Disciplina: Eletrônica Embarcada **Código:** 120871 **Turma:** A
Professor: Diogo Caetano Garcia
Aluno/Matrícula: Fábio Barbosa Pinto – 11/0116356

Questionário: 06_07_Questoes_Assembly

Para cada questão, escreva funções em C e/ou sub-rotinas na linguagem Assembly do MSP430. Reaproveite funções e sub-rotinas de uma questão em outra, se assim desejar. Leve em consideração que as sub-rotinas são utilizadas em um código maior, portanto utilize adequadamente os registradores R4 a R11. As instruções da linguagem Assembly do MSP430 se encontram ao final deste texto.

1)
a) 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) = (x(n) + S/x(n))/2
```

O protótipo da função é:

```
unsigned int Raiz_Quadrada(unsigned int S);
```

```
int Divisao(int dividendo,int divisor)
{
	if(dividendo >= divisor)
	{
		dividendo -= divisor;
		return (1 + Divisao(dividendo,divisor));
	}
	else
		return 0;
}

int Raiz_Quadrada(float s)
{
	float x=0;
	do
	{
		if(x==0)
		{
			x=Divisao(s,2);
			x = Divisao((x + Divisao(s,x)),2);
		}
		else
		{
			x = Divisao((x + Divisao(s,x)),2);
		}
```

```
        }
        while((x - Divisao((x + Divisao(s,x)),2)) > 1);

        return x;
}
```

b) Escreva a sub-rotina equivalente na linguagem Assembly do MSP430. A variável 'S' é fornecida pelo registrador R15, e a raiz quadrada de 'S' (ou seja, a variável 'x') é fornecida pelo registrador R15 também.

Calcula a divisão de inteiro da forma R15/R14, sem sinal:

```
divisao_unsigned:
        MOV #1,R13
        CMP R14,R15 ; R15 = dividendo, R14 = divisor
        JGE divisao_subtract ; if (R15 >= 14) go to Divisao_subtract
        CLR.W R15 ; return 0 caso R15 seja menor que R14
        RET
divisao_subtract:
        SUB.W R14,R15 ; R15 = R15 - R14
        PUSH.W R13 ; guarde 1 na pilha
        CALL #divisao_unsigned ; 303
        POP.W R14 ; recupere 1 na pilha
        ADD.W R14,R15 ; return 1 + divisao(dividendo,divisor)
        CLR.W R14; R14 = 0
        CLR.W R13; R13 = 0
        RET
```

Calcula a raiz quadrada de R15, um inteiro:

```
raiz_quadrada:
        MOV.W #0,R14 ; x = R14 = 0;
iteracao:
        TST R14; R14 = 0 ?
        JNE else_iteracao; se R14 != 0, vá pra else
        PUSH R15 ; guardar R15 = s na pilha
        MOV.W #2,R14 ; Nesse caso, ainda nao tem problema sobrescrever R14 porque ele é zero nesse momento
        CALL #divisao_unsigned ; Faça Divisao(s,2)
        MOV.W R15,R14 ; x =  Divisao(s,2)
        POP R15 ; recupere S para R15, R15 = s
        CALL #realizar_iteracao ; faça uma iteracao x = Divisao((x + Divisao(s,x)),2);
        JMP condicao_while ; vá para a condicao do laço

realizar_iteracao:
        PUSH R15; guarde S na pilha
        PUSH R14; guarde x na pilha
        CALL #divisao_unsigned ; faça Divisao(s,x)
```

```
        MOV.W R15,R13 ; R13 <= Divisao(s,x)
        POP R14 ; recupere x da pilha
        ADD.W R14,R13 ; faça R13 <= Divisao(s,x) + x
        MOV.W R13,R15 ; faça R15 <= R13 <= Divisao(s,x) + x. Esta operacao foi realizada para fazer
Divisao((x + Divisao(s,x)),2)
        MOV #2,R14 ; R14 <= 2
        CALL #divisao_unsigned
        MOV R15,R14 ; R14 <= x <= Divisao((x + Divisao(s,x)),2)
        POP R15 ; recupere s da pilha
        RET

else_iteracao:
        CALL #realizar_iteracao

condicao_while:
        PUSH R14 ; guarde x(n) na pilha
        PUSH R14 ; guarde x(n) na pilha
        CALL #realizar_iteracao
        MOV.W R14,R13 ; R13 <= R14 <= Divisao(x + Divisao(s,x)),2)
        POP R14 ; recupere x(n) da pilha
        SUB.W R13,R14 ; faça R14 = x(n) - Divisao(x + Divisao(s,x)),2)
        MOV.W R14,R13 ; faça R13 = R14
        POP R14 ; recupere x(n) da pilha
        CMP #1, R13 ; comparar R13 com 1
        JGE iteracao ; R13 >= 1, vá para iteracao

fim_raiz_quadrada:
        MOV.W R14,R15 ; R15 <= R14
        CLR.W R14
        CLR.W R13
        RET
```

2)

a) Escreva uma função em C que calcule 'x' elevado à 'N'-ésima potência, seguindo o seguinte protótipo:

```
        int Potencia(int x, int N);
```

```
int MULT_signed(int a, int b)
{
        if(a<0 && b<0)
        {
                a = -a;
                b = -b;
                return MULT_unsigned(a,b);
```

```
        }
        else if (a<0 && b>0)
        {
                a = -a;
                return -(MULT_unsigned(a,b));
        }
        else if (a>0 && b<0)
        {
                b = -b;
                return -(MULT_unsigned(a,b));
        }
        else
        {
                return MULT_unsigned(a,b);
        }

}

int MULT_unsigned(unsigned int a, unsigned int b)
{
if(b==0) return 0;
else
return a+MULT_unsigned(a, (b-1));
}

int Potencia(int x, int N)
{
        if(N==0) return 1;
        else return MULT_signed(x,Potencia(x, (N-1)));
}
```

b) Escreva a sub-rotina equivalente na linguagem Assembly do MSP430. 'x' e 'n' são fornecidos através dos registradores R15 e R14, respectivamente, e a saída deverá ser fornecida no registrador R15.

```
#include "msp430.h"                    ; #define controlled include file

      NAME    main                     ; module name

      PUBLIC  main                     ; make the main label vissible
                                       ; outside this module
      ORG     0FFFEh
      DC16    init                     ; set reset vector to 'init' label

      RSEG    CSTACK                   ; pre-declaration of segment
```

```
        RSEG    CODE                    ; place program in 'CODE' segment

init:   MOV     #SFE(CSTACK), SP        ; set up stack

main:   NOP                             ; main program
        MOV.W   #WDTPW+WDTHOLD,&WDTCTL  ; Stop watchdog timer
        MOV.W   #0xFFFB,R15             ; R15 <= -5
        MOV.W   #3,R14                  ; R14 <= 3
        CALL    #Potencia               ; Faça R15^(R14)
        JMP $                           ; jump to current location '$'
                                        ; (endless loop)
MULT_signed:

        PUSH    R15
        RLA     R15                                     ; c recebe o sinal de a
        JNC     Third_else_if_MULT_signed                       ; Pule se a>0
        POP     R15
        PUSH    R14
        RLA     R14                                      ; c recebe o sinal de b
        JNC     Second_else_if_MULT_signed                      ; Pule se b>0
        POP     R14
        INV     R15                                     ; realizando o complemento de 2
        INC     R15                                     ; nas quatro instruções abaixo
        INV     R14
        INC     R14
        CALL    #MULT_unsigned
        CLR     R14
        RET                                     ; return MULT_unsigned(a,b)

Second_else_if_MULT_signed:                             ; a<0 mas b>0
        POP     R14
        INV     R15                                     ; realizando o comp.2 para a
        INC     R15
        CALL    #MULT_unsigned
        INV     R15                                     ; realizando o comp. de 2 para
        INC     R15                                     ; retornar -(MULT_unsigned(a,b))
        CLR     R14
        RET

Third_else_if_MULT_signed:                              ; a>0 mas b<0 ?
        POP     R15
        PUSH    R14
        RLA     R14                                      ; c recebe o sinal de b
        JNC     else__MULT_signed                       ; Pule se b>0
        POP     R14
```

```
        INV     R14                             ; realizando o complemento de 2
        INC     R14                             ; para b
        CALL    #MULT_unsigned
        INV     R15                             ; realizando o comp. de 2 para
        INC     R15                             ; retornar -(MULT_unsigned(a,b))
        CLR     R14
        RET

else__MULT_signed:
        POP     R14
        CALL    #MULT_unsigned                          ; return MULT_unsigned(a,b)
        CLR     R14
        RET

MULT_unsigned:

        TST     R14                 ; b==0 ?
        JNZ     MULT_unsigned_else      ; Se b não é zero, vá para else
        CLR.W   R15                 ; return 0
        RET

MULT_unsigned_else:

        PUSH    R15                 ; guarde a na pilha
        DEC.W   R14                 ; b--
        CALL    #MULT_unsigned          ; calcule a*(b-1)
        POP.W   R14                 ; recupere a da pilha
        ADD.W   R14,R15                 ; return a + a*(b-1)
        RET

Potencia:
        TST     R14                 ; N==0?
        JNZ     Potencia_else           ; se N não é zero, vá pra else
        MOV.W   #1,R15                  ; return 1
        RET

Potencia_else:
        PUSH    R15                 ; guarde x na pilha
        DEC.W   R14                 ; b--
        CALL    #Potencia               ; calcule x*x^(n-1) 300
        POP.W   R14                 ; recupere x da pilha
        CALL    #MULT_signed            ; Faça R15*R14
        RET

        END
```

3. Escreva uma sub-rotina na linguagem Assembly do MSP430 que calcula a divisão de 'a' por 'b', onde 'a', 'b' e o valor de saída são inteiros de 16 bits. 'a' e 'b' são fornecidos através dos registradores R15 e R14, respectivamente, e a saída deverá ser fornecida através do registrador R15.

```
DIV_signed:

        PUSH    R15
        RLA     R15                             ; c recebe o sinal de a
        JNC     Third_else_if_DIV_signed                ; Pule se a>0
        POP     R15
        PUSH    R14
        RLA     R14                              ; c recebe o sinal de b
        JNC     Second_else_if_DIV_signed               ; Pule se b>0
        POP     R14
        INV     R15                             ; realizando o complemento de 2
        INC     R15                             ; nas quatro instruções abaixo
        INV     R14
        INC     R14
        CALL    #divisao_unsigned
        CLR     R14
        RET                                   ; return divisao_unsigned(a,b)

Second_else_if_DIV_signed:                      ; a<0 mas b>0
        POP     R14
        INV     R15                             ; realizando o comp.2 para a
        INC     R15
        CALL    #divisao_unsigned
        INV     R15                             ; realizando o comp. de 2 para
        INC     R15                             ; retornar -(divisao_unsigned(a,b))
        CLR     R14
        RET

Third_else_if_DIV_signed:                       ; a>0 mas b<0 ?
        POP     R15
        PUSH    R14
        RLA     R14                              ; c recebe o sinal de b
        JNC     else_DIV_signed                     ; Pule se b>0
        POP     R14
        INV     R14                             ; realizando o complemento de 2
        INC     R14                             ; para b
        CALL    #divisao_unsigned
        INV     R15                             ; realizando o comp. de 2 para
        INC     R15                             ; retornar -(divisao_unsigned(a,b))
```

```
        CLR     R14
        RET

else_DIV_signed:
        POP     R14
        CALL    #divisao_unsigned                       ; return divisao_unsigned(a,b)
        CLR     R14
        RET
```

Calcula a divisão de inteiro da forma R15/R14, sem sinal:

```
divisao_unsigned:
        MOV #1,R13
        CMP R14,R15             ; R15 = dividendo, R14 = divisor
        JGE divisao_subtract    ; if (R15 >= 14) go to Divisao_subtract
        CLR.W R15               ; return 0 caso R15 seja menor que R14
        RET

divisao_subtract:
        SUB.W R14,R15           ; R15 = R15 - R14
        PUSH.W R13              ; guarde 1 na pilha
        CALL #divisao_unsigned  ; 303
        POP.W R14               ; recupere 1 na pilha
        ADD.W R14,R15           ; return 1 + divisao(dividendo,divisor)
        CLR.W R14               ; R14 = 0
        CLR.W R13               ; R13 = 0
        RET
```

4. Escreva uma sub-rotina na linguagem Assembly do MSP430 que calcula o resto da divisão de 'a' por 'b', onde 'a', 'b' e o valor de saída são inteiros de 16 bits. 'a' e 'b' são fornecidos através dos registradores R15 e R14, respectivamente, e a saída deverá ser fornecida através do registrador R15.

```
int Remainder (int dividend,int divisor)
{
        while (dividend >= divisor)
        {
                dividend -= divisor;
        }

        return dividend;
}
```

Em Assembly para msp 430:

```
remainder:
        CMP R14,R15
```

```
        JL remainder_finish          ; R15 < R14 vá para finish
        SUB.W R14,R15                 ; R15 <= R15 - R14
        JMP remainder

remainder_finish:
        RET
```

5)

a) Escreva uma função em C que indica a primalidade de uma variável inteira sem sinal, retornando o valor 1 se o número for primo, e 0, caso contrário. Siga o seguinte protótipo:

```
int Primalidade(unsigned int x);
```

```
int Remainder (int dividend,int divisor)
{
        while (dividend >= divisor)
        {
                dividend -= divisor;
        }

        return dividend;
}

int Primalidade (int a)
{
        int i=3;
        /*Se a for 1 ele não é primo.*/
        if (a==1)
        {
                return 0;
        }
        else if (a==2)
        {
                return 1;
        }
        /*Se a divisão de a por 2 der zero o número é PAR e não é primo.*/
        else if (Remainder(a,2)==0)
        {
                return 0;
        }
        /*Saia verificando a com todos os outros números. Se a for divisível por i ele não é primo. Vá verificando até que i = a - 1.
        se i chegar a ser igual a i = a - 1, o número é primo.
        */
        else
        {
```

```
                while(Remainder(a,i)!=0 && i<a)
                {
                        i+=2;
                }
                /*A condição a seguir verifica porquê o laço acabou.*/
                if(i==a)
                {
                        return 1;
                }
                else
                {
                        return 0;
                }
        }

}
```

b) Escreva a sub-rotina equivalente na linguagem Assembly do MSP430. A variável de entrada é fornecida pelo registrador R15, e o valor de saída também.

```
Primalidade:
        TST     R15                 ; R15 == 0?
        JEQ     Nao_primo
        CMP     #1,R15                ; R15 == 1?
        JEQ     Nao_primo
        CMP     #2,R15                ; R15 == 2?
        JEQ     E_primo
        MOV.W   #3,R14                ; R14 = i = 3
While_verificar:
        CMP     R14,R15
        JEQ     E_primo               ; E' primo se a==i
        PUSH    R14                 ; Guarde R14 na pilha
        PUSH    R15                 ; Guarde R15 na pilha
        CALL    #remainder            ; Faça a%i
        TST     R15
        JEQ     Nao_primo_2           ; R15 == a%i == 0?
        POP     R15                 ; Recupere R15
        POP     R14                 ; Recupere R14
        ADD.W   #2,R14                ; i += 2
        JMP     While_verificar
Nao_primo:
        MOV.W   #0,R15                ; return 0
        CLR.W   R14
        RET
```

```
Nao_primo_2:
        POP     R15
        POP     R15
        MOV.W   #0,R15              ; return 0
        CLR.W   R14
        RET
E_primo
        MOV.W   #1,R15              ; return 1
        CLR.W   R14
        RET

remainder:
        CMP     R14,R15
        JL      remainder_finish    ; R15 < R14 vá para finish
        SUB.W   R14,R15              ; R15 <= R15 - R14
        JMP     remainder

remainder_finish:
        RET

        END
```

6. 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);
```

```
#include<stdio.h>

unsigned long long MULT_unsigned(unsigned long long a, unsigned long long b)
{
if(b==0) return 0;
else
return a+MULT_unsigned(a, (b-1));
}

unsigned long long Remainder (unsigned long long dividend,unsigned long long divisor)
{
        while (dividend >= divisor)
        {
                dividend -= divisor;
        }

        return dividend;
```

```
}

unsigned long long Duplo_Fatorial (long long n)
{
	if(n==0 || n==1)
	{
		return 1;
	}
	/*Aqui o código verifica se o número é ímpar ou par.*/
	else if(Remainder(n,2)==0)
	{
		/*Se o número for par vai entrar aqui.*/
		/*A função vai retornar n*(n-2) até que n seja igual a zero, onde irá retornar um.*/
		if(n>0)
		{
			return (MULT_unsigned(n,Duplo_Fatorial(n-2)));
		}
		else
		{
			return 1;
		}

	}
	else
	{
		if(n>1)
		{
			return (MULT_unsigned(n,Duplo_Fatorial(n-2)));
		}
		else
		{
			return 1;
		}
	}
}
```

7)

a) Escreva uma função em C que calcula a função exponencial utilizando a série de Taylor da mesma. Considere o cálculo até o termo n = 20. O protótipo da função é:

```
double ExpTaylor(double x)
```

```
double Fatorial (long long n)
{
	if(n==0 || n==1)
```

```
	{
		return 1;
	}
	else
	{
		return (n*Fatorial(n-1));
	}
}

double Pot(double x, int n)
{
	if(n>0)
	{
		return (x*Pot(x,(n-1)));
	}
	else
		return 1;
}

double ExpTaylor(double x)
{
	int i;
	double Exp;
	for(i=0;i<=20;i++)
	{
		Exp += Pot(x,i)/Fatorial(i);
	}
	return Exp;
}
```

8) Escreva uma sub-rotina na linguagem Assembly do MSP430 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 é fornecido pelo registrador R15, e o tamanho do vetor é fornecido pelo registrador R14. A saída deverá ser fornecida no registrador R15, valendo 1 quando o vetor estiver ordenado de forma decrescente, e valendo 0 em caso contrário.

Primeiro, em C, temos:

```
int Vetor_Ordenado_Decrescente(int *p,int n)
{
	/*Vetor_Ordenado_Decrescente retorna 1 se a função for decrescente e 0 se for decrescente.
	p* é um ponteiro que aponta para a primeira posição do vetor e n é um valor inteiro com o tamanho do vetor*/
```

```
	int i,anterior=0,proximo=0;
	for (i=0;i<n;i++)
	{
		if(i==0)
		{
			anterior = p[i];
			/*A variável anterior recebe o valor de p[0] na primeira chamada.*/
		}
		else
		{
			proximo = p[i];
			/*Próximo recebe p[i] para poder comparar com a variável anterior, pois anterior
== p[i-1]*/
			if(anterior>proximo)
			{
				anterior = proximo;
			}
			else
			{
				return 0;
			}

		}
	}
	/*Se todas as posições do vetor foram visualizadas, o vetor é decrescente e a função deve
retornar 1.*/
	return 1;
}
```

Agora para MSP 430:

```
#include "msp430.h"                  ; #define controlled include file
#include <msp430g2553.h>
        NAME    main                    ; module name

        PUBLIC  main                    ; make the main label vissible
                                        ; outside this module
        ORG     0FFFEh
        DC16    init                    ; set reset vector to 'init' label

        RSEG    CSTACK                  ; pre-declaration of segment
        RSEG    CODE                    ; place program in 'CODE' segment

init:   MOV     #SFE(CSTACK), SP        ; set up stack

main:   NOP                             ; main program
```

```
      MOV.W   #WDTPW+WDTHOLD,&WDTCTL  ; Stop watchdog timer
      MOV.W   #0x0A30, R15            ; Vai começar a preencher o vetor a partir do endereço
0x0A30 da memória.
      MOV.W   #90,0(R15)              ;
      MOV.W   #80,2(R15)              ;
      MOV.W   #70,4(R15)
      MOV.W   #90,6(R15)
      MOV.W   #10,8(R15)
      MOV.W   #10,R14                 ; R14 = 10, Recebe o tamanho do vetor na forma 2*n
                                      ; em que n é o número de bytes
      CALL #Vetor_Ordenado_Decres
      JMP $                           ; jump to current location '$'
                                      ; (endless loop)
 Vetor_Ordenado_Decres:
      MOV.W   #0,R13                  ; R13 = i = 0
 For_vetor_ord_decres:
      TST     R13                     ; i > 0?
      JEQ     Fim_Iteracao_for_ord_decres
      CMP     R14,R13                 ; i < n ?
      JGE     end_for_vetor_ord_decres
      MOV.W   R13,R12                 ; R12 = i
      SUB.W   #2,R12                  ; R12 = i - 1
      PUSH    R13                     ; Guarde i na pilha
      ADD.W   R15,R13
      ADD.W   R15,R12
      CMP     0(R12),0(R13)           ; p[i] >= p[i-1]
      JGE     Nao_decrescente
      POP     R13                     ; Recupere i da pilha

 Fim_Iteracao_for_ord_decres:
      ADD     #2,R13                  ; i++
      JMP     For_vetor_ord_decres

 Nao_decrescente:
      POP     R15                     ; Pegue o valor de i da pilha
      CLR     R15                     ; return 0
      CLR     R14
      CLR     R13
      CLR     R12
      RET

 end_for_vetor_ord_decres:
      MOV.W   #1,R15                  ; return 1
      CLR     R14
      CLR     R13
```

```
        CLR     R12
        RET

        END
```

9) Escreva uma sub-rotina na linguagem Assembly do MSP430 que calcula o produto escalar de dois vetores, 'a' e 'b'. O primeiro endereço do vetor 'a' deverá ser passado através do registrador R15, o primeiro endereço do vetor 'b' deverá ser passado através do registrador R14, e o tamanho do vetor deverá ser passado pelo registrador R13. A saída deverá ser fornecida no registrador R15.

Em C:

```
int MULT_signed(int a, int b)
{
        if(a<0 && b<0)
        {
                a = -a;
                b = -b;
                return MULT_unsigned(a,b);
        }
        else if (a<0 && b>0)
        {
                a = -a;
                return -(MULT_unsigned(a,b));
        }
        else if (a>0 && b<0)
        {
                b = -b;
                return -(MULT_unsigned(a,b));
        }
        else
        {
                return MULT_unsigned(a,b);
        }

}

int MULT_unsigned(unsigned int a, unsigned int b)
{
if(b==0) return 0;
else
return a+MULT_unsigned(a, (b-1));
}

int Produto_Escalar_int(int *a,int *b, int n)
{
```

```
/*O ponteiro *a aponta para o endereço do primeiro vetor na memória e o ponteiro *b aponta para o endereço do
do segundo vetor na memória. O número n indica o tamanho do vetor. Esta função retorna um valor inteiro.
*/

        int i,soma=0;
        for(i=0;i<n;i++)
        {
                soma += MULT_signed(a[i],b[i]);
        }
        return soma;
}
```

Em Assembly:

```
#include "msp430.h"                 ; #define controlled include file

      NAME    main                  ; module name

      PUBLIC  main                  ; make the main label vissible
                                    ; outside this module
      ORG     0FFFEh
      DC16    init                  ; set reset vector to 'init' label

      RSEG    CSTACK                ; pre-declaration of segment
      RSEG    CODE                  ; place program in 'CODE' segment

init:   MOV     #SFE(CSTACK), SP        ; set up stack

main:   NOP                           ; main program
      MOV.W   #WDTPW+WDTHOLD,&WDTCTL  ; Stop watchdog timer
      MOV.W   #0x0a10,R15             ; Passa o endereço de memória 0x0a10 para R15
      MOV.W   #0x0a20,R14             ; Passa o endereço de memória 0x0a20 para R14
      MOV.W   #0xFFFF,0(R15)          ; a[0] = -1
      MOV.W   #0xFFFF,2(R15)          ; a[1] = -1
      MOV.W   #0xFFFF,4(R15)          ; a[2] = -1
      MOV.W   #1,0(R14)               ; b[0] = -1
      MOV.W   #1,2(R14)               ; b[1] = -1
      MOV.W   #1,4(R14)               ; b[2] = -1
      MOV.W   #6,R13                  ; passe o n (tamanho do vetor para a função), mas 2*n
      CALL    #Produto_Escalar        ; pois são 2 bytes para inteiro.

      JMP $                           ; jump to current location '$'
                                    ; (endless loop)
```

```
Produto_Escalar:
; R15 = a
; R14 = b
; R13 = n
; R12 = i
; R11 = soma

      CLR    R12                ; i=0
      CLR    R11                ; R11 = soma = 0
For_Produto_Escalar:
      CMP    R13,R12
      JGE    end_for_produto_escalar ;
      PUSH   R15
      PUSH   R14
      ADD.W  R12,R15            ; some o endereço de a com i
                                ; que significa a[i]
      ADD.W  R12,R14            ; some o endereço de b com i
                                ; que significa b[i]

      MOV.W  0(R15),R15          ; acesse o valor de a[i] e jogue em R15
      MOV.W  0(R14),R14          ; acesse o valor de b[i] e jogue em R14
      CALL   #MULT_signed
      ADD.W  R15,R11            ; soma += a[i]*b[i]
      POP    R15
      POP    R14
      INCD   R12                ; i++
      JMP    For_Produto_Escalar


end_for_produto_escalar:

      MOV.W  R11,R15            ; return soma
      CLR    R14
      CLR    R13
      CLR    R12
      CLR    R11
      RET

MULT_signed:

      PUSH   R15
      RLA    R15                          ; c recebe o sinal de a
      JNC    Third_else_if_MULT_signed            ; Pule se a>0
      POP    R15
      PUSH   R14
```

```
        RLA     R14                             ; c recebe o sinal de b
        JNC     Second_else_if_MULT_signed              ; Pule se b>0
        POP     R14
        INV     R15                             ; realizando o complemento de 2
        INC     R15                             ; nas quatro instruções abaixo
        INV     R14
        INC     R14
        CALL    #MULT_unsigned
        CLR     R14
        RET                                     ; return MULT_unsigned(a,b)

Second_else_if_MULT_signed:                             ; a<0 mas b>0
        POP     R14
        INV     R15                             ; realizando o comp.2 para a
        INC     R15
        CALL    #MULT_unsigned
        INV     R15                             ; realizando o comp. de 2 para
        INC     R15                             ; retornar -(MULT_unsigned(a,b))
        CLR     R14
        RET

Third_else_if_MULT_signed:                              ; a>0 mas b<0 ?
        POP     R15
        PUSH    R14
        RLA     R14                             ; c recebe o sinal de b
        JNC     else__MULT_signed                       ; Pule se b>0
        POP     R14
        INV     R14                             ; realizando o complemento de 2
        INC     R14                             ; para b
        CALL    #MULT_unsigned
        INV     R15                             ; realizando o comp. de 2 para
        INC     R15                             ; retornar -(MULT_unsigned(a,b))
        CLR     R14
        RET

else__MULT_signed:
        POP     R14
        CALL    #MULT_unsigned                          ; return MULT_unsigned(a,b)
        CLR     R14
        RET

MULT_unsigned:

        TST     R14             ; b==0 ?
        JNZ     MULT_unsigned_else      ; Se b não é zero, vá para else
```

```
        CLR.W   R15                 ; return 0
        RET

MULT_unsigned_else:

        PUSH    R15                 ; guarde a na pilha
        DEC.W   R14                 ; b--
        CALL    #MULT_unsigned      ; calcule a*(b-1)
        POP.W   R14                 ; recupere a da pilha
        ADD.W   R14,R15             ; return a + a*(b-1)
        RET

        END
```

10)

a) 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.

Se o vetor for palíndromo, retorne o valor 1. Caso contrário, retorne o valor 0. O protótipo da função é:

```
int Palindromo(int vetor[ ], int tamanho);
```

```
int Palindromo(int *p,int tamanho)
{
        int i;
        for(i=0;i<tamanho;i++)
        {
                if(p[i]!=p[(tamanho-1)-i])
                {
                        return 0;
                }
        }
        return 1;
}
```

b) Escreva a sub-rotina equivalente na linguagem Assembly do MSP430. O endereço do vetor de entrada é dado pelo registrador R15, o tamanho do vetor é dado pelo registrador R14, e o resultado é dado pelo registrador R15.

```
#include "msp430.h"                 ; #define controlled include file

        NAME    main                ; module name

        PUBLIC  main                ; make the main label vissible
                                    ; outside this module
```

```
        ORG     0FFFEh
        DC16    init                    ; set reset vector to 'init' label

        RSEG    CSTACK                  ; pre-declaration of segment
        RSEG    CODE                    ; place program in 'CODE' segment

init:   MOV     #SFE(CSTACK), SP        ; set up stack

main:   NOP                             ; main program
        MOV.W   #WDTPW+WDTHOLD,&WDTCTL  ; Stop watchdog timer
        MOV.W   #0x0a00,R15             ; R15 = ponteiro para o endereço de memória 0x0a00
        MOV.W   #1,0(R15)               ; vetor[0] = 1
        MOV.W   #2,2(R15)               ; vetor[1] = 2
        MOV.W   #3,4(R15)               ; vetor[3] = 3
        MOV.W   #2,6(R15)               ; vetor[4] = 2
        MOV.W   #1,8(R15)               ; vetor[5] = 1
        MOV.W   #10,R14                 ; R14 = tamanho*2
        CALL    #Palindromo
        JMP $                           ; jump to current location '$'
                                        ; (endless loop)

Palindromo:
        CLR     R13                     ; R13 = i = 0
For_Palindromo:
        CMP     R14,R13                 ; i >= 5 ?
        JGE     End_For_Palindromo      ;
        PUSH    R13                     ; guarde i na pilha
        PUSH    R14                     ; guarde tamanho na pilha
        DECD    R14                     ; R14 =  2*tamanho - 2*1
        SUB.W   R13,R14                 ; R14 = tamanho -1 - i
        ADD.W   R15,R14                 ; R14 = 0x0a00 + tamanho - 1 - i
        ADD.W   R15,R13                 ; R13 = 0x0a00 + i
        CMP     0(R13),0(R14)           ; p[i] != p[(tamanho-1)-i] ?
        JNE     Nao_palindromo          ;
        POP     R14                     ; recupere tamanho da pilha
        POP     R13                     ; recupere i da pilha
        INCD    R13                     ; i++ (mas 2 porque cada int tem 2 bytes)
        JMP     For_Palindromo

End_For_Palindromo:
        MOV.W   #1,R15                  ; R15 = 1
        CLR     R14
        CLR     R13
        RET
```

```
Nao_palindromo:
        POP     R15                 ; limpando a pilha
        POP     R15                 ; limpando a pilha
        CLR     R15                 ; return 0
        CLR     R14
        CLR     R13
        RET

        END
```