Introducción a la computación

Taller Flash Nº 4

Fecha de entrega: Miércoles 28 de Octubre de 2019 10:00 hs.

1. Representación de números con coma

1.1. Punto fijo

Los números enteros se representan dentro de la computadora en binario. Si extendemos la representación suponiendo un componente divisor fijo sobre esos valores, podremos representar algunos números reales con notación fija. Esta primera aproximación tiene la desventaja de que no puede representar números muy grandes, ni fracciones muy pequeñas. Además, la parte fraccionaria del cociente en una división de grandes números puede perderse.

Tomemos como ejemplo:

$$\frac{10{,}05 - 15{,}75}{3{,}1415}$$

Para este ejemplo vamos fijar el divisor en 1000 de forma que podamos tener una notación fija de 3 decimales.

```
1  divisor = 1000
2
3  def fix(value):
4      return int(value * divisor)
5
6  def unfix(value):
7      return float(value) / divisor
8
9  print((fix(10.05)-fix(15.75))//fix(3.1415))
10
11 print(unfix(fix(fix(10.05)-fix(15.75))//fix(3.1415)))
```

Ambos `print` resuelven la división, pero con una leve diferencia. Uno es capaz de mantener la precisión y la otra no. **Justifique.**

Modificar la precisión a 5 decimales y codificar los siguientes números y cálculos en notación fija:

1. 0,00000456
2. 9223372036854775807
3. $43{,}001 \cdot 0{,}00751$

4. $0{,}00000001 \cdot 1{,}4142135623730951$
5. $0{,}1 + 0{,}1 + 0{,}1 - 0{,}3$

Los valores obtenidos los vamos a usar para comparar con la representación en punto flotante.

1.2. Punto Flotante

Para números decimales usamos tradicionalmente la notación científica. Con ella podemos representar números muy grandes como 976000000000000 con la expresión $9{,}76 \times 10^{14}$ y otros muy pequeños como 0,000000000000976 con la expresión $9{,}76 \times 10^{-14}$. Lo que se hace es deslizar el punto decimal a una ubicación conveniente y usar el exponente de 10 para mantener el valor original. Esto permite describir números muy grandes y muy pequeños con un pequeño número de dígitos.

Los números de punto flotante se describen cómo:

$$\pm S \cdot B^{\pm E}$$

Este número puede almacenarse usando estos tres campos:

- Signo: positivo o negativo.
- Mantisa S
- Exponente E

Representar los siguientes números y cálculos usando el tipo de datos `float` de Python y comparar con los obtenidos previamente:

1. 0,00000456
2. 9223372036854775807
3. $43{,}001 \cdot 0,00751$
4. $0{,}00000001 \cdot 1{,}4142135623730951$
5. $0{,}1 + 0{,}1 + 0{,}1 - 0{,}3$

Listar dos ventajas de la representación de punto fijo por sobre la de punto flotante.

2. Estándar IEEE-754

La representación de números de punto flotante ha sido estandarizada por la IEEE con el normativa 754, adoptada en 1985 y revisada en el 2008. El estándar fue desarrollado para facilitar la portabilidad de los programas y datos de un procesador a otro.

Dicho estándar define reglas aritméticas y de representación.

2.1. Representación

El formato define cinco formatos binarios de diferentes tamaños. Los básicos son de 32, 64 y 128 bits, cada uno con su correspondiente exponente de 8, 11 y 15 bits respectivamente. En la siguiente tabla describe los cinco formatos binarios detallados en la normativa.

Nombre	Nombre común	Bits Man.	Bits Exp.	Sesgo Exp.	E_{min}	E_{max}
binary16	Media precisión	11	5	$2^4 - 1 = 15$	-14	15
binary32	Simple precisión	24	8	$2^7 - 1 = 127$	-126	127
binary64	Doble precisión	53	11	$2^{10} - 1 = 1023$	-1022	1023
binary128	Cuádruple precisión	113	15	$2^{14} - 1 = 16383$	-16382	16383
binary256	óctuple precisión	237	19	$2^{18} - 1 = 262143$	-262142	262143

Todos los números, para todos los formatos, son representados normalizados. Esto significa que los números deben escribirse de tal manera que la mantisa siempre quede con el bit más significativo en 1, excepto para el cero. Ese 1 se toma implícito y no se almacena en la secuencias de bits de la mantisa.

Para representar la mantisa se usa la representación entera con signo. En cambio para el exponente se representa con un exceso a $2^{e-1} - 1$, donde e es la cantidad de bits del exponente.

Tomemos otra vez Python para ver el contenido de un número en punto flotante. El módulo struct[1] nos permite empaquetar (*pack*) y desempaquetar (*unpack*) contenidos en bits.

Vamos a utilizar la siguiente función para ver los valores de los bits correspondientes a la codificación en IEEE-754 de un número arbitrario:

```
1 from struct import pack
2 def verBits(n):
3     return ''.join(['{:08b}'.format(c) for c in pack('!f', n)])
```

Utilizando esta función para codificar el 1, responder las siguientes preguntas:

1. ¿Qué valor debería ser la mantisa? No se olvide que está normalizada.
2. Conociendo la mantisa, ¿qué valor debería tener el exponente?
3. Identifique los bits del exponente.
4. Realice el mismo experimento con el valor -1.
5. Identifique el bit de signo.
6. Identifique el cambio de signo del exponente usando el valor 0,5.
7. Diagrame la organización de los bits e indique a qué formato IEEE-754 corresponde.
8. Pruebe los siguientes valores y registre cómo se codifican: `float('Nan')`, `float('inf')`, `float('-inf')`.

[1] `https://docs.python.org/3/library/struct.html`

3. Unidad de Punto Flotante

La *Unidad de Punto Flotante* o *FPU*, es la unidad lógica que realiza los cálculos de los números representados en punto flotante. Aunque en un principio estuvieron separados del procesador, actualmente están integrados y son esenciales para los cálculos del día a día (`http://www.informit.com/articles/article.aspx?p=130978&seqNum=20`). La mayoría presentan otros beneficios tales como registros especiales para el cálculo en punto flotante, la ejecución de cálculos en paralelo, etc.

La unidad FPU de los procesadores Intel contiene 8 registros llamados `ST(0)` a `ST(7)`, que pueden accederse por las instrucciones desde una estructura de pila, donde `ST(0)` es el tope de la pila, o en forma indexada. Algunas de las instrucciones disponibles son:

FABS ABSolute value of ST(0)

FADD ADD two floating point values

FADDP ADD two floating point values and Pop ST(0)

FCHS CHange the Sign of ST(0)

FDIV DIVide two floating point values

FDIVP DIVide two floating point values and Pop ST(0)

FDIVR DIVide in Reverse two floating point values

FDIVRP DIVide in Reverse two floating point values and Pop ST(0)

FMUL MULtiply two floating point values

FMULP MULtiply two floating point values and Pop ST(0)

FRNDINT RouND ST(0) to an INTeger

FSQRT Square RooT of ST(0)

FSUB SUBtract two floating point values

FSUBP SUBtract two floating point values and Pop ST(0)

FSUBR SUBtract in Reverse two floating point values

FSUBRP SUBtract in Reverse two floating point values and Pop ST(0)

3.1. Programación de Pila

Vamos a usar algunas de estas instrucciones dentro de la FPU para poder realizar cálculos de punto flotante. Para simplificar su uso vamos a usar el ensamblador embebido en `gcc`. El código puede compilarse fácilmente usando el comando `gcc fpu.c -o fpu`, y testeado ejecutando `./fpu`.

```c
#include <stdio.h>

int main(void)
{
   double y;
   const double k = 150.0;

   __asm__ ("fldl %1;"
            "fld1;"
            "faddp;"
            "fstl %0;" : "=m" (y) : "m" (k) );

   printf("%e\n", y);

   return 0;
}
```

Este código simplemente incrementa 150 en 1. Para esto, primero apila 150 en la FPU y luego apila un 1 usando `fld1`. Suma los dos valores en el tope de la pila, y para terminar desapila y devuelve el valor en la variable `y`.

Muestre una forma de modificar el código para que sume 5 a 150, limitándose a las instrucciones ya usadas.

3.1.1. Dos operandos

En este ejercicio vamos a trabajar con instrucciones con dos parámetros, usando de base el siguiente código:

```
#include <stdio.h>

int main(void)
{
   double r;
   const double a = 1;
   const double b = 0.1;

   __asm__ ("fldl %1;"
            "fldl %2;"
            "faddp;"
            "fstl %0;" : "=m" (r) : "m" (a), "m" (b) );

   printf("%e\n", r);

   return 0;
}
```

En el código ejemplo sumamos $(1 + 0{,}1)$. Ahora, codifique las siguientes operaciones:

- $43{,}001 \cdot 0{,}00751$
- $0{,}00000001 \cdot 1{,}4142135623730951$
- $0{,}1 + 0{,}1 + 0{,}1 - 0{,}3$

¿Puede observar el mismo comportamiento que en los ejercicios anteriores?

Condiciones de entrega:

- Generar un único archivo Python con la solución, con los comentarios correspondientes.
- Se evaluará la correctitud del código producido, su claridad y legibilidad; y el uso de la herramienta `git`.
- Asegurarse de que todos/as los/as docentes del curso tengan permisos de lectura (*Reporter*) en el repositorio de Gitlab (ver usuarios en el Campus). Se descargará la última versión de los archivos directamente de ahí, luego de ser informados de que el taller se encuentra listo.
- Para la materia de grado, el mail deberá tener el siguiente *subject*:
  "[Flash 4]: Gonzalez 666/21 (grado)".
- Para posgrado, el mail deberá tener el siguiente *subject*:
  "[Flash 4]: <apellido> <DNI, Pasaporte o Libreta> (posgrado)". Por ejemplo,
  "[Flash 4]: Trump 34509ABD (posgrado)".
- Se revisará que la forma del *subject* se corresponda **exactamente** con el formato pedido.
- Enviar dicho aviso por correo electrónico a la lista de los docentes de la materia: **icb-doc@dc.uba.ar**.
- Entregar **de forma impresa** el código del taller.
  Importante: Solo se admite la entrega por medio de `Gitlab`. Se buscarán el/los archivos en la carpeta correspondiente (en este caso, dentro de `Taller4`).
  No se aceptarán las entregas de código por mail o solo en forma impresa.