Plancha 3:
Punto flotante

2020 – Arquitectura del Computador
Licenciatura en Ciencias de la Computación

1. Introducción

Esta plancha de ejercicios trabaja la representación de números en punto flotante, en los lenguajes C y ensamblador de x86_64.

2. Procedimiento

Resuelva cada ejercicio en computadora. Cree un subdirectorio dedicado para cada ejercicio, que contenga todos los archivos del mismo. Para todo ejercicio que pida escribir código, genere un programa completo, con su función `main` correspondiente; evite dejar fragmentos sueltos de programas.

Asegúrese de que todos los programas que escriba compilen correctamente con `gcc`. Se recomienda además pasar a este las opciones `-Wall` y `-Wextra` para habilitar advertencias sobre construcciones cuestionables en el código.

Una vez terminada la plancha, suba una entrega al repositorio Subversion de la materia, creando una etiqueta en el subdirectorio correspondiente a su grupo:
`https://svn.dcc.fceia.unr.edu.ar/svn-no-anon/lcc/R-222/Alumnos/2020/`

3. Ejercicios en C

1) Haga dos funciones o macros de C para extraer la fracción y el exponente de un `float` sin usar variables auxiliares.

Sugerencia: utilice corrimientos de bits y máscaras. Luego use los tipos definidos en la cabecera `ieee754.h` para corroborar.

2) El siguiente programa muestra algunas cualidades de *NaN* (*Not A Number*) y la función `isnan` de C, que indica si un flotante es *NaN*.

```
#include <stdio.h>
#include <math.h>

int main(void)
{
    float g = 0.0;
    float f = 0.0 / g;
    printf("f: %f\n", f);
    // ADVERTENCIA: 'NAN' es una extensión de GCC.
    if (f == NAN) {
```

```
        printf("Es NAN\n");
    }
    if (isnan(f)) {
        printf("isNaN dice que sí\n");
    }
    return 0;
}
```

a) El programa muestra que comparar con `NAN` retorna siempre falso y para saber si una operación dio *NaN* se puede usar `isnan`. Utilizando las funciones del ejercicio anterior, implemente una función `myisnan` que haga lo mismo que la función `isnan` de C.

b) Implemente otra función, `myisnan2`, que haga lo mismo pero utilizando solo una comparación y sin operaciones de bits.

c) ¿Ocurre lo mismo con $+\infty$?

d) ¿Qué pasa si se suma un valor a $+\infty$?

3) Convierta a `double` y `float` norma *IEEE 754* la constante número de Avogadro: $N = 6,02252 \times 10^{23}$. Realice el cálculo de manera explícita y luego corrobore el resultado mediante un programa que aproveche las herramientas provistas en el ejercicio 1. Tenga en cuenta que $\log_b x = n \Leftrightarrow x = b^n$.

4) Suponiendo que tenemos un sistema de representación donde un número en punto flotante N se representa usando base b y exceso q como:

$$N = (-1)^{signo} \times 1.f \times b^{e-q}$$

con $0 \leq f < 1$.

a) Implemente la suma y producto de números base 2 y exceso 30000. Use 18 bits para representar la fracción f y 16 para representar el exponente e.

b) ¿Cuáles números son el mayor y el menor en esta representación?

Sugerencia: use campos de bits.

5) Realice el procedimiento de suma (simple precisión) del número $1{,}75 \times 2^{-79}$ con el siguiente número expresado en IEEE 754:

`0|00110000|10100000000000000000000`

4. Ejercicios en ensamblador

6) Implemente en ensamblador de x86_64 la función:

```
int solve(float a, float b, float c, float d, float e, float f, float *x, float *y);
```

que resuelva el sistema de ecuaciones:

$$\begin{aligned} ax + by &= c \\ dx + ey &= f \end{aligned} \tag{1}$$

y escriba el resultado en los punteros `x` e `y`. La función debe devolver 0 si encontró una única solución y -1 en caso contrario.

7) Implemente en ensamblador la siguiente función:

```
void sum(float *a, float *b, int len);
```

que suma dos arreglos de flotantes de longitud `len` dejando el resultado en `a`. Utilice instruciones escalares (i.e.: `addss`).

8) Reimplemente la función anterior utilizando instrucciones SIMD (formato packed). Tenga en cuenta que la longitud del arreglo no necesariamente es múltiplo de 4 (en cuyo caso puede terminar los cálculos utilizando instrucciones escalares). Llámela `sum_simd`. Utilizando la función `clock_gettime` compare el tiempo computacional de cada implementación para arreglos de distinto tamaño (de 1000 a 100.000.000 elementos).