1. Programa Python para Conversion de Bases, Manipulacion de Numeros Binarios usando 2do complemento y Representacion IEEE754 Punto Flotante

Andreu Lechuga, 202073595-6, andreu.lechuga@usm.cl

1. Resumen

Se desarrollo un programa en Python 3.9.2 capaz de manipular una entrada de cierto formato y realizar una de tres actividades. Estas actividades fueron (1) La conversión de un entero de su base numérica original a otra base arbitraria, con un rango de 1 al 64, (2) Suma y resta de números binarios implementando complemento a 2 y (3) Conversión de un entero base 10 a su representación binaria de punto flotante en formato IEEE754. Todos estos programas quedaron plenamente funcionales y en este informe se detallará el proceso de concepción y desarrollo de dichos.

2. Introducción

En este experimento se buscó la realización de un código sólido, integro y capaz de lograr:

(A)
La conversión de un numero arbitrario dado por consola desde su base numérica a su expresión equivalente en una base numérica a elección.
Consideraciones importantes, se trabajó con un rango para las bases del 1 (base unaria) hasta el 64. Para las bases mayores a 10 se implemento la siguiente lista de símbolos:
0123456789abcdefghijklmnopqrstuvwxyzABCDE FGHIJKLMNOPQRSTUVWXYZ+?

La función que le corresponde el punto A se llama Conversion_BaseN y recibe 3 parámetros:
- El numero a convertir
- La base del dicho numero
- La base a la que se busca convertir

(B)
Realizar operaciones básicas, suma y resta, sobre números binarios implementando Complemento a 2 (2nd Complementary).

La función que le corresponde al punto B se llama BinaryOp_Comp2 y que recibe 2 parámetros:
- Dos números en binario de 8 bit c/u

(C)
La representación de un numero entero o decimal en base 10 a su representación binaria con punto flotante en formato IEEE754.

La función que le corresponde el punto C se llama FloatingP_Represent() y recibe un único parámetro:
- Un entero (con o sin parte decimal) en base 10

A modo de apoyo se creó también la función *sum_BinaryNumbers* que suma dos números binarios de 8 bits.
También se creó la función *get_C2*, que dado un numero retornaba su complemento 2, realizando el complemento 1 y luego sumando 1 con la función *sum_BinaryNumbers*.
Por último, se crearon y usaron las funciones auxiliares *Decimal_to_Binary* y *Binary_to_Decimal* que encontraban la expresión en binario de un numero decimal arbitrario y la expresión decimal de un numero binario arbitrario, respectivamente.

Para procesar los inputs se uso el formato explicado a continuación:
La entrada (sys.stdin) será por consola y siempre será un número entero M seguido de una de tres líneas posibles según corresponda. Debe continuar recibiendo una entrada hasta que se lea un 0 como se indica más abajo.
Si el primer número M es 1, entonces le seguirá una línea de la forma ‘N B T’, donde N es un número entero, B la base en que se encuentra dicho número, y T la base a la que se debe convertir.
Si el primer número M es 2, entonces le seguirá una línea de la forma ‘X Y’, donde ambos números X e Y son números binarios de 8 bits cada uno, incluyendo ceros adelante para completar los 8 bits en caso de ser necesario.
Si el primer número M es 3, entonces le seguirá una línea de la forma ‘D’, donde D puede ser un número decimal o cualquier otro string.
Si el primer número M es 0, entonces debe finalizar la ejecución del programa.

La salida (sys.stdout), por otro lado, se diseñó de la siguiente manera:
La salida será por consola a través de un mensaje impreso con uno de dos formatos según el valor del número M.
Si el número M es 1, entonces la salida será una sola línea con el formato ‘Base T: C’, donde T es la base a la que se convirtió, y C el número convertido.
Si el número M es 2, entonces la salida serán dos líneas con el formato ‘Suma: P + Q = S’ para la primera, y ‘Resta: P + (-Q) = S’ para la segunda. En ambos casos P corresponde al primer número de la entrada, Q al segundo, y S al resultado de la operación. En el caso de la resta, -Q será el valor del

segundo número de la entrada convertido usando complemento a 2, ya que A − B = A + (−B).
En caso de que exista un overflow en alguna de las dos operaciones, la salida debe ser 'Suma: P + Q = OVERFLOW' o 'Resta: P + (-Q) = OVERFLOW' según corresponda.
Si el número M es 3, entonces la salida será una línea con un número binario en punto flotante de precisión simple (IEEE754). Debe tratar los casos excepcionales según la tabla de códigos especiales del estándar.
**

Un sistema numérico busca lograr un método de conteo valido, agrupando
(1) el conjunto de símbolos permitidos en el sistema, que a su vez conforman el alfabeto y
(2) un conjunto de reglas que indican que numero y que operaciones son válidos dentro de dicho sistema*.

La norma IEEE754 es un estándar en computación y electrónica para la representación de numero binarios con punto flotante. Hoy es la manera mas confiable de representar puntos flotantes y la más común entre computadores, respondiendo a un formato como el señalado en la Figura 1: IEEE 754 Floating-Point 32bits, ubicada en el anexo.

3. Desarrollo

Comenzando por el primer objetivo señalado en la introducción, se comenzará hablando del proceso de creación del punto (A) **Conversión de Bases Numéricas**.
Para la resolución tanto de este problema como del resto se buscó jerarquizar y abstraer la problemática en sus partes y trabajaras por separado. Siendo de esta manera, el punto (A) se subdivide en 3 partes esenciales:

(A1) Convertir el numero dado desde su base dada a su equivalente en base 10.
Esto se logra multiplicando el valor de cada uno de los dígitos con el valor de su base actual elevado a la posición en la que se encuentra. Cada uno de los productos anteriores se suman para encontrar el valor decimal equivalente.
Tomar en consideración que el rango de bases va de 1 al 64, usando los símbolos dados por el departamento.
Estos son:
0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJK
LMNOPQRSTUVWXYZ+?
Por lo que se vuelve imperativo usar la tabla ASCII para representar los valores correctamente.

(A2) Convertir el numero decimal (base 10) recién convertido a su representación equivalente en la base numérica de destino solicitada.
Esto se logra dividiendo la representación decimal por el valor de la base de destino hasta que la primera sea menor que la segunda, almacenando los restos de cada división como el numero buscado. Este resultado queda almacenado en forma de lista tomando en cuenta el siguiente paso.

(A3) Como el numero queda invertido (se invierte por razones prácticas, ver código) damos vuelta la lista e iteramos por cada valor buscando la representación ASCII correspondiente a cada uno, considerando y respetando el orden del conjunto de símbolos mencionados en A1.
De esta manera formamos el numero en su base de destino, equivalente al número de entrada, y lo retornamos en el formato correspondiente.

Continuando con el 2do objetivo, se procederá a detallar el proceso para el punto (B) **Operaciones Básicas en Binario usando Complemento 2**
Para la abstracción de este problema, se subdividió en 3 partes:
(B1) Se detecto si existía Overflow tanto para la suma como para la resta. Esto se hizo consiguiendo los valores en base 10 de cada uno de los números dados y corroborar que se mantuvieran dentro del rango $[-2^{n-1}, 2^{n-1}-1]$ para ambas operaciones mencionadas.
(B2) Para la suma, en caso de estar bien definida, se aplico la función auxiliar *sum_BinaryNumbers* para obtener el resultado.
(B3) Para la resta, en caso de estar bien definida, primero se aplicó la función *get_C2* para obtener el complemento del 2do número binario (num2). Luego se aplicó la función auxiliar *sum_BinaryNumbers* para obtener el resultado.

De esta forma realizamos la suma y resta de los números binarios dados, retornando por consola los resultados en el formato correspondiente.

Para el 3er y ultimo objetivo procedemos a desglosar el punto (C) **Representación de un numero base 10** (decimal o no) **en formato IEEE754 Punto Flotante 32bits**
Las subpartes de esta solución son 3, siendo las siguientes:
(C1) Definir el bit del signo (sign_bit) comparando el numero sobre y bajo 0. Importante puesto que es el primer bit. Si es 0 se considera positivo, en otras palabras, con valor 0.
(C2) Se revisa si tiene punto decimal o no (base 10). En caso de tenerlo se separa ambas partes, entera y decimal, y se

trabajan por separado. En caso contrario se trabaja únicamente con la parte entera
(C) Se comienza a construir la mantisa.
En caso de no tener punto decimal, primero se convierte la parte entera en su equivalente en binario de X bits, usando la función *Decimal_to_Binary*. Luego se analiza dicho equivalente binario para obtener el valor del exponente. A continuación, se le suma un desplazamiento (Bias) al exponente (+127), para otorgarle la cualidad binaria de Unsigned. Luego se convierte el exponente a su representación binaria y se le acomoda/completa con 0s a la izquierda si es necesario para darle un tamaño de 8 bits.
Por ultimo se conforma el formato IEEE754 concatenando el sign_bit (1 bit) + el exponente en binario (8 bits) + la mantisa puramente conformada de la parte entera (23 bits)

En caso contrario se comienza analizando la mantisa por separado, parte entera y parte decimal. La parte decimal se obtiene multiplicando el valor decimal por 2 y restando 1 cuando dicho valor sea mayor a 1. Almacenando y concatenando la parte entera de cada resultado se obtiene el numero binario equivalente.
La parte entera se convierte a su forma binaria equivalente usando *Decimal_to_Binary*. A continuación, se calcula el valor decimal del exponente analizando el numero binario equivalente a la parte entera. Al igual que antes, luego se le suma un desplazamiento (Bias) al exponente (+127), para otorgarle la cualidad binaria de Unsigned. Se convierte el exponente a su representación binaria y se le acomoda/completa con 0s a la izquierda si es necesario para darle un tamaño de 8 bits.
Finalmente se conforma el formato IEEE754 concatenando el sign_bit (1 bit) + el exponente en binario (8 bits) + la mantisa conformada de la parte entera (x bits) + la mantisa conformada de la parte decimal (y bits), cumpliéndose la relación: x + y = 23

4. Resultados

En el siguiente apartado se presentarán distintas entradas que demuestren el correcto funcionamiento del programa, dando paso a un análisis más en detalle en la siguiente sección.

Conversión Bases Numéricas

Nombre función:	Parámetros:
Conversion_BaseN	(num., O_base, D_base)

Ejemplo 1:

Entrada:	Salida:
Dx? 64 60	Base 60: IWD

```
308
309    Conversion_BaseN("Dx?", 64, 60)

PROBLEMS 27   OUTPUT   TERMINAL   DEBUG CONSOLE

Base 60: IWD
PS C:\Letuce\programming\Python files>
```

Figura 2: Conversión, Ejemplo 1

Ejemplo 2:

Entrada:	Salida:
11111111 1 16	Base 16: 8
11111111 1 64	Base 64: 8

```
308
309    Conversion_BaseN("11111111", 1, 16)
310    Conversion_BaseN("11111111", 1, 64)

PROBLEMS 27   OUTPUT   TERMINAL   DEBUG CONSOLE

Base 16: 8
Base 64: 8
PS C:\Letuce\programming\Python files>
```

Figura 3: Conversión, Ejemplo 2

Suma y Resta Números Binarios Complemento 2

Función:	Parámetros:
BinaryOperations_Comp2	(num1, num2)

Ejemplo 1:

Entrada:	Salida
00011011 00000101	*En Figura 4

```
311
312    BinaryOperations_Comp2("00011011", "00000101")
313

PROBLEMS 27   OUTPUT   TERMINAL   DEBUG CONSOLE

Suma   00011011 + 00000101 = 00100000
Resta  00011011 + 11111011 = 00010110
-----  ----------------------------
PS C:\Letuce\programming\Python files>
```

Figura 4: Suma y Resta, Ejemplo 1

Ejemplo 2:

Entrada:	Salida
01011101 00110111	*En Figura 5

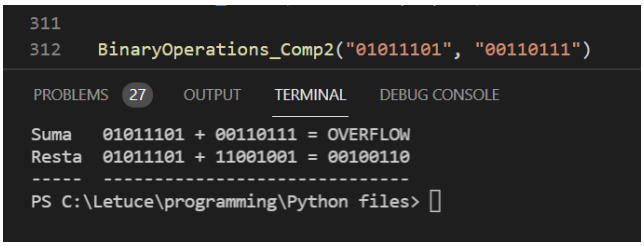

Figura 5: Suma y Resta, Ejemplo 2

Representación Numero Flotante en formato IEEE754

Función:	Parámetros:
FloatP_Represent	(num)

Ejemplo 1:

Entrada:	Salida:
-118.625	11000010111011010100000000000000

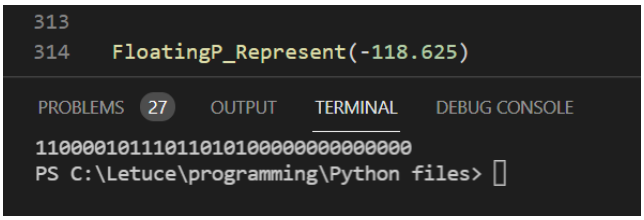

Figura 6: Floating Point IEEE754, Ejemplo 1

Ejemplo 2:

Entrada:	Salida:
263.3	01000011100000111010011001100110

Figura 7: Floating Point IEEE754, Ejemplo 2

Ejemplo 3:

Entradas:	Salida:
400000000000000000000 000000000000000000	*En Figura 8
400000000000000000000 00000000000000000	*En Figura 8

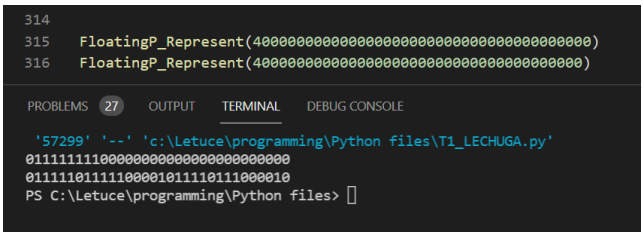

Figura 8: Floating Point IEEE754, Ejemplo 3

5. Análisis

Como es posible apreciar en el apartado anterior, existen algunos casos en especifico que valen la pena destacar. Comenzando con el Ejemplo 2 de la primera función, ilustrado en la Figura 3, es posible ver la particularidad de la base unaria (base 1). Al ser un sistema numérico basado en el número 1, todas sus casillas tendrán valores acordes con las potencias de su base. Esto haría pesar a su primera casilla 1^0, la segunda 1^1 y así sucesivamente, homogeneizando todas las casillas, un fenómeno propio del sistema unario.
Por esto mismo, la base a la que se le convierte un numero unario es indiferente al resultado de la conversión, como se muestra en la Figura 3.

Por otro lado, tenemos el caso de Overflow destacado en la Figura 8, dentro de los ejemplos de la tercera función.
Es un caso pensando para ilustrar el límite de Overflow para el formato IEEE754. En otras palabras, es lo que sucede cuando la representación binaria de 8 bits para el exponente desplazado se escapa de los rangos establecidos por [0, 2^n-1]. Importante consideración aquí es que el rango anterior es para números Unsigned, distinto al implementado para la segunda función, [-2^{n-1}, 2^{n-1}-1], para números con Complemento 2.
Los números dados como entrada difieren en 1 cero, $4x10^{38}$ y $4x10^{37}$ respectivamente. En este caso el primer numero genera un exponente de 128 (sin desplazamiento) o 255 (con desplazamiento), mientras que el exponente del segundo es de 124 (sin desplazamiento) y 251 (con desplazamiento). Considerando que son 8 bits, 2^n-1 = 255, podemos observar matemáticamente como el segundo número está dentro del rango mientras que el primero tiende a infinito, generando un resultado sin sentido.

6. Conclusión

Con el proyecto realizando se comprendieron en profundidad conceptos clave como los pilares de un sistema numérico y las reglas generales que los trascienden y comparten, la lógica binaria del complemento 2 como también su implementación y rangos de Overflow. Además del formato IEEE754, porque existe y como se utiliza (Digital Design and Computer Architecture, Harris) además de sus rangos de Overflow y Underflow también.

Mediante la superación teórica y practica de distintos imprevistos, de los más importantes señalados en el apartado de Análisis, se completó con éxito el programa a realizar.

7. Anexos

*Expresión inspirada en definición Wikipedia
**Ambos formatos de salida y entrada fueron tomados casi textuales de las instrucciones dadas por el departamento y puestas aquí

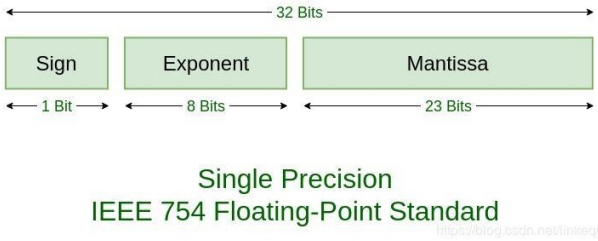

Figura 1: IEEE 754 Floating-Point 32bits