2025

Calculadora Multibase

PRACTICA II

MATEO CÁRDENAS PEDRAZA-HAROL PEREZ TRUJILLO
20231578086-20222578053

Contenido

Calculadora Multibase

Este proyecto pretende a evaluar la manipulación de las operaciones aritméticas en diferentes bases, con la comprobación del uso de circuito lógico

Introducción

El Proyecto N°2 de Calculadora Multibase tiene como propósito diseñar y verificar, tanto en código como en circuito lógico, la capacidad de realizar operaciones aritméticas básicas (suma y resta) en distintas bases numéricas: binaria, octal, decimal y bases extendidas (hasta 5000 símbolos). Se implementa en Python la lógica de conversión y cálculo, y se construye un circuito en Logisim para evidenciar la corrección de los resultados.

Objetivo

Desarrollar y validar una calculadora capaz de sumar y restar números en múltiples bases (2, 8, 10 y extendidas), con demostración tanto en software (Python) como en hardware lógico.

Marco Teórico

El estudio de los cálculos multibase se fundamenta en la extensión de los sistemas de numeración posicional, donde la base o radix determina el conjunto de símbolos y el peso de cada posición. Tradicionalmente, en informática y electrónica digital se emplean las bases 2 (binaria), 8 (octal) y 16 (hexadecimal), debido a su relación directa con el uso de bits y agrupaciones de tres o cuatro bits respectivamente. Sin embargo, en aplicaciones avanzadas o educativas resulta útil considerar bases arbitrarias —incluso superiores a 62— para explorar el comportamiento de los algoritmos de conversión y operaciones aritméticas en contextos de amplio rango de símbolos.

La conversión de un número de base n a decimal implica la evaluación de un polinomio, donde cada dígito d_i aporta $d_i \cdot n^i$ al valor total. En sentido inverso, para convertir un valor decimal a base n, se utiliza la descomposición por divisiones sucesivas, almacenando los restos y reordenándolos. Estos dos procesos constituyen los algoritmos fundamentales, cuyo análisis de complejidad revela un coste proporcional al número de dígitos y al uso de operaciones de potencia y módulo en el caso de bases grandes.

En el ámbito de la aritmética multibase, las operaciones de suma y resta se extienden a cualquier base manteniendo la lógica de acarreo (carry) y préstamo (borrow). Para la suma en base n, un full adder en un circuito digital realiza la suma bit a bit de dos operandos y un acarreo de entrada, generando una suma y un acarreo de salida, con ecuaciones de la forma $S = A \oplus B \oplus C_{in}$ y $C_{out} = (A \cdot B) + (C_{in} \cdot (A \oplus B))$. En la resta, puede emplearse la técnica de complemento a la base (equivalente al complemento a dos en binario) o implementar una red de restadores que manejen explícitamente el borrow.

El diseño de circuitos lógicos para operaciones multibase comienza con la representación de cada dígito en binario, utilizando vectores de ancho adecuado según la base (por ejemplo, 4 bits para hexadecimal). La construcción de sumadores de k bits se realiza mediante la cascada de k full adders (ripple-carry), lo cual conlleva un retardo acumulativo lineal al número de bits. Para optimizar el tiempo de propagación, se investigan arquitecturas de carry-lookahead o técnicas de carry-select, que permiten generar grupos de bits con un cálculo anticipado del acarreo, reduciendo el tiempo de respuesta a O(log k).

Otro concepto relevante en el diseño lógico es la minimización de expresiones booleanas, donde mediante mapas de Karnaugh o álgebra de Boole se simplifican las funciones de control de acarreo y suma, reduciendo significativamente el número de compuertas y la complejidad física del circuito. Este paso resulta crítico cuando se implementan en hardware real (ASIC, FPGA o microcontroladores) para optimizar área, consumo de energía y coste.

Adicionalmente, en bases extendidas (superiores a 62) la implementación en hardware puro se complica, por lo que se recurre a representaciones híbridas: por un lado, se codifican símbolos complejos en múltiples líneas de datos; por otro, se externaliza parte del procesamiento a software embebido o microcontroladores, donde la flexibilidad de memoria y lógica programable facilita el manejo de alfabetos muy grandes.

Finalmente, el estudio de la propagación de señales, el fan–in/fan–out y la integridad de la señal en buses de datos paralelos resulta esencial para garantizar la fiabilidad del circuito. La sincronización con señal de reloj, el uso de buffers y retardo controlado (skew) asegura que las operaciones multibit lleguen a un estado estable antes de ser leídas o alimentadas a etapas posteriores, permitiendo diseñar sistemas multibase robustos y escalables.

Sistemas de Numeración Posicional

- Base 2 (Binaria): dígitos {0,1}.
- Base 8 (Octal): dígitos {0,…,7}.
- Base 10 (Decimal): dígitos {0,…,9}.
- Bases extendidas: Uso de un alfabeto de hasta 5000 símbolos, generados mediante combinaciones de dígitos, letras mayúsculas/minúsculas y pares de símbolos para cubrir rangos más amplios.

Cada posición en un número en base b aporta un peso de b^k según su índice k.

Conversión Entre Bases

- De base n a decimal: suma ponderada de cada dígito por n^k.
- De decimal a base n: división repetida por n, acumulando restos.
- Bases > 62: se representan símbolos complejos separados por guiones.

Circuitos Combinacionales de Aritmética

- Half Adder y Full Adder: bloques básicos para sumas binarias con generación de acarreo.
- Resta por complemento o por préstamo: lógica que ajusta bits con “borrow” si el minuendo es menor.

Diseño de la Solución

El diseño de la solución se estructura en dos arquitecturas complementarias. En software, se construyen seis componentes principales: uno para generar el alfabeto de símbolos hasta 5000 elementos; otro para mapear símbolos a valores numéricos y viceversa; un módulo de conversión de base *n* a decimal; otro de conversión de decimal a base *n*; una rutina que muestra paso a paso la suma y resta de dígitos indicando acarreos o préstamos; y finalmente una interfaz de línea de comandos que gestiona el flujo de operandos y operaciones, acumulando resultados y mostrando salidas en todas las bases utilizadas. En hardware, el circuito en Logisim emplea cadenas de full adders para sumar los vectores de bits de dos pares de operandos (A+B y C+D), un bloque restador que sustrae un tercer operando de la suma inicial y multiplexores que dirigen las salidas a indicadores de resultado positivo o negativo. Los buses de datos, manejados con splitters, permiten el tratamiento paralelo de varios bits para garantizar la velocidad y la visibilidad de cada línea de bits.

La separación en módulos tanto en software como en hardware facilita el desarrollo incremental, la prueba aislada de cada componente y la integración gradual de la solución completa, asegurando una mayor mantenibilidad y capacidad de expansión.

Implementación

En software, la aplicación se desarrolló con Python 3.8+ sin dependencias externas, concentrando todo el código en un único archivo lab2_multibase.py. La lógica de generación del alfabeto, mapeo, conversión y gestión de operaciones se prueba de manera manual mediante la línea de comandos, comparando los resultados con los obtenidos en el circuito lógico. En hardware, se empleó Logisim 2.7.1, utilizando componentes básicos como full adders, inversores y compuertas AND/OR para el restador, así como splitters y cabinas para la manipulación de buses de 8 a 16 bits. El procedimiento consistió en diseñar dos agrupaciones de full adders para la suma de operandos, encadenar el acarreo final a un bloque restador, y finalmente conectar las salidas a una serie de indicadores binarios que representan el resultado en cada bit

Pruebas / Resultados

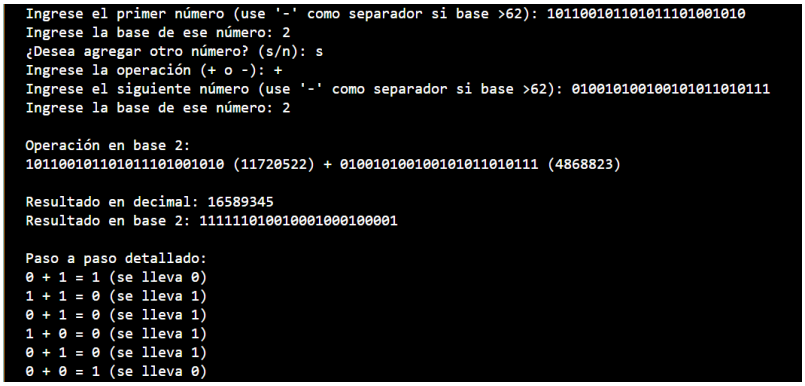
Binario

A: 10110010 11010111 01001010

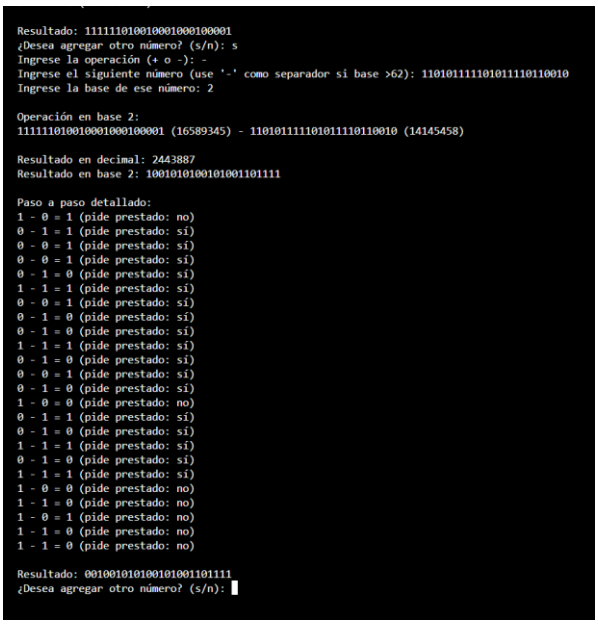
B: 01001010 01001010 11010111

C: 11010111 11010111 10110010

D: 11010111 10110010 11011001



A+B: 111111010010001000100001–programa



(A+B)-C: 0010010101001010011011111 –programa

```
PS C:\Users\MATEO CARVAJAL> & "C:/Users/MATEO CARVAJAL/AppData/Local/Microsoft/WindowsApps/pyth
=== Calculadora Multibase ===
Ingrese el primer número (use '-' como separador si base >62): 1101011111010111110110010
Ingrese la base de ese número: 2
¿Desea agregar otro número? (s/n): s
Ingrese la operación (+ o -): +
Ingrese el siguiente número (use '-' como separador si base >62): 1101011110110010110110001
Ingrese la base de ese número: 2

Operación en base 2:
1101011111010111110110010 (14145458) + 1101011110110010110110001 (14136025)

Resultado en decimal: 28281483
Resultado en base 2: 1101011111000010101010001011

Paso a paso detallado:
0 + 1 = 1 (se lleva 0)
1 + 0 = 1 (se lleva 0)
0 + 0 = 0 (se lleva 0)
0 + 1 = 1 (se lleva 0)
```

C+D: 1101011111000010101010001011 – programa

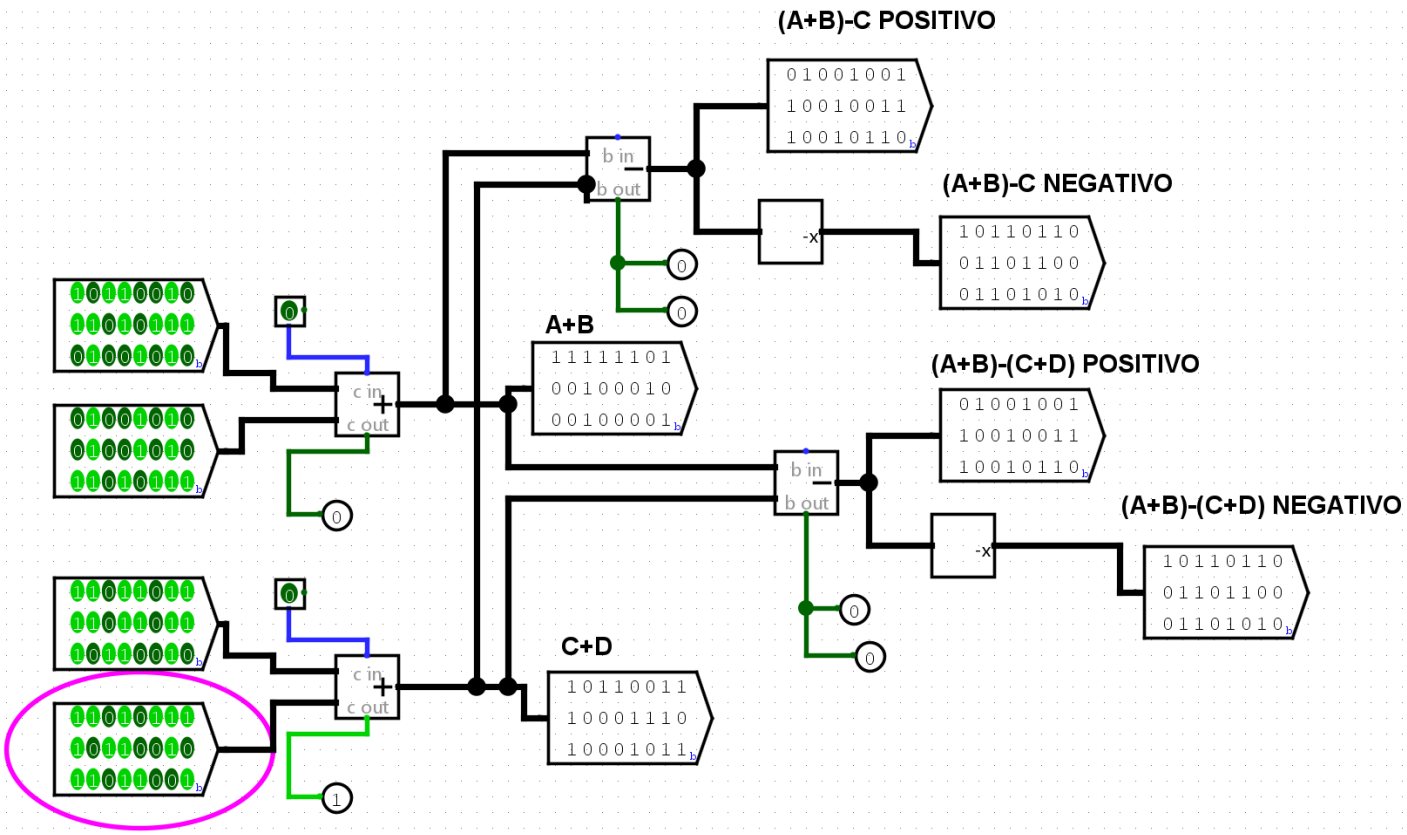
```
PS C:\Users\MATEO CARVAJAL> & "C:/Users/MATEO CARVAJAL/AppData/Local/Microsoft/WindowsApps/python3
=== Calculadora Multibase ===
Ingrese el primer número (use '-' como separador si base >62): 1111110100100010001000100001
Ingrese la base de ese número: 2
¿Desea agregar otro número? (s/n): s
Ingrese la operación (+ o -): -
Ingrese el siguiente número (use '-' como separador si base >62): 1101011111000010101010001011
Ingrese la base de ese número: 2

Operación en base 2:
1111110100100010001000100001 (16589345) - 1101011111000010101010001011 (28281483)

Resultado en decimal: -11692138
Resultado en base 2: -101100100110100001101010

Paso a paso detallado:
El resultado será negativo; se intercambian los valores para mostrar la resta.
1 - 1 = 0 (pide prestado: no)
1 - 0 = 1 (pide prestado: no)
0 - 0 = 0 (pide prestado: no)
1 - 0 = 1 (pide prestado: no)
0 - 0 = 0 (pide prestado: no)
0 - 1 = 1 (pide prestado: sí)
```

(A+B)-(C+D): -101100100110100001101010–programa


(A+B)-C POSITIVO
01001001
10010011
10010110
b in
b out
(A+B)-C NEGATIVO
-x
10110110
01101100
01101010
A+B
11111101
00100010
00100001
c in
c out
(A+B)-(C+D) POSITIVO
(A+B)-(C+D) NEGATIVO
C+D
10110011
10001110
10001011

Octal

A: 7432107265421325

B: 4371230521676542

C: 7123143246512367

D: 6214276542107643

```
=== Calculadora Multibase ===
Ingrese el primer número (use '-' como separador si base >62): 7432107265421325
Ingrese la base de ese número: 8
¿Desea agregar otro número? (s/n): s
Ingrese la operación (+ o -): +
Ingrese el siguiente número (use '-' como separador si base >62): 4371230521676542
Ingrese la base de ese número: 8

Operación en base 8:
7432107265421325 (265679074108117) + 4371230521676542 (157869127728482)

Resultado en decimal: 423548201836599
Resultado en base 8: 14023340007320067

Paso a paso detallado:
5 + 2 = 7 (se lleva 0)
2 + 4 = 6 (se lleva 0)
```

A+B: 14023340007320067 –Programa

```
Resultado: 14023340007320067
¿Desea agregar otro número? (s/n): s
Ingrese la operación (+ o -): -
Ingrese el siguiente número (use '-' como separador si base >62): 7123143246512367
Ingrese la base de ese número: 8

Operación en base 8:
14023340007320067 (423548201836599) - 7123143246512367 (252007652431095)

Resultado en decimal: 171540549405504
Resultado en base 8: 4700174540605500

Paso a paso detallado:
7 - 7 = 0 (pide prestado: no)
6 - 6 = 0 (pide prestado: no)
0 - 3 = 5 (pide prestado: sí)
0 - 2 = 5 (pide prestado: sí)
2 - 1 = 0 (pide prestado: no)
3 - 5 = 6 (pide prestado: sí)
7 - 6 = 0 (pide prestado: no)
0 - 4 = 4 (pide prestado: sí)
0 - 2 = 5 (pide prestado: sí)
0 - 3 = 4 (pide prestado: sí)
4 - 4 = 7 (pide prestado: sí)
3 - 1 = 1 (pide prestado: no)
3 - 3 = 0 (pide prestado: no)
2 - 2 = 0 (pide prestado: no)
0 - 1 = 7 (pide prestado: sí)
4 - 7 = 4 (pide prestado: sí)
1 - 0 = 0 (pide prestado: no)

Resultado: 04700174540605500
```

(A+B)-C: 04700174540605500 –Programa

```
=== Calculadora Multibase ===
Ingrese el primer número (use '-' como separador si base >62): 7123143246512367
Ingrese la base de ese número: 8
¿Desea agregar otro número? (s/n): s
Ingrese la operación (+ o -): +
Ingrese el siguiente número (use '-' como separador si base >62): 6214276542107643
Ingrese la base de ese número: 8

Operación en base 8:
7123143246512367 (252007652431095) + 6214276542107643 (220752553480099)

Resultado en decimal: 472760205911194
Resultado en base 8: 15337442010622232

Paso a paso detallado:
7 + 3 = 2 (se lleva 1)
6 + 4 = 3 (se lleva 1)
```

C+D: 15337442010622232 –Programa

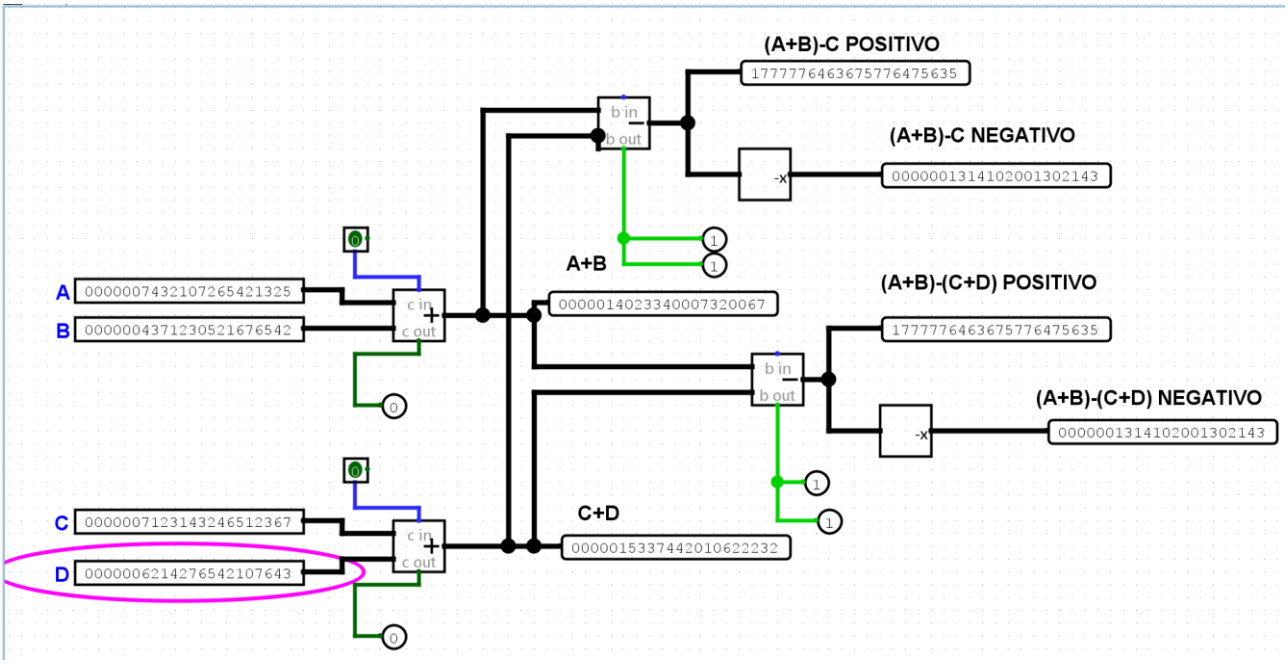
```
=== Calculadora Multibase ===
Ingrese el primer número (use '-' como separador si base >62): 14023340007320067
Ingrese la base de ese número: 8
¿Desea agregar otro número? (s/n): s
Ingrese la operación (+ o -): -
Ingrese el siguiente número (use '-' como separador si base >62): 15337442010622232
Ingrese la base de ese número: 8

Operación en base 8:
14023340007320067 (423548201836599) - 15337442010622232 (472760205911194)

Resultado en decimal: -49212004074595
Resultado en base 8: -1314102001302143

Paso a paso detallado:
El resultado será negativo; se intercambian los valores para mostrar la resta.
2 - 7 = 3 (pide prestado: sí)
3 - 6 = 4 (pide prestado: sí)
2 - 0 = 1 (pide prestado: no)
2 - 0 = 2 (pide prestado: no)
```

(A+B)-(C+D):-01314102001302143 –Programa



Hexadecimal

A: FA5BCDE4312FAB72

B: 5C7ABA427E95ECD6

C: FB6C79A437D5FBA5

D: F472159E68B39E5A

```
=== Calculadora Multibase ===
Ingrese el primer número (use '-' como separador si base >62): FA5BCDE4312FAB72
Ingrese la base de ese número: 16
¿Desea agregar otro número? (s/n): s
Ingrese la operación (+ o -): +
Ingrese el siguiente número (use '-' como separador si base >62): 5C7ABA427E95ECD6
Ingrese la base de ese número: 16

Operación en base 16:
FA5BCDE4312FAB72 (18040239112324098930) + 5C7ABA427E95ECD6 (6663843393402432726)

Resultado en decimal: 24704082505726531656
Resultado en base 16: 156D68826AFC59848

Paso a paso detallado:
2 + 6 = 8 (se lleva 0)
7 + D = 4 (se lleva 1)
B + C = 8 (se lleva 1)
```

A+B:156D68826AFC59848

```
Resultado: 156D68826AFC59848
¿Desea agregar otro número? (s/n): s
Ingrese la operación (+ o -): -
Ingrese el siguiente número (use '-' como separador si base >62): FB6C79A437D5FBA5
Ingrese la base de ese número: 16

Operación en base 16:
156D68826AFC59848 (24704082505726531656) - FB6C79A437D5FBA5 (18116989147223030693)

Resultado en decimal: 6587093358503500963
Resultado en base 16: 5B6A0E8277EF9CA3

Paso a paso detallado:
8 - 5 = 3 (pide prestado: no)
4 - A = A (pide prestado: sí)
8 - B = C (pide prestado: sí)
9 - F = 9 (pide prestado: sí)
```

(A+B)-C:05B6A0E8277EF9CA3

```
=== Calculadora Multibase ===
Ingrese el primer número (use '-' como separador si base >62): FB6C79A437D5FBA5
Ingrese la base de ese número: 16
¿Desea agregar otro número? (s/n): s
Ingrese la operación (+ o -): +
Ingrese el siguiente número (use '-' como separador si base >62): F472159E68B39E5A
Ingrese la base de ese número: 16

Operación en base 16:
FB6C79A437D5FBA5 (18116989147223030693) + F472159E68B39E5A (17614164862705049178)

Resultado en decimal: 35731154009928079871
Resultado en base 16: 1EFDE8F42A08999FF

Paso a paso detallado:
5 + A = F (se lleva 0)
A + 5 = F (se lleva 0)
B + E = 9 (se lleva 1)
F + 9 = 9 (se lleva 1)
```

C+D: 1EFDE8F42A08999FF

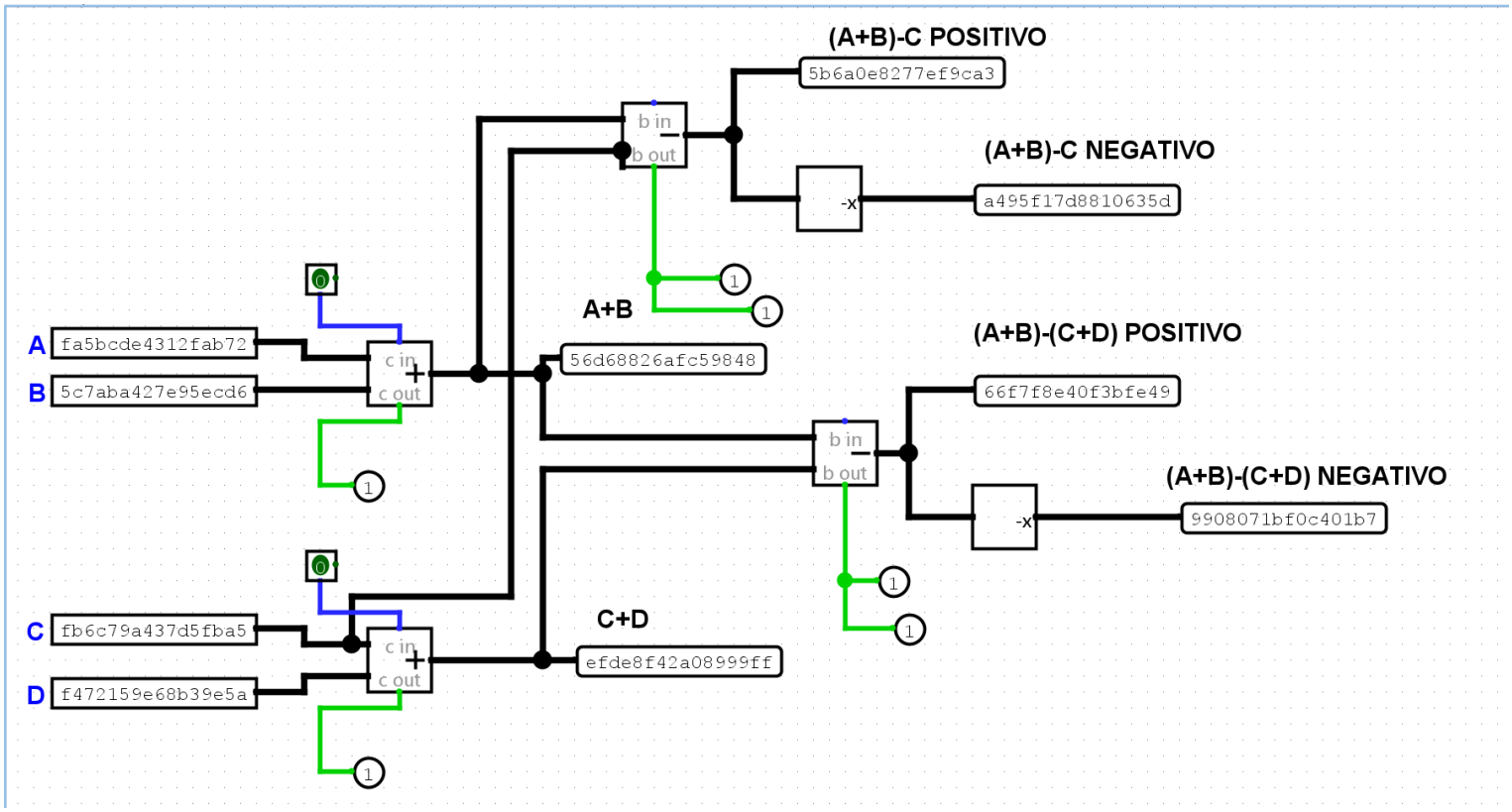
```
=== Calculadora Multibase ===
Ingrese el primer número (use '-' como separador si base >62): 156D68826AFC59848
Ingrese la base de ese número: 16
¿Desea agregar otro número? (s/n): s
Ingrese la operación (+ o -): -
Ingrese el siguiente número (use '-' como separador si base >62): 1EFDE8F42A08999FF
Ingrese la base de ese número: 16

Operación en base 16:
156D68826AFC59848 (2470408250572653165 6) - 1EFDE8F42A08999FF (3573115400992807987 1)

Resultado en decimal: -11027071504201548215
Resultado en base 16: -9908071BF0C401B7

Paso a paso detallado:
El resultado será negativo; se intercambian los valores para mostrar la resta.
F - 8 = 7 (pide prestado: no)
F - 4 = B (pide prestado: no)
9 - 8 = 1 (pide prestado: no)
9 - 9 = 0 (pide prestado: no)
```

(A+B)-(C+D):-09908071BF0C401B5

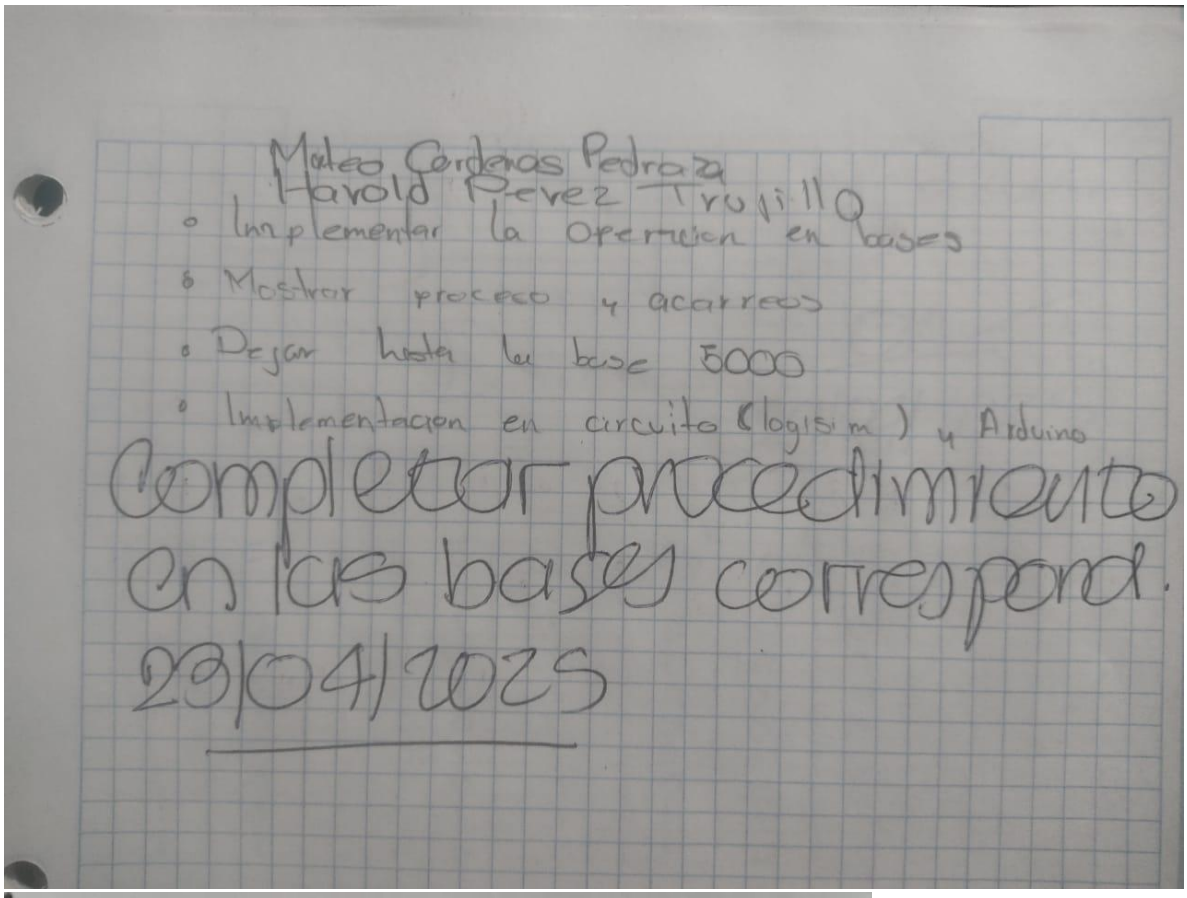


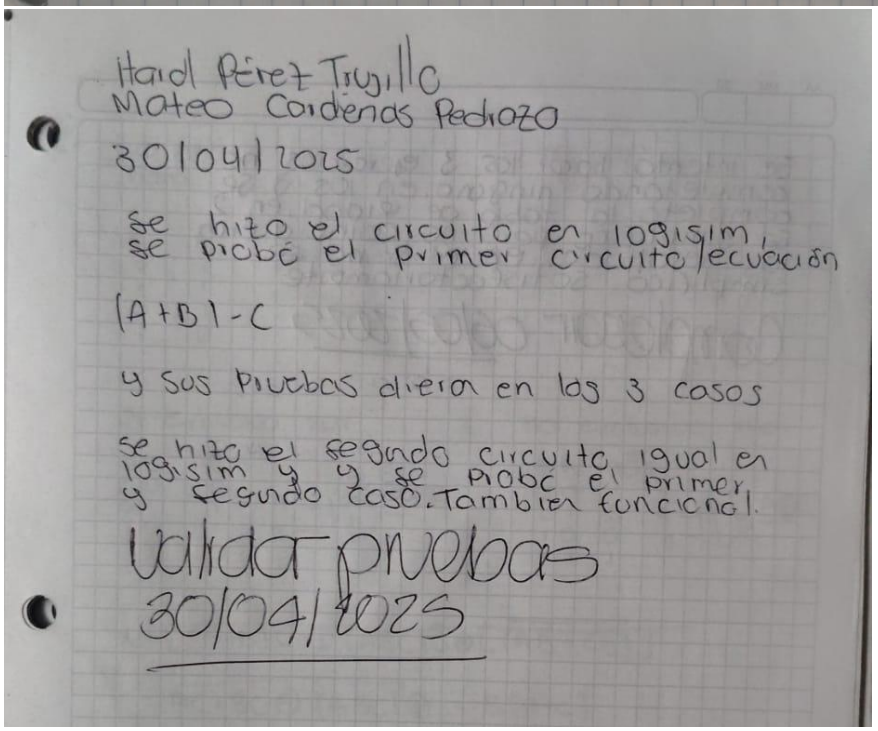
Hipotética implementación en Arduino

La migración de la calculadora multibase a un Arduino Uno (ATmega328P) es factible siempre que se atiendan las limitaciones de memoria y procesamiento. El sistema requeriría un teclado matricial 4×4 para la entrada de dígitos y comandos y una pantalla LCD 16×2 (o con interfaz I²C) para la visualización de operaciones y resultados. En firmware se usarían las bibliotecas Keypad.h para la captura de teclas y LiquidCrystal.h (o LiquidCrystal_I2C.h) para el control del display. El diseño del programa incluiría un buffer de entrada que almacene los símbolos hasta la confirmación, funciones en C++ que repliquen la conversión de base y las operaciones aritméticas, y rutinas que muestren el proceso dígito a dígito en la LCD. Deberán optimizarse la gestión de símbolos extendidos y el cálculo de potencias para adaptarse a la memoria y velocidad de la placa

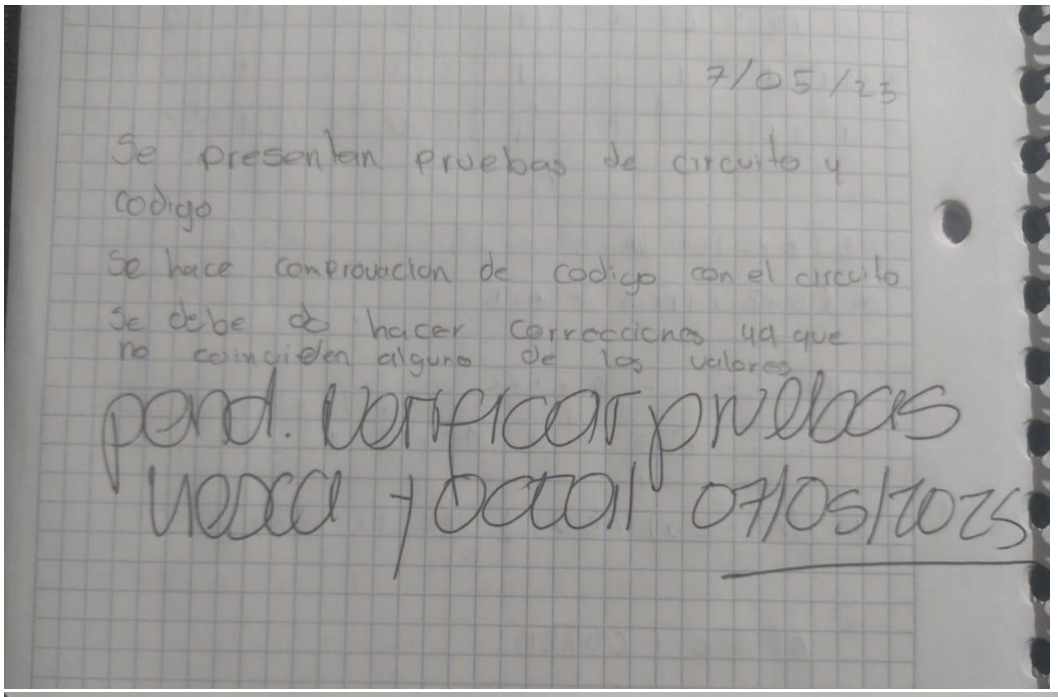
Evidencia de entrega y trabajos en clase

Entrega en laboratorio

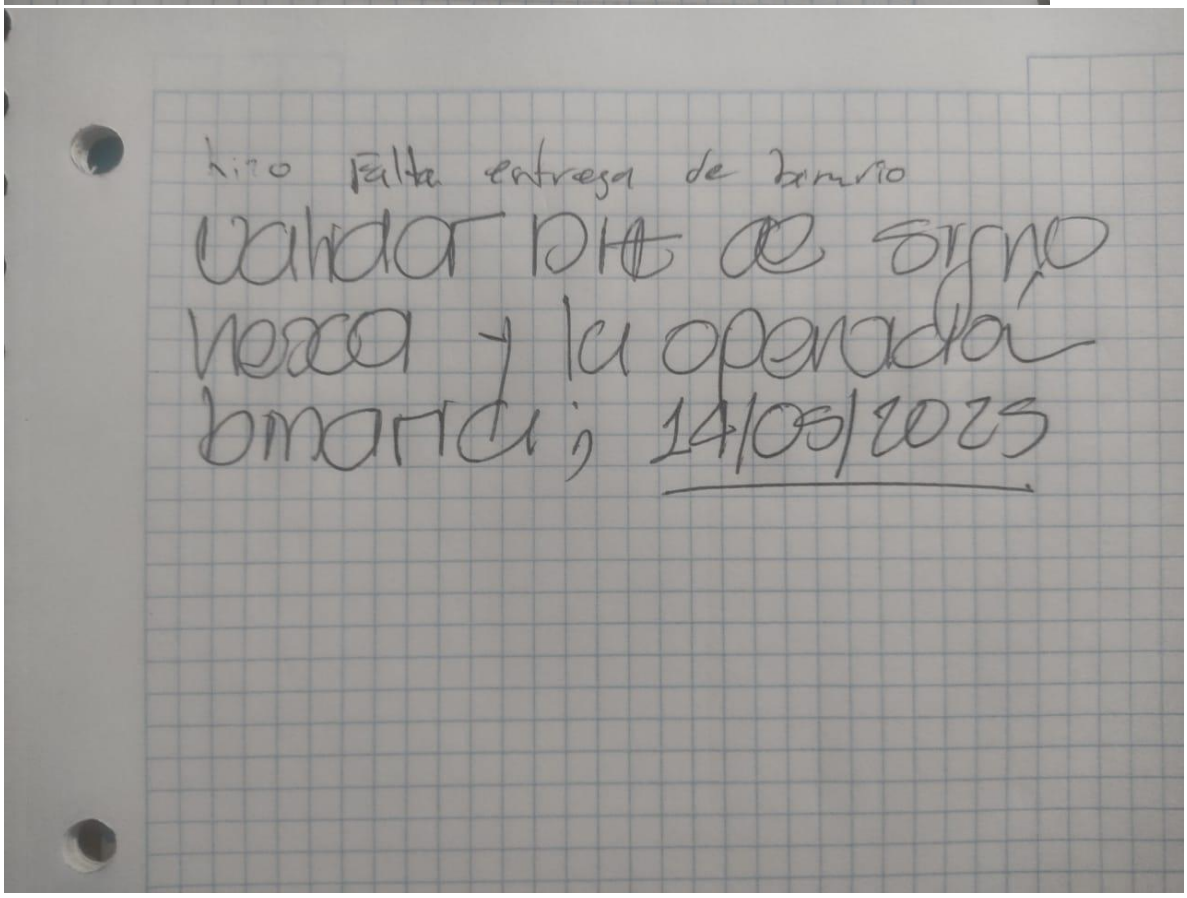
1. 

2. 

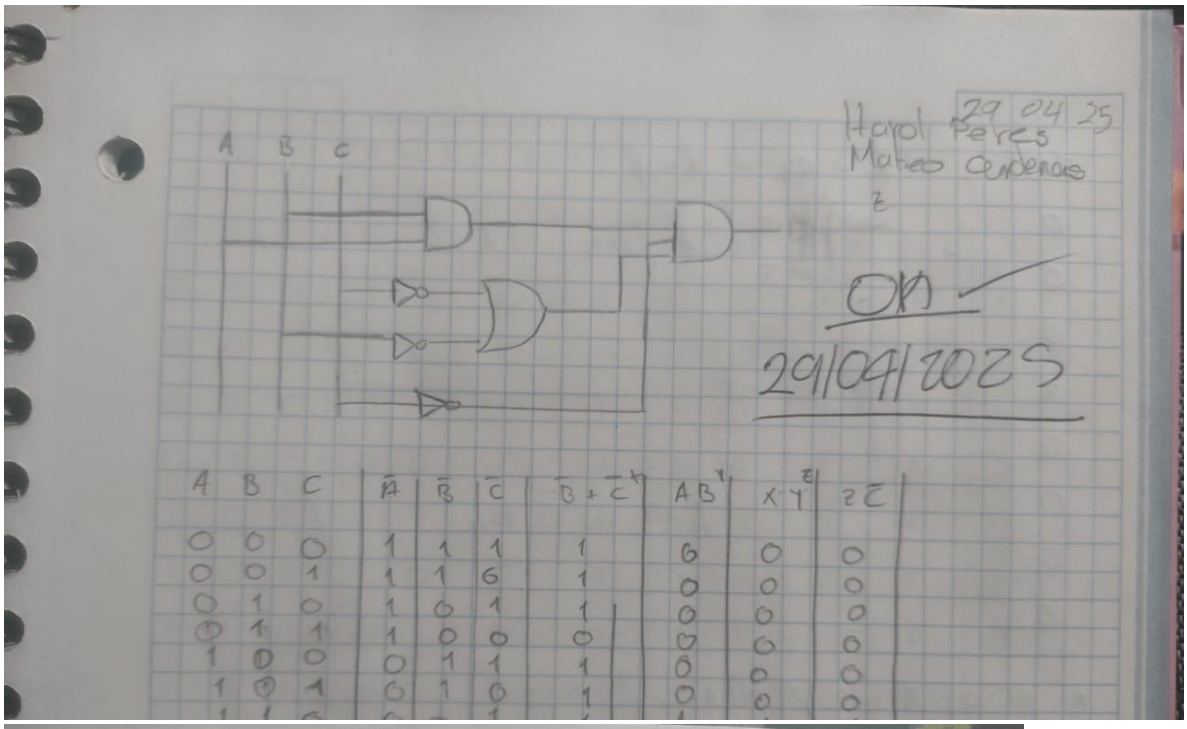
3.

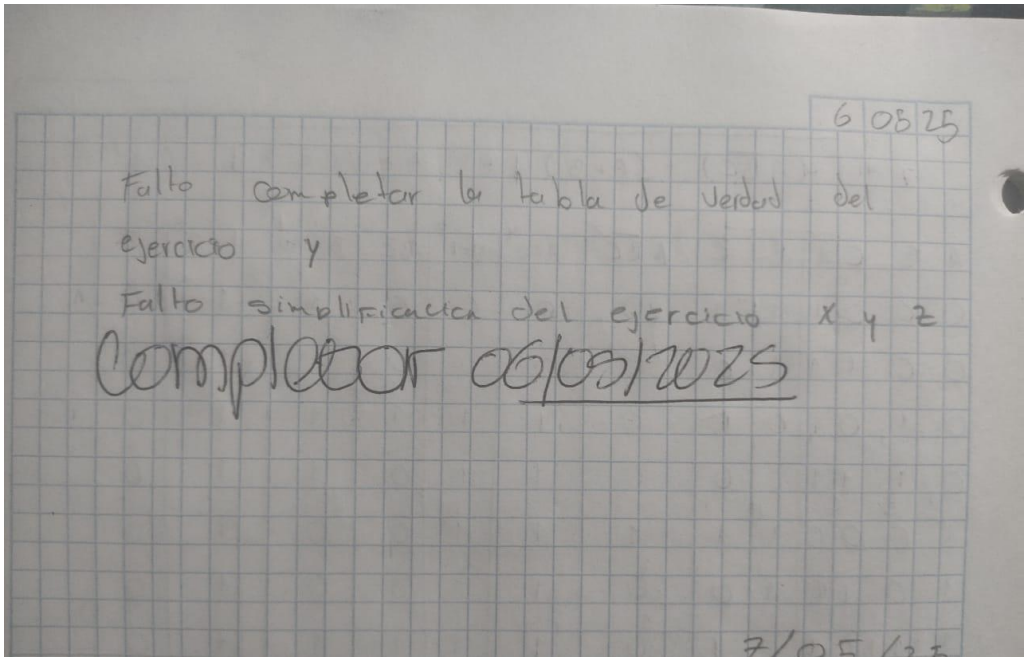

7/05/23
Se presentan pruebas de circuito y codigo
Se hace comprovacion de codigo con el circuito
Se debe de hacer correcciones ya que no coinciden algunos de los valores
pend. verificar pruebas hexa y octal 07/05/2025

4.


Listo Falta entrega de binario
validar bit de signo hexa y la operacion binaria; 14/05/2025

Trabajo en clase

1. 

2. 

Conclusiones

El Proyecto N° 2 confirma la correspondencia y fiabilidad entre la lógica de software y de hardware para la implementación de una calculadora multibase. La arquitectura modular demostrada facilita la prueba, la depuración y futuras ampliaciones, mientras que la simulación en Logisim permite verificar a nivel de bits el funcionamiento de sumadores y restadores. La posible implantación en Arduino abre un camino práctico para llevar la solución a un entorno físico, aunque exige ajustes de optimización. Como evolución futura, se sugiere incorporar operaciones de multiplicación y división multibase y explorar técnicas de suma rápida como carry-lookahead para mejorar el rendimiento en bases elevadas.

Bibliografía

Mano, M. M., & Kime, C. R. (2017). Logic and Computer Design Fundamentals (5th ed.). Pearson Education.

Patterson, D. A., & Hennessy, J. L. (2014). Computer Organization and Design: The Hardware/Software Interface (5th ed.). Morgan Kaufmann.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.

Wakerly, J. F. (2005). Digital Design: Principles and Practices (4th ed.). Pearson Prentice Hall.

Rabaey, J. M., Chandrakasan, A., & Nikolic, B. (2003). Digital Integrated Circuits: A Design Perspective (2nd ed.). Prentice Hall.

Knuth, D. E. (1998). The Art of Computer Programming, Volume 2: Seminumerical Algorithms (3rd ed.). Addison-Wesley.

Brown, S. D., & Vranesic, Z. (2009). Fundamentals of Digital Logic with VHDL Design (2nd ed.). McGraw-Hill Education.

Hennessy, J. L., & Patterson, D. A. (2017). Computer Architecture: A Quantitative Approach (6th ed.). Morgan Kaufmann.

Stallings, W. (2012). Computer Organization and Architecture: Designing for Performance (9th ed.). Pearson.

Malvino, A. P., & Brown, D. (2011). Digital Computer Electronics (2nd ed.). McGraw-Hill Education.