito empleada como el re-

sultado final donde se han empleado cables cortados con un extremo macho para conectar directamente con los pines del Arduino. Estos pines serán desde el 30 hasta el 44 conectando el LED del dígito menos significativo al pin 30 y el más significativo al 44.

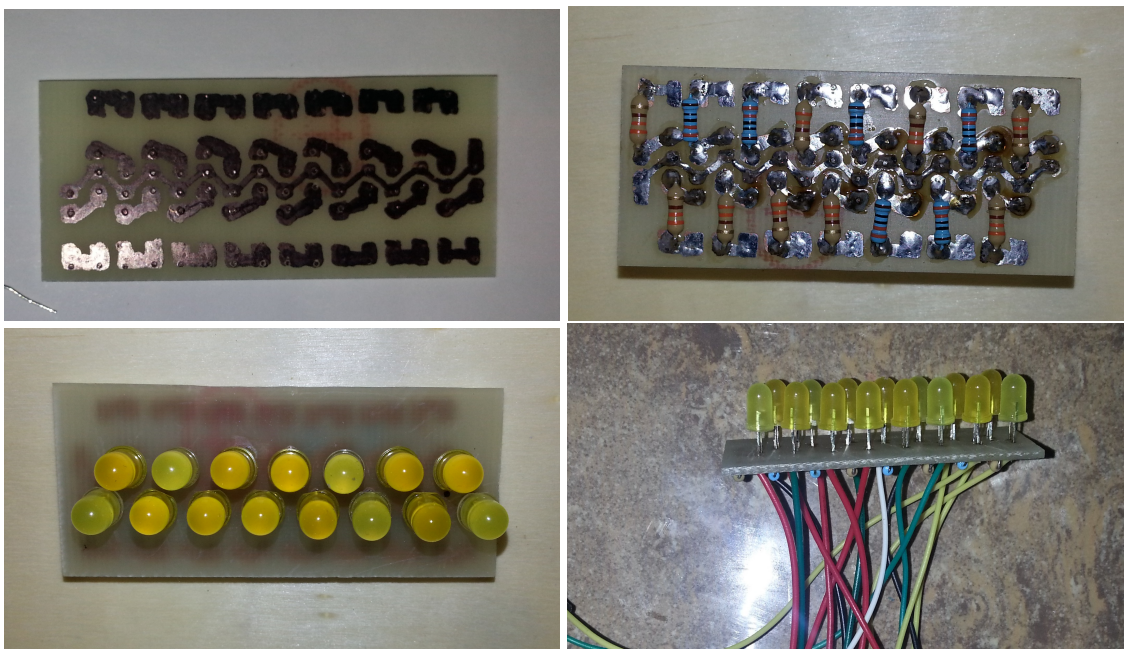


Figura 5.9: Montaje de los LED de acarreo

Los dos LED que quedan por montar son el verde y el rojo. El LED rojo será para alertarnos de un evento de máxima prioridad, el desbordamiento, mientras que el LED verde será de una prioridad baja avisando de que hemos completado la multiplicación. No se va a realizar el montaje en esta parte pues viene a ser una simplificación de lo que hemos hecho y ya se ha comentado cómo deben de ir.

Para finalizar este apartado queda hablar del otro visualizador, el de siete segmentos que, como su nombre indica, está formado por siete partes para mostrar un dígito o carácter. Cada una de estas partes esta formada por un LED que se enciende y apaga de forma individual. Por tanto vamos a tener que realizar conexiones para cada una de estas partes. Nos vamos a basar en el esquema de la Figura 5.10 donde se emplea una resistencia de 330 Ω para que no se queme.

Como se ve a la derecha de la Figura 5.10, el visualizador de siete segmentos tiene numerados cada uno de los LED que lo componen además de dos pines comunes. En nuestro caso, el visualizador es el *5161AS* con lo que estos pines son cátodo común o, dicho de otra forma, van conectados a masa. Para el montaje de esta parte es independiente conectar a masa uno de los dos por lo que se ha elegido conectarlo a la parte superior como aparecía en la imagen. De esta manera nos quedaría la parte superior del visualizador con cuatro conexiones más la resistencia a masa y la inferior con solo tres (no usamos el punto). Podemos ver tanto las patas superiores como inferiores en las imágenes de la Figura 5.11, izquierda y derecha respectivamente. Conectaremos los pines 28, 27, *GND* (con la resistencia), 22 y 23 a la parte superior y los pines 24, 25 y 26 a la parte inferior, ambas como aparecen en la imagen de izquierda a derecha. Con esto damos por finalizado el montaje de este visualizador y de la salida.

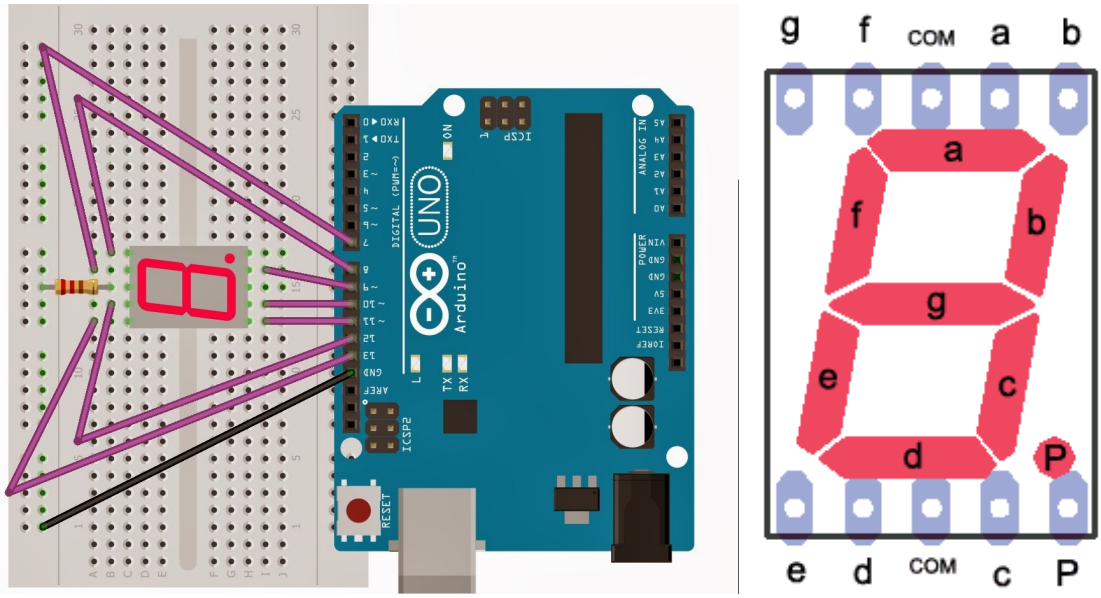


Figura 5.10: Esquema de conexión de un visualizador de siete segmentos

Figura 5.11: Montaje completo del visualizador de siete segmentos, patas superiores (izquierda) e inferiores (derecha)

5.5 Salida: funcionamiento

En esta parte vamos a explicar el código y por qué se ha elegido usar la forma de expresar la información que veremos en el visualizador así como los distintos LED informativos.

Para empezar hablaremos del visualizador, el cual es el encargado de mostrar la información más importante de la calculadora y cuyo montaje se ha visto en el apartado anterior. Una de las partes fundamentales que debe tener una calculadora es un sitio donde mostrar los cálculos, en este caso el visualizador LCD, el cual podemos ver en la Figura 5.12 mostrando la estructura básica que se le ha dado a la información que recibimos y calculamos con Arduino. Por una parte vemos que en la primera línea aparece una *S* seguida de un símbolo = representando que los dígitos que aparecen a continuación corresponden al valor de la salida, en este caso 0. La siguiente línea muestra el valor de la entrada, el que se lee a través de los potenciómetros, y al igual que con la salida viene precedida con los carácteres *E* y =. A continuación tenemos la linea encargada de mostrar el registro del contador, el cual sirve de anotación para multiplicaciones y divisiones y viene con una estructura similar a las anteriores. Para finalizar, vemos

la última línea con un solo símbolo ^ el cual se encarga de marcar en la línea superior donde se van a hacer incrementos o decrementos cuando se efectuen las distintas operaciones.

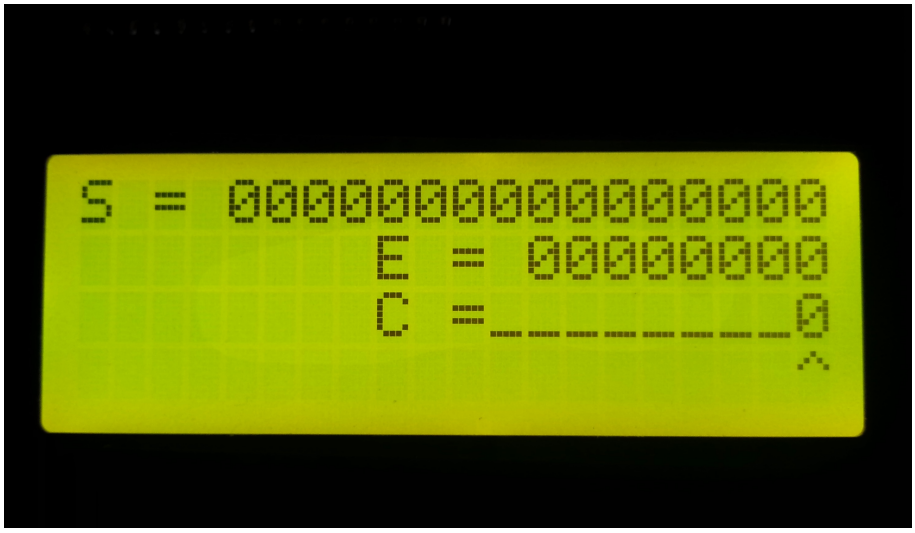


Figura 5.12: Visualizador LCD mostrando la estructura de los datos

Para realizar las distintas llamadas al visualizador LCD, se ha empleado la librería externa `LiquidCrystal_I2C.h`[1] la cual emplea el chip I2C para la comunicación y la librería interna de Arduino `Wire.h` necesaria para el funcionamiento de la anterior. La primera de las librerías, al ser externa, hay que instalarla como se ha visto en el capítulo anterior.

Para mostrar toda esta estructura comentada anteriormente, primero deberemos introducir en la función `setup()` la inicialización tanto del visualizador como la luz del fondo con las líneas de código 5.4, partiendo de que tenemos inicializada la variable `lcd` de forma global previamente.

```
    /*Inicializacion del visualizador*******/
    //Inicializacion del fondo retroiluminado
    lcd.backlight();
    //Iniciamos la pantalla
    lcd.init();
    /*************************************/
```

Código 5.4: Inicialización del visualizador

A continuación se hacen llamadas a distintas funciones que se encaragaran de imprimir cada una de las líneas. La primera línea se muestra a través de la llamada a la función que se muestra en 5.5, donde, la primera instrucción que vemos envía un mensaje al visualizador LCD para que cambie la posición actual del cursor a la fila de la salida que en este caso es la primera o la 0 y en el primer carácter que en este caso es el 0. Una vez colocado el puntero de escritura imprimimos la línea de carácteres $S =$ para darle un formato más adecuado y legible. Para finalizar, enviamos mensajes de impresión dentro de un bucle para que muestre los valores de las dieciséis cifras que componen la salida. Cabe destacar que se ha

[1] Librería disponible en: http://www.frostcode.es/descargas/LiquidCrystal_I2C2004V1.rar

preferido que muestre un 0 no significativo en vez de un blanco o cualquier otro carácter pues la calculadora original siempre mostraba los dígitos del 0 al 9 sin tener en cuenta si eran o no significativos.

```
void imprimirSalida()
{
    //Colocacion del cursor
    lcd.setCursor(0, filaSalida);

    //Embellecedor y clarificador de linea
    lcd.print("S = ");

    //Impresion de izquierda a derecha
    for (int numeroSalida = numeroSalidas - 1;
        numeroSalida >= 0;
        numeroSalida--)
    {
        lcd.print(salida[numeroSalida]);
    }
}
```

Código 5.5: Impresión de la salida

En cuanto a la siguiente línea, la de la entrada, se hace de forma similar, llamando a la función correspondiente y situando el cursor en la posición deseada, para que a continuación simplemente tengamos que mostrar el embellecedor de línea y las distintas cifras asociadas a la entrada leyendo el vector de entrada comentado anteriormente y enviando la información al visualizador LCD para que la muestre. Cabe destacar que esta línea no se comporta igual que la de la salida, pues se puede mover a izquierda o derecha mientras que la salida es estática. Para solucionar este pequeño problema se hace uso de una variable desplazamiento para rellenar el principio de la línea de blancos y el final. Además, esta variable es usada en otras funciones que comentaremos más adelante. Este cambio lo podemos ver en el código 5.6, donde además se ha decidido colocar la impresión de blancos para eliminar los carácteres residuales que quedaban a la derecha después de realizar un desplazamiento a la izquierda

```
    ...
//Relleno a la izquierda
for (int relleno = numeroEntradas - desplazamiento; relleno > 0;
relleno--)
{
    lcd.print(" ");
}
    ...
//Relleno a la derecha
for (int relleno = desplazamiento; relleno > 0; relleno--)
{
    lcd.print(" ");
}
```

Código 5.6: Cambios respecto a la salida en la impresión de la entrada

Por otro lado tenemos la impresión del contador que en este caso no está almacenado en un vector como los anteriores, sino que se encuentra en una variable de tipo `long`. Para imprimir este valor se ha necesitado el uso de una variable auxi-

liar que almacena la cantidad de dígitos que forman el contador. A partir de esta variable se han calculado la cantidad de símbolos _ (barra baja) que se muestran antes del valor del contador. Para empezar, se coloca el cursor justo una posición a la izquierda de la *E* de la línea anterior y se coloca un símbolo + o un − dependiendo de si el valor del contador es positivo o negativo. Esto se usa para diferenciar el valor mostrado cuando se hacen multiplicaciones o divisiones. Posteriormente colocamos el embellecedor que funciona para diferenciar qué datos van a mostrarse, en este caso *C* =. Una vez tenemos esto, se colocan las barra bajas comentadas anteriormente y finalmente el valor absoluto del contador, pues no queremos mostrar su signo. Dicho código se puede ver en 5.7.

```
1   void imprimirContador()
    {
3       lcd.setCursor(numeroEntradas - 1, 2);

5       if (contador > 0)
        {
7           lcd.print("+");
        }
9       else if (contador < 0)
        {
11          lcd.print("-");
        }
13      else
        {
15          lcd.print("_");
        }
17
        lcd.print("C =");
19      for (int i = 0; i < numeroEntradas - digitosContador + 1; i++)
        {
21          lcd.print("_");
        }
23      lcd.print(abs(contador));
    }
```

Código 5.7: Impresión del contador

Para finalizar, imprimimos en la última línea el símbolo ^ en la posición deseada, marcando de esta forma el lugar donde se efectuarán los incrementos y decrementos del contador como se ha hablado previamente. Para realizar esta colocación se ha usado el método empleado en la entrada de colocar blancos a ambos lados pues, al igual que la entrada, esta línea cambia dependiendo del desplazamiento. Se coloca el cursor justo debajo de la *C* de la línea anterior y se empiezan a colocar blancos. Podemos ver la colocación del carácter ^ en el código 5.8.

```
1
    for (int i = 0; i <= numeroEntradas - desplazamiento + 2; i++ )
3   {
        lcd.print(" ");
5   }

7   lcd.print("^");

9   for (int i = numeroEntradas * 2 - desplazamiento + 2; i <
    numeroSalidas + 2; i++)
    {
```

```
11          lcd.print(" ");
    }
```

Código 5.8: Colocación del marcador

Con esto terminamos con el código encargado de mostrar la salida en el visualizador LCD y pasamos a explicar el código del visualizador de siete segmentos el cual podemos ver en la Figura 5.13 mostrando un 6. Para esta parte se han necesitado varias funciones encargadas del cálculo intermedio para que se consiguiese mostrar el valor deseado en el visualizador de siete segmentos pero se van a explicar en apartados posteriores pues en este apartado solamente nos interesa la parte donde se muestra la salida de información.

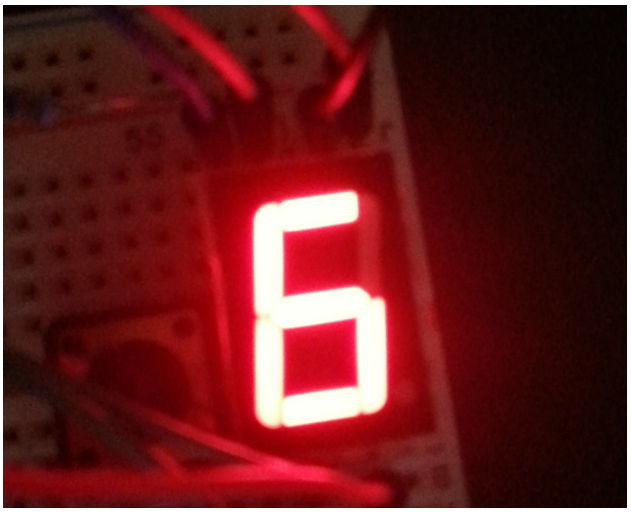


Figura 5.13: Visualizador siete segmentos con un 6

Como se ha comentado anteriormente, este visualizador esta formado por siete LED que tendremos que encender y apagar individualmente. Una posible aproximación y la más sencilla es el uso de un vector de siete posiciones que almacena cada LED si debe de estar encendido o apagado. Esto es lo que se va a usar para mostrar los diferentes caracteres seleccionados, en este caso solamente los dígitos del 0 al 9. Al igual que se ha hecho con anterioridad para mostrar los diferentes datos, hemos creado una función encargada exclusivamente para encender o apagar los LED del visualizador. Esta función es llamada en el bucle principal en cada iteración. Podemos ver el código de la función en 5.9 en el cual se recorre el anteriormente nombrado vector con el valor de cada LED y se envía al pin de cada uno de los LED del visualizador de siete segmentos si tiene que encenderse o apagarse. Podemos ver que se emplea la función `digitalWrite` la cual toma como argumentos el pin, el cual hemos calculado en base al primer pin del visualizador y la iteración del bucle, y el valor (0 o 1) que se le va a enviar pues es una función digital.

```
1   void muestraDisplaySieteSegmentos()
        {
3       //Recorre el array que contiene la codificacion
        for (int i = 0; i < 7; i++)
5       {
            digitalWrite(i + primerPinDisplaySieteSegmentos,
7               codificacionDisplaySieteSegmentos[i]);
```

```
        }
    }
```

Código 5.9: Mostrar dígitos en el visualizador de siete segmentos

Ahora pasaremos a explicar el LED asociado al visualizador de siete segmentos, el LED verde. Este LED se usa como un recordatorio o nota de que hemos completado la multiplicación. En cualquier caso, el usuario puede hacerle caso o continuar realizando operaciones pues la máquina no se bloqueará. Este LED se enciende cuando se realiza la llamada a la función encargada para ello. Solamente existe una llamada a este método y es a través de otro que describiremos más adelante encargado del cálculo del valor que se muestra en el visualizador anterior. Una vez se llama a esta función el LED se enciende durante cuatrocientos milisegundos, se apaga durante doscientos milisegundos y vuelve a estar encendido durante cuatrocientos más, de tal manera que el método dura un total de un segundo en ejecutarse. Este código se puede ver en 5.10.

```
        digitalWrite(ledMultiplicacionCompletada, HIGH);
        delay(400);
        digitalWrite(ledMultiplicacionCompletada, LOW);
        delay(200);
        digitalWrite(ledMultiplicacionCompletada, HIGH);
        delay(400);
        digitalWrite(ledMultiplicacionCompletada, LOW);
```

Código 5.10: Alerta al usuario de multiplicación completa

Por otro lado tenemos el LED rojo, encargado de avisarnos si se ha producido *overflow* o desbordamiento, es decir, hemos sobrepasado el valor máximo permitido por la calculadora o hemos pasado a números negativos. Este LED está gobernado por otra función que se llama cuando se da uno de los casos anteriores y que veremos con más detenimiento próximamente. Podemos ver el código de la función en 5.11. En este caso el LED permanece encendido durante un segundo y luego se apaga. Igual que con el LED de las multiplicaciones, este no es restrictivo pues no bloquea la máquina aunque tiene máxima prioridad pues, si lo ignoramos, nos haría tener todos los cálculos erróneos. En caso de que se encienda basta con realizar la operación contraria a la realizada actualmente, suma o resta, arreglando el valor mostrado en el visualizador LCD.

```
    void encenderLedOverflow()
    {
        digitalWrite(ledAlertaOverflow, HIGH);
        delay(1000);
        digitalWrite(ledAlertaOverflow, LOW);
    }
```

Código 5.11: Alerta al usuario de desbordamiento

Para finalizar este apartado, vamos a hablar de los LED amarillos encargados de mostrarnos las posiciones donde se requiere la intervención del usuario para realizar el acarreo manual. Disponemos de un total de quince de estos LED, uno para cada dígito de la salida a excepción del dígito menos significativo. Cada LED se enciende y apaga por separado y no interviene con sus vecinos. Para el control de estos LED tenemos otro método que acepta como entrada la posición respecto

a la salida del LED que debe de encenderse. El método lo podemos ver en 5.12. Igual que se ha hecho con el visualizador de siete segmentos, se ha realizado un cálculo para situar el pin a partir de la posición dada y el pin del primer LED de acarreo. Además, esta función coloca a `true` una variable encargada de bloquear la máquina cuando se enciende algún LED. A diferencia de las funciones anteriores, esta función se llama una vez por LED cuando se necesita haciendo que cada uno sea independiente de los demás.

```
1  void encenderLedAcarreoManual(int posicion)
   {
3      digitalWrite(posicion + primerLedAcarreo, HIGH);
       alertaAcarreoManual = true;
5  }
```

Código 5.12: Enciende el LED en la posición donde hay que hacer acarreo manual

Con esto ya tenemos explicado el funcionamiento de toda la salida, desde los visualizador, tanto LCD como de siete segmentos, hasta los distintos LED que intervienen en informar al usuario de lo ocurrido en la calculadora.

5.6 Cálculos intermedios

En esta sección se pretende abordar la parte del código encargada de realizar los cálculos entre que el usuario realiza una acción con la entrada hasta que ve su resultado en la salida. Para ello vamos a empezar hablando de las funciones que realiza cada botón y que internamente conllevan un cambio de estado de la calculadora.

Tenemos un total de ocho botones como se ha comentado anteriormente y cada uno de ellos permite realizar una acción. Empezaremos pues, por los más sencillos que se encarga de limpiar varios registros.

El primer código que vamos a explicar es el encargado de limpiar la salida, es decir, volver a colocar los valores iniciales por defecto que aparecen en el visualizador LCD en la linea donde se muestra la salida de nuestra máquina. Para que se active la función encargada de ello, necesitaremos pulsar el botón correspondiente. Podemos ver el código asociado a dicho botón en 5.13, donde se puede apreciar que tenemos dos vectores para los dígitos de la salida y uno que marca las posiciones bloqueadas. De los dos vectores de la salida, se emplea el `salida[]` para mostrar su valor en el visualizador LCD mientras que el otro se encarga de almacenar el valor correcto como su nombre indica. Este segundo vector es necesario pues, a la hora de hacer operaciones, cabe la posibilidad de que tengamos que hacer un acarreo manual y el valor del primer vector estará erróneo. Esto lo veremos más detenidamente en un apartado siguiente. Por último tenemos el vector de las posiciones bloqueadas encargado de almacenar en qué posiciones se ha producido un acarreo automático y que, por tanto, no pueden volver a sufrirlo. El código simplemente vacía cada posición de estos vectores a los valores por defecto, 0 y `false`.

```
   //Limpiamos el registro
2  salida[posicion] = 0;
   salidaCorrecta[posicion] = 0;
```

```
4
    //Limpiamos los bloqueos
6   posicionBloqueada[posicion] = false;
```

Código 5.13: Código para la limpieza de la salida

Pasamos ahora a explicar los dos últimos códigos de limpieza, el del contador y el de acarreo manual. Hablaremos primero del de limpieza del contador, el cual funciona de una forma similar al anterior. Podemos ver que ocurre cuando pulsamos el botón para limpiarlo en el código 5.14. En este caso tenemos también dos contadores distintos, el primero de ellos (`contador`) es el que almacena el valor que se muestra en el visualizador LCD, mientras que el segundo sirve de soporte para la transformación a distintas bases (binario, octal, decimal, etc.) y que veremos más adelante su uso. Vemos la aparición de otra variable que contiene la cantidad de dígitos que componen el contador, pues este último es una variable de tipo `long` y se necesita saber cuántos dígitos la componen para poder hacer una buena impresión. La cantidad de dígitos se reinicia a 1 pues el valor más pequeño que puede tomar el contador es un 0 y queremos que se muestre. Para terminar, aparece otra variable asociada al visualizador de siete segmentos, concretamente contiene el valor que se va a mostrar en él y que tiene que volver al valor 0 pues esta relacionado con el contador. La función que llamamos al final se explicará más adelante pero podemos decir que transforma el número que le pasamos a un vector que podemos leer para encender los LED del visualizador.

```
1   contador=0;
    contador_aux=0;
3   digitosContador = 1;
    numeroDisplaySieteSegmentos = 0;
5   codificaNumeroDisplaySieteSegmentos(0);
```

Código 5.14: Código para la limpieza del contador

Por último, el código del acarreo manual simplemente se encarga de apagar todos los LED de acarreo y reiniciar todos los valores asociados a ellos como ocurre con las dos funciones anteriores. El código lo podemos ver en 5.15 y a diferencia de los anteriores, este tiene una comprobación inicial en la que se mira si hay que hacer un acarreo manual y en caso afirmativo se desbloquean las posiciones bloqueadas. Esto se debe a que el usuario puede pulsar en cualquier momento el botón y eliminar los bloqueos cuando aún no son efectivos y se encienden los LED. En la calculadora mecánica, el usuario realizaba el acarreo manual cuando se veían las puntas de los pentágonos como se ha explicado en un capítulo anterior, pero no podía hacerlo si no se veían. En este caso las puntas de los pentágonos corresponden a los LED que estarán encendidos si la alerta está a `true`. Al final del bucle ponemos la alerta a `false` para indicar que ya no hay valores erróneos en la salida y que no se modifique el vector de posiciones bloquedas si no es estrictamente necesario otra vez.

Igual que se ha hecho con anterioridad, se usa el primer pin de los LED de acarreo junto a la iteración del bucle para calcular el pin al que corresponde el LED que hay que apagar. Para simplificar esto, se ha decidido recorrer todos los LED y apagarlos con la instrucción `digitalWrite`.

Finalmente se copia el valor de la salida correcta, la cual almacena los dígitos como si se produciese siempre el acarreo automático, sobre el vector que se muestra en el visualizador. Se ha elegido esta opción porque reduce en gran medida las comprobaciones y cálculos para determinar que valores de la salida están incorrectos.

```
    //No se puede realizar acarreo manual si no hay necesidad
    if (alertaAcarreoManual) {
        posicionBloqueada[posicion] = false;
    }
    digitalWrite(posicion + primerLedAcarreo, LOW);
    salida[posicion] = salidaCorrecta[posicion];

    //Fuera del bucle
    alertaAcarreoManual = false;
```

Código 5.15: Código para el acarreo manual

A partir de este punto, vamos a hablar de las funciones asignadas a los botones restantes, empezando por las más sencillas y terminando por las encargadas de realizar las operaciones de sumas y restas.

Empezaremos hablando del botón asignado al visualizador de siete segmentos, el cual simplemente se encarga de incrementar en 1 el valor que se muestra en él. Esto es una explicación resumida de lo que realmente ocurre internamente, pues el usuario solo va a percibir ese cambio.

Internamente, nos aseguramos que el valor que se muestra no sobrepase el valor 9 ni llegue a negativos pues nuestro visualizador de siete segmentos solo puede mostrar los dígitos del 0 al 9. Esto último se debe a que no se ha definido la codificación para otros símbolos. Cada vez que pulsamos dicho botón, se hace una llamada a la función 5.16 la cual toma como entrada un `$+1$` o un `$-1$`, en este caso le pasamos como entrada un `$+1$`, pues queremos que incremente su valor. Primero comprobamos que el valor no haya salido del rango establecido (0-9 pues la base por defecto es la decimal), en caso contrario asignamos un 0 a dicho valor. De esta manera, si pasamos de 9, volveríamos al 0 y también se impide que decremente por debajo de 0. Antes de asignar este valor, se comprueba si el valor que tiene es un 0, llamando a la función que enciende el LED verde de multiplicación completada si es así. Para finalizar vuelve a codificar el nuevo valor para que podamos mostrarlo en nuestro visualizador de siete segmentos.

```
    numeroDisplaySieteSegmentos += masOmenos;

    if (numeroDisplaySieteSegmentos <= 0 || numeroDisplaySieteSegmentos
    >= baseSistema)
    {
        if (numeroDisplaySieteSegmentos == 0)
        {
            encenderLedMultiplicacionCompletada();
        }
        numeroDisplaySieteSegmentos = 0;
    }
    codificaNumeroDisplaySieteSegmentos(numeroDisplaySieteSegmentos);
```

Código 5.16: Código para actualizar el visualizador de siete segmentos

Pasamos ahora a ver cómo se codifican dichos valores. Para ello, hay que explicar que cada uno de los valores que puede tomar nuestro visualizador se ha transformado a un valor entre 0 y 127 para que se pueda pasar a binario y almacenarlo en un vector que es el que se lee a la hora de mostrar el dígito. Podemos ver un pequeño trozo del código de esta función en 5.17 la cual acepta un valor y mediante un `switch` se asigna a la variable global `codigo` el valor que al pasarse a binario genera el dígito en nuestro visualizador. Finalmente, mediante el método para pasar de decimal a binario, asignamos a cada una de las posiciones de nuestra codificación el valor correcto. Podemos ver en el comentario que el valor 91 se traduce a 0101 1011 en binario el cual mostrará un 5. Cabe destacar que nuestro `switch` solo cuenta con las codificaciones del 0 al 9, por lo que nuestro sistema solamente puede trabajar con bases en las que aparezcan dígitos.

```
        ...
    case 5:
    {
        codigo = 91; // 1011011
        break;
    }
        ...
    int aux = codigo;
    for (int posicion = 0; posicion < 7; posicion++)
    {
        codificacionDisplaySieteSegmentos[6 - posicion] = aux %2;
        aux /= 2;
    }
```

Código 5.17: Codificación para el display de siete segmentos

Como bien sabemos, la calculadora Leibniz tiene una parte móvil y ésta va a ser emulada mediante una variable llamada `desplazamiento`. Esta variable ya ha sido comentada previamente y es usada entre otras cosas para calcular la posición donde debe colocarse en el visualizador LCD la línea de la entrada. Para cambiar el valor de esta variable se hace uso de los dos botones creados esclusivamente para ello. Ambos botones, al ser pulsados llaman a la misma función pero con un argumento de entrada diferente, entendiendo el carácter `$+$` como desplazamiento a la derecha y el carácter `$-$` a la izquierda. La función la podemos ver en el código 5.18 donde además de la dirección del desplazamiento, se acota el rango que puede tomar nuestra variable para que la línea de la entrada no se salga de la pantalla.

```
    //Derecha
    if (direccion == '+' && desplazamiento > 0)
    {
        desplazamiento--;
    }

    //Izquierda
    else if (direccion == '-' && desplazamiento < numeroEntradas)
    {
        desplazamiento++;
    }
```

Código 5.18: Código encargado de modificar el desplazamiento

A parte de esto último, se ha decidido que, al mover la entrada, el valor que muestra el visualizador de siete segmentos vuelva a ser 0. Esto ha sido así pues al desplazar la parte móvil en la máquina original, el usuario daba a entender que ya había terminado con las operaciones de multiplicación o division para la posición donde se encontraba, de tal manera que si deseaba realizar más cálculos en la nueva posición, partiría de cero. Podemos ver esto en las dos líneas finales del código 5.14 las cuales se añaden al final del método del desplazamiento y donde colocamos un 0, tanto en el valor del visualizador de siete segmentos como en el valor que se muestra en él.

Por último, queda hablar sobre la tarea de los botones de suma y resta, que como sus nombres indican, son los encargados de hacer estas operaciones. Por un lado tenemos el botón de suma que al ser pulsado llama a una función encargada de sumar todos los dígitos de la entrada sobre el valor de la salida. De igual forma ocurre con la resta.

Cada una de estas funciones llama a su vez a otra función encargada de realizar las operaciones sobre un solo dígito y que al final llamarán a otra función común encargada de detectar acarreos y solucionarlos en caso de que se tenga permiso. Podemos ver la función de resta de todos los dígitos en el código 5.19. En dicha función, primero se comprueba si hay desbordamiento. Este desbordamiento solo puede suceder cuando se intenta restar un número grande de uno más pequeño. La comprobación del desbordamiento se hace en otra función que explicaremos a continuación. Dentro del bucle vemos la llamada a la función de resta dígito a dígito, a la cual le tenemos que pasar la posición sobre la salida de la cual queremos restar y el valor que sustraemos. A primera vista puede parecer lioso por el uso de una instrucción condicional ternaria, pero ésta simplemente comprueba si en la posición donde vamos a restar hay correspondencia con la entrada, es decir, que en la entrada hay una cifra que está justamente debajo de dicha posición. En caso afirmativo, le restamos el valor asociado a la posición de la entrada y en caso contrario un 0. Finalmente, se actualiza el contador mediante una función dedicada a ello que veremos más adelante y el valor del visualizador de siete segmentos incrementándolo. Este incremento se debe a que al realizar divisiones aplicamos restas sucesivas y el valor del resultado aparece en dicho visualizador, el cual incrementa en 1 por cada sustracción.

```
1   void restar()
    {
3       if (hayOverflowRestando())
        {
5           encenderLedOverflow();
        }
7       for (int posicion = 0; posicion < numeroSalidas; posicion++)
        {
9
            //Los valores de la salida que no tienen
11          //correspondencia con la entrada se les resta 0
            restaDigito(posicion,
13                      (posicion - desplazamiento >= numeroEntradas) ?
                        0 : entrada[posicion - desplazamiento]);
15      }
        actualizarContador('-');
17      actualizaNumeroDisplaySieteSegmentos(+1);
    }
```

Código 5.19: Resta de todos los dígitos

Las diferencias en la suma con respecto al código anterior son la desaparición de la comprobación del desbordamiento pues esa función se encarga solo de las restas, y las llamadas a los demás métodos donde se cambia el argumento de entrada por el adecuado.

Vamos a adentrarnos más en el código de las restas que hemos visto anteriormente. Como habíamos dicho, nos faltaba explicar el funcionamiento de la resta dígito a dígito. El funcionamiento es simple: sustraemos de la posición de la salida el valor que le pasamos como entrada y luego comprobamos el acarreo. Este código lo vemos en 5.20 donde se sustrae tanto del vector que se muestra en el visualizador LCD como del que almacena el valor correcto, para finalmente llamar a la combrobación de acarreo.

```
1    //Restamos en la posicion de la salida el valor
     salida[posicion] -= valor;
3    salidaCorrecta[posicion] -= valor;

5    //Comprobamos si hay acarreo
     comprobarAcarreo(posicion, '-');
```

Código 5.20: Resta de un solo dígito

Como se ha dicho anteriormente, tenemos que explicar cómo funciona el método de detección de desbordamiento para restas. Este médodo simplemente comprueba de izquierda a derecha los valores de la entrada con los correspondientes valores de la salida, dígito a dígito, pues solo habrá desbordamiento si la entrada es mayor. Va recorriendo el vector hasta que encuentra un valor diferente de 0, en cuyo caso, si vemos que este valor es mayor que su correspondiente en la salida decidimos que hay desbordamiento y finalizamos. En caso de ser menor, finalizamos indicando que ha habido un desbordamiento. Existe la posibilidad de que ambos números sean iguales, con lo que se continuará la comprobación hasta que suceda una de las dos opciones anteriores o bien se recorra toda la entrada, en cuyo caso devolviendo un `false` pues no hay desbordamiento al ser ambos valores iguales.

La otra función nombrada anteriormente y que quedaba pendiente es la de actualizar el contador, en la cual interfieren varios métodos. Tenemos el código de dicho método en 5.21 que, al igual que muchas otras funciones, posee un parámetro de entrada indicativo de incremento o decremento.

Esta función es sencilla y a la vez complicada, tanto de entender como de explicar. Por una parte, necesitamos calcular el valor del incremento. Para ello realizamos un bucle con tantas iteraciones como desplazamiento. Como se puede ver, dentro del bucle hay una instrucción que multiplica de forma consecutiva la cantidad a sumar o restar del contador por una variable que contiene la base de nuestro sistema. Por defecto esta base es 10 (decimal). Posteriormente se decide si se incrementa o decrementa usando el argumento que le hemos pasado como entrada, para finalmente contar los dígitos que tiene nuestro nuevo contador.

Se ha podido observar un par de instrucciones que no se han explicado, pero que se van a comentar ahora. Por una parte se crea una nueva variable de tipo

`String` que va a contener el valor de nuestro nuevo contador en forma de cadena de caracteres. Esta transformación se hace mediante el uso de una función que transforma cualquier valor dado en base decimal a cualquier base que decidamos. En este caso, `contador_aux` contiene el valor del contador en base 10 (siempre) y es el que pasamos como entrada a la función previamente citada junto a la base de nuestro sistema, que como ya sabemos puede tomar cualquier valor en el rango de 2 a 10. La siguiente instrucción transforma el `String` en `int`. Esto último no es estrictamente necesario pero se ha preferido el uso de números al de cadena de caracteres. Finalmente se devuelve el signo al valor del contador.

Para comprender mejor por qué en el bucle inicial se multiplica por la base del sistema y se hacen esas transformaciones, vamos a explicarlo con un ejemplo. Imaginemos que nuestro sistema está en base 10, nuestro desplazamiento es 2 y estamos sumando. En este caso, tendremos que incrementar el valor de las centenas (desplazamiento 2) de nuestro contador en 1. Como el contador en tipo `long`, no podemos acceder directamente a su posición, por lo que tendremos que sumar 100. Dicho esto, el bucle haría dos iteraciones, dejando el valor de la variable `cantidad` en 100. Finalmente se aplica este incremento a la variable `contador_aux` que, como se ha dicho anteriormente, siempre almacena el valor en base 10. Seguimos con el cambio de base el cual devuelve un `String` que representa el mismo número pues hemos trasnformado de base 10 a base 10.

Lo anterior comentado parece coherente porque el sistema es base 10, pero en caso contrario no parece tan normal. Si en cambio decidimos cambiar la base a octal (base 8), la cosa no es tan coherente. Primeramente nos topamos con el bucle, el cual multiplicaría por 8 la cantidad tantas veces como desplazamiento tengamos. Digamos que tenemos el mismo ejemplo anterior solo que la base es 8, por tanto, el valor de `cantidad` pasaría a ser 64. Al principio este número puede parecer extraño y lo es, pero se hace esto por la transformación posterior. Digamos que el valor 10 en base 10 es 10 pero el mismo valor en base 8 es 8 en decimal y esto mismo ocurre con el valor 100, que en base 10 es un 100 pero en base 8 se transforma a un 64 en decimal. Esto mismo se aplica a cualquier base y cualquier potencia de la base.

Lo que hacemos en el bucle inicial, por tanto, es calcular el valor decimal (las variables son decimales) que tomaría la representación de una potencia de 10 en cualquier otro sistema. De tal manera, la representación del 1 000 en base octal sobre el sistema decimal sería de 512. Así obtenemos el valor decimal que al transformarlo a la otra base, nos devuelve un `String` ya con el cambio de base realizado que pasamos a número que, aunque sea decimal, decidimos representarlo como si fuese de otra base.

```
void actualizarContador(char masOmenos)
{

  //Calculamos la cantidad que hay que sumar o restar dependiendo del
    desplazamiento
  long cantidad = 1;
  for (int i = 1; i <= desplazamiento; i++)
  {
    cantidad *= baseSistema; //Necesario para la posterior
    transformacion
  }
```

```
10
   /*Decicimos si sumamos o restamos************/
12    ...
   /*******************************************/
14 String nuevoContador = cambioBase(abs(contador_aux), baseSistema);
   contador = nuevoContador.toInt();
16
   if(contador_aux<0)
18 {
     contador*=-1;
20 }

22 digitosContador = nuevoContador.length();
}
```

Código 5.21: Actualiza el contador

Todo lo comentado anteriormente sirve exclusivamente para obtener un cambio de base proyectado sobre un sistema decimal que es la representación que le da Arduino a los bits y poder usar esta proyección para mostrar al usuario unos valores como si realmente se hubiese realizado un cambio de base.

Nos queda una cuestión de la que hablar antes de terminar este apartado, la función encargada del acarreo. Esta función es la misma tanto para sumas como para restas, con la diferencia del parámetro de entrada que le pasemos, $+$ cuando estamos sumando y $-$ cuando restamos.

Vamos a empezar explicando la parte del acarreo en sumas. Para ello vamos a tener presente el código 5.22. Primero hay que comprobar si hay que realizar un acarreo. Comprobamos esto mirando si el valor en dicha posición es superior o igual a la base de nuestro sistema (por defecto 10). Si se da el caso, miramos que no esté bloqueada, o lo que es lo mismo, que se haya producido anteriormente un acarreo en dicha posición. Si está bloqueada la posición encenderemos el LED correspondiente. En cambio, si no lo está, procederemos con el acarreo al dígito siguiente, incrementando su valor en 1 y bloqueando la posición actual siempre y cuando no estemos en la posición más significativa, en cuyo caso encenderemos el LED de desbordamiento. Al final, restamos de la posición actual el valor de la base, 10 por defecto, dejando el valor en el rango de 0 a 9. Para terminar, hacemos lo propio con la salida con los valores correctos, mirando si la cifra en la posición actual ha sobrepasado a la base del sistema y realizando las operaciones pertinentes.

```
1     if (salida[posicion] >= baseSistema) //Realizamos acarreo
      {
3         if (!posicionBloqueada[posicion]) //Acarreo automatico
          {
5             if (posicion < numeroSalidas - 1) //No desbordamiento
              {
7                 salida[posicion + 1]++;
                  posicionBloqueada[posicion] = true;
9             }
              else //Desbordamiento
11            {
                  encenderLedOverflow();
13            }
          }
```

```
15          else //Acarreo manual
            {
17              encenderLedAcarreoManual(posicion);
            }
19
            salida[posicion] -= baseSistema; //Recalculamos el digito
        actual
21      }
        //NO realizamos acarreo
23
        //Calculamos de todas maneras el valor real de la salida
25      if (salidaCorrecta[posicion] >= baseSistema)
        {
27          salidaCorrecta[posicion + 1]++;
            salidaCorrecta[posicion] -= baseSistema;
29      }
```

Código 5.22: Control del acarreo en sumas

En cuanto al acarreo en restas, el código es bastante similar a excepción de que las comprobaciones se hacen en base al 0 y no a la base del sistema y no hay que comprobar si va a haber desbordamiento pues se comprueba antes de realizar las restas.

5.7 Cambio de base

Aquí expondremos una modificación que se le ha añadido a la calculadora para dotarla de más funcionalidad: el cambio de base mediante el cual el usuario podrá realizar operaciones en cualquier base inferior a la decimal. Para poder activarlo tendrá que introducir una serie de comandos. Empezaremos pulsando los botones de limpieza del contador y del registro de salida a la vez, con lo que nos llevará a una nueva pantalla donde aparecerá escrito *BASE: baseActual* siendo `baseActual` la base que se va a emplear. Podremos modificar este valor girando el potenciómetro del dígito más significativo, el izquierdo. Para terminar deberemos pulsar el botón de acarreo manual para aceptar los cambios. Podemos ver dicho menú en la Figura 5.14 donde a la izquierda tenemos la base decimal y a la derecha la octal. Cabe destacar que al aplicar los cambios todo lo que hayamos realizado con la calculadora se reiniciará, incluyendo el valor que tengamos en el registro así como en el contador.

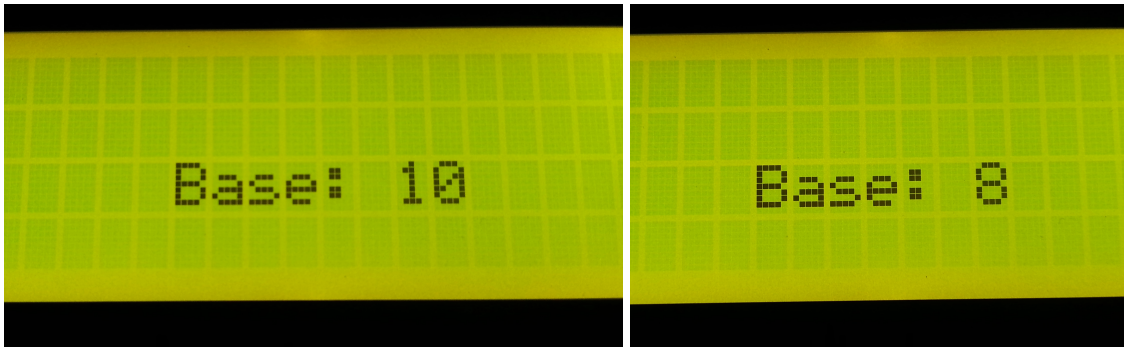


Figura 5.14: Pantalla del cambio de base

5.8 Montaje dentro de la caja

En este apartado final del montaje de la calculadora veremos cómo introducimos los conjuntos de componentes que hemos creado previamente dentro de una caja de madera para que nos quede un diseño compacto y a la vez agradable a la vista.

Primero de todo vamos a necesitar de dicha caja de madera, la cual se puede ver en la Figura 5.15 a la izquierda. A su derecha tenemos una plancha de metacrilato transparente a la que se le han hecho perforaciones para poder acoplar los LED, los visualizadores y los botones superiores, a la vez que se puede ver el interior y apreciar el cableado y la distribución.

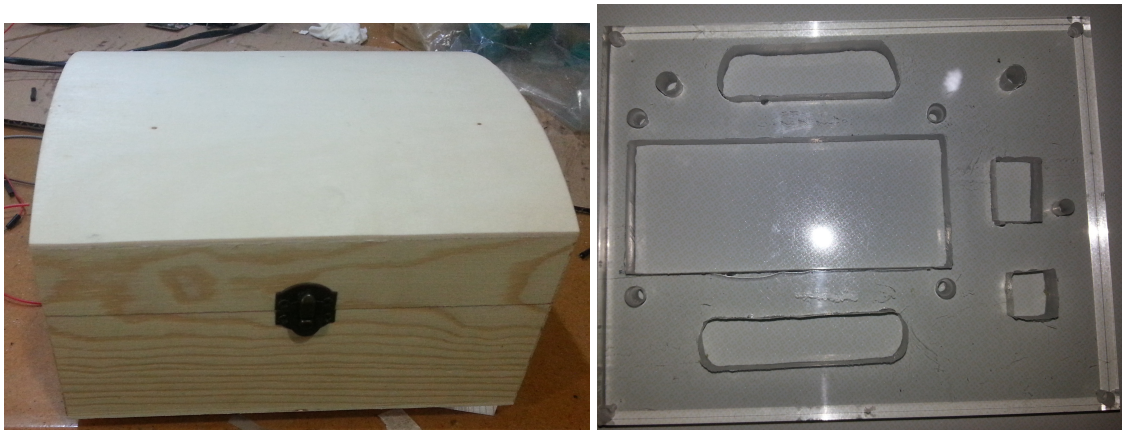


Figura 5.15: Caja y metacrilato empleados en el montaje

Una vez tenemos estos materiales, procederemos a introducir lo que hemos montado anteriormente. Empezaremos colocando los potenciómetros y botones laterales en la caja, haciendo los agujeros necesarios. Además, habrá que colocar el Arduino en el interior de la caja y hacer perforaciones para que podamos conectar el cable de alimentación. Continuaremos con el metacrilato, colocando en cada posición lo que corresponde, dando un acabado que podemos ver en la Figura 5.16 donde además vemos el interior de la caja en el paso actual. Se ha decidido trenzar los cables del metacrilato para reducir el espacio y darle un apartado visual más atractivo. Cabe destacar que los botones en esta parte del montaje llevan escritos unos símbolos que no se corresponden con su función y que se solventará al terminar el diseño completo.

Para finalizar con el montaje, se han colocado etiquetas a cada uno de los cables para saber en qué pin hay que colocarlos en caso de que se requiera un mantenimiento y se han realizado las distintas conexiones al Arduino. Podemos ver a la izquierda de la Figura 5.17 todos los cables y conexiones que hay en la caja, mientras que a la derecha vemos el diseño final de nuestra calculadora donde se aprecian los botones laterales, los potenciómetros sobresaliendo y los símbolos correctos en cada uno de los botones superiores.

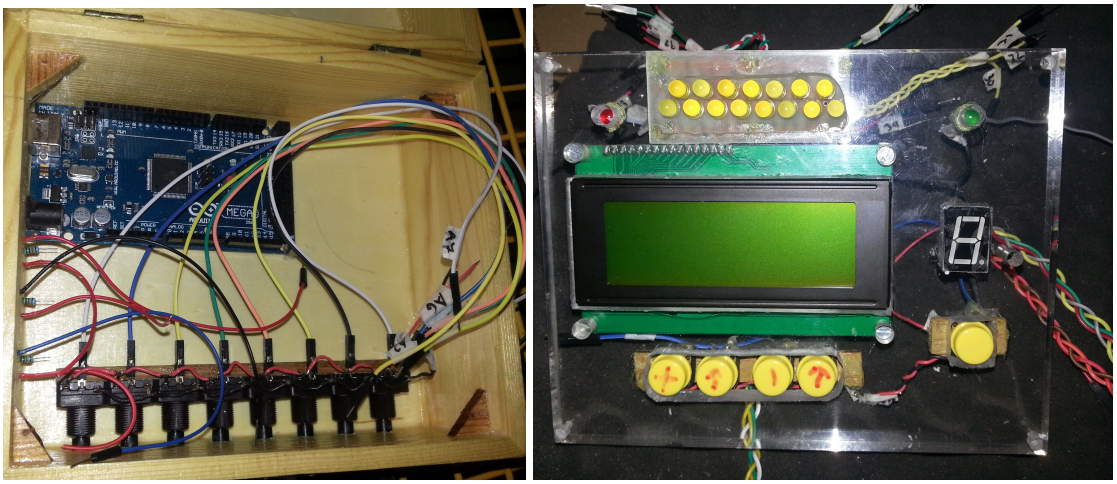


Figura 5.16: Colocación del interior de la caja y en el metacrilato

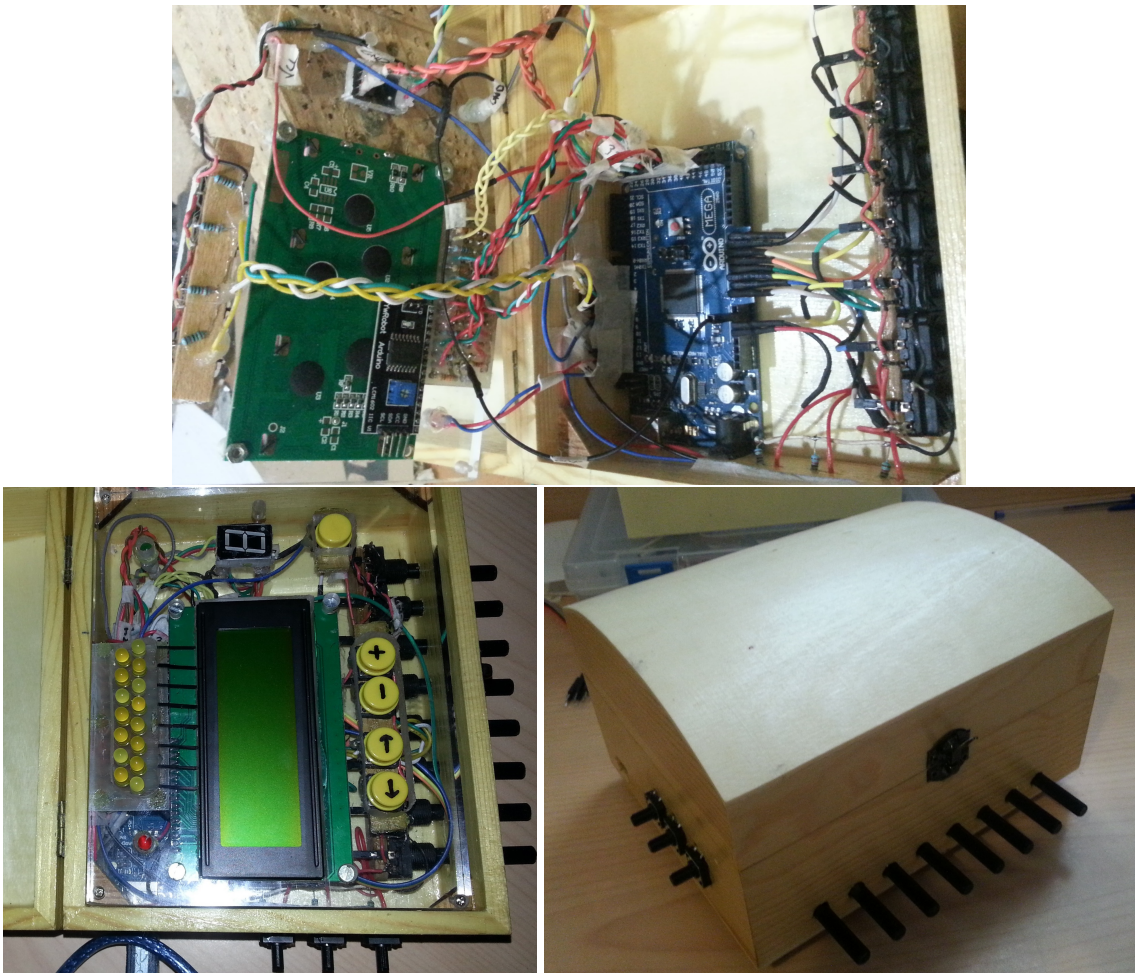


Figura 5.17: Conexiones completas y diseño final

5.9 Realización de las operaciones

En este apartado vamos a realizar las distintas operaciones que es capaz de realizar nuestra máquina y la original. Para ello nos vamos a basar en las ope-

raciones realizadas en un capítulo anterior y las vamos a replicar con nuestra máquina emuladora.

5.9.1. Sumas y restas

Para empezar, vamos a realizar una suma sencilla, la misma planteada en la sección 3.3 con la calculador original. En este apartado solamente nos centraremos en los registros de entrada y de salida, obviando el registro contador y el valor del visualizador de siete segmentos.

Realizaremos la suma de 489 y 7 123. Primero colocaremos el valor 489 en la entrada y accionaremos el botón de suma. El resultado se muestra en la imagen izquierda superior de la Figura 5.18.

Continuaremos colocando el segundo valor, 7 123 y, aplicando el mismo procedimiento, obtendremos la imagen superior derecha de la Figura 5.18, en la cual podemos apreciar que no hay que realizar acarreo manual por ser la primera operación realizada y en que se ha añadido correctamente este nuevo valor al que teníamos en la salida. Tendremos el valor 7 612 en la salida.

Para comprobar mejor el funcionamiento de los acarreos, procederemos a sumar otro valor al que tenemos actualmente. Sumaremos 6 238 al 7 612 que tenemos en la salida. Esto resultará en la imagen inferior izquierda de la Figura 5.18 donde ya podemos ver en funcionamiento el sistema de acarreo manual.

Vemos cómo se ha encendido el LED de las decenas, que corresponde al valor 4 que tenemos en la salida, indicando que ese dígito es incorrecto y hay que corregirlo realizando un acarreo manual. Para ello pulsaremos el botón lateral superior y obtendremos el valor correcto después de realizar el dicho acarreo. Veremos en la salida la última imagen de la Figura 5.18 donde se indican con flechas roja (la grande para el botón de acarreo manual) los distintos botones laterales y, con ello, dando por finalizada la suma con un resultado de 13 850.

En cuanto a las restas, se realizan de forma similar, con los mismos problemas que conlleva sumar y solventándolos con el mismo procedimiento.

5.9.2. Multiplicaciones

Continuaremos este capítulo realizando la multiplicación que habíamos completado en la sección 3.4 pero esta vez empleando nuestra emuladora. En este apartado sí prestaremos atención tanto al contador como al visualizador de siete segmentos.

Multiplicaremos 345 por 23 realizando los correspondientes desplazamientos y operaciones. Primero colocaremos uno de los dos valores, en este caso elegimos el mayor (multiplicando) para reducir el número de operaciones. Por tanto, colocamos 345 en la entrada tal y como se ve en la imagen superior izquierda de la Figura 5.19, además de incrementar el valor del visualizador de siete segmentos a 3 pues es el dígito menos significativo y el primero por el que multiplicar del segundo valor, 23 (multiplicador).

Aplicaremos sumas sucesivas hasta que nos lo indique el LED verde de multiplicación completa. Con esto obtendremos la imagen de la superior derecha de Figura 5.19 donde el visualizador de siete segmentos ha alcanzado el valor 0 pues se ha completado la multiplicación y el contador almacena un 3, que es el número por el que hemos multiplicado actualmente.

Aplicaremos seguidamente el desplazamiento a la izquierda una posición e introduciremos en el visualizador anterior el valor 2 del siguiente dígito del multiplicador obteniendo la imagen inferior izquierda de la Figura 5.19.

Finalmente aplicaremos las sumas sucesivas en esta nueva posición hasta que nos lo indique el LED verde. Obtendremos la imagen inferior derecha de la Figura 5.19 donde podemos ver en el contador el valor del multiplicador y dar por terminada la operación con el resultado final de 7 935. Cabe destacar que, a lo largo de esta operación han surgido acarreos manuales que han debido de ser corregidos con el botón correspondiente, igual que ha sucedido con la suma.

5.9.3. Divisiones

Terminaremos este apartado realizando la misma división que habíamos realizado en la sección 3.5. En este caso se emplearán los mismos instrumentos que se habían usado con las multiplicaciones además de hacer servir el botón de reinicio del contador.

Dividiremos 723 entre 35 y para ello primero deberemos decidir cuantos decimales queremos, en este caso, uno. Desplazaremos a la izquierda tantas posiciones como decimales queramos introducir. Colocaremos el dividendo en la entrada y sumaremos para que salga en la salida. Ahora colocaremos el divisor en la entrada y desplazaremos a izquierda o derecha para que el dígito más significativo de ambos coincida en posición y reiniciaremos el contador pulsando el botón lateral correspondiente, pues va a almacenar el resulatado y queremos que empiece en 0. Veremos algo similar a la imagen superior izquierda de la Figura 5.20 donde la flecha roja grande indica la posición del botón lateral de reinicio del contador.

Procederemos a realizar restas hasta que nos demos cuenta de que el valor no se puede seguir dividiendo o que nos avise el LED rojo de desbordamiento. En caso de que nos avise el LED, simplemente deberemos realizar la operación contraria, en este caso, una suma. Podemos ver la situación actual en la imagen superior derecha de la Figura 5.20 donde además se aprecia el valor 2 en el visualizador de siete segmentos indicativo de que el primer dígito del cociente es un 2.

Continuaremos desplazando a la derecha una posición y realizando restas. En este caso vemos que no se puede restar por ser inferior el valor al divisor, con lo que sabemos que el siguiente dígito es un 0. Desplazamos una vez más y comprobamos que ya se puede restar obteniendo la imagen inferior izquierda de la Figura 5.20.

Puesto que nos encontramos a la derecha del todo en la entrada, sabemos que este va a ser nuestro último dígito. Realizaremos las restas necesarias hasta que no podamos más y obtendremos la imagen inferior derecha de la Figura 5.20 donde

el último dígito es el 6 que aparece en el visualizador mientras que en el registro contador tenemos el resultado final, 206, del cual una de esas cifras corresponde a la parte fraccionaria que habíamos elegido calcular. Por tanto, nos queda un valor final de $20,6$ como resultado de dividir 723 entre 35 con precisión de un decimal.

Cabe destacar que se han obviado los acarreos manuales de igual manera que con las multiplicaciones pues ya se han explicado en la suma y se sabe cuándo hay que proceder a su realización.

Figura 5.18: Suma con la emuladora

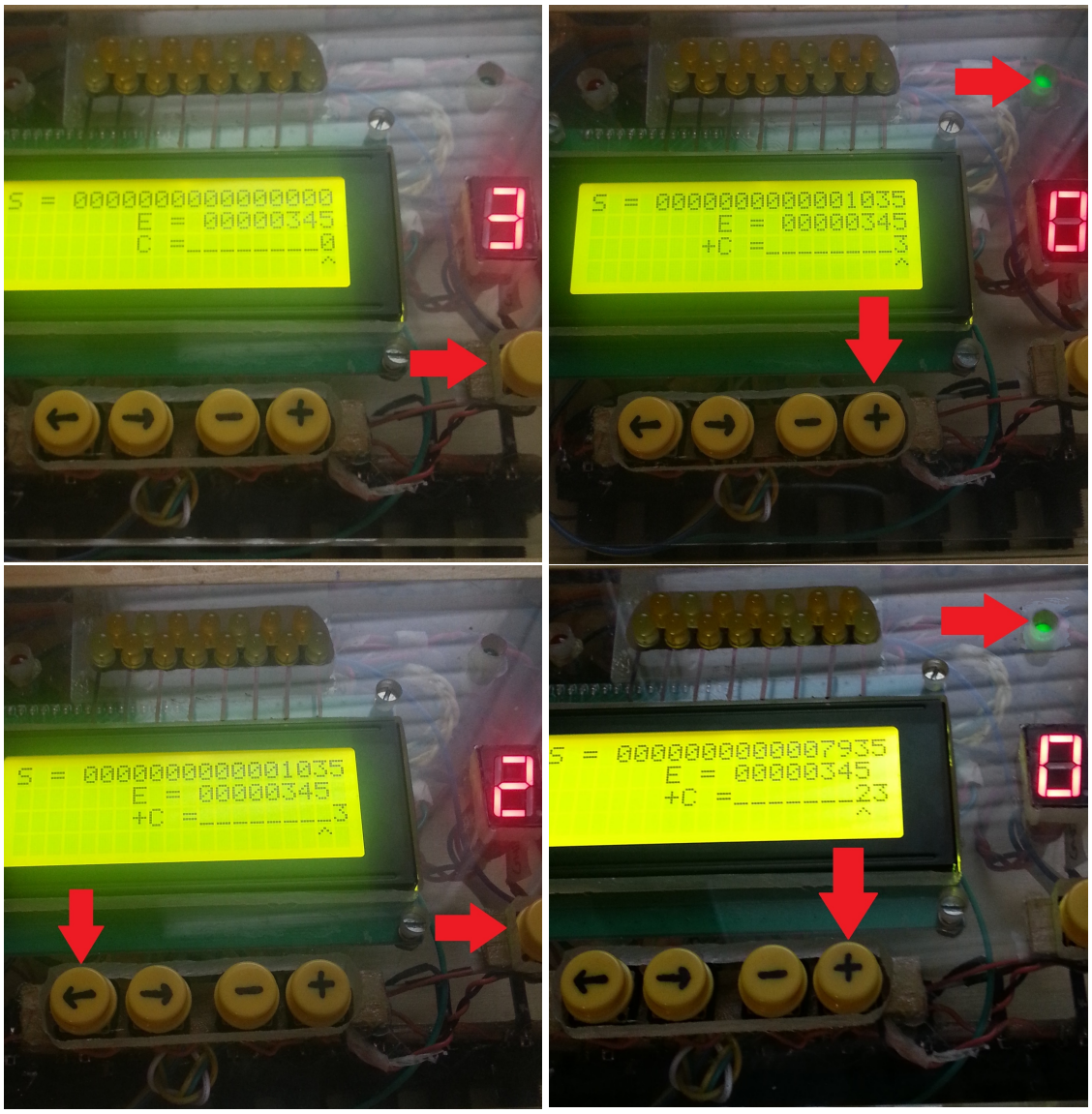


Figura 5.19: Multiplicación con la emuladora

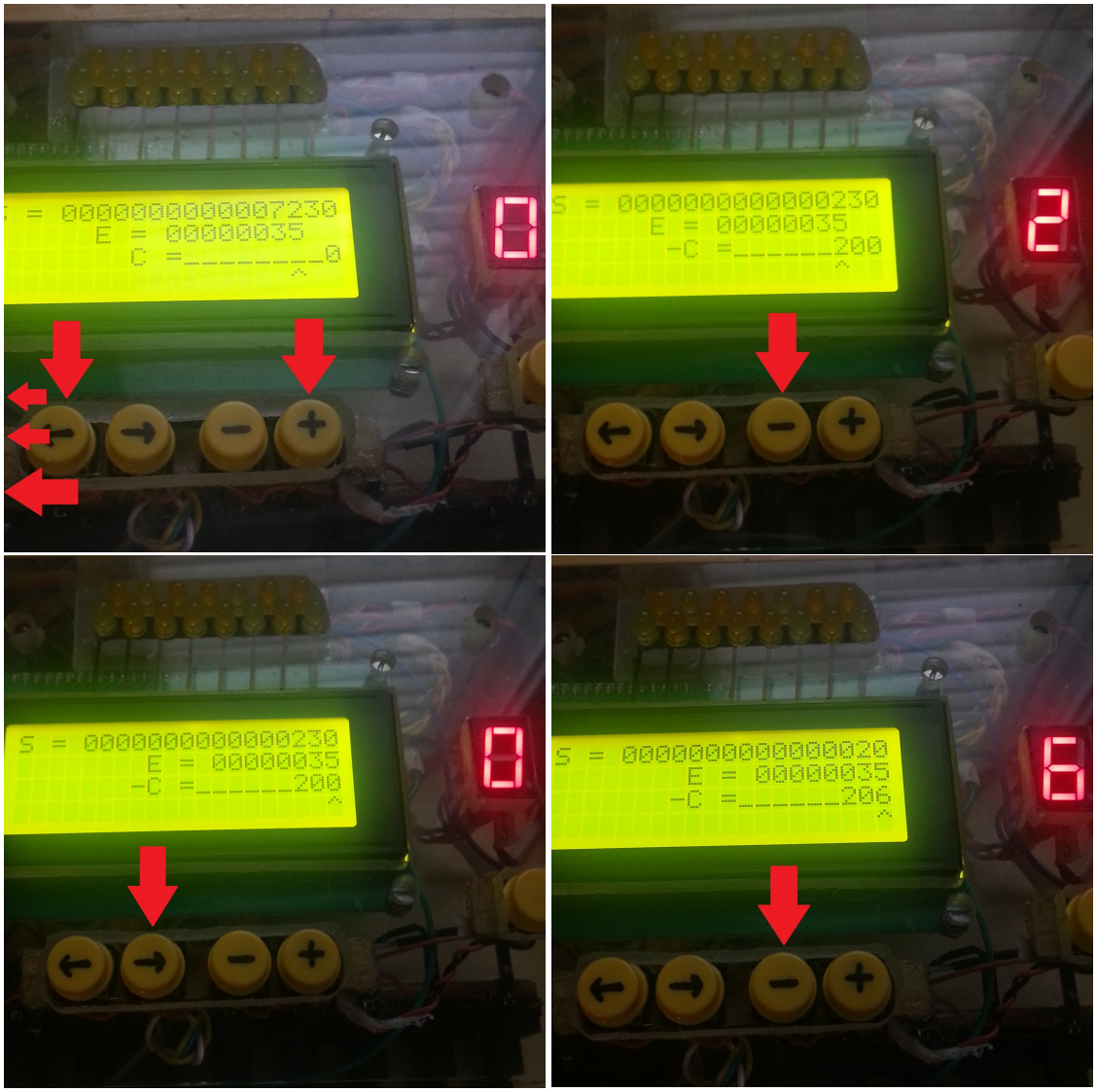


Figura 5.20: División con la emuladora

CAPÍTULO 6
Conclusiones

En este capítulo final incluimos un apartado en el que se hablará de los distintos porblemas que hemos sufrido con la realización de este proyecto. Además, se plantean las consideraciones finales obtenidas mediante el desarrollo de este trabajo, realizando un breve resumen de todo lo que hemos logrado y comparándolo con los objetivos iniciales.

6.1 Contratiempos y problemas sufridos

A lo largo del desarrollo del trabajo hemos sufrido algunos problemas y contratiempos con respecto a la parte *software* y *hardware* de Arduino. A continuación se expone una breve descripción y resumen de estos problemas además de la solución encontrada.

- **Arquitectura del Arduino:** Para empezar, se pensaba que Arduino Mega 2560 estaba basado en una arquitectura de 32 bits por lo que se planteó usar la entrada en un formato `int`. Posteriormente nos dimos cuenta de que su arquitectura era de 16 bits con lo que con un `int` como entrada nos quedabamos cortos. Se acabó usando la variable de tipo `long` para referirnos a enteros grandes, esto es, de 32 bits.

- **Realización de operaciones:** En un principio se pensó usar dos variables diferentes, una para la entrada y otra para la salida, pero nos quedabamos cortos con la salida pues necesitabamos dieciséis dígitos y lo máximo que podíamos obtener con el tipo `long` eran diez. Planteamos dividir la salida en un vector de dos posiciones y operar sobre él pero resultaba muy complicado gestionar y marcar los acarreos manuales. Finalmente llegamos a la conclusión de usar un vector para la entrada con ocho posiciones y otro de dieciséis para la salida y realizar las operaciones dígito a dígito.

- **Poco control de excepciones:** Llegamos a un punto en el que la salida tomaba valores extraños y no sabíamos por qué era. Resulta que al realizar operaciones, había veces que los accesos del programa se salían de un vector en memoria con lo que escribía directamente en la siguiente posición sin comprobar que pertenecía al mismo vector. La solución fue aumentar la

cantidad de espacios de los distintos vectores al doble, con lo que si volvía a ocurrir escribiría en posiciones de memoria que nunca usaríamos.

- **Falta de conexión:** Montando el primer prototipo de la calculadora, usábamos una placa de conexiones donde clavar los diferentes cables y componentes. Dado el volumen de cables necesarios y componentes, nos quedamos sin espacio y hubo que conectar algunas partes «al aire» con lo que se producían interferencias. Solucionamos esto al decidir que se iba a soldar todo con lo que las partes «al aire» fueron las primeras en soldarse y se pudo probar el primer prototipo.

- **Creación del metacrilato:** Uno de los problemas que más miedo nos daba fue la creación del los distintos agujeros del metacrilato, pues solo disponíamos de un panel. Al principio los LED de acarreo tenían sus agujeros independientes y lo mismo ocurría con los botones pero luego no encajaban o se caían con demasiada facilidad. Lo solucionamos colocando una varilla de madera a modo de soporte y que se pegó al propio metacrilato. Con los visualizadores se pretendía pegarlos directamente por debajo del metacrilato, ahorrando algunos agujeros pero, dado el tamaño de la caja, se tuvieron que realizar para que cupiese todo.

- **Modificación de la librería I2C:** Al empezar a usar la librería I2C para el proyecto nos dimos cuenta que la función `print` definida en dicha librería solamente permitía imprimir en el visualizador LCD un caracter, con lo que imprimir una linea entera requería de varias llamadas a `print`. La solución la encontramos buscando por los foros de Arduino donde otros usuarios habían tenido el mismo problema con librería. Dicha solución[1] consistía en modificar la propia librería, cambiando el valor de retorno de la función `write` a 1, indicando a las capas superiores que había imprimido con éxito un caracter.

6.2 Consideraciones finales

Aquí comprobaremos los distintos conocimientos que hemos adquirido además de comprobar que se han cumplido los distintos objetivos propuesto.

I La invención de la calculador de Leibniz fue un hecho innovador pues fue la primera calculador descubierta[2] capaz de realizar las cuatro operaciones básicas: sumas, restas, multiplicaciones y divisiones (incluso raíces si sabemos cómo).

II El propio creador de esta calculadora además contribuyó enormemente al desarrollo de la ciencia, tanto en el apartado del cómputo con la invención del sistema binario, como en las matemáticas con el desarrollo de las integrales y derivadas dotándolas de una nomenclatura clara y sencilla.

[1] Solución encontrada en: http://forum.arduino.cc/index.php?topic=361411.0
[2] El reloj calculante no se descubrió hasta trescientos años después de su invención.

III Conocemos más acerca de la vida del autor, como la vivió, los viajes realizados y las amistades que fue consiguiendo que le proporcionaron el conocimiento necesario para seguir desarrollando el suyo propio.

IV Hemos aprendido acerca del marco y contexto histórico del siglo XVII, el siglo de la revolución científica, donde se produjeron los avances más importantes que llevaron al abandono de las creencias antiguas para dar paso al nuevo pensamiento mejorando enormemente el desarrollo científico.

V Se ha aprendido a fondo el funcionamiento interno de la calculadora mecánica así como la realización de las distintas operaciones básicas para después emularlas con la construcción de este proyecto.

VI El desarrollo de la calculadora usando Arduino ha conseguido que aprendamos más acerca de las características técnicas de los distintos modelos y saber cuál es el más apropiado para cada uno de los proyectos que tengamos pensados realizar además de aprender a programar en esta plataforma.

VII El desarrollo del código junto a la construcción de las distintas partes de la calculadora nos ha enseñado la importancia del *software* junto al *hardware* consiguiendo una completa integración entre ellos.

VIII Durante el desarrollo completo del proyecto se ha conseguido emular todas las características importantes de la máquina original (la que ha sobrevivido) además de los defectos que dotaban a la calculadora de carácter propio como es el acarreo manual.

Dicho todo esto, podemos concluir que se han alcanzado todos los objetivos propuestos al inicio del trabajo.

6.3 Trabajo futuro

Ahora hablaremos de los posibles trabajos futuros como ampliación de este. Por lo tanto, tendrán una relación directa con el actual trabajo o pueden ser de interés para el pripio museo.

Una posible ampliación directa sería el desarrollo de esta misma calculador usando el lenguaje de programación Scratch[3] desarrollado por el MIT para facilitar el aprendizaje de la programación a los más pequeños. Usando este lenguaje podríamos crear una página en la web del Museo de Informática de la Escola Tècnica Superior d'Enginyeria Informàtica de la Universitat Politècnica de València donde colocar este emulador y poder usarlo desde cualquier lugar.

Otro trabajo futuro podría centrarse en el desarrollo de otras calculadoras mediante la plataforma Arduino para que el propio museo tuviese una emuladora como la construida en este trabajo y que los estudiantes pudiesen ver cómo funcionaba y tenerla entre sus manos.

[3]Enlace a la página oficial de Scratch en: https://scratch.mit.edu/

Bibliografía

[1] Vetus ALM. El reloj calculador de Schickard. *Informática básica*, 29 de agosto de 2010.

[2] Especificaciones técnicas de los distintos modelos de Arduino. Consultado en https://www.arduino.cc/en/ el 27 de octubre de 2016.

[3] Joseph Balsach Peig. *De la mano al cálculo electrónico. 4000 años calculando.* Edición personal del autor, 2010.

[4] Miquel Barceló *Historia de la Informática.* UOC (UNIVERSITAT OBERTA DE CATALUNYA), Cataluña, edición del 2008.

[5] Entrada en la enciclopedia libre, modificada el 12 de octubre de 2016. Consultado en http://enciclopedia.us.es/index.php/%C3%81baco_neperiano .

[6] Juan D. Godino. *Matemáticas para maestros.* Departamento de Didáctica de la Matemática. Facultad de Ciencias de la Educación. Universidad de Granada.

[7] Georges Ifrah. *Historia universal de las cifras*. Espasa Calpe, S.A., Madrid, sexta edición, 2008.

[8] Sandra Isabel Jiménez Mateos. El «reloj calculante», la primera calculadora de la historia, construida en 1623. *Libera el conocimiento*, 20 septiembre del 2015.

[9] Biography of Gottfried Leibniz (1646-1716) [Recurso Web]. Consultado el 11 de octubre de 2016 en http://history-computer.com/People/LeibnitzBio.html .

[10] Daniel Lozano Equisoain. *Arduino Práctico*. Ayana Multimedia, España, 2016.

[11] J. J. O'Connor y E. F. Robertson. Gottfried Wilhelm von Leibniz. *School of Mathematics and Statistics University of St Andrews, Scotland, 1998* .

[12] Pascalina. Consultado en https://es.wikipedia.org/wiki/Pascalina .

[13] Francisco Luis Redondo Álvaro. *Algunos rasgos de la revolución científica en el siglo XVII*. Consultado el 18 de octubre de 2016 en https://dialnet.unirioja.es/descarga/articulo/2986385.pdf .

[14] Siglo XVII. Consultado en http://www.ujaen.es/investiga/cts380/historia/siglos_xvii.htm .

[15] The Stepped Reckoner of Gottfried Leibniz [Recurso Web]. Consultado el 10 de octubre de 2016 en http://history-computer.com/MechanicalCalculators/Pioneers/Lebniz.html .

[16] Vicente Trigo Aranda. *Del ábaco a Internet*. Creaciones Copyright S.L., España, 2010.

[17] Germán Tojeiro Calaza. *Taller de Arduino: un enfoque práctico para principiantes*. MARCOMBO, S.A., España, 2014.

[18] Michael R. Williams. *A history of computing technology*. IEEE Computer Socienty, Los Alamitos, California, segunda edición, 1997.

APÉNDICE A
Código de la calculadora Leibniz

```
/*Librerias necesarias*****************************************/
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

/*************************************************/

/*Conjunto de variables necesarias
*****************************************/

////EXTRA/////
/*Variable para sistema usado*************/
int baseSistema = 10; //Problema al usar mas de 10
/*****************************************/

/*Variables 7 segmentos*********************************/
int codificacionDisplaySieteSegmentos[7] = {1, 1, 1, 1, 1, 1, 0};
//Valor del 0
byte numeroDisplaySieteSegmentos = 0;
int codigo = 0; //Codigo del digito a mostrar

//NOTA: Los pines del visualizador de siete segmentos van desde el
//22 hasta el 29
int primerPinDisplaySieteSegmentos = 22;
int ultimoPinDisplaySieteSegmentos = primerPinDisplaySieteSegmentos
+ 7;

//LED de multiplicacion completada
int ledMultiplicacionCompletada = 12; //led en pin 12
/*********************************************************/

/*Variables del visualizador***************/
const int direccionDisplay = 0x27;
const int filaSalida = 0;
const int filaEntrada = 1;
const int caracteres = 20;
const int filas = 4;

//Configuramos la direccion de nuestro visualizador y las medidas
LiquidCrystal_I2C lcd(direccionDisplay, caracteres, filas);
/***********************************/

/*Entradas de los potenciometros*************************/
const int numeroEntradas = 8;
```

```
    const int pinEntrada[] = {A0, A1, A2, A3, A4, A5, A6, A7};
42  /*********************************************************/

44  /*Variables led alerta overflow***********/
    const int ledAlertaOverflow = 13; //led en pin 13
46  /*******************************/

48  /*Variables boton suma*************/
    const int botonSuma = 5; //boton en pin 5
50  bool pulsadoBotonSuma = false;
    /*********************************/
52
    /*Variables boton resta***********/
54  const int botonResta = 6; //boton en pin 6
    bool pulsadoBotonResta = false;
56  /*******************************/

58  /*Variables boton reset***********/
    const int botonReset = 2; //boton en pin 2
60  bool pulsadoBotonReset = false;
    /*******************************/
62
    /*Variables boton reset contador***********/
64  const int botonResetContador = 9; //boton en pin 9
    bool pulsadoBotonResetContador = false;
66  /*****************************************/

68  /*Variables boton acarreo*****************/
    const int botonAcarreoManual = 7; //boton en pin 7
70  bool pulsadoBotonAcarreoManual = false;
    /*************************************/
72
    /*Variables boton 7 segmentos**********************/
74  const int botonDisplaySieteSegmentos = 8; //boton en pin 8
    bool pulsadoBotonDisplaySieteSegmentos = false;
76  /***************************************************/

78  /*Variables registro entrada***************/
    int entrada[numeroEntradas];
80  /*******************************************/

82  /*Variables registro salida*********************/
    const int numeroSalidas = 16;
84  int salida[numeroSalidas * 2]; //longitud doble porque al salirse
    del array opera sobre el array anterior (BRAVO!!!!)
    int salidaCorrecta[numeroSalidas + 1];
86  /*************************************************/

88  /*Variables registro de acarreo***************/
    bool posicionBloqueada[numeroSalidas + 1];
90  bool alertaAcarreoManual = false;

92  //NOTA: Los LED de acarreo van desde el 30 hasta el 45
    int primerLedAcarreo = 30;
94  int ultimoLedAcarreo = primerLedAcarreo + numeroSalidas - 1;
    /*****************************************/
96
    /*Variables contador***************/
98  long contador = 0;
```

```
    long contador_aux = 0;
100 int digitosContador = 1;
    /*******************************/
102
    /*Variables desplazamiento****************/
104 //Desplazamiento a la izquierda
    int botonDesplazamientoIzquierda = 3; //Boton en el pin 3
106 bool pulsadoBotonDesplazamientoIzquierda = false;

108 //Desplazamiento a la derecha
    int botonDesplazamientoDerecha = 4; //Boton en el pin 4
110 bool pulsadoBotonDesplazamientoDerecha = false;

112 //Posicion del desplazamiento
    int desplazamiento = 0;
114 /*****************************************/

116 /************************************************/

118 /*
        Funcion encargada de llevarnos a un menu donde
120     elegir la base que queremos usar en nuestro sistema
    */
122 void menuBase()
    {
124     lcd.clear();
        int baseActual;
126     while(!pulsadoBotonAcarreoManual)
        {
128         baseActual= map(analogRead(pinEntrada[0]), 0, 1024, 2, 11);
            lcd.setCursor(7, 2);
130         lcd.print("Base: ");
            lcd.print(baseActual);
132         lcd.print("  ");
            pulsadoBotonAcarreoManual= digitalRead(botonAcarreoManual);
134         delay(50);
        }
136     baseSistema = baseActual;
    }
138


140 /*
        Funcion encargada de pasar de decimal a cualquier otra base
142
        @param valor - Numero en decimal para transformar
144     @param base - Base a la cual se va a transformar
    */
146
    String cambioBase( long valor, int base) {
148     static char baseChars[] = "0123456789";
        String resultado = "";
150     do {
            //Anyadir a la izquierda
152         resultado = String(baseChars[valor %base]) + resultado;
            valor /= base;
154     } while (valor != 0);
        return resultado;
156 }
```

```
158
159     /*
160         Enciende el LED de los acarreos manuales correpondiente
161
162         @param posicion - Posicion donde hay
163         que realizar acarreo manual
164     */
165     void encenderLedAcarreoManual(int posicion)
166     {
167         digitalWrite(posicion + primerLedAcarreo, HIGH);
168         alertaAcarreoManual = true;
169     }
170
171     /*
172         Enciende el LED del Overflow
173     */
174     void encenderLedOverflow()
175     {
176         digitalWrite(ledAlertaOverflow, HIGH);
177         delay(1000);
178         digitalWrite(ledAlertaOverflow, LOW);
179     }
180
181
182     /*
183         Funcion que calcula si hay overflow cuando
184         restamos. Se basa en el hecho de que no podemos restar
185         un numero grande de uno pequenyo
186     */
187     bool hayOverflowRestando()
188     {
189         for (int posicion = numeroEntradas - 1;
190             posicion >= 0; posicion--)
191         {
192             //Comprobacion posicion de la entrada
193             //respecto a la de la salida
194             if (entrada[posicion] < salida[posicion + desplazamiento])
195             {
196                 break; //NO overflow
197             }
198             if (entrada[posicion] > salida[posicion + desplazamiento])
199             {
200                 return true; //Overflow
201             }
202         }
203         return false; //NO overflow
204     }
205
206     /*
207         Funcion encargada de comprobar los acarreos
208
209         @param posicion - Marca la posicion donde
210         se quiere comprobar el acarreo
211         @param operacion - Marca la operacion que se
212         estaba haciendo (+ Suma, - Resta)
213     */
214     void comprobarAcarreo(int posicion, char operacion)
215     {
216         //Seleccionamos operacion
```

```
        /*SUMA************************/
218     if (operacion == '+') //Suma
        {
220         //Comprobamos si en la posicion hay que realizar acarreo
            if (salida[posicion] >= baseSistema) //Realizamos acarreo
222         {
                //Comprobamos si se puede producir acarreo automatico
224             if (!posicionBloqueada[posicion])//Puede haber
                {
226                 if (posicion < numeroSalidas - 1)
                    //NO es digito mas significativo
228                 {
                        salida[posicion + 1]++; //+1 digito izquierda
230                     posicionBloqueada[posicion] = true;
                    }
232                 else //Digito mas significativo
                    {
234                     //Encendemos el LED indicativo de Overflow
                        encenderLedOverflow();
236                 }
                }
238             else //NO puede haber acarreo automatico
                {
240                 //Encendemos el LED indicandolo
                    encenderLedAcarreoManual(posicion);
242             }
                //Recalculamos el digito actual
244             salida[posicion] -= baseSistema;
            }
246         //NO realizamos acarreo

248         //Calculamos de todas maneras el valor real de la salida
            if (salidaCorrecta[posicion] >= baseSistema)
250         {
                salidaCorrecta[posicion + 1]++;
252             salidaCorrecta[posicion] -= baseSistema;
            }
254     }
        /*************************************************************/
256
        /*RESTA*************************************************/
258     else if (operacion == '-') //Resta
        {
260         //Comprobamos si en la posicion hay que realizar acarreo
            if (salida[posicion] < 0) //Realizamos acarreo
262         {
                //Comprobamos si se puede producir acarreo automatico
264             if (!posicionBloqueada[posicion]) //Puede haber
                {
266                 salida[posicion + 1]--; //-1 digito izquierda
                    posicionBloqueada[posicion] = true;
268             }
                else //NO se puede realizar acarreo automatico
270             {
                    //Encendemos el LED indicando
272                 encenderLedAcarreoManual(posicion);
                }
274             //Recalculamos el digito actual
                salida[posicion] += baseSistema;
```

```
276             }
                //NO realizamos acarreo
278
                //Calculamos de todas maneras el valor real de la salida
280             if (salidaCorrecta[posicion] < 0)
                {
282                 salidaCorrecta[posicion + 1]--;
                    salidaCorrecta[posicion] += baseSistema;
284             }
            }
286         /*************************************************/
        }
288     /*
            Funcion encargada de realizar la suma sobre
290         el digito de la salida en la posicion marcada
            con el valor
292
            @param posicion - Marca la posicion de la salida
294         donde se realiza la suma
            @param valor - Valor del digito a sumar
296     */
        void restaDigito(int posicion, int valor)
298     {
            //Restamos en la posicion de la salida el valor
300         salida[posicion] -= valor;
            salidaCorrecta[posicion] -= valor;
302
            //Comprobamos si hay acarreo
304         comprobarAcarreo(posicion, '-');
        }
306


308     /*
            Funcion encargada de realizar la resta completa
310         de la entrada sobre el valor actual de la salida
        */
312     void restar()
        {
314         if (hayOverflowRestando())
            {
316             encenderLedOverflow();
            }
318         for (int posicion = 0; posicion < numeroSalidas; posicion++)
            {
320             //Los valores de la salida que no tienen
                //correspondencia con la entrada se les resta 0
322             restaDigito(posicion,
                (posicion - desplazamiento >= numeroEntradas) ?
324             0 : entrada[posicion - desplazamiento]);
            }
326         actualizarContador('-');
            actualizaNumeroDisplaySieteSegmentos(+1);
328         delay(120);
        }
330


332


334
```

```
336     /*
            Funcion encargada de realizar la suma sobre
            el digito de la salida en la posicion marcada
338         con el valor

340         @param posicion - Marca la posicion de la salida
            donde se realiza la suma
342         @param valor - Valor del digito a sumar
        */
344     void sumaDigito(int posicion, int valor)
        {
346         //Sumamos en la posicion de la salida el valor
            salida[posicion] += valor;
348         salidaCorrecta[posicion] += valor;

350         //Comprobamos si hay acarreo
            comprobarAcarreo(posicion, '+');
352     }


354
        /*
356         Funcion encargada de realizar la suma completa
            de la entrada sobre el valor actual de la salida
358     */
        void sumar()
360     {
            for (int posicion = 0; posicion < numeroSalidas; posicion++)
362         {
                //Los valores de la salida que no tienen
364             //correspondencia con la entrada se les suma 0
                sumaDigito(posicion,
366             (posicion - desplazamiento >= numeroEntradas) ?
                0 : entrada[posicion - desplazamiento]);
368         }
            actualizarContador('+');
370         actualizaNumeroDisplaySieteSegmentos(-1);
            delay(120);
372     }


374


        /*
376         Transforma un numero en el codigo necesario para
            visualizarlo en el visualizador 7 segmentos
378     */
        void codificaNumeroDisplaySieteSegmentos(int numero)
380     {
            switch (numero)
382         {
                case 0:
384             {
                    codigo = 126; // 1111110
386                 break;
                }
388             case 1:
                {
390                 codigo = 48; // 0110000
                    break;
392             }
```

```
394             case 2:
                {
396                 codigo = 109; // 1101101
                    break;
398             }
                case 3:
400             {
                    codigo = 121; // 1111001
402                 break;
                }
404             case 4:
                {
406                 codigo = 51; // 0110011
                    break;
408             }
                case 5:
410             {
                    codigo = 91; // 1011011
412                 break;
                }
414             case 6:
                {
416                 codigo = 95; // 1011111
                    break;
418             }
                case 7:
420             {
                    codigo = 112; // 1110000
422                 break;
                }
424             case 8:
                {
426                 codigo = 127; // 1111111
                    break;
428             }
                case 9:
430             {
                    codigo = 123; // 1111011
432                 break;
                }
434         }
            int aux = codigo;
436         //Codificamos
            for (int posicion = 0; posicion < 7; posicion++)
438         {
                codificacionDisplaySieteSegmentos[6 - posicion] = aux %2;
440             aux /= 2;
            }
442     }


444
    /*
446     Muestra el numero en el visualizador de 7 segmentos
        a partir de la codificacion
448 */
    void muestraDisplaySieteSegmentos()
450 {
        //Recorre el array que contiene la codificacion
452     for (int i = 0; i < 7; i++)
```

```
        {
454         digitalWrite(i + primerPinDisplaySieteSegmentos,
            codificacionDisplaySieteSegmentos[i]);
456     }
    }
458
    /*
460     Se encarga de limpiar los registros de salida
        que se muestran en la pantalla junto a los acarreos
462 */
    void limpiarRegistroSalida()
464 {
        //Recorremos los registros de salida
466     for (int posicion = 0; posicion < numeroSalidas; posicion++)
        {
468         //Limpiamos el registro
            salida[posicion] = 0;
470         salidaCorrecta[posicion] = 0;

472         //Limpiamos los bloqueos
            posicionBloqueada[posicion] = false;
474     }
        delay(120);
476 }

478 /*
        Funcion encargada de hacer el acarreo
480     de forma manual si hay algun led de acarreo
        encendido
482 */
    void hacerAcarreoManual()
484 {
        for (int posicion = 0; posicion < numeroSalidas; posicion++)
486     {
            //NOTA: No se puede realizar acarreo manual
488         //si no hay necesidad
            if (alertaAcarreoManual) {
490             //Realizamoe el acarreo manual
                posicionBloqueada[posicion] = false;
492         }
            digitalWrite(posicion + primerLedAcarreo, LOW);
494         salida[posicion] = salidaCorrecta[posicion];
        }
496     alertaAcarreoManual = false;
        delay(120);
498 }

500 /*
        Reinicia el contador encargado de las multiplicaciones
502     y divisiones asi como el visualizador siete segmentos
    */
504 void reiniciarContador()
    {
506     contador = 0;
        contador_aux = 0;
508     digitosContador = 1;
        numeroDisplaySieteSegmentos = 0;
510     codificaNumeroDisplaySieteSegmentos(0);
    }
```

```
512     /*
            Avisa mediante el uso de un LED de que la multiplicacion
514         se ha completado con exito
        */
516     void encenderLedMultiplicacionCompletada()
        {
518         //Duracion total de 1 segundo
            digitalWrite(ledMultiplicacionCompletada, HIGH);
520         delay(400);
            digitalWrite(ledMultiplicacionCompletada, LOW);
522         delay(200);
            digitalWrite(ledMultiplicacionCompletada, HIGH);
524         delay(400);
            digitalWrite(ledMultiplicacionCompletada, LOW);
526     }

528     /*
            Incrementa o decrementa el valor del visualizador siete
        segmentos
530
            @param masOmenos - Incremento o decremento
532     */
        void actualizaNumeroDisplaySieteSegmentos(int masOmenos)
534     {
            numeroDisplaySieteSegmentos += masOmenos;
536         //Si NO estamos entre 0 y la base
            if (numeroDisplaySieteSegmentos <= 0 ||
538         numeroDisplaySieteSegmentos >= baseSistema)
            {
540             //Si el valor es 0
                if (numeroDisplaySieteSegmentos == 0)
542             {
                    encenderLedMultiplicacionCompletada();
544             }
                //Al salir del rango lo colocamos a 0
546             numeroDisplaySieteSegmentos = 0;
            }
548         codificaNumeroDisplaySieteSegmentos
            (numeroDisplaySieteSegmentos);
550         delay(120);
        }
552
        /*
554         Incrementa correctamente el valor del contador
            ademas del valor que semuestra en el visualizador
556
            @param masOmenos - Indica incremento o decremento
558         (+ Incremento, - Decremento)
        */
560     void actualizarContador(char masOmenos)
        {
562         //Calculamos la cantidad que hay que sumar o restar
            //dependiendo del desplazamiento
564         long cantidad = 1;
            for (int i = 1; i <= desplazamiento; i++)
566         {
                //Necesario para la posterior transformacion
568             cantidad *= baseSistema;
            }
```

```
570
        /*Decicimos si sumamos o restamos************/
572     //Incrementamos
        if (masOmenos == '+')
574     {
            contador_aux += cantidad;
576     }
        //Decrementamos
578     else if (masOmenos == '-')
        {
580         contador_aux -= cantidad;
        }
582     /*************************************/
        String nuevoContador =
584     cambioBase(abs(contador_aux), baseSistema);

586     contador = nuevoContador.toInt();
        //Si el valor era negativo
588     if (contador_aux < 0)
        {
590         //Lo hacemos negativo
            contador *= -1;
592     }
        digitosContador = nuevoContador.length();
594 }


596
    /*
598     Se encarga de aumentar o decrementar el desplazamiento
        dependiendo del parametro de entrada
600
        @param direccion - Indica la direccion del
602     desplazamiento (+ Derecha, - Izquierda)
    */
604 void desplazar(char direccion)
    {
606     //Derecha
        if (direccion == '+' && desplazamiento > 0)
608     {
            desplazamiento--;
610     }
        //Izquierda
612     else if (direccion == '-' && desplazamiento < numeroEntradas)
        {
614         desplazamiento++;
        }
616
        //Reiniciar el valor del visualizador
618     numeroDisplaySieteSegmentos = 0;
        codificaNumeroDisplaySieteSegmentos(0);
620     delay(120);
    }
622


624
    /*
626     Lee las entradas de los potenciometros
        y calcula el numero resultante de dichas entradas
628     mediante la transformacion de analogico a digital
```

```
630     */
        void leerValoresEntrada()
        {
632         //Calculamos para cada posicion de la entrada su valor digital
            for (int posicionEntrada = 0;
634         posicionEntrada < numeroEntradas;
            posicionEntrada++)
636         {
                entrada[numeroEntradas - posicionEntrada - 1] =
638             map(analogRead(pinEntrada[posicionEntrada]),
                0, 1024, 0, baseSistema);
640         }
        }
642


644     /*
            Imprime los valores del array entrada en la fila
646         de entrada del visualizador dandole un formato adecuado
        */
648
        void imprimirEntrada()
650     {
            //Colocacion del cursor
652         lcd.setCursor(0, filaEntrada);

654         //Relleno a la izquierda
            for (int relleno = numeroEntradas - desplazamiento;
656         relleno > 0; relleno--)
            {
658             lcd.print(" ");
            }
660
            //Embellecedor y clarificador de linea
662         lcd.print("E = ");

664         //Impresion de izquierda a derecha
            for (int numeroEntrada = numeroEntradas - 1;
666         numeroEntrada >= 0; numeroEntrada--)
            {
668             lcd.print(entrada[numeroEntrada]);
            }
670
            //Relleno a la derecha
672         for (int relleno = desplazamiento; relleno > 0; relleno--)
            {
674             lcd.print(" ");
            }
676     }


678
        /*
680         Imprime los valores del array salida en la fila de
            salida del visualizador dandole un formato adecuado
682     */
        void imprimirSalida()
684     {
            //Colocacion del cursor
686         lcd.setCursor(0, filaSalida);
```

```
688
        //Embellecedor y clarificador de linea
690     lcd.print("S = ");

692     //Impresion de izquierda a derecha
        for (int numeroSalida = numeroSalidas - 1;
694     numeroSalida >= 0; numeroSalida--)
        {
696         lcd.print(salida[numeroSalida]);
        }
698 }

700 /*
        Imprime el valor del contador en la linea correspondiente
702     ademas de darle un formato correcto
    */
704 void imprimirContador()
    {
706     lcd.setCursor(numeroEntradas - 1, 2);
        if (contador > 0)
708     {
            lcd.print("+");
710     }
        else if (contador < 0)
712     {
            lcd.print("-");
714     }
        else
716     {
            lcd.print(" ");
718     }

720     lcd.print("C =");

722     for (int i = 0; i < numeroEntradas - digitosContador + 1; i++)
        {
724         lcd.print("_");
        }
726     lcd.print(abs(contador));
    }
728


730 /*
        Se encarga de imprimir en el visualizador LCD una flecha
732     indicando la posicion donde se efectuan las operaciones
    */
734 void imprimirIndicadorPosicion()
    {
736     lcd.setCursor(numeroEntradas, 3);
        //Colocacion de blancos izquierda
738     for (int i = 0; i <= numeroEntradas - desplazamiento + 2; i++ )
        {
740         lcd.print(" ");
        }
742
        lcd.print("^");
744
        //Colocacion de blancos derecha
746     for (int i = numeroEntradas * 2 - desplazamiento + 2;
```

```
        i < numeroSalidas + 2; i++)
748     {
            lcd.print(" ");
750     }
    }
752
754 /*
        Funcion encargada de leer cada boton y
756     determinar si ha sido pulsado o no, haciendo
        la llamada perteneciente al metodo que
758     corresponde con cada boton
    */
760 void botonesPulsados()
    {
762     /*Obtenemos los valores de los botones (pulsado o no-pulsado)*/
        pulsadoBotonSuma = digitalRead(botonSuma);
764     pulsadoBotonResta = digitalRead(botonResta);
        pulsadoBotonReset = digitalRead(botonReset);
766     pulsadoBotonDesplazamientoDerecha =
        digitalRead(botonDesplazamientoDerecha);
768     pulsadoBotonDesplazamientoIzquierda =
        digitalRead(botonDesplazamientoIzquierda);
770     pulsadoBotonAcarreoManual = digitalRead(botonAcarreoManual);
        pulsadoBotonDisplaySieteSegmentos =
772     digitalRead(botonDisplaySieteSegmentos);
        pulsadoBotonResetContador =
774     digitalRead(botonResetContador);

776     /*****************************************/
        //Cambio de base
778     if (pulsadoBotonReset && pulsadoBotonResetContador)
        {
780         menuBase();
            reiniciarContador();
782         limpiarRegistroSalida();
        }
784
        /*Solo funcionan si no hay acarreo manual**************/
786     if (!alertaAcarreoManual)
        {
788         //Comprobamos si se ha pulsado algun boton
            if (pulsadoBotonSuma)
790         {
                sumar();
792         }
            if (pulsadoBotonResta)
794         {
                restar();
796         }
            if (pulsadoBotonReset)
798         {
                limpiarRegistroSalida();
800         }
        }
802     /*****************************************/
        //Comprobamos si se ha pulsado algun boton
804     if (pulsadoBotonDisplaySieteSegmentos)
        {
```

```
806             actualizaNumeroDisplaySieteSegmentos(+1);
        }
808     if (pulsadoBotonResetContador)
        {
810         reiniciarContador();
        }
812     if (pulsadoBotonAcarreoManual)
        {
814         hacerAcarreoManual();
        }
816     if (pulsadoBotonDesplazamientoIzquierda)
        {
818         //Desplazar izquierda
            desplazar('-');
820     }
        if (pulsadoBotonDesplazamientoDerecha)
822     {
            //Desplazar derecha
824         desplazar('+');
        }
826 }


828
    /*
830     Esta funcion se encarga de imprimir en el visualizador LCD
        el nombre del trabajo y autor
832
        Se ejecuta al principio del programa
834 */
    void animacionInicio()
836 {
        lcd.clear();
838     lcd.setCursor(4, 0);
        lcd.print("CALCULADORA");
840     lcd.setCursor(6, 1);
        lcd.print("LEIBNIZ");
842     lcd.setCursor(2, 3);
        lcd.print("VICTOR ZARAGOZA");
844     delay(2500);
        lcd.clear();
846 }


848
    /*
850     Punto de entrada del programa

852     Arduino ejecuta esta funcion 1 vez al arrancar el codigo
        y contiene las incializaciones de los distintos pines
854 */
    void setup()
856 {
        //Inicializacion del LED de alerta overflow
858     pinMode(ledAlertaOverflow, OUTPUT);

860     /*Inicializacion de los botones*****************/
        pinMode(botonSuma, INPUT);
862     pinMode(botonResta, INPUT);
        pinMode(botonReset, INPUT);
864     pinMode(botonDesplazamientoIzquierda, INPUT);
```

```
866         pinMode(botonDesplazamientoDerecha, INPUT);
            pinMode(botonDisplaySieteSegmentos, INPUT);
            pinMode(botonAcarreoManual, INPUT);
868         pinMode(botonResetContador, INPUT);
            /*********************************************/
870
            /*Inicializacion LED de alerta acarreo**********/
872         for (int ledAlertaAcarreo = primerLedAcarreo;
            ledAlertaAcarreo < ultimoLedAcarreo; ledAlertaAcarreo++)
874         {
                pinMode(ledAlertaAcarreo, OUTPUT);
876         }
            /*********************************************/
878
            /*Inicializacion pines de entrada de los potenciometros*******/
880         for (int entrada = 0;
            entrada < numeroEntradas; entrada++)
882         {
                pinMode(pinEntrada[entrada], INPUT);
884         }
            /*******************************************
886
            /*Inicializacion visualizador 7 segmentos*********/
888         for (int pinDisplaySieteSegmentos =
            primerPinDisplaySieteSegmentos;
890         pinDisplaySieteSegmentos < ultimoPinDisplaySieteSegmentos;
            pinDisplaySieteSegmentos++ )
892         {
                pinMode(pinDisplaySieteSegmentos, OUTPUT);
894         }
            /*****************************************/
896
            /*Inicializacion del visualizador***********/
898         //Inicializacion del fondo retroiluminado
            lcd.backlight();
900         //Iniciamos la pantalla
            lcd.init();
902         /************************************/

904         //Animacion de arranque
            animacionInicio();
906     }


908
        /*
910         Bucle principal del programa

912         Arduino ejecuta este bucle de forma infinita
            y se encarga de contener el codigo a ejecutar
914     */
        void loop()
916     {

918         //Leemos los distintos valores de entrada
            leerValoresEntrada();
920
            //Imprimimos los valores de entrada calculados anteriormente
922         //en el visualizador
            imprimirEntrada();
```

```
924
        //Imprimimos los valores del registro de salida
926     imprimirSalida();

928     //Mostramos el valor adecuado en el
        //visualizador siete segmentos
930     muestraDisplaySieteSegmentos();

932     //Imprimimos la linea del contador
        imprimirContador();
934
        //Imprimimos el indicador de la posicion donde
936     //aplicamos las operaciones
        imprimirIndicadorPosicion();
938
        //Miramos que botones han sido pulsados
940     botonesPulsados();

942     delay(20);
    }
```