AArch64: Sub-rotinas, vírgula flutuante e SIMD

Nos exercícios cuja solução requer código "assembly" deve ser usada a seguinte abordagem:

1. Escrever a solução do exercício em *assembly* na forma de uma sub-rotina. Ter em consideração a convenção de chamada de sub-rotinas – a passagem de argumentos e a devolução do resultado faz-se nos registos `X0` a `X7`, por ordem.
2. Na função `main` do programa em C declarar os dados necessários para testar a sub-rotina, chamar a sub-rotina em causa e imprimir o resultado da execução. Ter em consideração a compatibilidade do tipo de dados das variáveis e os registos a utilizar (`Xn` ou `Wn`).
3. Caso o programa não execute como esperado, fazer *debug* no modo passo a passo e seguir atentamente o conteúdo dos registos e de memória resultante das instruções e respetivo fluxo de execução.

1. Suponha que se realiza uma sequência "misturada" de operações *push* e *pop*. As operações *push* usam os números de 0 a 9 por ordem; a seguir a cada operação de *pop* o resultado é impresso. Qual das seguintes sequências não pode ocorrer?

A. 4 3 2 1 0 9 8 7 6 5
B. 4 6 8 7 5 3 2 9 0 1
C. 2 5 6 7 4 8 9 3 1 0
D. 4 3 2 1 0 5 6 7 8 9

2. Assumir que o conteúdo conhecido da memória é:

Endereço	Conteúdo
0x7010	0x05
0x7008	0x01
0x7000	0x03
0x6FF8	0x00
0x6FF0	0x02

0x0E = 14

O conteúdo dos registos é: `X0=0x0E`, `X1=0x07` e `SP=0x7000`. Mostrar as alterações que ocorrem no conteúdo da memória e dos registos após cada uma das instruções da sequência seguinte:

```
str   X0, [SP, #-16]!
ldr   X2, [SP], #16
stp   X1, X0, [SP, #-16]!
```

SP = 0x6FF0
X2 = 0x03 | SP = 0x7000
0x6FF0 = 0x07 | 0x6FF8 = 0x0E

3. Considerar um programa constituído pela função `main()` e pelas sub-rotinas `sA`, `sB` e `sC`. A função `main()` chama `sA`, esta chama `sB` e esta última chama `sC`.

Tendo em consideração a convenção de chamada de sub-rotinas e as regras de preservação do conteúdo de registos, indicar quais os registos que devem ser preservados, e em que condições, pela:

a) sub-rotina `sA`. Preservar registos x19-x28, x29 (FP), x30 (LR) Guardar apenas x29 e x30, porque os outros fazem automaticamente

b) sub-rotina `sB`. Preservar registos de main e sA

c) sub-rotina `sC`. Preservar registos de main, sA e sB

4. Considerar o seguinte programa formado por dois ficheiros.

```
// ======= prog.c =======

extern long int POLI(long int
    x);                          64 bits

int main(void)
{
  long int r;
  // ...
  r = POLI(7);
  printf("Resultado = %d\n",
     r);
  return EXIT_SUCCESS;
}
```

```
// ======= subrot.s =======   Preservar x29, x30, x20

POLI: ...  stp X29, x30, [SP, -32]!   str X20, [SP, 16]   mov X29, SP
                                  // ***
   mov   X10, X0  // <1>          X10 = 7
   bl    QUAD                    X0 = 64
   mov   X20, X0                 X20 = 64
   mov   X0, 3                   X0 = 3
   mul   X10, X10, X0            X10 = 7 x 3 = 21
   add   X0, X20, X10            x0 = 64 + 21 = 85
   add   X0, X0, 1               x0 = 85 + 1 = 86
   ...          // ***           ldr X20, [SP, 16]
   ret   // <2>                  ldp X29, X30, [SP], 32

QUAD:
   mul   X0, X0, X0
   ret
```

a) Analise o programa e descreva o que calcula a sub-rotina `POLI`. Indique o que aparecerá no monitor após a execução. 86 POLI(X0) = X0^2 + 3*X0 + 1

b) Completar o código nos locais assinalados por ***.

c) Indique o conteúdo da pilha do sistema imediatamente antes da execução das instruções assinaladas com <1> e <2>. <1> tamanho 32 bytes com x29 | x30 | x20 | 0x0 <2>

5. Utilizando a pilha, escrever uma sib-rotina que:

a) imprime uma cadeia de carateres por ordem inversa.

b) verifica se uma sequência de carateres tem parêntesis curvos, parêntesis retos e chavetas corretamente emparelhadas.

6. Escrever fragmentos de código *assembly* `AArch64` que implementem o seguinte código em 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;
```

7. Escrever um programa para calcular:

a) o valor da expressão $\frac{(A-B)\times C}{D+A-3}$, assumindo valores com precisão simples.

b) o valor da área de um círculo dado o respetivo raio (considerar $\pi \approx 3.141\,592\,653$).

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

8. Considere o polinómio $p(x) = 1.5\,x^3 - 12.5\,x + 7$. Escreva a sub-rotina `calc_poly_tab` que calcula o polinómio para valores de x pertencentes a $\{0; 0.1; 0.2; \cdots; 9.9; 10\}$ (ao todo são 101 valores). O

protótipo desta sub-rotina em C é:

```
void calc_poly_tab(float *tab);
```

em que `tab` é o vetor a ser preenchido com os valores $p(0), p(0.1), \cdots, p(9.9)$ e $p(10)$.

9. 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 uma sub-rotina que calcula, para um dado x, o valor de um polinómio definido pelos seus n coeficientes $a_0, a_1, \ldots, a_{n-1}$ contidos no vetor `coefs`. Assumir que para executar esta sub-rotina é chamada a função em C com o protótipo

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

10. 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:

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

Apresentar o código da sub-rotina que calcula o produto interno de X e Y. Considerar o seguinte protótipo da função a chamar em C para executar a sub-rotina:

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

11. Considerar um vetor V com n valores do tipo `float`. Escrever uma sub-rotina 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:

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

12. 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)`.

13. 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 $a_1 = 0.278\,393$, $a_2 = 0.230\,389$, $a_3 = 0.000\,972$ e $a_4 = 0.078\,108$.

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

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

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

a) Escrever a sub-rotina `cosseno` que calcula o cosseno de um valor real expresso em radianos (assumir a declaração `double cosseno(double x)`), usando a seguinte variante da fórmula de Taylor:

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

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 $y = 100 + 50 \times \cos(x)$ com x em graus. Considerar o protótipo `double func(double graus)`.

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

Nos exercícios seguintes assumir por omissão que o número de elementos dos vetores a processar é múltiplo de 4.

15. 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)
```

16. 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)
```

17. 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);
```

18. 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);
```

19. 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. O protótipo desta sub-rotina em C é:

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

20. 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.

21. 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);
```

22. Considerar a seguinte função escrita em C.

```
#include <math.h>
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.

23. 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` $= \{a_1, b_1, a_2, b_2, \ldots, a_n, b_n\}$ 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. O protótipo desta sub-rotina em C é:

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

Fim do enunciado