Processamento paralelo de dados (SIMD)

Assumir por omissão que o número de elementos dos vetores a processar é múltiplo de 4.

1. Pretende-se realizar operações vetoriais utilizando instruções SIMD (*Single Instruction Multiple Data*). Considerar os vetores `P`, `Q` e `R`, contendo `n` números representados em vírgula flutuante com precisão simples. Assumir o protótipo da função a chamar em C para executar as sub-rotinas seguintes.

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

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

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

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

c) Implementar a sub-rotina `msubV` que, utilizando as sub-rotinas anteriores, calcula `P` − `k` × `Q` e armazena o resultado em `R`.

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

2. Considerar os vetores `R` e `S` com `n` elementos do tipo `int`. Implementar a sub-rotina `prodintV` que calcula o produto interno de `R` e `S` aproveitando o paralelismo das instruções SIMD.

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

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

Pretende-se implementar a sub-rotina `conta_ocorr` que determina o número de ocorrências de um inteiro `val` no vetor `V` com dimensão `n`. O protótipo da função em C a usar para invocar a sub-rotina é:

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

4. Implementar a sub-rotina `incsatV` que soma com saturação o escalar `x` a cada elemento de um vetor `Z` com `n` números inteiros. O protótipo da função em C a usar para invocar esta sub-rotina é:

```
void incsatV(int *Z, int n, int x)
```

5. Uma sequência de `n` pontos (x_i, y_i) do plano está guardada em memória como um vetor de `2n` números reais $\{x_1, y_1, x_2, y_2, \ldots, x_n, y_n\}$. Escrever uma sub-rotina que troca as coordenadas horizontal e vertical de cada ponto.

Assumir que para executar esta sub-rotina é chamada a função em C com o seguinte protótipo:

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

6. A norma de um vetor $\vec{v} = (c_1, c_2, ..., c_n)$ é definida por $\sqrt{\sum_{i=1}^{n} c_i^2}$.

Pretende-se implementar a sub-rotina `normV` que calcula a norma de $\vec{v}$ utilizando SIMD. Assumir que a dimensão do vetor é múltipla de 2.

Considerar o protótipo `double normV(double *ptV, long int n)`, em que `ptV` é o endereço do vetor e `n` é a respetiva dimensão.

7. Desenvolver uma sub-rotina para determinar quantos elementos de um vetor com `n` números de precisão simples são inferiores a um valor `lim`. Para executar esta sub-rotina é chamada a função em C com o seguinte protótipo:

```
long int conta_inf(float *V, long int n, float lim)
```

8. Considerar a seguinte função escrita em C++.

```
#include <cmath>
void  ajuste(float *X, float *Y, int n, float da)
{
  int i;
  for (i = 0; i < n; i++)
     Y[i] = Y[i] - da * fabs(X[i]);
}
```

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

9. A expressão seguinte mostra o cálculo do produto de dois números complexos $z_1 = a + b \cdot i$ e $z_2 = c + d \cdot i$.

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

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` $= \{a_1, b_1, a_2, b_2, \ldots, a_n, b_n\}$ e `Z2` $= \{c_1, d_1, c_2, d_2, \ldots, c_n, d_n\}$, 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` números reais.

Considerar que para executar esta sub-rotina é chamada a função em C com o seguinte protótipo:

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

Fim