Tratamento de dados em vírgula flutuante (SIMD)

Nota: as operações com dados em vírgula flutuante devem ser realizadas com instruções SSE.

Parte 1: Operações escalares

1. Escrever fragmentos de código *assembly* IA-32 que implementem o seguinte código C++:

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

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

2. Escrever um fragmento de código que calcule o valor da expressão $\frac{(A-B)\times C}{(D+A)+E}$, em que todos os valores são de precisão simples.

3. Escrever um programa para calcular o valor da área de um círculo dado o respetivo raio ($\pi \approx 3.1415926535897932$).

4. Escrever a sub-rotina `dist` para calcular a distância D de um ponto $P(X,Y)$ à origem. $D = \sqrt{X^2+Y^2}$. O protótipo é: `dist PROTO x:REAL8, y:REAL8, ptrD:PTR REAL8`

5. Considere o polinómio $p(x) = 1{,}5\,x^3 - 12{,}5\,x + 7$. Escreva a sub-rotina `calc_poly_tab` que calcula uma tabela dos valores do polinómio para valores de x pertencentes a {0; 0,1; 0,2; ...; 9,9; 10} (ao todo são 101 valores). A sub-rotina tem o protótipo:

`calc_poly_tab PROTO tab:PTR REAL8`

A sequência `tab` deve ser preenchida com os valores $p(0), p(0{,}1), ..., p(9{,}9)$ e $p(10)$.

6. O cálculo do polinómio $p(x) = a_0 + a_1x + a_xx^2 + \cdots + a_nx^n$ também pode ser realizado através do cálculo de $p(x) = a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + a_nx)))$ [método de Horner]. Desenvolver uma sub-rotina que calcula, para um dado x, o valor de um polinómio definido pelo seus `ncoefs` coeficientes $a_0, a_1, \ldots, a_n$ e guarda o resultado em `res`.

`horner PROTO xval:REAL8, coefs:PTR REAL8, ncoefs:DWORD, res:PTR REAL8`

7. Apresentar o código da sub-rotina que calcula o produto interno de dois vetores de n números reais ($N > 0$). Sejam $X = [x_1, x_2, ..., x_n]$ e $Y = [y_1, y_2, ..., y_n]$. O produto interno é dado por:

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

O protótipo da sub-rotina é:

`prodint PROTO vectX:PTR REAL4, vectY:PTR REAL4, N:dword, res:PTR REAL4`

8. Pretende-se escrever um programa que produza uma tabela da função $y = 100 + 50\cos(x)$ com $x \in [0°; 90°]$ (x em graus). Para isso, procede-se da seguinte maneira:

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

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

```
cosseno PROTO x:REAL8, resultado:PTR REAL8
```

Sugestão: Usar uma tabela com constantes pré-calculadas.

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

```
func PROTO graus:REAL8, resultado:PTR REAL8
```

c) Usando a sub-rotina da alínea anterior, escrever um programa para imprimir uma tabela de $y(x)$ para os valores inteiros de x entre 0° e 90°.

9. 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 a função $f(x)$ para qualquer valor de x. O respetivo protótipo é:

```
rotF PROTO argX:REAL8, resultado:PTR REAL8
```

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

$$\text{erf}(x) \approx 1 - \frac{1}{(1 + a_1x + a_2x^2 + a_3x^3 + a_4x^4)^4}$$

com

$$a_1 = 0{,}278393 \qquad a_2 = 0{,}230389 \qquad a_3 = 0{,}000972 \qquad a_4 = 0{,}078108$$

a) Apresentar uma sub-rotina que calcula o valor de erf(x) usando a aproximação indicada. Assumir que $x \geq 0$. O protótipo da sub-rotina é:

```
erfpos PROTO argX: REAL8, res:PTR REAL8
```

b) A função erf(x) é ímpar: erf($-x$) = $-$erf(x).
Apresentar uma sub-rotina que calcula erf(x) para qualquer valor de x com recurso à sub-rotina da alínea anterior. O protótipo da nova sub-rotina é:

```
erf PROTO argX: REAL8, res:PTR REAL8
```

11. Considerar uma sequência de `N` elementos do tipo `REAL4`. Escrever uma sub-rotina que determina o número de elementos desta sequência que pertencem ao intervalo `[A; B]`. A sub-rotina tem o seguinte protótipo:

```
conta_intervalo PROTO pt:ptr REAL4, N:dword, limA:REAL4, limB:REAL4
```

12. Durante a realização de uma experiência registou-se a velocidade de um corpo em função do tempo. A informação encontra-se armazenada em duas sequências de `N` elementos `seqT` e `seqV` (`N >= 2`). A velocidade medida no instante `seqT[i]` está guardada em `seqV[i]`. Os valores de `seqT` são estritamente crescentes.

Escrever uma sub-rotina que determina a velocidade do corpo no instante `instT`. Se existir um valor de `i` tal que `seqT[i]=instT`, então o valor a retornar é `seqV[i]`. No caso contrário, a função deve estimar a velocidade no instante desejado por interpolação linear dos valores registados nos dois instantes mais próximos. Assumir que `seqT[0] <= instT <= seqT[N-1]`.

A interpolação linear entre dois pontos de uma função $y_i = f(t_i)$ e $y_{i+1} = f(t_{i+1})$ aproxima $y = f(t)$, com $t_i < t < t_{i+1}$, por

$$y = y_i + \frac{y_{i+1} - y_i}{t_{i+1} - t_i} \times (t - t_i)$$

A sub-rotina deve ter o seguinte protótipo:

```
interpol PROTO seqT:PTR REAL8, seqV:PTR REAL8, instT:REAL8, resY:PTR REAL8
```

Parte 2: Operações sobre dados empacotados

Assumir que todas as sequências estão corretamente alinhadas e têm um número de elementos que é múltiplo de 4.

13. Desenvolver uma sub-rotina para somar a cada elemento de uma sequência `seqX` o elemento correspondente de outra sequência `seqY` (sequências de `N` números de precisão simples).

```
addSeq PROTO C seqX:PTR REAL4, seqY:PTR REAL4, N:DWORD
```

14. Desenvolver e testar uma sub-rotina para multiplicar todo os elementos de uma sequências de N números de precisão simples (com N múltiplo de 4) pelo mesmo valor α.

```
scaleSeq PROTO seq:PTR REAL4, N:DWORD, alfa:REAL4
```

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

```
mirrorSeq PROTO seq:PTR REAL4, N:DWORD
```

16. Desenvolver e testar uma sub-rotina para determinar quantos elementos de uma sequência de `N` números de precisão simples são inferiores a um valor β.

```
lessThan PROTO seq:PTR REAL4, N:DWORD, beta:REAL4
```

[Nota: a instrução `popcnt r32,r32/m32` coloca no destino o número de bits a "1" existentes a origem. Também pode usar dados de 16 bits.]

17. Pretende-se avaliar a eficiência de usar instruções SIMD em vez de instruções "escalares".

a) Desenvolver um programa em linguagem C/C++ que meça o tempo de execução da sub-rotina `prodint` (do exercício 7) aplicada a duas sequências de 10000 valores reais pseudo-aleatórios.(Nota: consultar anexo.)

b) Desenvolver a sub-rotina `prodintSIMD` que calcula o produto interno de dois vetores de N números reais de precisão simples (N>0, N múltiplo de 4) usando instruções SIMD.

```
prodintSIMD PROTO seqX:PTR REAL4, seqY:PTR REAL4, N:dword, res:PTR REAL4
```

Refazer as medições da alínea anterior usando agora a sub-rotina `prodintSIMD`. Comentar os resultados.

c) Refazer as medições das alíneas anteriores para sequências de 20000, 30000, 40000 e 50000 elementos e produzir um gráfico com os tempos obtidos.

18. Refazer o exercício 11 usando instruções SIMD (precisão simples), assumindo que `N` é múltiplo de 4 e que os dados estão devidamente alinhados.

19. Considerar a seguinte sub-rotina 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]);
}
```

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

```
ajusteSIMD PROTO seqX:PTR REAL4, seqY:PTR REAL4, N:DWORD, DA:REAL4
```

b) Escrever um programa em C/C++ que compare os tempos de execução das sub-rotinas `ajuste` e `ajusteSIMD` para sequências de vários tamanhos ($N \in \{10000, 20000, \ldots, 100000\}$) com valores pseudo-aleatórios. (Nota: consultar anexo.)

Fim

Anexo – Medição de tempos de execução

```
#include <ctime>
#include <iostream>
#include <random>
using namespace std;

extern "C" void prodint(float* vecX, float* vecY, int N, float* res);
extern "C" void prodintSIMD(float* vecX, float* vecY, int N, float* res);

// Numero de repetições (invocação da função)
const unsigned NREP = 100000;
// Numero de elementos dos vetores
const unsigned NELEM = 10000;
// garantir alinhamento correto
_declspec(align(16)) float seqX[NELEM];
_declspec(align(16)) float seqY[NELEM];

int main()
{
        clock_t startTime, endTime;
        float res1;
        double tempo1;

        // preeencher com valores pseudo-aleatórios
        default_random_engine generator;
        uniform_real_distribution<float> distribution(-10.0f, 10.0f);
        for (int j = 0; j < NELEM; j++) {
                seqX[j] = distribution(generator);
                seqY[j] = distribution(generator);
        }

        // início da contagem de tempo
        startTime = clock();
        for (int n = 0; n<NREP; n++)
                prodint(seqX, seqY, NELEM, &res1);
        endTime = clock();  // fim da contaem de tempo

        tempo1 = double(endTime - startTime) / CLOCKS_PER_SEC;
        cout << "[prodint "<< NELEM << "x" << NREP << "]\t\tTempo gasto (s): "
                 << tempo1 << endl;

        // mais código ...
        return EXIT_SUCCESS;
}
```