Trabajo Práctico N^{o}1

Implementación de punto flotante IEEE754 de 16 bits en Python

93.54 Métodos Numéricos

Grupo N°4

Legajo N°	Nombre
61428	Kevin Amiel Wahle
61430	Francisco Basili
61431	Nicolás Bustelo

24/03/2022

Índice

1. Introducción

El objetivo de este informe es mostrar y explicar el funcionamiento de un programa realizado en *Python*, el cual consiste en crear una clase que pueda convertir un número punto flotante de doble precisión (64 bits) a un número IEEE 754 de 16bits, a la vez que también se puedan realizar ciertas operaciones. El código se encuentra en el siguiente repositorio de GitHub.

el repo es público?

2. Clase binary16

2.1. Constructor

En el constructor de la clase se definió *"bits"* que es una lista de 16 elementos donde se encontraran los bits correspondientes a los números convertidos al formato IEEE 754 de 16bits. También se definió *"d"* el cual contiene al número que está en el formato IEEE 754 convertido en float de 16bits. Las conversiones decimal a binario y binario a decimal se realizan gracias a las funciones "dec2bin()" y "bin2dec()" respectivamente.

2.1.1. dec2bin()

El método **dec2bin(**number**)** toma el argumento number e inicialmente determina si es un caso "especial"(+inf, -inf, NaN). Si lo es, genera el arreglo de bits con el formato que tiene el estándar. En caso contrario, con funciones auxiliares (ver apartado 3.1 y apartado 3.2) calcula el signo, el exponente y la mantiza (analizando si esta última corresponde a un número Normal o uno Sub-Normal). Finalmente, los guarda en binary16.bits concatenados.

2.2. bin2dec()

El método **bin2dec()** toma el número en formato binario, lo convierte a decimal y lo guarda en una variable de 64bits. Al iniciar, tiene una etapa en la que se verifica si el arreglo binario tiene el formato de algún caso de los considerados como "especial", para crearlos específicamente. Si no tiene ninguno de dichos formatos, calcula el número pasando exponente y mantisa de binario a decimal y luego aplicando la fórmula correspondiente al caso Sub-Normal o Normal según corresponda.

2.3. Sobrecarga de Operadores

Se realizó la sobrecarga del operador multiplicación por 1, multiplicación por -1, suma y resta. El código de cada una de las implementaciones, respectivamente, se puede observar en el código a continuación:

```
1    # Multiplicaci n por +1
2    def __pos__(self):
3        if self.d == float('-inf'):
4            return binary16(float('-inf'))
5        return binary16(self.d)
6
7    # Multiplicaci n por -1
8    def __neg__(self):
9        if self.d == float('inf'):
10           return binary16(float('-inf'))
11       return binary16(-self.d)
12
13   # Suma
14   def __add__(self,other):
15       d=self.d+other.d
16       return binary16(d)
17
18   # Resta
19   def __sub__(self,other):
```

```
20          d=self.d-other.d
21          return binary16(d)
22
23      # In-Place Suma
24      def __iadd__(self,other):
25          self = binary16(self.d+other.d)
26          return self
27
28      # In-Place Resta
29      def __isub__(self,other):
30          self=binary16(self.d-other.d)
31          return self
```

Se puede observar como en todos los casos se devuelve un nuevo objeto producto de instanciar la misma clase *binary16* con el resultado de la operación realizada en cuestión.

3. Funciones varias

Para preservar la legibilidad del código se definieron dos funciones que permitían convertir de decimal a binario los diferentes atributos de un número:

3.1. exp2bin()

Se utiliza para convertir los exponentes en una lista de bits. En este caso la conversión se hace con potencias de 2 positivas (2^1, 2^2, 2^3 ,...), ya que de esta manera se ordenan en la representación IEEE 754. Para lograr esto, se va dividiendo al número por 2 y se va obteniendo el resto de manera reiterativa.

3.2. man2bin()

Se utiliza para convertir la mantisa en una lista de bits. En este caso la conversión se hace con potencias de 2 negativas (2^{-1}, 2^{-2}, 2^{-3} ,...), ya que de esta manera se ordenan en la representación IEEE 754. Para lograr esto se va multiplicando la mantisa por 2 y, en caso de que el resultado sea (o no) mayor a 1, se va reconstruyendo el número.

4. TestBench

El objetivo del TestBench es comprobar el correcto funcionamiento de los operadores al mismo tiempo que se analiza la conversión de los números a la normativa IEEE 754. Se realizaron pruebas con todos los tipos de números posibles, dentro de los cuales se encontraban: los **normales** (negativos y positivos), los **sub-normales** (negativos y positivos), el **0** o los números muy pequeños indistinguibles con el 0, el **infinito** (tanto $-\infty$, como $+\infty$) y el **NaN**.

5. Anexo

5.1. Código Clase binary16

```
# ------------------------------------------------------------------------------
#  @file     +PuntoFlotante.py+
#  @brief    +Implementaci n del punto flotante IEEE 754 de 16 bits+
#  @author   +Grupo 4+
# ------------------------------------------------------------------------------

# ------------------------------------------------------------------------------
# LIBRARIES
# ------------------------------------------------------------------------------
import math

# ------------------------------------------------------------------------------
# CLASSES
# ------------------------------------------------------------------------------
class binary16:
    def __init__(self, number):
        self.d = 0
        self.bits = [0]*16
        self.dec2bin(number)  # Devuelve si el numero es un caso extremo y cual
        self.bin2dec()

    def dec2bin(self, number):
        ne = 5  # Cantidad de Bits de Exponente
        nm = 10 # Cantidad de Bits de Mantisa
        sesgo = 2**(ne-1)-1
        expTotal = 0 # Es a lo que se eleva el 2

        # Si no es un caso extremo
        if number != float('inf') and number != float('-inf') and not math.isnan(number):
            self.bits[0] = 1 if number < 0 else 0   # Guardo el signo
            modulo = abs(number)                    # y tomo su m dulo

        # Si el n mero es NaN
        if math.isnan(number):
            self.bits = [0] + [1]*(ne+nm)
            return True

        # Si el n mero es infinito
        if number == float('inf') or number == float('-inf') or modulo > (2-2**(-nm))
    *2**(2**ne-sesgo-2):
            self.bits[0] = 1 if number < 0 else 0           # Guardo el signo
            self.bits = [self.bits[0]] + [1]*ne + [0]*nm    # Coloco 1s en el exponente y
     0s en la mantisa
            return "+inf" if self.bits[0] == 0 else "-inf"

        # Si el n mero es menor al  n mero subnormal mas peque o , lo consideramos 0
        if modulo < 2**(-nm)*2**(1-sesgo):
            self.bits = [0]*(ne+nm+1)       # Coloco 0s en el exponente y 0s en la
    mantisa
            return True

        # Si el n mero es Sub-Normal
        if modulo < 2**(1-sesgo):
            expTotal=1-sesgo                # Calculo el exponente (1 - sesgo)
            self.bits[1:6]=[0]*ne           # Coloco 0s en el exponente
            mantisa=(modulo/2**expTotal)    # Calculo la mantisa
        # Si el n mero es Normal
        else:
            expTotal = math.floor(math.log2(modulo))    # Calculo el exponente total (e -
     sesgo)
            exp = expTotal + sesgo                      # Calculo el exponente (e)
            self.bits[1:6]=exp2bin (exp, ne)            # Guardo el exponente en binario
            mantisa = (modulo/2**expTotal) - 1          # Calculo la mantisa

        self.bits[6:16]=man2bin (mantisa, nm)           # Guardo la mantisa en binario

```

```
63      def bin2dec(self):
64          ne = 5  # Cantidad de Bits de Exponente
65          nm = 10 # Cantidad de Bits de Mantisa
66          sesgo = 2**(ne-1)-1
67
68          # Caso infinito
69          if self.bits[1:] == [1]*ne + [0]*nm:
70              self.d= float('inf') if self.bits[0] == 0 else float('-inf')    #
    Depenendiendo del signo el n mero sera -inf o +inf
71              return True
72
73          # Caso NaN
74          elif self.bits[1:6] == [1]*ne:
75              self.d= float('NaN')
76              return True
77
78          # Caso 0
79          if self.bits[1:] == [0]*ne + [0]*nm:
80              self.d = float(0)
81
82          # Caso Sub-Normal
83          elif self.bits[1:6] == [0]*ne:
84              mantis = 0
85              for i in range(nm):
86                  mantis += self.bits[i+6]* 2**(-i-1)             # Calculo la mantisa en
    decimal
87              self.d = (-1)**self.bits[0]*mantis*2**(1-sesgo)     # Armo mi n mero en
    decimal
88
89          # Caso Normal
90          else:
91              mantis = 1
92              for i in range(nm):
93                  mantis += self.bits[i+6] * 2**(-i-1)            # Calculo la mantisa en
    decimal
94              expo = 0
95              for j in range(ne):
96                  expo += self.bits[j+1] * 2**(ne-j-1)            # Calculo el exponente en
     decimal
97              self.d = (-1)**self.bits[0]*mantis*2**(expo-sesgo)  # Armo mi n mero decimal
98
99      # Multiplicaci n por +1
100     def __pos__(self):
101         if self.d == float('-inf'):
102             return binary16(float('-inf'))
103         return binary16(self.d)
104
105     # Multiplicaci n por -1
106     def __neg__(self):
107         if self.d == float('inf'):
108             return binary16(float('-inf'))
109         return binary16(-self.d)
110
111     # Suma
112     def __add__(self,other):
113         d=self.d+other.d
114         return binary16(d)
115
116     # Resta
117     def __sub__(self,other):
118         d=self.d-other.d
119         return binary16(d)
120
121     # In-Place Suma
122     def __iadd__(self,other):
123         self = binary16(self.d+other.d)
124         return self
125
126     # In-Place Resta
127     def __isub__(self,other):
128         self=binary16(self.d-other.d)
129         return self
```

```
130
131 # ------------------------------------------------------------------------------
132 # FUNCTION DEF
133 # ------------------------------------------------------------------------------
134 def exp2bin (exp, expBits):     # Convierte un numero decimal en binario con potencias
    positivas
135     cont = 0
136     expb=[]
137
138     if exp > 2**expBits:        # Numero mayor de lo que puedo guardar
139         return [1, 1, 1, 1, 1]
140
141     while cont<expBits:
142         expb = [exp%2] + expb   # Divido por 2 y me quedo con el resto
143         exp = exp // 2
144         cont += 1
145     return expb
146
147 def man2bin (man, manBits):     # Convierte un numero decimal en binario con potencias
    negativas
148     cont = 0
149     manb=[]
150
151     while cont<manBits:
152         man *= 2                # Multiplico al numero por 2, y dependiendo de si es
    mayor a 1 o no, armo el binario
153         if man>=1:
154             manb = manb + [1]
155             man -= 1
156         else:
157             manb = manb + [0]
158         cont += 1
159
160     return manb
161
162 # ------------------------------------------------------------------------------
163 # ------------------------------------------------------------------------------
164 #                                 TestBench DEF
165 # ------------------------------------------------------------------------------
166 # ------------------------------------------------------------------------------
167 def operationTest(numb):
168     IeeeNumb = binary16(numb)       # Tomo un n mero
169     IeeeNumb2 = binary16(numb*2)    # Tomo otro n mero
170     res = binary16(0)               # Aqu  se almacenar  el resultado
171
172     print('Numero inicial:', numb)
173
174     print("SUMA")
175     res = IeeeNumb + IeeeNumb2
176     print('a+b: ',IeeeNumb.d, '+', IeeeNumb2.d,'=', res.d, '-> IEEE754: ', res.bits)
177
178     print("RESTA")
179     res = IeeeNumb - IeeeNumb2
180     print('a-b: ', IeeeNumb.d, '-', IeeeNumb2.d, '=', res.d, '-> IEEE754: ', res.bits)
181
182     print("SUMA2")
183     res =  IeeeNumb
184     res += IeeeNumb2
185     print('a+=b: ',IeeeNumb.d, '+=' , IeeeNumb2.d, '=', res.d,'-> IEEE754: ', res.bits)
186
187     print("RESTA2")
188     res =  IeeeNumb
189     res -= IeeeNumb2
190     print('a-=b: ',IeeeNumb.d, '-=' , IeeeNumb2.d, '=', res.d,'-> IEEE754: ', res.bits)
191
192     print("POS")
193     res = +IeeeNumb
194     print('+a: ', IeeeNumb.d, '=', res.d, '-> IEEE754: ', res.bits)
195
196     print("NEG")
197     res = -IeeeNumb
198     print('-a: ', '-', IeeeNumb.d , '=', res.d, '-> IEEE754: ', res.bits)
```

```
199
200 def test():
201     # Numero positivo
202     operationTest(4.7)
203     print("\n")
204
205     # Numero negativo
206     operationTest(-3.14)
207     print("\n")
208
209     # Numero subnormal
210     operationTest(3e-7)
211     print("\n")
212
213     # Numero subnormal negativo
214     operationTest(-3e-7)
215     print("\n")
216
217     # Numero muy cercano a cero -> 0
218     operationTest(-5e-10)
219     print("\n")
220
221     # Numero mayor al mas grande -> inf
222     operationTest(999999999)
223     print("\n")
224
225     # Numero menor al mas chico -> -inf
226     operationTest(-999999999)
227     print("\n")
228
229     # Infinito de float
230     operationTest(float('+inf'))
231     print("\n")
232
233     # Infinito negativo de float
234     operationTest(float('-inf'))
235     print("\n")
236
237     # Not a Number de float
238     operationTest(float('NaN'))
239     print("\n")
240
241 # ------------------------------------------------------------------------------
242 # MAIN
243 # ------------------------------------------------------------------------------
244 test()
```