Ficha 2 – Vírgula flutuante e SIMD

1. Escrever fragmentos de código *assembly* `RISC-V` que implementem o seguinte código em C.

a)
```
float B = 7.8, M = 3.6, N = 7.1;
float P = -M * (N+B);
```

b)
```
int    W = 7;
double X = 7.1;
double Y = sqrt(X) + W;
```

2. Escrever e testar programas, envolvendo dados em vírgula flutuante com precisão simples, para calcular:

a) o valor da expressão $\frac{(A-B)\times C}{D+A-3}$.

b) a área de um círculo dado o respetivo raio (considerar $\pi \approx 3{,}141\,59$).

c) a distância entre dois pontos, $P(x_1, y_1)$ e $Q(x_2, y_2)$, dada por $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

3. Escrever a sub-rotina `calc_poly` que calcula o polinómio $p(x) = 1{,}5\,x^3 - 12{,}5\,x + 7$ para valores de x pertencentes a $\{0;0{,}1;0{,}2;\cdots;9{,}9;10{,}0\}$ (ao todo são 101 valores). O único argumento a passar à sub-rotina é o endereço do vetor a ser preenchido com os valores $p(0), p(0{,}1), \cdots, p(9{,}9)$ e $p(10{,}0)$. Considerar dados com precisão simples.

4. O cálculo do polinómio $p(x) = a_0 + a_1x + a_xx^2 + \cdots + a_{n-1}x^{n-1}$ pode ser realizado através do cálculo de $p(x) = a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-2} + a_{n-1}x)))$. Esta expressão minimiza o número total de operações necessárias para o cálculo do polinómio, sendo o processo conhecido por método de Horner.

Desenvolver a sub-rotina `Horner` que, para um dado x, calcula o valor de um polinómio definido pelos seus n coeficientes $a_0, a_1, \ldots, a_{n-1}$ contidos no vetor `coefs`. Assumir que o protótipo desta sub-rotina em C é:

```
double Horner(double x, double *coefs, int n)
```

5. Sejam $X = [x_1, x_2, ..., x_n]$ e $Y = [y_1, y_2, ..., y_n]$ dois vetores de n números reais ($n > 0$).

O seu produto interno é dado por:

$$X \cdot Y = x_1 \times y_1 + x_2 \times y_2 + ... + x_n \times y_n$$

Escrever e testar a sub-rotina `prodint_VF` que calcula o produto interno de X e Y assumindo que não ocorre *overflow*. Considerar o seguinte protótipo da função a chamar em C para executar a sub-rotina:

```
double prodint_VF(float *X, float *Y, int n)
```

6. Considerar um vetor V com n valores do tipo `float`. Escrever a sub-rotina `conta_intervalo` que determina o número de valores do vetor que pertencem ao intervalo $[a; b]$. Assumir que para executar esta sub-rotina é chamada a função em C com o seguinte protótipo:

```
int conta_intervalo(float *V, int n, float a, float b)
```

7. Considerar a função $f(x), x \in \mathbb{R}$, definida por

$$f(x) = \begin{cases} \sqrt{(x+\pi)^3} & \text{se} \quad x \geq 0 \\ \dfrac{1}{\sqrt{4-x}} & \text{se} \quad x < 0 \end{cases}$$

Implementar a sub-rotina `rotF` que calcula o valor da função para qualquer valor de x. Considerar que o protótipo da função a invocar em C é:

```
double rotF(double x)
```

8. A função erf(x) tem a seguinte aproximação racional para $x \geq 0$:

$$\text{erf}(x) \approx 1 - \frac{1}{(1 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4)^4}$$

em que $a_1 = 0{,}278\,393$, $a_2 = 0{,}230\,389$, $a_3 = 0{,}000\,972$ e $a_4 = 0{,}078\,108$.

a) Implementar a sub-rotina `erfpos` que calcula o valor de $\text{erf}(x)$ usando a aproximação indicada. Considerar que o protótipo da função a invocar em C é:

```
double erfpos(double x)
```

b) A função $\text{erf}(x)$ é ímpar, ou seja, $\text{erf}(-x) = -\text{erf}(x)$.
Apresentar uma sub-rotina que calcula $\text{erf}(x)$, para qualquer valor de x, com recurso à sub-rotina `erfpos` da alínea anterior. O protótipo da nova sub-rotina é:

```
double erf(double x)
```

9. Implementar um programa que produza uma tabela de valores da função $y(x) = 100 + 50\cos(x)$ com $x \in [0°; 90°]$ (x em graus). Para isso, proceder da seguinte maneira:

a) Escrever a sub-rotina `cosseno` que calcula o cosseno de um ângulo, expresso em radianos, usando a seguinte variante da fórmula de Taylor:

$$\cos(x) \approx 1 - x^2\left(\frac{1}{2!} - x^2\left(\frac{1}{4!} - x^2\left(\frac{1}{6!} - x^2\left(\frac{1}{8!} - x^2\left(\frac{1}{10!}\right)\right)\right)\right)\right)$$

Assumir o seguinte protótipo:

```
double cosseno(double x)
```

Sugestão: Declarar um vetor com as constantes $n!$ pré-calculadas.

b) Usando a sub-rotina da alínea anterior, apresentar uma sub-rotina `func` para calcular o valor de $y(x) = 100 + 50 \times \cos(x)$ com x em graus. Considerar o protótipo seguinte:

```
double func(double x)
```

c) Escrever o programa que utiliza a sub-rotina `func` para calcular e armazenar em memória a sequência de valores de $y(x)$ resultante dos valores inteiros de x entre $0°$ e $90°$.

Obs.: Nos exercícios seguintes pretende-se utilizar instruções SIMD (*Single Instruction Multiple Data*) para processamento de dados inteiros. Por omissão, assumir que o número de elementos dos vetores a processar é múltiplo de 4.

10. Considerar um vetor V com n números inteiros de 8 bits (tipo `char` em C/C++) com dimensão favorável ao paralelismo do processamento a realizar.

a) Desenvolver a sub-rotina `conta_inf` que determina quantos elementos do vetor V são inferiores a um valor lim. Considerar que o protótipo da função a invocar em C é:

```
int conta_inf(char *V, int n, char lim)
```

b) Desenvolver a sub-rotina `conta_ocorr` que determina o número de ocorrências de um elemento val no vetor V. O protótipo da função a invocar em C é:

```
int conta_ocorr(char *V, int n, char val)
```

11. Considerar os vetores R e S com n inteiros do tipo `short int` (*halfword*). Implementar a sub-rotina `prodintV` que calcula o produto interno de R e S aproveitando o paralelismo das instruções SIMD. Considerar o protótipo seguinte:

```
short int prodintV(short int *R, short int *S, int n)
```

12. Considerar os vetores P, Q e R, contendo n valores representados em 16 bits. Assumir o protótipo da função a chamar em C para executar as sub-rotinas seguintes. Assumir a não ocorrência de *overflow*.

a) Implementar a sub-rotina `somaV` que calcula o vetor soma, $P + Q$, e armazena-o em R.

```
void somaV(short int *P, short int *Q, short int *R, int n)
```

b) Implementar a sub-rotina `altV` que multiplica cada elemento de P pelo escalar k.

```
void altV(short int *P, int n, short int k)
```

c) Implementar a sub-rotina `msubV` que, utilizando as sub-rotinas anteriores (`somaV` e `altV`), calcula $P - k\,Q$ e armazena o resultado em R.

```
void msubV(short int *P, short int *Q, short int *R, int n, short int k)
```

13. Considerar a seguinte função escrita em C.

```c
#include <stdlib.h>
void ajuste(short int *X, short int *Y, int n, short int da)
  {
    int i;
    for (i = 0; i < n; i++)
        Y[i] = Y[i] - da * abs(X[i]);
  }
```

Implementar em *assembly* a sub-rotina `ajusteSIMD` que, utilizando instruções SIMD, produz os mesmos resultados que o código apresentado.

14. Uma sequência de n pontos (x_i, y_i) do plano está guardada em memória como um vetor de $2n$ números inteiros com 8 bits $\{x_1, y_1, x_2, y_2, \ldots, x_n, y_n\}$. Escrever a sub-rotina `mirrorSeq` que troca as coordenadas horizontal e vertical de cada ponto. O protótipo desta sub-rotina em C é:

```
void mirrorSeq(char *pt, int n)
```

15. O produto de dois números complexos $z_1 = a + b \cdot i$ e $z_2 = c + d \cdot i$ é dado por:

$$z_1 \times z_2 = (a \cdot c - b \cdot d) + (a \cdot d + b \cdot c) \cdot i$$

Pretende-se calcular o produto de dois vetores, `Z1` e `Z2`, de `n` números complexos (`n` é par) representados pelas respetivas partes real e imaginária. Considerar `Z1` $= \{a_1, b_1, a_2, b_2, \ldots, a_{\mathrm{n}}, b_{\mathrm{n}}\}$ e `Z2` $= \{c_1, d_1, c_2, d_2, \ldots, c_{\mathrm{n}}, d_{\mathrm{n}}\}$, em que a_i e c_i representam a parte real e b_i e d_i representam a parte imaginária do i-ésimo complexo dos vetores. O vetor produto é definido por

$$\texttt{Z1} \times \texttt{Z2} = \{(a_1 \cdot c_1 - b_1 \cdot d_1), (a_1 \cdot d_1 + b_1 \cdot c_1), \ldots, (a_n \cdot c_n - b_n \cdot d_n), (a_n \cdot d_n + b_n \cdot c_n)\}.$$

Implementar a sub-rotina que calcula o vetor `Z` resultante do produto dos números complexos que formam os vetores `Z1` e `Z2`. Notar que estes vetores possuem `2n` elementos. Assumir valores inteiros com 16 bits e que não ocorre *overflow* nos cálculos. O protótipo desta sub-rotina em C é:

```
void prod_complexosV(float *Z1, float *Z2, float *Z, long int n);
```

Fim do enunciado