Laboratório 16
Representação de Ponto Flutuante

O objetivo deste laboratório é implementar operações sobre números de ponto flutuante **sem usar operações de ponto flutuante**, isto é, operando diretamente sobre a representação binária desses números.

Como, em C, operações de manipulação de bits só são permitidas para valores do tipo inteiro, podemos utilizar uma *union* como a definida abaixo para obter a representação de bits de um *float* como um *unsigned int* e vice-versa, ou seja, obter um valor *float* a partir de sua representação binária.

```
typedef union {
  float f;
  unsigned int i;
} U;

U u;
float f1 = 0.125;
unsigned int u1;

u.f = f1;  /* armazena o float na union */
u1 = u.i;  /* obtem a representação "manipulável" do float */

u.i = u1;  /* armazena a representação binária na union        */
f1 = u.f;  /* obtém o float correspondente a essa representação */
```

Algumas macros úteis:

- Macro para compor a representação de um valor float, dados seus componentes (s,frac,exp):

```
#define makefloat(s,e,f) ((s & 1)<<31 | (((e) & 0xff) << 23) | ((f) & 0x7fffff))
```

- Macros para extrair as partes de um float a partir da sua representação:

```
#define getsig(x)  ((x)>>31 & 1)
#define getexp(x)  ((x)>>23 & 0xff)
#define getfrac(x) ((x) & 0x7fffff)
```

1) Escreva uma função C chamada “float2” que, **sem usar operações de ponto flutuante**, multiplique o valor de seu parâmetro por 2:

```
float float2(float f);
```

DICA: se você tem a representação de um valor na forma $\mathbf{x * 2^y}$, qual é o novo valor de y se você multiplicar esse valor por 2?

2) Escreva uma função C chamada “int2float” que converte um número no formato *int* para o formato *float*.

```
float int2float(int i);
```

Atenção: Não é para fazer uma conversão direta, como “`f = (float)i`”, pois o C irá converter o formato inteiro para ponto flutuante, e a proposta do laboratório é que você faça conversão.

Dicas:

- Para converter um inteiro para o formato *float*, precisamos encontrar os valores de **s**, **M** e **E** que correspondem à representação do valor do inteiro no formato *float*, isto é, na forma
  $\mathbf{(-1)^s\ M\ 2^E}$
  A partir dos valores de **M** e **E** você pode determinar os valores das “partes” **frac** e **exp**.
- Trate o 0 (zero) como um caso especial (sua conversão é imediata).
- Verifique se o valor inteiro é positivo ou negativo, e guarde essa informação para preencher o bit de sinal da representação IEEE. Trabalhe então com o valor **absoluto** do inteiro a ser convertido para encontrar os valores do expoente e mantissa.
- Considere a representação do valor absoluto do inteiro em binário (representação sem sinal). Esse valor é uma soma $\mathbf{2^i + 2^j + 2^k + ...}$, onde **i** é a posição do bit "1" mais significativo dessa representação (isto é, o bit “1” com a “maior” posição). Você vê alguma relação entre **i** e **E**?
- Lembre-se que **M** é um valor na forma “**1.frac**”. A sequência de bits de **frac** deve iniciar na posição 22 da representação IEEE do valor *float*.

Use o código da próxima página para testar a sua função de conversão:

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

#define getsig(x)         ((x)>>31 & 1)
#define getexp(x)         ((x)>>23 & 0xff)
#define getfrac(x)        ((x) & 0x7fffff)
#define makefloat(s,e,f) ((s & 1)<<31 | (((e) & 0xff) << 23) | ((f) & 0x7fffff))

typedef union {
  float f;
  unsigned int i;
} U;

float int2float(int i);

int main() {
   int i;

   printf("\n******** int2float ****************\n");
   i = 0;
   printf(" %d -> %+10.4f\n", i, int2float(i));
   i = 1;
   printf(" %d -> %+10.4f\n", i, int2float(i));
   i = -1;
   printf(" %d -> %10.4f\n", i, int2float(i));
   i = 0x7fffffff;
   printf(" %d -> %+10.4f\n", i, int2float(i));
   i = -i;
   printf(" %d -> %+10.4f\n", i, int2float(i));
   i = 12345;
   printf(" %d -> %+10.4f\n", i, int2float(i));
   i = -12345;
   printf(" %d -> %+10.4f\n", i, int2float(i));
   return 0;
}
```