



Instituto Tecnológico
de Buenos Aires

TRABAJO PRÁCTICO N°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	2
2. Clase binary16	2
2.1. Constructor	2
2.1.1. dec2bin()	2
2.2. bin2dec()	2
2.3. Sobrecarga de Operadores	2
3. Funciones varias	3
3.1. exp2bin()	3
3.2. man2bin()	3
4. TestBench	3
5. Anexo	4
5.1. Código Clase binary16	4

1. Introducción

El objetivo de este informe es presentar 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 re-
po es
público?
Convie-
ne?

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 64bits. 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  # Multiplicación por +1
2  def __pos__(self):
3      if self.d == float('-inf'):
4          return binary16(float('-inf'))
5      return binary16(self.d)
6
7  # Multiplicación 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

```

1 # -----
2 # @file      +PuntoFlotante.py+
3 # @brief     +Implementación del punto flotante IEEE 754 de 16 bits+
4 # @author    +Grupo 4+
5 # -----
6
7 # -----
8 # LIBRARIES
9 # -----
10 import math
11
12 # -----
13 # CLASSES
14 # -----
15 class binary16:
16     def __init__(self, number):
17         self.d = 0
18         self.bits = [0]*16
19         self.dec2bin(number) # Devuelve si el numero es un caso extremo y cual
20         self.bin2dec()
21
22     def dec2bin(self, number):
23         ne = 5 # Cantidad de Bits de Exponente
24         nm = 10 # Cantidad de Bits de Mantisa
25         sesgo = 2**(ne-1)-1
26         expTotal = 0 # Es a lo que se eleva el 2
27
28         # Si no es un caso extremo
29         if number != float('inf') and number != float('-inf') and not math.isnan(number):
30             self.bits[0] = 1 if number < 0 else 0 # Guardo el signo
31             modulo = abs(number) # y tomo su modulo
32
33         # Si el numero es NaN
34         if math.isnan(number):
35             self.bits = [0] + [1]*(ne+nm)
36             return True
37
38         # Si el numero es infinito
39         if number == float('inf') or number == float('-inf') or modulo > (2-2**(-nm))
40         *2**(2**ne-sesgo-2):
41             self.bits[0] = 1 if number < 0 else 0 # Guardo el signo
42             self.bits = [self.bits[0]] + [1]*ne + [0]*nm # Coloco 1s en el exponente y
43             0s en la mantisa
44             return "+inf" if self.bits[0] == 0 else "-inf"
45
46         # Si el numero es menor al numero subnormal mas pequeño, lo consideramos 0
47         if modulo < 2**(-nm)*2**(1-sesgo):
48             self.bits = [0]*(ne+nm+1) # Coloco 0s en el exponente y 0s en la
49             mantisa
50             return True
51
52         # Si el numero es Sub-Normal
53         if modulo < 2**(1-sesgo):
54             expTotal=1-sesgo # Calculo el exponente (1 - sesgo)
55             self.bits[1:6]=[0]*ne # Coloco 0s en el exponente
56             mantisa=(modulo/2**expTotal) # Calculo la mantisa
57
58         # Si el numero es Normal
59         else:
60             expTotal = math.floor(math.log2(modulo)) # Calculo el exponente total (e -
61             sesgo)
62             exp = expTotal + sesgo # Calculo el exponente (e)
63             self.bits[1:6]=exp2bin (exp, ne) # Guardo el exponente en binario
64             mantisa = (modulo/2**expTotal) - 1 # Calculo la mantisa
65
66             self.bits[6:16]=man2bin (mantisa, nm) # Guardo la mantisa en binario
67

```

```

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') #
71         Dependiendo del signo el n mero sera -inf o +inf
72         return True
73
74     # Caso NaN
75     elif self.bits[1:6] == [1]*ne:
76         self.d= float('NaN')
77         return True
78
79     # Caso 0
80     if self.bits[1:] == [0]*ne + [0]*nm:
81         self.d = float(0)
82
83     # Caso Sub-Normal
84     elif self.bits[1:6] == [0]*ne:
85         mantis = 0
86         for i in range(nm):
87             mantis += self.bits[i+6]* 2**(-i-1) # Calculo la mantisa en
88             decimal
89         self.d = (-1)**self.bits[0]*mantis*2**(1-sesgo) # Armo mi n mero en
90         decimal
91
92     # Caso Normal
93     else:
94         mantis = 1
95         for i in range(nm):
96             mantis += self.bits[i+6] * 2**(-i-1) # Calculo la mantisa en
97             decimal
98         expo = 0
99         for j in range(ne):
100             expo += self.bits[j+1] * 2**(ne-j-1) # Calculo el exponente en
101             decimal
102         self.d = (-1)**self.bits[0]*mantis*2**(expo-sesgo) # Armo mi n mero decimal
103
104     # Multiplicaci n por +1
105     def __pos__(self):
106         if self.d == float('-inf'):
107             return binary16(float('-inf'))
108         return binary16(self.d)
109
110     # Multiplicaci n por -1
111     def __neg__(self):
112         if self.d == float('inf'):
113             return binary16(float('-inf'))
114         return binary16(-self.d)
115
116     # Suma
117     def __add__(self,other):
118         d=self.d+other.d
119         return binary16(d)
120
121     # Resta
122     def __sub__(self,other):
123         d=self.d-other.d
124         return binary16(d)
125
126     # In-Place Suma
127     def __iadd__(self,other):
128         self = binary16(self.d+other.d)
129         return self
130
131     # In-Place Resta
132     def __isub__(self,other):
133         self=binary16(self.d-other.d)
134         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()
```