Linguagem C / Assembly

1. Escreva uma função em C que calcule a raiz quadrada 'x' de uma variável 'S' do tipo float**, utilizando o seguinte algoritmo: após 'n+1' iterações, a raiz quadrada de 'S' é dada por**

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{S}{x_n}\right).$$

2. Escreva uma função em MIPS-light que calcule 'x' elevado à 'n'-ésima potência, onde 'x', 'n' e o valor de saída são inteiros de 32 bits.

'x' e 'n' deverão ser passados através dos registradores $a0 e $a1, respectivamente, e a saída deverá ser fornecida no registrador $v0. Leve em consideração que a função será utilizada em um código maior, portanto utilize adequadamente os registradores $s0-$s7 e $t0-$t9.

3. Complete o código em C abaixo, COM NO MÁXIMO 8 CARACTERES DE PONTO E VÍRGULA, para salvar na memória os seguintes vetores:

- A = 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368 75025 121393 196418 317811 514229 832040 1346269
- B = 1 2 4 8 16 32 64 128 256 512 1024 2048 4096 8192 16384 32768 65536 131072 262144 524288 1048576 2097152 4194304 8388608 16777216 33554432 67108864 134217728 268435456 536870912

```
#include <stdio.h>
#define L 30
int main(int argc, char **argv)
{
        int A[L], B[L];
        /* INSIRA O CÓDIGO AQUI, UTILIZANDO NO MÁXIMO
        8 CARACTERES DE PONTO E VÍRGULA */
        return 0;
}
```

4. "Traduza" o código da questão anterior de C para MIPS-light.

5. Escreva uma função em C que indica se um vetor é palíndromo. Por exemplo:

[1 2 3 2 1] e [0 10 20 20 10 0] são palíndromos.

[5 4 3 2 1] e [1 2 3 2] não são.

O protótipo da função é int Palindromo(int vetor[ ], int tamanho)**;**

6. Escreva uma função em C que calcula o superfatorial. Por exemplo:

$$\mathrm{sf}(4) = 1! \times 2! \times 3! \times 4! = 288.$$

O protótipo da função é unsigned long long SuperFatorial(unsigned long long n);

7. Escreva uma sub-rotina em MIPS-light que calcula o resto da divisão de 'a' por 'b', onde 'a', 'b' e o valor de saída são inteiros de 32 bits. 'a' e 'b' deverão ser fornecidos através dos registradores $a0 e $a1, respectivamente, e a saída deverá ser fornecida através do registrador $v0. Leve em consideração que a função será utilizada em um código maior, portanto utilize adequadamente os registradores $s0-$s7 e $t0-$t9.

8. Escreva uma sub-rotina em MIPS-light que indica a primalidade de uma variável inteira de 32 bits. Se o número for primo, retorne o valor 1; caso contrário, retorne o valor 0. A variável deverá ser passada através do registrador $a0, e a saída deverá ser fornecida no registrador $v0. Leve em consideração que a função será utilizada em um código maior, portanto utilize adequadamente os registradores $s0-$s7 e $t0-$t9.
Se quiser, aproveite a sub-rotina da questão anterior, chamando-a diretamente.

9. Escreva uma função em C que calcula o duplo fatorial de n, representado por n!!. **Se n for ímpar,** n!! = 1*3*5*...*n**, e se n for par,** n!! = 2*4*6*...*n**. Por exemplo,** 9!! = 1*3*5*7*9 = 945 **e** 10!! = 2*4*6*8*10 = 3840**. Além disso,** 0!! = 1!! = 1**.**
O protótipo da função é unsigned long long DuploFatorial(unsigned long long n);

10. Escreva uma função em C que calcula a função exponencial da seguinte forma:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$

Considere o cálculo até o termo n = 10. O protótipo da função é double ExpTaylor(double n);

11. Escreva uma sub-rotina em MIPS-light que indica se um vetor esta ordenado de forma decrescente. Por exemplo:

[5 4 3 2 1] e [90 23 20 10] estão ordenados de forma decrescente.

[1 2 3 4 5] e [1 2 3 2] não estão.

O primeiro endereço do vetor deverá ser passado através do registrador $a0, e o tamanho do vetor deverá ser passado pelo registrador $a1. A saída deverá ser fornecida no registrador $v0, valendo 1 quando o vetor estiver ordenado de forma decrescente, e valendo 0 em caso contrário. Leve em consideração que a função será utilizada em um código maior, portanto utilize adequadamente os registradores $s0-$s7 e $t0-$t9.

12. Escreva uma sub-rotina em MIPS-light que calcula o produto interno de dois

vetores, 'a' e 'b':

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n = \mathbf{a}\,\mathbf{b}^T$$

O primeiro endereço do vetor 'a' deverá ser passado através do registrador $a0, o primeiro endereço do vetor 'b' deverá ser passado através do registrador $a0, e o tamanho do vetor deverá ser passado pelo registrador $a2. A saída deverá ser fornecida no registrador $v0. Leve em consideração que a função será utilizada em um código maior, portanto utilize adequadamente os registradores $s0-$s7 e $t0-$t9.

13. Considere o código a seguir.

```
#include <stdio.h>
#define L 12
void Escreve_Vetor(int A[L], char texto[])
{
      int i;
      printf("%s",texto);
      for(i=0; i<L; i++) printf("%5d ", A[i]);
      printf("\n");
}
int main(int argc, char **argv)
{
      int i, A[L], B[L], C[L], prod_int[3];
```

Preencha este espaço com código para obter a saída a seguir. Não use mais do que 15 caracteres de ponto-e-vírgula.

```
      Escreve_Vetor(A, "A = ");
      Escreve_Vetor(B, "B = ");
      Escreve_Vetor(C, "C = ");
      printf("<A,B> = %d\n", prod_int[0]);
      printf("<B,C> = %d\n", prod_int[1]);
      printf("<A,C> = %d\n", prod_int[2]);
      return 0;
}
```

SAÍDA:

```
A =    1   2   3   5   8   13   21   34   55   89   144   233
B =    1   2   4   8   16   32   64   128   256   512   1024   2048
C =    1   4   9   16   25   36   49   64   81   100   121   144
<A,B> = 690585
<B,C> = 503805
<A,C> = 68320
```

14. Considere o código a seguir.

```
#include <stdio.h>
#define L 6
#define H 6
void Escreve_Matriz(int A[H][L], char texto[])
{
      int i, j;
      printf("%s\n",texto);
      for(j=0; j<H; j++)
      {
            for(i=0; i<L; i++) printf("%5d ", A[j][i]);
            printf("\n");
      }
      printf("\n");
}
int main(int argc, char **argv)
{
      int i, j, A[H][L], B[H][L], C[H][L];
```

Preencha este espaço com código para obter a saída a seguir. Não use mais do que 10 caracteres de ponto-e-vírgula.

```
      Escreve_Matriz(A, "A = ");
      Escreve_Matriz(B, "B = ");
      Escreve_Matriz(C, "C = A+B = ");
      return 0;
}
```

SAÍDA:

```
A =
   0   1   2   3   4   5
   1   2   3   4   5   6
   2   3   4   5   6   7
   3   4   5   6   7   8
   4   5   6   7   8   9
   5   6   7   8   9  10

B =
   0   1   2   3   4   5
  -1   0   1   2   3   4
  -2  -1   0   1   2   3
  -3  -2  -1   0   1   2
  -4  -3  -2  -1   0   1
  -5  -4  -3  -2  -1   0

C = A+B =
   0   2   4   6   8  10
   0   2   4   6   8  10
   0   2   4   6   8  10
   0   2   4   6   8  10
   0   2   4   6   8  10
   0   2   4   6   8  10
```

15. Considere o código a seguir.

```
#include <stdio.h>
#define L 10
void Escreve_Vetor(float A[L], char texto[])
{
      int i;
      printf("%s",texto);
      for(i=0;i<L;i++) printf("%3.3f ",A[i]);
      printf("\n");
}
float Potencia(float x, unsigned int n)
{
      Preencha este espaço com código para calcular
      "x" elevado à n-ésima potência. Use somente 2
      caracteres de ponto-e-vírgula.
}
int main(int argc, char **argv)
{
      float x[L], y[L], coefs[]={-0.1, -1.67, 0.91, 3.2, 0.5};
      int i;
      for(i=0; i<L; i++)
      {
```

Preencha este espaço com código para fazer:

x = {0.0, 0.1, 0.2 ...}
yi = 0,5*xi^4 + 3,2*xi^3 + 0,91*xi^2 -1,67*xi -0,1.

onde yi e xi correspondem a cada elemento dos vetores x e y, respectivamente. No seu código, use somente 2 caracteres de ponto-e-vírgula.

```
    }
    Escreve_Vetor(x,"x = ");
    Escreve_Vetor(y,"y = ");
    return 0;
}
```

16. Considere o código a seguir.

```
#include <stdio.h>
#include <string.h>
struct login
{
```

Inclua as variáveis necessárias para armazenar informações de login de cada usuário.

```
};
void Preenche_Login(struct login *a, char nome[], unsigned char dia,
unsigned char mes, unsigned char ano)
{
```

Crie código para que esta função preencha os dados de login do usuário, de acordo com os parâmetros de entrada, e mostre estes dados na tela. A senha numérica será criada da seguinte maneira:

- O primeiro dígito é dado pelo resto da divisão do primeiro caractere do nome pelo dia de nascimento.
- O segundo dígito é dado pelo resto da divisão do segundo caractere do nome pelo dia de nascimento.
- O terceiro dígito é dado pelo resto da divisão do terceiro caractere do nome pelo mês de nascimento.
- O quarto dígito é dado pelo resto da divisão do quarto caractere do nome pelo mês de nascimento.
- O quinto dígito é dado pelo resto da divisão do quinto caractere do nome pelo ano de nascimento.
- O sexto dígito é dado pelo resto da divisão do sexto caractere do nome pelo ano de nascimento.
- Como os dígitos acima só podem estar entre 0 e 9, cada dígito será dado pelo resto da divisão dos resultados acima por 10.

```
}
int main()
{
      struct login usuarios[5];
      Preenche_Login(&(usuarios[0]), "Marcos",      22,  3, 70);
      Preenche_Login(&(usuarios[1]), "Joao Vitor", 10, 12, 74);
      Preenche_Login(&(usuarios[2]), "Marcelo",      6,  9, 85);
      Preenche_Login(&(usuarios[3]), "Gabriel",     10, 10, 93);
      Preenche_Login(&(usuarios[4]), "Marcos",      30,  5, 60);
      return 1;
}
```

SAÍDA:

```
### Marcos, 22/3/70
--- Senha = 190015
### Joao Vitor,
10/12/74
--- Senha = 411322
### Marcelo, 6/9/85
--- Senha = 516063
### Gabriel, 10/10/93
--- Senha = 178428
### Marcos, 30/5/60
--- Senha = 774415
```

7. Qual é a senha para Gilberto, nascido em 20/07/1980?

18. Dada uma matriz float A[6][8]**, crie um código para calcular a média de cada coluna, de cada linha, e de toda a matriz.**

19. O código abaixo deveria calcular a média da matriz A. Encontre os erros no código, que impedem o cálculo e a visualização do resultado correto.

```
#include <stdio.h>
#include <string.h>
int main()
{
      float A[4][3];
      int i, j, media;

      A[0][0] = A[0][1] = A[1][0] = 21.067;
      A[2][0] = A[0][2] = A[2][1] = 30.09;
      A[2][2] = A[3][1] = A[3][2] = 10.3;
      A[1][1] = A[1][2] = A[3][0] = 57.28;

      for(i=0;i<4;i++)
            for(j<0;j<3;i++)
                  media = media+A[j][i];
      media = media/4*3;
```

```
        printf("Media = %d",media);
        return 1;
}
```

INSTRUÇÕES MIPS-LIGHT: http://www.stanford.edu/class/ee282h/projects/info/isa.html

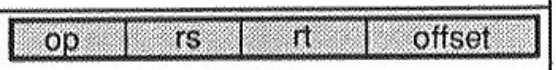
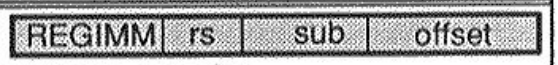
Instruction	Format and Description	op	base	rt	offset
Load Word	*LW rt,offset(base)* Sign-extend 16-bit *offset* and add to contents of register *base* to form address. Load contents of addressed word into register *rt*.				
Store Word	*SW rt,offset(base)* Sign-extend 16-bit *offset* and add to contents of register *base* to form address. Store the contents of register *rt* at addressed location.				

Instruction	Format and Description 
ADD Immediate	*ADDI rt,rs,immediate* Add 16-bit sign-extended *immediate* to register *rs* and place the 32-bit result in register *rt*. Trap on 2's-complement overflow.
ADD Immediate Unsigned	*ADDIU rt,rs,immediate* Add 16-bit sign-extended *immediate* to register *rs* and place the 32-bit result in register *rt*. Do not trap on overflow.
Set on Less Than Immediate	*SLTI rt,rs,immediate* Compare 16-bit sign-extended *immediate* with register *rs* as signed 32-bit integers. Result = 1 if *rs* is less than *immediate*; otherwise result = 0. Place result in register *rt*.
Set on Less Than Immediate Unsigned	*SLTIU rt,rs,immediate* Compare 16-bit sign-extended *immediate* with register *rs* as unsigned 32-bit integers. Result = 1 if *rs* is less than *immediate*; otherwise result = 0. Place result in register *rt*.
AND Immediate	*ANDI rt,rs,immediate* Zero-extend 16-bit *immediate*, AND with contents of register *rs* and place the result in register *rt*.
OR Immediate	*ORI rt,rs,immediate* Zero-extend 16-bit *immediate*, OR with contents of register *rs* and place the result in register *rt*.
Exclusive OR Immediate	*XORI rt,rs,immediate* Zero-extend 16-bit *immediate*, exclusive OR with contents of register *rs* and place the result in register *rt*.
Load Upper Immediate	*LUI rt,immediate* Shift 16-bit *immediate* left 16 bits. Set least significant 16 bits of word to zeros. Store the result in register *rt*.

Instruction	Format and Description	op	rs	rt	rd	sa	function
Add	*ADD rd,rs,rt* Add contents of registers *rs* and *rt* and place the 32-bit result in register *rd*. Trap on 2's-complement overflow.						
Add Unsigned	*ADDU rd,rs,rt* Add contents of registers *rs* and *rt* and place the 32-bit result in register *rd*. Do not trap on overflow.						
Subtract	*SUB rd,rs,rt* Subtract contents of registers *rt* from *rs* and place the 32-bit result in register *rd*. Trap on 2's-complement overflow.						
Subtract Unsigned	*SUBU rd,rs,rt* Subtract contents of registers *rt* from *rs* and place the 32-bit result in register *rd*. Do not trap on overflow.						
Set on Less Than	*SLT rd,rs,rt* Compare contents of register *rt* to register *rs* as signed 32-bit integers. Result = 1 if *rs* is less than *rt*; otherwise result = 0.						
Set on Less Than Unsigned	*SLTU rd,rs,rt* Compare contents of register *rt* to register *rs* as unsigned 32-bit integers. Result = 1 if *rs* is less than *rt*; otherwise result = 0.						
AND	*AND rd,rs,rt* Bitwise AND the contents of registers *rs* and *rt*, and place the result in register *rd*.						
OR	*OR rd,rs,rt* Bitwise OR the contents of registers *rs* and *rt*, and place the result in register *rd*.						
Exclusive OR	*XOR rd,rs,rt* Bitwise exclusive OR the contents of registers *rs* and *rt*, and place the result in register *rd*.						
NOR	*NOR rd,rs,rt* Bitwise NOR the contents of registers *rs* and *rt*, and place the result in register *rd*.						

Instruction	Format and Description	op	target
Jump	*J target* Shift the 26-bit *target* address left two bits, combine with high order four bits of the PC, and jump to the address with a 1-instruction delay.		
Jump And Link	*JAL target* Shift the 26-bit *target* address left two bits, combine with high order four bits of the PC, and jump to the address with a 1-instruction delay. Place the address of the instruction following the delay slot in *r31* (*Link* register).		

Instruction	Format and Description	op	rs	rt	rd	sa	function
Jump Register	*JR rs* Jump to the address contained in register *rs*, with a 1-instruction delay.						
Jump And Link Register	*JALR rs, rd* Jump to the address contained in register *rs*, with a 1-instruction delay. Place the address of the instruction following the delay slot in register *rd*.						

Instruction	Format and Description
Branch on Equal	*BEQ rs,rt,offset*  Branch to target address if register *rs* is equal to register *rt*.
Branch on Not Equal	*BNE rs,rt,offset* Branch to target address if register *rs* is not equal to register *rt*.
Branch on Less than or Equal Zero	*BLEZ rs,offset* Branch to target address if register *rs* is less than or equal to zero.
Branch on Greater Than Zero	*BGTZ rs,offset* Branch to target address if register *rs* is greater than zero.
Branch on Less Than Zero	*BLTZ rs,offset*  Branch to target address if register *rs* is less than zero.
Branch on Greater than or Equal Zero	*BGEZ rs,offset* Branch to target address if register *rs* is greater than or equal to zero.
Branch on Less Than Zero And Link	*BLTZAL rs,offset* Place address of instruction following the delay slot in register *r31* (Link register). Branch to target address if register *rs* is less than zero.
Branch on Greater than or Equal Zero And Link	*BGEZAL rs,offset* Place address of instruction following the delay slot in register *r31* (Link register). Branch to target address if register *rs* is greater than or equal to zero.

Instruction	Format and Description 
Shift Left Logical	*SLL rd,rt,sa* Shift the contents of register *rt* left by *sa* bits, inserting zeros into the low order bits. Place the 32-bit result in register *rd*.
Shift Right Logical	*SRL rd,rt,sa* Shift the contents of register *rt* right by *sa* bits, inserting zeros into the high order bits. Place the 32-bit result in register *rd*.
Shift Right Arithmetic	*SRA rd,rt,sa* Shift the contents of register *rt* right by *sa* bits, sign-extending the high order bits. Place the 32-bit result in register *rd*.
Shift Left Logical Variable	*SLLV rd,rt,rs* Shift the contents of register *rt* left. The low order 5 bits of register *rs* specify the number of bits to shift left; insert zeros into the low order bits of *rt* and place the 32-bit result in register *rd*.
Shift Right Logical Variable	*SRLV rd,rt,rs* Shift the contents of register *rt* right. The low order 5 bits of register *rs* specify the number of bits to shift right; insert zeros into the high order bits of *rt* and place the 32-bit result in register *rd*.
Shift Right Arithmetic Variable	*SRAV rd,rt,rs* Shift the contents of register *rt* right. The low order 5 bits of register *rs* specify the number of bits to shift right; sign-extend the high order bits of *rt* and place the 32-bit result in register *rd*.